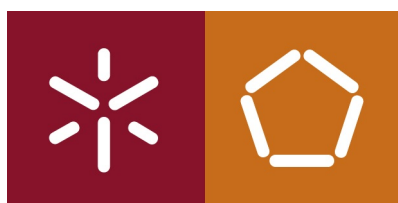


UNIVERSITY OF MINHO

INTEGRATED MASTER IN INFORMATICS ENGINEERING



TuiChain

PROJETO EM ENGENHARIA INFORMÁTICA

Alberto Faria (a79077)
Alexandru Domente (pg41063)
Henrique Pereira (a80261)
João Silva (a82005)
Nelson Sousa (a82053)
Pedro Moreira (a82364)
Pedro Ferreira (a81135)
Ricardo Caçador (a81064)
Ricardo Milhazes (a81919)
Rui Ribeiro (a80207)
Tiago Sousa (a81922)

December 2020

Contents

1	The Purpose of the Project	1
1.1	Context	1
1.2	Goals of the Project	1
2	The Stakeholders	3
2.1	The Client	3
2.2	The Customer	3
2.3	Other Stakeholders	3
2.3.1	Banks	3
2.3.2	<i>Fundação José Neves</i>	4
2.3.3	Lambda School	4
2.3.4	Developers, Software Engineers and Infrastructures	4
2.3.5	<i>Comissão Nacional de Proteção de Dados</i>	4
3	Nomenclature and definition conventions	5
4	Functional Requirements	6
5	Non-Functional Requirements	12
5.1	Look and Feel Requirements	12
5.2	Usability and Humanity Requirements	12
5.3	Performance Requirements	13
5.4	Operational and Environmental Requirements	14
5.5	Maintainability and Support Requirements	14
6	Application Structure	15
7	Design Models	15
8	Frontend	15
8.1	Research	15
8.2	UI/UX	22
9	Backend	26
9.1	Research	26
9.2	App Development	31
10	Blockchain	34
10.1	Cryptocurrency platforms and tooling	34
10.2	Design and Specification	35

10.3 Implementation	41
11 Deployment and Maintenance	42

1 The Purpose of the Project

1.1 Context

The lack of monetary possessions is one of the main obstacles to higher academic education and professional specialization. The cost of higher education in a country like the United States of America is around 100,000 dollars [1]. In view of these exorbitant figures, it is not expected that everyone will be able to bear such a financial burden. In addition, there are several underdeveloped countries that lack social programs to support education, making access to higher education even more difficult. Despite the cost, higher education courses are becoming increasingly important for entering the labor market. Companies are increasingly looking for people who are highly specialized in particular technical and scientific fields.

To cover the costs associated with education, students traditionally resort to loans from banks, to which they will have to pay an associated interest, possibly entering credit spirals (successive loans to pay off previous ones). Finally, many students are unable to borrow from these institutions because they represent a high risk investment, *i.e.*, students are often disregarded.

Currently, there exist institutions that provide education financing through Income Share Agreements (ISAs). An example in Portugal is the *Fundação José Neves* [2], which establishes contracts with students so that their training can be financed. In return, the students, when entering the job market, pay a percentage of their salary for a certain number of years, in order to cover the financed amount. However, in this model, the capital available for student funding is limited to the capital of the institution originally providing the funds.

1.2 Goals of the Project

With this in mind, we propose a platform based on blockchain technology, which brings together lenders and students who need financing. Through this platform, students should be able to publish requests for funding, subject to the usual conditions of an ISA. Any individual may browse the platform for such requests and decide to finance, in part or in full, the tuition costs of one or more students.

The platform must ensure that when funding is raised for the student's entire application, the money is transferred to the student and the lenders receive tokens representing the student's debt (the amount of tokens is proportional to the fraction financed). After the student makes the final payment (as stipulated by the conditions agreed upon at the time of the request's publishing), the full returned amount is transferred to the holders of the tokens corresponding to the debt in question, in proportion to the amount of tokens owned.

These tokens should also be able to be traded between potential stakeholders,

thus forming a secondary market where it is possible to announce the intention to sell/purchase tokens. In this case, the value of the token will be defined by the market, through the existing supply and demand. In addition, revenue may be generated from the application of fees on these transactions. Compared to existing exchanges, the platform should have the advantage of providing specialized search functionalities, in order to enable the identification of more or less attractive offers (+/- risk, +/- reward), according to the desire of each user.

2 The Stakeholders

In this section, we describe the people who have interest in the product, which are commonly called the *stakeholders*. Stakeholders are not only individuals or institutions that intend to invest capital in the product — all those who intend to use or otherwise take advantage of the product can be considered stakeholders.

2.1 The Client

Since the client is defined as someone who makes an investment in the product, there is no one who respects this definition as well as our own working group, in partnership, and with *Subvisual*'s mentoring.

After well defining our motivations, already described in the first section, we decided to invest in the project and create the desired platform.

2.2 The Customer

The customer is defined as someone who pays to acquire the system when it is ready to be used. Given this notion it becomes clear who are the customers of this project, not least because they coincide with the ultimate consumers and users of the product. They are the *students* and the *investors*.

This kind of users, don't necessarily pay to acquire the system, but, by using it, they will be paying because that's how the system works. It needs these two types of users to work properly. Students that need funding, and investors who lend them the money they need.

2.3 Other Stakeholders

All other stakeholders with a direct or indirect interest in the product will be listed below. It will be presented the knowledge they can provide to the team and how they will be affected by the product.

2.3.1 Banks

Banks are the first competitors of our product, since they are the ones we want to replace in the task of lending money to students for their higher education courses. They apply high interest rates, and require monthly payment from the moment the loan is made. The problem is that students do not have the money to pay for the education they need.

They'll be affected by our product because they will lose a portion of their usual student clients to our product, which offers a solution that fits them better.

2.3.2 *Fundação José Neves*

This foundation [2] was created by a portuguese philanthropist, who had the idea of improving the education of the portuguese population. They use ISAs, which are agreed with the students, fighting one of the banks disadvantages. This foundation has a problem, though. They established a ceiling for funding, and they don't fund everyone who need, instead they choose which students to fund. Besides, this foundation is the only investor, not allowing anyone else to invest.

In this way they'll be affected by our product because the students that are not funded will use our product. The people who wants to give a little contribute and invest in some student education will be able to.

2.3.3 *Lambda School*

This institution [3] also use ISAs, which are agreed with the students, where all have the same value (30,000\$). Also, to be able to use this ISAs you have to be either a US citizen, a US permanent resident or a DACA (Deferred Action for Childhood Arrivals) recipient, which by itself is a disadvantage. They only offer courses that exist in the organization (Data Science and Full Stack Web), which is not an accredited institution and does not offer a degree but a certificate of completion.

By expanding this product to other countries, namely the USA, we will be giving a real opportunity to these students that may want to study in a accredited university or get, for example, the PhD that they really want. Consequently, a significant part of these students may want to use our product instead of seeking an ISA through Lambda Schools.

2.3.4 *Developers, Software Engineers and Infrastructures*

It's expected that the team where these elements are integrated have the skills necessary for the development of the product, such as, know object-oriented programming, manage a project and test software.

The development of the product will promote the creation of new job opportunities in the area of software development.

2.3.5 *Comissão Nacional de Proteção de Dados*

The product that is being developed must comply to the norms of CNPD for the management and use of user data gathered.

3 Nomenclature and definition conventions

Students People who need funding to increase their degrees, including doctorates, MBAs, bachelor's degree, post-graduations and others.

Tuition The needed amount to get a degree.

Investors People who contribute to the debts of students in order to make a profit.

Blockchain Distributed ledger technology that allows data to be stored globally on thousands of peers, while letting anyone on the network see everyone else's entries in near real-time. No one can control or own the data.

Ethereum Decentralised open source blockchain network with smart contracts functionality, allowing users to create their own programs on top of the network infrastructure.

Smart Contract Piece of code that specifies a set of rules to manage transactions in a blockchain environment.

ISA An income share agreement (or ISA) is a financial structure in which an individual or organization provides something of value (often a fixed amount of money) to a recipient who, in exchange, agrees to pay back a percentage of their income for a fixed number of years.

