

THE BUCHAREST UNIVERSITY OF ECONOMIC STUDIES
THE FACULTY OF ECONOMIC CYBERNETICS, STATISTICS AND
INFORMATICS



BACHELOR THESIS
APPLICATION FOR SUPPORTING
THE RECRUITMENT AND SELECTION PROCESSES IN
AN IT COMPANY

Coordinator,

Lect. ZOTA RĂZVAN, PhD

Graduate,

IOSIM LIVIU-DANIEL

BUCHAREST

2020

Statement regarding the originality of the content

I hereby declare that the results presented in this paper are entirely the result of my own creation unless reference is made to other authors' results. I confirm that any material used from other sources (magazines, books, articles, and Internet sites) is referenced in the paper and is indicated in the bibliographic reference list.

Contents

Introduction	4
Chapter 1 - Analysis	6
1.1. Context analysis.....	6
1.2. Informatic system analysis.....	8
1.2.1. Functionality assessment.....	8
1.2.2. Use Case diagrams.....	9
1.2.3. Activity diagrams.....	10
1.2.4. Interaction diagrams	14
Chapter 2 - Planning and Design.....	17
2.1. Detailed Class diagram.....	17
2.2. Database design	18
2.3. User interface design	19
2.4. Component diagram	21
2.5. Deployment diagram	22
Chapter 3 - Implementation.....	23
3.1. Back-End.....	24
3.2. Front-End	28
Chapter 4 - Information System Presentation.....	31
4.1. Applicant side.....	31
4.2. Recruiter side	37
Conclusions	41
Table of Figures	43
Bibliography	44

Introduction

For every working company, employment is vital. The recruitment and selection of applicants has always represented a keystone in paving the road for any successful business. The reality and standard of employing better working individuals from a vast array of people have risen from year to year, primarily because of the continuous digitalization of the world. In order to do well on the marketplace, indifferently of the conducted activity, continually keeping up with the latest trends is a requirement. This statement could not be more accurate in regards to IT companies. Technology is susceptible to change, thus adopting the latest and greatest hardware, software, and talent is a must in order to stay on top.

The application makes for a better and more efficient way to select and recruit participants in companies where time, distance, or economic possibilities are limited. The system also addresses the forever blamed problem of bureaucracy, thus simplifying and accelerating the process.

After going through the points addressed in the existing paper, the solution I have come to terms with is presented in the form of a management information system, named Employmapp, that includes all of the agents present in the regular admission process of a job attendee, based on an adaptable and flexible choice for the majority of software developing companies. The analysis had as a starting point, my experience thus far on the Romanian IT job market, namely open to positions as a software engineer to various big multination tech companies. The paper's final objective is to tackle inconsistencies in recruitment and enable hard-skill testing of aspiring software developers through theoretical and practical challenges. These objectives are tackled through a reliable system that is both easy to sustain and improve, and scalable through a well-documented process so that it can cover immediate and real problems.

The following chapters represent the gradual steps that shall be taken in the development process of any software product. They contain information regarding the implementation, but most importantly, the documentation of the information system so that every engineer can evaluate the present functionalities, how they have been modeled, and why they were needed in the first place.

In the first chapter, it is taken a broader look at the actual context in which the application is needed and how it fits in the market, thus differentiating the needed functionalities and grouping them in the must-have and nice to have categories. In this step, I have also realized a general picture of how the system should behave and what actors take part to create an abstraction that could later be translated into technical terms.

The second chapter takes a look at the application's architecture, and it is the first step towards implementation, thus entering the technical field. The purpose of this step is to assess and validate the information obtained in the first step by taking a look at the possible corner-cases and how the multiple factors interact with one another.

The third chapter takes things a step further by bringing in the pictures a vast array of technical terms that the application uses. It contains the motivation and the advantages of the

choices made. This chapter also examines the infrastructure that ensures the reliability, publishment, and other maintenance aspects of the system.

The last chapter is a detailed report of the product. It addresses in detail all of the functionalities through user interfaces and back-end implementation, exemplifying their use.

The paper comes to an end with a conclusion that englobes all of the points made, appreciating the different aspects of the application while also giving defining potential paths that could be taken towards improving the system.

Chapter 1 - Analysis

1.1. Context analysis

Before diving into more known software analysis steps, it is mandatory to understand the context in which this solution makes sense. By taking a step back, it is more accessible to analyze the context by looking at social, historical, and economic factors that influence the demand for this kind of system.

The recruitment of workers has transformed drastically through time. [1] Selection refers to the system put in place by the employers in order to determine job applicants' qualifications, following the decision-making process of acting upon that collected information. Employing hard-skill able individuals has become harder and harder. This phenomenon is mainly present in the IT companies, with the regular changes in used technologies, actual culture, and working environments. The average quality of people looking to occupy a position as a software developer has gone down. It is a lot easier to trick the employer nowadays by throwing fancy words in personal resumes, adding projects with exciting names that have been very easy to complete and taking advantage of the employer recruitment system, exploiting its weak points, and tricking the recruiter.

In this profoundly ambitious job market, pre-employment testing software enables the recruitment of the best-suited talent by evaluating and assessing each candidate's capabilities using various types of skill-assessment tests [1]. Ultimately, this information system falls into this category, presenting the following advantages.

- Weeding out of the usual bias regarding hiring through objectively cumulated data that adds to the already recorded resume.
- Quicker traversing of the process by both sides involved, with such a tool, the recruiter can access specific information faster while the applicant can always login and have access to the job requirements without the need of installing software himself
- Improvement in the average quality of new hires, shortlisting candidates through the required skill, keeping a database of the past applicants leaving room for future job openings and references [2]

The increasing number of individuals found on the IT work market has imposed businesses to scale up all of their environments. Those succeeding in doing so are becoming actual economic agents that need substantial funding to conduct their corresponding activity. Considering this and the highly competitive free market with the necessity to keep costs at bay while having an appealing and marketable image in the face of customers and job candidates, the need for a recruitment product is noticeable.

Every company is looking to shave off costs wherever there is a possibility. The things that hinder and limit the recruitment process the most can be classified into several expenses. These can be regarded as employment taxes, basic salaries, benefits, but to make these kinds of expenses worthwhile, firstly, look at the recruiting expenses.

Unless the company employs a fully remote workspace, there will be a need for interview rooms that will always occupy space in the headquarter, hindering their utility for other purposes. These need to be refurbished to create an excellent first impression, and rent can also be a factor, imagining there is a large number of job attendees, which is to be tested space can quickly provide a problem.

Furthermore, following this problem equipment is also a significant expense. [3] By far, the biggest problem is represented by the time management aspect, with Employmapp, multiple candidates can be evaluated in parallel using automatic scoring. The time taken for every applicant to install the software required for the assessment is shaved through easy web access and an integrated development environment.

Another widely known aspect of the information technology recruitment world is that it is impossible to test the abilities of a software engineer only through theoretical questions. While these have an essential role, an accurate evaluation would be accompanied by practical problems that address the applicant ability to think on the spot, solve real problems, and see his knowledge put into practice. Long term costs can be reduced by employing quality employees. Costs of elaborate onboarding processes are significant, these implying a large number of trainings and trainers to hold them. Also, these can be time ineffective; by taking essential talent and knowledge, less money is spent on personnel integration.

A widely overlooked factor by some employers is the importance of design and marketability. It is always to be taken into account that the recruitment goes both ways. Not only does the company need to choose the right applicant, but the vice-versa is also implied. Making sure the used products are professional looking and up-to-date goes a long way in the eyes of the market.

After assessing multiple sources of specialty literature regarding the matter, I consider it is necessary to compose a synthesis of all the factors that, from my point of view, make the presented paper usable and relevant in the solving of a real-life problem in IT companies. In the elaboration of the solution, I considered my personal experience with the matter after going through the multiple analysis steps and gaining experience as a software engineer myself.

The first concept of Employmapp's system was inspired by my experience in the recruitment position at a volunteering, non-profit organization. The applications present in the regarded project had a similar objective, but it lacked the reliability of knowledge assessment given by the practical problems. All of the questions asked to participants were either theoretical or irrelevant from a technical standpoint. Even though we had centralized information about interviews, resumes, and participants, there was not enough to precisely estimate their skill and decide close calls between candidates.

The other influence on the paper idea came from my experience as a job attendee. Often the process is very long or unreliable. With a pre-employment solution like Employmapp no code has to be written on paper, the information and feedback is not inscribed in a forgotten notepad, and the applicants can have a seamless experience, increasing the chance that they would work for the in-cause company.

1.2. Informatic system analysis

1.2.1. Functionality assessment

As discussed previously, the information system that is to be implemented, has as objective the simplification and the streamlining of the enrollment process in IT companies, primarily, where practical tests have to be taken. The application targets to save costs and space, diminish paperwork and provide data centralization and manipulation through widely accepted technologies that have become standards in the industry.

To compose an excellent package for sustaining the selection and recruitment, the application needs to have some base, must-have, functionalities that assure the system can function and some extra or nice-to-have functionalities that generate competitive advantage over similar software solutions.

Thereby the must-have functionalities are represented by:

- user registration (account creation)
- data validation
- ability for applicants to write, test and solve practical problems
- integrate development environment
- theoretical questions
- storage of applicants' information and results
- ability for recruiters to create tests
- unique identification based on user email

Following these, another set of additional functionalities is implemented in order to give the system the possibility to handle complex scenarios by also making it usable in real-life situations:

- storing of applicants' resumes
- measurement of test progress (through both time spent and completion stage)
- test stats display after finishing attempt,
- ability to preview tests by their description and other characteristics (like type and time)
- supplementary page for company information
- exportable to ".csv" table for applicant information table
- search and sort operations on applicant information table
- ability to give, update and delete feedback to applicants based on their interviews and resumes
- ability to create additional recruiter accounts inside the application's environment
- access to solution files from applicants' code questions

The above bullet-points have been categorized, taking into account how critical the related function is to the application and its frequency of use. These concepts will significantly help with time management in the development of the information system. It is to be noted that some of the nice-to-have features have been nixed off the list, mainly because of time, complexity, and

corresponding relevance, the ones that would have had the more significant impacts can be found in the conclusions segment of the paper.

All diagrams presented throughout the paper are composed with respect to the Unified Modeling Language (UML). For their execution, the *Visual Paradigm Community Edition 16.1* was used.

1.2.2. Use Case diagrams

The above-described functionalities are found at the basis of the applications' system. These can be exemplified through the use case diagrams. These kinds of diagrams represent the flow of the system in its purest form, picturing in a simple, natural and easy to understand way the main actions that can be performed by the two kinds of participants (the applicant and the recruiter). Included in these types of diagrams are the extension possibilities, where a use case could be continued by the conduct of another, which is either supplementary or depends on some kind of condition to occur.

The graphic depiction between the elements of the users and the system's elements is present in the following figures.

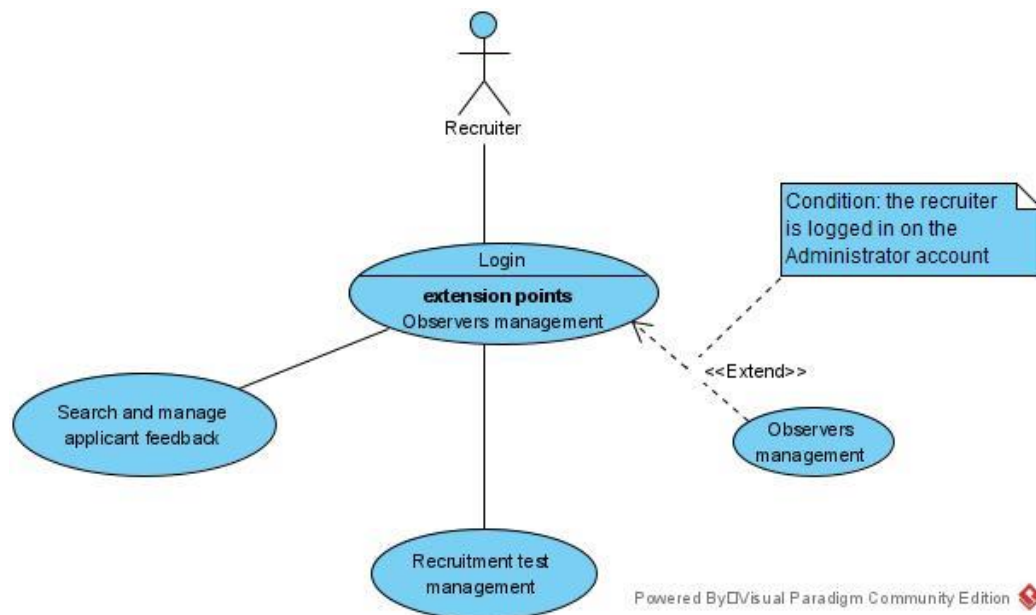


Figure 1.1 -Recruiter Use CaseDiagram

Once the application is found into the hands of a client company, it means that there is the need that the system must be independent and fully manageable with recruiter accounts. With this aspect in mind, there are two possible accounts for the company's employees, the administrator and the observer. The administrator can add and manage observer accounts so that there can be multiple personnel working on the recruitment tests, the review of applicant data, and feedback management. The observer account can do all the operations available on the administrator account, except the observer's management.

It is observed, looking at *Figure 1.1*, that the must-have functionalities are present on the recruiter platform through account creation, data validation, access to applicant information, and management of recruitment tests and observers.

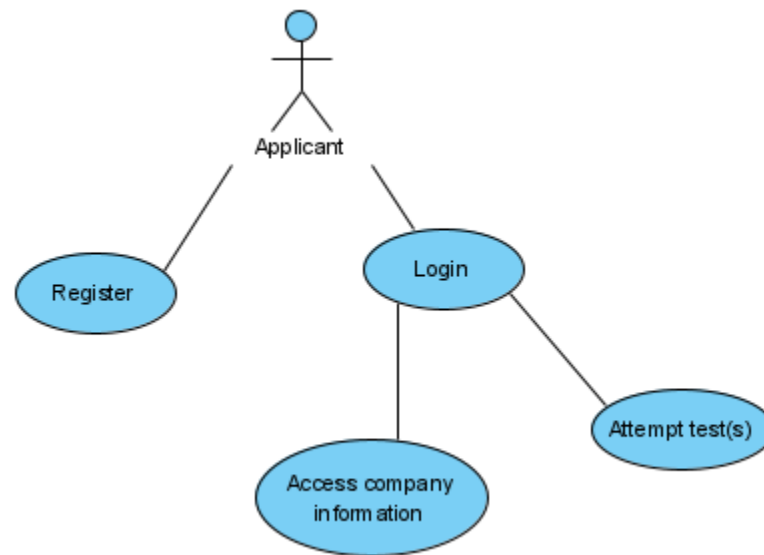


Figure 1.2 - Applicant Use Case Diagram

In order to make the recruitment as snappy and frictionless as possible for the applicant, his interactions are shortened. *Figure 1.2* presents the actions that this agent can imply in after registration and authentication in the system. There is the ability to view additional company information that can clarify possibly unanswered questions. Also, the ability to take recruitment tests enables the application ecosystem to exist by providing means for applicant evaluation.

As the standard table of the written evaluation of use case diagrams was not followed, some general terms can be clarified, although they do not significantly change the perception of the above figures. In both use cases, the diagrams have an outline state, the purpose of showing the general flow of actions, no preconditions, and a frequent occurrence.

