My Career

B.Sc.(Hons) in Software Development

Alexander Souza

G00317835@GMIT.IE

Supervisor

Damien Costello

Submission Date 16th April 2018

GitHub
https://github.com/alexpt2000gmit/4Year_MainProject_MyCareer

MyCareer

Applicant Tracking System

http://54.210.4.37/mycareer/

[2]

Table of Contents

1 Introduction 8
1.1 The Idea 8
1.2 The Application 8
1.3 Technology 9
1.4 Summary 11
2 Methodology 12
2.1 Planning 12
2.2 Methodology 13
2.3 Project Management 14
2.3.1 GitHub. 14
2.4 Validation and Testing 15
2.4.1 Validation 15
2.4.2 Verification 16
3 Technology Review 18
3.1 IDE - Development environment. 19
3.1.1 Spring Tool Suite 19
3.1.2 Visual Studio Code 19
3.2 Website 19
3.2.1 Joomla 19
3.3 API - Back end 20
3.3.1 Spring Framework 20
3.3.2 Flyway 20
3.3.3 JWT 21
3.3.4 Hibernate 21
3.4 Database Environment 21
3.4.1 MySQL 21
3.5 Front end 22
3.5.1 Angular 4 22
3.5.2 PrimeNG 22
3.5.3 Node.js 22
3.6 Test environment 23
3.6.1 Postman 23
3.6.2 Swagger UI 23
3.7 Repository 23
3.7.1 GitHub 23
3.8 Documentation 24
3.8.1 Javadoc 24
3.8.2 TypeDoc 24
3.9 Cloud services 24

3.9.1 Heroku 24
3.9.2 Amazon AWS 24
4 System Design 26
4.1 System Architecture 26
4.1.1 REST 28
4.1.1.1 Stateless: 28
4.1.1.2 HTTP methods 28
4.1.1.3 HTTP status codes 29
4.1.2 Design Patterns 29
4.2 Website 30
4.3 Front end (Applicants) 33
4.4 Back end (Admin - Applicants) 35
4.5 API 41
4.5.1 Security 42
4.5.1.1 JSON Web Tokens 43
4.5.1.2 Security Implementation 44
4.5.2 Database 48
4.5.3 Document 50
4.6 System Deployment 56
4.6.1 Deploy Joomla (AWS) 56
4.6.2 Deploy Angular (Heroku) 58
4.6.3 Deploy API (Heroku) 61
5 System Evaluation 66
5.1 Security 66
5.1.1 OAuth 2.0 66
5.1.1.1 OAuth Roles 66
5.1.1.2 Grant Types 67
5.1.2 JWT 68
5.1.3 What is a test in Postman? 68
5.1.3.1 Testing the API 69
5.2 Login 71
5.3 Block unauthorized URL 72
5.4 Redirect non-existent URL 73
5.5 Performance 73
6 Conclusion 76
6.1 Future work 77
7 References 78
References 78

Table of Illustrations

Fig. 1. The initial page of Project 9
Fig. 2. Architecture and Technologies 10
Fig. 3. Slides for presentation of project. 12
Fig. 4. Microsoft Access, planning for database 13
Fig. 5. GitHub – Projects tasks. 14
Fig. 6. GitHub – List of issues. 15
Fig. 7. Validation testing can be best demonstrated using V-Model 16
Fig. 8. Verification testing can be best demonstrated using V-Model 16
Fig. 9. System Architecture 26
Fig. 10. System Architecture for deploy. 27
Fig. 11. Basic Rest 28
Fig. 12. Stateless 28
Fig. 13. System Architecture for deploy website. 30
Fig. 14. Website – frontend 31
Fig. 15. Website – backend 31
Fig. 16. Joomla: Menu Careers 32
Fig. 17. Joomla: Careers Page 32
Fig. 18. System Architecture for deploy Front end - Applicant. 33
Fig. 19. Integration Joomla page and Angular (Applicant module) 34
Fig. 20. Sign Up for Alerts 34
Fig. 21. Form for a Job position. 35
Fig. 22. System Architecture for deploy Front and - Admin. 36
Fig. 23. Screen login 36
Fig. 24. Dashboard – Admin 37
Fig. 25. List Jobs 38
Fig. 26. New Job form. 38
Fig. 27. Applicants for a Job. 39
Fig. 28. Evaluate Candidate (Notes). 39
Fig. 29. Evaluate Candidate (Questions). 40
Fig. 30. Add application to schedule for interview. 40
Fig. 31. List of interviews or commitment. 41
Fig. 32. Spring Tool Suite (STS) 42
Fig. 33. UML API 42
Fig. 34. JWT requesting JSON web token 44
Fig. 35. Security - spring-security-jwt 44
Fig. 36. Security - Authorization Server to use JwtTokenStore 45
Fig. 37. Security - Resource Server configuration 45
Fig. 38. Security – CustomTokenEnhancer 46
Fig. 39. Security - key 46
Fig. 40. Security – Decoded key 47
Fig. 41. UI Login 47
Fig. 42. Security – Roles Payload 48
Fig. 43. Security - user with few privileges. 48
Fig. 44. Design database 49
Fig. 45. Hibernate configuration file snippet. 50
Fig. 46. Defining an entity and mapping to a table. 50
Fig. 47. Javadoc comments 51
Fig. 48. Javadoc HTML 51
Fig. 49. Swagger dependency 52
Fig. 50. Swagger methods 52
Fig. 51. Swagger, testing a method GET 53
Fig. 52. Swagger, return testing a method GET 53
Fig. 53. Swagger, return testing a security method GET 54
Fig. 54. Swagger config 54
Fig. 55. Swagger UI 55
Fig. 56. TypeDoc 55

Fig. 57. Deploy Joomla website – Configuration 57
Fig. 58. Deploy Joomla website – Database 57
Fig. 59. Deploy Joomla website - Overview 58
Fig. 60. Deploy Joomla website - Congratulations 58
Fig. 61. Postinstall - Package.json 59
Fig. 62. Angular – dist folder 59
Fig. 63. File Server.js 60
Fig. 64. Heroku – Create a new app 60
Fig. 65. Heroku – List of apps 60
Fig. 66. Heroku CLI 61
Fig. 67. Login Heroku 61
Fig. 68. Create a new app on Heroku 61
Fig. 69. Add MySQL to the API 62
Fig. 70. JawsDB – Settings 62
Fig. 71. Database profile on API 62
Fig. 72. Heroku - Additional parameters (JAR) 63
Fig. 73. Deploy – Commit API and Push to master 63
Fig. 74. Deploy process 64
Fig. 75. Heroku print the log 64
Fig. 76. The overall flow of a Password Grant 67
Fig. 77. Access rules 68
Fig. 78. Add body, key and value to request token 69
Fig. 79. Add Authorization on Headers to request token 69
Fig. 80. Decode Authorization value 70
Fig. 81. Get JSON Web Token with Postman 70
Fig. 82. Decode JSON web Token with JWT 71
Fig. 83. Request applicants 71
Fig. 84. User - unauthorized access 72
Fig. 85. Block unauthorized URL (Administrator) 72
Fig. 86. Block unauthorized URL (Alexander) 73
Fig. 87. Redirect non-existent URL 73
Fig. 88. BlazeMeter – Original Test Configuration 74
Fig. 89. BlazeMeter – Dashboard Summary 74
Fig. 90. BlazeMeter – Timeline Report 75
Fig. 91. BlazeMeter – Summary 75

Abstract.

This application is intended to publish work positions and also to classify the applicants for jobs published in the system. The application has to be modern and easy to use.

Another feature of this work is to be able to learn more about different software frameworks (Spring and Angular) and how to implement them.

The main problems were designing the structure of the front-end and back-end parts of the application, how to combine these two, which software frameworks to choose from for development to make the application as attractive as possible.

The application is easy to use on devices with different screen sizes. The application was de-developed with the help of different modern and highly evaluated structures. The author has gained an expiration for the use of different technologies.

1 Introduction

Introduction (5 Pages)

- *Provide a clear context for your project.*
  - What is it about? Is it at the right level (8)?
  - Is the scope correct?
  - Do not assume that the reader knows anything about the domain.
  - Why should a reader care or be interested?
- Set out the objectives of the project clearly.
  - You will have to address each of these in the evaluation / conclusion.
  - The metrics by which success or failure is measured.
- Briefly list each chapter / section and provide a brief description of what each section contains.
  - List the resource URL (GitHub address) for the project and provide a brief list and description of the main elements at the URL.
- *After reading the introduction, a reader should be 100% certain of what the project is all about and why it is relevant.*

Web applications are ruling the internet world since the 1990s. In today's world, there are thousands of web applications. Web applications are usually implemented in programming languages such as Java, C#, PHP, etc. RESTful applications are becoming very popular since the early 2000s. RESTful web applications are used in almost all the industries such as banking, hospitals, government organizations, etc.

The RESTful services include CRUD operations such as create, read, update and delete. These operations make use of GET, POST, PUT and DELETE. RESTful APIs are one of the main elements for web applications. The RESTful services can be implemented on various platforms according to the required specifications. The RESTful application can also be cross-platform where one application is implemented in one programming language and another application in another programming language. In this case, both the applications can communicate with each other, share data and services through REST operations. REST API is needed to make a REST call. All the REST calls are public, but it can be restricted through security. There are many ways to secure the RESTful API, but one of the ways is through Role Based Access Control.

