Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

Open source e-learning platform

Final Report

Authors:
44598 André L. A. Q. de Oliveira
44847 João Eduardo Santos
44823 Rodrigo Mogárrio F. Leal

Final Report for Unidade Curricular de Projeto e Seminário
Licenciatura em Engenharia Informática e de Computadores

Advisors: Cátia Vaz, José Simão

26 – September – 2020

Abstract

In today's competitive job market, programming jobs are amongst the most desirable careers. The ability to innovate, create and troubleshoot all kinds of technologies, on a daily basis, is what drives many individuals to seek experience and pursue a future in computer science or coding.

To accomplish this ambition one can be a self-taught enthusiast, or one can seek the knowledge of professionals through all sort of courses and universities to acquire a considerable high amount of skills sets that will allow to succeed in whichever field of choice they commit to program.

But one thing is for sure, we live in a fast-paced world, and Information Technology is no different. It is constantly changing and evolving, and new trends appear every day, along with new technologies and marketing strategies.

To this is clear that no matter how long one codes, eventually will be faced with the need to keep learning new skills and improve oneself, to prevent becoming outdated, or to be better prepared for a new job interview, or even if just to improve the academic performance.

For this reason, there are avilable some platforms that provide an environment for defining algorithms and testing. However, many of them are not open source, or they do not have such an appealing environment or just do not allow multi-language.

Therefore, this project intends to combine all the strengths mentioned above with none of barriers and create the *IS E-Learning* platform, an e-learning platform to help other programmers achieve their goals.

Resumo

No atual competitivo mercado de trabalho, os empregos relacionados com programação estão entre as carreiras mais desejáveis. A capacidade de inovar, criar e solucionar problemas de todo o tipo de tecnologias diariamente, é o que leva muitos indivíduos a procurar experiência e seguir um futuro na ciência da computação.

Para realizar esta ambição, pode-se ser um entusiasta autodidata, ou pode-se procurar o conhecimento de profissionais em todos os tipos de cursos e universidades para adquirir uma quantidade respeitável de habilidades que permitirão ter sucesso em qualquer área de programação que de escolha.

Mas uma coisa é certa, vivemos em um mundo acelerado e em Tecnologias de Informação não é diferente. Está em constante mudança e evolução, e novas tendências aparecem todos os dias, juntamente com novas tecnologias e estratégias de marketing.

Deste modo fica claro que por mais que se programe, acabará sempre se deparando com a necessidade de continuar a aprender novas habilidades e se aperfeiçoar, para evitar ficar desatualizado, ou para estar mais bem preparado para uma nova entrevista de emprego, ou mesmo apenas para melhorar o desempenho acadêmico.

Por este motivo, existem disponíveis algumas plataformas que fornecem um ambiente para definição de algoritmos e testes. No entanto, muitas delas não são de código aberto, ou não têm um ambiente tão atraente, ou simplesmente não permitem vários idiomas.

Este projeto pretende combinar todos os pontos fortes mencionados acima com nenhuma das barreiras e criar a plataforma *IS E-Learning*, uma plataforma de e-learning para ajudar outros programadores a atingir seus objetivos.

Index

The C# Execution was developed in a C# application targeting .Net Core framework. This application adheres to the same contract as the other application: it receives an HTTP message and based on the contents it is identified if it is necessary to run unit tests or the code. In this application a new solution is created based on a previously created template, with two projects (one for the code and another for the unit tests), the application will then overwrite the contents of a file in each

List of Figures

List of Tables

1. Introduction

In today's competitive job market, programming jobs are amongst the most desirable careers. The ability to innovate, create and troubleshoot all kinds of technologies, on a daily basis, is what drives many individuals to seek experience and pursue a future in computer science or coding.

To accomplish this ambition one can be a self-taught enthusiast, or one can seek the knowledge of professionals through all sort of courses and universities to acquire a considerable high amount of skills sets that will allow to succeed in whichever field of choice they commit to program. But one thing is for sure, we live in a fast-paced world, and Information Technology (IT) is no different. It is constantly changing and evolving, and new trends appear every day, along with new technologies and marketing strategies. To this is clear that no matter how long one codes, eventually will be faced with the need to keep learning new skills and improve oneself, to prevent becoming outdated, or to be better prepared for a new job interview, or even if just to improve the academic performance.

For this reason, there are out there some platforms that provide an environment for defining algorithms and testing [1]. However, many of them are not open source, or they do not have such an appealing environment or just do not allow multi-language. Therefore, this project intends to combine all the strengths mentioned above and create the *IS E-Learning platform*, an e-learning platform to help other programmers achieve their goals.

Being an e-learning platform brings to the table certain inherent aspects, like allowing to be accessed from anywhere provided that exists an internet connection. This specific trait gives a very attractive perk to the client, which is, the code-execution environment. The idea is to deliver an uncomplicated and easy-going experience to the user where it is possible to write solutions, build test cases, run the code and check the output directly on our website without having to configure an environment, or download endless libraries. Another positive aspect of this single attribute is that it has the potential to serve as a powerful tool to ISEL, if in the future it could be implemented in school computers.

Coding out solutions to algorithm problems is the best way to practice and learn, but the truth is, that doing so with just a tool to run code without any structured guidance makes the process of learning more challenging. Understanding the inner workings of complex algorithms is no easy task, and even experience programmers nowadays struggle with coding interviews for the simple fact that they are hard and go beyond algorithms and data structures. Companies want to hire the best of the best, and they value someone who can develop an high class product, which means the programmer must be able to create something that is performant, stable, scalable and bug free, and to be able to deliver such a system or product, one must be proficient and understand algorithms and have mastery in programming languages. For this reason, and because the best way to learn is from examples of someone who understands the subject, the IS E-Learning platform comes with a service that provides *Challenges*, which are programming problems that needs to be solved. And because in our own path

we learn much by reaching out to the coding community, through forums or other people examples, we also want to foment this concept of community, by allowing any registered user to make good use of the his own gathered expertise and create his own challenge and share it on the platform so that others might learn from it.

But despite of how much an individual studies or practices, he/she will only know if he/she has mastered the topic when put to the test. Sometimes is not all about the smartness or skills, but flexibility, stress-resistance, and the ability to iterate approaches fast. To validate this preparation state, IS E-Learning platform has a service where it is possible to create *Questionnaires*. *Questionnaires* are a selected number of pre-existent challenges all grouped up and put together to create a single assessment.

One outstanding thing about programming, is that it is everywhere, and that it can be used in any field area to help solve a problem. But with that, comes that not everyone speaks the same programming language, mainly because languages were created to better suit a specific theme, like web development, machine learning or data analysis. As engineers it is not enough to be only good at one thing, since that will not only limit our work opportunities but as our own problem-solving skills. For that reason, we wanted our platform to support multilanguage, and currently it provides 5 popular ones.

In the end, not only we want to provide an appellative e-learning platform that can be useful in academic environment, professional interviews, or even for just a programming enthusiast who wants to learn more, where every user can solve and create coding challenges, as well as questionnaires to put to test the best of his abilities, but also we want to do it in openly manner, so that it can be freely accessed, used, changed, and shared by everyone.

1.1. Open Source

According to opensource.org [2], and Open Source software (OSS) “is software that can be freely accessed, used, changed, and shared (in modified or unmodified form) by anyone”.

Open source software is often developed in a decentralized and collaborative way, relying on peer review and community production. Open source usually more flexible and has more longevity than its proprietary peers because it is developed by communities rather than a single author or company.

Open source has become a movement and a way of working that reaches beyond software production. The open source movement uses the values and decentralized production model of open source software to find new ways to solve problems in their communities and industries.

1.2. Methodology

As stated, we would like that the *IS E-Learning* platform to be an open source project and that its development to be done using the best practices [3].

To that goal, alongside with the platform itself, vast and detailed documentation about the API was also created, so that in the future, anyone that would like to give its contribution could do it so in an orderly and logic manner, and with this in mind, we developed our project using some of the appropriate methodologies for open source projects.

The project is located in public git repository [4], allowing the development to be in branch-based workflow. In our case everyone has admin permissions levels, as a way to allow full access to the project, but in the future, this levels should be managed by administrators or moderators, so that not everyone could push new code un-review to the project.

Every task planed was monitored resorting to Azure Boards service [5] using a Kanban board to keep track of development tasks and its status work items are updated accordingly, allowing everyone who works on the project to have an accurate and transparent overview of the project's state.

New features are implemented in a feature branch, in order to isolate development without affecting other branches in the repository, and can be merged to the master branch via pull request for other team members to review the changes before merging them into the project, this methodology follows the Trunk based development principals [6]. During the project, merges only occurred when the majority of the group elements were on agreement about the code that was presented.

This peer review methodology is already established as one of the most useful practices in software engineering since a very long time, as stated by Watts Humphrey in Managing the Software Process [7].

As for versioning, a tag-based system was used, following a specific semantic [8]. When a new version was ready it was create a tag with the format `v<MAJOR>.<MINOR>.<PATCH>`, for the beta version a `-beta` was added in front of the version.

1.3. Solution

To achieve the goal of meeting the requirements described previously the solution will take shape as a full-fledged application. An UI for the final user to interact with, which will communicate with a server application exposing a rich HTTP API supporting the operations necessary for persisting data, handling business logic and executing code defined by the user written in one the supported languages.

Adding to this development, to showcase the application and take full advantage of the scalable architecture of the solution, the project will be deployed to the cloud using several GCP services.

1.4. Outline

This project is divided into 8 chapters.

Chapter 2 describes both functional requirements, where the technical details are explained in order to illustrate what our platform is supposed to accomplish, and the non-functional ones, which are mainly focused on specific design and implementation concerns of the solution, so it can meet the requirements, with great performance and a solid security.

Chapter 3 briefly describes the current state of art regarding similar platforms, and a comparison them and our own solution is performed.

Chapter 4 introduces some of the technologies that are used for the development of the solution.

Chapter 5 addresses focus on the architecture, implementation details regarding introducing each component that composes *IS E-Learning* platform, their functionalities, and their interactions.

Chapter 6 addresses on how the modules discussed in chapter 5 were designed, explaining their functionalities and details of their implementation.

Chapter 7 gives an overview about the cloud deployment, witch architecture and its services were used, and how we approached and implemented our solution.

And finally, in Chapter 8, a briefly discussion about the state of the project, what has been completed, some considerations on the work developed, and what is expected to be the future of IS E-Learning platform.

2. Requirements

For this project a series of functional and non-functional requirements [9] were identified and are listed below.

2.1. Functional Requirements

2.1.1 Execute Code

Users will have a UI element where they run code in a multitude of languages:

- Users don´t have to be authenticated in to use this functionality;
- Users can choose a language to write code;
- Users can run then written code and verify the output;
- Users can run then written code and verify the output;

2.1.2 Challenges

Challenges – *Challenges* are a programming problem that needs to be solved. Every *Challenge* has a built-in solution that will be compared with the user submitted answer to determinate its "correctness" through unit tests.

- *Challenges* can be solved on one or more programming language;
- To respond to a *Challenge* a user doesn't need to be logged in;
- Only authenticated users can create *Challenges*;
- To create a *Challenge* a solution and unit tests must be provided;
- To create a *Challenge* the code must compile and the tests must pass;
- *Challenges* can be associated with tags, which can be used to search specific topics;
- Only authenticated users can consult the *Challenges* he/she submitted;
- A user can create a private *Challenge* that is unreachable as a single *Challenge* and can only be visible in a *Questionnaire* created by the *Challenge's* creator;
- Only authenticated users can track and consult previously answered *Challenges*;
- A *Challenge's* solution can only be edited by the *Challenge's* creator;
- A public *Challenge's* solution can be always consulted;
- A private *Challenge's* solution can only be seen by its creator;
- *Challenge's* answer can only be consulted by the user that submit it;

2.1.3 Questionnaire

Questionnaire is a group of *Challenges*.

