Technical Report - Project specifications

CodeSpell

Course: PI - Projeto em Informática

Students:

 Artur Romão, nMec 98470

 Daniela Dias, nMec 98039

 Diogo Cruz, nMec 98595

 Hugo Gonçalves, nMec 98497

 Paulo Pereira, nMec 98430

Project abstract: A Graphical Reactive Platform for Programming Learning

Members roles:
Artur Romão - DevOps
Daniela Dias - Team Manager
Diogo Cruz - Product Owner
Hugo Gonçalves - Architect
Paulo Pereira - Architect

Contents

1 Introduction

Context

Knowing how to read and write code is becoming more and more important nowadays. It is no longer something just for engineers or IT personnel, in a world that is becoming more digital every day[1], it is a useful skill that can pave the way for great future opportunities. The need for programmers that can develop technological solutions for today's problems is also growing fast and is a market in expansion[2].

Teaching or learning how to code can sometimes be frustrating and difficult. New students can feel overwhelmed with so many new concepts and teachers may have a hard time finding real-life examples that will help their students better understand the problems and where programming can be useful. This may get new students unmotivated and make them quit learning to program.

Therefore, the context that led to the development of our solution can be summarized by the following topics:

- There is a pent-up demand for programmers in the technology market;
- The need for programmers should keep on growing in the next few years;
- There are more and more people interested in learning how to code.
- As the world gets progressively more digital, understanding the basic concepts of how a digital system works will be a huge advantage.

2 Product Concept

2.1 Problem

- Learning how to write code can be a challenging and frustrating task (particularly for non IT-related engineering students);
- It is hard to maintain new programmers motivated to keep learning;
- Illustrating where the initial concepts of a programming language can be applied is a difficult task;
- Creating examples, where the concepts can be applied, is a time-consuming task and many times depends on abstract examples.

---

[1] https://www.publico.pt/2022/06/05/sociedade/noticia/nova-matematica-secundario-poe-alunos-programar-reforca-estatistica-2008768

[2] https://www.economicmodeling.com/supply-and-demand-of-software-developers/

2.2 Goals

- Support teaching our users how to write code in a simple, visual, and fun way;
- Create a competitive experience to keep users interested in improving their skills;
- Create an easy-to-use platform that can be used by regular users and instructors;
- Provide documentation, tips, and possible solutions to teach our users;

2.3 State-of-the-Art

In this chapter, we will take a look at the current state-of-the-art focusing on the existing work in this area, the comparison between that work and our vision for the application, as well as illustrating the technologies we will use.

2.3.1 Related Work

In order to inspire our future work and establish expectations, we had the need to explore what technologies similar to our conception of the product were already on the market. As a result of our research, we found several examples of software that closely resemble our goals and ambitions.

Of those, we would like to highlight the following:

Robo Instructus[3], a puzzle game in which players maneuver a robot by issuing instructions via a simple programming language.

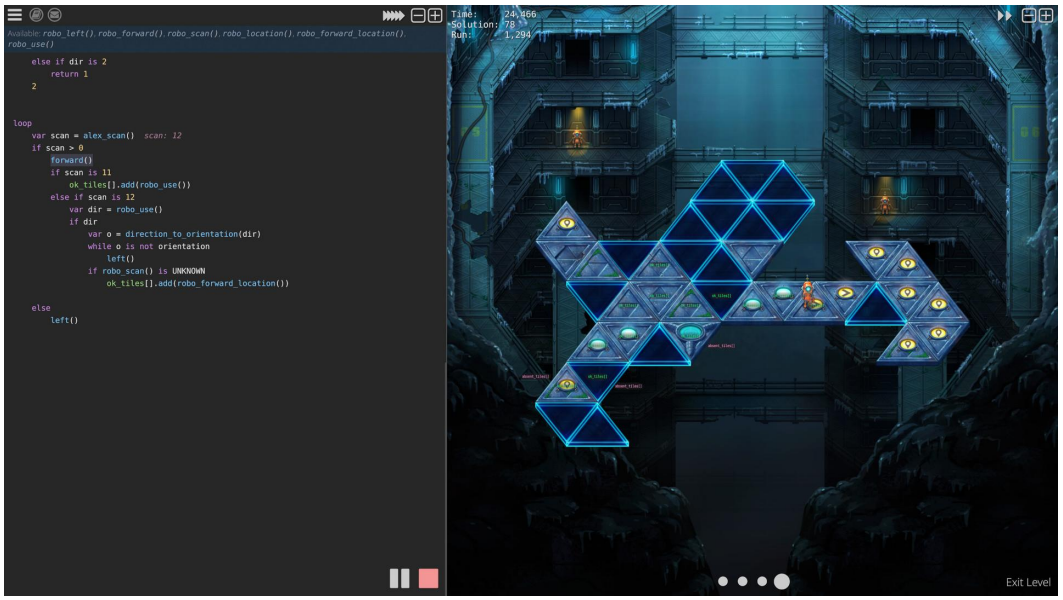


Fig. 1: Screenshot of “Robo Instructus”

[3] https://store.steampowered.com/app/1032170/Robo_Instructus/

while True: learn()[4], another puzzle game whose purpose is to build a cat-to-human translation system using machine learning principles.

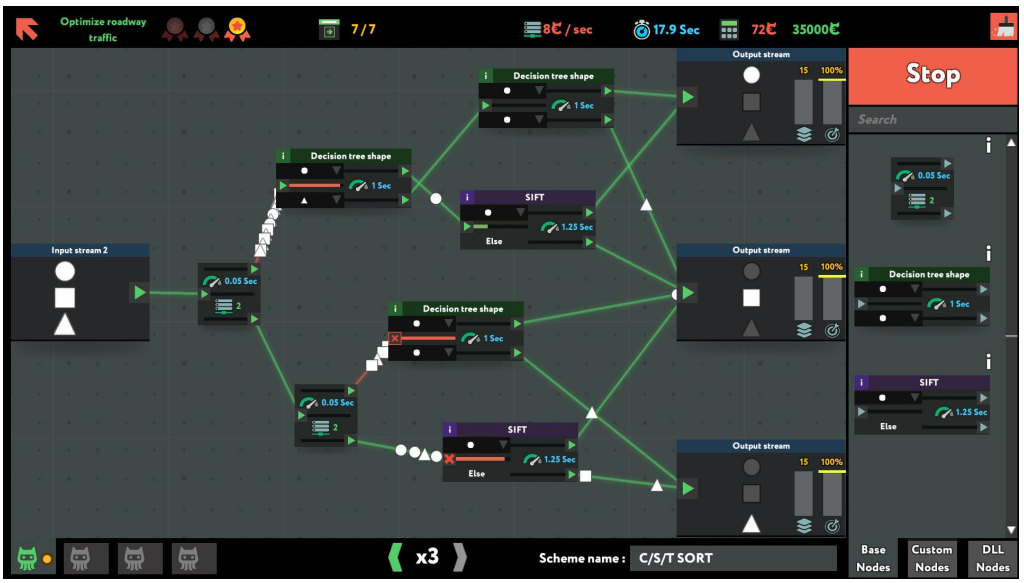


Fig. 2: Screenshot of “while True: learn()”

Human Resource Machine[5], a game whose aim is to program little office workers to solve puzzles.

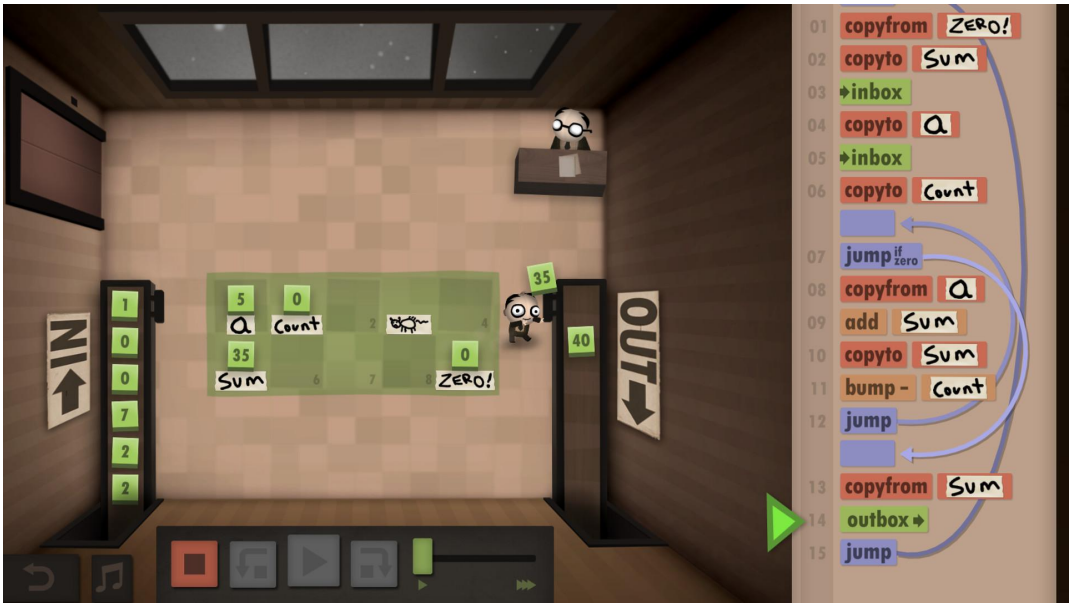


Fig. 3: Screenshot of “Human Resource Machine”

[4] https://store.steampowered.com/app/619150/while_True_learn/
[5] https://store.steampowered.com/app/375820/Human_Resource_Machine/

Grasshopper[6], a free coding app for beginners that approaches the most relevant and nuclear programming aspects and skills.

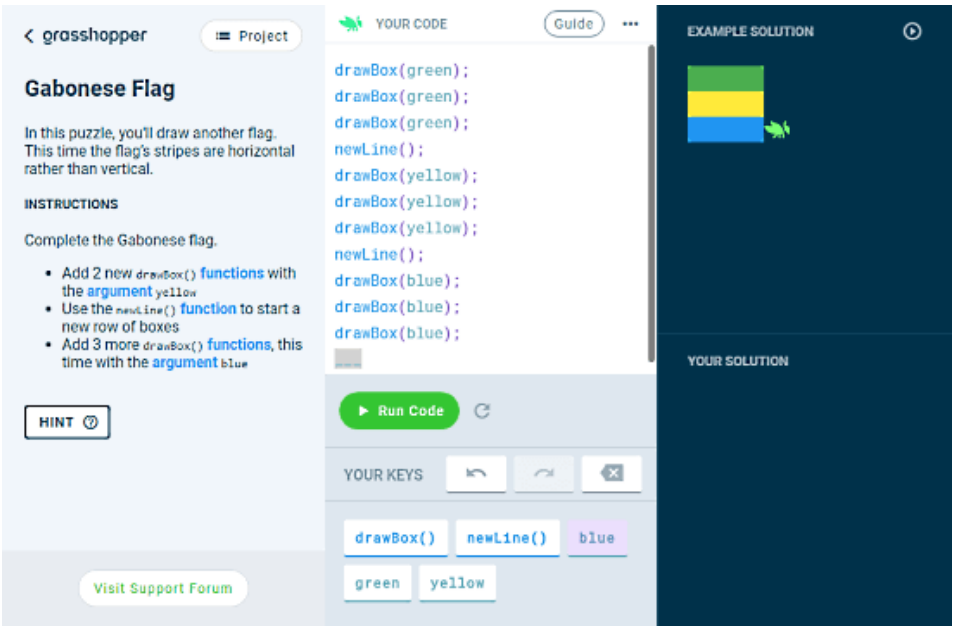


Fig. 4: Screenshot of "Grasshopper"

- **Scratch**[7], the world's largest free coding community for children with a simple visual interface allowing users to create digital stories, games, and animations.