1.2.3. Activity diagrams

Activity diagrams represent another important UML diagram that captures the dynamic of the system, thus differentiating from use case diagrams.

The activity diagrams treat in detail the processes, evidentiating the steps that need to be made in the context of a well-defined action. All these actions determine the final state of the product and manage the application's existing flow. So that the steps and stages of the operations can be represented, control flows are drawn between elements. Often these also come with segmented workflows into multiple swimlanes, threading the actions. This helps structure and group the different factors that come into play, making it easier to track what every entity has done.

This concept is applied in the following diagrams, often implying the automated system and its user.

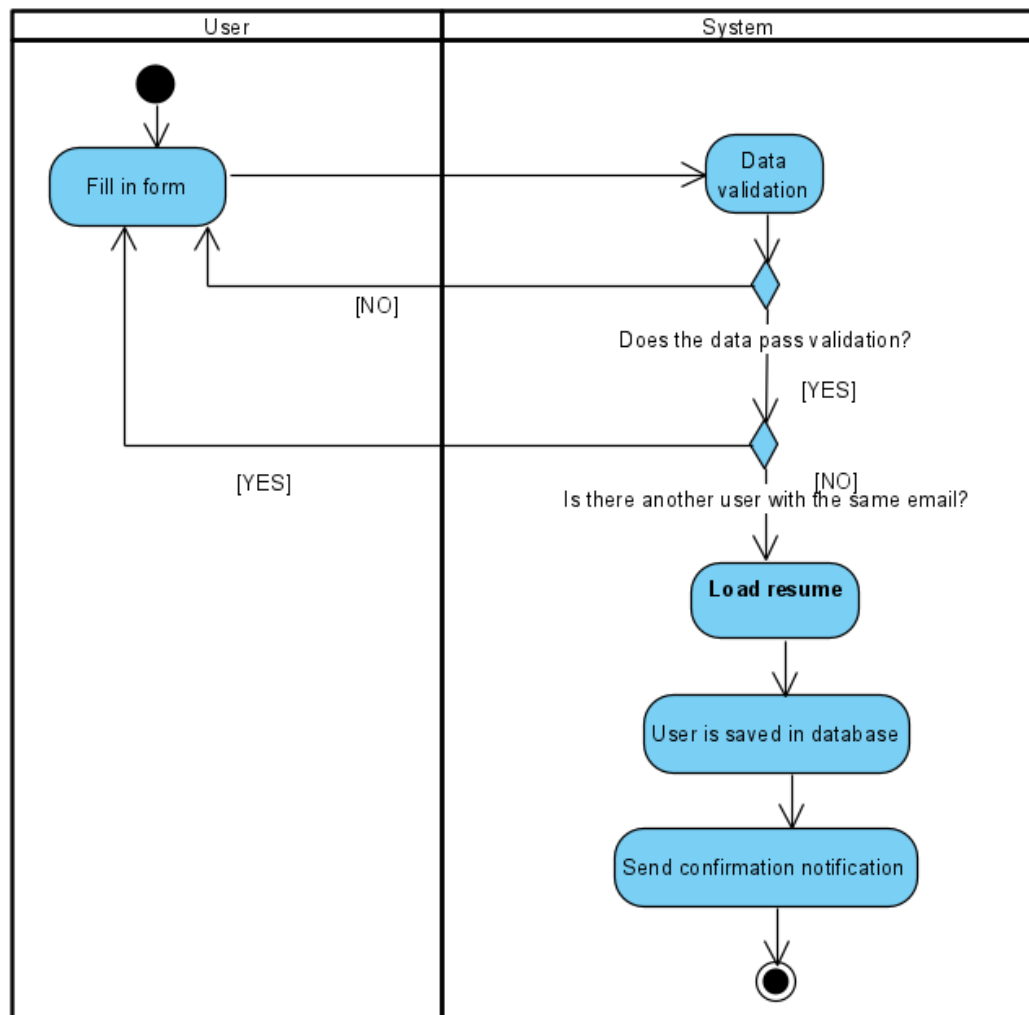


Figure 1.3 - RegistrationActivity Diagram

The first activity diagram, *Figure 1.3*, represents precisely the first step that the applicant has to make in order to take part in the process. The data inserted by the applicant in the registration form is validated and checked, assuring unique authentication based on the inserted email. If the password, phone, and name fields comply with known standards, the applicant can proceed to resume uploading. One very important aspect of registration is the functionality of resume storage. If all of the data complies, the system registers the applicant in the database, and only then will he have access to the requested account.

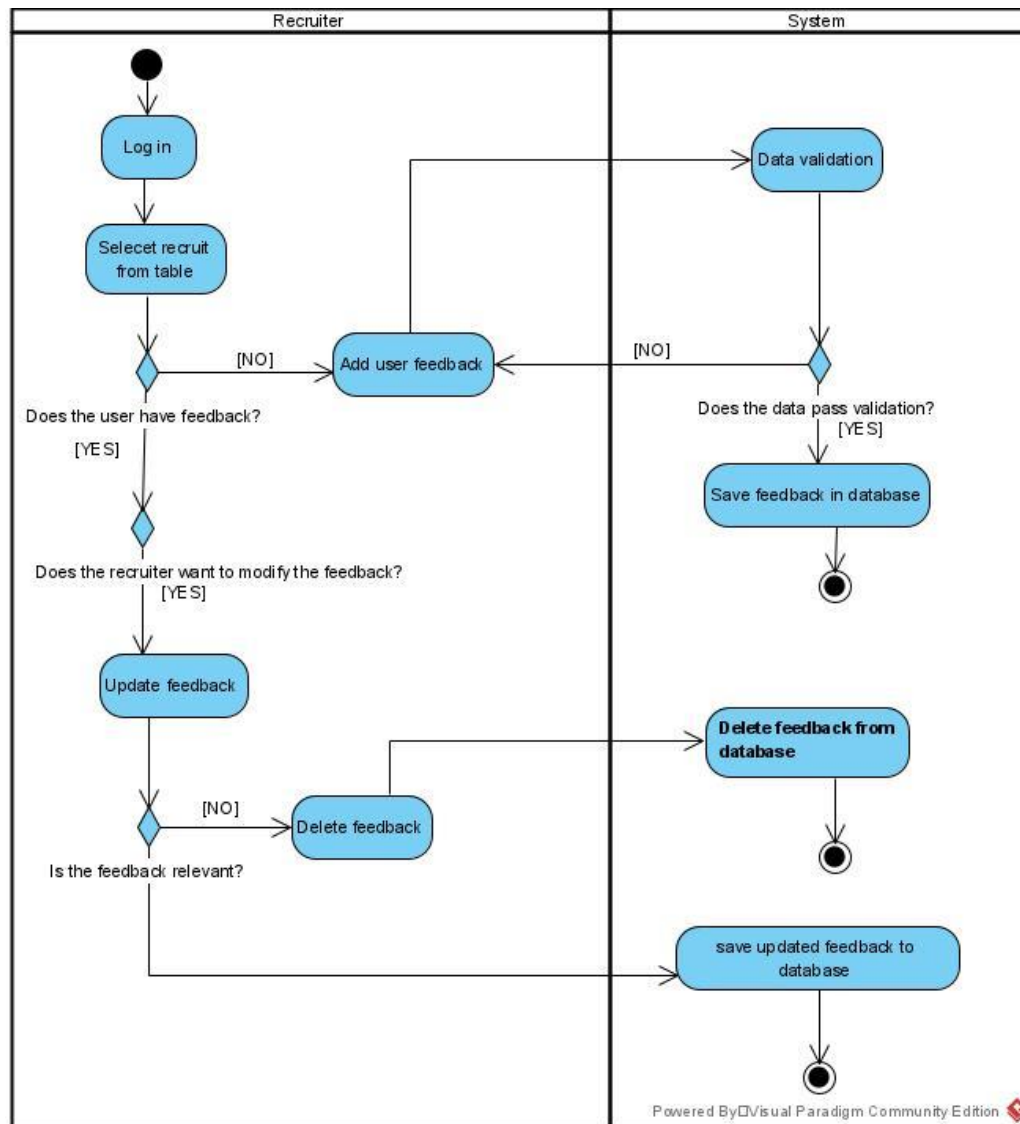


Figure 1. 4 - Feedback Management Activity Diagram

The second activity diagram refers to a couple of actions, the diagram, present in *Figure 1. 4*, represents the feedback management given by the recruiter. To make sure the right applicant has the right feedback, the recruiter needs to go through a couple of steps. Check if the user has feedback, check if the feedback is relevant, and add/update/delete based on the reflected situation. The user, logged in on the administrator or an observer account, after selecting the desired job attendee, is met with a feedback form, this form must be completed according to data validations. Subsequent to a positive form response, all actions are assisted by the system, making sure everything is translated into the database so the changes could persist.

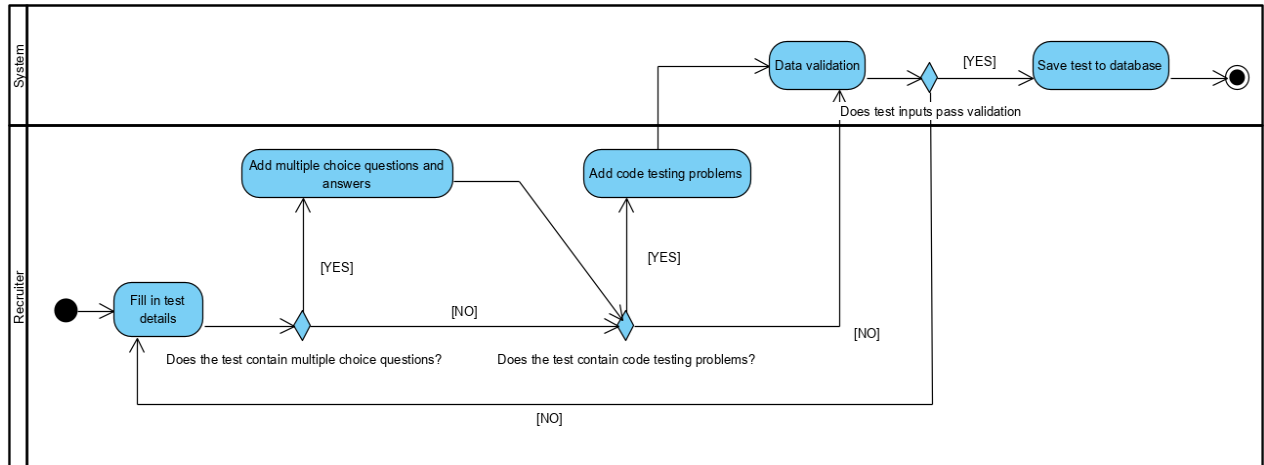


Figure 1.5 - Test Creation Activity Diagram

The third diagram, *Figure 1.5*, englobes the steps the recruiter has to make in order to create the right test for the company's needs. Firstly, based on the vacant position he is aiming to find a recruit for, he will compose a suitable recruitment test. The examination can contain multiple-choice questions or code-based challenges. For each type of question, there needs to be provided answers or unit tests based on which, the final score will be computed. The creation of such a test, like in the previous aspects, is done through completing a validated form. After complying with the form's requirements, the test will be added to the centralized database, subsequently being available on all the other accounts, albeit observer or applicant alike.

The BPMN (Business Process Model and Notation) diagrams can enable a more detailed view of the activity's components providing well-known standards. [4] Such a diagram is included in this chapter due to the fact it looks at a single operation that can also be covered with an activity diagram, due to their similarities in structure. The main reason behind using a BPMN communication diagram is because of the added clarity and elements that can better explain a complicated concept.

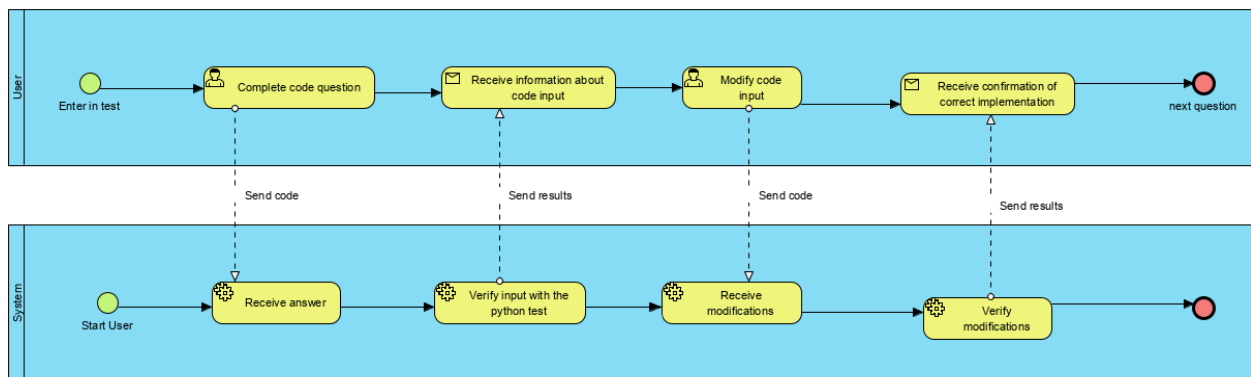


Figure 1.6 - Code Question Communication Diagram

The aspect which represents the most complex factor of the Employmapp application is the ability to provide code-based questions. In *Figure 1.6*, the process of code answering by the

applicant is reviewed. To have a communication diagram, two participants need to be present; in this case, the user and the system. The candidate sends responses to the system, which runs them in a safe environment, providing back responses such as how many unit tests have been passed or if there is a syntax error in the provided code. The number of verifications can be unlimited, albeit the test will run out of time, submitting the answer automatically. In the figure, it can be easily observed the steps, starting points, and the nature of the performed actions. The Message Flow composes the communication, operations are done automatically by the system and manually by the user.

1.2.4. Interaction diagrams

The interaction diagrams have a high grade of complexity because those also unlock a temporal point of reference upon the functionalities schemas. These share the content of activity diagrams. Three main functionalities are represented in the following diagrams, the feedback management, test attempting and creation.