- Only authenticated in users can create *Questionnaires*;
- Can have public and private *Challenges*;
- Only the creator can edit the *Questionnaire*;
- *Questionnaires* can be shared through a link created by the platform.
- *Questionnaire* can be associated with tags, which can be used for searching
- *Questionnaire* can have a timer associate with it;
- *Questionnaire's* timer starts when link is accessed;
- *Questionnaire's* creator can define what programming language can be used in any *Challenge*;
- *Questionnaire's* creator can decide whether the user responding can view the final evaluation or not;
- Submitted answers for a *Questionnaire* challenges can only be viewed by the *Questionnaire's* creator;
- Submitted answers cannot be modified or deleted;
- Non submitted answers are considered as wrong;

2.1.4 Authentication

Certain features can only be used if a user is authenticated.

- Users can create an account;
- Authentication uses a basic username/password scheme;
- When creating an account, user must provide username, password, name and email

2.1.5 Multi-Language

Platform must provide an environment to run code for multiple programming languages:

- Java
- Kotlin
- JavaScript
- C#
- Python

2.2. Non-Functional Requirements

- Scalability: This platform may be accessed by hundreds or even thousands of users. As such the platform must be able to maintain a high level of performance.

- Security: Executing third party code in a machine raises security concerns.
    - A self-contained execution environment limits the impact of malicious code to the container which executes it, protecting the remaining infrastructure.
- Solution Maintenance
    - Maintaining a complex solution requires a balance between many moving parts. As such, this project's architecture reflects the principles of loose coupling and modularity which facilitate the solution maintenance.
- Efficiency:
    - Hosting the solution in a cloud-based environment improves efficiency of the solution.

2.3. User Journeys

With the previous enumerated requirements, it was possible to define a series of user journeys that will help to identify and implement functionalities.

Figure 1 represents the user journey for registration that reflects the account creation and information needed for Authentication's requirements.

Figure 2 represents the user journey for solving a *Challenge* that reflects the basic authentication scheme and the multiple language, everyone can solve a challenge, read only solution and public solution *Challenge's* requirements.

Figure 3 represents the ability to create *Challenges* and reflects the user's permissions, code compilation, mandatory fields, tags and challenge tracking *Challenge's* requirements.

Figure 4 represents the ability to create a *Questionnaire* and reflects the user's permissions, *Challenge's* privacy, shareable link creation, timer and final evaluation visibility requirements present in the Questionnaire section of Chapter 2.

Figure 5 represents the ability to run a piece of code in the platform and reflects all requirements present in the Run Code section of Chapter 2.

Figure 6 represents the ability to answer a *Questionnaire* in the platform and reflects the shareable link, timer, final evaluation visibility and submit answers visibility requirements present in the Questionnaire section of Chapter 2. With these user journey we get a good coverage of the requirements and base usability of the platform.

2.3.1. Run code

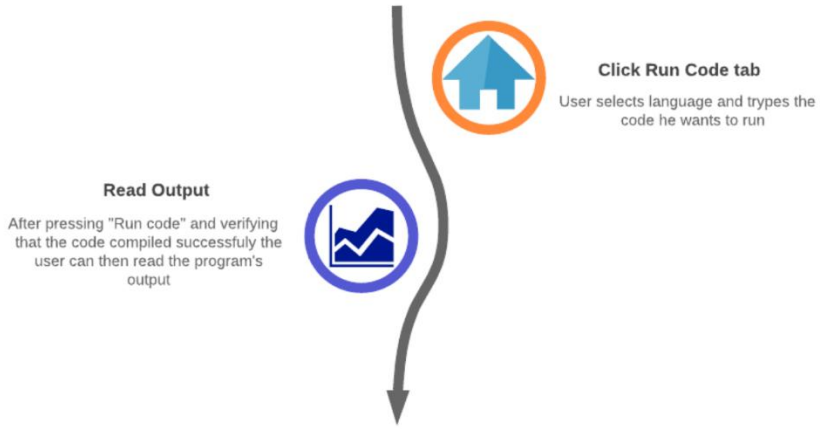

Figure 1 - User Journey for running a piece of code

2.3.2. Registration

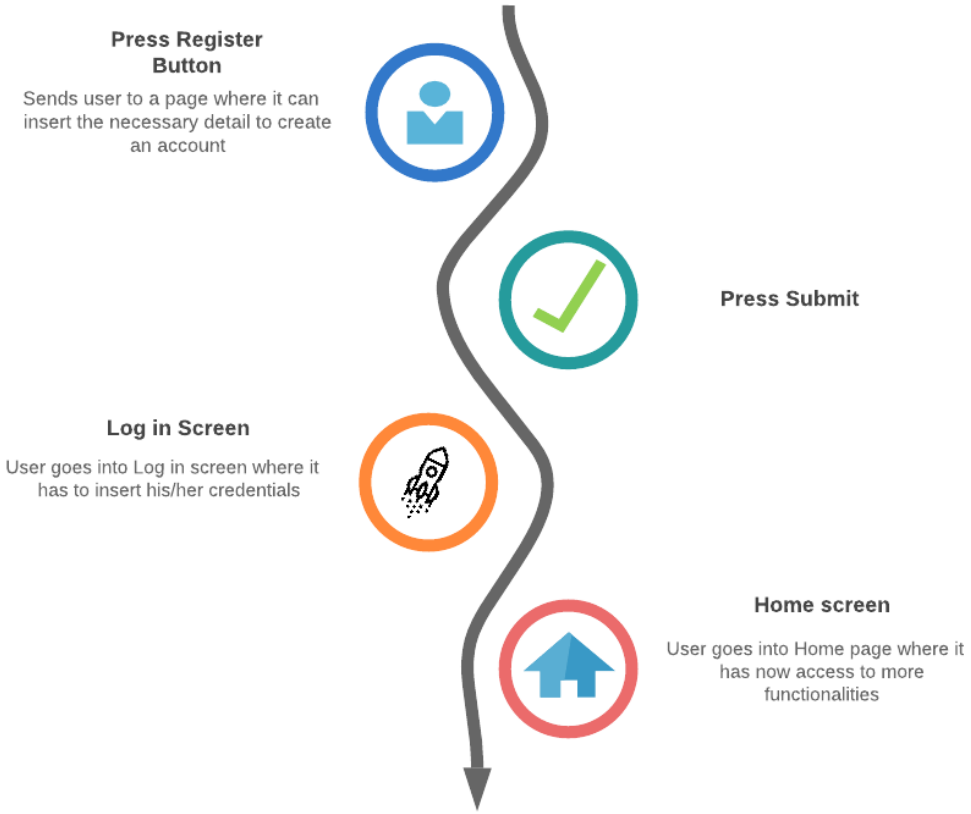

Figure 2 - User Journey for user's registration

2.3.3. Solving a Challenge

Figure 3 - User Journey for solving a Challenge

2.3.4. Create a Challenge

Figure 4 - User Journey for creating a Challenge

2.3.5. Create a Questionnaire

Figure 5 - User Journey for creating a Questionnaire

2.3.6. Answer a Questionnaire

Figure 6 - User Journey for answering a Questionnaire

3. Related work

As emphasized in the introduction, there are plenty of other e-learning platforms available, each one of them serving their own purpose and having unique characteristics. In this chapter we aim to briefly showcase some of them, to demonstrate what are the most common features between them and our own platform, and their differences weighing their pros and cons.

1.1. AlgoExpert

AlgoExpert [10] was made to serve as a resource to prepare for coding interviews, by providing everything someone needs in one streamlined platform. It has 90 hand-picked questions, where only 4 of them are free, but it this possible to get the full platform content for the price of 115€ per year. Despite only having 7 programming languages, they differ from other e-learning platforms by providing over 60 hours of video content. Each question is accompanied by a two-part video, explaining a conceptual overview of the algorithm in how to approach, implement, optimize and how to analyze its space-time complexity, followed by code walkthrough in order to maximize learning. They also have coding interview tips videos to help coders stand out from other software engineers and publicize full projects contests for their clients to promote their programming skills.

1.2. HackerRank

HackerRank [11] is one of the most famous websites platforms for aspiring developers and its highly rich feature wise. On our comparison list it is also the most expensive one, where the individual package costs 230€ per month, but then again it offers beyond the basic coding challenges. It has a clean design, and it is possible to learn on over 25 languages by resolving coding problems, where each problem has a unique leaderboard as well as a solution that provides an explanation of how to approach. It also gives the possibility for a user to create his own problems and share them with friends and participate in challenge competitions. Additionally it also provides the ability for users to submit applications and apply to jobs by solving company-sponsored coding challenges, offering a win-win service not only for developers that want to practice their coding skills and prepare for interviews, but also for companies through their interview platform that helps identify and hire developers with the right skills.

1.3. LeetCode

LeetCode [12] has a very intuitive and appealing interface that makes the navigation on their website very satisfying. They have over 1500 problems, categorized by tags and difficulty, and available in 14 programming languages. Most of the problems are free for the common user, but there is premium content to subscribers that pay 36€ per month or 147€ per year, which includes more questions

commonly asked in famous companies like Google or Amazon, solutions and premium solutions to the problems, and other features like possibility to write with autocomplete or debug the code. They also have an online judge for the problems as well as a service that mocks interviews, where a session is launched for a certain amount of time where the users have to submitted the correct answer for each question before the time expires or they end the session manually. Not only does LeetCode prepare candidates for technical interviews, but also help companies identify talent through sponsoring contests.

1.4. Codewars

Different from all other platforms, Codewars [13] makes learning programming a lot of fun. Offering a huge repository of over 8600 problems in more than 56 programming languages, and ranking system as well as the ability to form coding clans, this platform has a strong active community. A user with a certain amount of ranking points, obtainable by solving problems, may help the platform grow by creating his own he unique problem. This problem may enter the Codewars repository collection if it receives a high positive feedback, which is also given by the community, and may later be translated to other languages, also with the efforts of the community. Each problem has its own feedback comment session where users may discuss about their implementations, and it is possible to always see others users solutions as long as one has already completed the challenge or if it "give ups" and loses ranking points. Although it is not an e-learning platform *per se*, it accomplishes the same effect by making people addicted to coding by making it a stimulant friendly competition with an excellent user interface experience. Codewars also works with tech companies to find good problems solvers and has an optional subscription for 4.5€, that offers not so substantial features such as profile badges, ad-free experience or member-only cluster environments to get faster results.

1.5. CodeChef

CodeChef [14] was born as non-profit educational initiative with the aim to providing a platform for students and young software professionals to practice and hone their skills through online contests. Even having over 4000 problems to practice in more than 55 languages, and a big community, the platform itself is simple and does not offer many features. The reason being that Codechef exists more like an initiative. It excels at promoting coding events in schools, hosting various contests and competitions with not only cash wining prizes but also teach gear, organizing workshops and doubt sessions. There is also the "CodeChef For Schools" program that aims to reach out to young students and encourage them a culture of programming in Indian schools.

On Table 1 it is possible to look at an overview of the most conventional features on each platform.

