

**Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores (traduzir)**

**Progress Report**

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

Relatório de progresso para a Unidade Curricular de Programação da Licenciatura em Engenharia Informática e de Computadores

(traduzir)

Advisors: Cátia Vaz, José Simão

04 – May – 2019

Resumo

(não temos isto)

Índice

[1. Introdução 10](#_Toc38739003)

[2. Requirments 12](#_Toc38739004)

[3. Related work 14](#_Toc38739005)

[4. Arquitecture 16](#_Toc38739006)

[Front end 16](#_Toc38739007)

[Services 17](#_Toc38739008)

[Data Model 18](#_Toc38739009)

[Execution Environments 19](#_Toc38739010)

[Java & Kotlin 20](#_Toc38739011)

[JavaScript 20](#_Toc38739012)

[C# 20](#_Toc38739013)

[Python 20](#_Toc38739014)

[5. Project progress 22](#_Toc38739015)

[Lexicon 25](#_Toc38739016)

Lista de Figuras

[Figure 1 – Project Modules interactions 16](#_Toc38739051)

[Figure 2 – Detailed view of Services Module including DB 17](#_Toc38739052)

[Figure 3 – Data Model 18](#_Toc38739053)

[Figure 4 – Detailed view of ExecutionEnvironments Module 19](#_Toc38739054)

[Figure 5 – Planned Schedule before progess report delivery 22](#_Toc38739055)

[Figure 6 – Planned Schedule after progess report delivery 23](#_Toc38739056)

Lista de Tabelas

# Introdução

(quintela)

# Requirments

(Santos)

# Related work

(quintela)

# 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, in this chapter a subset of the most relevant technologies were selected to be described in more detail.

## 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 (ref - <https://reactjs.org/>).

One of the big advantages of using react is being able to build components which can be independent from other components and reused in different 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 (<https://reactjs.org/docs/introducing-jsx.html> – referencia?).

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.

## 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 (referencia algures para <https://spring.io/projects/spring-framework>).

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. (refs - <https://docs.spring.io/spring/docs/4.3.x/spring-framework-reference/html/overview.html>, <https://spring.io/projects/spring-framework>).

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 will be used on this project.

### Spring Boot

Spring boot makes it easier for the develop Spring applications. Includes embedded Tomcat, Jetty or Undertown as web application servers allowing the development of standalone applications, automatically configure 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 (ref - <https://spring.io/projects/spring-boot>).

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

### Spring Security

The Spring security is authentication and authorization framework. A big advantage of using Spring security is its highly customizable and extendable to support the necessary requirements (ref - <https://spring.io/projects/spring-security>).

## Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. (ref - <https://www.docker.com/> https://docs.docker.com/)

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 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 the containers is that they are lightweight, require fewer resources and have very quick start up times, and secure, the container provides 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 from that image and only then can the image with the application be executed. 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 (ref - <https://hub.docker.com/>). For most cases custom docker images need to be built and these can be built recurring to Docker files.

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.

## Swagger

Swagger enables developers to describe their API’s structure in such a way that it is possible to build beautiful and interactive API documentation (ref - <https://swagger.io/>). 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.

# Arquitecture

To develop the ISE-Learning application 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 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.

On the image below is shown 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 1 – Project Modules interactions

## Front end

The front end will be a single page application enabling a user to interact with the application through an UI. This module will be implemented with React, because it will be a SPA the module React Router will also be used.

For this use case the components were built using NodeJS. Adding to this some external libraries were used to support the UI development. Such library is Material UI, a popular React UI framework which contains many implemented components enabling faster and easier development.

To take advantage of React and follow good development practices, on this project there is the 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.

(mais la para a frente fazer um diagrama de navegação das páginas?)

## 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 the image below it’s shown in more detail how the service modules are structured. There are 4 main sub modules: Challenges; Questionnaires; Users; Authentication.

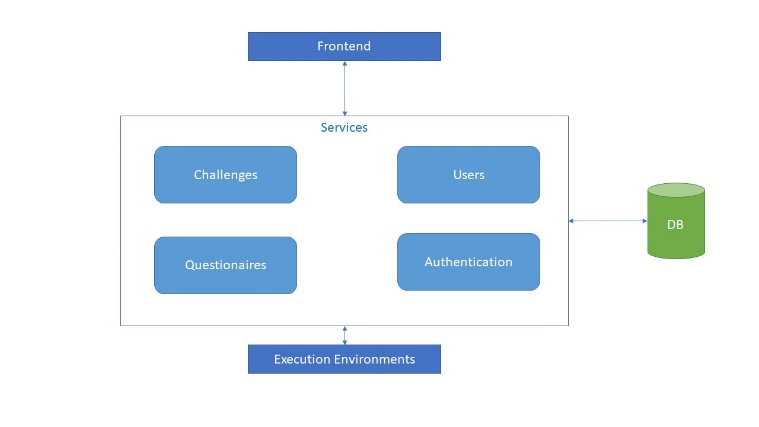


Figure 2 – Detailed view of Services Module including DB

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

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

Another detailed shown on the picture above is the Database, this 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.