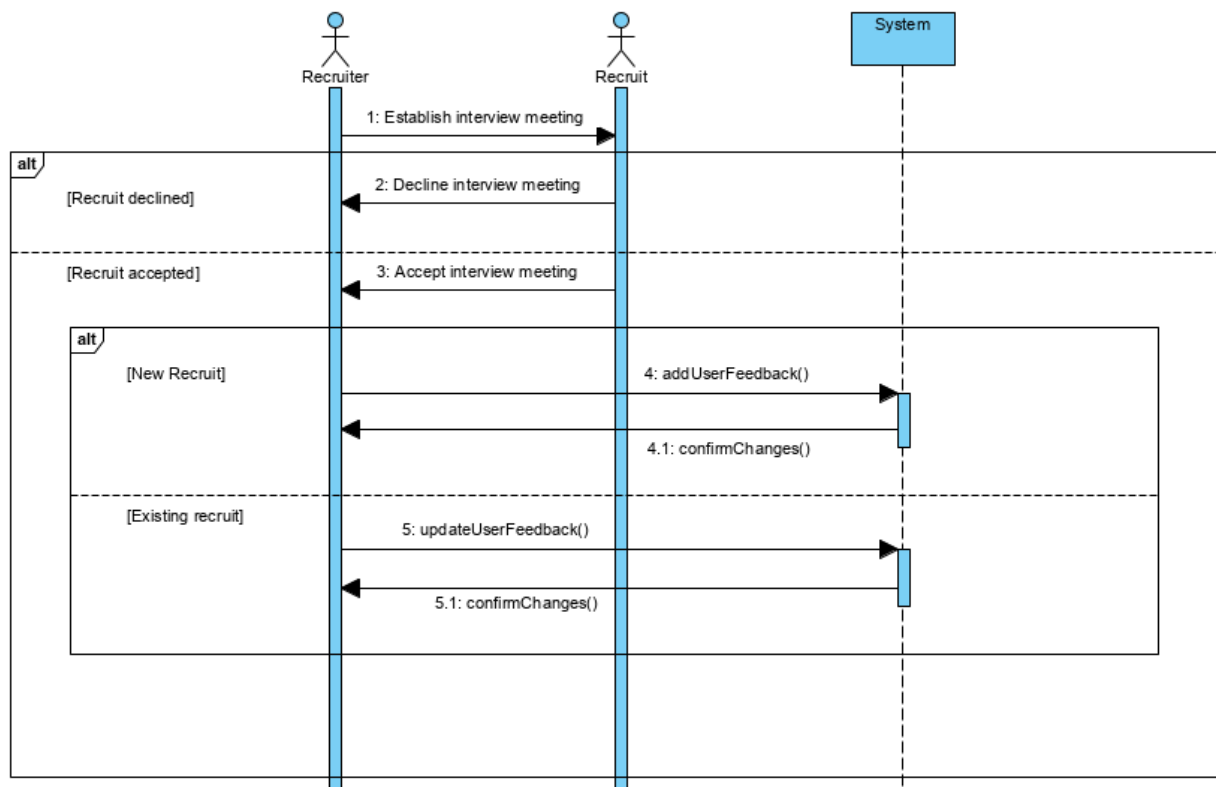


Figure 1. 7 - Feedback Management Interaction Diagram

Following the feedback management functionality, the Figure 1. 7 shows the successive steps the recruiter has to make to achieve his goal, with whom he is interacting, and how the informatic system responds in turn to his inputs.

Firstly, in order for the feedback to be assigned, there is the precondition that the recruit and the recruiter must have some kind of interview-like interaction. The activity can diverge in two possible significant ways; in the case, the applicant declines the job interview or the feedback

management flow if he already has been put in the system. The recruiter needs to assess if whether he should add, update, or delete its feedback, all the changes being, in turn, reflected automatically by the system inside the database.

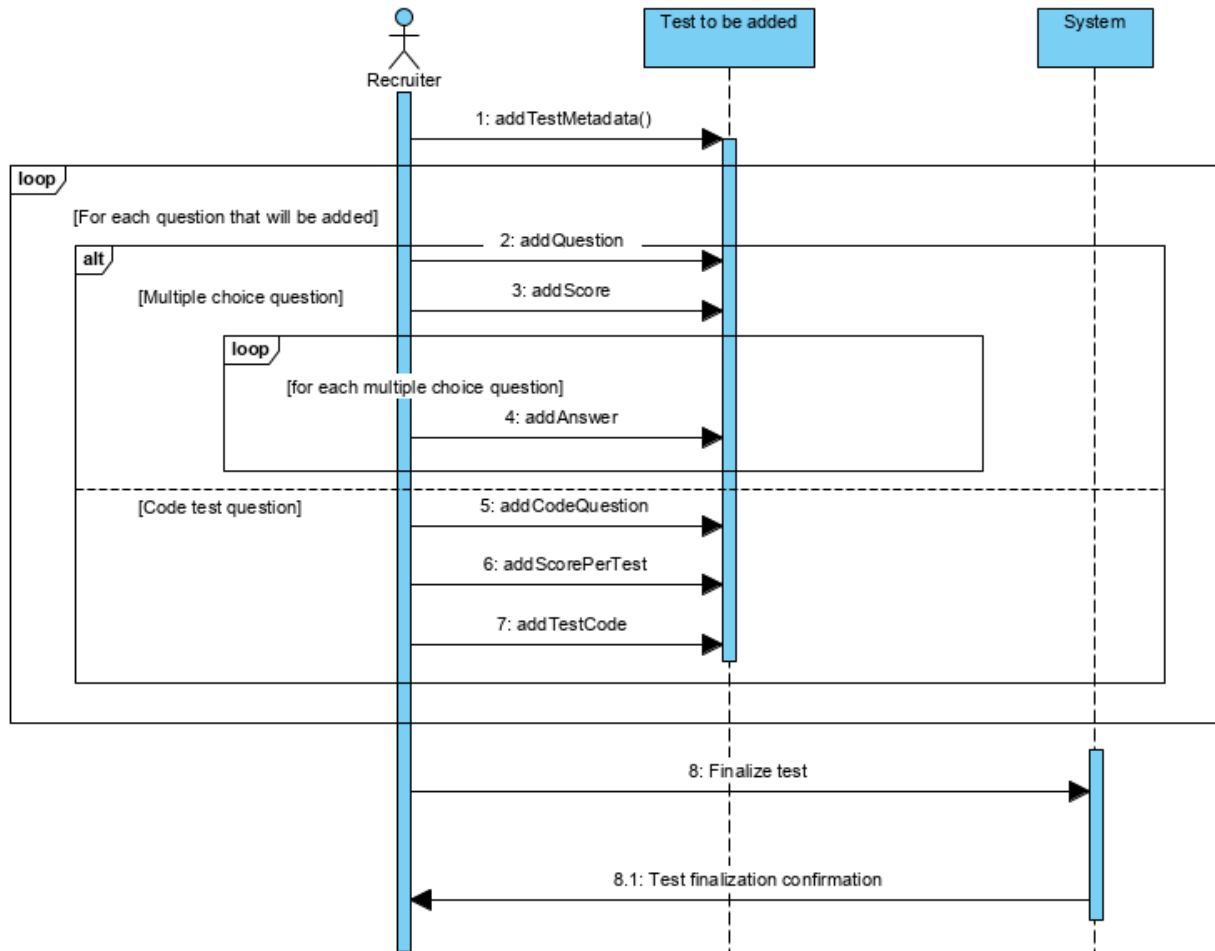


Figure 1. 8 - Test Creation Interaction Diagram

Another necessary functionality that projects the application in a standalone field of action is adding a new recruitment exam from a user-friendly interface. Firstly the general test information needs to be added; then, the recruiter can progress into adding multiple-choice questions for the theoretical part of the examination. Once that is done, the most critical aspect is following the code testing possibilities of the application. The user uploads the unitary test case for each code question to the application, along with the score and practical question. After finishing, he can confirm the data and progress, saving it into the system so the applicant can view and take their challenge.

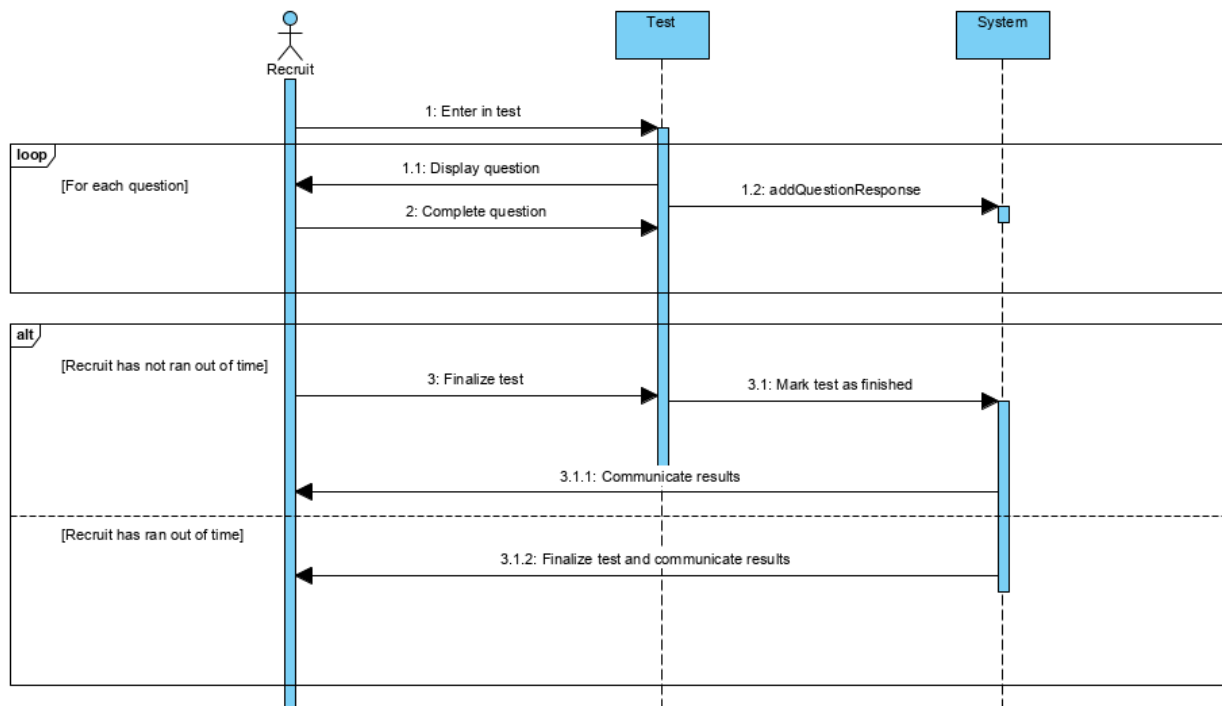


Figure 1.9 - Test Attempt Interaction Diagram

The application's main functionality is based on the applicant's ability to test their field experience. Once they are found inside the examination, each question is feed progressively to the participant. The responses are systematically saved as they are completed to ensure that the progress is not lost in the case some problems are encountered during the examination period. At least the test is finalized, and the recruit can get their final results.

Chapter 2 - Planning and Design

The Planning and Design chapter has as purpose the linkage of the factors presented in the analysis stage to the technological field. This stage represents a crucial step, translating the utilization models presented above into technical terms so the system could be implemented. Each of the following subchapters treat the planning and design process from a different point of view, taking into account the three fundamental aspects of an information system: the data modeling stage, the software stage, and the hardware stage.

2.1. Detailed Class diagram

Concerning the fact that the system is built on the shoulders of an object-oriented architecture, the detailed class diagram represents the primary building block in the data modeling process. The diagram has a meaningful impact on the design part of the system by presenting each entity that takes part in the application's ecosystem. These entities are accompanied by the functionalities that they can invoke, their attributes, and their relationships. It gives a strictly defined set of rules that entail defining the application's composing objects and behaviors.

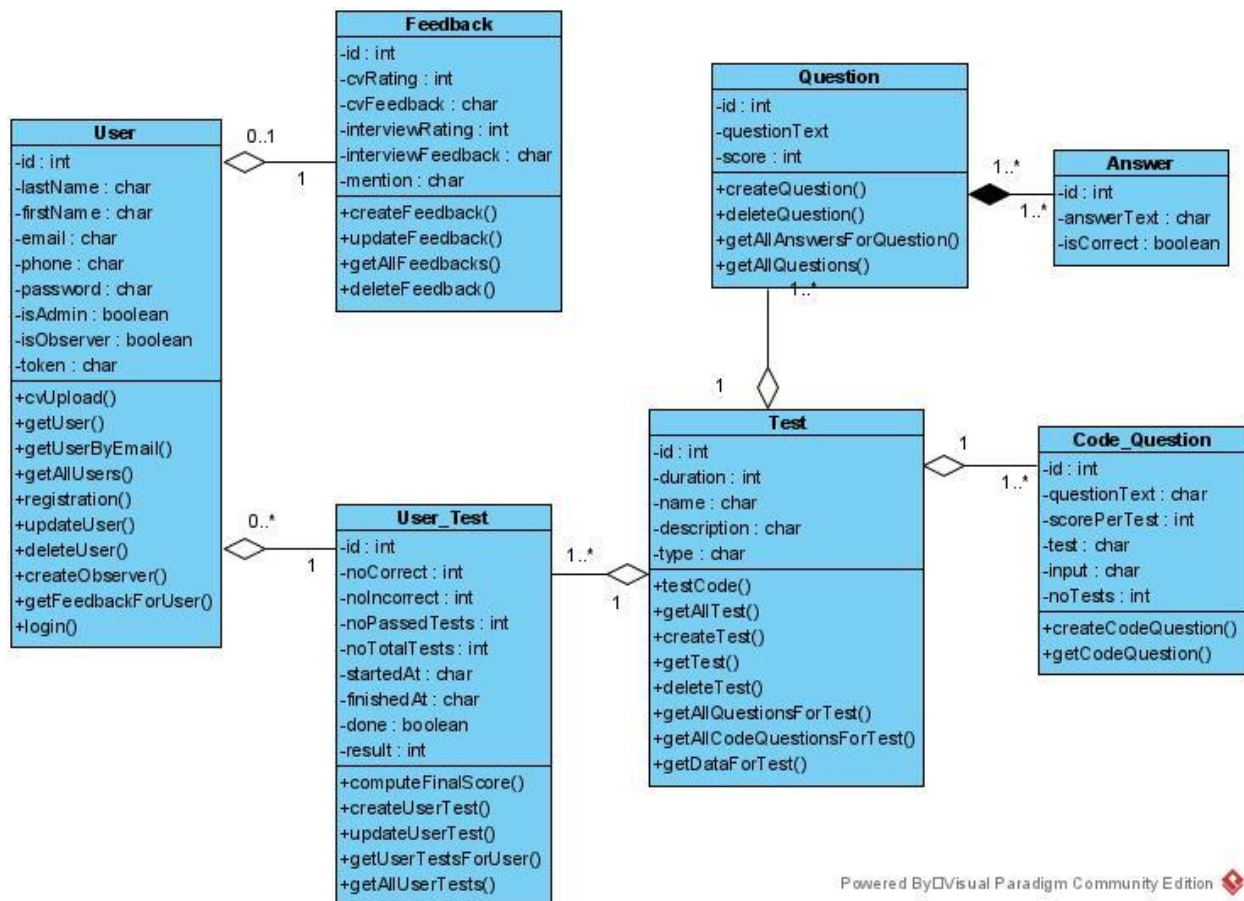


Figure 2.1 - Detailed Class Diagram

2.2. Database design

Following the diagram present in *Figure 2. 1*, the process of creating the database has to translate the involved agents into a general technical form, which is not dependent on the software which implements the solution as long as it respects the fundamental relational database's rules. This part of the data modeling process is vital for maintaining the persistence of data throughout the information system. It is desired to preserve the design of the detailed class diagram. Some crucial steps need to be made to achieve that, such as :

- each class entity is devised into a database table
- the classes' attributes will become fields in the corresponding table, prolonging their nature through a suitable type
- the attribute which realizes the unique identification of a class is to become the primary key
- the entities that depend on one another will become dependent tables in the schemas and will contain, as a foreign key, the primary key of the table they are dependent upon
- the many-to-many relationships will be resolved through an additional link table, as the relational databases do not allow this kind of relationship to exist [5]