Table 1 - Feature comparison of select platforms

Platform	***Problems***	***Unit tests***	***Languages***	***Community Spirit***	***Design***	***Price***	***Driven***	***Open Source***
AlgoExpert	90	Yes	7	Good	Clean	115€/yr	Job	No
HackerRank	> 100	Yes	35+	Good	Clean	230€/mo	Job	No
Leetcode	> 1500	Yes	14	Good	Good	147€/yr	Job	No
Codewars	> 8800	Yes	55+	Very Good	Very Good	Free	Entertainment	No
CodeChef	> 4300	No	55+	Good	Poor	Free	Educative	No
IS E-Learning	N/A	Yes	5	N/A	Clean	Free	Educative	Yes

4. Related Technologies

For the development of this application specific technologies were selected. Due to the nature of this projects the number of used technologies is vast, and this chapter describes in more detail a subset of the most relevant technologies.

4.1. React

React is a JavaScript library for building user interfaces. Create by Facebook, it is currently a widely used library used for front end development [15] [16].

One of the big advantages of using react is being able to build components which can be independent from each other and can be reused across all application's components. This dramatically improves modularity, provides loose coupling between components and facilitates maintenance of the solution.

The initial configuration of the project is done with the help of a `npm` package, `create-react-app`. This package creates the barebones of the client-side code including the first component to be rendered. That component can be edited, and other components can be built using the JSX language. JSX is a syntax extension to JavaScript, it looks like HTML but has the full power of JavaScript [17].

React Router is a library which enables route handling using dynamic routing. This allows developers to build a single-page web application with navigation without the page refreshing as the user navigates.

4.2. Spring

Spring is one of the most popular application development frameworks. This lightweight and open source framework enables high performance, easily testable and reusable code [18].

Spring offers several core functionalities like inversion of control (specifically dependency injection), aspect-oriented programming, database access, transaction management, web service development through Spring MVC, amongst many others [18] [19].

Besides the core functionalities, Spring has several projects which allow to extend these functionalities for specific needs. Two projects worthy of mention are Spring Boot and Spring Security which are used on this project.

4.2.1. Spring Boot

Spring boot makes it easier to develop Spring applications. Includes embedded Tomcat, Jetty or Undertown as web application servers allowing the development of standalone applications, automatically configures Spring and 3rd party libraries when possible, offers a set of dependencies to

help build the application (starter dependencies), requires no XML configurations and no code generation [20].

Adding to this, Spring Boot is a widely used project which has a very active community.

4.3. Swagger

Swagger enables developers to describe their API's structure in such a way that it is possible to build both beautiful and interactive API documentation [21]. Swagger UI enables automatic generation of a rich user interface with the API documentation, this UI is generated from documentation compliant with the Open API standard.

4.4. Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers images and execution runtime [22] [23] [24].

Containers are a standardized unit of software that allows developers to isolate their app from its environment, solving the "it works on my machine". Includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Any docker client will be able to run the container image in any machine. For developers, it means that they can focus on writing code without worrying about the system that it will ultimately be running on.

Another advantage of using containers is that they are lightweight, require fewer resources than virtual machines have very quick start up times, and are secure providing isolation from other containers.

Docker containers are built from Docker images, in order to run an application inside a container an image with the application needs to be built, build a container that executes that image. A Docker image is an immutable file which contains the source code, libraries, dependencies, tools, and other files needed for an application to run.

There are several images available for use in docker in image registries like Docker Hub [25]. For most cases, custom docker images need to be built and these can be built recurring to a configuration file known as Docker file.

A Dockerfile is a text file which includes the instructions to build a Docker image. A Dockerfile specifies the operating system, the runtimes, environmental variables, file locations, network ports, other components it needs and what the container will be doing once we run it. With a Dockerfile a Docker client can build an image, build a container from that image and execute it.

5. Architecture

To develop the IS E-Learning platform three main modules were identified: UI, Services and Execution Environments.

The UI module is the presentation layer, with which the final user will interact. This interface will be developed as a single page application.

The Services module will provide a REST API [26] which is the core of the platform. This REST API can be used standalone or with the UI module and will be used to support the UI module.

The Execution Environment module will be responsible for executing code provided by an external source. This module will support several runtime environments, where each application will be developed and hosted on a separate container.

Figure 7 presents how these modules interact, the frontend module only communicates with the services module which in turn communicates with the execution environments, increasing the solution's modularity.

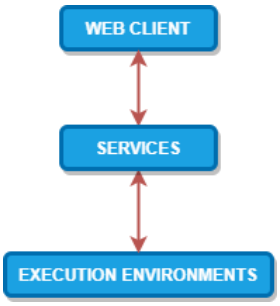

Figure 7 – Architectural Layered Module view, with inter module interactions

5.1. Web Client

The web client end will be a Single Page Application (SPA) enabling a user to interact with the application through an UI. This module will be implemented with React and React Router.

For this use case, the web application was built using NodeJS. Adding to this some external libraries/frameworks worthy of noting were used to support the UI development: Material UI; CodeMirror; Formik; Yup. These are explained in more detail below.

To take advantage of React and follow good development practices, on this project there is a concern to implement components with a modular design. With this approach, the components can be reused in different pages making the application more modular and loosely coupled.

5.1.1 Material-UI

Material-UI [27] is one of the most famous React UI frameworks, and uses grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows. It provides React components that implement Google Material Design [28], that are inspired by the physical world objects and their textures on how they reflect light and cast shadows.

It provides comprehensive, modern UI components that work across the web, mobile and desktop, that are fast and consistent and that were fully tested across modern browsers. Material-UI components also work without any additional setup, working in isolation and they are self-supporting, injecting only the styles they need to display, i.e. the global scope is not affected.

All web client front-end was built using Material-UI components.

5.1.2 CodeMirror

CodeMirror [29] is a versatile text editor implemented in JavaScript. It is specialized for editing code, and comes with a number of languages modes and addons that implement custom UI themes or more advanced editing functionality such as auto close brackets or auto matching tags that will cause the tags around the cursor to be highlighted.

Some of these features seem only “nice to haves”, but they fulfill an important role in the concept of our application, which is be educational driven and to make the learning process easier.

In the context of the developed e-learning platform, the CodeMirror library is used in all the built-in components that use a text editor, such as the *Challenges* interface where one can write the code to be executed.

5.1.2 Formik

Formik is a lightweight, easy to use form helper library which is concerned in helping with 3 particular points: getting values in and out of form state, validation and error messages; handling form submission [30]. In React forms are usually verbose with a lot of boilerplate, with Formik this issue is mitigated making the code more readable and easier to maintain or change.

Another advantage is the integration this library has with Material UI, which is used on this project. This requires the use of another library “formik-material-ui” and allows the use and configuration of some Material UI components when building a form [31].

5.1.2 Yup

The Yup library is a lightweight, widely used, client-side JavaScript schema builder for value parsing and validation [32]. This allows validation of any type of object in JavaScript, with plenty out of the box

validations for number types and string types, and also allows the creation of custom validators making this library very flexible.

On the context of this project it is used for validation when building Formik scripts making the validations even less verbose [33].

5.2. Services

This module is an application which exposes a REST API, enabling its clients to interact with the domains identified during the requirement section.

In Figure 8 it is shown in more detail how the service modules are structured. There are 5 main sub modules: Users; Authentication; Execute code; Challenges; Questionnaires.

This module is developed as a Spring Boot application using the Kotlin language and the Gradle framework as a build and dependency management tool. The database was administered using PostgresSQL and the API is documented with the Open API 3.0 standard [34] hosted on Swagger UI [35].

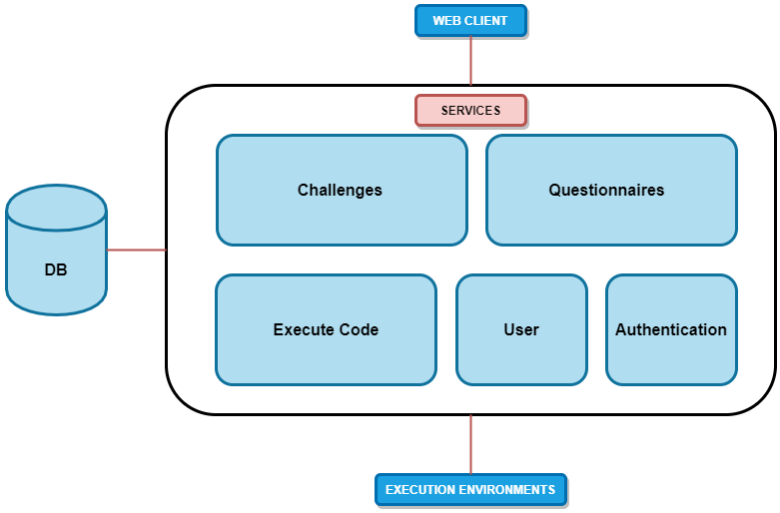

Figure 8 – Detailed view of Services Module including DB

The *Users* submodule is responsible for interaction and business logic with *User* domain.

The *Authentication* module is responsible for allowing user basic authentication and managing endpoint authentication for the whole application.

The *Challenges* submodule is responsible for interaction and business logic with *Challenges* and *Challenge Answers* domains.

The *Questionnaires* submodule is responsible for interaction and business logic with *Questionnaire* and *Questionnaire Answers* domains.

The *Execute Code* submodule is responsible for handling code execution requests redirecting the requests to the correct execution environment depending on the language.

Another detailed shown on the picture above is the Database. The services module is the only module with access to the database and it is responsible for directly connecting this database which maintains the state application for the different domains.

To store the platform data we are relying in PostgreSQL, which is an open source object-relational database, well known for its strong reliability, feature robustness and performance.

5.2.1 Data Model

The data model reflects the necessary structure to comply with the functional requirements and other support structures necessary to the application. Since the database is relational, the design of the data model was done using an Entity Relationship Diagram, which can be seen on Annex B

To follow good practices of data model design this data model follows 3NF rules for normalization [36] [37], below is a more detailed explanation of what represents each entity.

- *App user* - This entity represents a user on the platform.
- *Code Language* - This entity contains the different coding languages supported by the platform
- *Challenge* - This entity maps directly to the *Challenges* identified on Section 2. The *Challenge* has a many to one relationship with the user table. This relationship represents the creator of the *Challenge* which can create multiple *Challenges*. This is a weak entity of user, i.e., does not exist if there is no *user*.
- *Challenge Solution* - This entity represents the solution to the challenge. It is on a separate table to allow defining multiple solutions per *Challenge* (many to one relationship), one for each language. This is a weak entity of *Challenge*. This table has a one to many relationship with the code language table because each answer is written for a specific supported language.
- *Challenge Answer* - This entity represents an answer of a *Challenge*. There is a many to one relationship with the *Challenges* since a *Challenge* can be answered by multiple different people, as a result this table also has a many to one relationship with the user table. This is a weak entity of *Challenge*.
- *Questionnaire* - This entity on the data model maps directly to the *Questionnaires* identified on Chapter 2. The questionnaire has a many to one relationship with the user table, this

relationship represents the creator of the *Questionnaire* which can create multiple *Questionnaires*. This is a weak entity of user.