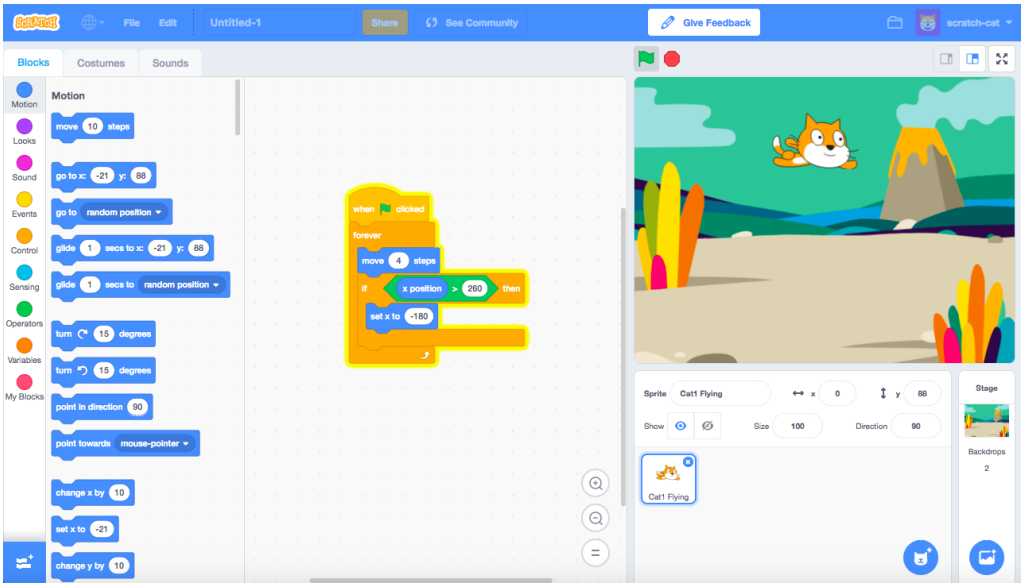


Fig. 5: Screenshot of "Scratch"

[6] https://grasshopper.app/
[7] https://scratch.mit.edu/

2.3.2 Comparison between products

Code side-panel

Every technology shown in the previous section has a side panel. Some of them contain pieces of code to drag and drop, like “Human Resource Machine” and “Scratch”, while others feature an implemented IDE like “Robo Instructus” and “Grasshopper”. This last implementation resembles our idealization of the project and served as an inspiration.

Didactic Software

We would like our software to be, above all, a fun and educative experience just like “Robo Instructus”, “Scratch” and “Grasshopper”.

“Robo Instructus” is a fun game made up of puzzles that start with simple functions (ex.: moving left or right) and gradually get harder (ex.: implementation of for/while loops), just like the teaching of programming.

“Scratch”, as said before, is a coding community for young people whose objective is to teach children, since early the concepts and bases of programming, just like we want to do with novice programmers, that is our main audience.

“Grasshopper” is a didactic software organized in lessons whose aim is to teach people who are interested in programming.

Visual representation

Regarding all the software we analyzed, we can conclude that all have a visual representation of the code produced by the user. Our approach would more likely resemble “Robo Instructus”' one which consists in compiling the code in the side panel and, in case of success, the character, and the environment react according to the code.

Story-telling & Achievements

Just like “Robo Intructus”, “while True: learn()” and “Human Resource Machine” our intention is to tell a story so that the gameplay is more fluid and has a concrete goal.

Also, we had the concept of implementing achievements in order to keep our users more motivated while solving the coding puzzles, the games above also comprehend this feature. New achievements are unlocked when a user successfully completes a challenge.

3 Requirements

3.1 Functional requirements

In this section we will present the product features and functions that we must be able to implement to enable our users to accomplish their tasks and goals while using our system. The following list describes the functional requirements of our application:

FR #1: Our users should learn programming in a simple, visual, and fun way, practicing some of the programming concepts through a game.

FR #2: Each user must be able to create an account.

FR #3: The user must be able to select a programming language and a difficulty level - beginner, normal, or expert.

FR #4: The game must be divided into several levels. Each level must have an objective: helping the main character to pick up an object, helping to solve a puzzle, reaching from position A to position B, etc. For this objective to be completed, the player has to solve a challenge by writing some code.

FR #5: Levels can be linear or in episodes. That is, the levels can tell a story that evolves as they progress, or each one of them can have an isolated story and/or objective. However, these levels will increase in difficulty as our users progress through the game, starting with basic challenges (i.e., operators and conditional structures) and moving to more advanced programming techniques (i.e., functions and classes).

FR #6: Documentation should be provided for each level. The provided documentation should focus on the functionalities that are intended to be applied in that specific challenge, so users can learn all the necessary concepts on their own and in their own time.

FR #7: Leaderboards for each level must be implemented in order to promote competition between users. The leaderboards will be ranked by the score obtained at each level. This score will be assigned taking into account certain parameters, such as efficiency, use of internal programming language functions, etc.

FR #8: Each user should have an achievements section where they can consult the medals they have won throughout their journey. These medals will attest that the player has, in fact, completed the challenges (and therefore learned something from the game). Can be considered a kind of certificate.

FR #9: The web app should contain a section where the users can check other users' solutions as well as their obtained scores.

FR #10: The embedded code editor should provide a simple code completion tool in order to help users to write code faster and with minimal mistakes.

FR #11: *(Optional)* A way to allow a community to create its own levels could be created - this opens doors to a more personalized use in classrooms, for example.

3.2 Non-Functional requirements

In this section we will be discussing the non-functional requirements, i.e., how the system should perform its tasks. These requirements are very important since we don't only need to define what tasks will be implemented but also how they will be implemented.

NFR #1: Each user's password should be stored securely - using a strong hashing algorithm (like Bcrypt).

NFR #2: The environment where the code written by each user will be executed and evaluated should be completely isolated and disposable.

NFR #3: The system should be able to provide good feedback about errors in the user code. That is, the user should be able to find out where he made mistakes easily.

NFR #4: The system should quickly provide the user with a visual response in order to keep user engagement high.

3.3 Personas/Actors

		
Nolan Bushnell	**Carol Shaw**	**Dennis Ritchie**
• 17 years old; • Recently graduated high school; • Wants to be a software engineer; • Has no experience programming; • Wants to learn the basics before going to college; • Doesn't like to watch tutorials and is searching for a different approach to learning how to code.	• 18 years old; • In 1st year of Software Engineering course; • Wants to be a successful Software Engineer; • Is facing difficulties learning some of the most advanced programming principles; • Wishes to be able to see what her code is doing in an interactive way.	• 48 years old; • Computer Science and Engineering professor; • Wants to teach in a more modern and interesting way; • Is struggling to keep his students interested in the classes; • Would like to change his teaching ways by finding and using something more modern that generates interest, healthy competitiveness, and curiosity amongst his students.

3.4 Use Cases

#	Use Case	Description
1.1	Create a new account	A user wants to register in our system. They fill in the required data on the sign-in page and click the “Sign Up” button.
1.2	Start a new game	A user intends to start a new game. They select the difficulty level and the programming language and proceed to click on the “Play” button.
1.3	Play levels	A user wants to play a level. They select the chapter and the level, confirm their selection and start coding in order to solve a given programming challenge.
1.4	See achievements	A user wants to see their achievements. Clicks on the "Achievements" button from the main menu to access them.
1.5	Change personal information	A user wants to change their personal information. The user selects the “Account” button, changes the necessary information, and finally saves it.
1.6	Check records and scores	A user wants to know the records and scores of other users. So, they select "Leaderboard" on the main menu and apply the filters that they want in their search.
1.7	Check progress	A user wants to see their progress and other users’ progress. The user presses his own “Account” button (or the “Account” button on other users’ profiles) and clicks on “Progress”.

4 Project Calendar

4.1 Agile Method

We've decided to take an agile approach for software development due to multiple factors. Besides being a methodology we've applied in previous projects, Agile proves to be an excellent solution for managing a medium-sized project like ours, not to mention within the timespan we have. We create value through multiple weekly iterations.

After all, Agile values different ideas that favor constant collaboration and continuous improvement at every stage, such as:

- Individual interactions over process and tools
- Customer collaboration over contract negotiation
- Working software over comprehensive documentation
- Responding to change over following a plan

4.2 Calendar

We first designed a calendar taking into account due dates (for example, Students@DETI event), milestones and deliverables. However, our calendar has gone through multiple changes due to revisions made at the end and/or beginning of iterations. Our process of revision would consider time adjustments, team needs, and even changes to requirements (another main quality of Agile, rather than pursuing "rigid" requirements).

We can find our project calendar at the following link: Project Calendar.

4.3 Gantt Diagram

We then designed a Gantt diagram with the main tasks described previously (in the calendar). This helps us plan our work around deadlines and milestones more efficiently, at the same time we're able to define the start and end dates of tasks and the dependencies between tasks.