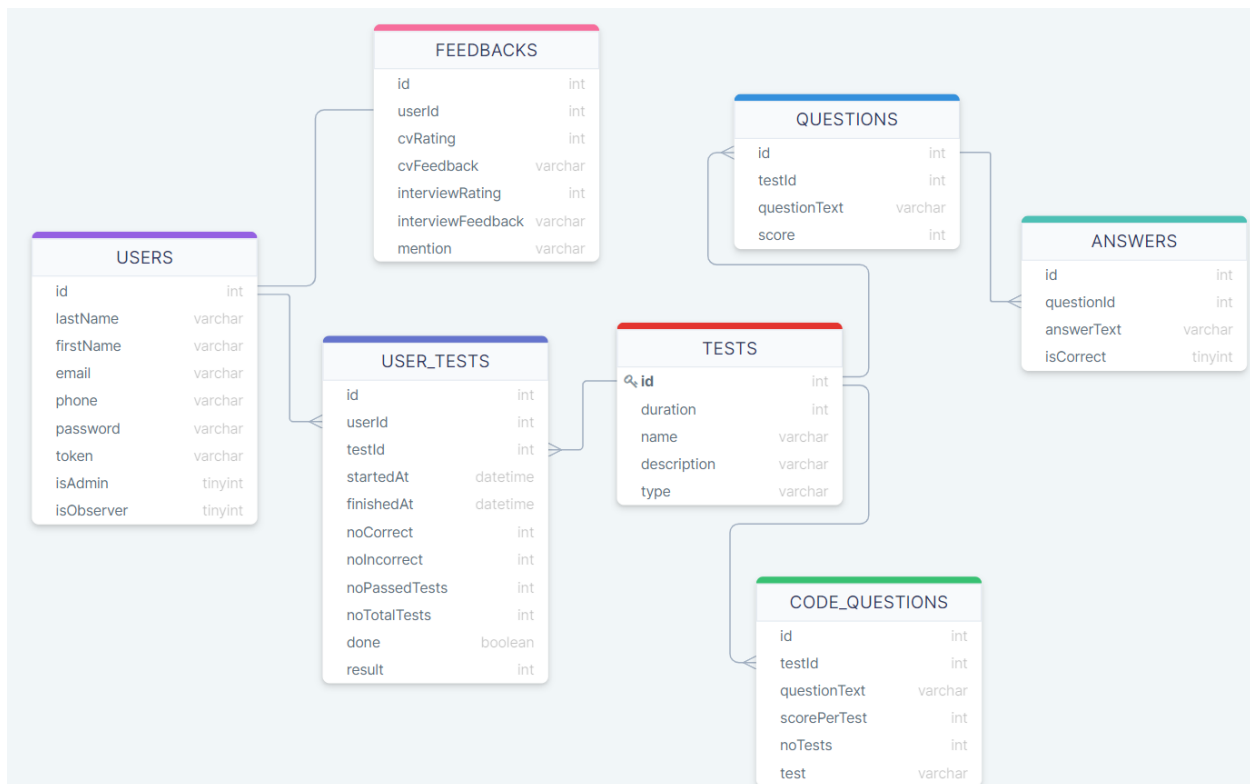


Figure 2. 2 - Database Schema

After applying the precise steps, the obtained schema can be used to promptly address each of the database entities and the utility they provide inside the system.

The USERS table holds all the user's information, no matter if these are recruiters or applicants. The distinguishment between the two entities is made through the *isAdmin* and *isObserver* fields. Depending on the value of these boolean fields, the user type is decided, which subsequently dictates the following flow of actions inside the application. The information stored inside this table will identify the users, give the ability of account creation, and recruitment operations on observers and applicants.

The FEEDBACKS table is responsible for assuring the persistence of the impact that the recruit made through the interview and resume screening processes. Its nature is also given by the one to one relationship with the USERS table.

The USER_TESTS table holds the results obtained by each applicant to each of the available tests, either code-based, multiple questions based, or both. It is also an implementation detail which decomposes the possible many to many relationships between users and tests.

The TESTS table is implemented with the objective of holding general metadata about each test, such as duration, name, description, and type. It also has to role to bring the array of different questions together for the applicant to take. Each test can contain multiple questions of each type, thing given by the one to many relationships with the CODE_QUESTIONS and QUESTIONS tables.

The theoretical multiple-choice questions will populate the QUESTIONS table. This entity will give leverage in testing the applicant also from a theoretical point of view.

The ANSWERS table is surprising the array of possible responses. Each one of them is marked through the "isCorrect" variable, which will allow the question to count in regards to the final score.

Finally, The CODE_QUESTIONS table has the objective of allowing the IT recruiters to test their participants from a practical standpoint of view, making the selection process of a developer more accurate. Each question passes the implementation requirements to the user and is given a unitary test to check the correctitude of the implemented algorithm.

2.3. User interface design

The design of user interfaces maps the use cases to possible real-life scenarios inside the application's environment. Through this step, the interactions of the user with the system are modeled and controlled. Then, after the data modeling stage, this subchapter helps to pick a starting point for which tools to use to bring the idea to life, which back-end and front-end technologies.

The excellent design of an information system helps in numerous categories. It helps the marketability of the product by making it more appealing to potential users and buyers, it helps in the decision-making process of which tools to use in the development process, and it also helps with the competitiveness aspect, giving the application originality and feel.

Inside the ecosystem of Employmapp, there is the possibility of three types of accounts. The administrator and observer account are for the recruiters, while the regular accounts are for applicants. Each of these has a different user interface that is designed to be as optimal as possible, relating the possible user types.

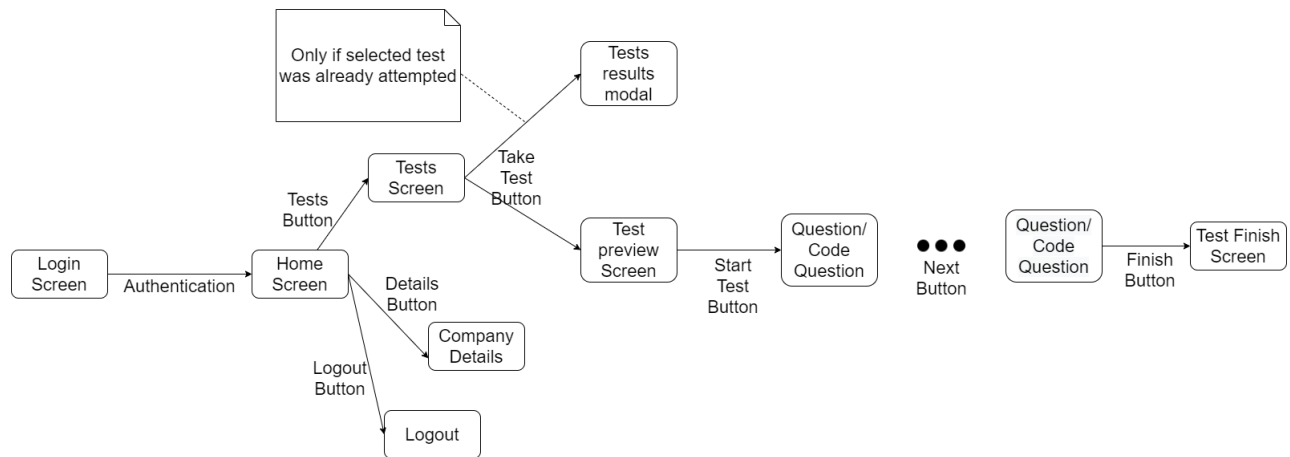


Figure 2.3 - Applicant Graphical Interface Schema

The applicant graphical interface schema takes into consideration that today's candidates apply to multiple job offers, especially in the IT field of work. It is essential to have a snappy interface that allows fast completion of recruitment requirements. Thus the applicant interface is as simple as possible. The main aspects that the employer is looking to get are the resume, an interview, and the assessment of theoretical and practical skills that the competitor has. The contact information and resume are given upon registration while the test is met once inside the application. Here he has an instant incentive to get to the page where the test options are displayed by giving him a call to action button. Once inside the test, the flow is straight forward and finishes like most tests, with a time limit or a submit button.

Also, keeping the participants as informed as possible is very important, like so, an additional information page is available. The logout is available and straightforward at ease throughout the app pages.

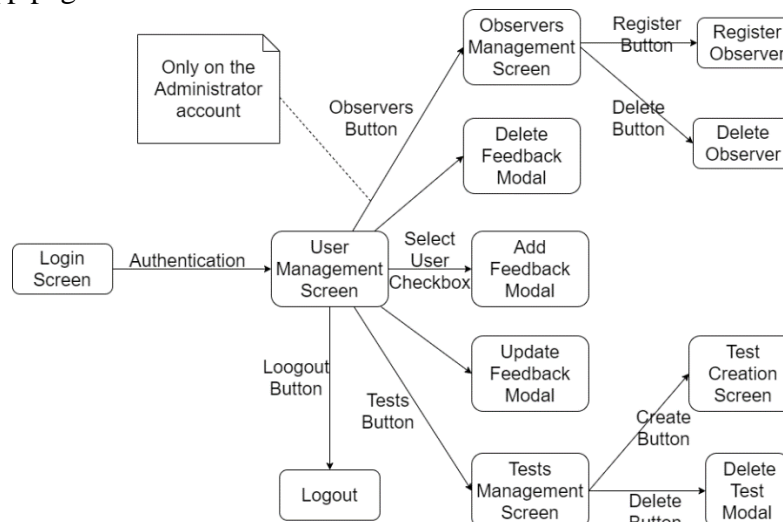


Figure 2.4 - Recruiter Graphical Interface Schema

The recruiter benefits from a richer interface that has to give access to several functionalities in order for the application to stay relevant. He needs to make contact with user information as fast

as possible, so that is why his landing page is represented by the applicant table. This table, in turn, gives access to contact information, test results, and possible feedback that the applicant might have. Also, the test management panel is essential; it gives the recruiter the ability to create, preview and delete recruitment tests by himself. As in the case of applicants, the option to log out is easily accessible throughout the recruiter interface.

As it was specified previously, there are two possible types of recruiter accounts, admin and observer. The observers are managed through the interface of the administrator. He can create, delete, and view all of the existing observer accounts. The difference between these kinds of accounts is the existence of the observer's management page. In the rest of the application, they benefit from the same functionalities.

2.4. Component diagram

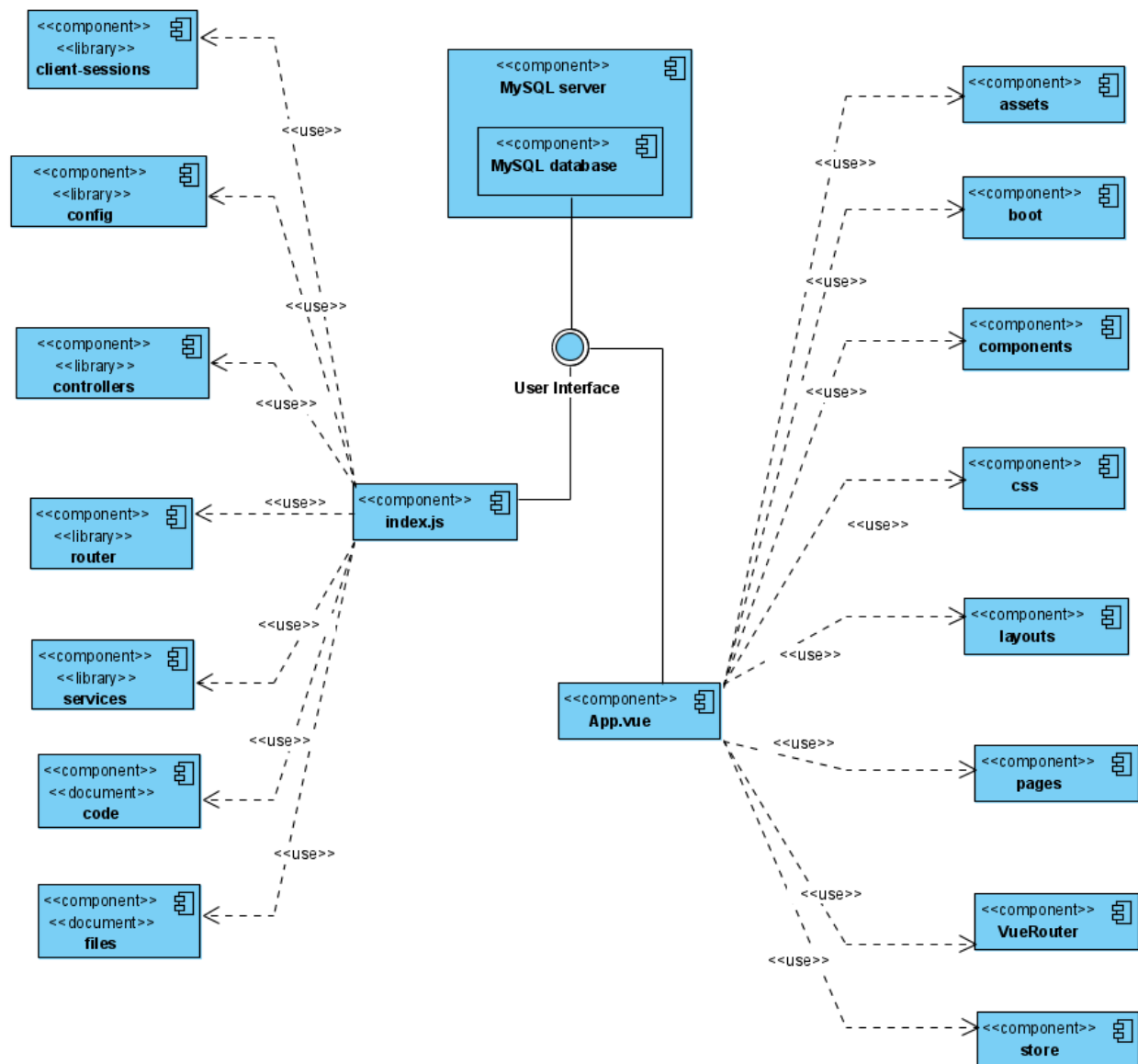


Figure 2.5 - Component Diagram

The component diagram englobes the main modules that need to be implemented in order for the system to take a touchable shape.