- *Questionnaire Answer* - This entity represents an answer of a *Questionnaire*. There is a many to one relationship with the questionnaires since a *Questionnaire* can be answered by multiple different people. This is a weak entity of *Questionnaire*.
- *Answer* - This entity represents the abstract concept of an answer. The type of the answer is determiner through the mandatory mutually exclusive relationships between *Answer* and it's "children", *Challenge Answer* and *Questionnaire Answer*. This was done to normalize answer related data since both *Challenges* and *Questionnaire* answers share data but have specificity to their domain. This was enforced on a database level through the usage of triggers. This table has a one to many relationship with the code language table because each answer is written for a specific supported language.
- *Tag* - This entity supports the tag search functionality identified on Chapter 2. There is a one-to-many relationship with the challenges since a challenge can have multiple tags.
- One special connection is also worthy of note, the many to many connection between the tables *Challenge*, *Questionnaire* and *Questionnaire Answer*. This relationship exists in order to support a questionnaire associating to many *Challenges* each with a language (it could only be solved for a specific language even if it is available with more) and also associating the questionnaire answer to the *Challenge* connected with *Questionnaire*.

5.3. Execution Environments

This module contains multiple containers, which are grouped in container execution managers and are grouped by type of language runtime, i.e. there will be a container execution manager for each type of runtime environment. The scalability of this module is provided with the execution manager, providing more containers to the manager will increase capacity. This is particularly relevant when considering that each language can have different demands, i.e. on a given hour or day there might be more executions requested for Python than C# and the current architecture supports this segmented scalability per language, i.e. execution environment.

The goal of this module is to make it possible to execute external code send by this module's clients while supporting multiple runtime environments.

As can be seen on Figure 9, this was achieved by having multiple applications running in separate containers, each container supporting a single runtime. Each application is listening to HTTP requests to execute the code, and when it receives a request it compiles the code (if necessary), executes the code and returns the result of the execution.

These applications all share the same API contract, this means the clients only needs to respect the contract and send the request to the correct application (endpoint) depending on the runtime of the code to be executed. This allows the clients to be abstracted from any implementation, increasing to

modularity of the solution. If the need arises to change a specific runtime environment or even add more it would be a seamless change.

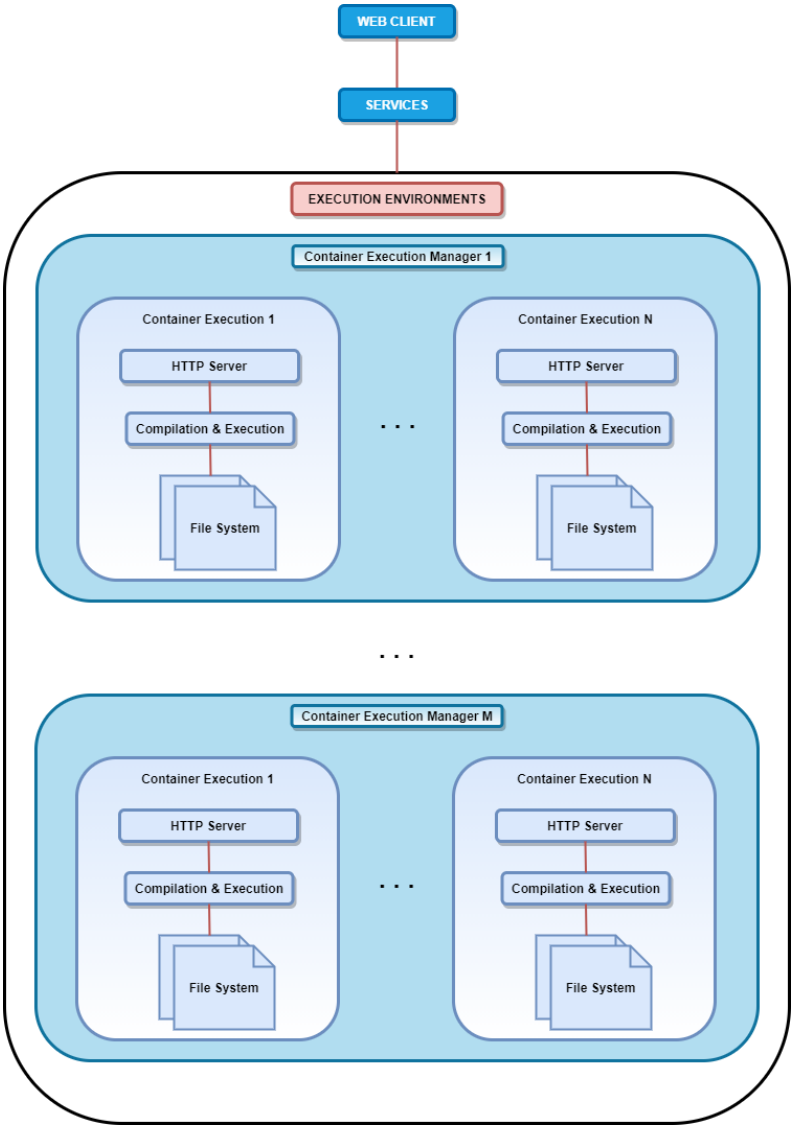

Figure 9 – Detailed view of *Execution Environments* Module

Docker will be used to build and run containers for each execution environment application and 5 runtime environments will be supported: Java, Kotlin, JavaScript, C# and Python.

6. Implementation Details

This chapter describes some relevant implementation details regarding the modules described in chapter 5

6.1. Authentication

For this application only basic authentication is supported. Because authentication is not a major focus of the development, this authentication type meets the business and security requirements while having a simpler and faster implementation.

For authentication purposes it was developed the class `AuthenticationFilter`. This class is responsible to verify the credentials present in the request headers in the form of `Authorization: Basic <credentials>` by consulting the database. In the method `doFilterInternal` which is called if the URL of the request matches what was registered in `authenticationFilterRegistration` method in the `ApplicationConfiguration` class.

The method `doFilterInternal` is also responsible to add an attribute to the request that represents the logged user. This, in conjunction with an implementation of the class `HandlerMethodArgumentResolver` (`UserArgumentResolver` in this case), will enable Spring to pass an instance of User class to the controllers.

The implementation of the `AuthenticationFilter` also provides an additional method, `shouldNotFilter`, which determines if the method `doFilterInternal` should actually be called or not, i.e. if the request will be authenticated or not. In the application described in this document this method was used to implement some extra business logic to the authentication process, for example given an URL that was initially registered as needing authentication verify if the HTTP request method can be called without authentication, i.e. optional authentication, if the answer is positive than the method `doFilterInternal` will not be called.

6.2. Services

Services Users, Execute Code, Challenges and Validations have implementation details which are specific of each service, but also share other cross-service implementations, like input validation and error handling, which are described in the following sections.

6.2.1 Users

User service is responsible to interact with user domain models and enforce business logic. Database records are accessed through an implementation of Spring's interface `CrudRepository`, that

provides several methods to access this information. More details for each method and data structures can be found on this project's Swagger documentation [35].

6.2.2 Execute code

The scope of the Execute submodule purpose is to receive requests to execute code remotely and if the request language is supported redirect the request to the execution environment module where the execution of the code will take place. The contract exposed by the controller of this submodule can be checked on the Swagger documentation [35].

This submodule needs a property file to work properly named `executionEnvironments.properties`. This properties file has information about the endpoint to which the redirection of the execution requests is sent, more details of each property can be found on the wiki of this project's repository [38].

6.2.3 Challenges

The *Challenges* submodule is responsible for interaction and business logic with *Challenges* and *Challenge Answers* domains.

The scope of the *Challenges* submodule includes the domains of *Challenges*, *Challenge Answers*, *Challenge Tags* and *Tags*.

Implementation wise each of these domains exposes a different Spring RestController and has a different service to handle the business logic. The possible operations exposed by these controllers can be found in the Swagger documentation [35].

The *Challenge* domain entity is represented by a combination of the data model entities *Challenge* and Challenge Solution, i.e. a challenge on the service module contains information not only about the challenge itself but also about its solutions.

The *Challenge Answers* domain entity is represented by a combination of the data model entities *Challenge Answer* and *Answer*. This is only natural since the Answer data model entity exists as part of a mandatory mutually exclusive relationship with different types of answers, i.e. every type of answer is related to an *Answer* data model entity.

The *Challenge Tags* domain entity is represented by a combination of the data model entity *Tag* and a record from the many to many entity *CT*, which represents the associations between a Tag and a challenge data model entities.

The *Tags* domain entity is represented by a data model entity *Tag*.

Figure 10 - Workflow to answer a *Challenge*demonstrates the a workflow that an user has to follow to answer a challenge.

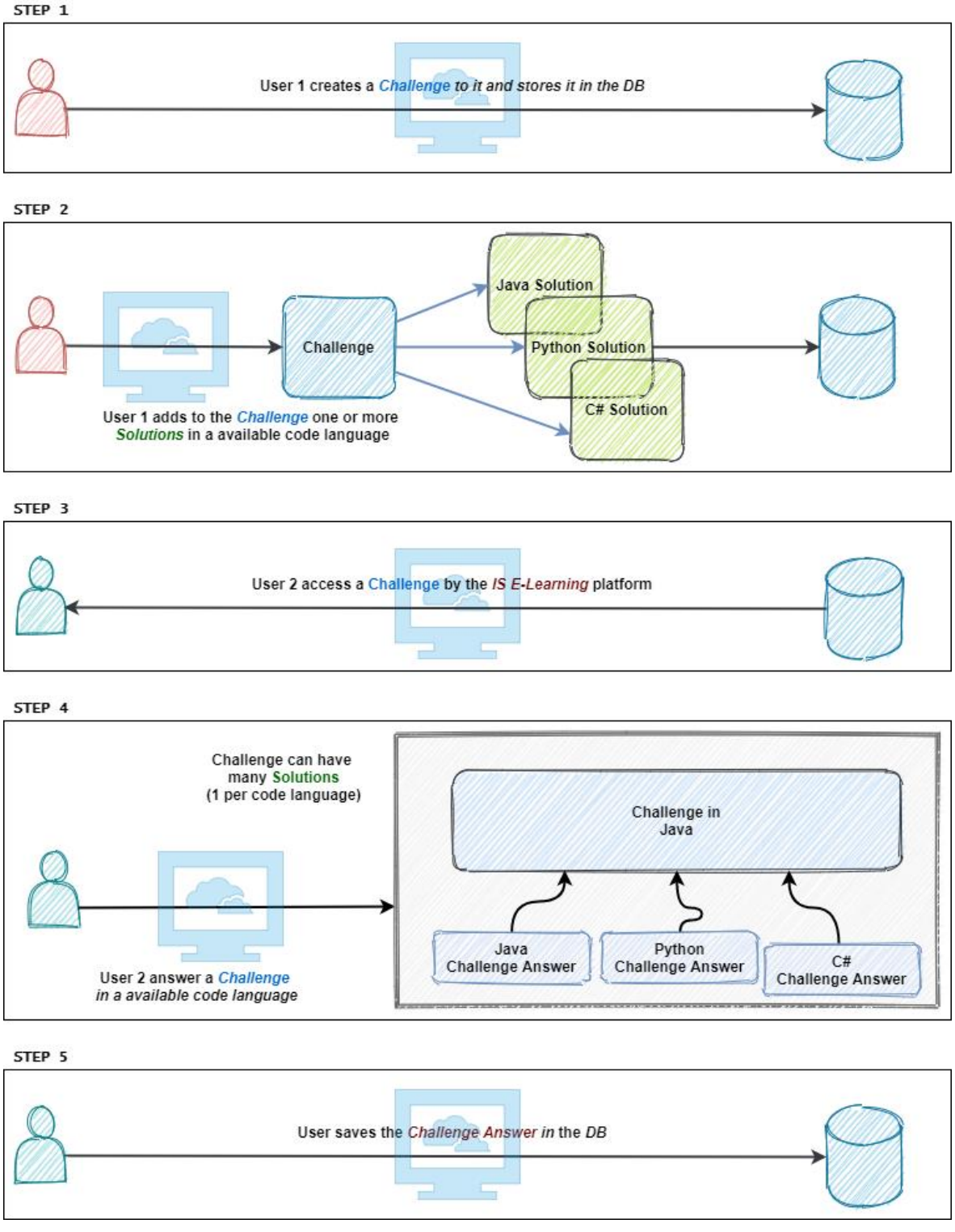

Figure 10 - Workflow to answer a *Challenge*

6.2.4 Questionnaires

The *Questionnaires* submodule is responsible for interaction and business logic with *Questionnaire* and *Questionnaire Answers* domains.