1.1 The Idea

A year later, I began to study with more intentions in the use of Spring API and Angular, I pre-tended to put the two large ones in some project that interested me, so I had the idea to use them in my final project, even with little knowledge in Angular, for this I decided to create an application management application for job vacancies.

Knowing that it should be a system that manages personal data, I decided to invest more in knowledge of security and authentication.

The idea of creating this type of System would be a great way to learn and incorporate several new technologies into one Web application and also know that this type of application could be scalable.

The idea was that candidates could apply for the vacancies available in a System integrated the current page of the company, and that the administrators could manage the candidates for the vacancies in which they will apply.

1.2 The Application

MyCareer is a system that fully supports the recruitment and selection process of HR, provides an internal system to manage resumes, jobs, and a tool that can be integrated with the current system of the company that will make the company's own web site in an online recruiter of candidates.

MyCareer is an application that, in addition to being scalable, uses the best in development, appearance, security and robustness frameworks.

The application allows features like:

Applicants

- View available jobs.
- Apply the selected work through the website.
- Submit the email to the job alert receipt

Administrator

- Manage all available jobs.
- View all applicants.
- View applicants per job.
- Process of evaluation of the candidate in the act of the interview.
- Assessment with technical questions for candidates, divided by topics.
- Manage schedule for interviews and other appointments.

Fig. 1. The initial page of Project

1.3 Technology

Several new technologies that were used that hadn't been used before, prior to commencing software development. For the project management side of things, GitHub was used to manage Milestones and Issues.

Fig. 2. Architecture and Technologies

The technologies are:

- IDE - Development environment
  - Spring Tool Suite
  - Visual Studio Code

- API - Back end
  - Spring Security
  - JSON Web Token (JWT)
  - Spring Boot
  - Maven
  - Hibernate
  - Flyway
  - Swagger UI

- Database Environment
  - MySQL

- Front end
  - Joomla
  - HTML5 and CSS
  - Angular 4
  - PrimeNG - is a collection of rich UI components.

- Cloud services
  - AWS
  - Heroku

- Test environment
  - Postman
  - Swagger UI

- Repository
    - GitHub

- Documentation
    - JavaDoc
    - JSDoc

More detail on these technologies will be better clarified in the chapters 3 and 4.

1.4 Summary

Throughout this report, several different topics are going to be discussed, ranging from how the project was started and how requirements were gathered, to the actual final product that is being delivered. For clarity, here's a synopsis of what each chapter is about.

Chapters

- 2 - Methodology – page 12
- 3 - Technology Review – page 18
- 4 - System Design – page 26
- 5 - System Evaluation – page 66
- 6 - Conclusion – page 76
- 7 - References – page 78

2 Methodology

Methodology (3 -5 Pages)

- *Describe the way you went about your project. Was your approach to the problem valid?*
  - Software development v/s Research methodology.
- Agile / incremental and iterative approach to development.
  - Planning. Did you storyboard? How did you determine the requirements for the project?
  - Meetings. Frequency, structure, checks & balances, feedback.
- What about validation and testing?
  - Junit or some other framework.
- If team based, did you use GitHub during the development process? What about other development tools?
  - Selection criteria for algorithms, languages, platforms and technologies.
- Was an empirical approach used? How were problems solved?
  - Was any research undertaken first?

2.1 Planning

Before any project commences, its quintessential that the proper planning and organization is done to ensure as smooth a development process as possible.

The first port of call before starting was to write down the various high-level components that are involved in a software development environment.

Following the original idea shown in **Fig. 3.** Slides for presentation of project., which was part of the initial presentation, I started the entire research process for the implementation and integration of the technologies to be applied in this project.

Fig. 3. Slides for presentation of project.

These are the key parts of the Software Development process, that were focused on.

- **Project Management** – Pivotal in any project and allowed the team to manage time and efforts in the most efficient manner.

- **Architecture** – Relates to the overall abstract view of the system.
- **Technology** – Looked at the different technologies that were used as part of the development process, for example, programming languages and IDE's (Integrated Development Environments).

- **Testing** – Looked at various testing techniques associated with the Application.

- **Cloud** – Looked at how the system was going to be deployed.

In this project, to avoid any type of scalability problem in the future, one of the ways was to develop a database prototype in order to solve any bias in development.

For this project, Microsoft Access **Fig. 4** was used to create the tables and related relationships, in this way it simplified the development and solved some problems like;

- Safety;
- Scalability;
- Dependencies;
- Relationships;

Fig. 4. Microsoft Access, planning for database

2.2 Methodology

As students, the feeling was that the best way to do this was by using flavours of the Agile methodology and its associated subsets such as RAD (Rapid Application Development), Kanban and XP Programming.

This route was chosen as fundamentally, Agile offers a prototype based, iterative approach which not only allowed for rapid development but introduced a loose structure that suited the team's development skillsets.

2.3 Project Management

Project management is one of the most important points for the development of a project, it provides a framework that allows efficient time planning, robust work delivery and a smooth journey throughout the project development life cycle relative to In this project, the planning was of paramount importance, since the development time of each part was determined for this project, in order to guarantee a better organization and prioritize the tasks.

2.3.1 **GitHub**.

For this project, we used GitHub for version control and also to manage the whole project. There are many different project management tools available in the software market and they all do specific things. GitHub was chosen because it offers a simple yet effective approach to project management. GitHub was used to follow the project tasks, creating list of problems and tasks to be performed.

Here are some screenshots of how GitHub's project management features were used;

Fig. 5. GitHub – Projects tasks.

Fig. 6. GitHub – List of issues.

2.4 Validation and Testing

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

2.4.1 Validation

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

Validation Testing - Workflow:
Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.

Fig. 7. Validation testing can be best demonstrated using V-Model

Activities:

- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing

2.4.2 Verification

Verification is the process of evaluating work-products of a development phase to determine whether they meet the specified requirements.

verification ensures that the product is built according to the requirements and design specifications.

Verification Testing - Workflow:
verification testing can be best demonstrated using V-Model. The artefacts such as test Plans, requirement specification, design, code and test cases are evaluated.

verification testing in Test Life Cycle

Fig. 8. Verification testing can be best demonstrated using V-Model

Activities:

- Reviews
- Walkthroughs
- Inspection

A fully functional web application, there remained a keen focus on quality, whilst still maintaining a firm grip on weekly deliverables. To do this, testing was done in two ways.

Integration Testing

This is a type of self-explanatory testing where certain sections or components that were created, are analyzed to ensure that there is a seamless transition into the system. To do this, each section was tested on an individual basis first, then added as a newly added part of the system.

3 Technology Review

Technology Review (10+ Pages)

- *The "literature review" part of the dissertation. Should be tightly coupled to the context and objective from the introduction.*
  - Proves that you researched what you were doing!
- Describe each of the technologies you used at a conceptual level.
  - Standards, Database Model (e.g. MongoDB, CouchDB), XMl, WSDL, JSON, JAXP.
  - Use references (IEEE format, e.g. [1]), Books, Papers, URLs (timestamp)
  - Sources should be authoritative!
- A technology review that includes a lot of *de facto* or *de jure* standards supports the methodology!
  - Each chapter buttresses some other aspect of the dissertation.

MyCareer is an enterprise web application and several new technologies that were used that hadn't been used before, prior to commencing software development. For the project management side of things, GitHub was used to manage Milestones and Issues.

The technologies are:

- IDE - Development environment
  - Spring Tool Suite
  - Visual Studio Code

- API - Back end
  - Spring Security
  - JSON Web Token (JWT)
  - Spring Boot
  - Maven
  - Hibernate
  - Flyway
  - Swagger UI

- Database Environment
  - MySQL

- Front end
  - Joomla
  - HTML5 and CSS
  - Angular 4
  - PrimeNG - is a collection of rich UI components.

- Cloud services
  - AWS
  - Heroku

- Test environment
  - Postman
  - Swagger UI

- Repository
  - GitHub

- Documentation
  - JavaDoc
  - JSDoc

3.1 IDE - Development environment.

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools, and a debugger. Most modern IDEs have intelligent code completion.

3.1.1 Spring Tool Suite

We provide a customized all-in-one Eclipse based distribution that makes application development easy. The tool suites provide ready-to-use combinations of language support, framework support, and runtime support, and combine them with the existing Java, Web and Java EE tooling from Eclipse. An integrated dashboard in the distribution makes it trivial to install further extensions that add support for technologies such as Pivotal Cloud Foundry, Gradle or Pivotal tcServer.[1]

3.1.2 Visual Studio Code

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring.[2]

3.2 Website

The website will serve as a basis and demonstration of how the entire system can be integrated even on the company's current website.

3.2.1 Joomla

Joomla! is a free and open-source content management system (CMS) for publishing web content. Over the years Joomla! has won several awards. It is built on a model–view–controller web application framework that can be used independently of the CMS that allows you to build powerful online applications.