The diagram marks the software subsystems of the application and the nature of the connections between them. As can be seen, the system diverges in two distinct directions. This subchapter differs from the previous ones by addressing, for the first time, a tangible aspect of the software's implementation. There is a hint given to the technologies that have to be put in use. The diagram, in *Figure 2. 5*, approaches topics that will be addressed in the next chapter. [6]

2.5. Deployment diagram

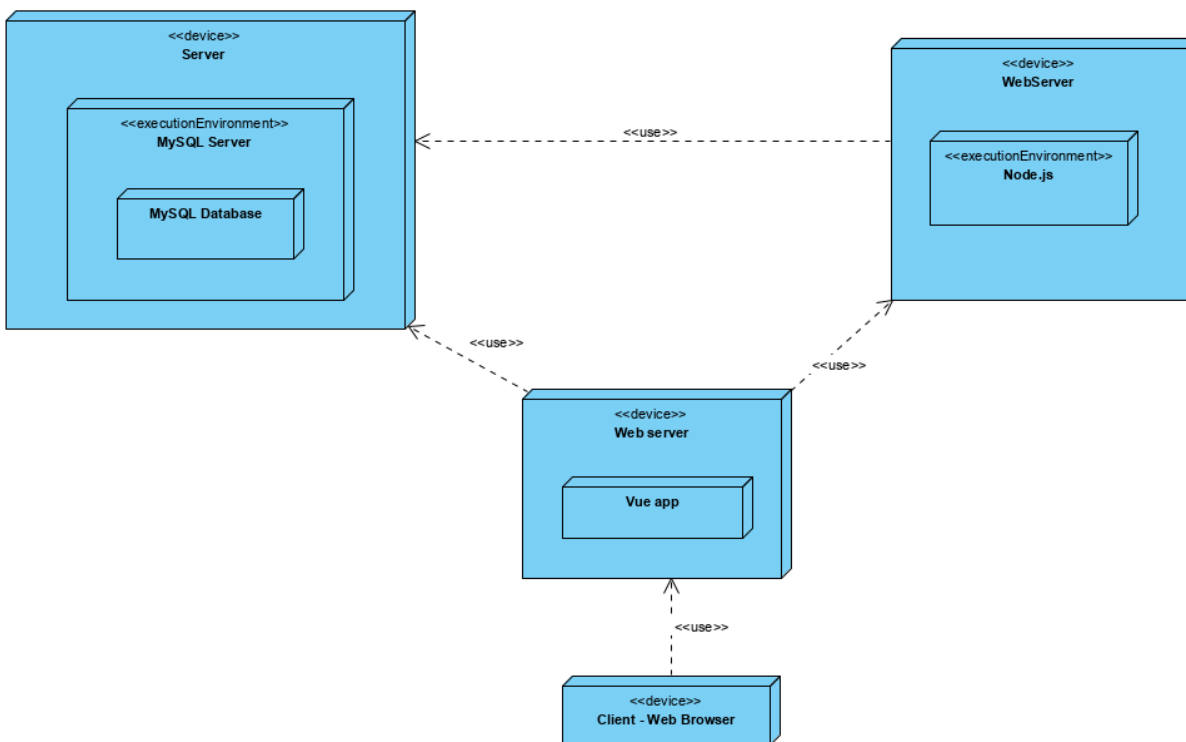


Figure 2. 6 - Deployment Diagram

The deployment diagram, displayed in *Figure 2. 6*, englobes all the physical and logical devices of the ecosystem and the nature of their relationships. The elements are built on an inclusion schema, gradually transferring the information from hardware to software. The diagram represents a general view on the architecture of the system that can be important and further elaborated in the publication and scaling processes.

Chapter 3 - Implementation

Employmapp is built on the Web platform, with the well-known client-server architecture. All the used frameworks, tools, dependencies, and libraries are well documented and stable, having solid backgrounds and a promising future, thus assuring forward compatibility. The asynchronous principles are used all throughout the application highlighting the flow of data and events.

On a general note, the product has, at its core, some technologies that strongly influence both sides of the system. It is essential to take notice of how the different kinds of extensions are brought in on both fields of the application. The most extensive Software Library (Registry) currently existing was used, named npm (Node Package Manager). It englobes and embeds the majority of components used.

```
est > {} package.json > {} dependencies
{
  "name": "employmapp",
  "version": "0.0.1",
  "description": "Application for supporting the selection and recruitment processes in an IT company",
  "productName": "Employmapp",
  "cordovaId": "org.cordova.quasar.app",
  "author": "Iosim Liviu <liviuiosim@gmail.com>",
  "private": true,
  "dependencies": {
    "@quasar/extras": "^1.0.0",
    "axios": "^0.18.1",
    "moment": "^2.26.0",
    "monaco-editor": "^0.20.0",
    "monaco-editor-vue": "^1.0.9",
    "monaco-editor-webpack-plugin": "^1.9.0",
    "quasar": "^1.0.0",
    "sass": "^1.26.3"
  },
  "devDependencies": {
    "@quasar/app": "^1.0.0",
    "@quasar/quasar-app-extension-qpdfviewer": "^1.0.0-beta.5"
  },
  "engines": {
    "node": ">= 10.18.1",
    "npm": ">= 6.13.4",
    "yarn": ">= 1.21.1"
  },
  "browserslist": [
    "last 1 version, not dead, ie >= 11"
  ]
}
```

Figure 3.1 - Front-End Package.json

In Figure 3.1, the package.json document of the front-end module can be seen. It is observed, in the *dependencies* object, some packages that are key in the development process, such as Quasar, moment, monaco-editor, all of which are brought in using the Node Package Manager.

The file was created using the "npm init" command, which plated with the quasar create CLI skin, guiding the base information about the project such as name, description, and author.

The Cordova attribute is related to the quasar ability to help in the integration of mobile applications. The browser list configures the compatibility with Internet Explorer, version 11 and above, indifferently of how active its maintenance may be. There is also support for the yarn engine.

It is also worth taking a look at the code management of the system. GIT is the most excellent information centralization and versioning system currently present, and it has become an industry-standard since its creation, being taught in computer science universities and courses. Also, through the use of GitHub, the system's implementation can be easily tracked and documented inside a corresponding repository. [7]

3.1. Back-End

As seen from the Component Diagram (*Figure 2. 5*), the contains of the application are diverged in two, namely Back-End and Front-End. In the implementation stage of the project, it is mandatory to decide which part will be first implemented. It is strongly advised to go for the Back-End side first because here you can mold the shape of how messages will be transmitted throughout the system so that Front-End can be modeled based on the shape of the transferred information, given the fact that it has a lower complexity, it is easier to change and adjust its implementation.

Further noticed in the Component Diagram, there is also the crucial part that englobes the database, the common point where all the instances of the application meet. The design of database architecture, discussed in subchapter 2.2, is the start of the Back-End architecture, as all of its elements are based on how the information is modeled and transferred, this subchapter, essentially describing the middle-man between the database and Front-End.

As implementing the Back-End side of things imposes much more complex tasks, a standard, well-documented architecture has to be integrated. As so the system can be characterized as RESTful (REpresentational State Transfer), an architectural style of providing standards between the web and various computer systems, facilitating the convenient transport of data. What stays and defines a REST interface is the detachment of client and server and their respective statelessness related to each other. This is already strongly hinted in the shape of the Component Diagram. [8]

The detachment of the client and server implies that the implementation regarding the two entities can be done individually, without one knowing about the existence of the other. A classic REST architectural style implies that the implementation of the server-side can be modified without altering the client's operation and vice-versa. Bearing in mind that both sides know the format of communication, the system can be kept modular and separate. [8]

The stateless aspect of REST implies that the server and client do not need to be aware of each other's states to function correctly. This aspect is reached through the utilization of resources rather than commands. In the case of the represented information system, the previously identified resources refer to the database, the common point from which all the instances of the application defer. [8]

For the data retention process, MySQL is used. It represents an open-source database management system that is widely accessible and adapted for its ease of use and well-documented status. With over 600.000 StackOverflow questions, it is very likely to find the solution to any problem that might be encountered.

In Employmapp, the primary tool is settled to be Node.js. It governs the server-side. Its easy-to-use asynchronous event-driven javascript architecture is the right solution for assuring the scalability of the system's network by presenting a non-blocking event-driven I/O model overshadowing other alternatives like PHP and Ruby on Rails for their threaded environments. [9]

Express is the most usual choice when it comes to working with Node.js. It is a free MIT license, open-source framework used for routing and middleware purposes. Middleware Express functions contain the request object (req), the response object (res), and the implicit function in the application's request-response cycle. These are used for every significant functionality of the application, from user registration to code testing. [10]

After establishing and understanding the base ideology and technologies that are to be used on the server-side, it is mandatory to put an architecture in place, followed by a suitable folder structure. This ensures the application's maintainability and scalability in time, respecting the standards of software quality like single responsibility, open-close principle, YAGNI, KISS, and many others.

In order to comply with RESTful principles and to make the system less susceptible to change, as seen in the Component Diagram, the principle modules are as follows: controllers, models, routes, services, code, and files. Each one of these components translates into folders that are brought together through and javascript index file.

Following the previously addressed ideology, that every implementation of a data-driven system should start with its database design, the first module that will be addressed should be the models.

Here the database tables are translated to desirable attributes inside javascript notation and environment using an ORM (Object-relational mapping) tool. Sequelize is used for its support of MySQL inside a Node.js governed system, transforming the tables into objects that can be used in an object-oriented related fashion. It is also used for the database configuration parameters, database account, name, and other constraints like the timezone, dialect, and host. Sequelize also conserves the dependency relation between entities through specialized functions. After all the models of the instances are defined in specific files, they are brought together, like in every module, in an index file, where the said dependencies are inflicted.

The "controllers" module represents the point where the system's logic is implemented. Following the single responsibility principle, each of the classes are dedicated to specific entities, where functions accomplish only one objective. These, as specified previously, are Express specific functions, containing the request and response objects. [11]

Controllers imply that data is being manipulated, giving its desired functionality. The data is imported from the "models" module, giving the database desired table in the form of an accessible object.

```

backend > controllers > JS auth.js > ...
1  const User = require("../models/index").User;
2
3  const userLogin = async (req, res) => {
4    const { email, password } = req.body;
5
6    const user = await User.findOne({ where: { email, password }, raw: true });
7
8    if (!user) {
9      res.status(403).send({ message: "Incorrect email or password" });
10   } else {
11     if (req.session.id) {
12       res.status(202).send({ message: "Already logged it" });
13     } else {
14       req.session.id = user.id;
15       req.session.token = user.token;
16       res.status(200).send({
17         message: "Successful login", details: {
18           id: user.id,
19           name: user.firstName + " " + user.lastName,
20           isAdmin: user.isAdmin,
21           isObserver: user.isObserver
22         }
23       });
24     }
25   }
26 }
27
28 module.exports = {
29   userLogin
30 };

```

Figure 3. 2 - Authentication Controller

In Figure 3. 2, an example is observed through the implementation of the authentication functionality. This dedicated file imports the User object from the model module. What follows is an Express standard function, with the req and res parameters. The user that wants to log in to his account is identified and validated.

In controller functions, the HTTP codes are a staple; they dictate the received feedback of both the system and the user. Here the transfer of information is also modeled for the Front-End to use. As so, responses are sent with information about users, like their name and type. The message can be used and displayed, for example, as a welcome user-friendly pop-up.

Some other important aspects are the use of cookies-sessions and javascript tokens technologies. Cookie sessions are pieces of information that are stored in temporary memory designed to help the identification of the user through other stages of the application. The javascript tokens, give bulletproof reliability, giving the user another layer of individuality, excepting the unique authentication by email.

The majority of the functions included in this module represent basic CRUD functionalities, while others are complex. All of the files respect the pattern of "auth.js" differing in the number of functions and imported elements.

Routing represents one of the staple aspects of web software solutions. For this application, the solution used for Back-End routing was the Express router because it already integrates with the middleware functions and is extremely intuitive for any inexperienced developer. [12]

The process of linking the application's endpoints (URIs) to client requests represents routing. In Express, each of the HTTP methods, also known as web's verbs, have a corresponding function, thus efficiently communicating the CRUD logic.

The module contains routes that relate to each entity of the system and are strongly tied to controllers, able to call their logic, thus creating the popular APIs (Application Programming Interfaces). The structure of the code gives it significant efficiency, neatly segmenting the system. Considering that multiple functions can be called on a single route, though the majority only stand for a single functionality.

```
backend > routes > JS tests.js > ...
1  const router = require("express").Router();
2  const testController = require("../controllers/tests.js");
3
4  router.get("/", testController.getAllTest);
5  router.get("/:id", testController.getTest);
6  router.post("/", testController.createTest);
7  router.delete("/:id", testController.deleteTest);
8  router.get("/questions/:test_id", testController.getAllQuestionsForTest);
9  router.get("/codeQuestions/:test_id", testController.getAllCodeQuestionsForTest);
10 router.get("/data/:test_id", testController.getDataForTest);
11 router.post("/testCode", testController.testCode)
12
13 module.exports = router;
```

Figure 3.3 - Test Routes

For example, in the provided picture, the Express router and corresponding controller for the entity are imported. The generic routing functions can be easily identified, corresponding to those are paths that are called in by the client, who call their corresponding functions. For tests, there are the basic post, get, put delete operations, but also some more intricate variations, like returning all corresponding questions for the test with the id given as a query param in place of the ":test_id". At the end of the file, the constructed router is exported to be further used in the main index of the router module. Looking at the routes of a project can also hint to its complexity, as so, in the test's routes, the code testing functionality is foreshadowed.

The module called services represents another essential part of the Back-End architecture, representing mainly an implementation detail with the purpose of maintaining code cleanliness. Here functions that can be found in multiple places throughout the system are implemented. This module has a similar structure as the controllers.

The config folder has the objective of holding sensitive data about the system's agents like the admin and database accounts. These are stored in the form of JSON, which can be called and used throughout the application. It is to be noted that this file is confidential and should not be uploaded on a public server. Thus, it is being included in the *gitignore*. For experimental purposes, the JSON structure can be disclosed.

The files module is strictly implemented for the resume storage functionality and is populated as more applicants successfully complete the registration process.

Finally, the code module serves as a management tool for code testing functionality. Here are stored the unit tests inserted in the recruiter's test creation screen along with the applicant's code. Also stored here are some necessary particularities related to the tested programming language. In this case, along with python, come the "_pycache_" folder at the runtime of tested code questions.

In the end, all the Back-End architecture unifies in a single point that imports all of the modules either directly or through dependents. Here some other relevant technologies come to light. The body-parser is a staple of Express applications, that once even made part of the framework. Like its name suggests, it facilitates the processing of a JSON's body.

Given the fact that code is directly picked up from the user and sent to be run in the heart of our server, huge security risks are generated. To mitigate some of these risks, the bandwidth throttling tool of Express was used in the attempt to limit resource-consuming activities that normally should not occur. Also, CORS takes a look at the file management of the application stopping unusual behaviors.

3.2. Front-End

On its basis, the front-end architecture is implemented on the fundamental, standard, and recommended tools, HTML5, CSS3, JavaScriptES6. All of which represent a standard in the industry and mold many frameworks and products. For managing time and maintainability, the app implements a javascript framework named Vue.js, on top of which was added Quasar-framework built, especially for Vue.js3 powered platforms and products. In addition to these, some other tools have been introduced regarding more specific tasks.

Vue.js is an open-source implemented framework first introduced by Evan You. Distinguishing from its rivals Angular.js and React.js, which are supported by IT giants, respectively Google and Facebook, Vue.js is community-based, meaning it does not have to push updates forcefully, and it has the advantage of being reviewed by thousands of developers.