The scope of the *Questionnaires* submodule includes the domains of *Questionnaire*, *Questionnaire* Instance, *Questionnaire Answers* and *Questionnaire-Challenge*.

Implementation wise each of these domains exposes a different Spring RestController and has a different service to handle the business logic. The possible operations exposed by these controllers can be found in the Swagger documentations [35].

The *Questionnaire* domain entity represents the wrapper of all questionnaire components. It contains the information about the *Questionnaire* domain but also references all the *Challenges* in the questionnaire. The *Questionnaire* itself cannot be solve, but it serves as a template in which instances of it can be created and send to multiple users to be solved.

The *Questionnaire Instance* represents the entity that will store all the Questionnaires answers, and it is this resource that is going to be sent to a user for further operations. Multiple *Questionnaires Instance* can derive from a single parent *Questionnaire*.

The *Questionnaire-Challenge* domain entity is represented by a combination of the data model entity *Questionnaire* and a record from the many to many entity *QC*, which represents the associations between a questionnaire and a challenge data model entities, and the *Answer* to the *Challenge* presented in the *Questionnaire*.

The *Questionnaire Answers* domain entity is represented by a combination of the data model entities *Questionnaire Answer* and *Answer*. The *Questionnaires Answer* in data are equal to the *Challenges* answer ones since they both aim to resolve a *Challenge* problem. They also respect the same mandatory mutually exclusive relationship with different types of answers, with the difference that they are bound to a different resource.

Figure 11 - Workflow to answer *a Questionnaire*demonstrates the workflow that an user has to follow to answer a challenge.

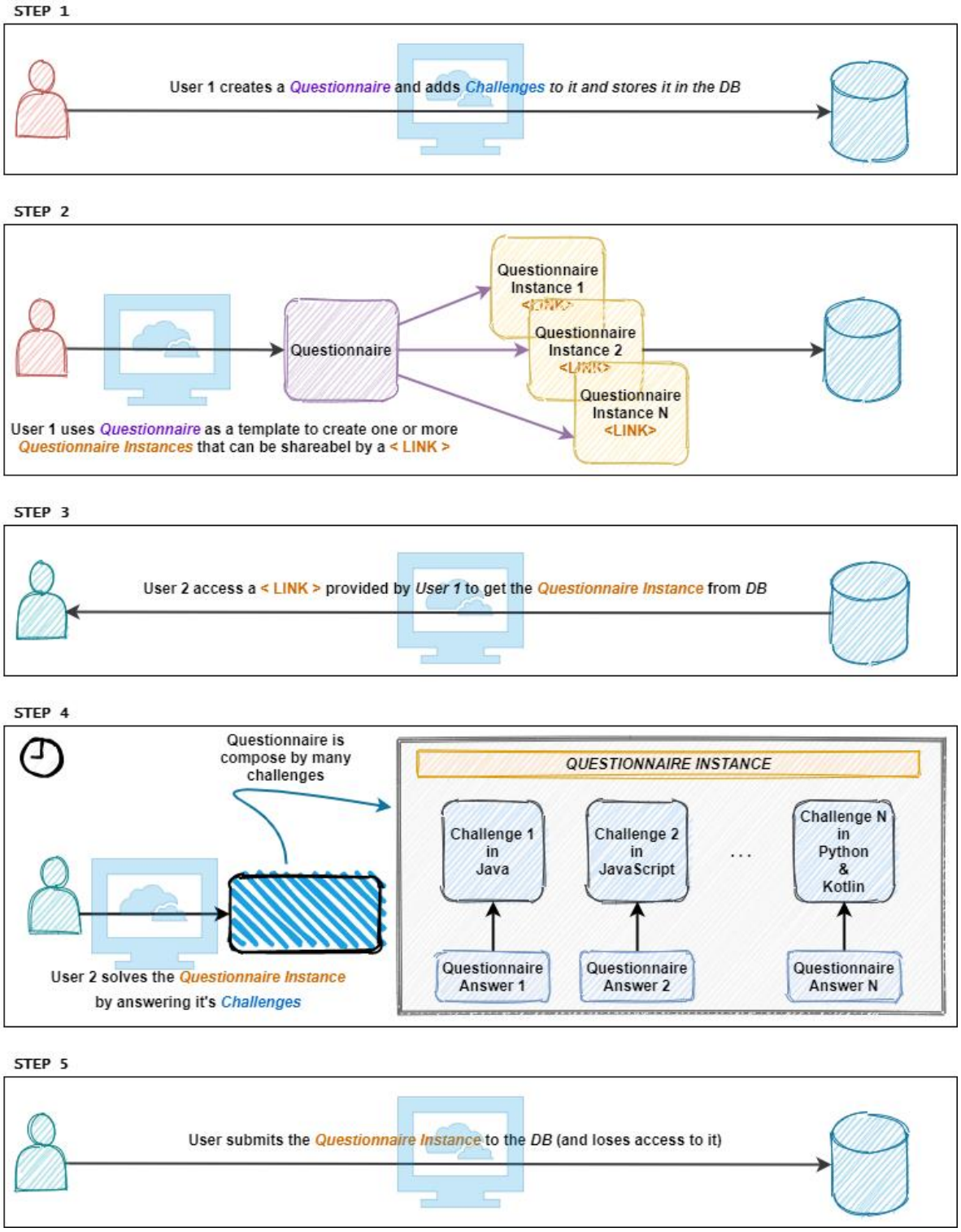

Figure 11 - Workflow to answer *a Questionnaire*

6.2.5 Code Languages

From a client's perspective there is the need to know which coding languages are supported for this solution, with that in mind a dedicated controller was created to provide that information. This controller contains a single non authenticated endpoint to allow for API clients to get a list of all supported coding languages.

6.2.6. Validations

The parameter validation is done on the service layer using the Javax validators. For these validators to work the service classes and methods need to be annotated with `@Validated` annotation. The validations were done on the service classes's methods, for each input parameter. This was done with annotations from package `javax.validation.constraints` for specific validations on String and Number types, e.g. `@Positive` , and using the annotation `@Valid` for other custom Reference types. For these custom reference types the class also had annotations from package `javax.validation.constraints` on fields which were to be validated.

6.2.7. Error Handling

Error handling responsibility in this solution is divided in several classes.

The class `ExceptionHandler` was marked with `@ControllerAdvice` annotation to enable application wide exception handling in a single class. This class has method to capture application specific exceptions, such as `ServiceException`, or `ConstraintViolationException`, and of course, the most generic Exception.

The basic flow in this class is as follows: an exception will be caught and then will be mapped to ApiError's instance that maps to the json+problem standard [35].

For instance, the class `ServiceException` extends Exception type, and its instantiated when an error on the services occurs. This object is constructed with the structure described on the RFC 7807 [39]and is mapped to an http error code when returning to the client.

6.3. Database

To configure the database a single master script "`CreateDB.sql` was created and can be found on the Wiki of the project repository [38].

This script includes not only the creation of the model with all its tables and dependencies, but also all the triggers and store procedures that allow a robust consistency on the database on the insertion of new data.

The single master script was created through the merge of several individual ones used on the development phase such as scripts to create, delete, drop and fill the database. Adding to this there is also a test script for the database. The other scrips can be found as store procedures on the project repository.

6.3.1. Database access

For database access to be possible a data source been was created to configured database related configurations, allowing a database connection to be used by the application to communicate with a given DB. On the application, every endpoint will need to access the database for perform the corresponding action except for the execution endpoint. In these cases it redirects the response to an execution environment with no need to access the DB.

To configure the database connection a properties file with the name `application.properties` must be added to the resource folder. An example of its configuration can be found on the wiki repository [38].

On the application spring Repositories were used to interact with the database, specifically `CrudRepositorty`. The `CrudRepositorty` is a repository with some CRUD operations already implemented.

To enable the repositories to perform database operations each repository was associated with a given class which represented the DB tables with which the repository would interact. As such these classes needed some configuration so the repository would be able to know the table name, the field names and other related entities were there. To this effect the package `pt.iselearning.services.domain` contains several classes which were annotated with annotations from `org.hibernate.annotations` and `javax.persistence` packages in order to identify DB table names, column names, which of the properties were primary keys, identity keys and relations to other domains line one to many or many to one.

6.4. Execution Environments

The implemented execution environments share some implementation details but for the most part require different dependencies and configurations.

In order to have fewer dependencies inside the container each container has only one runtime execution, which means the environment in which the code is meant to be executed will have to be the same on which the application will have to run. For the Java execution environment there will be a Java application listening to HTTP requests, for the C# execution environment there will be a C# application and so on. As a result, each application will use specific technologies. The technology in common between every application is Swagger, which will be used to document the REST API shared amongst every application [35].

Once the request for execution is sent to an execution environment the code is executed and its execution time is measured returning the result: a flag indicating if it was an error or not and the execution time in milliseconds. As an added security measure execution on these environments times out after 60 seconds (configurable) to avoid that a single request may consume prohibiting processing which could compromise infrastructure usage, e.g. infinite loop.

6.4.1. Java & Kotlin

Because both Java and Kotlin can be compiled to be executed on the JVM, the same application was used for both execution environment, with minor changes for each.

For these execution environments the application executed inside a Docker container is a Spring Boot application developed in Java using Maven as a build and dependency management tool.

The application is a simple one by design. Once the application receives an HTTP request it determines if there is the need to execute the code or the unit tests, writes the code to the file system, compiles the files and executes them. Both the compilation and the execution processes are done by executing bash or command line commands depending if the system is running on windows or Linux system. After the execution is complete with error or not, the result of the execution which was dumped to a text file is returned.

One of the main differences between the Java and Kotlin execution Environments is the environment on which the applications are executed, i.e. the docker container have different dependencies.

For the Java execution environment, the container is built on top of the OpenJDK 13 docker image, this allows the java application to run and the commands to compile and execute Java code to work.

For the Kotlin execution environment is not as simple, besides needing the JDK to run the Java application and executing Kotlin code compiled to the JVM it also needs the Kotlin compiler. This container was also built on top of the OpenJDK 13 docker image but the docker file also contained instructions to download and install the Kotlin JVM compiler.

6.4.2. JavaScript

For this execution environment was developed a NodeJS application, using the Express module [40] that process HTTP messages and based on the request body creates a file with to code that need to be executed and a file that contains the unit tests, the latter imports the function exported in the running code file to run the tests. With the file(s) created, a child process is spawn to run the defined code or the unit tests if defined in the message body.

The container is built on top of the node 14 image to enable the container to run the application.