Joomla! is one of the most popular website software, thanks to its global community of developers and volunteers, who make sure the platform is user-friendly, extendable, multilingual, accessible, responsive, search engine optimized and so much more.[3]

Joomla! can be used for:

- Corporate websites or portals, intranets and extranets
- Small business websites
- Online magazines, newspapers, and publications

1 https://spring.io/tools/sts/all

2 https://code.visualstudio.com/

3 https://www.joomla.org/

- E-commerce and online reservations
- Government, non-profit and organisational websites
- Community-based, school and church websites or portals
- Personal or family homepages ...

3.3 API - Back end

APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets. An API approach is an architectural approach that revolves around providing programmable interfaces to a set of services to different applications serving different types of consumers. When used in the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format.

3.3.1 Spring Framework

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to, or even replacement for the Enterprise JavaBeans (EJB) model. The Spring Framework is open source.[4]

For this project be used:

- Spring Security
- Spring Boot

3.3.2 Flyway

Flyway is an open source database migration tool. It strongly favors simplicity and convention over configuration.

It is based around 7 basic commands: Migrate, Clean, Info, Validate, Undo, Baseline and Repair.

Migrations can be written in SQL (database-specific syntax (such as PL/SQL, T-SQL, ...) is supported) or Java (for advanced data transformations or dealing with LOBs).[i]

It has a Command-line client, a Java API (also works on Android) for migrating the database on application startup, a Maven plugin and a Gradle plugin.

Plugins are available for Spring Boot, Dropwizard, Grails, Play, SBT, Ant, Griffon, Grunt, Ninja and more.

Supported databases are Oracle, SQL Server, DB2, MySQL (including Amazon RDS), MariaDB, PostgreSQL (including Amazon RDS and Heroku), CockroachDB, Redshift, H2, Hsql, Derby, SQLite, SAP HANA, Sybase ASE and Phoenix.[5]

4 https://spring.io/

5 https://flywaydb.org/

3.3.3 JWT

JSON Web Token (JWT) is a compact token format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. JWTs encode claims to be transmitted as a JSON object that is base64url encoded and digitally signed and/or encrypted. Signing is accomplished using JSON Web Signature (JWS). Encryption is accomplished using JSON Web Encryption (JWE).[6]

3.3.4 Hibernate

Hibernate ORM (Hibernate in short) is an object-relational mapping tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate handles object-relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions.[7]

3.4 Database Environment

A database environment is a collective system of components that comprise and regulates the group of data, management, and use of data which consist of software, hardware, people, techniques of handling database and the data also.

3.4.1 MySQL

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation. For proprietary use, several paid editions are available and offer additional functionality.

MySQL is a central component of the LAMP open-source web application software stack (and other "AMP" stacks). LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python". Applications that use the MySQL database include TYPO3, MODx, Joomla, WordPress, Simple Machines Forum, phpBB, MyBB, and Drupal. MySQL is also used in many high-profile, large-scale websites, including Google, Facebook, Twitter, Flickr, and YouTube.[8]

6 https://jwt.io/

7 http://hibernate.org/orm/

8 https://www.mysql.com/

3.5 Front end

In software architecture, there may be many layers between the hardware and end user. Each can be spoken of as having a front end and a back end. The front is an abstraction, simplifying the underlying component by providing a user-friendly interface, while the back usually handles business logic and data storage.

3.5.1 Angular 4

Angular (commonly referred to as "Angular 4" or "Angular 2") is a TypeScript-based open-source front-end web application platform led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite of the same team that built AngularJS.[9]

3.5.2 PrimeNG

PrimeNG is a collection of rich UI components for Angular. All widgets are open source and free to use under MIT License. PrimeNG is developed by PrimeTek Informatics, a vendor with years of expertise in developing open source UI solutions. [10]

3.5.3 Node.js

Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side. Historically, JavaScript was used primarily for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML, to be run client-side by a JavaScript engine in the user's web browser. Node.js enables JavaScript to be used for server-side scripting and runs scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.[11]

9 https://angular.io/

10 https://www.primefaces.org/primeng/

11 https://nodejs.org/

3.6 Test environment

Test Environment Management (TEM) is a function in the software delivery process which aids the software testing cycle by providing a validated, stable and usable test environment to execute the test scenarios or replicate bugs.

3.6.1 Postman

An API Development Environment - or ADE - is a platform that supports and enhances API development. A good ADE will streamline the development process, create a single source of truth for an organization's APIs, and enhance collaboration on APIs across the organization.[12]

3.6.2 Swagger UI

Swagger UI allows anyone — be it your development team or your end consumers — to visualize and interact with the API's resources without having any of the implementation logic in place. It's automatically generated from your Swagger specification, with the visual documentation making it easy for back end implementation and client side consumption.[13]

3.7 Repository

A source code repository is a file archive and web hosting facility where a large amount of source code, for software or for web pages, is kept, either publicly or privately.

3.7.1 GitHub

GitHub (originally known as Logical Awesome LLC) is a web-based hosting service for version control using git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

GitHub offers plans for both private repositories and free accounts which are commonly used to host open-source software projects. As of April 2017, GitHub reports having almost 20 million users and 57 million repositories, making it the largest host of source code in the world.[14]

12 https://www.getpostman.com

13 https://swagger.io/

14 https://github.com/

3.8 Documentation

Software documentation is written text or illustration that accompanies computer software or is embedded in the source code. It either explains how it operates or how to use it, and may mean different things to people in different roles.

3.8.1 Javadoc

Javadoc is a documentation generator created by Sun Microsystems for the Java language for generating API documentation in HTML format from Java source code. [15]

3.8.2 TypeDoc

TypeDoc is a documentation generator for typescript projects.

TypeDoc runs the TypeScript compiler and extracts type information from the generated compiler symbols. Therefore you don't have to include additional metadata within your comments, TypeScript specific elements like classes, enumerations or property types and access modifiers will be automatically detected.

3.9 Cloud services

Cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with pay-as-you-go pricing.

3.9.1 Heroku

Heroku is a cloud platform as a service (PaaS) supporting several programming languages that is used as a web application deployment model. Heroku, one of the first cloud platforms has been in development since June 2007, when it supported only the Ruby programming language, but now supports Java, Node.js, Scala, Clojure, Python, PHP, and Go. For this reason, Heroku is said to be a polyglot platform as it lets the developer build, run and scale applications in a similar manner across all the languages.[16]

3.9.2 Amazon AWS

Amazon Web Services (AWS) is a subsidiary of Amazon.com that provides on-demand cloud computing platforms to individuals, companies and governments, on a paid subscription basis. The technology allows subscribers to have at their disposal a full-fledged virtual cluster of computers, available all the time, through the Internet.[17]

15 http://www.oracle.com/technetwork/java/javase/tech/index-137868.html

16 https://www.heroku.com/

17 https://aws.amazon.com/

4 System Design

Languages, tools, and technologies used during development are key factors in making applications different from one another. Once decisions are made on which tools and technologies to use, the overall system architecture, entity-relationship model, and REST API and Front-end.

4.1 System Architecture

The MyCareer application has been completely created from the ground up using a methodical and structured approach.

By analyzing the initial specifications of the project. As the work began and the various parts of how the System was to be structured were separated, it became more evident that large amounts of time were needed to obtain the overall system architecture from the outset in order to facilitate maintenance, robustness, and scalable Web application.

To begin with, the analysis began by noting what kind of users would be using the system and how they would interact with it. At the most basic level, various ways to send and receive information were scanned, allowing information to be sent from the user side to the servers at the end of the application. To put this into some kind of context and create a visual perspective.

Fig. 9. System Architecture

For the MyCareer project, the modules were separated in order to better uncouple the functionality, maintenance and security.

The API, Website Angular module for applications and the Angular module of administration, can be found in different servers. **Fig. 10**

Fig. 10. System Architecture for deploy.

For the deploy of the Joomla website, one was used in the AWS, whose greater detail you are without in the topic Deploy Joomla (AWS).

For API, Angular applicants and Angular admin, we used Heroku for deploy.

The beauty of Heroku is in its simple model: push your code to a remote Heroku repo and it will run everything necessary to deploy it. The key is that we need to tell Heroku a few things about how the app should be deployed.

Access URL for the Website page:

- http://54.210.4.37/mycareer/

Access URL for the front-end Applicants page:

- https://mycareer-webajobs.herokuapp.com/

Access URL for the front-end Admin page:

- https://mycareer-webadmin.herokuapp.com/

Access URL for the API page:

- https://mycareer-api.herokuapp.com/

4.1.1 REST

REST stands for Representational State Transfer. REST architecture style is mainly based on the stateless, client-server and HTTP protocol. This architectural style is a key aspect in designing network applications and distributed systems. REST does not completely rely on HTTP but mostly linked with it. The properties of REST play the vital role in the REST architecture style and make the REST architecture simpler. REST architecture is a lightweight alternative to other mechanisms like RPC4, SOAP5, and WSDL6. Moreover; REST is a platform-independent, and language-independent service. The following sub-sections explain in detail about REST.