Secondary market A place where investors can resell their tokens to another investor in the hope of making some profit or avoiding losing money.

Token Digital asset that represents a portion of a student's debt. The student has legal obligation to pay each part of his debt to each token owner.

4 Functional Requirements

As mentioned before, there are two types of users, each with specific needs and objectives. The functional requirements for students are presented below, followed by the requirements of investors. The Volere Requirements Specification was used to describe each requirement.

Req. n^o: F01 Type: 9

Description: As either a student or investor, I want to register on the platform so that I can be identified and have exclusive access to personal data.

Acceptance Criteria: Given a register form, when I provide my name, age, address, email, password, ID and a photograph, then I'm registered on the system.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F01-b Type: 9

Description: As either a student or investor, I want to register on the platform via external services (ex: Google), so that I can be identified and have exclusive access to personal data.

Acceptance Criteria: Given a service prompt and I successfully login on it, when I provide additional information such as my name, age, address, ID and a photograph, then I'm registered on the system.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F02

Type: 9

Description: As either a student or investor, I want to login on the platform so that I can access my personal data.

Acceptance Criteria: Given a login form, when I provide my email and password, then I'm authenticated in the system.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F02-b

Type: 9

Description: As either a student or investor, I want to login on the platform via external services (ex: Google) if I previously registered with it.

Acceptance Criteria: Given a service prompt, when I successfully login on it, then I'm authenticated in the system.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F03

Type: 9

Description: As a student I want to register a request for funding so that I can be financed.

Acceptance Criteria: Given a funding form and I'm logged in, when I provide my CV, a letter of motivation, the institution I want to study, the course I intend to take and the cost of its tuition, then I can create a request for funding.

Originator: Brainstorming

Priority: *Must have*

Req. n°: F04 Type: 9

Description: As a student I need to sign a contract so that the loan can be materialized.

Acceptance Criteria: Given a contract issued by the company, when we both sign the contract, then the contract is materialized.

Originator: Brainstorming

Priority: *Must have*

Req. n°: F05 Type: 9

Description: As a student I want to be able to change my professional status as well as my income so I can pay or suspend my loan according to those information.

Acceptance Criteria: Given that my professional status or income changes, when I'm available to change that information, then I'm able to modify my profile with the correct information

Originator: Brainstorming

Priority: *Must have*

Req. n°: F06 Type: 9

Description: As a student I want to pay back to the various investors who hold parts of my debt so that I can fulfill the contract I signed.

Acceptance Criteria: Given that I'm working, then I'm able to pay back my debt, so I can pay off my debt regularly.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F07

Type: 9

Description: As an investor, I want to login with my virtual wallet so that I can invest in students.

Acceptance Criteria: Given I'm logged in, when I introduce my wallet information, then I can start using my funds.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F08

Type: 9

Description: As an investor I want to buy a portion or all of a student's tuition so that I can help her/him get the education (s)he needs, and hope to profit later.

Acceptance Criteria: Given my wallet has funds, when I buy student tokens, then I can manage them as I see fit.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F09

Type: 9

Description: As an investor, I want to cancel my financing, so that I can get my funds back.

Acceptance Criteria: Given a funding is not fulfilled, when I cancel my contribution, then I retrieve the funds I've invested.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F10

Type: 9

Description: As an investor, I want to sell tokens I have so that I can have some profit

Acceptance Criteria: Given I have tokens, when I sell them, then I receive the corresponding value.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F11

Type: 9

Description: As an investor, I want to see all students asking for financing, so that I can fund them.

Acceptance Criteria: Given I have money to invest, when I search for students asking for financing, then I can fund them

Originator: Brainstorming

Priority: *Must have*

Req. n^o: F12

Type: 9

Description: As an investor, I want to access a student's profile, so that I can evaluate and fund her/him.

Acceptance Criteria: Given I want to invest in a student, when I search for students asking for funding, then I can check their profile

Originator: Brainstorming

Priority: *Must have*

Req. n^o: **F13**

Type: **9**

Description: As an investor, I want to manage my portfolio

Acceptance Criteria: Given I want to check any of my investments,
when I access my profile, then I can check my portfolio

Originator: Brainstorming

Priority: *Must have*

As a side note, the Hierarchical Task Analysis presented in Section 8.1.2 had an impact when gathering these requirements.

5 Non-Functional Requirements

5.1 Look and Feel Requirements

Req. n^o: NF01

Type: 10

Description: The application shall have a simple and elegant feel, in order to simplify the user experience.

Fit Criterion: A sample of representative users shall be able to use the product effectively after 20 minutes of use.

Originator: Brainstorming

Priority: *Must have*

5.2 Usability and Humanity Requirements

Req. n^o: NF02

Type: 11

Description: The application shall be easy to use for people who have basic education and have some e-currency experience

Fit Criterion: 90% of a test panel should be able to successfully complete given list of tasks within 30 minutes.

Originator: Brainstorming

Priority: *Must have*

5.3 Performance Requirements

5.3.1 Capacity Requirements

Req. n^o: NF01

Type: 12

Description: The application shall be able to handle large customer loads.

Fit Criterion: The application will have to handle hundreds of client requests per minute

Originator: Brainstorming

Priority: *Must have*

5.3.2 Scalability or Extensibility Requirements

Req. n^o: NF01

Type: 12

Description: The application shall be able to support an increase in the number of customers.

Fit Criterion: If the platform shows a daily/monthly/annual load increase, then it should be prepared to tolerate it.

Originator: Brainstorming

Priority: *Must have*

5.4 Operational and Environmental Requirements

Req. n^o: NF01

Type: 13

Description: The application shall be able to work in any web browser.

Fit Criterion: Given any browser, when the website is accessed, then the interfaces load correctly.

Originator: Brainstorming

Priority: *Must have*

5.5 Maintainability and Support Requirements

Req. n^o: NF01

Type: 14

Description: The application shall be modular in order to guarantee a more accessible maintenance.

Fit Criterion: The application shall be divided in layers, creating boundaries between interfaces, logic and services.

Originator: Brainstorming

Priority: *Must have*

Req. n^o: NF01

Type: 14

Description: The application source code must be well documented.

Fit Criterion: The source code of the application must provide at least 1 comment for each function / method developed that explains the manipulated variables and values.

Originator: Brainstorming

Priority: *Must have*

6 Application Structure

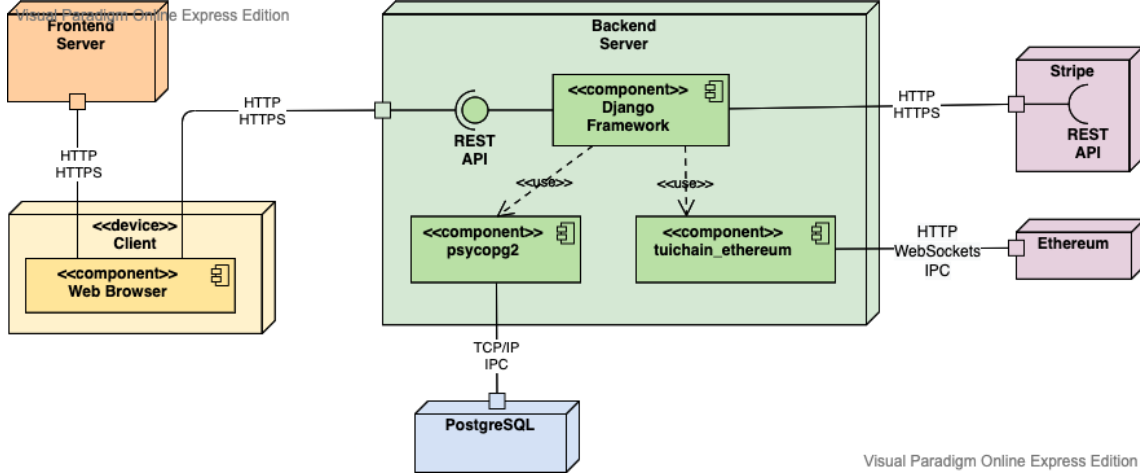


Figure 1: Application Structure.

7 Design Models

8 Frontend

The development of the interface was divided in two parts: the first involved a research process with subsequent Hierarchical Task Analysis (HTA) [4] and prototypes, while the second was the development of the user interface itself. This chapter describes the process mentioned before on Section 8.1 and the best practices used for developing a good UI/UX in Section 8.2.