6.4.3 C#

The C# Execution was developed in a C# application targeting .Net Core framework. This application adheres to the same contract as the other application: it receives an HTTP message and based on the contents it is identified if it is necessary to run unit tests or the code. In this application a new solution is created based on a previously created template, with two projects (one for the code and another for the unit tests), the application will then overwrite the contents of a file in each project that represents the code to run. With these files created a new process is spawn to run the code or the tests.

For this execution environment the container is built on top of Microsoft's image for .Net Core SDK 3.1 to run C# code.

6.4.4 Python

Python execution environment was developed in a python application that uses Flask [41] web framework. The API receives an HTTP message and will process the information on the request body, and will create a file with the code that needs to be executed and a file that contains the unit tests, if there is any to be tested. With the file(s) created, a subprocess is spawn to execute the, and will return result of the operation in case of success, or an according exception if anything goes wrong.

The details of mentioned operation as the expect input and output objects can be found in the Swagger documentations [28].

The container is built on top of a python official image from docker hub with version 3.8.5, which allows the python execution environment API to execute python code.

6.4.5 Additional language support

Adding support to a new language requires changes on 3 elements of this solution: Execution Environments, service layer and in the data base.

For the latter, a new entry must be added to the `languageUrlMap` of the service layer, with the key being the new language and the value the address and port of the machine running the new execution environment. It is also necessary to add the supported language to the enum `SupportedLanguages` and update the `executionEnvironments.properties` file to reflect the new execution environment implemented.

The Execution Environment is basically an application that provides exposes one endpoint, which will be used to run code and/or unit tests remotely.

This endpoint must respect a specific contract. The endpoint receives an Executable object, that must have a parameter that contains a field named `code` which is a string that represents the code that the user wants to run; a field named `executeTests` which is a Boolean that represents if the user wants

to test the code being sent against the unit tests defined and a field named `unitTests` which is a string that contains the unit tests to run with the code sent by the user.

This endpoint must return a structure that an `ExecutableResult` object, that contains a field named `rawResult` which is a string that represents either the correct result of the code submitted by the user or the errors that appeared while compile/running the code, a filed named `wasError`, which is a Boolean that represents the success of the code that was executed, e.g., if a unit test failed or the code was unable to be executed due to syntax errors, the flag `wasError` will be set as true; and a field named `executionTime` with the value of time in milliseconds with the time it took to execute the code.

The data base only requires that the record of the new language to be supported to be inserted in the code language table.

For this project it was devised that the Execution Environments would run in a container driven deployment methodology and as such in order for the new execution environment to run the same way a new docker file with the necessary commands to install any dependencies needs created, these commands also need to enable the container to expose a port so that the a new endpoint can be used by the service application.

7. Cloud deployment

As a side objective, we wanted *IS E-Learning* to be deployed into the cloud as containers, since both the server application and the execution environments were developed to be executed in Docker containers. This containerization is possible due to the stateless nature of the application and allows easier portability between different machines, among other advantages explained on Chapter 4.

Another concern regarding the deployment of the application is the scalability. With this in mind, there is the need to consider container orchestrators on which this deployment is possible and can scale container instances on demand.

7.1. Container orchestration tools

There are many container orchestrations tools that can be used for container lifecycle management. Some of the most popular options are Docker Swarm and Kubernetes.

7.1.1. Docker Swarm

Docker Engine v1.12.0 and later allow developers to deploy containers in Swarm mode [42]. A Swarm cluster consists of Docker Engine deployed on multiple nodes. Manager nodes perform orchestration and cluster management. Worker nodes receive and execute tasks from the manager nodes.

A service, which can be specified declaratively, consists of tasks that can be run on Swarm nodes. Services can be replicated to run on multiple nodes. In the replicated services model, ingress load balancing and internal DNS can be used to provide highly available service endpoints.

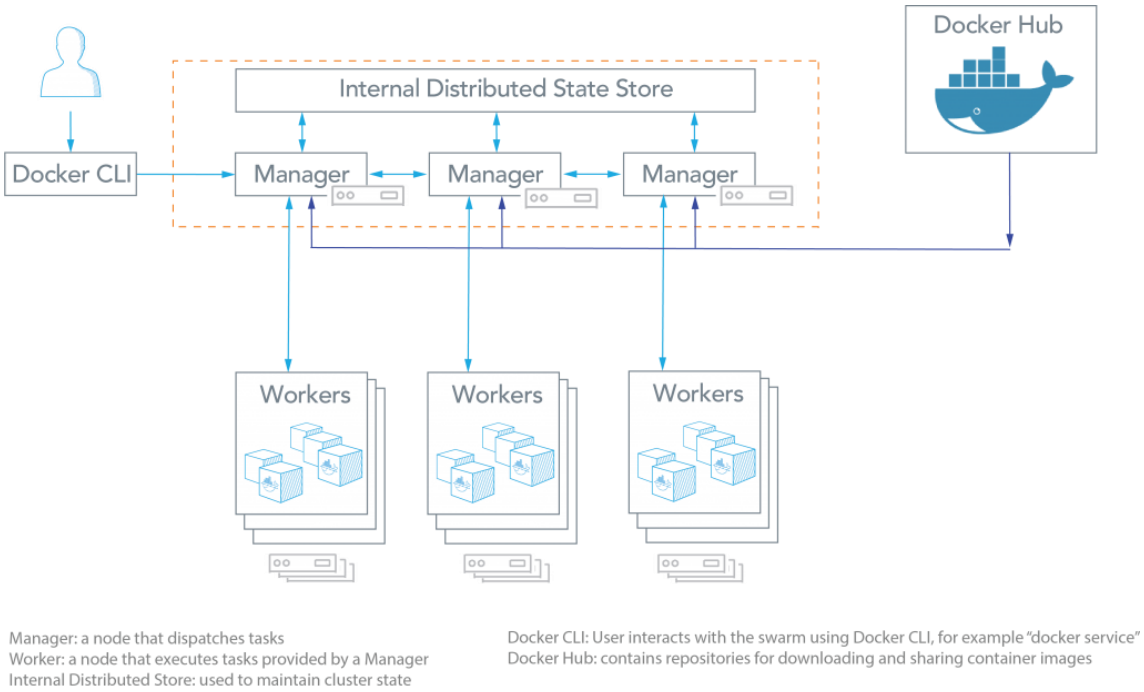

Figure 12 - Docker Swarm architecture

As can be seen from the figure above, the Docker Swarm architecture consists of managers and workers. The user can declaratively specify the desired state of various services to run in the Swarm cluster using configuration files.

Here are some common terms associated with Docker Swarm:

- **Node:** A node is an instance of a Swarm. Nodes can be distributed on-premises or in public clouds.
- **Swarm:** a cluster of nodes (or Docker Engines). In Swarm mode, services are orchestrated, instead of running container commands.
- **Manager Nodes:** These nodes receive service definitions from the user, and dispatch work to worker nodes. Manager nodes can also perform the duties of worker nodes.
- **Worker Nodes:** These nodes collect and run tasks from manager nodes.
- **Service:** A service specifies the container image and the number of replicas.

7.1.2. Kubernets

Kubernetes [43] is an open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes was built by Google based on their experience running containers in production using an internal cluster management system.

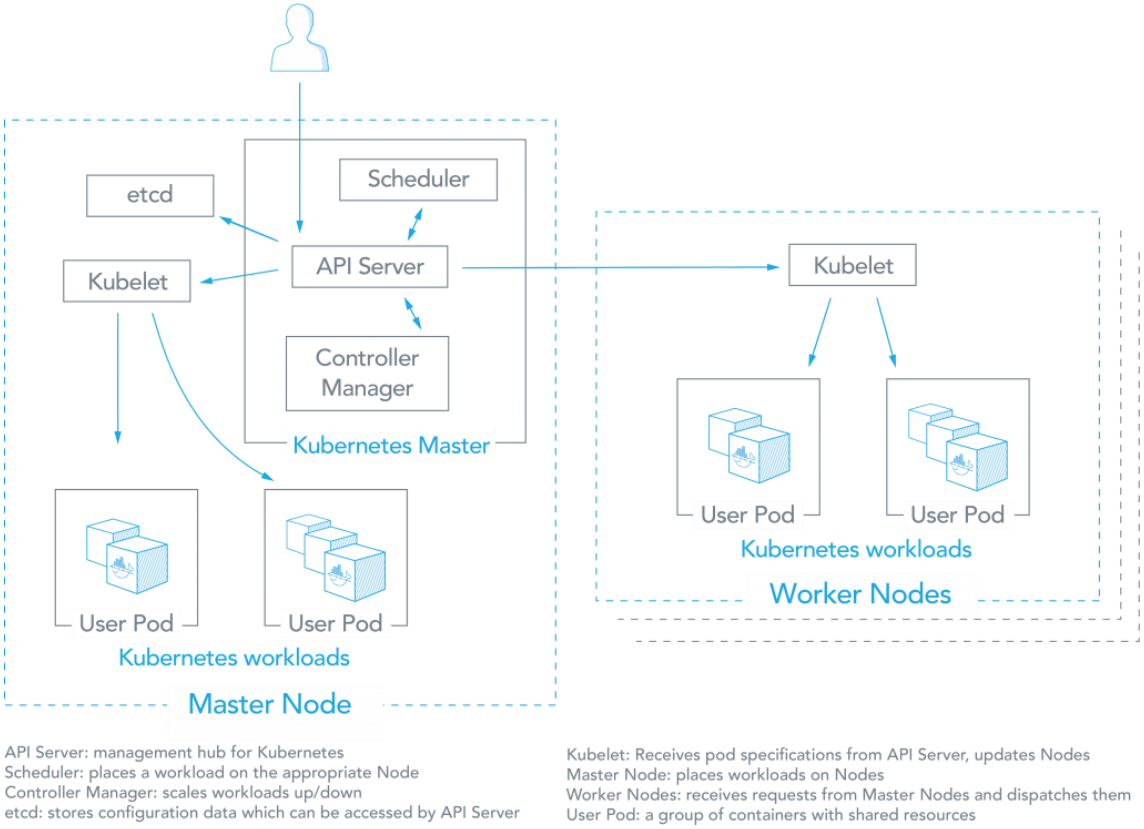

Figure 13 - Kubernetes architecture

As can be seen from the figure above, there are several components associated with a Kubernetes cluster. The aster node places container workloads in user pods on worker nodes or itself. The other components include:

- **etcd:** This component stores configuration data which can be accessed by the Kubernetes Master's API Server using simple HTTP or JSON API.
- **API Server:** This component is the management hub for the Kubernetes master node. It facilitates communication between the various components, thereby maintaining cluster health.
- **Controller Manager:** This component ensures that the cluster's desired state matches the current state by scaling workloads up and down.
- **Scheduler:** This component places the workload on the appropriate node – in this case all workloads will be placed locally on the user's host.

- **Kubelet:** This component receives pod specifications from the API Server and manages pods running in the host.

The following list provides some other common terms associated with Kubernetes:

- **Pods:** Kubernetes deploys and schedules containers in groups called pods. Containers in a pod run on the same node and share resources such as filesystems, kernel namespaces, and an IP address.
- **Deployments:** These building blocks can be used to create and manage a group of pods. Deployments can be used with a service tier for scaling horizontally or ensuring availability.
- **Services:** Services are endpoints that can be addressed by name and can be connected to pods using label selectors. The service will automatically round-robin requests between pods. Kubernetes will set up a DNS server for the cluster that watches for new services and allows them to be addressed by name. Services are the "external face" of your container workloads.
- **Labels:** Labels are key-value pairs attached to objects and can be used to search and update multiple objects as a single set.