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 is a Postgres relational database and the API is documented with the Open API 3.0 standard (<https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md> - referencia) hosted on Swagger UI ([https://joaoesantos.github.io/ise\_learning/apiDocumentation](https://joaoesantos.github.io/ise_learning/apiDocumentation/) - referencia).

### 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 the image below.

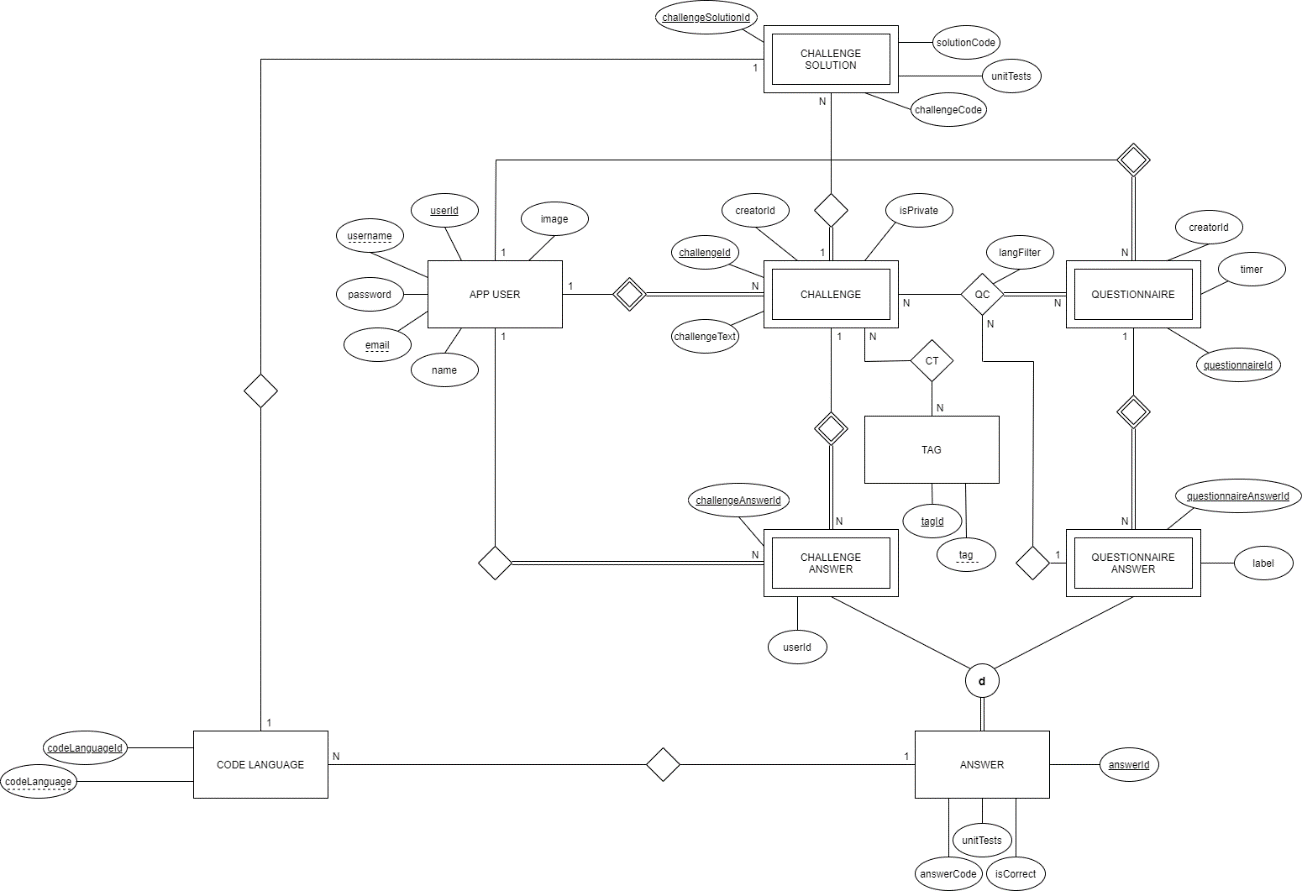


Figure 3 – Data Model

To follow good practices of data model design this data model follows 3NF rules for normalization (ref - <https://www.tutorialspoint.com/sql/third-normal-form.htm>).

An implementation detail worthy of note is the mandatory mutually exclusive relationships between Answer and its 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.

## Execution Environments

This module contains multiple applications, one for each type of runtime 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 below, 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 is 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.