8.1 Research

8.1.1 Personas

In order to know our ideal users' backgrounds, their goals, and their challenges, the idea of *personas* arises.

They are essential to create an effective strategy to attract, understand our users better, and make it easier for us to tailor content to the specific needs, behaviors, and concerns of different types of users.

Based on our survey we've created four personas, presented below:

Jim Parsons



Background

Jim comes from a humble family, in the suburbs of London. From a young age, he always loved math. In fact, going to the store with his mother to make the groceries total amount, really thrilled him. Over his education path, he was always one of the best students in his class. At college, he was awarded an honorary scholarship, which aimed to finance the education of underprivileged students. He graduated with first class honors and is, now, the financial planner of a million-dollar multinational company. Analyzing his path, Jim realizes that the scholarship he received was fundamental to his current economic well-being. In that way, he would like to repay the opportunity he was given and support students who are in the same situation he was.

Goals and Objectives

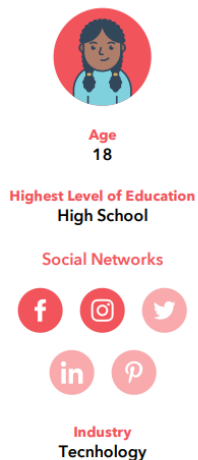
- Invest some of his savings
- Help students who can't afford tuitions

Biggest Challenges

- Doesn't have enough money to join philanthropist programs
- Lack of platforms to finance student grants
- Little information about students that need help paying their tuitions

Figure 2: Persona 1

Mayim Bialik



Background

Mayim lives in Senegal and comes from a big and needy family, being the oldest of 6 siblings. She has grown up helping her family, sometimes putting her own wishes aside. However, she has an objective that she does not want to give up: take a college course. For that, she would need bank financing, which will be hard to obtain, given her family's status. Despite the difficulties her family faces, Mayim has access to a smartphone and community internet. Therefore, she can apply for a loan, through TuiChain.

Goals and Objectives

- Get a Bachelor's degree in Computer Science
- Have a stable and well paid job
- Build a family without the troubles her parents had

Biggest Challenges

- Poor financial services in her country
- Access to universities
- Banking institutions won't give her a loan

Figure 3: Persona 2



Figure 4: Persona 3

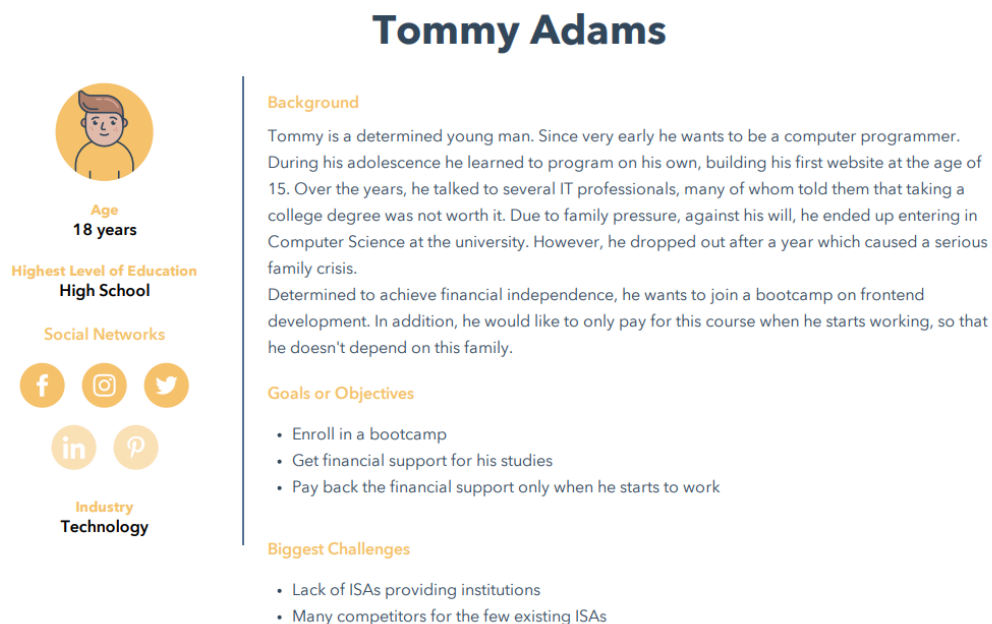


Figure 5: Persona 4

8.1.2 HTA

A task [4] is the human activity that makes it possible to achieve a goal. Analysing possible tasks, makes it is possible to understand the behavior of our users in the context of our system, giving us the information needed in order to develop the best UI/UX possible.

There is a big number of tasks a user can perform in our system, but we only present the two that are the most important: publishing a loan request (Figure 6) and financing a student (Figure 7).

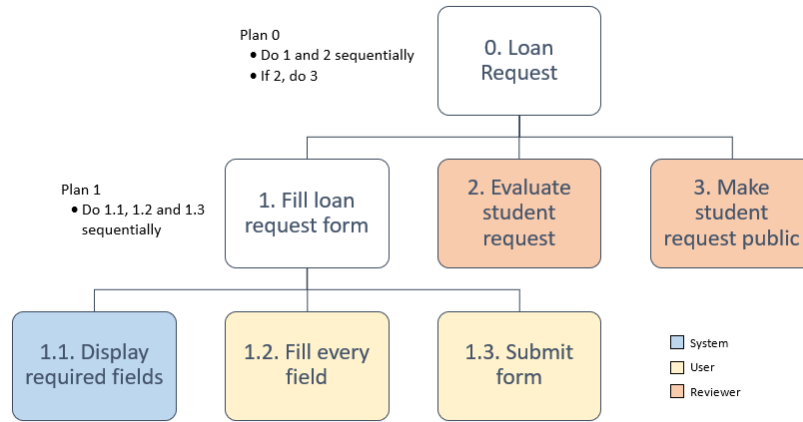


Figure 6: Loan request (HTA)

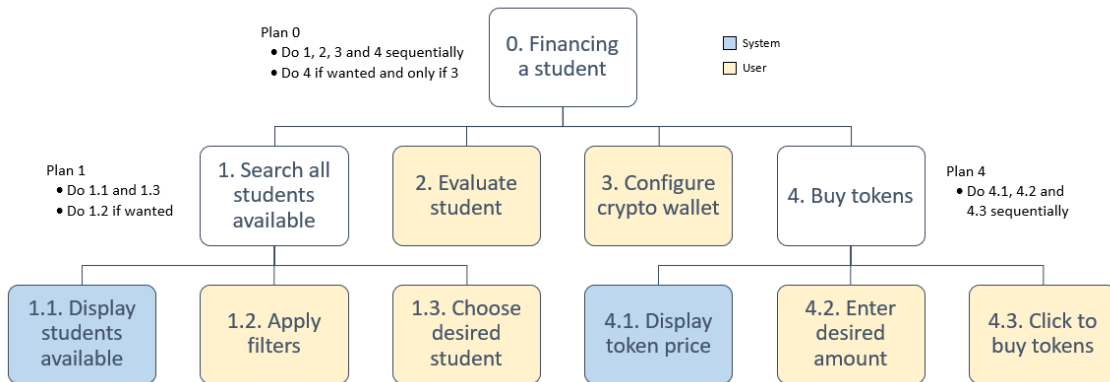


Figure 7: Financing a student (HTA)

8.1.3 Prototypes


The purpose of prototypes is to have a low-budget look & feel of the application we want to develop, so that the design can be validated and confronted with personas and potential users, and the HTA.

With this in mind, we designed the main pillars of our application, to better express the idea and objectives of our project. From Figure 8 to Figure 13 are, respectively, the following prototypes:


- Student signup
- Student loan request
- Catalogue of students asking for a loan
- A student page (with loan in progress)
- Secondary market
- A student page (with load finished)

The image shows a web form for student sign-up. At the top, there is a teal header bar with a graduation cap icon on the left, an 'APPLY' button in the center, and a hamburger menu icon on the right. Below the header, the word 'Student' is displayed in a large, bold, black font. The form itself is a light gray box containing several input fields: 'Full Name' with the value 'John Doe', 'Phone Number' with the value '999-999-99' and a clear button (X), 'Birth Date', 'Nationality', and 'Documents'. Below these fields is a checkbox with a green checkmark and the text 'I accept the Terms & Conditions'. At the bottom of the form are two buttons: a dark gray 'Cancel' button and a teal 'Sign Up' button.