Fig. 11. Basic Rest

4.1.1.1 Stateless:

The client-server communication must be stateless in nature. The stateless nature is because that server must have all the information that are needed to respond to the request made by the client. The session state entirely depends on the client. In this constraint, the properties such as visibility, scalability and reliability are improved based on different aspects. The main drawback is that is the decrease in the performance of the network by sending the same data in the cluster of requests.

Fig. 12. Stateless

4.1.1.2 HTTP methods

HTTP methods are used to map CRUD operations to HTTP requests. HTTP methods are used with REST to form as a RESTful service. GET, POST, PUT, and DELETE are the four main HTTP methods.

1. **GET**

‘GET’ is used to retrieve the data from the database. The GET requests can be partial or conditional. The partial request retrieves all the information from the particular table. The conditional request retrieves only the specific data from the database based on the condition.

Example:
GET /applicants – gets all the plants of the respective table
GET / applicants /1 – gets the plant with an ID of 1 of the respective table

2. POST
This HTTP method is mainly used to create a new entity in the table. However, it can also be used for update an existing entity.

Example:
POST / applicants – creates a new plant

3. PUT
Like POST, ‘PUT’ can be used to create a new entity and also used to update an existing entity in the table. PUT is idempotent.

Example:
PUT / applicants /1 – update the plant with an ID of 1

4. DELETE
If a resource has to be removed, then DELETE method can be used.

Example:
DELETE / applicants /1 – deletes the plant with an ID of 1

4.1.1.3 HTTP status codes

HTTP response status codes are the codes that are results of the HTTP requests. When an HTTP request is made from the client, the server will send an appropriate status code along with the data, if any. The browser translates these status codes. The types of HTTP status codes are:

- 1XX - informational
- 2XX - success
- 3XX - redirection
- 4XX – client error
- 5XX – server error

4.1.2 Design Patterns

Design Patterns are general reusable solutions to commonly occurring problems. Patterns are not complete code, but it can be uses as a template which can be applied to a problem. Patterns are re-usable and they can be applied to similar kinds of design problems regardless of the domain. A pattern used in one practical context can be re-usable in other contexts also. Here are some of the reasons to use design patterns;

1. **Flexibility**: Using design patterns makes code flexible. It helps to provide the correct level of abstraction due to which objects become loosely coupled to each other which makes the code easy to change.

2. **Reusability**: Loosely coupled and cohesive objects and classes can make the code more reusable. This kind of code becomes easy to be tested as compared to the highly-coupled code.

3. **Shared** Vocabulary: Shared vocabulary makes it easy to share the code with other team members. It creates more of an understanding between the team members in relation to the code.

4. **Capture** best practices: Design patterns capture solutions which have been successfully applied to problems. By learning these patterns and the related problems, an inexperienced developer learns a lot about software design.

4.2 Website

To offer a better user experience, a website has been implemented to better represent the use of the MyCareer system.

Fig. 13. System Architecture for deploy website.

The system also demonstrates how the system can also be used on the company's current website, in which case it has been used by CMS Joomla, a powerful website development tool that allows users to manage website content from a backend.

Access URL for the Website page:

- http://54.210.4.37/mycareer/

Fig. 14. Website – frontend

1. The backend or administration **Fig. 14** area of the website allows users to be able to maintain content, menus, images, videos or change the template.

Fig. 15. Website – backend

The integration between Website and the Applicant application module (Angular) is done through the creation of a new menu in which the application modules will be within a iframe **Fig. 16** of the page.

Fig. 16. Joomla: Menu Careers

Fig. 17. Joomla: Careers Page

4.3 Front end (Applicants)

Through the job application module, the applicant can view a list of all available jobs on the page and also be able to submit their personal data to the selected position.

Fig. 18. System Architecture for deploy Front end - Applicant.

Access URL for the Website page:

- http://54.210.4.37/mycareer/

Access URL for the front-end Applicants page:

- https://mycareer-webajobs.herokuapp.com/

Fig. 19. Integration Joomla page and Angular (Applicant module)

On the same page, there is an alert area in which the applicant can also register to a list of alerts through his email.

Fig. 20. Sign Up for Alerts

Once the applicant has selected the desired position, he / she will have access to a fill form with fill fields for the selected position, which will be sent to the API where the data will be stored for later evaluation.

Fig. 21. Form for a Job position.

4.4 Back end (Admin - Applicants)

To implement better system security, there is a separation between the application module and the administrator module, allowing any intervention on one of the modules does not affect the operation of the system, even if both are connected to the same API.

Through this module, the administrator can manage the data provided by the API, such as:

- Manage published work;
- Manage applicants;
- Manage the system agenda;
- Candidate evaluation system;
- List of questions for the interview process.

Fig. 22. System Architecture for deploy Front and - Admin.

The application also has a login page, which can be assigned different levels of access to the system, blocking some modules depending on the user trying to login.

Fig. 23. Screen login

Once the user is logged in, the administrator will have access to the system dashboard, in which it will be available:

- Arranging interviews and appointments;

- Total number of new applicants and applicants;
- Total of works published in the system;
- On the system menu bar.

Fig. 24. Dashboard – Admin

Through the works menu, the administrator can manage the Jobs, being able to:

- Delete;
- Add a new Job;
- To edit;
- Change the Job Status;
- View the applicants for the selected job;

The form has features that facilitates browsing of the work list, such as;

- Search in any column;
- Filter by column;
- Pagination;

Fig. 25. List Jobs

The same form style allows the administrator to have access to form fields for both a new and an edit. Some fields have validation.

Fig. 26. New Job form.

When you select the Applications button through the Jobs list, the administrator can have access to the list with all of them for the published position.

In this list you will also find some options, with:

- Name;
- Date;
- Contact;
- Status (Corresponds to information entered by the HR);
- Notes (Corresponds to an assessment score inserted by the HR);

- Questions (Corresponds to a score of evaluation of technical questions for position, inserted by the HR);

The buttons are actions that will be applied to each applicant, with:

- Evaluate (Access to the candidate evaluation form)
- Schedule (Allows HR to add the applicant directly to the system calendar.
- To edit;
- Delete;

Fig. 27. Applicants for a Job.

In the interview process the administrator (HR) may as well evaluate the candidate through a score between 0 and 5, which will also be represented by the star.

Any information that the HR judges important about the candidate may also be filled in a text field.

All saved data can later be reused for further query and filtering.

Fig. 28. Evaluate Candidate (Notes).

Through the Questions tab, the administrator (HR) can continue the interview by selecting a set of previously added questions.

The questions are grouped by type and can be selected according to the profile of the candidate.

Each question can also be assigned a score, when saving the question, will be available in a question list assigned to the respective candidate.

Each question that has been saved to the candidate can be re-selected for display, editing or deletion.

Fig. 29. Evaluate Candidate (Questions).

When you save the evaluation, the system will automatically calculate the average score of the questions so that it is presented in the Questions column of the applicant list as shown in **Fig. 27**.

In the **Fig. 27** application list, a button is available in which you can add an applicant directly to the schedule as an interview.

Fig. 30. Add application to schedule for interview.

By including the applicant in the schedule, the applicant will automatically make available both the list **Fig. 31** and the available calendar in the system dashboard **Fig. 24**.

+ Add Schedule | Applicants | List of Schedules

Title	Start	End	Notes
Intv-Alexander Souza	07/02/2018 10:10	07/02/2018 12:10	Test Notes
Albert	07/02/2018 11:10	07/02/2018 15:10	Test Notes
Kevin	08/02/2018 10:10	08/02/2018 12:10	Test Notes
Food	17/02/2018 08:10	17/02/2018 09:00	Test Notes
All day	17/02/2018 00:00		Test Notes
John	17/02/2018 08:10	19/02/2018 09:00	Test Notes
Meeting	13/02/2018 00:00	15/02/2018 00:00	Test Notes
Intv-Kevin Souza	19/02/2018 10:10	19/02/2018 12:10	Test Notes
Intv-Sandra Souza	17/02/2018 10:10	17/02/2018 12:10	Test Notes
Intv-Alexander Souza	07/03/2018 10:10	07/03/2018 12:10	Test Notes
Albert	07/03/2018 11:10	07/03/2018 15:10	Test Notes
Kevin	08/03/2018 10:10	08/03/2018 12:10	Test Notes
Food	17/03/2018 08:10	17/03/2018 09:00	Test Notes
All day	17/03/2018 00:00		Test Notes
John	17/03/2018 08:10	19/03/2018 09:00	Test Notes

1 2

Fig. 31. List of interviews or commitment.

4.5 API

API stands for Application Programming Interface. At some point or another, most large companies have built APIs for their customers, or for internal use.

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

For the MyCareer project the API will serve as the data for the UI, the persistence of the data and also to ensure the security of the application.

This is a RESTful API that uses the following technologies:

- Spring Boot;
- String Security;
- JWT;
- Hibernate;

Spring Boot provides mechanisms for creating microservices based on an opinionated view of what technologies should be used. Spring Boot applications are Spring applications, and use a programming model unique to that ecosystem.

Spring Boot applications can be packaged to run as an application hosted by an application server, or run as a runnable .jar file containing the composition of dependencies and some type of embedded server (usually Tomcat). Spring Boot emphasizes convention over configuration, and uses a combination of annotations and classpath discovery to enable additional functions.