7.2. Kubernetes vs Docker Swarm comparison

Docker Swarm focuses on ease of use with integration with Docker core components, providing a simple solution that is fast to get started, while Kubernetes remains open and modular, aiming to support higher demands with higher complexity. For much of the same reasons, Docker has been popular among developers who prefer simplicity and fast deployments. At the same time, Kubernetes is used in production environments by many high-profile internet companies running popular services.

7.2.1. Application deployment

Docker Swarm: Applications can be deployed as services in a Swarm cluster. Multi-container applications can specify using YAML files. Docker Compose can deploy the app. Tasks (an instance of a service running on a node) can be distributed across datacenters using labels. Multiple placement preferences can be used to distribute tasks further.

Kubernetes: Applications can be deployed using a combination of pods, deployments, and services. A pod is a group of co-located containers and is the atomic unit of a deployment. A deployment can have replicas across multiple nodes. A service is the "external face" of container workloads and integrates with DNS to round-robin incoming requests.

7.2.2. Application scalability constructs

Docker Swarm: Services can be scaled using Docker Compose YAML templates. Services can be global or replicated. Global services run on all nodes, replicated services run replicas (tasks) of the services across nodes. Tasks can be scaled up or down and deployed in parallel or in sequence.

Kubernetes: Each application tier is defined as a pod and can be scaled when managed by a deployment, which is specified in YAML. The scaling can be manual or automated and pods can be used to run vertically integrated application stacks such.

7.2.3. High availability

Docker Swarm: Services can be replicated among Swarm nodes. Swarm managers are responsible for the entire cluster and manage the resources of worker nodes. Managers use ingress load balancing to expose services externally. Swarm managers use Raft Consensus algorithm to ensure that they have consistent state information.

Kubernetes: Deployments allow pods to be distributed among nodes to provide HA, thereby tolerating application failures. Load-balanced services detect unhealthy pods and remove them. High availability of Kubernetes is supported. Multiple master nodes and worker nodes can be load balanced for requests from kubelet and clients. etcd can be clustered and API Servers can be replicated.

7.2.3. Load balancing

Docker Swarm: Swarm mode has a DNS component that can be used to distribute incoming requests to a service name. Services can run on ports specified by the user or can be assigned automatically.

Kubernetes: Pods are exposed through a service, which can be used as a load-balancer within the cluster. Typically, an ingress is used for load balancing.

7.2.4. Auto-scaling

Docker Swarm: Not directly available. For each service, it is possible to declare the number of tasks to be run. Manually scale up or down can be done and the Swarm manager automatically adapts by adding or removing tasks.

Kubernetes: Auto-scaling using a simple number-of-pods target is defined declaratively using deployments. CPU-utilization-per-pod target is available. Other targets are on the roadmap.

Table 2 - Docker Swarm vs Kubernetes

Docker Swarm vs Kubernetes

Pros	
▪ Deployment is simpler and Swarm mode is included in Docker Engine. ▪ Integrates with Docker Compose and Docker CLI – native Docker tools. Many of the Docker CLI commands will work with Swarm. Easier learning curve.	▪ Based on extensive experience running Linux containers at Google. ▪ Easy service organization with pods ▪ Can overcome constraints of Docker and Docker API. ▪ Autoscaling based on factors such as CPU utilization.
Cons	
▪ Does not have as much experience with production deployments at scale. ▪ Limited to the Docker API's capabilities. ▪ Services can be scaled manually.	▪ Do-it-yourself installation can be complex, but flexible. ▪ Uses a separate set of tools for management, including kubelet CLI.
Common features	
▪ Open source projects. ▪ Various storage options. ▪ Networking features such as load balancing and DNS. ▪ Logging and Monitoring add-ons.	

7.3. Google Cloud Run

In our planning we always aimed to deploy the IS E-Learning platform in a cloud environment as a way to simulate its viability in the future as an online e-learning application that is accessed by multiple users in simultaneous. That said, the details of configuration a container orchestration environment, although interesting, it is not a subject that we will address in our project, only its practical usage.

Like discussed, Docker Swarm and Kubernetes are great choices for containers orchestration platforms, offering advanced scalability and configuration flexibility, with the possibility to control over every aspect of container orchestration, from networking, to storage. However, at the current state of our project, we don't need that level of cluster configuration and monitoring, we just want to know how our platform behaves in a cloud environment scenario, and for that reason we choose to deploy our application on Google Cloud Run.

Another advantage of going with Cloud Run is the Platform-as-a-Service (PaaS) nature of this fully managed service. PaaS has many levels of abstraction, being the most relevant for our project, the low abstraction level platforms, as it is the Container-as-a-Service (CaaS) solutions, like Cloud Run. This enables several benefits, like application scaling automation, application management and DevOps tools that can be run on shared infrastructure. As a result, it eliminates the complexity and time-consuming factor of building and maintaining the underlying infrastructure.

Cloud Run is a serverless platform for deploying stateless containerized applications that don't require orchestration features like namespaces, or node allocation and management. It is possible to have all its configurations being fully managed by the Google Cloud Services "black box" - meaning that after implementing a microservice, it will have automatically scalable serverless execution, scaling to zero if there are no requests, using no resources in such cases. As managed compute platform, managed Cloud Run also supports essential configuration settings: the maximum concurrent requests a single container receives, the memory size to be allocated to the container as well as request timeout can be configured. No additional configurations or management operations are required.

To deploy *IS E-Learning* platform on Cloud Run, all it is necessary to implement is an address management service, by creating one containerized service for each operation - this is creating a Docker Image, which is the Cloud Run's unit of deployment, for each service: API and execution environments. Once the images have been created and registered in a container registry, they can be deployed to the managed Cloud Run with a single command, making the service up and running on a completely serverless platform.

7.4. Deployment on Google Cloud Run services

For deployment on Cloud Run both Execution environments and *IS E-Learning* applications need to follow three different steps:

1. Build Docker image
2. Push Docker image to Google Container Registry
3. Deploy image from google Container Registry to a Cloud Run Service

7.4.1. Pre-requisites

One of the prerequisites for the steps below to function properly is to build the project's locally to comply with the Dockerfile commands, these builds are documented on the repository's wiki [38].

The whole deployment process for Cloud Run Services is well documented by google [44] [45].

And before the deployment can take place there are a few steps which needed to be performed, such as: activating the necessary APIs for the whole deployment process; install the *gcloud* tool to perform the deployments; setup service user with the necessary permissions to execute the deployment commands.

Several APIs were activated: Cloud Build, Cloud Run, Container Registry, and Resource Manager. The Cloud Build API is necessary for building images and artifacts in the cloud; the Cloud Run API is necessary to deploy user-provided container images to Cloud Run Services; Container Registry API is necessary to store, manage and secure Docker images; Resource Manager API provides methods for creating, reading and updating project metadata.

As for the service user, a new user was created as owner of the project, as such the user had edit access to every resource of the project.

7.4.2. Build and Push Docker image

Executing the command `gcloud builds submit --tag gcr.io/<project-id>/<image-name>` for a given project with a given image name (arbitrary unique name provided by the user) will build the image and deploy it to Google's container Registry of the project.

7.4.3. Deploy to Cloud Run

Executing the command `gcloud run deploy --image gcr.io/<project-id>/<image-name> --platform managed` will deploy the image the a given name on a given project to a Cloud Run service. After this command is executed successfully a public endpoint is provided for access to the newly deployed service. It is also possible to deploy to a Cloud Run service from a Container registry image on the GCP UI.

Other relevant configurations can be done on the Cloud Run service to adjust container capabilities, e.g. the memory available for each container which had to be adjusted from some containers, environmental variables, and service connections.

Cloud Run service connections are particularly relevant when deploying the application which needs to connect to the Cloud SQL database. Because fully managed Cloud Run Services are executed on their own VPC, to access private other VPC's resources (like the Cloud SQL database) an explicit connection between the Cloud Run service and the other VPC needs to be configured [46]. For this connection to be configures a Serverless VPC Access connector needs to be created and configured to be used by the Cloud Run Service [47] [48], once configured the Cloud Run Service has access to resources inside the VPC, including the database on Cloud SQL.

7.5. Database Cloud SQL

For the application to function properly, it needs to be connected to a database to persist relevant state. To that effect a Database Cloud SQL PostgreSQL client was configured with minimum possible specs as to reduce operation cost. A relevant detail of the configuration is that public access was turned off for security reasons, this means the database can only be accessed through a private IP accessible only within the project's VPC which is accessible by the services deployed on Cloud Run.

The initial database setup requires the execution of several SQL scripts, to that effect the import functionality was used to import all the script and *gcloud* CLI to execute PL/SQL commands directly. To connect to the DB the command `gcloud sql connect <instance-name> --user=postgres` was used, after which the user is prompted to insert the password for the specified user. With the connection open the CLI allowed PL/SQL commands to be user, the command `call <stored-procedure-name>();` was used to run the necessary stored procedures to create and fill the database model.

8. Final Remarks

This project is the epitome of all the knowledge we acquired during our stay at ISEL.

We believe that we delivered a solid project, that corresponds to everything that we committed to do. It offers many diversified services in a user friendly interface; it is modular, being capable of being deployed with minimal effort in local our cloud environments; scalable and secured, being an stateless API, it can run on containerized services; and nonetheless it is a open source project, that we hope that in the future, ISEL students may contribute, work together and put their knowledge to good use in a common goal.

We are confident that the platform is set and ready to be use on a regular basis. All services mentioned on the RequirementsIntroductionwere implemented with success, and were extensively tested in every step of the development. The platform was designed to be easily extensible, since the best practices that were described on the 1.2. Methodology chapter are followed. Also, as an important note, there is substantial documentation about this project: every class, method or function has clean and succinct comments; all contracts and API is well documented; there are postman collections with every possible test case; how to deploy the execution environments or how to deploy in project on a cloud service can be found on our swagger or wiki documentation.

8.1. Future work

Even though a lot was accomplished in this project, there is plenty of room for improvement on existing functionalities as well opportunities to add value with new functionalities.

Although the UI followed development good practices and is easy to use, it can be always improved both from a design as well as a UX perspective, taking into account usability tests.

The backend is currently one application that contains every end point and every operation (monolith), but in the future, maybe it would be a better solution, if these services could be migrated to individual microservices. Although this has several implications in these changes, the modern microservice architecture provides several advantages [49].

Many of our services were implemented but some are not used on the web client, unfortunately due to the lack of time. For instance, the ability to add tags to a challenge to ease the search, or the possibility to track the record of challenges completed by.

There is support for 5 programming languages, but there are still many out there missing in *IS E-Learning*. Furthermore, support for external libraries for every coding language could be added to allow more flexibility while solving challenges.

Currently the platform is scalable and supports concurrency, being possible to be accessed by a large number of users, yet this is only true, due to the use of the Google Cloud services, that handles the load balancing automatically. This is not viable solution if this platform is to be implemented in ISEL, due to the monetary costs that come with the service. One alternative solution could be to implement a new service that would handle the requests, storing and managing them in a queue data structure, and that ultimately would serve as reverse proxy.

