BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE IN ENGLISH

DIPLOMA THESIS

MICROSERVICES: LEARNING MANAGEMENT SYSTEM

Supervisor

Lect. Dr. Horea Greblă

Author

Achim Timiș

Cluj-Napoca

2017

UNIVERSITATEA „BABEŞ-BOLYAI”
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ IN LIMBA ENGLEZA

LUCRARE DE LICENŢĂ

MICROSERVICII: SISTEM DE MANAGEMENT AL ÎNVĂȚĂRII

Conducător științific
Lect. Dr. Horea Greblă

Absolvent
Achim Timiș

Cluj-Napoca
2017

Table of contents:

Chapter 1. Introduction

We have been using technology so much these days in each and every domain of our lives, be it education or the regular household work, to improve and advance the quality of certain aspects. Technology in education was a debatable topic amongst the society. Everyone had their own views on modernizing education and making it technology aided, but gradually it was embraced by the educational institutes and communities and the common reason was that with technology, educators, students and parents have a larger variety of learning tools at their fingertips. Teachers can collaborate to share their ideas and resources online, students can develop valuable research skills at a young age while access to an expanse of material.

In the context of elearning, tests and quizzes play an essential role and provide a wide array of benefits for both the learner and the instructor. Automating this type of work provides several benefits for both sides. The more early advantages come for the teacher since most of the hassle-free task as corrections can be automated via specific software system solution. E-learning systems usually provide automatic grading if the correct options for an individual question is known or some of them are equipped with keyword tracking tools that grade depending on what has been mentioned in the essays. This isn't a foolproof system but it helps save some time in comparison with long grading sessions.[1]

The process of quizzing can also be made unique by randomizing question and answer order. This is really useful when a student has to re-do a test such that that test is not completed from memory, but rather by thinking through the correct solution once again. Another area that is greatly improved with online quizzes is the feedback aspect. Since the instructor has to provide tips and feedback for each individual learner to improve on particular areas, the Learning Management Systems can provide thorough analysis of student reports and provide automatic progress tracking, rankings, basicly business intelligence solutions. That way, an instructor has the ability to analyze which students scored highest/ lowest, and which questions were hardest/ easiest for the majority of students. This kind of reporting allows the instructor to improve the curriculum if deemed necessary.

Another very important benefit, that can be easily overlooked is the fact that going from hard-copy tests/ quizzes to offering the same capabilities online, reduces consumption of goods such as paper, the latter being more environment friendly.

In my approach of implementing this Learning Management System, described with this paper, I aimed and managed to partially address all the points mentioned above. The role of this application is to make the quiz taking process easier and more efficient, both for the teacher and for the student. It is a web application that can be easily accessed by any device with an internet connection. It allows professors to create different type of quizzes, each type having an existing template to ease the creation process, send those quizzes to the interested parties and go through a set of useful graphs describing the impact of his curriculum. The student has access to all the quizzes, can submit his response on a particular topic, receiving either instant results and feedback or the old fashioned way of waiting for his input to be graded.

This paper is structured in 6 chapters, as follows:

- **Chapter 1: Introduction -** offers a short description in the context of the theme of the paper and the need of these kind of software solutions
- **Chapter 2: Problem Description -** talks about the context, the role and the motivation that drove the choosing of this subject. It also offers a look into the functional specification of the application.
- **Chapter 3: State of the art -** it lists existing implemented approaches to this subject, and the advantages and disadvantages of each compared to my solution.
- **Chapter 4: Technologies -** presents the technologies used and how each of them helped in end to the solution
- **Chapter 5: Solution Description -** describes the general architecture, the most important use case flows and how they translate to the software code level.
- **Chapter 7: Conclusions -** offers the conclusions resulting from the implementation, the most important takeaways and also, list some functionalities that could be added in the future, debates the ease of addition and the advantages each would bring.
- **Chapter 8: Bibliography -** lists all the sources that were used in redacting this paper.

Chapter 2. Problem Description

2.1 Context

Examination is an important part of higher education. The examination methods and questions have a large impact on how and when students study and what they learn. Examination should not only be used as a control that a student is qualified, but also as an educational tool to influence the learning process. If the assessment is mainly for factual knowledge, the students will primarily learn, memorize and recall facts and details. Such tests may have a high reliability, i.e. are easy to grade, but the validity might be low. For measuring that the aims and objectives of higher education have been fulfilled, assessments need to include tests of understanding (such as ability to interpret, exemplify, summarize, compare and explain), as well as ability to apply, analyse, evaluate and synthesize; assessment of skills, such as ability to communicate, should also be included.[2]

The standard examination method and the most commonly used is the written close books exam, including short-answer and essay questions. Short-answer questions are usually easy to judge and grade, but mainly test how students recall specific facts.

In recent times the Computer-based Assessment grew in popularity, with the growth of technology, and provides several improvements over the existing evaluation methods like getting an immediate feedback on their answers or the fact that the tests can also be used for continuous self-assessment. But maybe the most important thing it bring to the game is that a Computer-based Assessment Platform is a system that can be continuously upgraded and improved, and may provide, in the end, what is being said as the best examination method: using a variety of them, if possible within each course, but at least within the education programme. So the best assessment tool is to combine written exams, open-books exams, multiple-choice tests and many others.

Making a good online Learning Management System for education is a challenging task, that may not always improve all the aspects of the traditional ways, but seeing the advantage that Computer-based Assessment can provide, and with the continuous evolution of technology, there is much potential to be unlocked and much room for innovation.

2.2 Specification, Objectives and Motivation

The main purpose of this paper is to develop a system aimed to provide an online Learning Management System to help the process of learning, both for the instructor and the learner, starting with a support of several types of quizzes,.
Another important role had my interest of studying microservices and the architecture of the applications based on this software-as-a-service model, that is slowly but steady becoming a standard in modern, scalable software solutions.

My motivation is driven by the fact that the commonly used evaluation methods are becoming obsolete and in many cases do not really help the two parties as much as they should be. The world around us is evolving, our ways of thinking is evolving too, so we should improve and innovate “our ways of doing” too. A more personal motivation is trying out a bunch of new and successful open-source technologies, thus improving my knowledge and experience as a software developer and putting them all together in a result that has actually a positive real-world impact.

The problem required the solution to be easily accessible, therefore it is a web application, whose two main focus points are the instructor and the learner. Each party has access to this application via an user account, every interaction with the system requiring them to be logged in. The code for this application is open-source so there is support for internal and external improvements that are even more encouraged and easy to do due to the adopted microservice architecture.

The instructor can easily create different types of quizzes and assign them to students enrolled on the platform. If at the quiz creation, each question is provided with the correct answer/s, then the

grading process is automated by the system and will give the student a direct result. Other types of quizzes require manual grading/evaluation which the teacher can also do directly from his account. At the end, he can view different charts, diagrams and graphs that provides him with a better understanding of the quiz results, progress of each student and take aways for future quiz design ideas.

The learner has access to all the quizzes on the platform, and can easily send a solution to each of them. He can get instant results or wait for the teacher that created the quiz to grade/evaluate his input. Also, he has access to a number of auto-generated charts and diagrams that provide information about his progress so far or his results in comparison with the results of others.

The application's goal to improve quizzing solutions is achieved by automating various parts of the process, like the creation through the use of predefined templates, automatic grading and result publishing and providing, through the use of elasticsearch and kibana, a way to better understand and evaluate the learning process.

Chapter 3. State of the art

There is a high number of existing online quiz-management platform on the internet that provide support for a limited number of quiz or survey types. However, much of them either provide a limited version where the most important features are added only in the paid option or they don't really encapsulate the functionalities needed to support a quizzing management system.

To better understand the context, here is the list of the most popular quiz types:

- *Single option* - where there is a list of potential correct answers listed and you can choose only one
- *Multiple options* - there can be multiple correct answers stated and you need to find all of them
- *Input answer* - where the correct answer had to be written and motivated essay style.
- *Timed Single/Multiple options* - where there is a predefined time limit to completing the quiz
- *Deadline w/ any options* - where there is a deadline until you can submit your response
- *YES or NO* type of questions

When I first started to look into this area, I've noticed that there is a high number of free online survey solutions mainly designed for games, and a low number of customizable free quiz software solutions. Usually the free subscription to the platform imposes a high number of limitations, like the number of question in a quiz, or the number of quizzes you can create.

Google Forms is also worth mentioning, since it is the most popular and provides support for custom forms for surveys and questionnaires at no extra cost. It does not quite provide the tools for a learning management system but it was a great example of accessibility.[3]

Another example of a robust, secure and integrated system to create personalised learning environments is *Moodle*, a platform that is actually used by Babes-Bolyai for assigning

homeworks, grading, different types of workshops, integrated notification system and offering course support materials. It can has a default mobile-compatible interface and cross-browser compatibility making the content on the Moodle platform easily accessible and consistent across different web browsers and devices.[4]

All the paid, close-source and partial implementations made me want to provide an open-source solution to this problem even more, and it might aswell have been the decisive factor for this paper's subject. I qualify my solution to be very easy to be improved due to the use of very popular frameworks and tools like Spring-Boot, Angular 2, Postgresql, ElasticSearch, open source code and microservice architecture from Spring-Boot-Cloud.

The most important addition to the existing solutions on the market are the automatic quiz correction by the backend system and the integrated dashboard and reporting tool provided by Elasticsearch Kibana. As for the disadvantages, I think the fairly small number of use cases is the one that stands out.