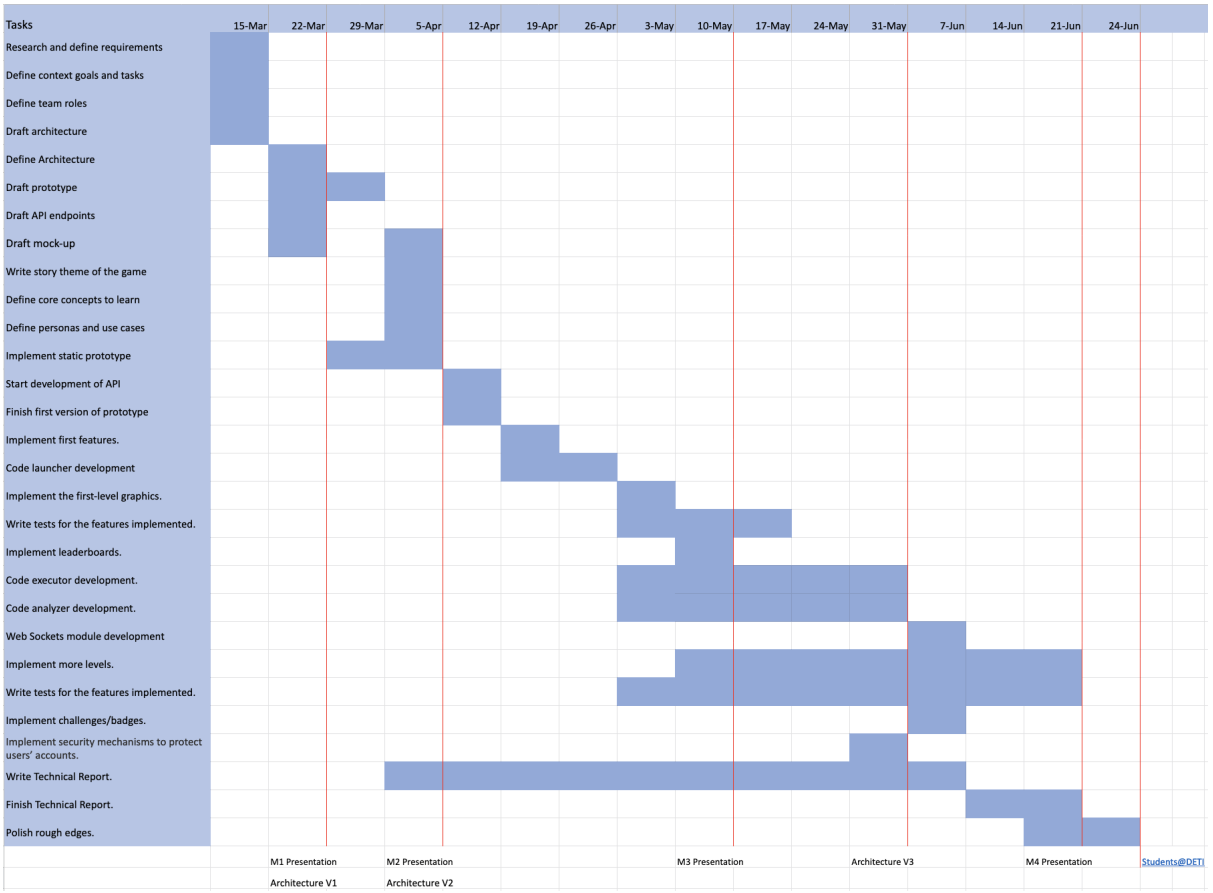


Fig. 6: Gantt Diagram for main tasks.

4.4 Tasks List

We also wrote a list of the main tasks organized by modules and assigned each task to different team members.

Module: Prototype

Task. #	Description	Dependencies	Responsible
P1	Implement visual prototype		Daniela
P2	Implement static frontend	P1	Daniela
P3	Implement main features	P1	All
P4	Finish up prototype	P2, P3	All

Module: Learning

Task. #	Description	Dependencies	Responsible
L1	Study and select basic programming concepts to teach (for example, by gathering material from other courses)		All
L2	Implement code documentation	L1	All
L3	Write simple, concise questions	L1	All
L4	Implement questions according to project context	L3, L2	All

Module: Code Interpretation

Task. #	Description	Dependencies	Responsible
CI1	Study different solutions of interpreters/compilers for web pages, along with mechanisms to embed them		Hugo
CI2	Study different solutions of code editors for the execution of code in web pages		Daniela, Paulo
CI3	Select mechanism for code interpretation and document decisions/justifications	CI1, CI2	All
CI4	Optional: Implement code interpretation for Python	CI3	---
CI5	Implement code interpretation for Java	CI3	Hugo

CI6	Optional: Implement code interpretation for Javascript	CI3	---

Module: Game Design / Levels Design

Task. #	Description	Dependencies	Responsible
GD1	Define and write the focal story/theme, along with its characters for each programming language and/or level		All
GD2	Illustrate world and characters	GD1	Diogo
GD3	Define character behavior according to the level and written code (output)	GD2	All
GD4	Implement graphics and animations	GD2	Diogo
GD5	Implement fully functional levels	GD3, GD4	Everyone

Module: Security

Task. #	Description	Dependencies	Responsible
S1	Select the main security mechanisms to implement		All
S2	Implement basic Sign up/Login pages with secure password restrictions		Hugo
S3	Implement database with encryption for sensitive data (e.g. passwords)	S2	Hugo
S4	Implement user sessions	S2	
S5	Implement advanced security features	S1, S3, S4	---
S6	Fix potential vulnerabilities	S5	All

4.5 Milestones, Deliverables

M1 Presentation of the life cycle objectives and calendar for the project.
Deliverables:

- PowerPoint presentation.

M2 Presentation of the lifecycle architecture. Validate architecture.
Deliverables:

- Visual prototype.
- Product prototype (static frontend).

M3 Prototype. Mid-term presentation with supervisors. Peer evaluation.
Deliverables:

- Product increment covering (at least) 1 level.
- Product increment covering graphics and leaderboards.

M4 Project presentation. All functionality has been developed!
Deliverables:

- Product increment covering (at least) 1 level.
- Product increment covering (at least) 1 level.
- Demo + poster for students@deti & video.
- Technical Report.
- Final Product.

5 Architecture

One of the main challenges of our project is its architecture. First, we need to design a robust system that will execute the users' code and return the results for each level. Secondly, and probably most important, is the security behind this code execution. Once we are executing all the code written by our users, i.e - handling remote code execution - we need to execute and analyze it in a secure environment that will not interfere in any way with the existing core modules. This is one of the main concerns that will have influenced our architecture and its choices and will be one of our main focuses while developing our project.

The second great challenge we face is how to execute the code at all and provide a certain smart analysis of the code, not allowing the users to cheat by printing the expected results, for example. To do this, we cannot only trust printed results to the console and analyze them to check if the code is complying with the levels' constraints - we need to go further.

Lastly, but not the least, we need to find a way to convert the code that is written by the user into actions in the graphical part of our system, i.e., convert the "fors" and "ifs" written by the user into something he can see, using the graphics layer, and verify that is working, or not. We also need to provide the errors and possible tips that can help the user to understand what's wrong with his code and what needs/can be improved.

5.1 Architecture V1

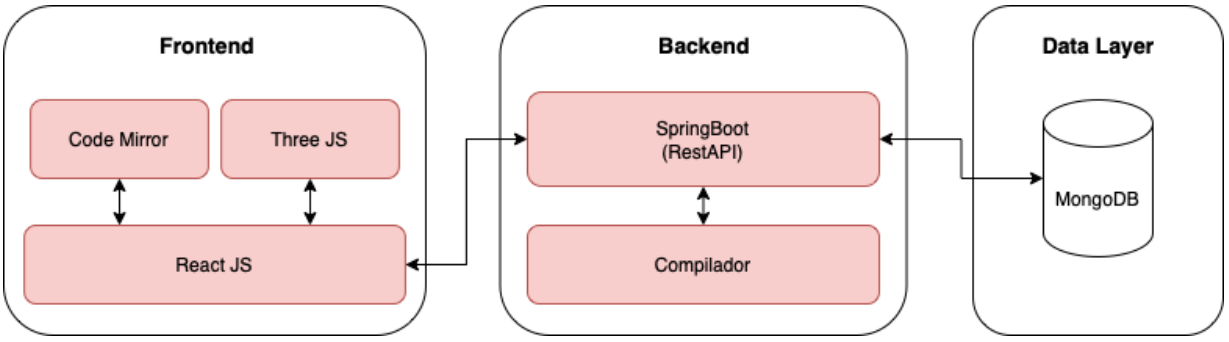


Fig. 7: Diagram for Architecture V1.

This is the first draft of our architecture and its main purpose is to enlighten us about what are the main top-level modules of our system and what role they have in our system. Therefore, in this version, we have three main components: the **Frontend**, **Backend,** and **Data Layer**, each one of them with its submodules.

5.1.1 Frontend

The Frontend layer will be responsible for performing all the interactions with the user and will be implemented using a SPA framework - ***ReactJS***[8] backed up by RestAPI in the backend.

One of the most important components of our application is the capability of allowing our users to write code and submit it as solutions to the levels. To allow our users to write code in our web application and provide some kind of code analysis in real-time, we will use ***CodeMirror***[9] - a super versatile text editor implemented in JavaScript that can easily run on almost all browsers available. It is specialized for editing code and comes with a number of language modes and addons that implement more advanced editing functionality.

The second important component of our front end is the graphical section. We need a way to quickly render the results of the code in a visual way. In order to do this, we will be using ***ThreeJS***[10] - a cross-browser JavaScript library and API used to create and display animated 3D computer graphics in a web browser using WebGL.

5.1.2 Backend

The ***Backend*** will be composed of a RestAPI (using SpringBoot[11]) that will process the Frontend features and a Compiler unit that will be responsible for compiling the code written by the users (in a secure way) and returning the results to the API so it can return them to the Frontend.

5.1.3 Data Layer

In the ***Data Layer*** we will have a MongoDB[12] database that will store the levels information, users' accounts, users' progress, etc. We have chosen MongoDB instead of a relational database because of its flexibility and scalability. Once our levels may have properties that are different from each other, this flexibility is very important as it allows us to keep storing the levels in the same collection although they may have some different fields.

[8] https://reactjs.org/
[9] https://codemirror.net/
[10] https://threejs.org/
[11] https://spring.io/projects/spring-boot
[12] https://www.mongodb.com/

5.2 Architecture V2

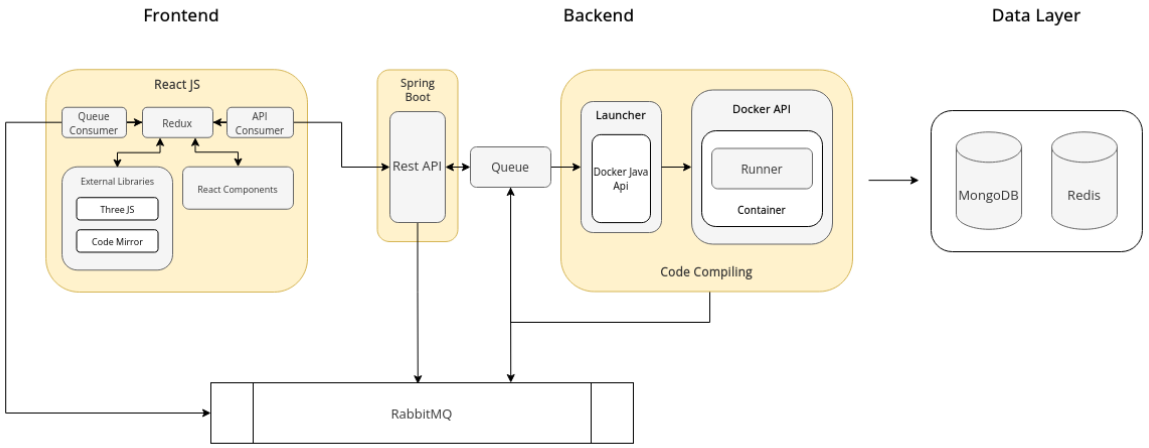


Fig. 8: Diagram for Architecture V2.

While the Architecture V1 was only a draft and contained nothing more than a high-level view of the different high-level components of the system, this version contains more details about the different components and about how will they interact with each other.