Vue.js was primarily chosen for its smooth learning curve; it is easy to understand and develop applications for any starting web developer. Other key advantages of choosing Vue.js over its rivals represent the very small size of 18-22KB, taking virtually no time to download and use it, overshadowing bulky frameworks like Angular.js and Ember.js. Flexibility is a significant factor that also relates to reliability. It is easy to run straight in the browser and can support complex applications using ES6, JSX, components, typescript, bundling, and countless preferences that developers might use. Furthermore, it addresses the drawbacks of the previously

dominating front-end giants. They take the component-based approach, virtual rendering ability, props, single flow for component hierarchy, state, and many other factors through a well-documented environment making for a robust package. [13]

As another additional library with high regard to code scalability is Vuex, inspired by Flux, Redux, and The Elm Architecture. As in any Vue application, there are three significant factors represented by state, view, and actions. When building a project, it is taken into account that multiple views can depend on the same piece of state, which can be mutated by actions from different parts of the application. Props can represent a solution, but if the project scales up, it can become extremely tedious. [14]

With the Singleton design pattern in mind, the Vuex store serves as a centralized way of manipulating data, so it behaves only in conventional ways by using simple modules through state, actions, getters, and mutations. As such, stores have been implemented for users, feedbacks, results, and tests, the four main present entities throughout the application. Vuex was used to look at long-term productivity, giving a way to create some boilerplate code for Employmapp.

The front-end technologies assortment could not be complete without a User Interface framework. After establishing the choice of Vue.js, there was a surprisingly large palette of choices for going to the UI side. Quasar Framework was chosen for its smooth engagement with Vue and its components and directives.

Quasar is a Romanian made product, with principal contributors being Razvan Stoenescu and Popescu Dan. It provides easy interface and design leverages, fully customizable with already known standard technologies (CSS, HTML, JS). As of today, Quasar includes over 123 components, a number which is continually increasing due to a dedicated community, open-source access, and notable code fragmentation. This framework is used throughout the application for creating all sorts of views: cards, tables, buttons, dialogs, and much more. Although a useful UI framework, Quasar is not just that. It provides helper functions, components, and props for flex and grid, router, Axios, some Vue directives, and components, making it an obvious choice over competitors like Vuetify. In Employmapp every touchable component has been influenced by Quasar, from test and observer cards, to the buttons in the drawer-like menu. [15]

Going back to the diagram of the components, after establishing the leading technologies used and the reasoning behind using them, some other modules of the Front-End architecture can be addressed. As mentioned previously, the system is built using Vue.js together with the Quasar framework, that alone defines the architecture of the client-side as well as its folder structure.

As already stated previously, in the Back-End subchapter, the more segmented and structured the code of the app is, the easier it is to maintain and further scale. For this purpose, some additional elements have been added over the classic Vue.js structure that also treat global influences on the application.

The boot module imports technologies into the ecosystem, like Axios.js, for example that are globally used. Also, taking a look at this aspect, the *css* module defines the branding of the application, storing here, in Stylus variables, the colors used throughout the layouts and pages of

Employmapp, further CSS influences can be added here if desired. Also, as for every design project, an *assets* module is mandatory, there the images, fonts, and other similar things are kept.

The Front-End is devised between pages and layouts. Layouts contain elements that continue or are used from page to page, like the drawer-menu and navbar, while pages contain the elements that make up the core of the application. The components module is used to segment the code by storing elements that can be used in multiple pages and layouts or for code structurality.

Looking at visual components, assuring that the main drive behind Employmapp is to have an active touch with coding environments, the Monaco editor was chosen as a solution. Using Quasar integration with Webpack solutions and Vue component-like structure, the monaco-editor-vue was imported through the Node Package Manager. This addition gives applicants a friendly environment to write their solution, an attractive feature for recruiters, and a familiarity aspect to the application, considering it is a solution embedded in the most popular text editor of 2019, Visual Studio Code.

The visual components could not take their functionality without the router and store modules.

VueRouter represents the standard and official router of the Vue.js framework. It eases the challenge of building Single Page Applications, deeply integrating in the landscape of the application with features like nesting, modularity, wildcards, route params, links, and transitions. It becomes a straight forward choice for the application. It is assigned with the task of assigning the link to the application's pages. [16]

The system is filled with functionality intercepted from Back-end through the Axios.js tool and the store module. Like discussed previously, Vuex facilitates the transfer of state, getter, mutations, and actions to the different pages.

For managing the intense communication between front-end and back-end, Axios.js library was a solution that could provide fast HTTP requests. It is a well-documented library that aims to improve the basic fetch approach that plain JS provided supporting promises, better error handling, and supplementary protection against cross-site request forgery. This module intercepts the endpoints from Back-End, assuring the flow of information. [17]

Chapter 4 - Information System Presentation

Considering that the chapter related to the implementation stage, the Employmapp application contains two main types of users, recruiters and applicants.

This chapter is meant to take a more detailed look at the application's interfaces and corresponding functionalities. As explained in the first chapter, the recruitment would represent a more massive challenge if it were to be done on paper. The objective of the application is to be as widely accessible as possible. That is why, considering further development into PWAs (Progressive Web Applications), the interfaces adopt a responsive design allowing access on mobile devices and tablets. The following scenarios take account of this aspect, presenting, depending on the case, the most optimal formula for the type of user involved.

4.1. Applicant side

The applicant agent is the first one to be analyzed. Like in any regular data-driven application, the first step is represented by registration, the candidate, being driven through a structured form to enter his corresponding personal data, alongside his resume.

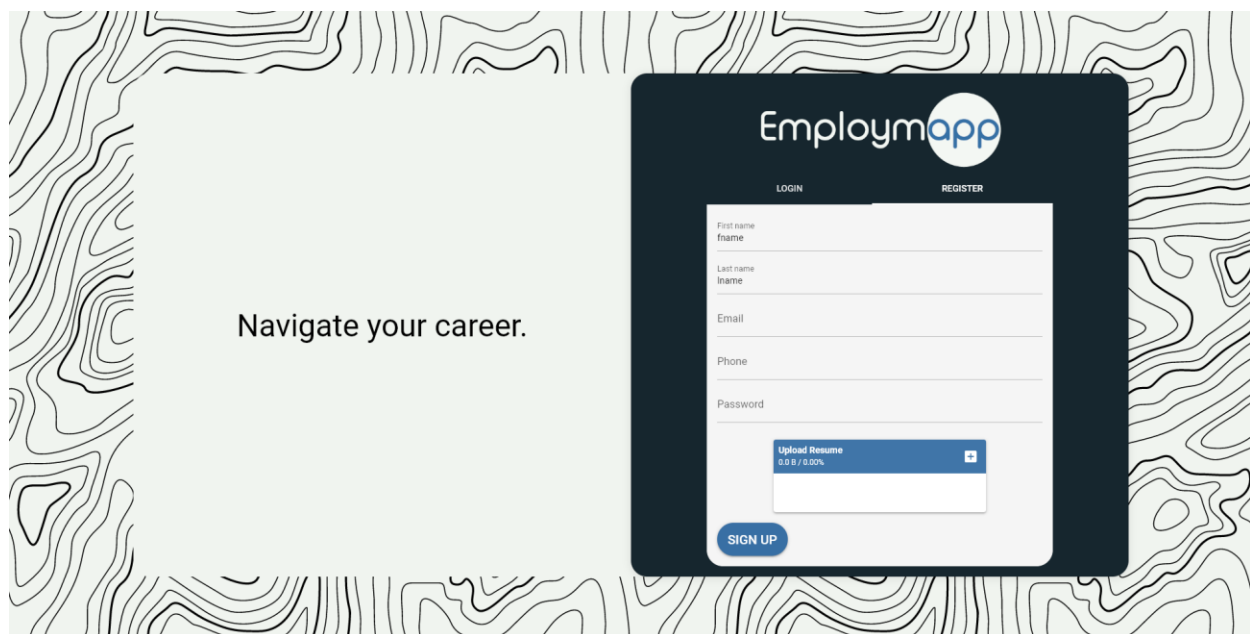


Figure 4.1 - Registration screen

The form's fields are categorized depending on the desired personal information, requiring an email, phone number and resume, on their basis the applicant can be invoked to the interview, and a general assessment can be made about him, even before taking the recruitment tests. After passing registration validations, the applicant account is created, giving him access to the rest of the system.

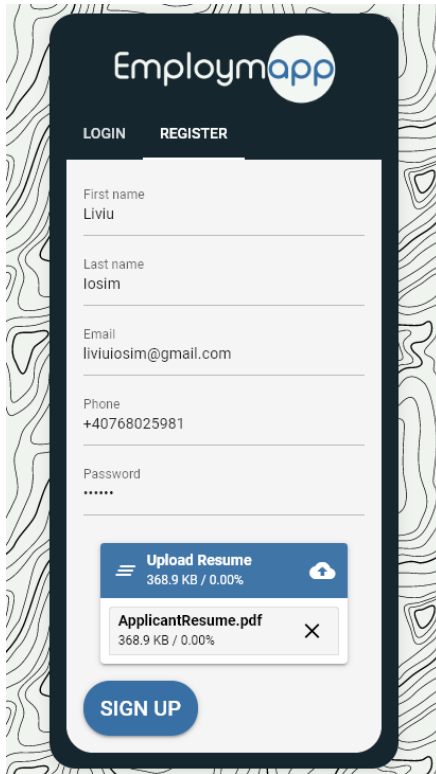


Figure 4.2 - Register Mobile Screen

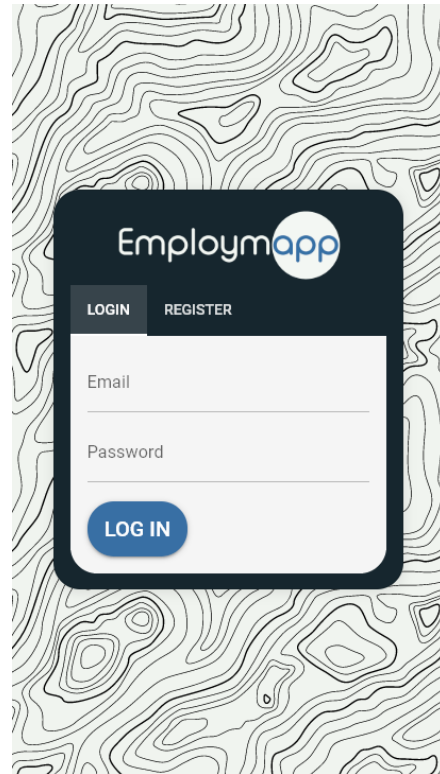


Figure 4.3 - Login Mobile Screen

The login screen, observed in *Figure 4.3*, represents a common portal for candidates and recruiters, filtering information and redirecting the user to his corresponding interface based on the introduced authentication data.

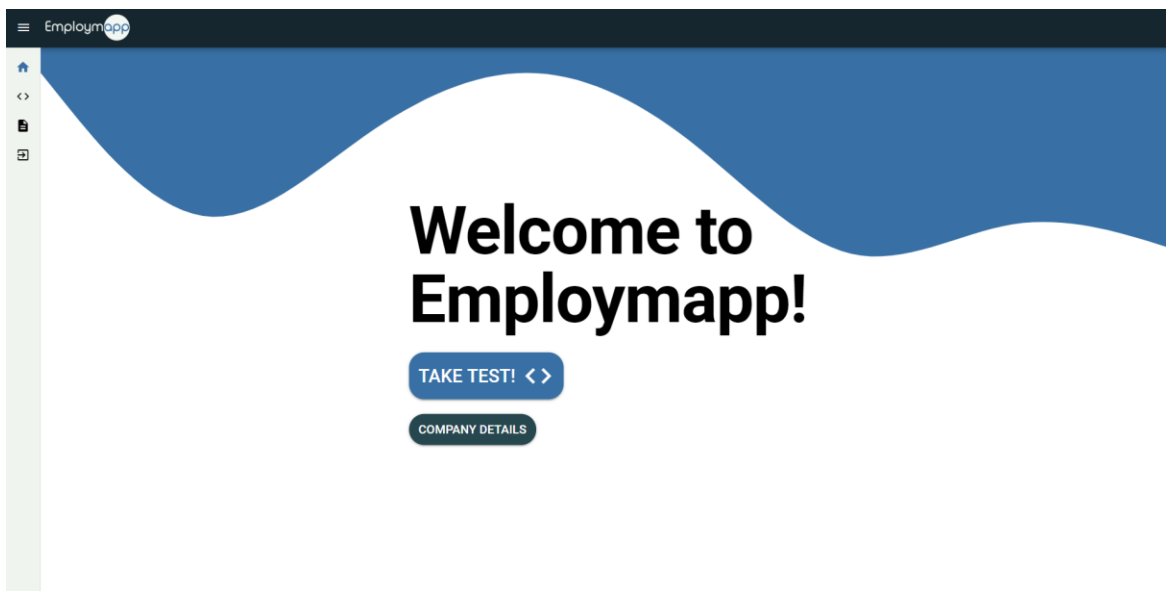


Figure 4.4 - Applicant Home Screen

The applicant is welcomed by a landing page, which gives him two options, the option to take and preview recruitment tests, and the option to review further company information. This screen was created for user experience purposes, so that the applicant is not hit with too much-unexpected information as soon as he passes authentication.

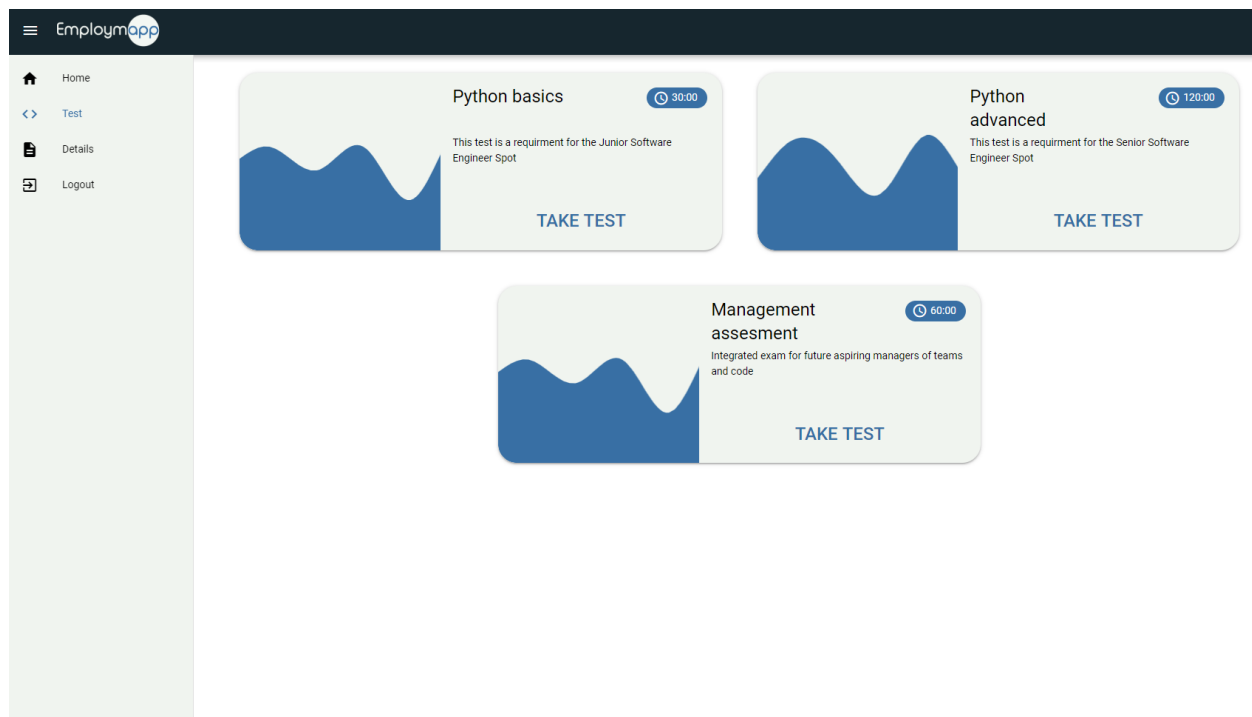


Figure 4. 5 -Applicant Tests Screen

Following the recruitment tests path, a list of all the available recruitment test is displayed, available in *Figure 4. 5*, alongside which their description and how much time the assessment will take. Once selecting a test, a preview screen is available, ensuring that the applicant will not start the test by mistake. If the selected test has already been attempted, a dialog box appears, which contains information about the results previously obtained, including when it has been taken and code and theoretical indicative grades. These results are available as soon as the test is completed.