Chapter 4. Technologies & Tools

In order to better structure the explanation of the technologies and tools I have used in this application, it will be separated in multiple parts to better reflect the actual project architecture, the big three areas: the front end web application, the backend business logic and the integrated reporting and analysing tools.

4.1 Backend Technologies

For the backend part of the application I have used several technologies to help me model the business logic of the application. The code is written in the Java 8, using the Spring Boot framework, having a microservice architecture with functionalities provided Spring Cloud Netflix and as a database, it uses Postgresql. In the next part I am going to briefly introduce each of them, what they bring to the table, how and where I have used them in the application.

4.1.1 Spring Boot

Spring Boot is a new framework from the team at Pivotal, designed to simplify the bootstrapping and development of a new Spring application. [5]

It is build from the ***Spring Framework***, which is a popular open source platform created in 2003, with the purpose of simplifying coding of any application written in the Java programming language. It is mostly used for writing Java EE(Enterprise Edition) applications. 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 platform.[6][7]

The biggest and best additions of *Spring Framework* are:

- ***Inversion of control container & Dependency Injection*** - IOC manages java objects from instantiation to destruction through its BeanFactory. Java components that are instantiated by the IoC container are called beans, and the IoC container manages a bean's scope, lifecycle events, and any AOP(Aspect Oriented Programming) features for which it

has been configured and coded. *Dependency Injection* can be done through setter injection and constructor injection and it basicly means that, for example in a method with an object as a parameter, the method does not have a direct dependency on a particular implementation of that object; any implementation that meets the requirements can be passed as a parameter.

- ***Data access***: working with relational database management systems on the Java platform using JDBC and object-relational mapping tools and with NoSQL databases.
- ***Model–view–controller (MVC)***: an HTTP- and servlet-based framework providing hooks for extension and customization for web applications and RESTful Web services.
- ***Testing***: support classes for writing unit tests and integration tests

Spring Boot is Spring's convention-over-configuration solution for creating stand-alone, production-grade Spring-based Applications that you can "just run". At its most fundamental level, Spring Boot is little more than a set of libraries that can be leveraged by any project's build system. It takes an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.[8][9]

The framework takes an opinionated approach to configuration, freeing developers from the need to define boilerplate configuration, become one of the most popular framework for microservice developments, as you can see in Fig. 4.1.

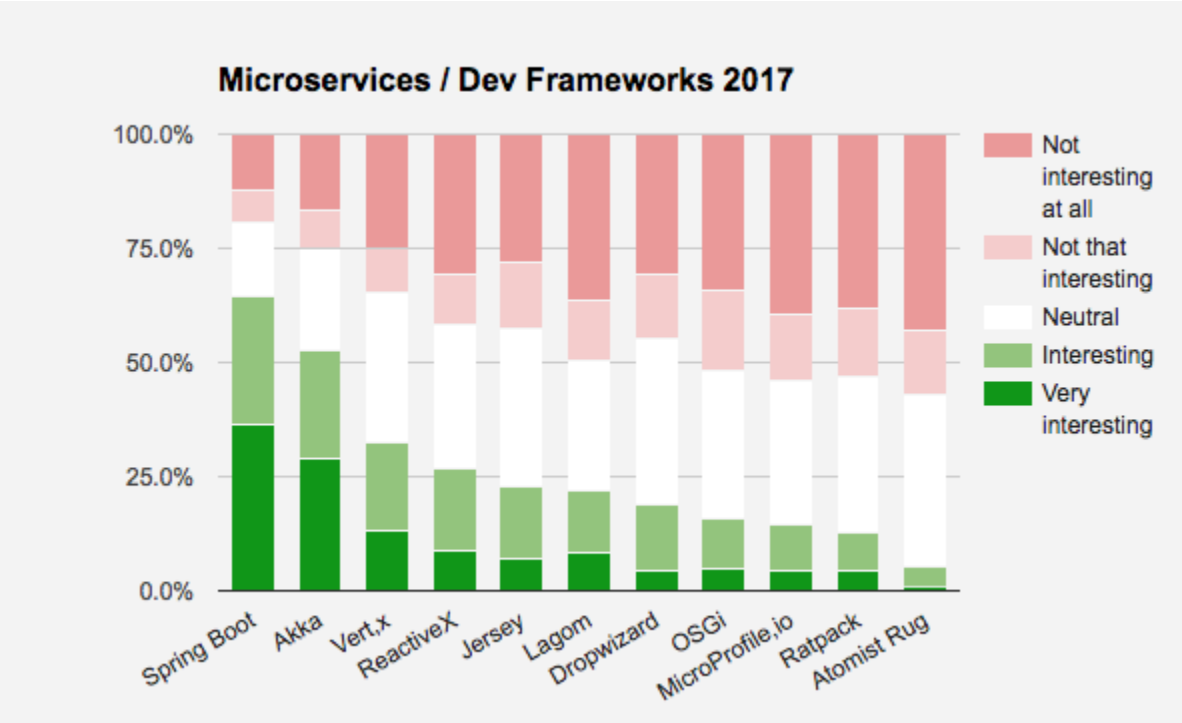

Fig. 4.1 *Spring Boot Microservice popularity[10]*

The best additions of ***Spring Boot*** are:

- ***Embed Tomcat server*** (no need to deploy WAR files)
- ***'starter' POMs*** to simplify your Maven configuration
- ***no more XML configurations*** required
- ***metrics, health checks and externalized configuration*** provided as production-ready features
- ***command-line interface***, which can be used to run and test Boot applications

4.1.2 **Spring Cloud Netfli**x

Spring Cloud Netflix provides Netflix OSS integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other Spring programming model idioms. With a few simple annotations you can quickly enable and configure the common

patterns inside your application and build large distributed systems with battle-tested Netflix components.[11]

Netflix is the industry leader in developing microservice architectures. One of the core tenets of the way Netflix builds systems is that any component should be able to suffer a fault, yet the system should keep running. Netflix built many of the components to achieve this fact, tested and validated them in production, and open-sourced them, spring cloud netflix being the Spring adaptation of those features.[12]

Spring Cloud Netflix features:

- *Service Discovery*: an embedded Eureka server created with declarative Java configuration and multiple eureka instances. Clients can discover those instances using Spring-managed beans. This takes care of the inter-services communication, provides some default load balancing and fault tolerance and enables most of features discussed in this list.
- *Circuit Breaker: Hystrix:* clients can be built with a simple annotation-driven method decorators. Circuit breakers can be placed between services and their remote dependencies. If the circuit is closed, calls to the dependency pass through normally. If a call fails, that failure is counted. If the number of failures reaches a threshold within a configurable time period, the circuit is tripped to open. While in the open state, calls are no longer sent to the dependency, and the resulting behavior is customizable (throw an exception, return dummy data, call a different dependency, and so on). Hystrix also emits a metrics stream from each breaker providing important telemetry such as request metering, a response time histogram, and the number of successful, failed, and short-circuited requests, which are available in an accessible dashboard.
- *Declarative REST Client: Feign* creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations. This is used throughout the project to connect one client microservice to the public API of another, for communication.
- *Client Side Load Balancer: Ribbon* : The load balancing provided by Eureka is limited to round robin. Ribbon provides a sophisticated client-side library with configurable load

balancing and fault tolerance enables additional load balancing algorithms such as availability filtering, weighted response times, and availability zone affinity.

- *External Configuration*: provides a centralized configuration service that is horizontally scalable. It uses as its data store a pluggable repository layer that currently supports local storage, Git, and Subversion. By leveraging a version control system as a configuration store, developers can easily version and audit configuration changes.
- *Router and Filter*: automatic registration of Zuul filters, and a simple convention over configuration approach to reverse proxy creation.

4.1.3 **Postgresql**

PostgreSQL is an object-relational database management system (ORDBMS) ,developed by the PostgreSQL Global Development Group, with an emphasis on extensibility and standards compliance. As a database server, its primary functions are to store data securely and return that data in response to requests from other software applications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications (or for data warehousing) with many concurrent users.[13]

PostgreSQL is ***ACID-compliant*** which stands for Atomicity, Consistency, Isolation, Durability and it is *transactional*, which is a processing type in computer science that is divided into individual, indivisible operations called transactions. Each transaction must succeed or fail as a complete unit, it can never be only partially complete.

PostgreSQL has updatable views and materialized views, triggers, foreign keys; supports functions, stored procedures and other expandability.

In this application, Postgresql is used as the only database, having 3 schemas : “user-service”, “quiz-service” and “quiz-main-service” used by the corresponding microservice. The connection between the spring application and the database is made by specifying the jdbc driver, the location and authentication credentials, in the configuration files.

4.2 Frontend Technologies

The frontend part of the quiz application is written in Angular 2 framework using Typescript. It communicates with the backend system through the “quiz-main-application” proxy spring-boot microservice, which handles the business logic and handles the communication with the other microservices.
Next up, more details about Angular 2, Typescript and how it helped me build a very modern and stable web application.

4.2.1 Angular 2

Angular (commonly referred to as "Angular 2+" 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 to address all of the parts of the developer's workflow while building complex web applications. Angular is a complete rewrite from the same team that built AngularJS. It is a development platform for building mobile and desktop web applications using Typescript/JavaScript (JS) and other languages.[14][15]

Angular 2 is the most advanced framework for the web. Angular has rebuilt the entire framework in TypeScript, so it is entirely new for a programmer to start using.