We still have three different main components, the Frontend, Backend, and Data Layer, however, compared to architecture V1 (the latter version), we have introduced more details and segregated the responsibilities of each module into different submodules. We also introduced a queue system that will be used for Event Notification (an Event-Driven pattern) between the Frontend, API, and Code Compiler.

This version of the architecture can also be classified into 3 different modules - **Frontend**, **Backend,** and **Data Layer** - although their boundaries are now less significant and start to merge at some points.

5.2.1 Frontend

We will keep using ThreeJS and CodeMirror to render the graphics and handle the input of code by the user respectively. To handle the React components' life cycle and update these components smoothly, we will be using ***Redux***[13] - an open-source JavaScript library for managing and centralizing the application state.

We will also have an ***API Consumer***, that will consume the RestAPI in the backend and handle all the data returned by it, and a ***Queue Consumer*** that will be responsible to get all the events sent by the remaining components of the architecture and act accordingly by performing some request to the API, displaying a message or updating some component.

[13] https://react-redux.js.org/

5.2.2 Backend

In the Backend, we still have a RestAPI that will be responsible for most of the operations (authentication, authorization, user account operations, etc.). This API will be connected to our queue system (***RabbitMQ***[14]) in order to receive and send Event Notifications from/to other components of the system.

One of the most important features of our system is the possibility of receiving code and executing it in a secure environment where it can be evaluated according to the respective level requirements. The biggest challenge here is to execute the code in a secure and scalable way in order to protect our system against possible attacks via remote code execution. This challenge will be achievable using a ***Launcher*** system and disposable ***Runners*** that will run the code provided by the users and return the results.

When the Frontend requests a new code execution, the API will post this request as an event to the queue system. This event will be consumed by the Launcher that will launch a new docker container to run the necessary code.

When the code is executed, the runners will post a new event to the queue messaging system and the container will be removed. This event will be received by the API allowing the latter to answer to the front. Most of the details about the results of the code execution will be stored in a ***Redis***[15] instance of the Data Layer. Those details can also be read by the API and returned in the response to the frontend request.

5.2.3 Data Layer

We will keep using MongoDB as our primary database but we will complement our data system with a ***Redis*** database. As pointed out above, the main purpose of this database will be to store, temporarily, the data returned by the compiler. Most of this data is only used once - when returned to the frontend as a response to the code execution request and can be deleted afterward, i.e., Redis will work as a cache to hold temporary data.

[14] https://www.rabbitmq.com/
[15] https://redis.io/

5.3 Architecture V3

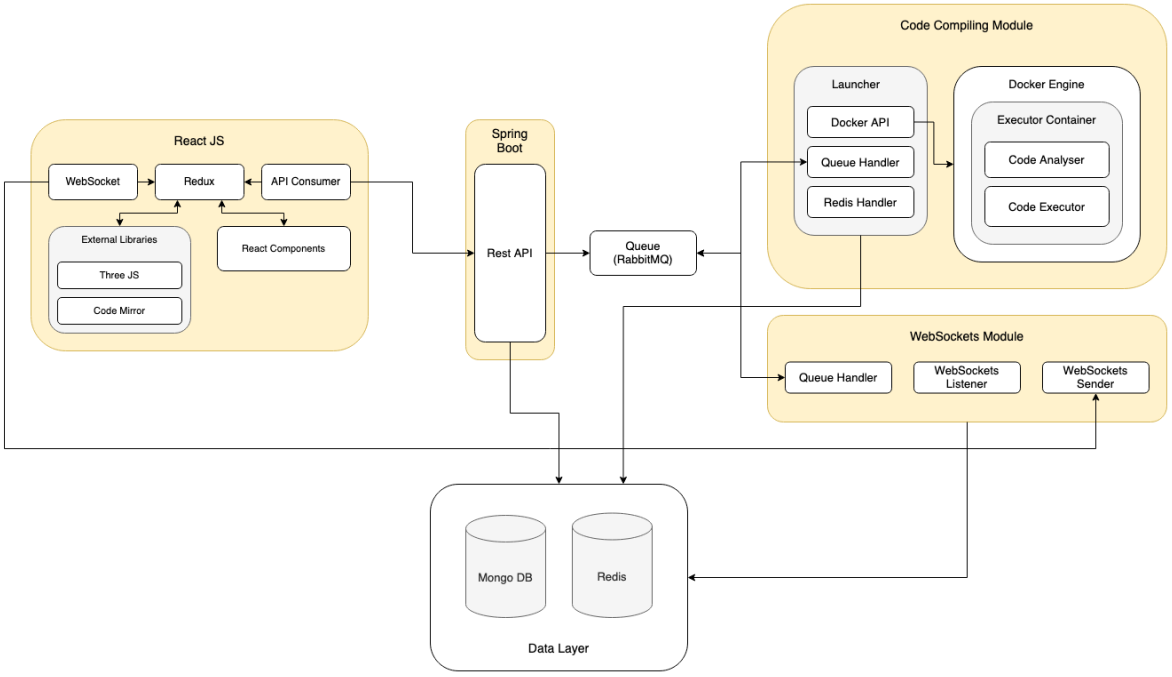


Fig. 9: Diagram for Architecture V1.

The Architecture V3 is the final architecture of our system, and it brought some expressive changes in the backend/code compiling unit. Firstly, we have decided to remove the direct connection between RabbitMQ and our Frontend.

One of the main motivations to change our architecture from V2 to V3 was that we needed to have our Frontend website (ReactJS) communicating with the Backend and the Compiling units using WebSockets, since we wanted to "push" the results of the code execution to the Frontend. In version 2 of our architecture, we decided to do it using RabbitMQ using its Web STOMP plugin[16]. This plugin allows the JavaScript WebSockets to subscribe to RabbitMQ queues and receive messages through WebSockets.

Although this seemed a great idea, and also very simple to implement, we faced some issues with the RabbitMQ acknowledge system as, once the messages would be read by one client (one instance of our website), those messages would be marked as acknowledged and deleted from the queue, not allowing the other connected clients to receive the messages too. The solution was to develop an entire WebSockets module that would have the task of communicating to all connected clients the necessary messages. This module was developed with Spring Boot and its built-in WebSockets library.

One other expressive change in architecture V3 was the segregation of concerns in the executor container (called "Runner" in architecture V2). After some brainstorming, we found out that we would need not one single module - that would do the syntactic analysis and the execution of the code - but two different modules, the **Code Executor** and the **Code**

[16] https://www.rabbitmq.com/web-stomp.html

Analyser. The main reason that led us to create different modules is that the code execution is independent of the code analysis, and if the code analysis fails, the code doesn't need to be executed at all. By having two different modules and running the code analysis before the code execution, we are saving a lot of time and resources wasted trying to run a code with syntactic errors that would not be compiled at all.

In this new version of the architecture and the system in general, we also started using the Redis database to store, temporarily, the **code execution reports**. The code execution reports are the report generated for each code when it is run.

5.3.1 Code Compilation and Execution

One of the main features of our system is its capability to run the code written by the users in a secure environment and return the results. As this was one of the features that took us the most of the time to implement, this certainly deserves special attention and explanation.

Code execution is the "invisible" part of our system. When the user writes some code on our website his code is sent to our backend using an endpoint in our **Rest API** (Submit Code Solution[17]). The code sent through this endpoint is then posted to RabbitMQ and consumed by our **Code Launcher** and a new code execution request is created. The **Code Launcher** will create a new **Disposable Docker Container** using the **Docker API** with all the support projects (**Code Executor** and **Code Analyser** inside of it). That container will then perform the syntactic analysis of the code - using the **Code Analyser** module - and if it passes, i.e., if the code has no syntactic errors, then the code will be executed by the **Code Executor**.

The **Code Executor** module will execute the code and perform an analysis of the code to make sure it complies with the defined constraints. This analysis is performed using an **Abstract Syntax Tree** and it allows us to check, dynamically, how many structures the code contains, how many variables the code contains, the name and type of the variables, etc. This analysis is done using the Java internal classes for annotation processing.

The Code Executor is executed within a timeout of 30 seconds, which means that if the execution of the code does not finish within 30 seconds then the process will be killed. This is extremely important to avoid problems with infinite loops created by a user intentionally or non intentionally as referred to in ("Building a remote code execution system. | by Yash Budukh" 2020) [2].

Our Disposable Containers are also limited in the amount of CPU and memory they can use to prevent issues with remote code execution.

All the results provided by these two modules - Code Analyser and Code Executor - will be saved to files inside the Disposable Docker Container. To prevent security problems, this disposable container is not connected to any of our main modules, nor to any of our

[17] https://documenter.getpostman.com/view/16743908/UVyswaxA

queues (RabbitMQ) or databases. So, in order to retrieve the results of the code execution from this Disposable Container, our **Code Launcher** will “pull” the files from the container using the Docker API. These files - result files - are then sent through the network and stored in the Code Launcher’s container where they can be read and interpreted.

Once all the result files were pulled and interpreted by the **Code Launcher**, a new **Code Execution Report** is generated, taking into account the results provided by the **Code Executor’s** files, and stored inside the **Redis** database, where these execution reports will remain stored for a short period of time as proposed in (“Building a remote code execution system. | by Yash Budukh” 2020) [2]. This report is generated for each code execution request and generally, it will look like this for a code that could not be compiled due to a syntactic error:

```
{"id":"5407eda5-7fb7-4a8f-835c-879d187c71a2","analysisStatus":"COMPILATION_ERROR","
executionStatus":null,"steps":null,"score":0,"output":null,"errors":["Error on Line 52…"]}
```

And should look like this, to a code that was successfully compiled:

```
{"id":"185c0a75-fb41-406f-90d2-c6556b3e51bb","analysisStatus":"SUCCESS","executionStatus":"
SUCCESS","steps":[{"id":1,"successful":true,"args":null},{"id":2,"successful":true,"args":n
    ull},{"id":3,"successful":true,"args":["Hello World!"]}],"score":0,"output":["Hello
                                    World!"],"errors":null}
```

A message will also be sent by the **Code Launcher** to the RabbitMQ to warn the **WebSockets Module** about a finished code execution and allow it to pass that information to the Frontend. This message will generally look like this:

```
{"codeUniqueId":"185c0a75-fb41-406f-90d2-c6556b3e51bb","codeExecutionStatus":"RAN"}
```

Once the Frontend receives the message sent by the **WebSockets Module**, it will convert the code execution to some action in the graphic part. It will also provide feedback about errors and provide tips related to the code execution itself.