Spring Tool Suite (STS) was used as a development tool.

Fig. 32. Spring Tool Suite (STS)

In order to maintain good practices, the entire code structure attempts to follow a functional organization, as shown in **Fig. 33**.

Fig. 33. UML API

4.5.1 Security

One of the goals of this project is to implement security features in a RESTful API. The security is the way of protecting the system from an anonymous user access. In computer security, the data should be

protected so that only authorized users can only access the data. This is typically known as Information security. Authentication and authorization are the two basic concepts in security. Authentication refers to the login access to the system. It means that the user who is registered to the system all with some role will have some login credentials to access the system.

Security is the enemy of convenience, and vice versa. This statement is true for any system, virtual or real, from the physical house entrance to web platforms. Engineers are constantly trying to find the right balance for the given use case, leaning to one side or the other. Usually, when a new threat appears, we move towards security and away from convenience. Then, we see if we can recover some lost convenience without reducing the security too much.

Although REST services do not have much specified, an important one is the lack of state. It means the server does not keep any client state, with sessions as a good example. Thus, the server replies to each request as if it was the first the client has made. However, even now, many implementations still use cookie based authentication, which is inherited from standard website architectural design. The stateless approach of REST makes session cookies inappropriate from the security standpoint, but nevertheless, they are still widely used. Besides ignoring the required statelessness, simplified approach came as an expected security trade-off. Compared to the WS-Security standard used for Web Services, it is much easier to create and consume REST services, hence convenience went through the roof. The trade-off is pretty slim security; session hijacking and cross-site request forgery (XSRF) are the most common security issues.

In trying to get rid of client sessions from the server, some other methods have been used occasionally, such as Basic or Digest HTTP authentication. Both use an Authorization header to transmit user credentials, with some encoding (HTTP Basic) or encryption (HTTP Digest) added. Of course, they carried the same flaws found in websites: HTTP Basic had to be used over HTTPS since username and password are sent in easily reversible base64 encoding, and HTTP Digest forced the use of obsolete MD5 hashing that is proven to be insecure.

Finally, some implementations used arbitrary tokens to authenticate clients. This option seems to be the best we have, for now. If implemented properly, it fixes all the security problems of HTTP Basic, HTTP Digest or session cookies, it is simple to use, and it follows the stateless pattern.

However, with such arbitrary tokens, there's little standard involved. Every service provider had his or her idea of what to put in the token, and how to encode or encrypt it. Consuming services from different providers required additional setup time, just to adapt to the specific token format used. The other methods, on the other hand (session cookie, HTTP Basic and HTTP Digest) are well known to developers, and almost all browsers on all devices work with them out of the box. Frameworks and languages are ready for these methods, having built-in functions to deal with each seamlessly.

4.5.1.1 JSON Web Tokens

JSON Web Tokens, commonly known as JWTs, are tokens that are used to authenticate users on applications. This technology has gained popularity over the past few years because it enables backends to accept requests simply by validating the contents of these JWTS. That is, applications that use JWTS no longer have to hold cookies or other session data about their users. This characteristic facilitates scalability while keeping applications secure.

During the authentication process, when a user successfully logs in using their credentials, a JSON Web Token is returned and must be saved locally (typically in local storage). Whenever the user wants to access a protected route or resource (an endpoint), the user agent must send the JWT, usually in the Authorization header using the Bearer schema, along with the request.

When a backend server receives a request with a JWT, the first thing to do is to validate the token. This consists of a series of steps, and if any of these fails then, the request must be rejected.

Fig. 34. JWT requesting JSON web token

4.5.1.2 Security Implementation

For REST services to work as expected, we need a slightly different authorization approach compared to classic, multi-page websites.

Instead of triggering the authentication process by redirecting to a login page when a client requests a secured resource, the REST server authenticates all requests using the data available in the request itself, the JWT token in this case. If such an authentication fails, redirection makes no sense. The REST API simply sends an HTTP code 401 (Unauthorized) response and clients should know what to do; for example, a browser will show a dynamic div to allow the user to supply the username and password.

On the other hand, after a successful authentication in classic, multi-page websites, the user is redirected by using HTTP code 301 (Moved permanently), usually to a home page or, even better, to the page the user initially requested that triggered the authentication process. With REST, again this makes no sense. Instead we would simply continue with the execution of the request as if the resource was not secured at all, return HTTP code 200 (OK) and expected response body.

As expected, Spring Security framework comes with many ready to plug-in classes that deal with “old” authorization mechanisms: session cookies, HTTP Basic, and HTTP Digest. However, it lacks the native support for JWT, and we need to get our hands dirty to make it work.

Configuration properties.

To enable the JWT support, we just have to add the spring-security-jwt artifact dependency to the dependencies section of our project's pom:

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-jwt</artifactId>
</dependency>
```

Fig. 35. Security - spring-security-jwt

Authorization Server

Will configure our Authorization Server to use JwtTokenStore – as follows:

```java
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();
    tokenEnhancerChain.setTokenEnhancers(Arrays.asList(tokenEnhancer(), accessTokenConverter()));

    endpoints
        .tokenStore(tokenStore())
        .tokenEnhancer(tokenEnhancerChain)
        .reuseRefreshTokens(false)
        .authenticationManager(authenticationManager);
}

 * Access token converter.
@Bean
public JwtAccessTokenConverter accessTokenConverter() {
    JwtAccessTokenConverter accessTokenConverter = new JwtAccessTokenConverter();
    accessTokenConverter.setSigningKey("mycareer");
    return accessTokenConverter;
}

 * Token store.
@Bean
public TokenStore tokenStore() {
    return new JwtTokenStore(accessTokenConverter());
}

 * Token enhancer.
@Bean
public TokenEnhancer tokenEnhancer() {
    return new CustomTokenEnhancer();
}
```

Fig. 36. Security - Authorization Server to use JwtTokenStore

Resource Server

Resource Server configuration – which is very similar to the config of the Authorization Server

```java
@Autowired
public void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
}

// http://www.baeldung.com/security-none-filters-none-access-permitAll
 * (non-Javadoc)
// Get values from API address /jobsweb/** , ignore security in this resource.
@Override
public void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().antMatchers("/jobsweb/**").permitAll().antMatchers("/applicantsweb/**").permitAll()
            .antMatchers("/alertsweb/**").permitAll().antMatchers("/swagger-resources/**").permitAll()
            .antMatchers("/v2/**").permitAll().antMatchers("/webjars/**").permitAll()
            .antMatchers("/swagger-ui.html").permitAll().anyRequest().authenticated().and().sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS).and().csrf().disable();
}

 * (non-Javadoc)
@Override
public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
    resources.stateless(true);
}

 * Password encoder.
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

 * Creates the expression handler.
@Bean
public MethodSecurityExpressionHandler createExpressionHandler() {
    return new OAuth2MethodSecurityExpressionHandler();
}
```

Fig. 37. Security - Resource Server configuration

Custom Claims in the Token

Infrastructure to be able to add a few custom claims in the Access Token. The standard claims provided by the framework are all well and good, but most of the time we'll need some extra information in the token to utilize on the client side.

We'll define a CustomTokenEnhancer to customize our Access Token with these additional claims.

```
public class CustomTokenEnhancer implements TokenEnhancer {

    @Override
    public OAuth2AccessToken enhance(OAuth2AccessToken accessToken, OAuth2Authentication authentication) {
        UserSystem userSystem = (UserSystem) authentication.getPrincipal();

        Map<String, Object> addInfo = new HashMap<>();
        addInfo.put("name", userSystem.getUser().getName());

        ((DefaultOAuth2AccessToken) accessToken).setAdditionalInformation(addInfo);
        return accessToken;
    }
```

Fig. 38. Security – CustomTokenEnhancer

key generated.
A sample response will look like this:


```
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJ1c2VyX25hbWUiOiJhZG1pbkBteWNhcmVlci5jb20iLCJzY29wZSI6WyJyZWFkIiwid3JpdGUiXSwibmFtZSI6IkFkbW1uaXN0cmF0b3IiLCJ1eHAiOjE1MjI3NjEwMDc
        sImF1dGhvcm10aWVzIjpbI1JPTEVfUkVBRF9TQ0hFRFVMRSIsI1JPTEVfQUREX0FQUExJQ0FOVCIsI1JPTEVfQUREX0pPQiIsI1JPTEVfUkVNT1ZFX1NDSEVEVUxFIiwiUk9
        MRV9SRU1PVkVfQVBQTE1DQU5UIiwiUk9MRV9BRERfU0NIRURVTEUiLCJST0xFX1JFTU9WRV9KT0IiLCJST0xFX1JFQUREFQVBQTE1DQU5UIiwiUk9MRV9SRUFEX0pPQiJdLCJ
        qdGkiOiJjMTgyYmN1OC0wNGYzLTQ1MDEtYTgzMy05N2QzM2VhOWQ4YTAiLCJjbGllbnRfaWQiOiJhbmd1bGFyIn0.QJGdhB8BKb_G
        -rv25lIaWdrIffcSBnTlPxNCdJnlAH8",
    "token_type": "bearer",
    "expires_in": 1799,
    "scope": "read write",
    "name": "Administrator",
    "jti": "c182bce8-04f3-4501-a833-97d33ea9d8a0"
}
```


Fig. 39. Security - key

Fig. 40. Security – Decoded key

By logging into the application from the UI, the system will provide the user and application data, and what the API will return will be the token encrypted with the access rules the user will have access to.

Fig. 41. UI Login

These rules (Payload) will be used in the system in order to be able to enable or disable system functionalities, thus creating different levels of access.

PAYLOAD: DATA