As these types of cards are available through multiple pages of the application, they have been managed through a Vue component, which is shares to common characteristics of the cards. The flow through a v-for instruction and are intercepted through their corresponding store that makes the link with Back-End functions with the help of Axios.

The test has two metrics for measuring the test's progress through a timer and available steps. The timer will start counting down automatically once the examination is started, and it will register the responses as soon as it hits the zero mark. The stepper is implemented through a Quasar component, thus providing the precise completion stage, so the candidate knows how to manage his time limit.

Two types of questions are available inside Employmapp, the multiple-choice questions which have the purpose of evaluating theoretical questions, and code questions which have the purpose of evaluating practical challenges.



Figure 4.6 - Test Confirmation Screen

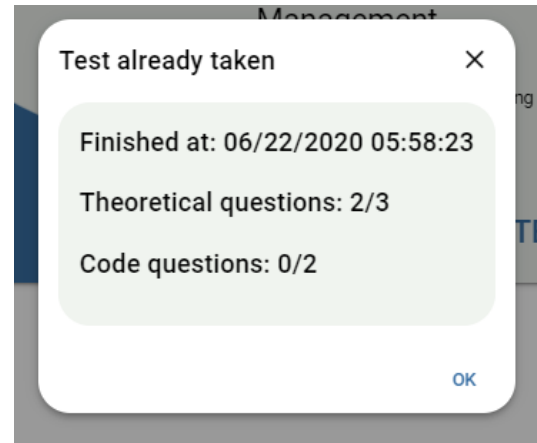


Figure 4.7 - Examination Results Dialog

In Figure 4.6 and Figure 4.7 are represented some implementation details that have a significant impact on user experience, further bringing forward the visual identity of the application through a bubbly and color-friendly design.

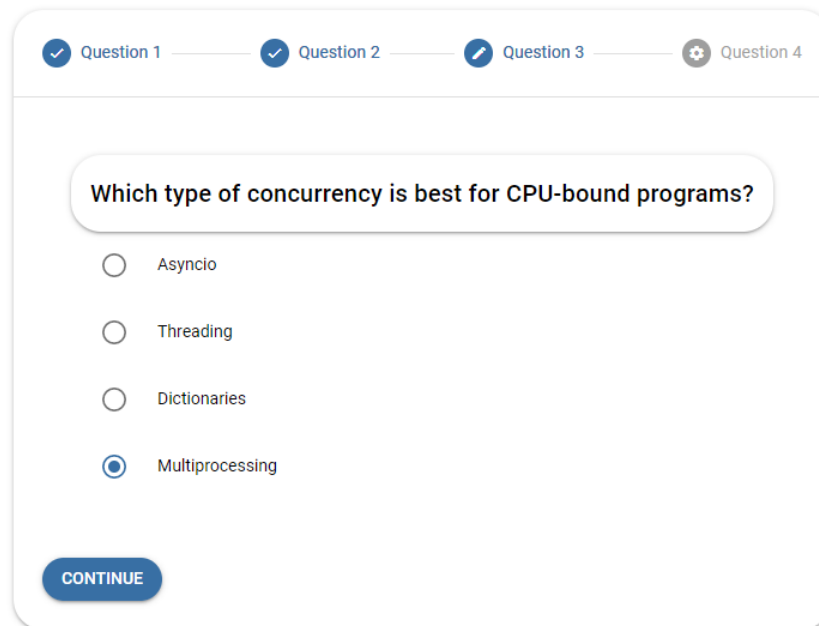


Figure 4.8 - Multiple-choice Question

Firstly, for the theoretical questions, simple radio buttons have been implemented. These track the applicant's response and add the question's number of points to the final result if the selected answer has the *isCorrect* attribute set to true.

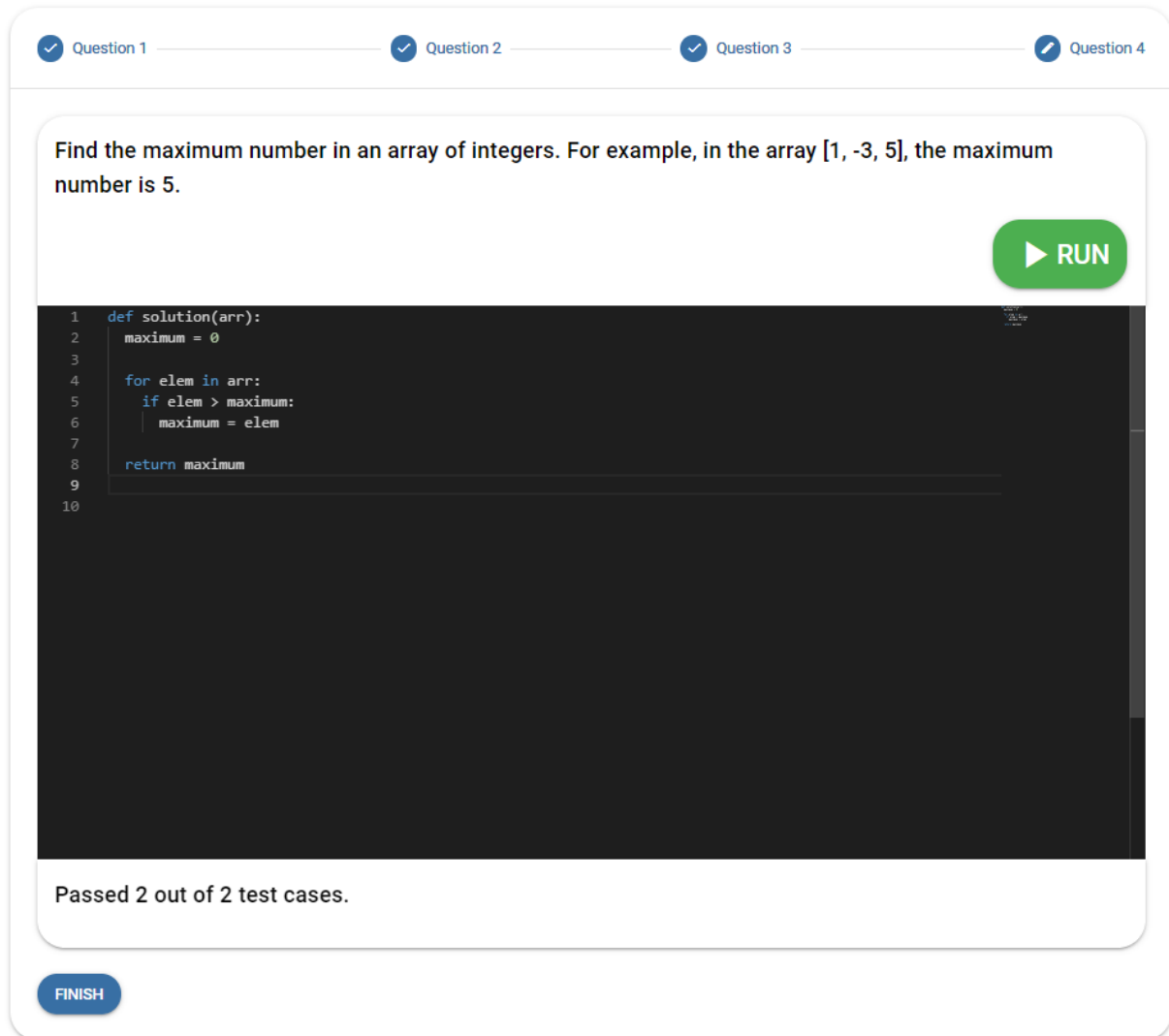


Figure 4. 9 - Code Question

The other type of questions, also the more relevant ones, are the code questions, presented in Figure 4. 9. These integrate a mini development environment, which poses a function that will have to be completed in order for the recruiter's unitary tests to pass, the points will be given per passed unit test.

Such a functionality was built from scratch, using only fundamental javascript management functions. Although the implementation could have been made using already more well developed APIs left by big corporations like HackerRank, these can be susceptible to change and can present significant access requirements that could require monetary costs, leaving the application unable to operate.

Present in the Communication Diagram, the design was meant to inflict some rules on how the process shall take place. The code should be tested multiple times, and communication with the server will be made continuously. In the server-side, an interpreter for the desired programming language lives that can examine the code through unitary tests written by the recruiter.

```

let sessionId = req.body.id;
let importInputCode = `from input_code${sessionId} import solution\n`;
let answer = req.body.answer;

let codeQuestion = await CodeQuestion.findOne({
  where: {
    id: req.body.codeQuestionId
  }
});

let test = importInputCode + codeQuestion.test;

fs.writeFileSync(path.join(__dirname, CODE_FOLDER, `input_code${sessionId}.py`), answer);
fs.writeFileSync(path.join(__dirname, CODE_FOLDER, `test${req.body.codeQuestionId}.py`), test);
const proc = exec("python " + path.join(CODE_FOLDER, `test${req.body.codeQuestionId}.py`));
const results = await proc.toString();

return res.send({
  results: results
});

```

Figure 4. 10 - Code Test Controller

In the code snippet above, the rudimental communication between user inputs and server-side can be addressed. There should be mentioned that this exemplifies the process only in a nutshell; the actual function has some additional elements, like error handling and regex expressions for extracting certain information from messages sent by the test cases.

Firstly, the candidate is identified with the help of cookie-sessions, which store his id. The applicant's response is saved in a variable named "answer" that holds the code intercepted from the Front-End input, and the test case is stored in the "test" variable. The test needs to know which file to import; thus, the import string is concatenated at the beginning of the recruiter's test case. After that, the response is saved into a file called "input_code," followed by the user's id, while the test is saved into a file that identifies the code question it corresponds to. The script is executed, and its output is collected in a "results" variable, which is sent back to the applicant's interface. This process is done with the possibility of Node's *exec* function, which runs a command at the console and returns its output. [18]

```

from input_code import solution
def get_test_cases():
    pass
def get_expected_outputs():
    pass
def test_code():
    pass
test_code()

```

Figure 4. 11 - Test Case Structure

Each of the test cases has the structure present in the *Figure 4. 11*. There are the test cases on which the applicant's code is executed against, followed by that, like the function's name suggests, their expected outputs, and lastly, the function that actually runs the applicant's code to the corresponding test cases written previously. [18]

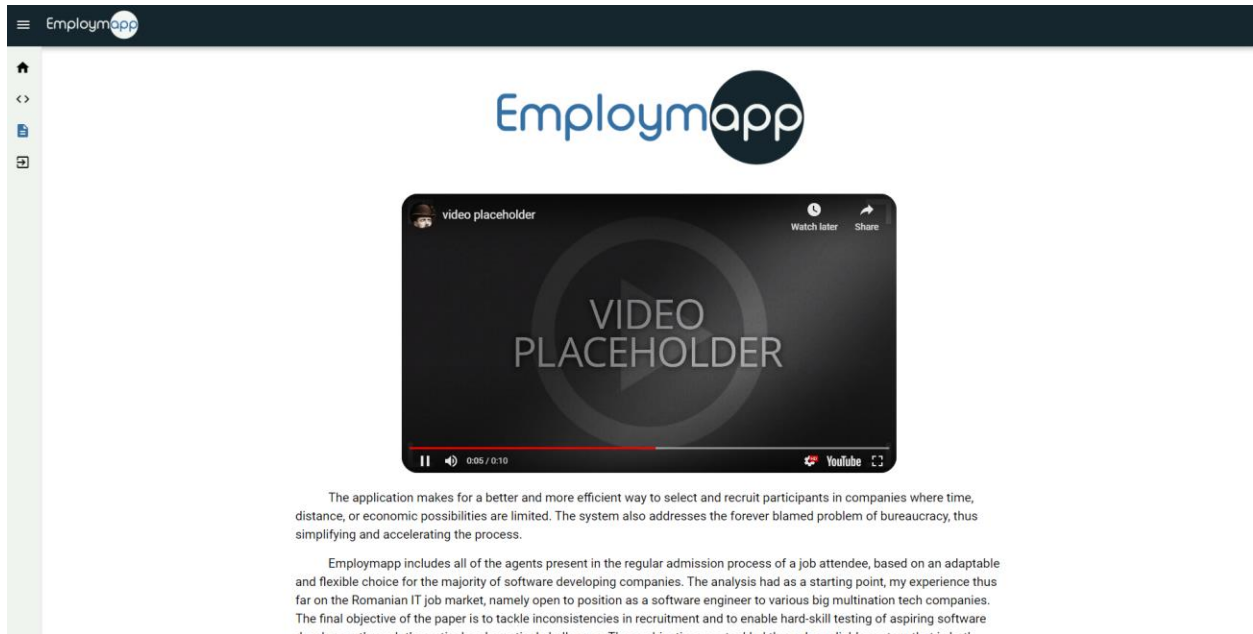


Figure 4.12 - Details Screen

As mentioned previously, there is also the page where additional company information can be found. This screen was implemented to demonstrate further ways that the application could be extended, this page could become quickly and interface where articles are posted and where the company can be further promoted. To illustrate further functionality, the page implements a multimedia insert in the form of a video.

4.2. Recruiter side

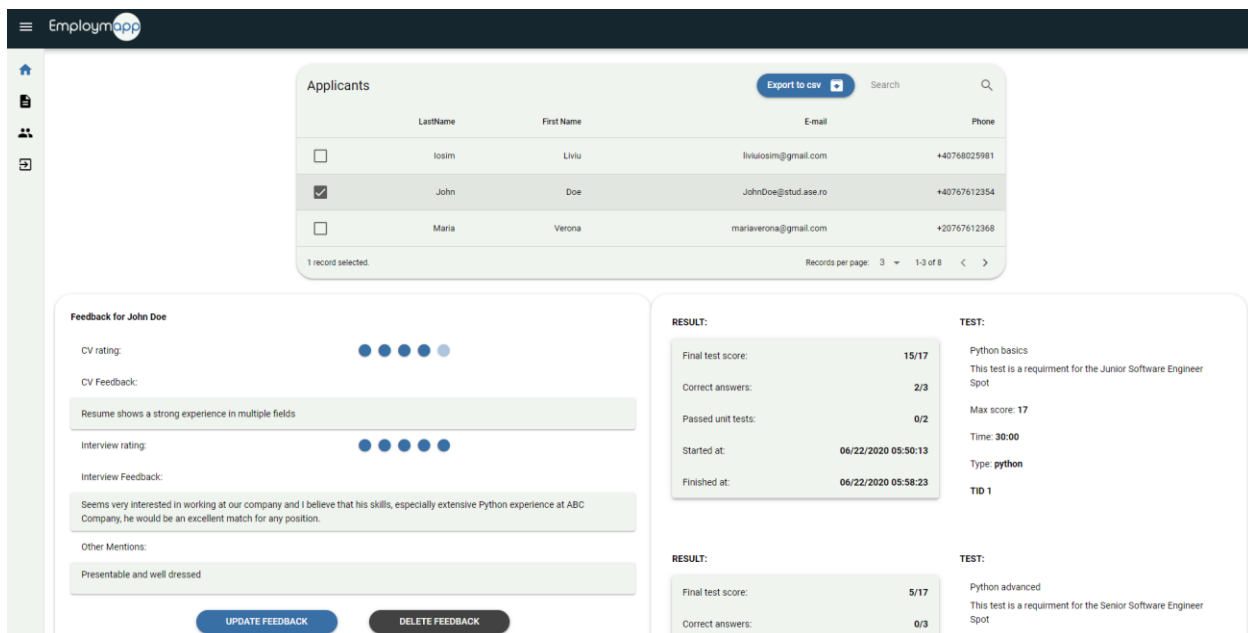


Figure 4.13 - Recruiter Home Screen

The recruiter account is opened with the applicants' table, which contains the contact information of all the candidates present in the system. This table is exportable to a CSV file and implements sorting and searching functionalities, also it is adjustable based on how many entries should be displayed. By selecting a row, and implicitly a participant, further information is displayed.