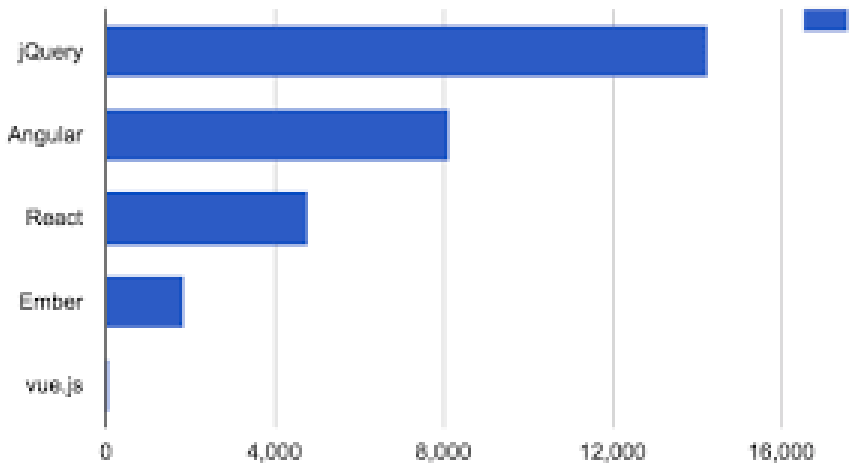

Fig. 4.2. *The most popular javascript frameworks[16]*

The primary objective of Angular 2 is to give developers an easy, detailed framework to develop an effective way of doing code, without any code complexity or delays. It has improved many things from previous versions such as making components simpler syntactically and semantically than they were in Angular 1. It has become one of the most popular javascript framework in 2017, as you can see in *Fig. 4.2*, only second to jQuery.[17][18]

The backend mechanism of Angular is grabbed by the ideas from Backbone.JS, Knockout, and many other web frameworks. The overall architecture of Angular 2 consists of Module, Component, Template, Metadata, Data Binding, Service, Directive, and the Dependency Injection.

For me, some of the most compelling features of Angular were:

- ***Typescript*** - Angular 2 and TypeScript are bringing true object oriented web development to the mainstream, in a syntax that is strikingly close to Java 8. TypeScript ensures safer code.

  TypeScript is developed by Microsoft and is the type superset of JavaScript that compiles your code into plain JavaScript. TypeScript can easily track the bugs in your code with the optional static type-checking along with the latest ECMAScript features.

- ***Dependency Injection*** - Dependency Injection works when you need to import any dependency in your application. It is the way to give a new object of a class with the required dependency. Mostly, dependencies are services. To provide the new component with services, Angular uses DI (Dependency Injection).

  It automatically tells about the services by looking into constructor parameters. And when it creates the component then it will ask an Injector for the service. If requested service is not in the container, Angular inject will auto-create and inject into your component, as it can be seen from the code example below, with the 3 dependencies in the constructor.

```
class LoginComponent implements OnInit {
 ...
 constructor(
   private route: ActivatedRoute,
   private router: Router,
   private authenticationService: AuthenticationService
 ){}
...
}
```

- ***Route guards*** **:** Protecting routes is a very common task when building applications, as we want to prevent our users from accessing areas that they're not allowed to access, or, we might want to ask them for confirmation when leaving a certain area. Angular's router provides a feature called Navigation Guards solve exactly that problem.[19] There are four different guard types we can use to protect our routes and I have used two of them:

- *CanActivate* - Decides if a route can be activated, was used to prevent access to some pages if the user is not logged in or the teacher tries to access student only pages. The example below states that the route "*/professor/results"* (so the teacher related information) can only be reached while logged in as user with the user roles: TEACHER.

  {path : 'professor/results', component: QuizResultsListComponent, canActivate: [LoginRouteActivator], data: { roles: [TEACHER]}},

- *CanDeactivate* - Decides if a route can be deactivated. It was used in the student's "solve quiz" form to prevent him to refresh or change the page if he had not submitted his response, to not loose the answered inputs. Instead a pop-up yes/no would appear where he was asked if he was sure to move to another page.

- ***Reusable services*** - special classes that can be injected in component acting more like a Java service. My *QuizServiceComponent* reusable service provides all the methods used to communicate with the backend system for quiz related requests and *UserServiceComponent* is injected in the login component to handle authentication related backend calls.

4.3 Reporting & Analysis Technologies

The application uses the Elastic Stack products to generate useful diagrams, graphs and projections. Elastic Stack is a group of open source products from Elastic designed to help users take data from any type of source and in any format and search, analyze, and visualize that data in real time.[20]

As part of the widespread movement towards open source software, the ELK Stack is rapidly moving from a niche platform to the most common log management platform in the world. It has become much popular that the previously market leading biggest enterprise grade solution Splunk, as you can see from Fig. 4.3.

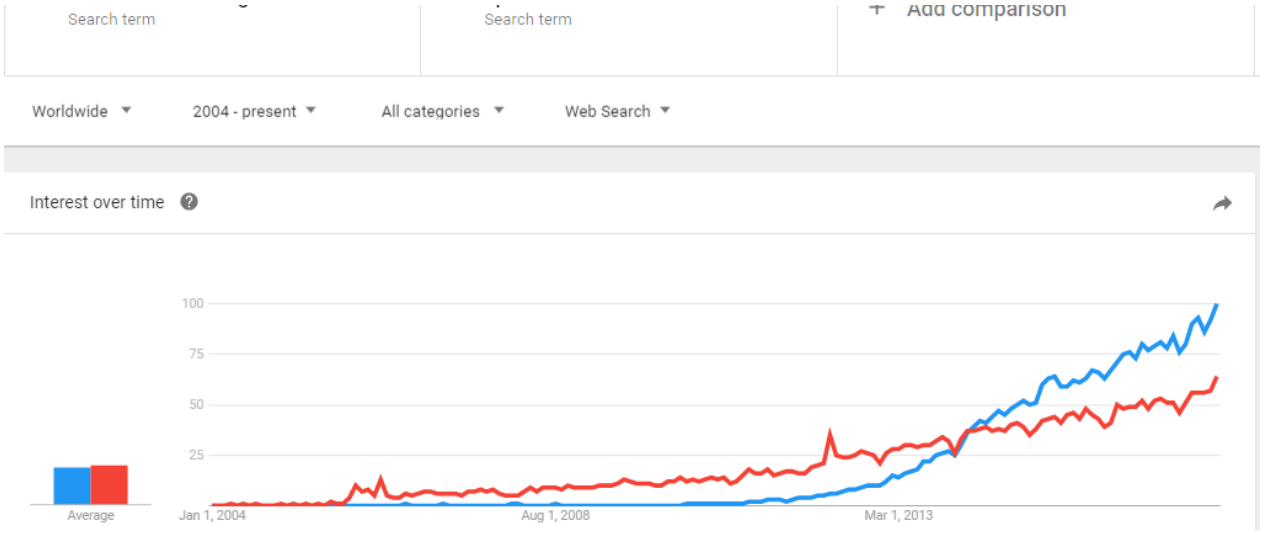

Fig. 4.3 *ELK vs Splunk popularity*

To give you a better understanding of the widespread adoption of ELK in various industries, here is a short list that shows how various companies are using ELK right now:

- ***NETFLIX*** - Netflix heavily relies on ELK within various use cases to monitor and analyze customer service operations and security logs. The company chose Elasticsearch for its automatic sharding and replication, flexible schema, nice extension model, and ecosystem with many plugins.[21]
- ***Stack Overflow*** - uses Elasticsearch as a means to support full-text search capabilities. Stack Overflow is using Elasticsearch because it performs better on SSDs and that Lucene.net could not handle the company's workflows as a result of locking issues.
- ***LinkedIn*** - The business-focused social network uses ELK to monitor performance and security. The IT team integrates ELK with Kafka to support their load in real time.

In this application I have used Elasticsearch, Logstash and Kibana Dashboard, such that I could parse my log-files and get a meaningful, easy to read bussiness valuable result.

4.3.1 Elastic Search

Elasticsearch is a RESTful distributed search engine built on top of Apache Lucene and released under an Apache license. It is Java-based and can search and index document files in diverse formats. It lets you perform and combine many types of searches: structured, unstructured, geo, metric, in a very large number of data. Elasticsearch aggregations let you zoom out to explore trends and patterns in your data.[22]

The process starting with indexing your JSON documents, then you can make a query and retrieve them, with no configuration needed. One of the reasons is that it's schema-less, which means it uses some nice defaults to index your data unless you specify your own mapping. For more precision it has an automatic type guessing mechanism which detects the type of the fields you are indexing.

Here is an example of creating an index on website: we choose a type called logwith the ID 123 and the data under JSON format, then the index request looks like this:
Request:

```
PUT /website/log/123
{
  "title": "My first log entry",
  "text":  "test",
  "date":  "2017/19/06"
}
```

the Response from elasticsearch being :

```
{
   "_index":    "website",
   "_type":     "log",
   "_id":       "123",
   "_version":  1,
```

```
  "created":   true
}
```

indicating that document has been successfully created and indexed.