9. Lexicon

API – Application Program Interface

CaaS – Container-as-a-Service

CLR – Common Language Runtime

DB – Database

ISEL – Instituto Superior de Engenharia de Lisboa

IT – Information Technology

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

JDK – Java Development Kit

JSX – JavaScript XML

JVM – Java Virtual Machine

OSS - Open Source Software

PaaS – Platform-as-a-Service

REST – Representational State Transfer

SPA – Single Page Application

UI – User Interface

10. References

[1] S. S. Matthias Muller-Hannemann, Algorithm Engineering: Bridging the Gap Between Algorithm Theory and Practice, Springer; 2010 edition, 2010.

[2] "opensource.org," [Online]. Available: https://opensource.org/. [Accessed 09 09 2020].

[3] G. W. Amy Brown, "The Architecture of Open Source Applications," , lulu.com, 2012.

[4] "is_learning," [Online]. Available: https://github.com/joaoesantos/ise_learning. [Accessed 01 06 2020].

[5] "Azure Boards," [Online]. Available: https://azure.microsoft.com/en-us/services/devops/boards/. [Accessed 09 09 20].

[6] "Trunk Based Development," [Online]. Available: https://trunkbaseddevelopment.com/. [Accessed 12 06 2020].

[7] W. S. Humphrey, "Managing the Software Process," 1989.

[8] "Semantic Versioning 2.0.0 | Semantic Versioning," [Online]. Available: https://semver.org/. [Accessed 24 7 2020].

[9] K. Wiegers, Software Requirements (Developer Best Practices), Microsoft Press, 2013.

[10] "AlgoExpert," [Online]. Available: https://www.algoexpert.io/product. [Accessed 25 04 2020].

[11] "HackerRank," [Online]. Available: https://www.hackerrank.com/. [Accessed 25 04 2020].

[12] "LeetCode," [Online]. Available: https://leetcode.com/. [Accessed 25 04 2020].

[13] "CodeWars," [Online]. Available: https://www.codewars.com/. [Accessed 25 04 2020].

[14] "CodeChef," [Online]. Available: https://www.codechef.com/. [Accessed 25 04 2020].

[15] "React – A JavaScript library for building user interfaces," [Online]. Available: https://reactjs.org. [Accessed 24 04 2020].

[16] A. Banks, Learning React: Functional Web Development with React and Redux, O'Reilly Media, 2017.

[17] "Introducing JSX – React," [Online]. Available: https://reactjs.org/docs/introducing-jsx.html. [Accessed 24 04 2020].

[18] "Spring Framework," [Online]. Available: https://spring.io/projects/spring-framework. [Accessed 24 04 2020].

[19] "2. Introduction to the Spring Framework," [Online]. Available: https://docs.spring.io/spring/docs/4.3.x/spring-framework-reference/html/overview.html. [Accessed 24 04 2020].

[20] "Spring Boot," [Online]. Available: https://spring.io/projects/spring-boot. [Accessed 04 24 2020].

[21] "API Documentation & Design Tools for Teams | Swagger | Swagger," [Online]. Available: https://swagger.io/. [Accessed 24 04 2020].

[22] "Empowering App Development for Developers | Docker," [Online]. Available: https://www.docker.com/. [Accessed 17 04 2020].

[23] "Docker Documentation | Docker Documentation," [Online]. Available: https://docs.docker.com/. [Accessed 17 04 2020].

[24] S. M. Jain, Linux Containers and Virtualization - A kernel perspective, Independently published, 2019.

[25] "Docker Hub," [Online]. Available: https://hub.docker.com/. [Accessed 17 04 2020].

[26] "What is REST – Learn to create timeless REST APIs," [Online]. Available: https://restfulapi.net/. [Accessed 01 05 2020].

[27] "Material-UI," [Online]. Available: https://material-ui.com/. [Accessed 13 06 2020].

[28] "Material Design," [Online]. Available: https://material.io/design. [Accessed 13 06 2020].

[29] "CodeMirror," [Online]. Available: https://codemirror.net/. [Accessed 13 06 2020].

[30] "Formik · Build forms in React, without the tears.," [Online]. Available: https://jaredpalmer.com/formik/. [Accessed 06 2020].

[31] "Formik Material-UI," [Online]. Available: https://stackworx.github.io/formik-material-ui/. [Accessed 06 2020].

[32] "yup - npm," [Online]. Available: https://www.npmjs.com/package/yup. [Accessed 06 2020].

[33] "Tutorial · Formik," [Online]. Available: https://jaredpalmer.com/formik/docs/tutorial#schema-validation-with-yup. [Accessed 06 2020].

[34] "OpenAPI-Specification/3.0.2.md at master · OAI/OpenAPI-Specification," [Online]. Available: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md. [Accessed 27 04 2020].

[35] "IS E-Learning Swagger UI," [Online]. Available: https://joaoesantos.github.io/ise_learning/apiDocumentation. [Accessed 27 04 2020].

[36] "Database - Third Normal Form (3NF) - Tutorialspoint," [Online]. Available: https://www.tutorialspoint.com/sql/third-normal-form.htm. [Accessed 27 04 2020].

[37] S. B. N. a. R. Elmasri, FUNDAMENTALS OF DATABASE SYSTEMS, Pearson Education, Inc., 2004.

[38] "Home · joaoesantos/ise_learning Wiki," [Online]. Available: https://github.com/joaoesantos/ise_learning/wiki. [Accessed 2020].

[39] "rfc7807," [Online]. Available: https://tools.ietf.org/html/rfc7807. [Accessed 12 09 2020].

[40] "Express," [Online]. Available: http://expressjs.com/. [Accessed 10 09 2020].

[41] "Flask," [Online]. Available: www.flask.palletsprojects.com. [Accessed 28 08 2020].

[42] "Docker Swarm Mode," [Online]. Available: https://docs.docker.com/engine/swarm/key-concepts/. [Accessed 03 09 2020].

[43] "Kubernetes," [Online]. Available: https://kubernetes.io/. [Accessed 03 09 2020].

[44] "Deploying to Cloud Run | Cloud Build Documentation | Google Cloud," [Online]. Available: https://cloud.google.com/cloud-build/docs/deploying-builds/deploy-cloud-run. [Accessed 2020 09 15].

[45] "Quickstart: Build and Deploy | Cloud Run Documentation | Google Cloud," [Online]. Available: https://cloud.google.com/run/docs/quickstarts/build-and-deploy#java. [Accessed 2020 09 15].

[46] "Configuring Serverless VPC Access | Google Cloud," [Online]. Available: https://cloud.google.com/vpc/docs/configure-serverless-vpc-access. [Accessed 15 09 2020].

[47] "Connecting from Cloud Run (fully managed) to Cloud SQL," [Online]. Available: https://cloud.google.com/sql/docs/postgres/connect-run#private-ip_1. [Accessed 15 09 2020].

[48] "Connecting to a VPC network | Cloud Run Documentation | Google Cloud," [Online]. Available: https://cloud.google.com/run/docs/configuring/connecting-vpc#configuring. [Accessed 15 09 2020].

[49] "Advantages and Disadvantages of Microservices Architecture," [Online]. Available: https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/. [Accessed 1 9 2020].

[50] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Algorithm_engineering. [Accessed 20 04 2020].

[51] "Spring Security," [Online]. Available: https://spring.io/projects/spring-security. [Accessed 24 04 2020].

11. Annex

Annex A. Supported versions of container dependencies

Table 3 - Supported versions of container dependencies

Container	Dependency	Supported Version
Java Execution Environment	Open JDK	13
Kotlin Execution Environment	Open JDK	13
Kotlin Execution Environment	Kotlin compiler	1.3.71
JavaScript Execution Environment	Nodejs Runtime	14.0.0
C# Execution Environment	SDK	3.1
Python Execution Environment	Python	3.8.5

Annex B.Data Model

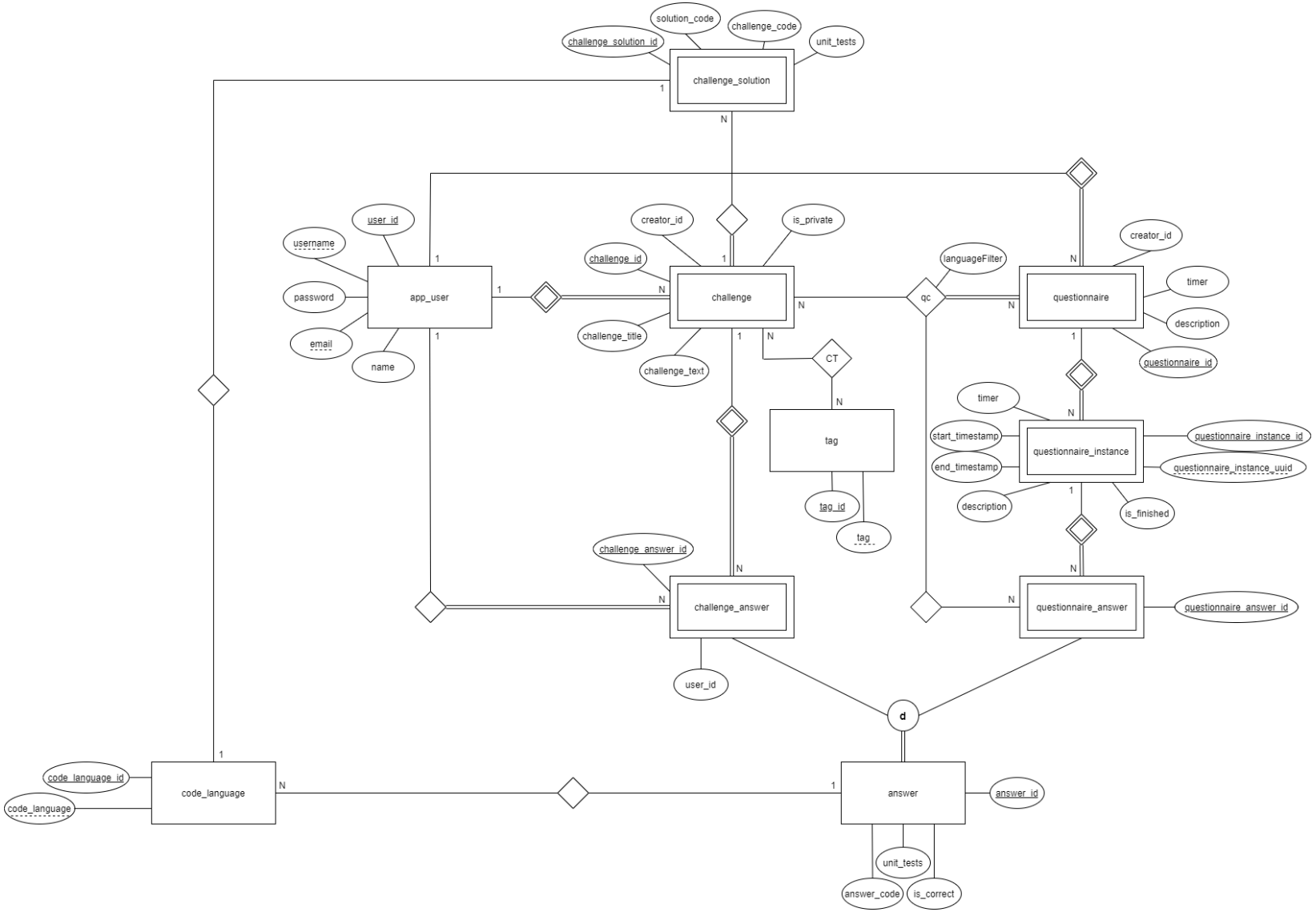

Figure 14 – Data model