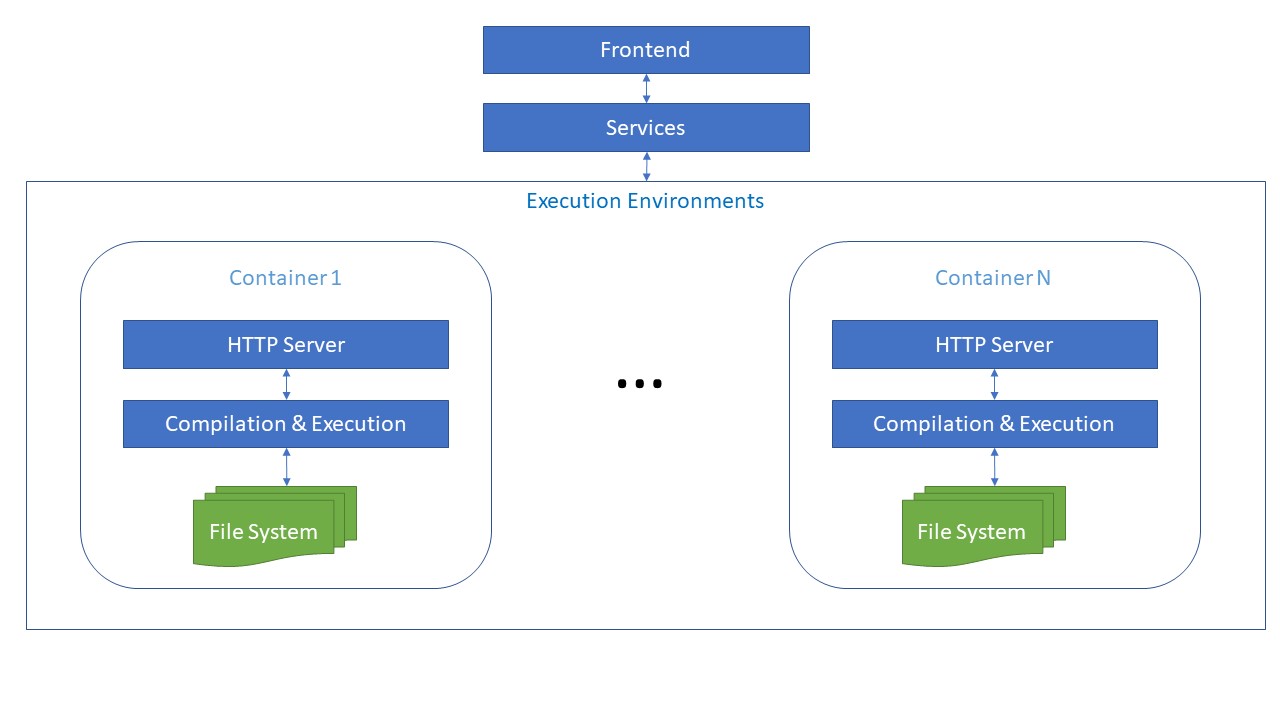


Figure 4 – Detailed view of ExecutionEnvironments Module

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#, Python.

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. To specify, 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.

### 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 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 it. 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 is executions, the docker container.

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.

### JavaScript

(not implemented)

### C#

(not implemented)

### Python

(not implemented)

# Project progress

The project has mostly been on schedule. Some time constraints have impacted the planned schedule such as jobs or college projects for other courses.

On the image below, which represents the planned scheduled up to the progress report delivery, are highlighted the activities which have not been finished. Only one activity was not finished, the Node execution environment is currently being developed.

This means the remaining activities are finished, summarizing:

1. The Database is set up and documented
2. React framework was configured for development
3. JVM execution environment is developed and documented, including for Java and Kotlin
4. The page on which the user could execute code is developed
5. The home page is developed
6. The progress report was finished and each team member has an individual presentation prepared



Figure 5 – Planned Schedule before progess report delivery

Despite this small set back, the plan is on the right path for a full delivery within the specified timeline.

On the image below is the planned schedule from the progress report delivery date onwards. An important detail regarding this plan is the home page, which was planned to be finished on the week starting on May 4th and has already been finishing, meaning there is a task finished ahead of schedule.



Figure 6 – Planned Schedule after progess report delivery

Regarding the remaining tasks there is some uncertainty, specifically on execution environments tasks for Python and CLR since some of the technologies necessary to perform those tasks are not yet well known and regarding the cloud environment deployment for similar reasons, the technology is not yet chosen (cloud provider) and the technology is not well known by the team members.

On a brighter note development capacity is predicted to pick up starting on July 27th once the exams are finished and the team will no longer has workload related to other courses.

# Lexicon

DB – Database

UI – User Interface

# References