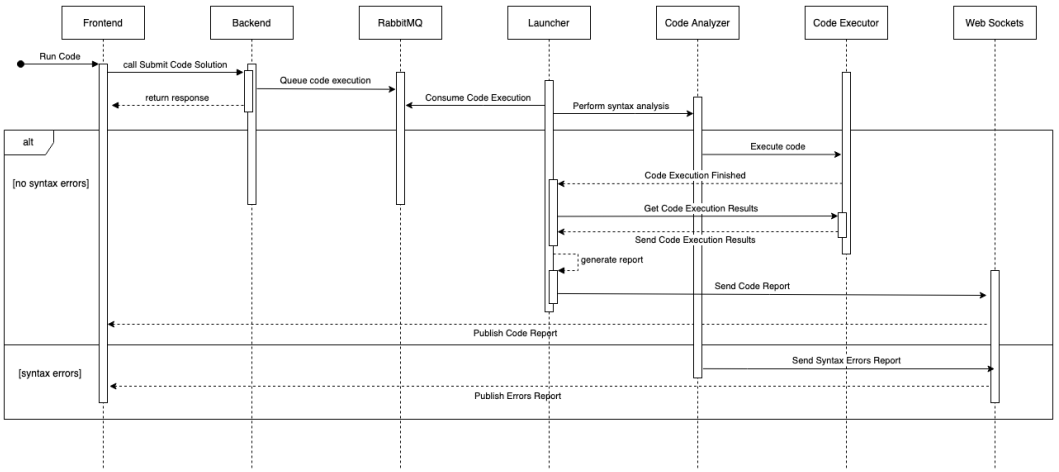


Diagram 1: Sequence diagram for code execution process.

5.3.2 Authentication and authorization process

When it comes to authorization, we decided to use JWTs[18] (JSON Web Tokens). It's a type of authorization token that contains in its generation the claimant's information. Besides granting a higher level of security, by authenticating the users, the server can use this token to identify the user that made the request and therefore answer with the information for that user.

These tokens can also be used for authorization mechanisms as, once we identify the user who made the request, we can easily figure out if the user has access to a certain use case.

In terms of authentication, a user must first register on the platform. After that, they can use the login page with the correct credentials. When they make the login, a new JWT token is generated according to the user's details and a response is sent. Upon receiving the response, the client will then save his token in local storage and the token will be further used in the authorization of requests.

The users' passwords are also stored securely in our databases using BCrypt. This grants our users a higher level of security and confidence in our system.

[18] https://jwt.io/

6 Technology Stack

6.1 ThreeJS

In order to fulfill one of our Functional Requirements (our users should learn programming in a simple, visual, and fun way [...] FR #1), we created something resembling a game that is playable by completing coding challenges. In order to create this graphical interface inside the client browser, we need to use WebGL. WebGL is a javascript API that offers support to the rendering of 2D and 3D objects inside the supported browsers.

However, WebGL is a low-level API, so in order to easily reach our goal, we are going to use some technology over WebGL that eases the development of a 3D environment. We've considered two different technologies that could fulfill our needs: ThreeJS and Unity[19].

Unity is a widely known framework used to build games for Desktop, Mobile, and even Web environments as it allows us to export our game project to WebGL. By exporting our project to WebGL, we can make it run inside any browser that is compatible with WebGL. Unity has a lot of advantages such as a huge community, support, and tutorials, however, once it has no native support for Javascript, it would be somehow hard to integrate our game with ReactJS (our frontend technology).

Once ThreeJS is written in JavaScript, it provides better support and integration with other technologies also using this language. Besides that, some of our team members know how to develop using ThreeJS while none of us has any kind of experience with Unity. This is the main reason why we chose ThreeJS.

6.2 React JS and Redux

ReactJS was our first choice for Frontend development as all the members of the team have worked with it in the past and had some previous experience which would make the development a little bit easier. Besides that, it has great support for ThreeJS and other third-party libraries which came in handy for our use case.

We've also adopted Redux to decouple the global state of our components and create a reactive application that would respond to the changes. In other words, Redux enables the communication and sharing of data across components, while creating a data structure to represent the state of our app (that we can read from and write to).

This open-source JavaScript library has proven to be highly useful for our application by solving the following problems:

[19] https://unity.com/

- We have large amounts of application states that are needed in many places in the app (for example, with game progress, programming language, and difficulty).
- The app state is updated frequently (for example, on the Level page, with repeated code submissions).
- We need to see how that state is being updated over time (for example, with game progress).

In this context, we have created a single store that holds the application state, allowing it to be accessed and updated. Here, we also add the reducer functions we wrote for the many contexts of our application.

We wrote reducer functions (pure functions that take the current state of an application, perform an action, and return a new state) to deal with data and actions concerning:

Settings
The user can read the current settings (programming language, difficulty) and update them. The programming language and the difficulty are needed in many components.

Progress
The user can read their current progress (in the different programming languages and difficulties) and update it. The progress is needed on the Account page and on the Level page.

Chapters and levels
The user can read the available chapters and their respective levels, and update them. The chapters and their levels are needed in various components, such as on the Achievements page and on the Levels page.

Code
The Code Execution reports that are received through the WebSockets are managed by Redux and used in several components such as the Level component.

At the same time, since we're dealing with web sockets in specific parts of our application (Level page, by submitting code and receiving the results asynchronously), we found the need for implementing a thunk middleware. "Thunk" means "a piece of code that does some delayed work", rather than executing some logic immediately. Middlewares, on the other hand, are what intercept all actions dispatched from components before they are passed to the reducer function.

To write this async logic, we use Redux Toolkit's configureStore API, which already adds the thunk middleware by default.

6.3 Spring Boot

Many of our system's modules were developed in Java using the Spring Boot framework. Although our system follows a decoupled and stateless fashion, allowing us to scale it up if needed by deploying new modules like Code Launchers, WebSockets module, etc, we used Java language in the vast majority of our backend systems to ensure the compatibility between all the modules and allow us to reuse some of the code from some modules to the others.

Since the time to develop the entire solution is somehow scarce, Spring Boot helped us and allowed us to focus on the more important tasks by removing a lot of the boilerplate code that usually is necessary to just set up the environment.

Spring Boot also provides us with a lot of dependencies that were used by us to perform communication with the databases, and communication with the queue system (RabbitMQ) and was certainly useful to easily create a WebSocket server that would serve our clients (Frontend).

6.4 Traefik

Our system is composed of a lot of different services. However, each one of those services is sharing the machine with the other ones, so, in order to create "subdomains" that could redirect the traffic to the correct services - without using ports - we needed to implement a reverse proxy technology.

After searching for technologies that could perform this job for us we got between two of them: Nginx and Traefik. We ended up choosing the latter for its simplicity and great integration with docker containers as well as a lot of other useful features like a monitor, an automatic services discovery mechanism, etc.

To set up Traefik in our VM, we have followed the tutorial available in (Thompson 2020) [4]. Traefik was also used to secure the connection between our modules and our clients with HTTPS. The CA that emitted the certificate we are using was Let's Encrypt[20]. After all the containers were configured to be reached by Traefik (as shown in the tutorial), we registered the DNS subdomains for each one of them using the DNS record editor of DigitalOcean[21] in order to create the subdomains and point them to the correct IP address.

[20] https://letsencrypt.org/
[21] https://cloud.digitalocean.com

7 Information perspective

In order to illustrate and explain the concepts (and their attributes) of our domain, CodeSpell, we can use a domain model. A domain model is a visual representation of conceptual classes or real-situation objects in a domain, along with their different relationships.

In our domain, we use the following entities to represent our data:

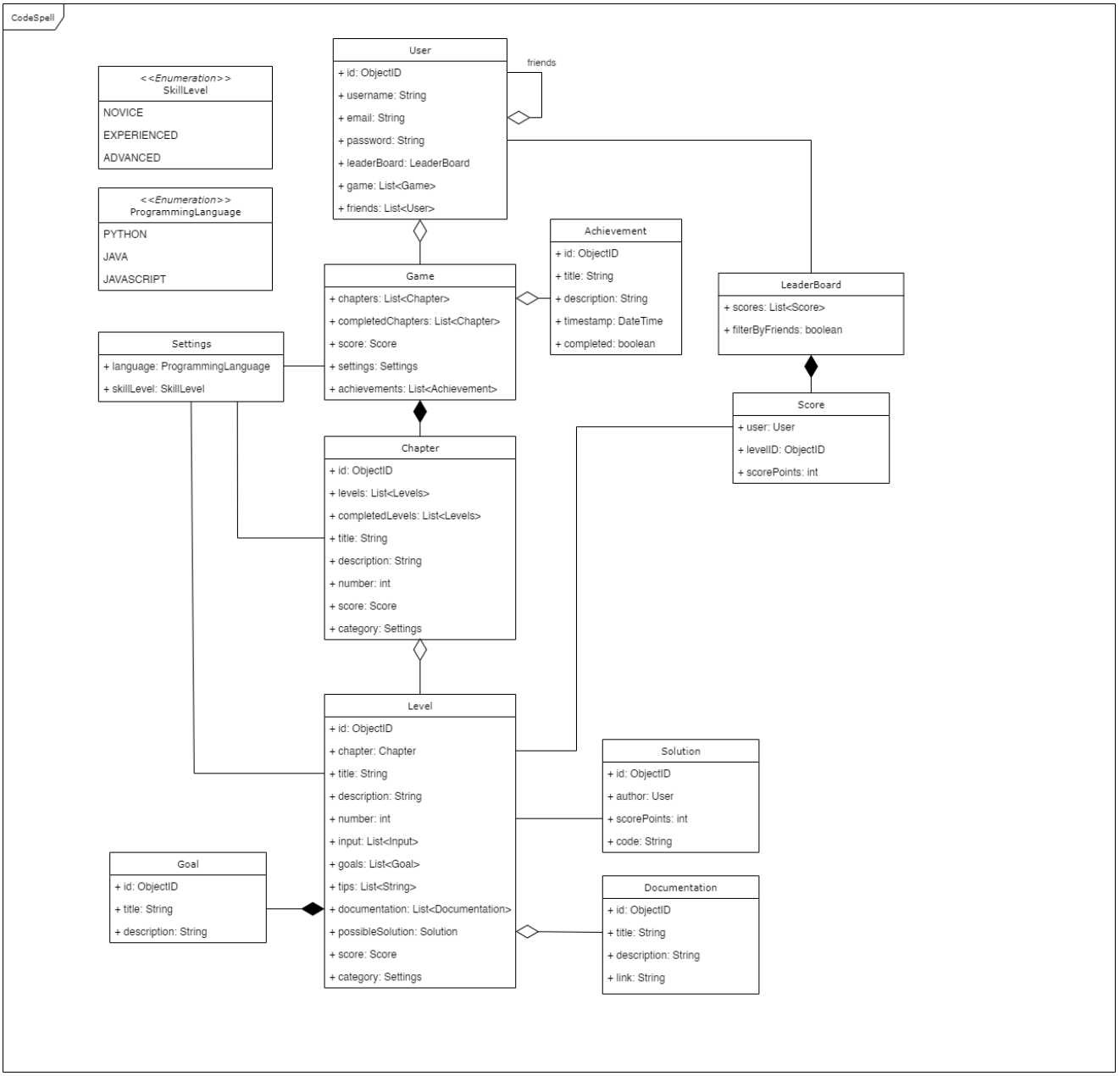


Diagram 2: Domain model diagram.

7.1 User

The User is the entity that interacts with the application CodeSpell. This entity will contain some basic information about our users: username, email, and more. Both the username and email are unique fields.

7.2 Leaderboard

This entity stores all of the scores from every user registered in the application. Where the user can apply filters to obtain the best scores for certain chapters, levels, and programming languages.