Figure 8: Sign Up



APPLY



Loan

University
University of Minho

Degree
PhD

Course

Tuition


Years

☒ I accept the Terms & Conditions

Cancel


Apply

Figure 9: Loan request



STUDENTS

MARKET



Students

Search...

Filters

Country

Portugal

Degree

☒ PhD

☐ Bachelor

Course

☒ Software Engineering


☐ Biomedical Engineering

☐ Art & Design


☐ Architecture


☐ Criminology


See more...





John Doe


 145


 PhD

 Software Eng.


 UM, Portugal


 Ukraine


 € 15,000





John Doe


 145


 PhD

 Software Eng.


 UM, Portugal


 Ukraine


 € 15,000





John Doe


 145

 PhD

 Software Eng.

 UM, Portugal

 Ukraine

 € 15,000

-

1

2

3

+

Figure 10: Searching students

20

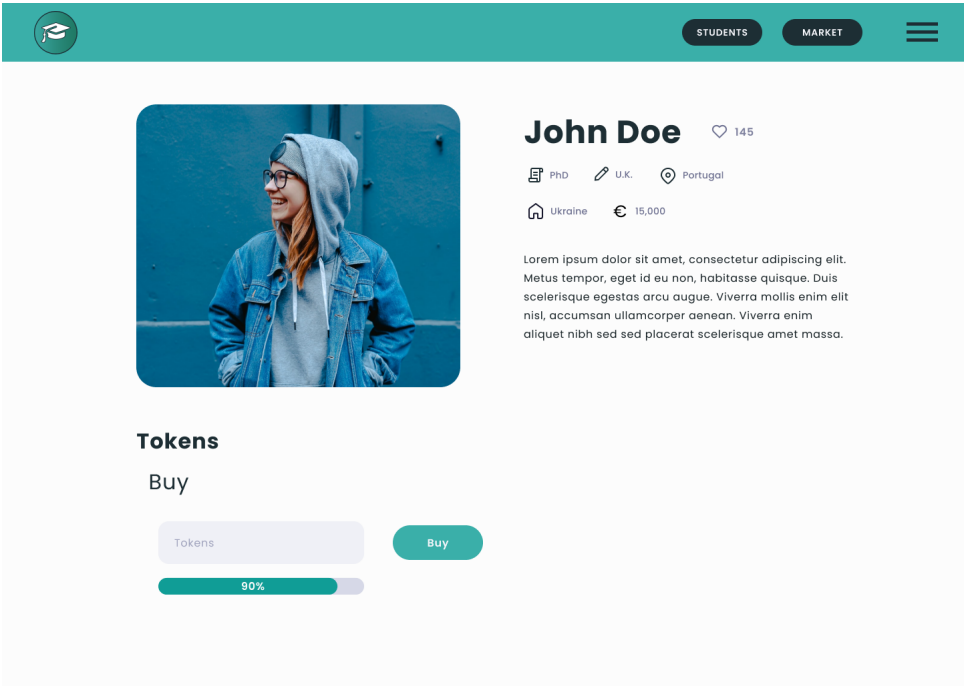


Figure 11: Student page

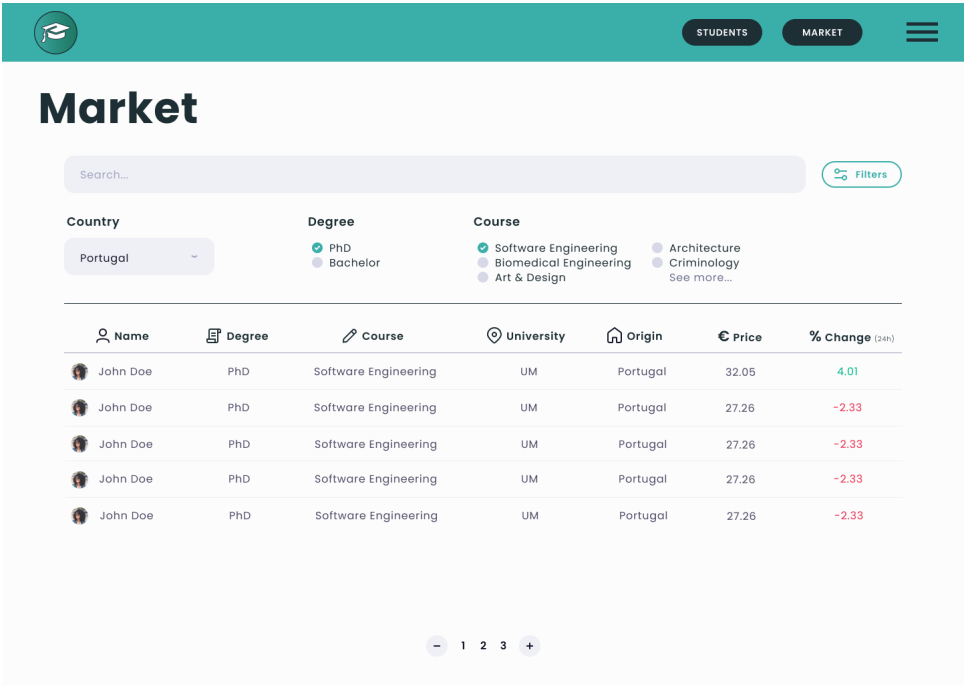


Figure 12: Secondary Market



Figure 13: Active student page

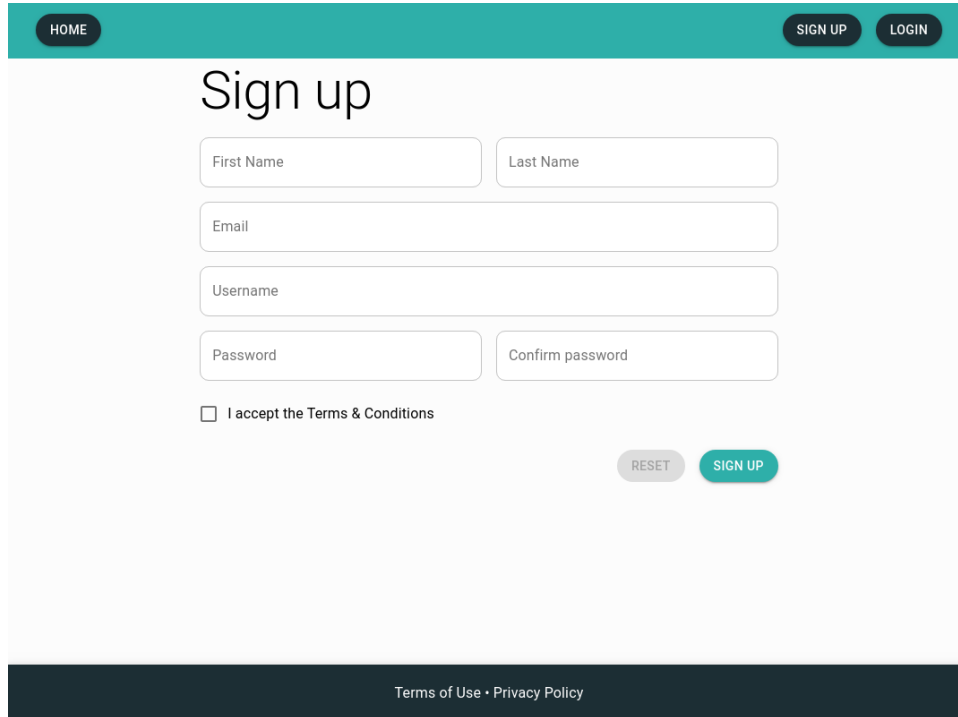
8.2 UI/UX

Throughout the building process of the user interface, some principles of usability [5] were put in practice. Alongside with this principles, the Nielsen's heuristics [6] were used to evaluate the UI. In this chapter will be discussed how this principles and evaluation techniques were used in the different stages of our application interface.