Firstly the recruiter can manage the feedback of an applicant after reviewing their resume and interview performance. They can register written information and give ratings based on a five points scale. The feedbacks implement CRUD functionalities through a form update and creation dialog. These, as implied, are also erasable.

Also, information the result obtained from each participant at each examination he has taken part in is displayed on the left side of Figure 4. 13. All the relevant fields are displayed, from the final score to the test's type and description.

Name	Duration(s)
Python basics	1800

Description
This test is a requirement for the Junior Software Engineer Spot.

Outlined
python

Which type of concurrency is best for CPU-bound programs?

Question	Score
Which type of concurrency is best for CPU-bound programs?	5
<input type="checkbox"/> Right Answer Answer Threading	
<input type="checkbox"/> Right Answer Answer Dictionaries	
<input checked="" type="checkbox"/> Right Answer Answer Multiprocessing	

ADD ANSWER REMOVE QUESTION

Figure 4. 14 - Test Creation Screen

The second screen the recruiter benefits from relates to test management. Here he has a list of all the available tests, similar to the one found on the applicant's account. The only difference is that here, he can delete them instead of taking them.

A huge step in giving the application sustainability is the test creation screen, present in Figure 4. 14. It contains all the fields required in the creation of a custom test and starts with the name, duration, description, and type. The description segment is englobed in a Textarea field to allow more substantial amounts of text to be more easily introduced.

The recruiter can insert as many multiple-choice or code questions as he pleases, as long as there is at least one question present, no matter its type. In the multiple-choice question's section, there can be selected the score of the question, along with its answers and their management, through a CRUD interface.

Question
Find the maximum number in an array of integers.\nFor example, in the array [1, -3, 5], the maximum number is 5.

Score per test
1

Total number of unit tests
2

Enter test case:

```

1  '''copy and paste your python test in here'''
2  def get_test_cases():
3      return {
4          'SMALL_INPUT': [1, 2, 3],
5          'LARGE_INPUT': [1, 2, 3] * 1000 + [4],
6      }
7
8  def get_expected_outputs():
9      return {
10         'SMALL_INPUT': 3,
11         'LARGE_INPUT': 4,
12     }
13
14  def test_code():
15      test_cases = get_test_cases()
16      expected = get_expected_outputs()

```

Enter user start input:

```

1  '''function(s) signature(s)'''
2  def solution(arr):
3

```

REMOVE CODE QUESTION

Figure 4.15 - Code Question Creation

The code question creation implies increased awareness. After introducing the number of implemented unitary tests and the points given per each of them, two necessary fields can be observed. In the first one, through a Monaco editor interface the recruiter can include the test case which will be sent and written to a file only if there exists an applicant which has attempted the test, the latter stands to complete with what code the user starts with when attempting the challenge, as the previously entered unitary tests depend on this function's signature.

Looking at more technical aspects of the situation, in the creation stage, all the inserted information is building up an object inside the state of the application. After the recruiter is satisfied and manages to complete all the mandatory fields, the instance is sent to the database persisting in the system on all the different accounts. There is also the option to reset a test during its creation, erasing all the information in the completed fields. This is useful in cases where more general mistakes have occurred.

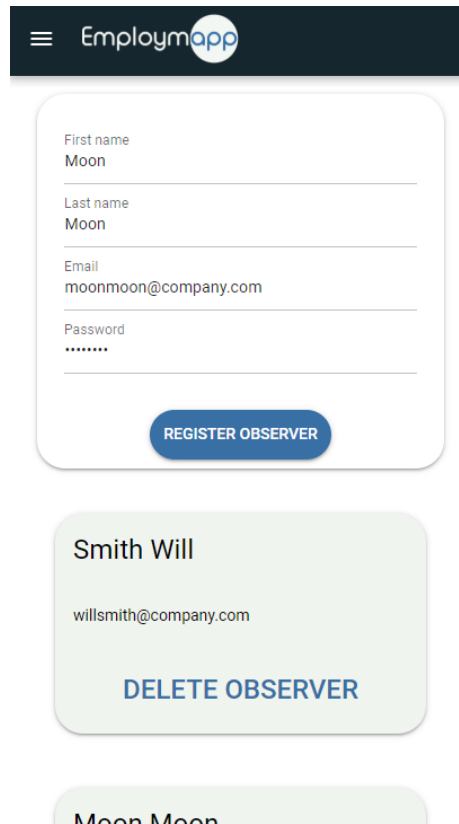
The image shows a mobile application interface for managing observers. At the top is a dark blue header with a hamburger menu icon and the 'Employmapp' logo. Below the header is a registration form with fields for 'First name' (Moon), 'Last name' (Moon), 'Email' (moonmoon@company.com), and 'Password' (represented by dots). A blue button labeled 'REGISTER OBSERVER' is at the bottom of the form. Below the form is a card for an existing observer, 'Smith Will', with email 'willsmith@company.com' and a blue button labeled 'DELETE OBSERVER'. At the bottom, a partially visible card for 'Moon Moon' is shown.

Figure 4. 16 – Observers Management Screen

The aspect that distinguishes the administrator's account from the observer's one is represented by the ability to register new recruiters in the system. This occurs through a similar interface to the tests management screen, including the same kind of cards and a registration form. There is also the ability to remove an observer, deleting their account.

Figure 4. 16 displays the mobile interface of the functionality. For every aspect previously presented in this chapter, there is a responsive interface that enables use on phones, tablets, and desktops. The only exception to this rule is the recruitment tests from the applicant's account, as taking a code related challenge on such a small screen would represent unfit conditions for a proper evaluation.

Conclusions

Employmapp application, or more formally addressed, application for supporting the recruitment and selection processes in an IT company, represents the final product of the process discussed in the previous chapters. It is ultimately found in an optimal operating state, the idea behind the system targeting only automation-related problems.

From my perspective, this information system can represent a solid stepping stone in creating a reliable recruiting tool in the IT job market. It implements a solution for applicant data centralization while also allowing a more complex form of assessment through practical challenges. All of that is presented through a friendly interface, with a bubbly visual identity and pleasant color schemes. The main benefits, besides performance and the modern design, are that it saves much time and monetary investments through its automation, things that inevitably make it attractive in the eyes of company managers. It also betters the external image of the recruiter agent in the eyes of applicants, diminishing possible uncertainties and making it into a robust public relations tool.

Looking at a macro aspect of usability, this application can become a staple in the recruitment world of start-ups that do not have the necessary resources to organize extensive recruitment processes. In this respect, Employmapp application, subsequent to the presentation of the present thesis, will be published in an open-source regime, under an MIT license that permits anyone to access, modify and share the source code without the influence of a coercive legal entity.

Even though the presented paper comes to an end, the system, although in a stable state, it is far from meeting its final form. Following the popular concept, *a software product is never finished, only released* [19], some further implementation directions have come to mind that could improve the net value of the application, with regards to security and additional functionalities.

The first development direction that could be addressed is the addition of supplementary security measures, to ensure that the server-side ecosystem does not get interrupted by malicious code written by the participant. One solution for this could be represented by using tools such as Docker, creating a new container for every active user session that would hold an instance of the system. The communication could be done through automated services like Jenkins that could push and pull the information from the container. In this solution, if a user destroys his instance, the application is still functional, and the server environment is not affected. Taking things a step further, an RBAC (Role-Based Access Control) could be implemented in order to manage user's rights upon the system. Different roles can be composed, with different rights, so that each entity can have different role that dictates its permissions and limitations.

The second aspect that could be further improved is the number of possible types for practical challenges. Right now, the system only implements the solution for interpreted programming languages like Python and Javascript. Changing the development operating system from Windows to a Linux distribution like Ubuntu could enable a lot more liberty in the management of compilers, and CLI commands, thus quickly enabling compatibility with compiled programming languages like C++ and Java.

The third development path refers to the maintenance and troubleshooting process of the system, which, in the ecosystem of more complex solutions, can take up much time. The most reliable technology that mitigates log management sent by the application's instances would be the ELK Stack that implements an engine for analytics search along with server-side data processing from multiple sources, further embedding a view tool for charts and graphs. [20]

Adding these auxiliary functionalities and solutions to an already implemented robust base of the Employmapp application could represent a significant step in the completion of the information system and its transformation into a reliable, complete, and complex software product independent of the environment in which it finds itself in.

Table of Figures

Figure 1. 1 - Recruiter Use Case Diagram.....	9
Figure 1. 2 - Applicant Use Case Diagram.....	10
Figure 1. 3 - Registration Activity Diagram	11
Figure 1. 4 - Feedback Management Activity Diagram	12
Figure 1. 5 - Test Creation Activity Diagram.....	13
Figure 1. 6 - Code Question Communication Diagram.....	13
Figure 1. 7 - Feedback Management Interaction Diagram.....	14
Figure 1. 8 - Test Creation Interaction Diagram.....	15
Figure 1. 9 - Test Attempt Interaction Diagram.....	16
Figure 2. 1 - Detailed Class Diagram	17
Figure 2. 2 - Database Schema.....	18
Figure 2. 3 - Applicant Graphical Interface Schema	20
Figure 2. 4 - Recruiter Graphical Interface Schema	20
Figure 2. 5 - Component Diagram	21
Figure 2. 6 - Deployment Diagram	22
Figure 3. 1 - Front-End Package.json.....	23
Figure 3. 2 - Authentication Controller.....	26
Figure 3. 3 - Test Routes.....	27
Figure 4. 1 - Registration screen.....	31
Figure 4. 2 - Register Mobile Screen, Figure 4. 3 - Login Mobile Screen.....	32
Figure 4. 4 - Applicant Home Screen	32
Figure 4. 5 - Applicant Tests Screen	33
Figure 4. 6 - Test Confirmation Screen, Figure 4. 7 - Examination Results Dialog.....	34
Figure 4. 8 - Multiple-choice Question.....	34
Figure 4. 9 - Code Question.....	35
Figure 4. 10 - Code Test Controller.....	36
Figure 4. 11 - Test Case Structure.....	36
Figure 4. 12 - Details Screen.....	37
Figure 4. 13 - Recruiter Home Screen	37
Figure 4. 14 - Test Creation Screen.....	38
Figure 4. 15 - Code Question Creation	39
Figure 4. 16 – Observers Management Screen.....	40

Bibliography

- [1] I. H. Buchen, "The HR-Tech Interface," in *Partnership HR: New Norms for Effective Recruitment, Performance, and Training of Today's Workforce*, Davies-Black Publishing, 2007, pp. 25-33.
- [2] Capterra, "Pre-employment Testing Software," Capterra, [Online]. Available: <https://www.capterra.com/pre-employment-testing-software/#buyers-guide>. [Accessed June 2020].
- [3] J. Hadzima, "How Much Does An Employee Cost?," *Boston Business Journal*, p. 3.
- [4] M. Skarabo, "Business Process Model and Notation," Wikipedia, 18 June 2020. [Online]. Available: https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation. [Accessed June 2020].
- [5] R. Stupay, "Learn about class diagrams," Openclassrooms, 3 June 2020. [Online]. Available: <https://openclassrooms.com/en/courses/4191736-design-a-database-with-uml/4191743-learn-about-class-diagrams>. [Accessed June 2020].
- [6] G. Bartlett, "Component diagram," Wikipedia, 30 September 2019. [Online]. Available: https://en.wikipedia.org/wiki/Component_diagram. [Accessed June 2020].
- [7] K. Finley, "What Exactly Is GitHub Anyway?," TechCrunch, 14 July 2012. [Online]. Available: <https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/>. [Accessed June 2020].
- [8] "What is REST?," Codecademy, n.d.. [Online]. Available: <https://www.codecademy.com/articles/what-is-rest>. [Accessed June 2020].
- [9] E. Elliott, "Introduction to Node & Express," Medium, 16 November 2016. [Online]. Available: <https://medium.com/javascript-scene/introduction-to-node-express-90c431f9e6fd>. [Accessed June 2020].
- [10] "Using middleware," Express, n.d.. [Online]. Available: <https://expressjs.com/en/guide/using-middleware.html>. [Accessed June 2020].
- [11] M. contributors, "Express Tutorial Part 4: Routes and controllers," MDN, 18 May 2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes. [Accessed June 2020].

- [12] "Routing," Express, n.d.. [Online]. Available: <https://expressjs.com/en/guide/routing.html>. [Accessed June 2020].
- [13] D. Kumar, "Advantages of building applications with Vue.js," gitconnected, 3 January 2019. [Online]. Available: <https://levelup.gitconnected.com/advantages-of-building-applications-with-vue-js-b2a23e4199c8>. [Accessed June 2020].
- [14] "What is Vuex?," Vue.js, n.d.. [Online]. Available: <https://vuex.vuejs.org/>. [Accessed June 2020].
- [15] R. Stoenescu, "Why every Vue developer should be excited by Quasar 1.0," Quasar Framework, 3 July 2019. [Online]. Available: <https://medium.com/quasar-framework/quasar-1-0-4bc696d60c1b>. [Accessed June 2020].
- [16] "Vue Router," Vue.js, n.d.. [Online]. Available: <https://router.vuejs.org/>. [Accessed June 2020].
- [17] E. Kollegger, "What is Axios.js and why should I care?," Medium, 14 May 2018. [Online]. Available: <https://medium.com/@MinimalGhost/what-is-axios-js-and-why-should-i-care-7eb72b111dc0>. [Accessed June 2020].
- [18] E. Ehiorobo, "BetterProgramming," MediumHow To Create a Web App for Coding Tests, 14 November 2019. [Online]. Available: <https://medium.com/better-programming/how-to-create-a-web-app-for-coding-tests-750ef97bf1e3>. [Accessed June 2020].
- [19] M. Rodriguez, "Software is never finished, only released," Hacker Noon, 27 February 2016. [Online]. Available: <https://hackernoon.com/software-is-never-finished-only-released-a026222176f7>. [Accessed June 2020].
- [20] Elasticsearch, "What is the ELK Stack?," elastic, n.d.. [Online]. Available: <https://www.elastic.co/what-is/elk-stack>. [Accessed June 2020].