Retrieving the document after indexing is just as easy, we use the same _index, _type, and _id, but the HTTP verb changes to GET:
Request:

```
GET /website/log/123?pretty
```

Response:

```
{
  "_index" :   "website",
  "_type" :    "log",
  "_id" :      "123",
  "_version" : 1,
  "found" :    true,
  "_source" :  {
     "title": "My first log entry",
     "text":  "test.",
     "date":  "2017/19/06"
   }
}
```

When it comes to search, elasticsearch provides its own Query Domain Specific Language through which you can express whatever complex query you want. You pick the query that you need and write it in JSON format. Some of the queries allow you to nest other queries as well. The query DSL is intuitive and powerful at the same time.

To better present how easy and intuitive is the searching process let's take a quick example on the data we created earlier.

The following request we will return the first blog entry, and will fetch it given the title parameter:

```
POST /website/_search?pretty
{
  "query": { "match": { "title": "My first log entry" } }
}
```

The query DSL is especially powerful and provide nearly instant results, while having a syntax somewhat similar to the *SQL SELECT FROM* statements. There is support for much more advances queries than the one shown in example, that work by filtering by specific fields, express complex searches with AND/OR, numerical ranges or patterns/

4.3.2 Logstash

Logstash is a data collection engine that unifies data from disparate sources, normalizes it and distributes it. The product was originally optimized for log data but has expanded the scope to take data from all sources.

In the quiz application presented in this paper, logstash indexes the json-formatted log lines, filters them to extract the most valuable information and indexes this information into Kibana. Logstash filters parse each event, identify named fields to build structure, and transform them to converge on a common format for easier, accelerated analysis and business value. This process is managed by a dedicated configuration file.

Log-line example:

{"@timestamp":"2017-06-19T22:14:30.316+03:00","@version":1,"message":"{\"quiz_name\":\"quiz\",\"quiz_type\":\"SINGLE_ANSWER\",\"questionText\":\"What is love?\",\"questionScore\":2.0,\"isAnsweredCorrectly\":false,\"student\":\"user1\",\"teacher\":\"prof1\"}","logger_name":"analytics","thread_name":"http-nio-8002-exec-9","level":"DEBUG","level_value":10000}

The configuration file used by logstash is composed in three major parts: the input section, the filter section and the output section.

```
input { file {
   path =>  [ "./quiz-service/analytics.log" ]
   codec => json { charset => "UTF-8" }
   start_position => "beginning"
   sincedb_path => "NUL"
 }}
```

In the **input** part, we specify the log-files we want to index with the *path* attribute, we specify the encoding and the fact that we want all the content of log file to be taken into consideration, so starting from the beginning.

```
filter{json{
      source => "message" }}
```

In the **filter** part, we specify to logstash that we want the content of the “message” parameter to be parsed as json and index each parameter as an individual one, so our fairly unstructured data becomes more structured and easy to work with.

```
output {
   elasticsearch {
     action => "index"
      hosts => "http://127.0.0.1:9200"
      index => "quiz-question"
      sniffing => false
      manage_template => false
   }
   stdout {}
}
```

In the **output** part we specify that our new filtered data should be indexed into elasticsearch, with the index name “quiz-question”. This will make our structured data available in the Kibana

Dashboard under the specified index name, ready to be used for different analysis and reporting methods.

4.3.3 Kibana

Kibana is an open source data visualization and exploration tool from that is specialized for large volumes of streaming and real-time data. The software makes huge and complex data streams more easily and quickly understandable through graphic representation.

In this application, Logstash has log events being shipped into Elasticsearch, and Kibana is used to query and visualize them. It gives really good results, and becomes even more helpful the more data it has from which it can learn from.

In the images below, Fig. 4.4 and Fig. 4.5, it is shown the Kibana Dashboard, with the range of tools that are available for analysis and reporting and respectively some of the diagrams that this application uses to better describe a business process. For this application I have multiple graphs for instance to show a more detailed view of a quiz result situation, the impact of a professor's quiz on the students and timelines on when are quis responses generally submitted.

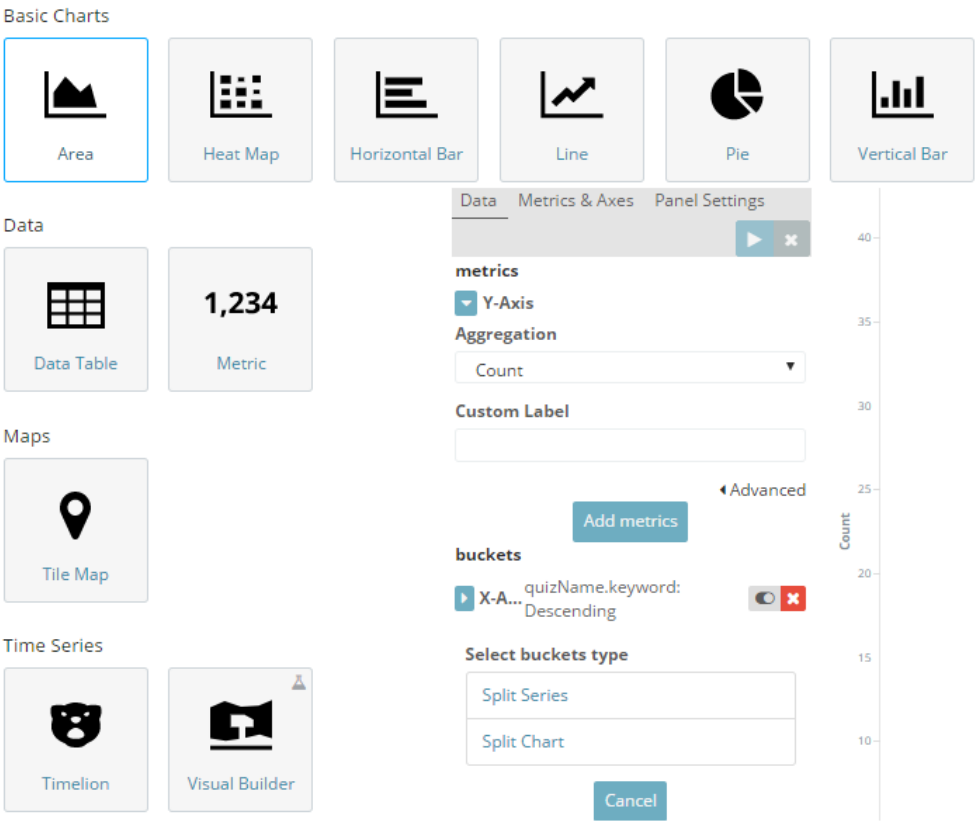

Fig. 4.4 *Kibana Dashboard. The range of different analytics tools available and the creation tool*

Fig. 4.5 *Different types of diagrams generated in kibana dashboard*

4.4 Tools

During the software development process for the Quiz management Application, I used a number of tools to help be write code easier, faster, more secure and bug free, and here are the ones that deserve a mention:

- ***Intellij Idea IDE*** - a very popular, capable and ergonomic Java IDE. In my opinion it is the most complete java editor, very fast once the code files have been indexed, very smart with intuitive auto-generated context dependent suggestions, highly customizable and upgradable with a wide range free plugin available for integration.
- ***Visual Studio Code*** - one of the most popular new javascript editors, very similar to Intellij in the context of customizability and upgradeability. I especially chose this editor due to the great support of Typescript and Angular syntax.
- ***PGAdmin III*** - it is an Open Source administration and development platform for PostgreSQL.
- ***Google Chrome*** - the most popular web browser in the world. I found myself using the Developer Tools very often when implementing the angular web application, as it was very helpful to debug my code.
- ***Postman*** - As they state in the official homepage: “Developing APIs is hard. Postman makes it easy.”. It is very useful to test and validate your APIs early on, making the current workflow faster and better.[23]
- ***Git and Github*** - I have used Git as version control system(VCS) and GitHub as a hosting service for Git repositories. The project can be found at: https://github.com/achimtimis/L-Project and https://github.com/achimtimis/L-project-frontend

Chapter 5: Solution Description

5.1 Introduction

Every new software solution follows, more or less, the current software trends and of those, the most popular seems to be delivering just-in-time services with frequent updates. With this is mind and also the fact that I wanted to tackle new technologies outside of the work or school sphere, I began researching the popular frameworks and design patterns that will also suit the theme and functional requirements of designing a Learning Management System. I ended up going with the new microservice architecture trend, using Spring Boot, Spring Cloud Netflix and Postgresql as the database system. For the frontend part I chose Angular 2, this decision being greatly influenced by the combo with Typescript, which was for me, as a backend developer, very compelling and surprisingly familiar as syntax with Java. Another rising technology I wanted to try was the Elastic Stack, with ElasticSearch, Logstash and Kibana. I had limited interactions with the Kibana dashboard before and I began to grew interest seeing how fast and efficient it is to extract and visualize valuable information from different types of data sources.

Another useful source of information I stumbled upon while researching technologies and patterns is the "*Twelve-Factor App*"(https://12factor.net/) which can be considered the "bible" of software-as-a-service apps. They describe it as "... a triangulation on ideal practices for app development, paying particular attention to the dynamics of the organic growth of an app over time, the dynamics of collaboration between developers working on the app's codebase, and avoiding the cost of software erosion."[24] A number of them deal with deploying or managing these applications which is intended more for the devops engineers, but nonetheless, I have followed some of them, as follows:

- I. ***Codebase*** : specifies that the application should be always tracked in a version control system, in my case Git, and the fact that there is only one codebase per app, codebase being any repo/s that share a root commit, but there will be many deploys of the app. Basically each individual microservice is a different app and should be part of a different codebase, under a different repository.