Disclaimer: UI development still in progress, not the final version.

8.2.1 User authentication

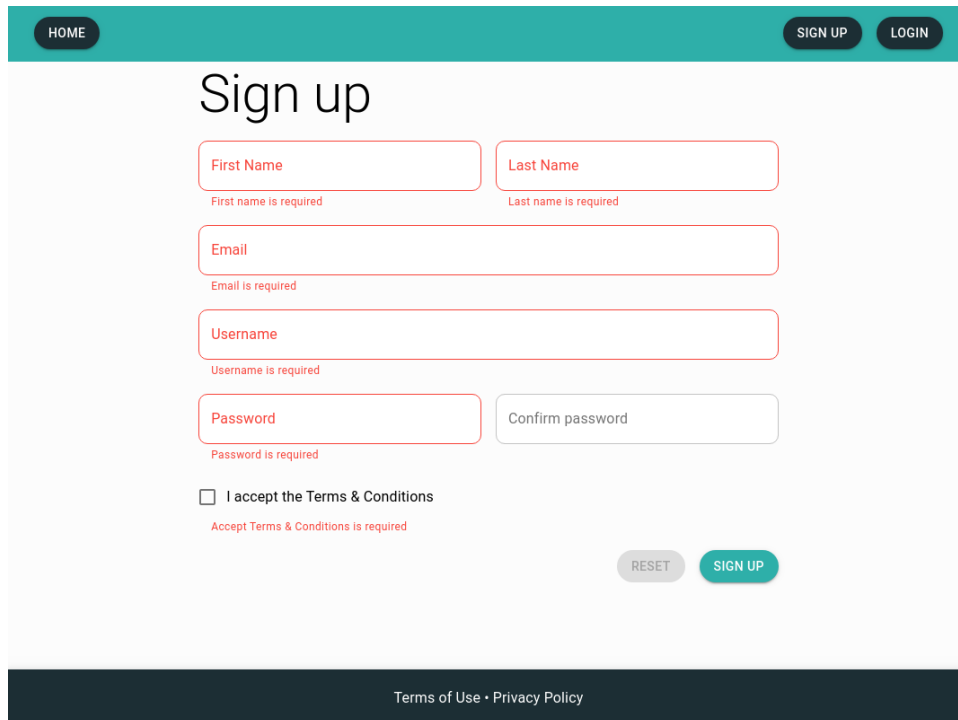
Starting with the user authentication (sign up and login), the user is presented to a regular form, illustrated in the following figure.



The image shows a web form titled "Sign up". At the top, there is a teal navigation bar with a "HOME" button on the left and "SIGN UP" and "LOGIN" buttons on the right. The form itself is on a light gray background. It contains the following fields: "First Name" and "Last Name" (side-by-side), "Email", "Username", "Password", and "Confirm password" (side-by-side). Below these fields is a checkbox labeled "I accept the Terms & Conditions". At the bottom right of the form are two buttons: a gray "RESET" button and a teal "SIGN UP" button. A dark teal footer bar at the very bottom contains the text "Terms of Use • Privacy Policy".

Figure 14: Sign Up

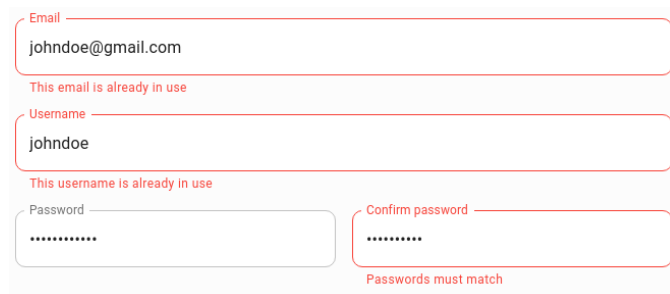
Every field is required, giving an error if the user tries to submit the form without filling some field. The errors, following the principle of *Observability*, can be seen in Figure 15.



The image shows a 'Sign up' form with a teal header bar containing 'HOME', 'SIGN UP', and 'LOGIN' buttons. The form title 'Sign up' is centered. Below it are five input fields: 'First Name', 'Last Name', 'Email', 'Username', and 'Password'. Each field has a red border and a red error message below it: 'First name is required', 'Last name is required', 'Email is required', 'Username is required', and 'Password is required'. There is also a 'Confirm password' field. Below the password fields is a checkbox labeled 'I accept the Terms & Conditions' with a red error message 'Accept Terms & Conditions is required'. At the bottom right are 'RESET' and 'SIGN UP' buttons. A dark footer bar contains 'Terms of Use • Privacy Policy'.

Figure 15: Sign Up - Errors

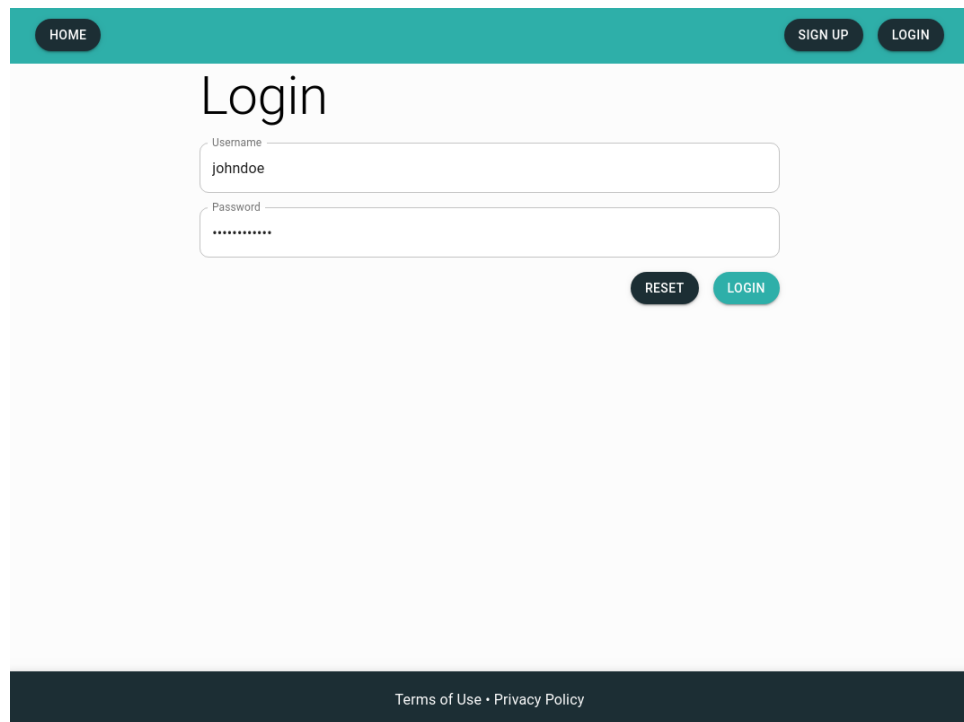
The fact that an error shows up when the passwords, visible in Figure 16, don't match illustrates both the principles of *Predictability* and *Observability*, and the *error prevention* heuristic. This way, the user knows that he/she can't submit the form and what is causing the problem.



The image shows a 'Sign up' form with the following fields and error messages: 'Email' (johnndoe@gmail.com) with 'This email is already in use'; 'Username' (johnndoe) with 'This username is already in use'; 'Password' (masked with dots) and 'Confirm password' (masked with dots) with 'Passwords must match'.

Figure 16: Sign Up - Errors (2)

The login follows the same principles (*consistency and standards*) but not to be repetitive, only the login form is presented in Figure 17.



The image shows a login form with a teal header bar. The header contains a 'HOME' button on the left and 'SIGN UP' and 'LOGIN' buttons on the right. The main content area has a large 'Login' title. Below the title are two input fields: 'Username' with the text 'johndoe' and 'Password' with masked characters. To the right of these fields are 'RESET' and 'LOGIN' buttons. At the bottom, a dark footer bar contains the text 'Terms of Use • Privacy Policy'.