```
{
  "user_name": "admin@mycareer.com",
  "scope": [
    "read",
    "write"
  ],
  "name": "Administrator",
  "exp": 1522761007,
  "authorities": [
    "ROLE_READ_SCHEDULE",
    "ROLE_ADD_APPLICANT",
    "ROLE_ADD_JOB",
    "ROLE_REMOVE_SCHEDULE",
    "ROLE_REMOVE_APPLICANT",
    "ROLE_ADD_SCHEDULE",
    "ROLE_REMOVE_JOB",
    "ROLE_READ_APPLICANT",
    "ROLE_READ_JOB"
  ],
  "jti": "c182bce8-04f3-4501-a833-97d33ea9d8a0",
  "client_id": "angular"
}
```


Fig. 42. Security – Roles Payload

In the example of the **Fig. 43**, a user who does not have the same administrator privileges will not have access to some system features, in which case he will not be able to edit or delete.

The UI applies this rule over the buttons, which can disable or make them invisible.

Fig. 43. Security - user with few privileges.

4.5.2 Database

All frameworks are implemented towards a common database schema shown in **Fig. 44**.

For this project, MySQL was used, but does not mean that it is mandatory since it is using Hibernate for object conversion to SQL.

Fig. 44. Design database

Hibernate

Hibernate is an ORM framework originally started in 2001. Since its version 3.0 release it has become JPA certified for API 2.0 in addition to providing its own native API. Hibernate utilizes HQL or Hibernate Query Language in place of SQL to provide querying capabilities against its own data objects in a high level manner rather than interact directly with the database. Hibernate provides support for performance tuning through features such as lazy initialization/loading and various fetching strategies for handling relationships.

The Hibernate implementation consists of a configuration file and object classes.

The configuration file **application.properties Fig. 45** holds the information pertaining to target database type, connection details, class mappings and Hibernate specific settings. The classes given in the mapping are POJOs marked with annotations to define persistence properties. The file structure for the project is shown in **Fig. 46**.

```
spring.jpa.database=MYSQL
spring.datasource.url=jdbc:mysql://localhost/mycareer?createDatabaseIfNotExist=true&useSSL=false
spring.datasource.username=root
spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Fig. 45. Hibernate configuration file snippet.

```
@Entity
@Table(name = "jobs")
public class Jobs {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long code;

    @NotNull
    private String title;

    private String position;
    private String location;
    private String organization;
    private LocalDate publication;
    private String description;
    private String status;
```

Fig. 46. Defining an entity and mapping to a table.

The @Entity annotation is used to designate the class as a persistence entity as shown in **Fig. 46**. It is followed by the @Table annotation which sets the database table it is to be persisted; this may be omitted if the class name matches the table name.

@Id declares our integer id as the identifier property of the post entity. We also denote that this field's value is auto-generated by annotating it with @GeneratedValue.

4.5.3 Document

Documenting your code is crucial to help others understand it, and even to remind yourself how your own older programs work. Unfortunately, it is easy for most external documentation to become out of date as a program changes. For this reason, it is useful to write documentation as comments in the code itself, where they can be easily updated with other changes. Javadoc is a documentation tool which defines a standard format for such comments, and which can generate HTML files to view the documentation from a web broswer.

Javadoc Comments

In general, Javadoc comments are any multi-line comments ("/** ... */") that are placed before class, field, or method declarations. They must begin with a slash and two stars, and they can include special tags to describe characteristics like method parameters or return values. The HTML files generated by Javadoc will describe each field and method of a class, using the Javadoc comments in the source code itself. Examples of different Javadoc comments are listed **Fig. 47**.

```java
/**
 * The Class BasicSecurityConfig.
 *
 * Basic authentication Config
 *
 * This config was raplace by Oath
 *
 * @author Alexander Souza
 */
@Profile("basic-security")
@EnableWebSecurity
public class BasicSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }

    /**
     * Password encoder.
     *
     * @return the password encoder
     */
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

Fig. 47. Javadoc comments

Fig. 48. Javadoc HTML

Swagger UI

The Swagger UI is an open source project to visually render documentation for an API defined with the OpenAPI (Swagger) Specification. Swagger UI lets you visualize and interact with the API's resources without having any of the implementation logic in place, making it easy for back end implementation and client side consumption.

Swagger UI is a built-in solution which makes user interaction with the Swagger-generated API documentation much easier.

To use Swagger UI, one additional Maven dependency is required:

```
<!-- swagger2 -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.6.1</version>
    <scope>compile</scope>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.6.1</version>
    <scope>compile</scope>
</dependency>
```

Fig. 49. Swagger dependency

Within Swagger's response is a list of all controllers defined in your application. Clicking on any of them will list the valid HTTP methods (DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT).

Fig. 50. Swagger methods

Expanding each method provides additional useful data, such as response status, content-type, and a list of parameters. It is also possible to try each method using the UI.

Fig. 51. Swagger, testing a method GET

Fig. 52. Swagger, return testing a method GET

The Swagger UI provides a number of very useful features – which we've covered well so far here. But we can't really use most of these if our API is secured and not accessible.

Fig. 53. Swagger, return testing a security method GET

Swagger's ability to be synchronized with your code base is crucial. To demonstrate this, you can add a new controller to your application.

The Docket bean of your application can be configured to give you more control over the API documentation generation process.

```
@EnableSwagger2
@Configuration
public class SwaggerConfig {

    /**
     * Product api.
     *
     * @return the docket
     */
    @Bean
    public Docket productApi() {
        return new Docket(DocumentationType.SWAGGER_2).select()
                .apis(RequestHandlerSelectors.basePackage("com.mycareer.api")).paths(regex("/.*")).build()
                .apiInfo(metaInfo());
    }
```

Fig. 54. Swagger config

Fig. 55. Swagger UI

To better document the UI code JobsAdmin and JobsWeb (Angular), TypeDoc is a documentation generator for typescript projects.

TypeDoc runs the TypeScript compiler and extracts type information from the generated compiler symbols. Therefore you don't have to include additional metadata within your comments, TypeScript specific elements like classes, enumerations or property types and access modifiers will be automatically detected.

Fig. 56. TypeDoc

4.6 System Deployment

The best practice to deploy a spring boot and angular application is to separate the user interface code with the business logic. This provides decoupling of the client code with server code and hence the application becomes highly scalable and manageable. The fronted developer can continue with the fronted development in parallel with the backend developer. The backend code becomes free to use and integrate different technology stacks and it becomes available for multiple clients such as the same APIs can be reused for building android application and same can be integrated with third party clients too. It also reduces the downtime of your application. Whenever, your APIs are not available or down, your client application is still up.

But sometimes it becomes an overhead to manage two server for a small team and a small application. If a single full stack developer is handling all the UI and server related configurations, packaging fronted and backend application into a single web application is sometimes more helpful. Still, you can expose REST APIs and integrate angular fronted code within the same application and deploy to a tomcat and other mobile client can reuse the same APIs.

4.6.1 Deploy Joomla (AWS)

Joomla is arguably the second most popular CMS in the world, after WordPress. “Core” Joomla (i.e., Joomla itself) is the free and open source. The functionality included in core Joomla is quite extensive, but if that’s not enough, there are over 6,000 extensions that can be added to a Joomla site to extend its functionality. Some examples of extensions are more support for advertising, elaborate support for SEO, and adding an e-commerce store. Finally, the look and formatting of content in a Joomla site is controlled by Joomla template(s). Core Joomla comes with a few templates built in, but there are many more available that can be installed into a Joomla site.

In this was install WampServer on an instance of Windows Server in AWS, and then how to install Joomla into the WampServer environment.

This excellent free software consists of Apache web server, MySQL database, the PHP language (interpreter), as well as other helpful tools, such as phpMyAdmin.

Once the entire system environment is configured, only the Joomla installation steps remain.

In this case, accessing the server link where the page is located.

- http://54.210.4.37/mycareer/

Fill out all the required fields, Site Name you makeup, Admin Email, Admin Username and Password.

Fig. 57. Deploy Joomla website – Configuration

Database Configuration page. Use MySQLi for database type. Host Name is localhost. The username, password and database name you enter must match what you set up in phpMyAdmin.

Fig. 58. Deploy Joomla website – Database

The rest of this screen is a summary of what you entered in previous screens and also includes a Pre-Installation Check and Recommended Settings.

Fig. 59. Deploy Joomla website - Overview

Fig. 60. Deploy Joomla website - Congratulations

4.6.2 Deploy Angular (Heroku)

Angular CLI provides ng build command to compile the application into the output directory. This generated output is static HTML and JS files and can be served via any web server such as Nginx, Apache or custom servers.

When we push the code to Heroku, we want Heroku to automatically run this command. This can be achieved by changing the package.json, add the following in the scripts section of package.json.