- ➔ 2. ***Dependencies*** : specifies that the app must never rely on implicit existence of system-wide packages and should declare the necessary dependencies in a dependency manifest file, in my case the pom.xml of each module.
- ➔ 3. ***Config*** : urges that the application must store config in environment variables that easy to change between deploys without changing any code and not as as constants in the code. Any change of configuration properties like credentials and resources must not affect the application code codebase. In the application described in this paper, the configuration files are stored in the Config Repository Microservice, on which each microservice executes calls to, on application startup or forced bean refresh, to read their assigned properties.
- ➔ 4. ***Backing services*** : this factor talks about the fact that there should be no distinction between local and third party services. "To the app, both are attached resources, accessed via a URL or other locator/credentials stored in the config. A deploy of the twelve-factor app should be able to swap out a local MySQL database with one managed by a third party (such as Amazon RDS) without any changes to the app's code." In my case this factor can only be applied so far for the Postgresql database and it is respected due to the fact that the connection string with credentials and the jdbc drivers all all specified in the configuration properties of each microservice, inside the Config Service microservice, which is a different codebase.
- ➔ 5. ***Logs*** : it encourages to treat logs as event streams, such that the developer will view this stream in the foreground of their terminal to observe the app's behavior. Furthermore I have used specific logs for a certain application business process, indexing them and sending them to the Elasticsearch Analysis system.

The rest are definitely worth looking into, dealing more with production ready and operations grade topics, with but I think I've managed to quickly describe some of the most important ones, their added value and the way I've actually implemented them to my services.

Another important software development principles I took as guidance are the SOLID principles (single responsibility, open-closed, Liskov substitution, interface segregation and dependency inversion) that stands for five basic principles of object-oriented programming and design. It

states that when applied together, the created system will be easier to maintain and extend over time.[25] *The Single Responsibility* principle states that a class should have only a single responsibility, principle that is also applied to methods, meaning that it should do exactly one thing, described clearly in the class or method name. The *Open/Closed* principle imposed the fact that software entities should be open for extension but closed for modification, basicly if there is a need of changes we should add new code that extends the currently available one, and not directly change the old one, thing that could potentially add unwanted refactorings. *Liskov substitution* principle
states that “objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program”, encouraging the use of inheritance and polymorphism for more reliable code. *Interface segregation* principle states that no client should be forced to depend on methods it does not use, so in this case, we should better implement a new contract that better represents its behaviour. *Dependency inversio*n principle talks about the fact that one should “depend upon abstractions, not concretions.”, describing efficient ways of decoupling between software modules, using interfaces or abstract classes and not actual implementation, to connect various parts of the application.

5.2 General Structure

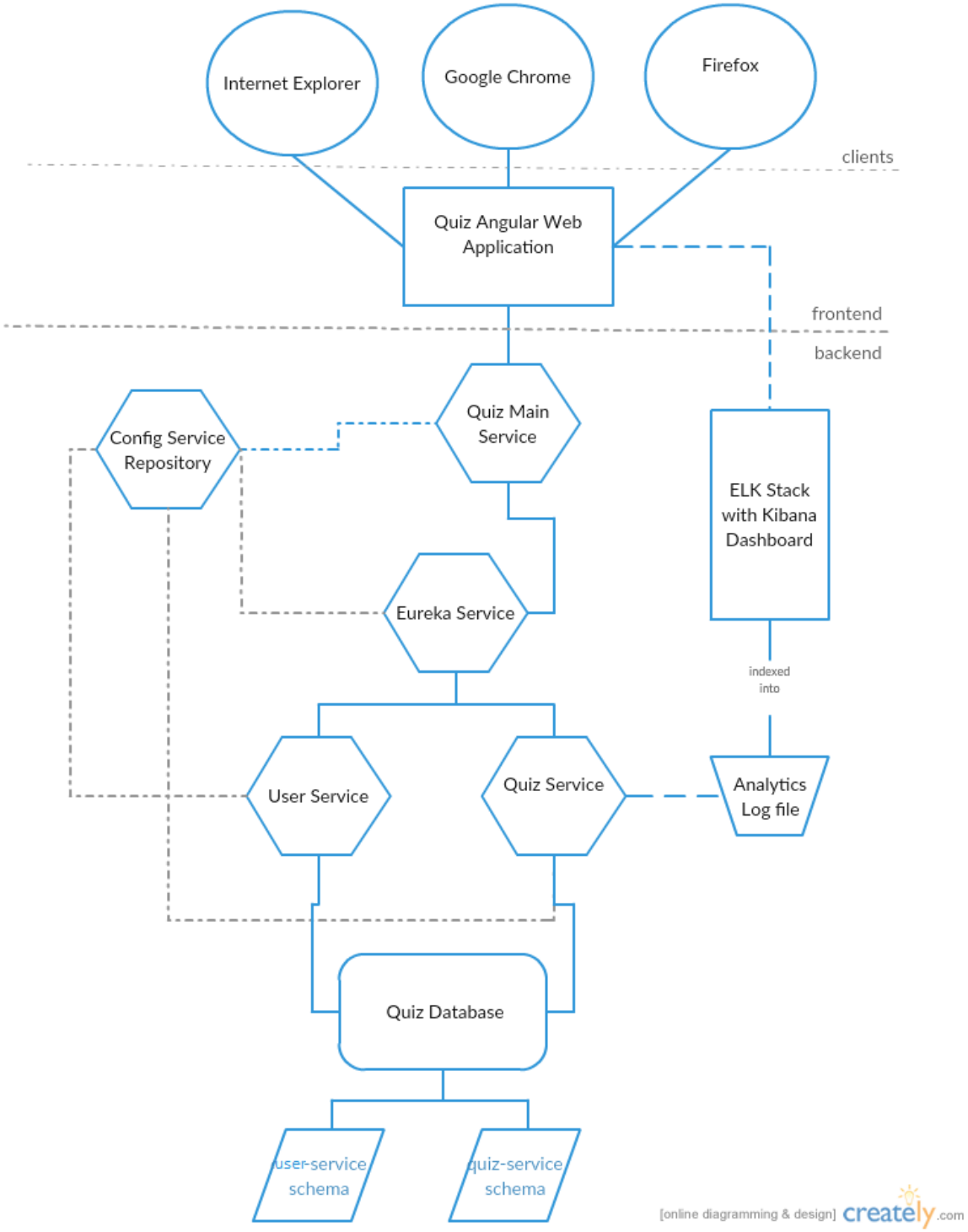

Fig. 5.1 *General architecture*

As you can see from from Fig. 5.1, describing the general structure, the clients, in our case, internet browser, access the Quiz application via the *Angular web service* exposed at

localhost:8808. The web application routes all the backend calls to the *Quiz-Main-Service microservice*, that in this project, acts more as a proxy for the rest of services. As you can see from the picture, the quiz proxy microservice routes the calls to the rest of the services that are found at runtime, by the service discovery provided by *Eureka Service*.
The *User service* handles user related functionalities, from the creation of new users, to login or profile updates. The *Quiz service* handles most of the complex business processes of the application, anything related to quiz management.
The *Config service* is a microservice whose only role is to provide externalized configuration management in this distributed system. Each microservice fetches the assigned configurations at start-up, and can even refresh them if those configuration were modified by calling an endpoint provided by spring boot. One of those configurations for each service is the datasource information , that as you can see from the diagram is a *Postgresql Server Database*, where each microservice has its own specific *schema* to hold the data.
The logs generated by the Quiz Service are indexed into *Elasticsearch* and used for the creation of different diagrams and reports that will be available to the users via the web application.

In the next part I am going to go into more details for each of the core part of application, to better describe the interaction between the services, and the contribution of each of them have for the end result.

5.3 Backend Spring Boot Microservices

The core platform of the software-as-a-service design of the quiz application is the microservice architecture, with the help and support offered by Spring Boot. This framework was was primarily chosen due to being familiar with Spring, and the ease of developing REST-service controllers, business services, and data repositories with it. This could just as easily be done in other combination of frameworks and languages, one of the benefits of microservice architecture is full independence of the services, so it shouldn't matter what language or platform each is built and deployed in.

The application uses in total five microservices that each serve to provide one specific business process. I'm am going to briefly present each one of them.

1. The Config Service provides externalized configuration management in this distributed system. With the Config Server you have a central place to manage external properties for applications across all environments. Each microservice is assigned an unique name and knows the location host and port of the config service, take as an example the user-service's *application.properties*:

spring.application.name=user-service

spring.cloud.config.uri=http://localhost:8888

This tells spring to look for any configuration at the specific URL, that are intended for "user-service". The config service has a list of properties files that belong to each of the microservice clients, as you can see in Fig. 5.2.