HOME SIGN UP LOGIN

Login

Username
johndoe

Password

RESET LOGIN

Terms of Use • Privacy Policy

Figure 17: Login

9 Backend

The development of this phase is also divided in two parts, one being the research of technologies to use in the background of the application and a database having in consideration that we were going to develop an application resting on the Ethereum network using Blockchain technology, and the second one being the development of the application itself. So next, we're going to address some of the reasons that led to our decision of technologies to use based on our research and their characteristics, the process of developing the app, which means the timeline of events, e.g. some decisions that might have changed midcourse, and describing how the app runs with the backup of tests to support it.

9.1 Research

9.1.1 Project framework

Regarding the development framework, we initially restricted the research to five main frameworks and tried to gather the pros and cons of each other to decide which one would be the best to use, taking into consideration their learning curves, scalability, modularity and the need of communicating with the Ethereum network. Next, we are going to present the researched frameworks and their most important characteristics.

- **Django**

This framework is great to develop products fast and safe, it helps developers speed up their projects. It includes its own Object Relation Mapping layer for handling database access, sessions, routing and multi-language support. It also includes a admin panel which makes the job of managing models and test methods created much more easier than other frameworks. Regarding security, it includes prevention of common attacks like Cross-site request forgery and SQL injections. Another advantage present is inheritance of python's benefits, for example, the great libraries python has and the facility of learning this language.

However, this framework can lead to slow websites, since python is not the fastest of languages combined with a possible badly-designed architecture so designing a properly optimized app is worth special attention. Another negative point about Django is that almost anything can be configurable, if one is not aware or doesn't analyse what Django has to offer it may be required to set almost everything from the beginning in contrast with frameworks like Ruby or Rails which may slow down the development process of the application. Django is designed to deliver standard web applications pretty fast.

- **Node.js**

This is not a framework but rather a runtime environment based on Chrome's V8 Javascript Engine. It offers a really robust technology stack with automatic benefits from JavaScript development. It's more efficient and more productive with the ability to reuse and share code since, for example, frontend and backend can both use JavaScript for their respective programming, is faster in terms of performance, can be really good to share insights inside a developing team and has a huge number of free tools. Node.js is very fast because, as we mentioned before, it is based on Chrome's V8 JavaScript engine which is really fast shown by performance benchmarks made by Google and because of the asynchronous request handling, it means it's capable of handling requests without delays. This means requests are processed without blocking others (don't have to be sequential). It's also a very scalable technology for microservices since it's a very light tool and the app logic can be broken into smaller modules, microservices to enable better flexibility and create a foundation to future work so it's easier to add more microservices on top of the ones that already exist. Furthermore, it contains npm which is a rich package manager with a lot of open source JavaScript tools and a great corporate support since more and more companies use Node.js in their production.

Even though there seem to be quite some advantages of using Node.js, there are also some disadvantages like not being a good tool for heavy computation since it's considered single threaded, as we already mentioned, it processes requests asynchronously so, if a CPU bound task arrives, Node.js sets all the CPU available for that task to be performed first while the others are queued. This results in slow processing and overlay delay of the event loop, which is why, Node.js is not recommended for heavy computing. Once again, it's asynchronous nature rises other problem which are callbacks. Since every task has a callback (function that runs each time a task in the queue is finished) this can lead to these functions being nested inside each other making it much more difficult at times to understand and maintain code. Finally, it's not an easy tool to master and it's necessary to have special attention to the tools that are used from npm since it's open source, they can be poorly built and/or documented.

- **Laravel**

This framework integrates PHP as programming language and uses its latest features like namespaces, anonymous functions, interfaces, etc. It also offers great documentation making it really developer friendly with descriptions about classes, approaches and code types. Furthermore, Laravel has a fast development cycle since integrations are a lot quicker and a big community that can always help when facing a problem or when stuck. Laravel allows reverse

routing which means that it's possible to create links within the structure to named routes by using the name of the specific router when creating links and also queue management which means that tasks that are unnecessary or no longer relevant are removed from the queue to decrease user response time. Furthermore, it integrates mail services because it can use drivers that enable sending email either through local or cloud-based services. It can also use the npm which, as we mentioned early, it's a rich package library that has a lot of open source tools and has a feature, that enables creation of models with corresponding tables in the database that when you change a model then the data will also change, called Eloquent ORM. Finally, it has an easy authentication system, automatic testing since it comes with a framework dedicated for that matter along with other methods and principles and also configurations and error handling with the integrated Monolog logging library.

PHP platforms, in general, have a few issues regarding version with long-term supports since upgrades can cause some problems to projects. Also PHP is not a very popular language making it a bit complex for someone who doesn't have knowledge about it since it involves heavy documentation which requires some experience from the start. When compared to other frameworks like ruby, rails or Django, Laravel has limited inbuilt support due to it being lightweight with the only solution being the installation of third-party tools. In a nutshell, this is a framework that is relatively easy to learn but hard to master with many developers overestimating themselves since it's more difficult than what initially appears.

- **Phoenix** Phoenix is a framework from the functional programming language Elixir that shares a similar syntax with Ruby. It is able to run a web app handling many requests at the same time since Elixir was created with this type of concurrency in mind meaning there aren't any slowing down of the application. It's also very scalable since it runs on Erlang VM, it is able to run different applications on multiple communicating nodes which makes it easier to create a larger web app that can be scaled over to different servers and improve performance. Phoenix has a very useful characteristic which is its fault-tolerance, because it provides built-in safety mechanisms that allow a web app to still run even something goes wrong. It also can be easy to understand code since it's functional and it's generally more logical and easy for beginners to understand.

One of the problems of working with Phoenix is that it's also required to know Erlang to deliver the best product since it contains a large number of libraries that can help developing a project. Furthermore, because this is a relatively new framework, the community is still small compared to other more popular frameworks which can make it harder for starting developers to learn and use

Phoenix. Even though, Elixir is a functional programming language, which as we already mentioned, can be easier to understand due to that functionality, it can also be seen as harder to understand since there aren't many functional programming languages and programmers, generally speaking, aren't used to this type of languages so this can be seen as a double-edged sword.

- **Flask**

Flask is another framework that uses python as its programming language like Django but, naturally, have their differences and situation or projects where they fit best. This is a lightweight Web application framework which is built for a fast and easy use that has minimal dependencies on external libraries. One of its main advantages is that it is easy to understand and is very good for someone is starting out developing web application and has some knowledge of python. Besides being easy to use and very simple, it also gives the user full control over the web development. This means that it's very flexible since there are only a few parts of Flask that cannot be changed, consequently, almost all parts of Flask are open for the user to change however they like, unlike other frameworks. Another great thing about Flask is that it allows testing through its integrated support, built-in development area, fast debugger and restful request dispatching.

Even though it's easy to learn and to create a web application it's also easy, for a bad developer to write bad code or deliver a bad web application. Flask has a singular source which means it handles all the requests one at a time creating a bottleneck resulting in a slow web application if the intent is for it to serve multiple requests at the same time. There are also a lot of modules available in python that can be used in Flask, and even though this generally means a good thing, using a lot of modules can lead to breaches in security because the process and development are no longer between the web framework and the developer because of the involvement of other modules. Furthermore, it lacks database and ORM which means it's required more work than other frameworks that contain this tool since it's required to use other database programs, build the database there, connect it to the web application and manage both the app and the database on two different spaces.

After much group deliberation, discussions and analysing the pros and cons of all these frameworks we mentioned above, we decided to use Django for the backend of our project. Some of the reasons that made us choose Django were that the people that were assigned to do the backend had all knowledge in python which is the programming language inherited by Django. It's an easy framework to use and learn even though, as we already said, it allows a lot of user configuration, we studied and had some previous knowledge that helped us get through the development.

9.1.2 Database

For the application's database, several options were on the table, but most importantly was deciding between a relational database or a NoSQL one.

Regarding relation databases, we had engines like MySQL, PostgreSQL, MariaDB and Oracle SQL. With relational databases, the data modeling process is easier, since they are based on schemas. Also, they are highly available and highly consistent, while ensuring that sensible and complex transactions are processed, thanks to ACID (Atomicity, Consistency, Isolation, Durability). ACID guarantees that transactions are valid even in the event that you may encounter an error, power failure or even a crash. Besides that, relational databases offer a level of maturity and widespread support that remains unrivaled by current NoSQL alternatives.