```
package.json ×

{
  "name": "jobsweb-ui",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "node server.js",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "postinstall": "ng build --prod"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^4.0.0",
    "@angular/common": "^4.0.0",
    "@angular/compiler": "^4.0.0",
    "@angular/core": "^4.0.0",
    "@angular/forms": "^4.0.0",
    "@angular/http": "^4.0.0",
    "@angular/platform-browser": "^4.0.0",
    "@angular/platform-browser-dynamic": "^4.0.0",
    "@angular/router": "^4.0.0",
```

Fig. 61. Postinstall - Package.json

This will make sure that every time we push new code to Heroku, Angular automatically build the code and push it into /dist directory.

Next, we need to move angular-cli dependency from dev dependencies section to dependencies section.

This is to make sure Heroku install this as the dependency module in the Server.

```
dist
  color.c7e33805ffda0d32bd2a.png
  favicon.ico
  fontawesome-webfont.674f50d287a8c48dc19b.eot
  fontawesome-webfont.912ec66d7572ff821749.svg
  fontawesome-webfont.af7ae505a9eed503f8b8.woff2
  fontawesome-webfont.b06871f281fee6b241d6.ttf
  fontawesome-webfont.fee66e712a8a08eef580.woff
  index.html
  inline.e30a0ea469d9e4cb1b0d.bundle.js
  line.567f57385ea3dde2c9ae.gif
  main.e7c8eb07eb68a80163c1.bundle.js
  polyfills.a35706baac714ab5ded9.bundle.js
  roboto-v15-latin-regular.3d3a53586bd78d1069ae.svg
  roboto-v15-latin-regular.7e367be02cd17a96d513.woff2
  roboto-v15-latin-regular.9f916e330c478bbfa2a0.eot
  roboto-v15-latin-regular.16e1d930cf13fb7a9563.woff
  roboto-v15-latin-regular.38861cba61c66739c145.ttf
  scripts.27264fee733d380c2d05.bundle.js
  styles.08f25a8aaa4eb30e2099.bundle.css
  vendor.edceb5b105cef93ac5fb.bundle.js
```

Fig. 62. Angular – dist folder

Create a new file called Server.js and place the following code in it.

```
JS server.js ×
1  const express = require('express');
2  const app = express();
3
4  app.use(express.static(__dirname + '/dist'));
5
6  app.get('/*', function(req, res) {
7    res.sendFile(__dirname + '/dist/index.html');
8  });
9
10 app.listen(process.env.PORT || 8080);
11
```

Fig. 63. File Server.js

Login to your Heroku account and click on the 'New' button located at top right corner and then click on 'Create New App'.

Fig. 64. Heroku – Create a new app

Fig. 65. Heroku – List of apps

Download and install the Heroku CLI.

```
Email: alexpt2000@gmail.com
Password: ************
Logged in as alexpt2000@gmail.com
PS C:\Users\Alexander Souza\Desktop\GIT\4Year_MainProject_MyCareer\UI-Angular\JobsWeb> heroku apps:create mycareer-webjobs
Creating mycareer-webjobs... done
https://mycareer-webjobs.herokuapp.com/ | https://git.heroku.com/mycareer-webjobs.git

C:\Users\Alexander Souza\Desktop\GIT\4Year_MainProject_MyCareer\UI-Angular\JobsWeb> heroku git:remote -a mycareer-webjobs

PS C:\Users\Alexander Souza\Desktop\GIT\4Year_MainProject_MyCareer\UI-Angular\JobsWeb> git init
Initialized empty Git repository in C:/Users/Alexander Souza/Desktop/GIT/4Year_MainProject_MyCareer/UI-Angular/JobsWeb/.git/
PS C:\Users\Alexander Souza\Desktop\GIT\4Year_MainProject_MyCareer\UI-Angular\JobsWeb> git add .

PS C:\Users\Alexander Souza\Desktop\GIT\4Year_MainProject_MyCareer\UI-Angular\JobsWeb> git commit -m "Deploy"

C:\Users\Alexander Souza\Desktop\GIT\4Year_MainProject_MyCareer\UI-Angular\JobsWeb> git push heroku master
```

Fig. 66. Heroku CLI

4.6.3 Deploy API (Heroku)

The Spring Boot model of deploying standalone applications is a great fit for Heroku. You can use either Maven or Gradle to deploy a Spring application on Heroku.

Heroku supports Java applications packaged as WAR and JAR out of the box. The prerequisites are

- Create a Heroku account.
- Install the Heroku CLI.
- Set up Heroku locally using your Heroku login.

Once installed, you can use the Heroku command from the terminal to log in using the email address and password you used when creating your Heroku account:

```
C:\Users\Alexander Souza\Desktop\GIT\4Year_MainProject_MyCareer\API-Spring>heroku login
Enter your Heroku credentials:
Email: alexpt2000@gmail.com
Password: *********
```

Fig. 67. Login Heroku

```
C:\Users\Alexander Souza\Desktop\GIT\4Year_MainProject_MyCareer\API-Spring>heroku create mycareer-api
Creating mycareer-api... done
https://mycareer-api.herokuapp.com/ | https://git.heroku.com/mycareer-api.git
```

Fig. 68. Create a new app on Heroku

You can attach a MySQL database to your app by Add-ons Marketplace to see what is available.

Fig. 69. Add MySQL to the API

Now you can list the configuration variables for your app to display the URL needed to connect to the database, DATABASE_URL:

Fig. 70. JawsDB – Settings

```
mycareer.security.enable-https=true

spring.datasource.url={JDBC_DATABASE_URL}
spring.datasource.username={JDBC_DATABASE_USERNAME}
spring.datasource.password={JDBC_DATABASE_PASSWORD}

mycareer.origin-allowed=https://mycareer-webadmin.herokuapp.com

spring.profiles.active=oauth-security
```

Fig. 71. Database profile on API

Once the database add-on has been created, Heroku will automatically populate the environment variables JDBC_DATASOURCE_URL, JDBC_DATASOURCE_USERNAME, and JDBC _DATASOURCE_PASSWORD.

```
heroku config:set JDBC_DATABASE_URL=jdbc:mysql://z37udk8g6jiaqcbx.cbetxkdyhwsb.us-east-1.rds.amazonaws.com:3306/wkgt0lbjg7r2hj7o JDBC_DATABASE_USERNAME=kj5hcza3lok330ed JDBC_DATABASE_PASSWORD=gfo2yz94v3mhh5r8 -a mycareer-api
```

Heroku supports AWS-style Procfile. Create one in the application root directory (with capital P) and enter.

The Heroku deploy: jar command can pass additional parameters to the jar file, so it should be possible to execute.

Fig. 72. Heroku - Additional parameters (JAR)

Before you can deploy the app to Heroku, you'll need to create a Git repository for the application and add all of the code to it by running these commands:

- Git init
- Git add .

then

Fig. 73. Deploy – Commit API and Push to master

Now the API been Deploy.

```
1.0.0-SNAPSHOT.jar
remote:        [INFO] Installing /tmp/build_dd1552d9a1403262da0cc22c7e01c5ae/pom.xml t
remote:        [INFO] ------------------------------------------------------------------
remote:        [INFO] BUILD SUCCESS
remote:        [INFO] ------------------------------------------------------------------
remote:        [INFO] Total time: 16.191 s
remote:        [INFO] Finished at: 2018-02-19T14:36:04Z
remote:        [INFO] Final Memory: 43M/353M
remote:        [INFO] ------------------------------------------------------------------
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 81.4M
remote: -----> Launching...
remote:        Released v5
remote:        https://mycareer-api.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/mycareer-api.git
 * [new branch]      master -> master

C:\Users\Alexander Souza\Desktop\GIT\4Year_MainProject_MyCareer\API-Spring>
```

Fig. 74. Deploy process

Heroku automatically detects the application as a Maven/Java app due to the presence of a pom.xml file. It installed Java 8 by default, but you can easily configure this with a system.properties file as described in the Specifying a Java version Dev Center article. It will run your app using the default command.

You can view the logs for the application by running this command.