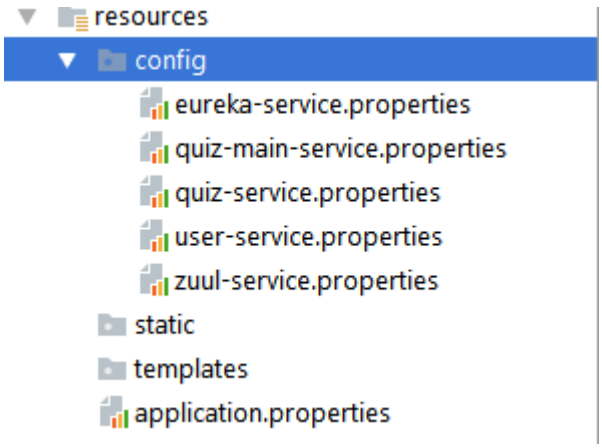

Fig. 5.2 Config Service property files

The pattern for the property file name is [*spring.boot.application.name*].*properties* and here is, as an example, the content of the *user-service.properties* file:

server.port=${PORT:8001}

spring.datasource.url= jdbc:postgresql://localhost/lproject?currentSchema=user-service

spring.datasource.username=postgres

spring.datasource.password=root

spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update

So in the case of the user service, the property list contains information about what server port it should be deployed to and datasource access configurations. Annotating the user service with *@RefreshScope* makes it so the properties can be updated at runtime without the need of restarting the application.

2. Eureka Service acts as the Eureka Server to our application and is is primarily used because it enables client-side load-balancing and decouples service providers from consumers without the need for DNS. It is the middleman through which each microservice can be discovered by the rest, enables communication between them and provides several important tools for maintenance like embedded Hystrix dashboard that allows you to monitor your servers or the Eureka Dashboard that show the status of each service as shown in Fig. 5.3.

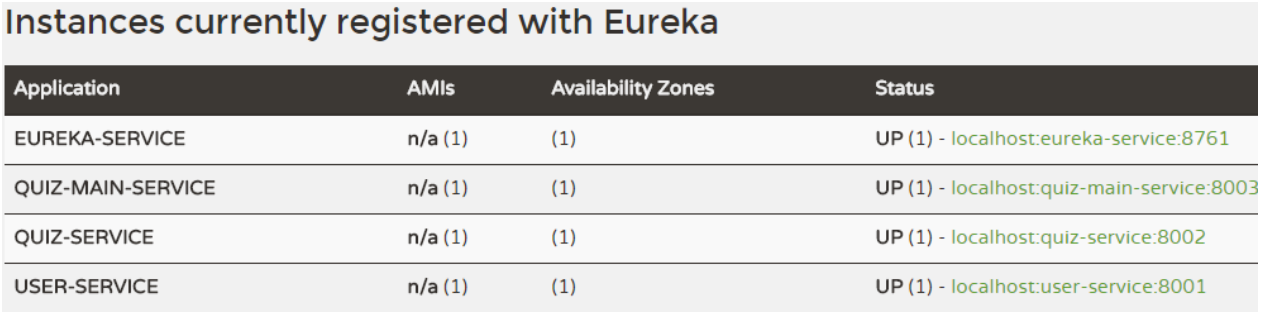

Fig. 5.3 *Eureka dashboard- microservices status*

The concept of Client Side Load Balancing basically means that instead of relying on another service to distribute the load, the client itself, is responsible for deciding where to send the traffic also using an algorithm like round-robin. It can either discover the instances, via service discovery, or can be configured with a predefined list. The one used by Eureka is Called Ribbon and the default algorithm is round-robin which basically means that calls are assigned to each client instances in equal portions and in circular order, handling all instances without priority. Another important role Eureka plays to the application is that is provides an easy and safe way for the microservices to communicate between each other, and for that I am using the Declarative REST Client: Feign that creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations.

```
@FeignClient(value = "user-service")
public interface IUserServiceFeign extends IUserServiceEndpoint { }
```

In the code above , taken from the Quiz-Main-Service microservice, we specify a feign client bean that extends the management interface from the user microservice, that exposes functionalities like login, user CRUD or user profile updates. The implementation itself is actually also inside the user service, so basicly this feign declaration, instantiates a java client to the API exposed by the user microservice, making him a “java to http client binder”.

3. User Service handles the user related business processes. As in the case of the rest of the microservices in this project, it is split in 2 modules:

- *User-service-api* : provides the interface for the user management endpoint, and the POJO classes used as request and response, such that the API does not deal directly with the entities
- *User-service-app* : has a dependency to the api module and represents the business logic. The user information is held in 2 entity classes as it can be seen from Fig. 5.4, along with their dependency relationship.

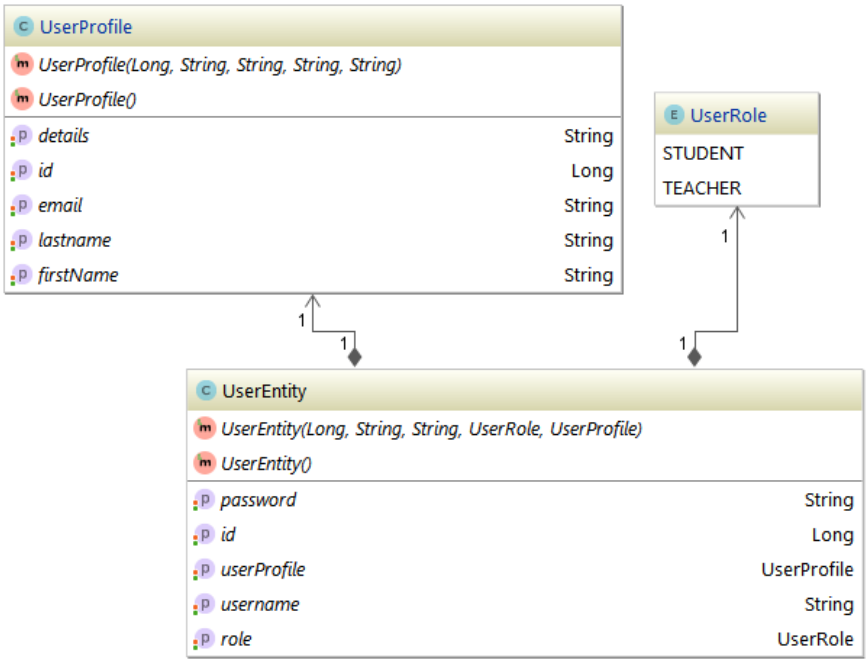

Fig. 5.4 *User service domain entities*

4. Quiz Service is providing the most complex functionalities to this application, the ones that deal with any quiz related workflows.

Starting from the domain, the entity classes are structured under different folders to better characterize their scope: question, answer and quiz. I am using the Java Persistence API(JPA) specification for accessing, persisting, and managing data between Java objects / classes and the relational database. It provides a very intuitive solution to querying with custom made methods for filtering data. In Fig. 5.5 you can see a self-explainable example of custom methods that are declared in the repository which do not require implementation, being auto-translated by JPA into the native language:

```
@Repository
public interface IQuizResponseEntityDao extends JpaRepository<QuizResponseEntity, Long> {
    List<QuizResponseEntity> findByQuiz(QuizEntity quizEntity);

    QuizResponseEntity findByQuizAndUserId(QuizEntity quizResponseEntity, String userId);

    List<QuizResponseEntity> findByUserId(String student_id);

}
```

Fig. 5.5 *JPA repositories with custom query filters methods.*

The service part of the app module contains the implementation of the business logic ranging from calls to the DAO objects, all quiz related functionalities and convertors from and to the entity classes to the POJO request and response classes used in the API definition. These processes are greatly aided by the use of JAVA 8 features like the stream api. In the example from Fig. 5.6 below, you can see how the list of quizzes that have to be graded are returned, given the teacher id, by first finding all his created quizzes and filtering the ones that are not yet corrected.

```
public List<QuizResponseEntity> getQuizesToCorrect(String creatorId) {
    List<QuizResponseEntity> quizes = new ArrayList<>();
    List<QuizEntity> quizEntities = quizEntityDao.findByCreatorId(creatorId);
    quizEntities.stream().forEach(quizEntity ->
            quizResponseEntityDao.findByQuiz(quizEntity).stream()
                    .filter(quizResponseEntity -> quizResponseEntity.isCorrected() == false)
                    .forEach(quizes::add));
    return quizes;
}
```

Fig. 5.6 *Java 8 stream api use example*

The automatic quiz correction is only applied to specific quiz types, more exactly the ones with SINGLE/MULTIPLE options if when the questions were created the correct answers were also supplied. This filtering is made by the quiz type field, inside a dedicated switch statement. In the other case, when the valid answers are not known, the teacher has to manually grade each answer, the business logic going through a different flow.