On the other hand, NoSQL databases do not have a strict schema, providing high flexibility to applications, as well as scalability and fast processing (NoSQL databases scale horizontally, through auto-sharding or consistent hashing). Unlike traditional, SQL based, relational databases, NoSQL databases can store and process data in real-time, due to their distributed nature. They can be divided in four categories:

1. Document Databases – These databases usually pair each key with a complex data structure which is called a document. Documents can contain key-array pairs or key-value pairs or even nested documents. Examples of document NoSQL: MongoDB, Apache CouchDB, Raven DB, ArangoDB, Couchbase, Cosmos DB, IBM Domino, MarkLogic, OrientDB.
2. Key-value stores – Every single item is stored as a Key-value pair. Key-value stores are the most simple database among all NoSQL Databases. Examples of Key-value NoSQL – Redis, Memcached, Apache Ignite, Riak.
3. Wide-column stores – These types of Databases are optimized for queries over large datasets, and instead of rows, they store columns of data together. Examples of Wide column NoSQL – Cassandra, Hbase, Scylla.
4. Graph stores – These store information about graphs, networks, such as social connections, road maps, transport links. Examples of Graph NoSQL – Neo4j, AllegroGraph.

9.1.3 Documentation

For documenting Django API we need a tool that could do just that so we had into account 3 of them, Swagger, ReDoc and Sphinx. When developing a web application it's always required to document the API so other people or even developers on the same team can understand what the API can do and how it works. This means internal and external users can save time by accessing this information

benefiting from the API directly while discarding the need to contact software's support. This also users who are not familiarized with coding to understand what the API does since it's easier to perceive it. Finally, it can improve a business or project's reputation and improve customer happiness. Having said this, we ultimately chose Swagger since it's one of the most if not the most popular tool when it comes to API documentation and it also contains something called 'drf-yasg' which is an automatic Swagger generator. This means that as we write the API code, the documentation is also being built even if some adjustments are required after the API is finished.

9.2 App Development

9.2.1 Tool Selection

Based on the research made, the framework and database selected for the application's backend development were, respectively, Django and PostgreSQL.

Django was selected due to its scalability, easy to setup and use ORM and security (CSRF attacks and SQL injections). Also, the team was already very familiar with Python, so the framework's learning curve would be even smaller, leading to a fast development, which was very important given the tight development scheduling. Along with Django, Django REST Framework was also used, providing many of the functionalities required for the Tuichain's backed, like token authentication.

PostgreSQL was selected because we think a relational database would fit our application the best, since Tuichain's data is predictable and structured, having a finite number of individuals accessing it. Also, since our application includes transactions (buy and sell tokens), the integrity and atomicity of the data is very important, and here relational databases and their locked transactions win. Although, this comes with a price: slower performance, but given the lack of complex queries we decided that the impact would not be relevant. We chose PostgreSQL instead of other RDMS because it is free and open source, is highly conformant to SQL standards, has replication features (which allows data availability) and is highly customisable.

9.2.2 Database Models

After a team analysis of the project, we came up with a model for our project consisting of four entities: Students, Investors, LoanRequests and Investments. This model was used in the first approaches and was our basis in an early stage of the project. With the progress that the team was making in the project, the need arose to make some changes. We needed a class for identity checks of both students and investors, so we added to our schema a table for that purpose. After some time, the initial model was again updated, since the team decided that a student could also invest in another students, so we grouped both entities into a single one, which we

called User. When the application requirements were stable, we came up with the last version of the database model, shown in Figure 18.

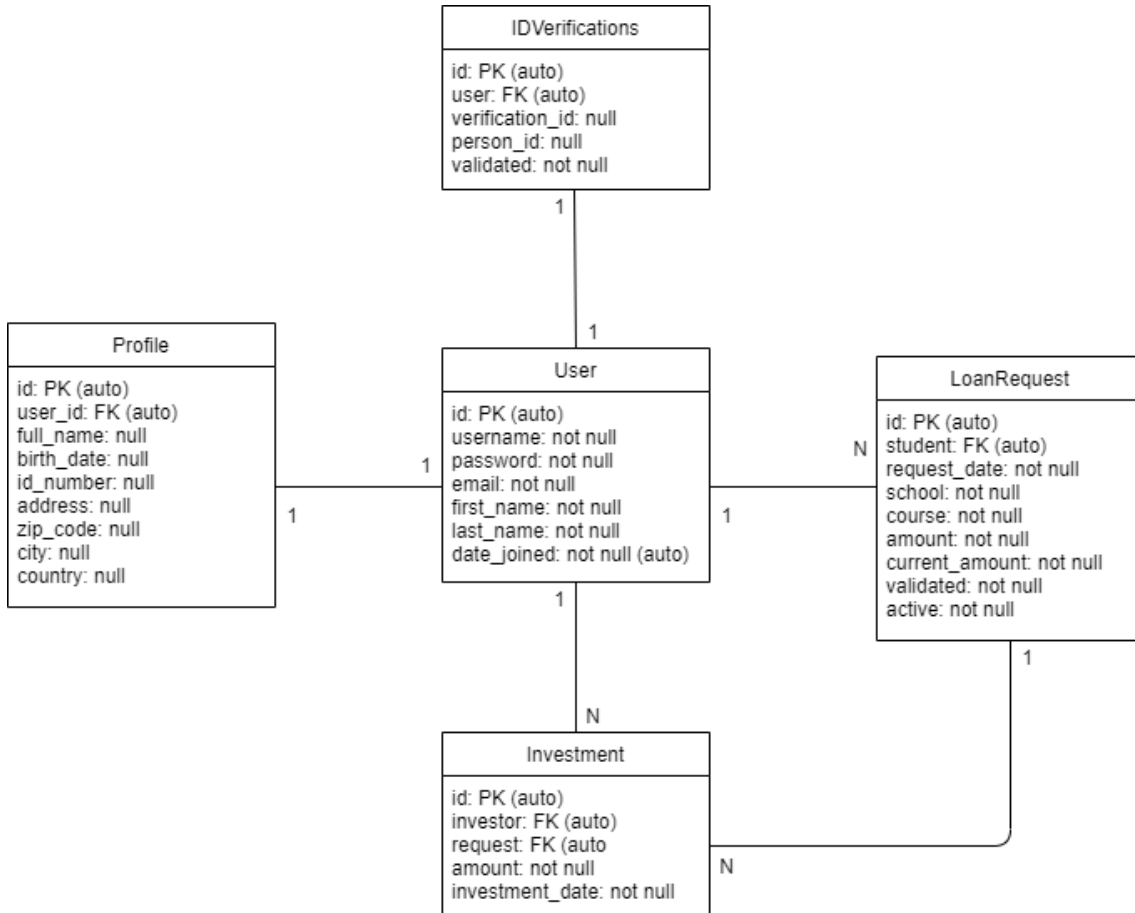


Figure 18: Database Model

In sum, we have:

1. **Users**, described by their username, email, password, first and last name and the date the account was created. For this purpose, we used the User model defined in the Django's authentication package.
2. Each User has a **Profile** with the personal details, such as full name, birth date, geographical information (country, city, address and zip code) and ID number.
3. Each User also has information about its **IDVerification**, such as the verification ID and person ID provided by the validation service and an indicator describing the verification status.

4. Users can also make **LoanRequests**, described by the request date, the school and the course (along with its tuition) they want to attend, the amount gathered at the moment and if the request is active and validated.
5. Besides Loan Requests, Users can also make **Investments** in another users' loan requests. These investments are described by its amount and its date.

9.2.3 API Routing

In order to provide the required data to the frontend of the application and to let it update the database, the backend needed to have routes that allowed that manipulation, so a set of methods with defined endpoints was developed in order to do so.

These endpoints can be divided in two categories: global and protected. The last ones are protected, as they require the user to be authenticated and therefore check if the access to the specified resource can be given.

The global category includes the authentication routes, which allow the user to use the protected ones.

The protected routes can also be divided in four categories: user related, loan request related, investment related and external services.

The documentation plays an important role here, since it allows the frontend development team to know exactly what each endpoint does and what are their parameters, type of request and response codes and body. So, an extra endpoint was defined containing the API documentation.