```
heroku logs --tail
```

```
$ heroku logs --tail
2014-09-11T17:16:53.118407+00:00 app[web.1]:  =========|_|==============|___/=/_/_/_/
2014-09-11T17:16:53.120445+00:00 app[web.1]:  :: Spring Boot ::        (v1.1.5.RELEASE)
2014-09-11T17:16:53.120470+00:00 app[web.1]:
2014-09-11T17:16:53.118202+00:00 app[web.1]:
2014-09-11T17:16:53.118216+00:00 app[web.1]:   .   ____          _            __ _ _
2014-09-11T17:16:53.118258+00:00 app[web.1]:  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
2014-09-11T17:16:53.118290+00:00 app[web.1]: ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
2014-09-11T17:16:53.118325+00:00 app[web.1]:  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
2014-09-11T17:16:53.118367+00:00 app[web.1]:   '  |____| .__|_| |_|_| |_\__, | / / / /
```

Fig. 75. Heroku print the log

Reload your application in the browser, and you'll see another log message generated for that request. Press Control+C to stop streaming the logs.

Using Spring Boot, Heroku and GitHub, a complete application can be created and deployed using free tools (or at least the free versions of paid tools). Heroku itself is very well documented and easy to use.

Now the API can access in:

- https://mycareer-api.herokuapp.com/

5 System Evaluation

To prove that this system was built on a solid foundation, several components of the application need to be analyzed. One of the main things that was needed was to perform allowing multiple concurrent users to access the system. The application will enable users to do add and view, without affecting other users.

One of the most important characteristics of any application is:

- Safety;
- Efficiency;
- Validation;
- Scalability;

5.1 Security

One of the most important point for this application is Security. Spring Security features with JWT, subject already described in the previous topic, have been implemented and used well throughout the application, users who do not have the credentials will not be able to the application, and users who do not have all the privileges, cannot access all features.

For this project will be use Spring Security + OAuth2 for securing REST API. Clients and user credentials will be stored in a relational database MySQL.

For implementation it will be necessary:

- Configure Spring Security + database.
- Create an Authorization Server.
- Create a Resource Server.
- Get an access token and a refresh token.
- Get a secured Resource using an access token.

5.1.1 OAuth 2.0

The OAuth 2.0 specification defines a delegation protocol that is useful for conveying authorization decisions across a network of web-enabled applications and APIs. OAuth is used in a wide variety of applications, including providing mechanisms for user authentication.

5.1.1.1 OAuth Roles

OAuth specifies four roles:

- Resource owner (the User) – an entity capable of granting access to a protected resource (for example end-user).
- Resource server (the API server) – the server hosting the protected resources, capable of accepting responding to protected resource requests using access tokens.

- Client – an application making protected resource requests on behalf of the resource owner and with its authorization.
- Authorization server – the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

5.1.1.2 Grant Types

OAuth 2 provides several "grant types" for different use cases. The grant types defined are:

- Authorization Code
- Password
- Client credentials
- Implicit

The overall flow of a Password Grant:

Fig. 76. The overall flow of a Password Grant

Based on CRUD operations for Applicants and Jobs objects ,we want to define following access rules:

PAYLOAD: DATA

```
{
  "user_name": "admin@mycareer.com",
  "scope": [
    "read",
    "write"
  ],
  "name": "Administrator",
  "exp": 1522761007,
  "authorities": [
    "ROLE_READ_SCHEDULE",
    "ROLE_ADD_APPLICANT",
    "ROLE_ADD_JOB",
    "ROLE_REMOVE_SCHEDULE",
    "ROLE_REMOVE_APPLICANT",
    "ROLE_ADD_SCHEDULE",
    "ROLE_REMOVE_JOB",
    "ROLE_READ_APPLICANT",
    "ROLE_READ_JOB"
  ],
  "jti": "c182bce8-04f3-4501-a833-97d33ea9d8a0",
  "client_id": "angular"
}
```


Fig. 77. Access rules

5.1.2 JWT

JSON Web Token (JWT) is a compact token format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. More information can be seen at 3.3.3 above

The API is now secured and supports authentication with JWTs. When the server is up and running, can test this out by querying https://mycareer-api.herokuapp.com/oauth/token, and should get a message saying "Access Denied." To authenticate ourselves, send a POST request to https://mycareer-api.herokuapp.com/oauth/token/ with our user's credentials in the body: {"username":"admin","password":"password"}.

For API tests it will not be necessary to have the UI fully developed, the tests can be performed through the Postman tool, described in 3.6.1 Postman.

5.1.3 What is a test in Postman?

Postman is a powerful tool used to test web services. It was developed for sending HTTP requests in a simple and quick way.

5.1.3.1 Testing the API

Now let's make our first call to the login endpoint so that we can get a token (we'll later pass the token to subsequent calls so the API knows that we are logged in). I see this is a POST request. If it were a GET, you'd be passing in your username and password through a URL. To ensure everything runs smoothly, make sure you have the following settings:

- **Method**: Post
- **URL**: https://mycareer-api.herokuapp.com/oauth/token
- Click on the Body tab and set the request body to form-data.

Value in the Body:

Fig. 78. Add body, key and value to request token

For testing the API it will be necessary to configure hears, passing Content-Type and authorization.

For authorization, the user and password for the identification of the client application accessing the API should be informed, to encrypt these values, using the website www.base64encode.org in **Fig. 76**.

Fig. 79. Add Authorization on Headers to request token

Client credentials are being sent to the API in BASE64.

Fig. 80. Decode Authorization value

In the response to this request, will get a token as part of the Authorization header, prefixed by "Bearer". Can copy this token, with the Bearer prefix, to issue POST requests to /applicants path.

Fig. 81. Get JSON Web Token with Postman

The copied token can be decoded on JWT.io web site. This can provides information's like authentications roles.

Fig. 82. Decode JSON web Token with JWT

Paste the JWT received access token to start get access to resources API (GET method).

Fig. 83. Request applicants

5.2 Login

Authentication is the system need of almost every application to protect it from unauthorized access.

The system will reject any user from the list and users.

Fig. 84. User - unauthorized access

5.3 Block unauthorized URL

The system will check the type of rule that the user has and will block even if the user tries to access the resource through the address bar.

In **Fig. 85**, **Fig. 86**, there is an admin user (administrator) and one with limited privileges (alexander), the limited user will be denied access.

Fig. 85. Block unauthorized URL (Administrator)

Fig. 86. Block unauthorized URL (Alexander)

Both users access the **https://mycareer-webadmin.herokuapp.com/jobs/newjob** feature, but only the Administrator has permission.

5.4 Redirect non-existent URL

All non-existent resources will be redirected to "404 Page not found!". E.g. **https://mycareer-webadmin.herokuapp.com/test**

Fig. 87. Redirect non-existent URL

5.5 Performance

BlazeMeter

BlazeMeter provides developers with tools using the Amazon Elastic Compute Cloud (EC2) service that provide mobile, web application, website, web-service or database testing that can simulate thousands of users.

As a paid service was used the free option that allows a maximum of 50 users, the images below illustrates the results obtained, proving robustness and stability.

Fig. 88. BlazeMeter – Original Test Configuration

Fig. 89. BlazeMeter – Dashboard Summary

Fig. 90. BlazeMeter – Timeline Report

Label	Hits		Response Time	
	Total	Failed	Avg.	90%
ALL	554	0	24,144	46,303
JobsWEB	554	0	24,144	46,303

Fig. 91. BlazeMeter – Summary

6 Conclusion

The main goal of this project is to provide the different technologies together to develop an application that exposes its functionality via a REST API. Based on a REST API specification, it is possible to generate the skeleton of the application, including the data access layer, the controller/service layer, the view layer (Angular 4 user interface) and role-based access code. The code generator produces a fully functional application using Spring Boot and views in HTML which is powered by Angular 4 and PrimeNG. All the necessary files that are needed for the front-end are already included in the generated view files.

The support to JSON Web Tokens is added to the application by levering the Spring security framework. Since the front-end is powered by Angular 4, Token functions are also added to the Angular 4 codes. The Token functions in Angular communicate with the Spring security and provides the authentication and authorization to the API. The communications are made as REST calls. There are also front-end views and back-end code to manager the application dynamically. All the codes that are generated are the skeleton of the RESTful API. The developer can able to add/modify the existing code.

The application

MyCareer is a system that fully supports the recruitment and selection process of HR, provides an internal system to manage resumes, jobs, and a tool that can be integrated with the current system of the company that will make the company's own web site in an online recruiter of candidates.

MyCareer is an application that, in addition to being scalable, uses the best in development, appearance, security and robustness frameworks.

The application allows features like:

Applicants

- View available jobs.
- Apply the selected work through the website.
- Submit the email to the job alert receipt

Administrator

- Manage all available jobs.
- View all applicants.
- View applicants per job.
- Process of evaluation of the candidate in the act of the interview.
- Assessment with technical questions for candidates, divided by topics.
- Manage schedule for interviews and other appointments.

Several new technologies that were used that hadn't been used before, prior to commencing software development. For the project management side of things, GitHub was used to manage Milestones and Issues.

The technologies are:

- IDE - Development environment
    - Spring Tool Suite
    - Visual Studio Code

- API - Back end

    - Spring Security
    - JSON Web Token (JWT)
    - Spring Boot
    - Maven
    - Hibernate
    - Flyway
    - Swagger UI

- Database Environment
    - MySQL

- Front end
    - Joomla
    - HTML5 and CSS
    - Angular 4
    - PrimeNG - is a collection of rich UI components.

- Cloud services
    - AWS
    - Heroku

- Test environment
    - Postman
    - Swagger UI

- Repository
    - GitHub

- Documentation
    - JavaDoc
    - JSDoc

6.1 Future work

7 References

References and Appendices

- *References should all be in IEEE format and correctly collated.*
  - Use Google Scholar's settings to specify Bibtex as the "Import into…" option.
  - Can build the bibliography as a separate Latex file or as an inline document.
- *The appendices should contain the URL of the GitHub repo and a full set of installation instructions if applicable.*

References

[1] "Spring Security," [Online]. Available: https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/.