Another important functionality the quiz service is responsible for is managing the logging process that will later be used by elasticsearch. For that purpose, each time a quiz is being corrected either by the automatic system or the professor, there is a log entry being published to the analytics log file, encapsulated in the class described in Fig. 5.7 :

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class LogQuestionModel {

    private String quizName;
    private String quizType;
    private String questionText;
    private double questionScore;
    private boolean isAnsweredCorrectly;
    private String student;
    private String teacher;
}
```

Fig. 5.7 *Log Question Model class that is published to the analytics.log*

The annotations the class is decorated with come from the lombok plugin, which is used throughout the project to generate all the boilerplate that is normally associated with simple POJOs, basicly auto-generating the constructors, getter, setters, toString, equals, hashCode and others. It is a very handy tool that aids the java developers work making it faster and more efficient.[26]

This microservice uses a dedicated logger configuration such that the object is logged in json format to be much more easier to be parsed by the reporting tools. For this task I have used the *Logback JSON encoder*, which is a highly-configurable, general-purpose, JSON logging mechanism and is as easy to add to our spring project as declaring the dependency inside the pom.xml file.[27] An example of the log entry generated:

{"@timestamp":"2017-06-19T22:14:30.316+03:00","@version":1,"message":"{\"quizName\":\"quiz\",\"quizType\":\"SINGLE_ANSWER\",\"questionText\":\"What is love?\",\"questionScore\":2.0,\"isAnsweredCorrectly\":false,\"student\":\"user1\",\"teacher\":\"prof1\"}","logger_name":"analytics","thread_name":"http-nio-8002-exec-9","level":"DEBUG","level_value":10000}

The logback encoder is configured such that only the desired class actually logs to the analytics file and the rest of the application debugging information is redirected to the standard output.

5. Quiz Main Service is used more as a proxy for the quiz and user microservices. It it dependent to the API of both of them, as it can be seen in Fig. 5.8 below and it deals with instantiating the feign clients so it can forward the calls to the respective service. Its only client is the Web Application that forwards its requests via http and are consumed by the exposed rest endpoints. Another responsibility of this microservice is validating the request data, handling and throwing exceptions.

```
<dependencies>
    <dependency>
        <groupId>com.lproject</groupId>
        <artifactId>quiz-service-api</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>com.lproject</groupId>
        <artifactId>user-service-api</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
```

Fig. 5.8 *Quiz-Main-Service dependencies*

5.4 Angular Web Application

Using the Angular Framework in developing the web application brought a lot with it more advantages that I initially considered, the end result being a responsive SPA(single page application). The starting point of the application, setup and initialization process was very easy by using the CLI provides by node js, with little manual configuration needed. Further along the line, the component based architecture, where each is focused on a specific task or workflow provide much flexibility and reusability. To better put thing in the perspective let's look at an example:

```
@Component({
  selector: "login",
  templateUrl: 'login.component.html'
})
export class LoginComponent implements OnInit {
...
}
```

The *component* annotation declares this class at being a component, the templateUrl property defines the view associated with this component, and the selector property provides the unique id by which this component, along with its view, can be inserted and used anywhere in the project

by simply adding <login></login> to an html file. The component hold the model objects and properties that are being rendered in the view, basicly every page that the client sees while browsing the web application being composed by one or multiple components glued together.

The *routing system* manages navigation to different pages in your application, the application being a SPA (Single Page Application), with no page reloading. Some useful functionalities the routing system provides are the easy implementation of Route Guards, covered in Chapter 4.2.1, that add a layer of security and improve the accessibility. The reusable services, that are injected in a client component, in this application, provide the methods that are used to communicate with the backend API, and an example can be seen below:

```
@Injectable()
export class QuizServiceComponent {
   constructor(private http: Http) { }
  getAllQuizes(userid: string): Observable<QuizRequest[]> {
      return this.http.get('http://localhost:8003/quizes/' + userid, {
         headers: new Headers({
            'Content-Type': 'application/json'
         })
      }).map(res => res.json());
   }
...
}
```

As it can be seen from the above code the json result of the API call is actually mapped to an Observable<*QuizRequest[]*>. Observables open up a continuous channel of communication in which multiple values of data can be emitted over time, asynchronously.[28] The return type being an array of QuizRequest objects, is automatically created from the json response, feature enabled by the use of *Typescript*, which provides familiar OOP constructs, like classes and constructors, to javascript. With the help of typescript I have brought the same model classes

from the quiz service API to the web application such that the HTTP calls request and response format agree, without any need of extra javascript mappings and parsing.

5.5 Kibana Reporting

As we discussed more in detail in chapter 4.3, the application uses the Elastic stack, containing Elasticsearch, Logstash and Kibana, to provide the analysis and reporting tools. The analytics log file generated by the Quiz Service is being indexed in the Elasticsearch with the help of logstash, process explained in chapter 4.3.2, filtered to make the desired information more accessible and used inside the Kibana Dashboard in order to create different diagrams and graphs that better describe a certain application flow result.
In the next part we will take a look on a few examples of generated analysis tools and explain the business process each describe.

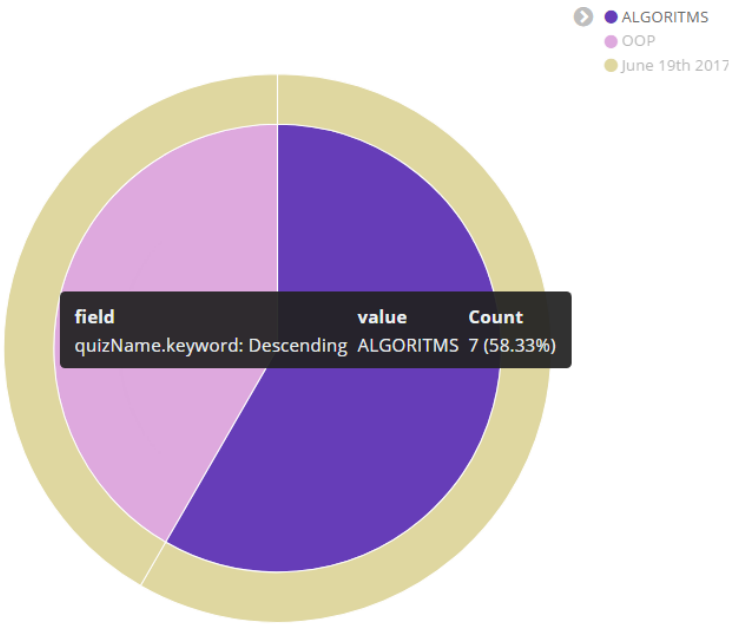

Fig. 5.9 *Pie chart showing the number of quiz responses and the submission date*

On Fig. 5.9, it it shown the number of quiz submissions on the 19th of June 2017, and what quizes were solved, in this case the OOP and Algorithms.

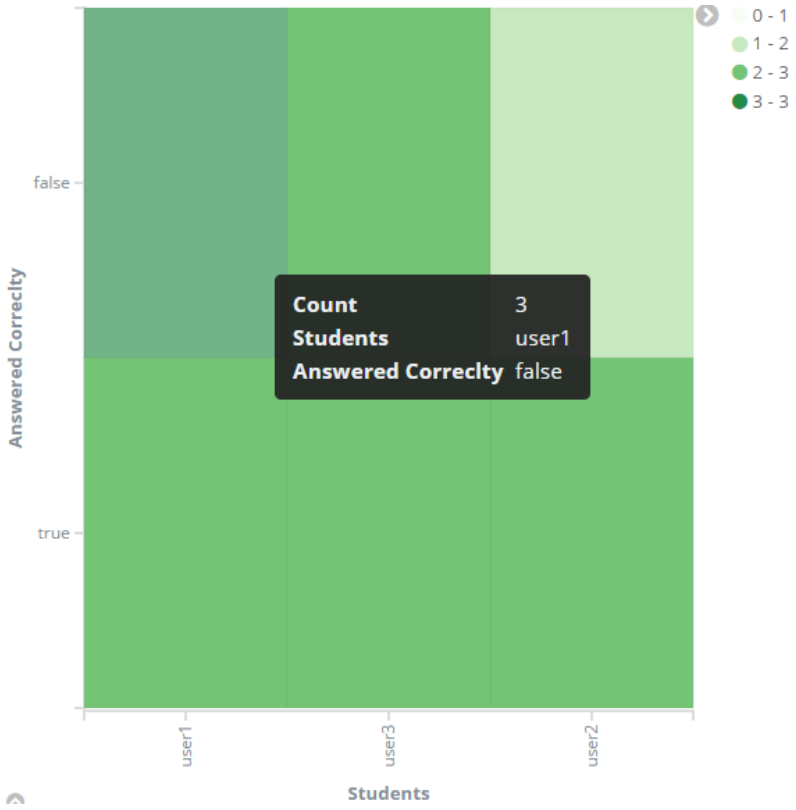

Fig. 5.10 *Heat map showing how many correct/false answers each student has*

In Fig. 5.10 there is a custom Kibana Heat map listing the number of students that submitted a response in a certain time frame, and the proportions of correct and incorrect answers he has given. For example the student with the username user1 has 3 incorrect answers, and the intensity of the color showing that he is the top in this category.

In Fig. 5.11 below, it is presented a chart showing the quiz response submissions grouped by quiz type and professor. An useful information that can be extracted from there is the fact that single answer quiz type have the most student response submissions and is generally the one that is answered correctly with almost 20 valid answers and the teacher with id *prof2* has the best quiz results.

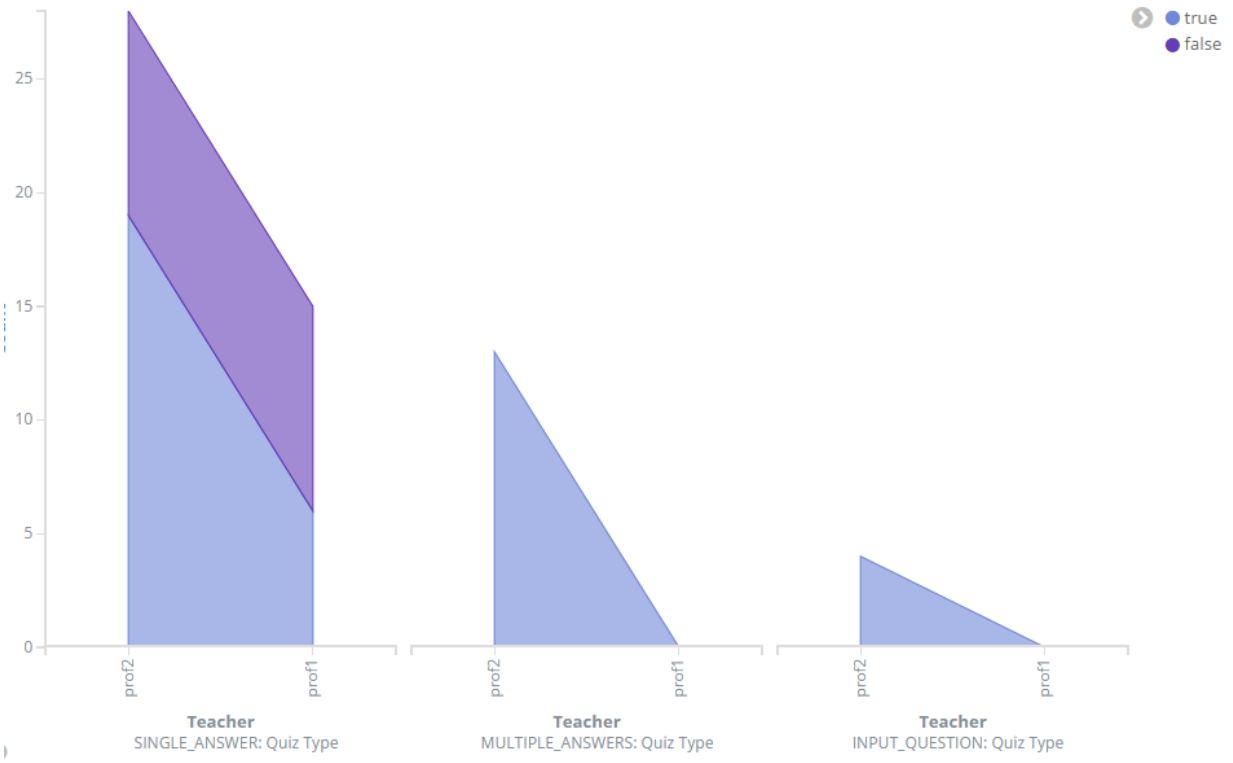

Fig. 5.11 *Quiz types metric grouped by teacher, quiz type and the validity of each student result.*

The graphs presented above were created in the dashboard, saved and can be directly linked inside an iframe to the web application, with the latest data from logstash since log events are being published asynchronously. Furthermore each teacher has access to creating more of them, to better illustrate the business flow result of interest, the Kibana dashboard interface being very accessible and intuitive to use.

I find Elasticsearch a great tool for quickly searching through a large number of text input and for doing some simple visualizations. It's super simple to use, but very powerful. It’s interesting to use to find patterns in your data and retrieve useful, human readable information about certain aspects of an application.

Although not applied to this software solution, Kibana can also be configured to track the application debugging logs. Searching for a particular error across hundreds of log files on multiple servers is difficult without good tools. The elk stack solution to this problem is to set up a centralized logging solution so that multiple logs can be aggregated in a central location, to

effectively consolidate, manage, and analyze these different logs, using Kibana as well. In the future, as more microservices get added to the application, the centralized logging mechanism should be also added to the project, the implementation effort being resumed to more logstash configuration and indexing, since the rest of the needed configuration is already done in order to support the current features.

Chapter 6. Conclusions

With this application I managed to provide a Learning Management System, designed to improve the task of quizzing, from creation by the teacher to completion by the student, and help the people involved in this process by offering a way to better analyze the results.

It is a platform that enables the teacher to creates multiple type of quizzes like Single Options, Multiple Options, Timed, or with classic Essay Questions, to have those quizzes be automatically graded by an automatic system if he provides the correct answers, to grade quizzes, to offer feedback and to consult or even create different types of analysis tools to better characterize the results.

At the same time the student can submit answers to different quizzes, receive feedback and, as in the case of the teacher, access different types of diagrams and etc that summarize his results and receive hints at potential areas he should focus on in the future. For example he can check on which quiz topic has the most poor results, what is the topic the students are more interested in and many more.

The application has a microservice architecture which ensures easy application scalability since it is composed of multiple microservices which share no external dependencies, simplified application updates which aids agile development and continuous delivery and multi-language development since each microservice is an independent ‘product’.

There is a large list of potential improvements and upgrades to the existing functionalities and architecture and in the next part I am going to go through some of them. The first one is adding support for more types of quizzes like Open-Books questions with integrated documentation per question or topic, Fill In The Blank, where you provide a sentence with blanks in it and no hints and even YES/NO type of questions. Another addition should be to improve the auto-correction system to work even if no correct answers are provided when the questions are created, with the

help of machine learning or even artificial intelligence through neural networks. To better structure the topics covered by the quizzes, there should be classes or teams added to the User-Microservice such that a quiz can be targeted to a specific list of students.

Architecture wise improvements on the future should target to better support different types of quizzes. Eventually if automatic more automatic quiz correction is added, then a better, or lets say a more correct solution, would be extracting each quiz type to dedicated microservice and the common classes in an external library. The front-end can be improved by upgrading the user interface to provide more accessibility and better handling of current functionalities, UI design not really being one of my strongest points.

Another potential improvement is porting the application to the mobile platform like Android, iOS or Windows Phone since the current architecture and used technologies facilitate this process and would increase the range of users that could use the platform.

I state that this work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Chapter 7. Bibliography

[1]. The importance of tests & quizzes for students in eLearning, Accessed June 15, 2017 https://www.talentlms.com/elearning/tests-and-quizzes

[2]. Examination methods influence learning, Accessed June 15, 2017 http://agtr.ilri.cgiar.org

[3]. Google Forms, Accessed June 16, 2017 https://www.google.com/forms/about/

[4]. Moodle, Accessed June 16, 2017 https://moodle.org/

[5]. Spring Framework 4.3.9, Accessed June 16, 2017, https://en.wikipedia.org/wiki/Spring_Framework#Spring_Boot

[6]. Mak, Gary (September 1, 2010). Spring Recipes: A Problem-Solution Approach (Second ed.)

[7]. Walls, Craig (November 28, 2010). Spring in Action (Third ed.).

[8]. Spring Boot, Accessed June 16, 2017 http://projects.spring.io/spring-boot/

[9]. Dan Woods, Mar 18, 2014, Exploring Micro-frameworks: Spring Boot, https://www.infoq.com/articles/microframeworks1-spring-boot

[10]. JAX Editorial Teams, Technology trends 2017: Here are the top frameworks, March 2, 2017,, https://jaxenter.com/technology-trends-2017-top-frameworks-131993.html

[11]. Spring Cloud Netflix, Accessed on June 17, 2017, https://cloud.spring.io/spring-cloud-netflix/

[12]. Matt Stine MAY 26, 2015, Build self-healing distributed systems with Spring Cloud, http://www.javaworld.com

[13]. "What is PostgreSQL?". PostgreSQL 9.3.0 Documentation. PostgreSQL Global Development Group. Retrieved 2013-09-20.

[14]. Angular, Accesed June 17, 2017, https://en.wikipedia.org/wiki/Angular_(application_platform)#Version_2.0.0

[15]. Angular, Accessed June 17, 2017, https://angular.io/.

[16]. Eric Elliot, Dec 10, 2016, Top JavaScript Frameworks & Topics to Learn in 2017,https://medium.com/javascript-scene/top-javascript-frameworks-topics-to-learn-in-2017-700a397b711

[17]. Yakov Fain on Apr 26, 2016,, Angular 2 and TypeScript - A High Level Overview, https://www.infoq.com/articles/Angular2-TypeScript-High-Level-Overview

[18]. Arsalan Rashid and Uroosa Sehar, Dec. 08, 2016, Does Angular 2 Surpass React?, https://dzone.com/articles/difference-between-react-and-angular-2-does-angula

[19]. Pascal Precht on Jul 18, 2016, PROTECTING ROUTES USING GUARDS IN ANGULAR, https://blog.thoughtram.io/angular/2016/07/18/guards-in-angular-2.html

[20]. Elasticsearch, Accessed June 18, 2017, https://www.elastic.co/products/elasticsearch

[21]. Samuel Scott, February 17th, 2016, These 15 Tech Companies Chose the ELK Stack Over Proprietary Logging Software, https://logz.io/blog/15-tech-companies-chose-elk-stack/

[22]. Gathering insights from data: An overview of the Elastic stack, 07 Jun 2016 Tyler Langlois, https://opensource.com/life/16/6/overview-elastic-stack

[23]. Postman, Accessed June 19, 2017, https://www.getpostman.com/

[24]. The twelve factor app, Accessed June 19, 2017, https://12factor.net/

[25]. Robert C. Martin, “Principles Of OOD”, (“Uncle BOB”), butunclebob.com, Last verified 2014-07-17. Dates back to at least 2003.

[26]. Project Lombok, Accessed June 19, 2017, https://projectlombok.org/

[27]. Logstash Logback Encoder, Accessed June 20, 2017, https://github.com/logstash/logstash-logback-encoder

[28]. Observables, Accessed June 20, 2017, https://angular-2-training-book.rangle.io/handout/observables/