7.3 Score

This represents how much progress the user has made throughout the game and, at the same time, provides a measurement/evaluation of how good the user's solutions are. Scores have different categories and can be accessed from the leaderboard or from inside the game.

7.4 Achievement

This is another measurement of the user's progress, representing certain checkpoints in the learning process, accomplishments, and milestones met by the user.

7.5 Game

We decided to develop this entity to allow the user to create different game sessions for a single account, just like we're used to seeing in every video game, *i.e*, for each *Game* associated with a user's account, there might be different settings, chapters, achievements, and scores.

7.6 Settings

The Settings entity has two attributes, *ProgrammingLanguage* and *SkillLevel*. With this feature, the user is able to select both the desired programming language and skill level, the latter has three possibilities: *Novice*, for a beginner programmer; *Experienced*, for a programmer that is familiar with general coding practices; and *Advanced,* for programmers that possess more complex knowledge and that feel comfortable with the selected programming language.

This was our initial thought on the settings, although, due to the lack of time, the implementation of *ProgrammingLanguage and SkillLevel* did not reach our expectations.

7.7 Chapter

This entity is pretty much a collection of Levels. Each *Chapter* is focused on a programming topic, divided into subtopics which are the levels. This entity contains a list of levels and a list of already completed levels.

7.8 Level

Each level is focused on teaching a programming subtopic (ex.: Variables, Classes, Conditions, Branching Statements, etc). It contains a list of steps provided to the user so that he knows what to code and what the goal of the level is. It also has a list of tips, documentation, as well as error output.

7.9 Goal

This entity was created to explain the finality of the level and the code to be implemented by the user, simply containing a title and a description.

7.10 Documentation

This entity is highly important for solving levels by providing means to learn how to implement blocks of code necessary to solve the given challenges, along with new programming concepts.

7.11 Solution

This entity represents the solutions submitted by users when solving levels. It is identified by the user who wrote the code, the score obtained, and the code itself.

8 Game Specification

8.1 Concept

The aim of CodeSpell is to provide a reactive graphic platform for programming teaching and learning. By exploring basic programming concepts, we allow our users (students, beginners at programming/coding) to visually study the effect of different programming mechanisms.

In order to make the experience more fun, while motivating the user to keep learning, we created a story-based game with different linear and/or episodic levels.

8.2 Story and Characters

We're set in a fantasy world. Following us through this journey, we have one of our little cube knights, "Java", to fight against confusing tasks and problems.

Monsters stole many pages from our spellbook and, without them, we've been struggling to navigate the world. Our main objective is to retrieve all the pages back from the monsters, making the book, once again, complete. We find pages along the way (throughout the levels, with the documentation), which help us learn and relearn spells that are critical to overcoming many difficult situations.

8.3 Game Structure

The game is made of an extensive list of chapters, which aggregate the levels by similar content, techniques, and stage of learning. By organizing the learning process by these ordered "steps", the player is able to learn faster, easier, and more comfortably. At the same time, they're able to keep track of their progress and revisit old concepts.

Each level has its own setup, world, and objectives: the user must interact with distinct elements and tools, within the given context, by writing code and solving problems. These levels will increase in difficulty as our users progress through the game, starting with basic challenges like operators and conditional structures and moving to more advanced programming techniques like functions and classes.

The player can check the current pages of the spellbook in the section "Documentation". As they progress through the game and gain access to new levels, the documentation gets more complete.

8.4 Levels

8.4.1 Java

During the elaboration and inception phases we have designed several levels that would be implemented. However, as already discussed, and as discussed in more details on Section 9 - Results and Discussion, only two of these levels could be indeed implemented.

Below is the design made for these two levels. Furthermore, the design for the remaining levels, that unfortunately could not be implemented due to a lack of time, can be found in a separate document.

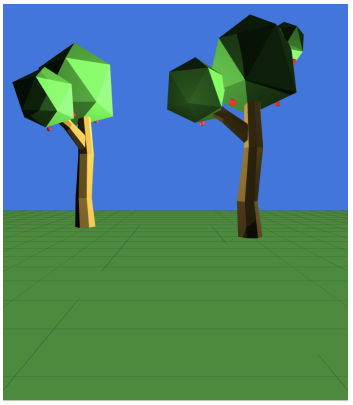
Chapter: Introduction

Lv.1 Hello World

Description	It's time to write your first application and become acquainted with the main components of our language Java! Your application, HelloWorldApp, will simply display the greeting "Hello World!".
Visual World	• Main character
External Documentation	https://docs.oracle.com/javase/tutorial/getStarted/application/index.html
Code provided	---

Step 1

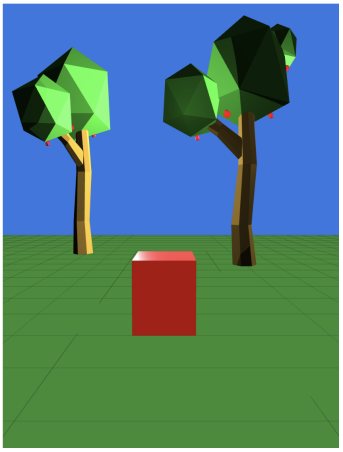
Goal	Define a class with the name “HelloWorldApp”.
Output	The world becomes visible, hinting at the fact that every java application begins with a class definition.
Code to write	`class HelloWorldApp {` `}`

Graphic	

Step 2

Goal	Define the main method.
Output	The character becomes visible.

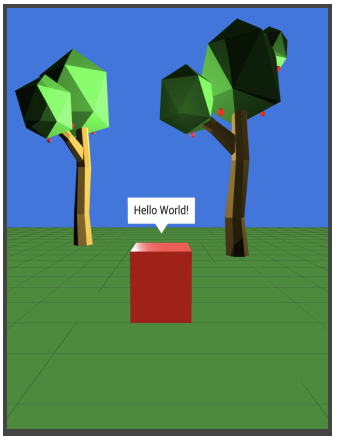
Code to write	`class HelloWorldApp {`
	`  public static void main(String[] args) {`
	
	`  }`
	`}`

Graphic	

Step 3

Goal	Print "Hello World!" to standard output.
Output	The character has a speech bubble with the text provided in print(). The user has the freedom to try other sentences before moving on to the next level (the speech bubble changes accordingly).
Solution	`class HelloWorldApp {` `  public static void main(String[] args) {`

	`System.out.println("Hello World!");`
	`    }`
	`}`

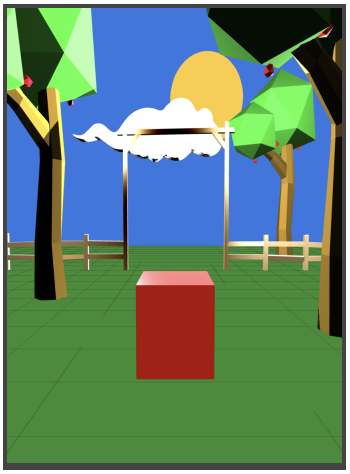
Graphic	

Chapter: Language Basics

Lv. 3 The *if-then* and *if-then-else* Statements

Description	The *if-then* and *if-then-else* statements are the most basic of all the control flow statements. It tells your program to execute a certain section of code only if a particular test evaluates to true. With *else*, we have a secondary path of execution when an *if* clause evaluates to false. Let's control our world and characters with these statements.
Visual world	• Main character • Sun and other daytime elements • Stars and other nighttime elements
External Documentation	• https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html

Code provided	`boolean day = true;`
	`int stars;`
	
	`if (day) {`
	`    stars = 0;`
	`    dayTime();`
	`} else {`
	`    stars = 50;`
	`    nightTime();`
	`}`

Graphic	

Step 1

Goal	Update the value of variable *day* to *false*.
Output	The world changes from daytime to nighttime.
Code to write	`// Step 1` `day = false;`
Graphic	

Step 2

Goal	Write an *if-then-else* statement that evaluates if variable *stars* has a value greater than 40. Print “What a starry sky!” to standard output if true.. Otherwise, print "What a beautiful sky!" to standard output.
Output	The character has a speech bubble with the text provided in print(). The user has the freedom to try other sentences before moving on to the next level (the speech bubble changes accordingly).
Code to	`// Step 1`

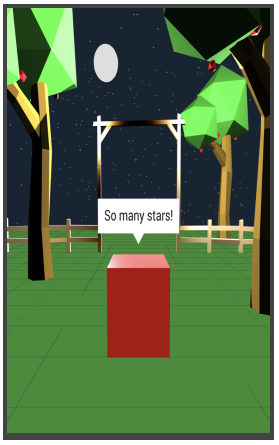
write	`day = false;`
	
	`// Step 2 & 3`
	`if (stars > 40) {`
	`    System.out.println("What a starry sky!");`
	`} else {`
	`    System.out.println("What a beautiful sky!");`
	`}`

Graphic	

Step 3

Goal	Rewrite the previous *if-then-else* statement in order to only print "So many stars!" to standard output if the value of *stars* is equal to 50.
Output	The character has a speech bubble with the text provided in print(). The user has the freedom to try other sentences before moving on to the next level (the speech bubble changes accordingly).

Code to write	`// Step 1`
	`day = false;`
	
	`// Step 2 & 3`
	`if (stars == 50) {`
	`    System.out.println("So many stars!");`
	`} elif (stars > 40) {`
	`    System.out.println("What a starry sky!");`
	`} else {`
	`    System.out.println("What a beautiful sky!");`
	`}`

Graphic	

9 Results and Discussion

9.1 General discussion

Overall, the main objectives and requirements for the project were successfully accomplished. We were able to create a reactive platform that will help new students, in the programming field, to better understand what are the results of their code, by checking what is happening with the help of the game implemented. Unfortunately, some of the requirements could not be finished and were only partially implemented, as discussed further in the topic “Requirements Completion”. Other ones, although fully implemented, could be improved with more time.

One example of an improvement that could be done is reducing the time that takes to compile and execute the code written by the user. Currently, for security reasons, every time a user clicks on the “Run” button, the code is sent to the backend and a new container is created from scratch to run that code. Inside the container, the code is injected inside a Maven project. When the code is successfully injected, that project will be packaged into a *jar* file using the maven package goal. Once the container where this packaging takes place is brand new, i.e., was created only to run that code, it won’t have any maven cache and, therefore, will need to download all the necessary dependencies to run the code. This process can take up to 15 seconds and will delay the code execution for 15 seconds.

This problem could be fixed by creating a new project that would not use the maven and would be compiled manually using the Java Compiler. This would reduce the necessary time to compile the project and, therefore, improve the necessary time to run the code executed by the user.

An example of a feature that was not implemented is the multiple programming languages and difficulties. While in the inception phase of the project we introduced support to several difficulties and languages for each level, we could not keep up with those plans due to a lack of time. However, our project was designed in a modular way, hence, the addition of more languages and difficulties would work as an addon that could be easily added in the future.

9.2 Results discussion

In this section we will show some screenshots of the different pages of our project and explain what is the intended usage and how each one of these features is aggregating value to the project.

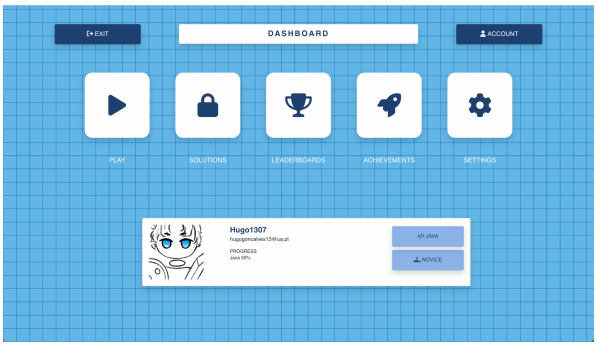


Fig. 10: Dashboard page

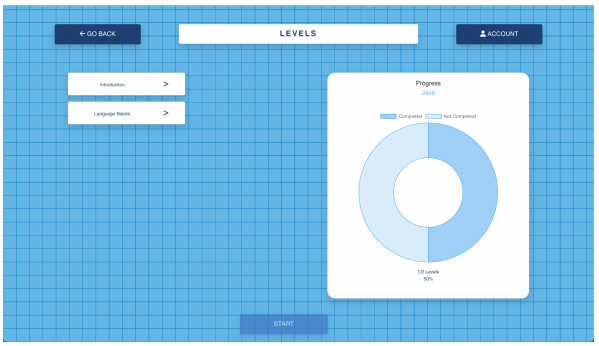


Fig. 11: Level Selection page

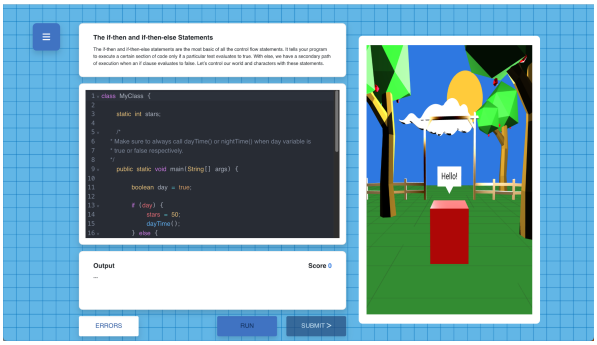


Fig. 12: Level With Graphics layer visible.

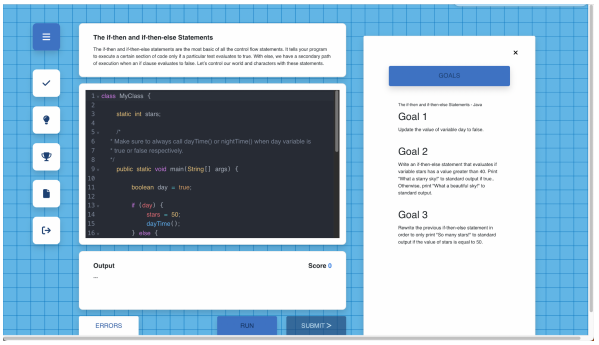


Fig. 13: Level goals pop-up.

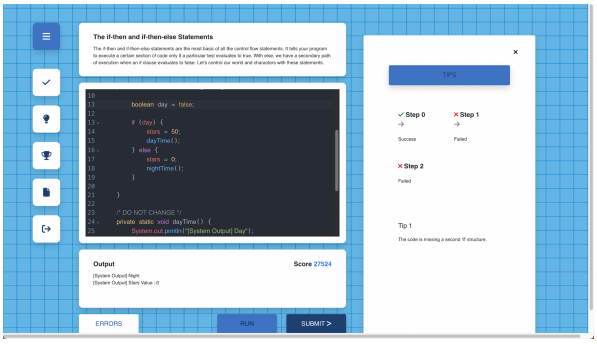


Fig. 14: Tips and steps completion pop-up.

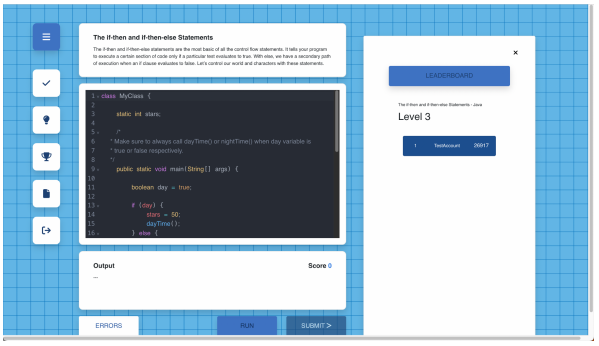


Fig. 15: Level Selection page

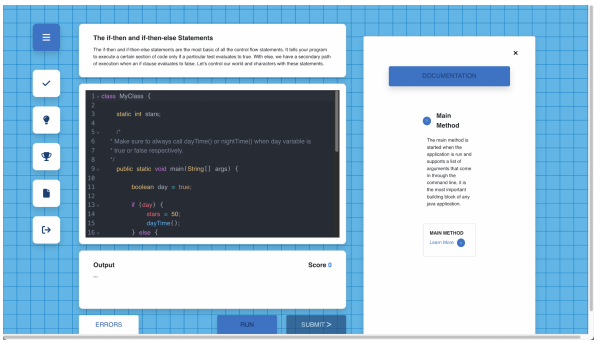


Fig. 16: Level documentation pop-up.

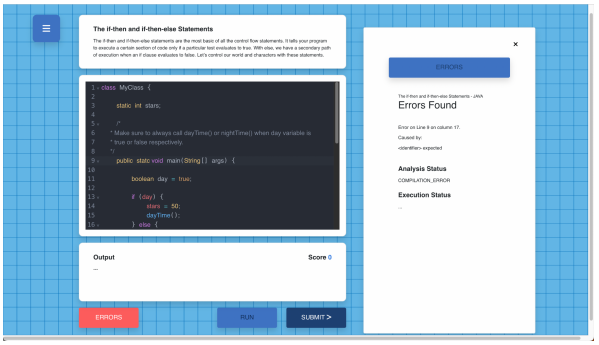


Fig. 17: Syntax errors pop-up.

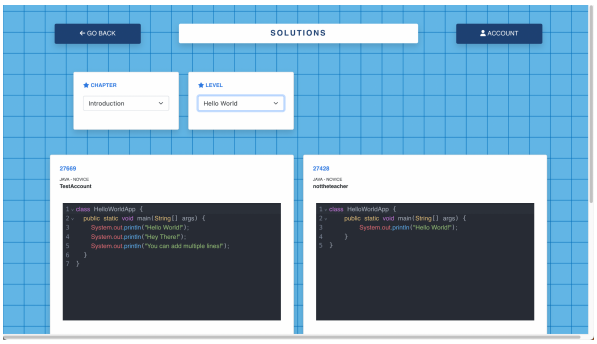


Fig. 18: Level's solutions page.

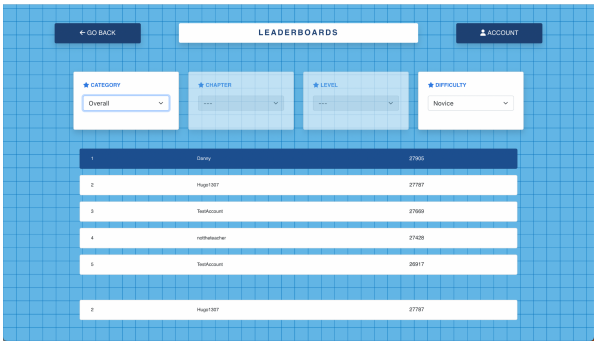


Fig. 19: Leaderboards page.

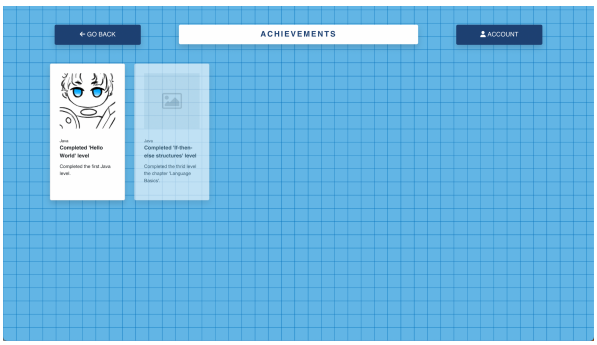


Fig. 20: Achievements page.

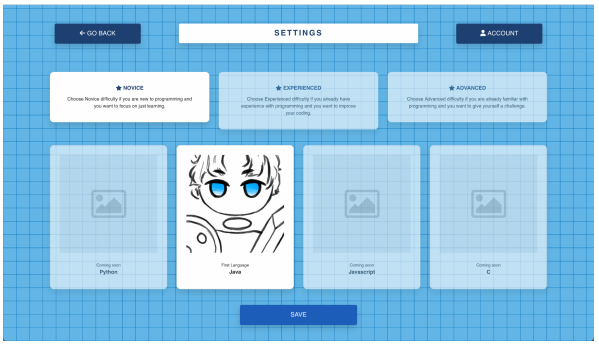


Fig. 21: Settings selection page.

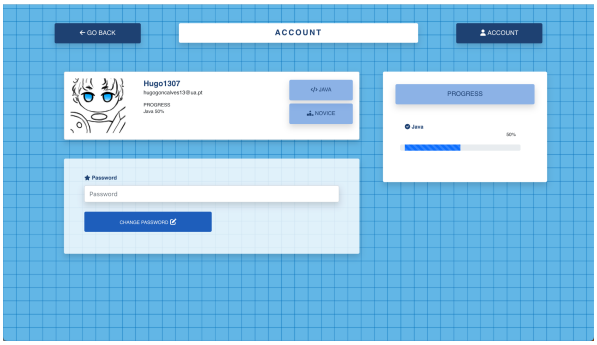


Fig. 22: Account page.

The Dashboard page - *Picture 1* - is the first page presented to the user after he logs into the application. There, he can select the different actions that are available in the project:

Play

To start playing one of the available levels. Clicking on the *"Play"* button will redirect the user to the page in *Picture 2*.

Solutions

The solutions page - *Picture 9* - can be accessed by clicking on the *"Solutions"* button. On this page it's possible to check other users' solutions for the implemented levels as well as the score obtained with each one of these solutions. The main purpose of this page is to allow our users to learn how to write better and more efficient code by learning with the solutions implemented by the community.

Leaderboards

The leaderboards page - *Picture 10* - can also be accessed using the dashboard by clicking the "*Leaderboards*" button. The main purpose of the leaderboards page is to create a certain degree of competitiveness among our users and keep them motivated to keep learning and improving their code/solutions to the levels. It is possible to filter the leaderboards results and access:

- Overal Leaderboard: Contains the highest scores of all levels in general.
- Chapter Leaderboards: Contains the highest scores of a certain chapter.
- Level Leaderboards: Contains the highest scores of a certain level.

We can also combine the latter filters with the difficulty filter. This allows us to filter all scores by the difficulty of the level where they were obtained.

Achievements

The achievements page in *Picture 11* is accessible through the "*Achievements*" button on the dashboard page. The main purpose of this Achievements page is to create a kind of certificate that can attest that a user, indeed, completed at least some of the levels.

Settings

The settings page that is visible in *Picture 12* is accessible by clicking on the "*Settings*" button on the dashboard page. This Settings page is intended to allow the user to change the difficulty and the programming language of the levels he is solving. Although currently it is not used, since we only support one programming language and one difficulty, it would be very useful in a future point when CodeSpell would support more languages and difficulties.

Account

The account page - *Picture 13* - can be accessed by clicking on the “Account” button on the top right corner of the screen. This page is intended to allow the user to review his personal details and change the password if needed.

9.3 Requirements completion

During the construction phase, we implemented totally or partially all initial functional and non-functional requirements defined in section 3.1 Functional Requirements and section 3.2 Non-Functional Requirements. However, during the development phase, we noticed that there were some requirements missing, and added them. Examples of these requirements are FR # 9 and FR #10.

Furthermore, some requirements were not totally completed due to a lack of time and therefore were only partially completed.

Requirement	Completion (%)	Justification
FR #1	100%	
FR #2	100%	
FR #3	20%	Currently, our system only supports Java as language and Novice as a difficulty level.
FR #4	100%	
FR #5	60%	Only two levels were implemented. The background story was not told during the levels.
FR #6	100%	
FR #7	100%	
FR #8	100%	
FR #9	100%	
FR #10	100%	
FR #11: *(Optional)*	0%	The level editor could not be implemented due to lack of time
NFR #1	100%	The users’ passwords are stored in the database using BCrypt.
NFR #2	100%	The code written by the user is executed in

		disposable Docker containers.
NFR #3	100%	The web app will provide a list of syntax errors with details about the errors and their location, and also tips about possible logic errors.
NFR #4	100%	The code written by the user is converted into a list of actions our character inside the game will execute.

10 Project Management

10.1 Project Management Tool

To easily keep track of the tasks of the work, as well as quickly assign them to the members of the team, we are using the project management tool ***Jira***[22]. The Jira platform is an agile project management tool that allows us to easily manage tasks and issues related to the project and do the necessary planning required. We can also organize the tasks into several ***Sprints***, organize them using ***Epics***, set up deadlines, etc.

10.2 Development Workflow

In order to make sure that all the developed code meets the requirements, was reviewed, and wouldn't break already existing code, we adopted the GitHub Flow[23] strategy. The GitHub Flow provides some recommendations on how the development flow for new features should be, i.e., what steps each feature should follow until it gets to the final version of the product. This means that when developing a new user story or feature in general we will follow the next steps (in order):

1. **Create a new branch using a short and self-descriptive name.**

   This step allows us to perform our tasks and write our code in a controlled, stable, and safe environment making sure that no one will interfere with our current job. This is also very important as it allows us to ask for reviews from other team members before merging our current job with the main branches.

2. **Perform changes in the created branch**

   Once a branch for the new feature/task was created, all the code related to that feature should be committed to this branch.

3. **Create a Pull Request and ask collaborators for review**

   Once the feature is completed, a pull request should be opened to the "develop" branch. Each Pull Request should include a self-describable title and a short description including:

   - The news/changes that Pull Request will provide.
   - A link to the JIRA user story related to that specific Pull Request.

[22] https://www.atlassian.com/software/jira
[23] https://docs.github.com/pt/get-started/quickstart/github-flow

Each team member should ask other collaborators to review and approve his pull request, and the latter ones should review the code and either comment/ask for changes or approve the changes.

At the end of each sprint, a Pull Request to the main branch is done, if possible.

4. Merge the Pull Request

Once the Pull Request was created, at least one approval is required to merge the pull request. In order to ensure this, the main and develop branches were locked and configured to only be mergeable once there is at least one approval and all the **checks** have passed.

5. Delete the branch

After the Pull Request is accepted and merged into one of the main branches (main, develop, etc.), the origin branch of that pull request should be deleted. This will avoid the pollution of branches in the git repository.

10.3 CI/CD pipelines

Continuous Integration and Continuous Delivery/Deployment are very important paradigms as they allow developers and organizations, in general, to deploy new features quickly as well as deploy bug fixes in an efficient way.

In our project, we've set up a CI pipeline in the Rest API module and created unit and integration tests to make sure that new changes in the Rest API module would not break the already existing code and would also comply with the accorded quality gates. Unfortunately, there was no time to test and implement CI pipelines for the remaining modules of our system. The Backend's CI pipeline was implemented using GitHub Actions.

```yaml
name: Maven CI

on:
  push:
        branches: [ master, develop ]
  pull_request:
        branches: [ master ]

jobs:
  build:
        name: Package & Test
        runs-on: ubuntu-latest

        steps:
        - uses: actions/checkout@v3
        - name: Set up JDK 11
        uses: actions/setup-java@v3
        with:
        java-version: '11'
        distribution: 'temurin'
        cache: maven
        - name: Build with Maven
        run: mvn -B package --file pom.xml
```

Snippet 1: CI pipeline for Rest API module

Continuous Delivery, in our case, was very important as our system is built by a lot of different modules that need to cooperate in a very specific way to make sure the platform will act and work as expected. Therefore, deploying manually all of those modules would take a huge amount of time, and it would need to be done every time a new feature in any of those modules would be implemented.

To avoid this inconvenience CD pipelines were set up for all the modules of our project using the CircleCI tools. The CircleCI pipeline calls a Bash Script in the remote deployment machine that is configured to perform all the necessary steps and updates in the environment to successfully deploy the new version of the module. The CircleCI platform will access our VMs using an ssh key that grants access to a user called “*circleci*” in our system. To prevent possible security issues with this approach, the user has few permissions in the system (don’t have access to “sudo” group, etc.) and can only perform the necessary operations to deploy a new version of the system.

```
version: 2.1

jobs:
  pull-and-build:
    docker:
      - image: arvindr226/alpine-ssh
    steps:
      - checkout
      - run: ssh -oStrictHostKeyChecking=no -v $USER@$IP
"~/Code_Spell/Backend/deploy.sh"

workflows:
  version: 2
  build-project:
    jobs:
      - pull-and-build:
          filters:
            branches:
              only:
                - develop
```

Snippet 2: CD pipeline for Rest API module

10.4 Deployment Environment

Our VPS was hosted by DigitalOcean[24]. We have chosen Digital Ocean for its ease of use and low prices compared to other solutions like AWS and Azure.

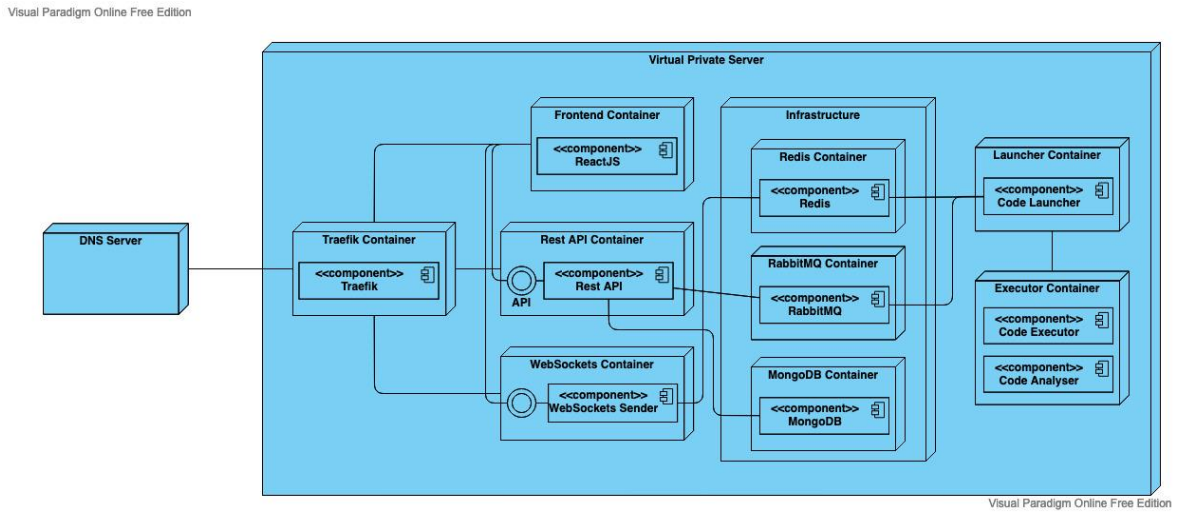


Diagram 3: Deployment diagram for VPS.

[24] https://www.digitalocean.com/

11 Conclusion

The goal of this project was to develop a reactive graphic platform that was capable, using a graphic interface, of teaching the most basic mechanisms of a programming language in a different way than is the conventional way of using abstract challenges (like implementing a certain algorithm).

We can say that our main goal was achieved by developing a platform with different levels that explore the very beginning, like if-else statements, and present the user with a graphic animation that is dynamic and reacts to errors in the code. Our platform is also able to present a user with the achievements they have made, calculate the score of one's code, and show how it ranks on the public leaderboards. Besides that, we also felt that it was fitting to include a way of checking out other people's solutions after finishing a level, as another learning tool (because through them a user can learn different implementations and approaches to the same code challenge). We also have extensive documentation for each level, as well as tips. In case the code submitted has any errors, we display a simplified and easier-to-read version of the error so that a novice programmer can easily figure out where it went wrong.

In future work, we would implement different languages as well as difficulties. We would also implement a level editor that would help both teachers and the developers create levels more easily.

When it comes to improvements. We would improve the scalability of the code compiler and executer, as well as improve the compilation speed by adding a cache to the Launcher module. We would also implement more levels to the Java language by touching on deeper topics.

12 References

[1] Hafiz, Ayaz & Jin, Kevin. (2021). Architecture of a Flexible and Cost-Effective Remote Code Execution Engine.

[2] Yash Budukh. (2020). “Building a remote code execution system”. Medium. https://medium.com/@yashbudukh/building-a-remote-code-execution-system-9e55c5b248d6

[3] Gunasekaran, Vijay P. 2020. “How To Automate Deployment Using CircleCI and GitHub on Ubuntu 18.04.” DigitalOcean. https://www.digitalocean.com/community/tutorials/how-to-automate-deployment-using-circleci-and-github-on-ubuntu-18-04

[4] Thompson, Keith. 2020. “How To Use Traefik v2 as a Reverse Proxy for Docker Containers on Ubuntu 20.04.” DigitalOcean. https://www.digitalocean.com/community/tutorials/how-to-use-traefik-v2-as-a-reverse-proxy-for-docker-containers-on-ubuntu-20-04

[5] “Writing Logic with Thunks.” *Redux*, 16 April 2022, https://redux.js.org/usage/writing-logic-thunks. Accessed 8 June 2022

[6] “Redux Essentials, Part 5: Async Logic and Data Fetching.” *Redux*, 11 May 2022, https://redux.js.org/tutorials/essentials/part-5-async-logic

[7] “Redux FAQ: General.” *Redux*, 23 February 2022, https://redux.js.org/faq/general

[8] “Using tweens.js”, *SBCode,* 7 May 2022 https://sbcode.net/threejs/tween/