9.2.4 Authentication

The API authentication is done with a token, as provided by the Django REST Framework. The DRF token is stored in our database and is related to a single user, with no expiration date. This authentication method is easier and more logical to program and leaves less space to interpretations and programming flaws or errors. However, it implies a database hit on all requests to validate the user's identity, slowing its performance in comparison with other token authentication alternatives (e.g. JWT). DRF Token also allows forced-logout by replacing the token in the database (e.g. password change).

When the User creates an account, an authentication token is also created and returned as the signup/login request response. The frontend application needs to store that token and use it in every request's header for the protected routes as follows: "Authorization: Token <your_token>".

Users cannot modify other users' data and anything related to them, they can only have access to their own information and resources.

10 Blockchain

This component of the project aims to be a library which helps Backend to access the Blockchain network. Besides that, it also includes the smart contracts which have to be deployed in the Blockchain network, to make our business logic to work.

The development of this phase is divided in three parts, one being the research of platforms which allow the deployment of smart contracts, and consequently, the available languages and frameworks to do so; the second being the design of the solution using UML diagrams; and the last one being the actual implementation of the solution.

10.1 Cryptocurrency platforms and tooling

The Ethereum platform was selected as the application's cryptocurrency platform in brief due to its widespread use and programmability. As of Oct. 23, 2020, the 5 cryptocurrencies with the highest market capitalization were, in decreasing order:

1. Bitcoin.
2. Ether;
3. Tether;
4. Ripple's XRP;
5. Bitcoin Cash.

The plots in Figure 19 depict the evolution of market capitalization of these 5 cryptocurrencies over the past 7 years, as reported by CoinGecko. As of Oct. 23, 2020, Bitcoin had 60.2% (\$239.6B) of the total global cryptocurrency market capitalization (\$399.0B), Ether had 11.6% (\$46.2B), and Tether had 4.0% (\$16.1B).

The Bitcoin blockchain is too limited to allow for the implementation of the TuiChain application with the desired functionality. In contrast, the Ethereum platform fully satisfies our functionality requirements and is currently the *de facto* ecosystem to create this kind of financial applications.

Regarding the tools and frameworks used for the implementation of the application's blockchain component, we selected the Solidity smart contract programming language due to its widespread use and stability, and the Truffle framework for the validation and testing of the implemented contracts.

Further, the web3.py library was used to develop a Python layer which exposes all functionality implemented by the blockchain component to the backend layer, as the latter is also implemented in Python.

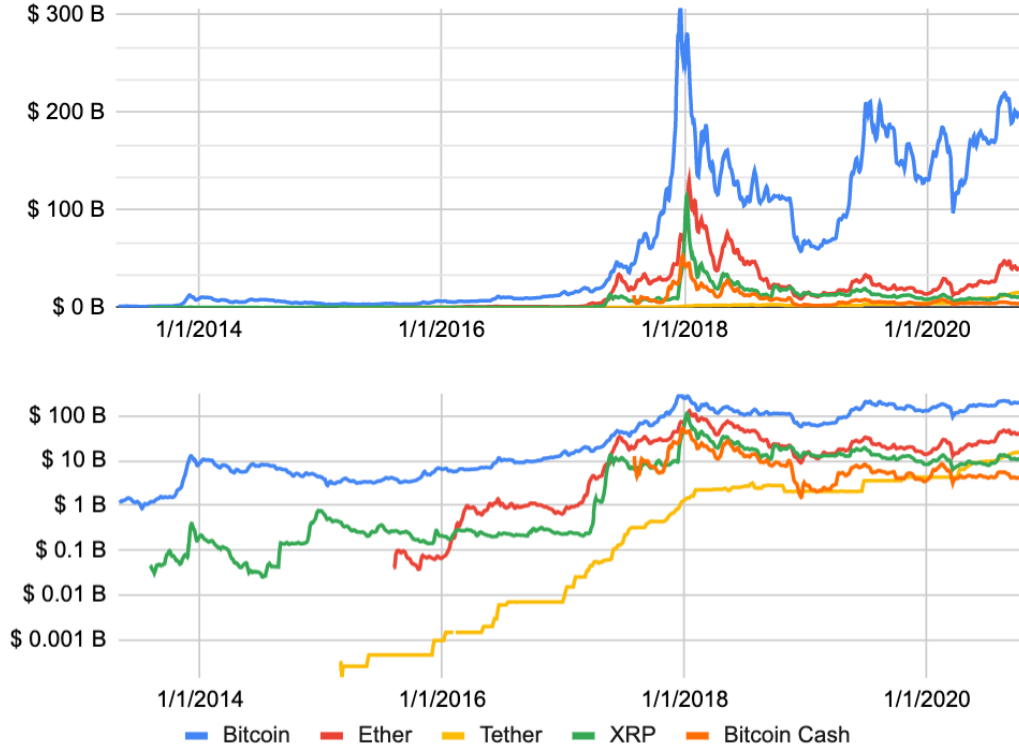


Figure 19: Market capitalization in USD of the 5 top cryptocurrencies from Apr. 28, 2013 to Oct. 23, 2020 (7-day moving average of daily values; top: linear y-axis scale; bottom: logarithmic y-axis scale).

10.2 Design and Specification

The Blockchain project will have a package and folder structure as the one represented in the Package Diagram (figure 20), with the clear distinction of the two different parts which are the access library, and the Smart Contracts.

According to the figure 20, we organized this project in three main packages. The first one is the `tuichain_ethereum` which contains all the logic behind the access to Ethereum network, and abstracts its implementation. The second one is the `test` package which contains all the required tests to the `tuichain_ethereum` package. The last one is the `truffle` package, representing a typical structure of a truffle project, which contains, mainly, the `contracts` package which includes all the contracts, and the `test` package for testing `contracts` package.

10.2.1 Contracts

First, we started to design the structure of the various components of the Smart Contracts in a class diagram, bearing in mind that at the end, we intend that the infrastructure deployed on the Ethereum chain, would be composed of:

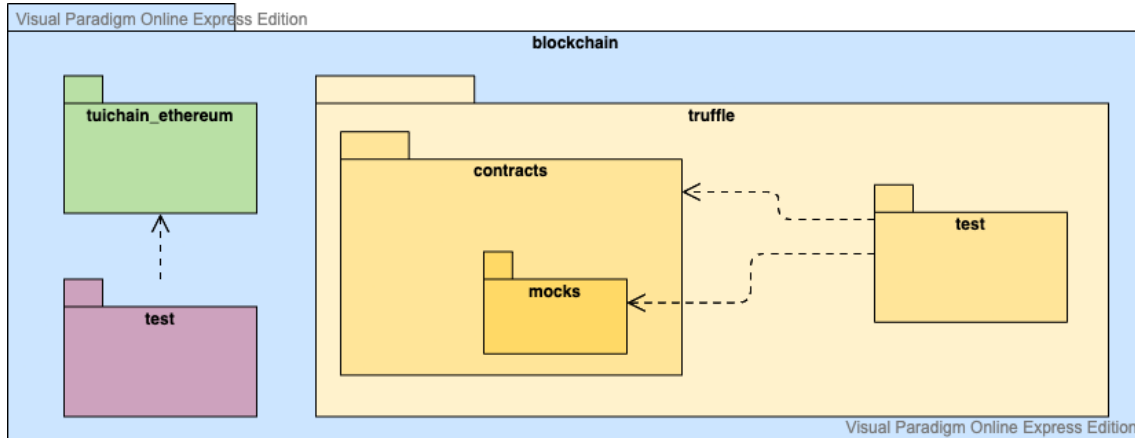


Figure 20: Package Diagram.

- a controller contract;
- a market contract;
- any number of loan contracts;

The idea for the final application is to have a single instance of this contract infrastructure deployed in the Ethereum mainnet. Initially, only the controller and market contracts exist. Every time a loan is created, a loan contract is deployed. The market contract and all loan contracts are reachable from the controller contract, which ties the whole thing together.

The figure 21, represents the simplified class diagram showing the needed classes/contracts with the various attributes, and the relationships between classes. We intend to use **openzeppelin** libraries, which provides security products to build, automate, and operate decentralized applications, as well as some implementations for token standards.



Figure 21: Class Diagram of Smart Contracts Simplified.

The figure 22, represents a detailed class diagram that takes into account all the methods declared or implemented in each class, as well as library usages.

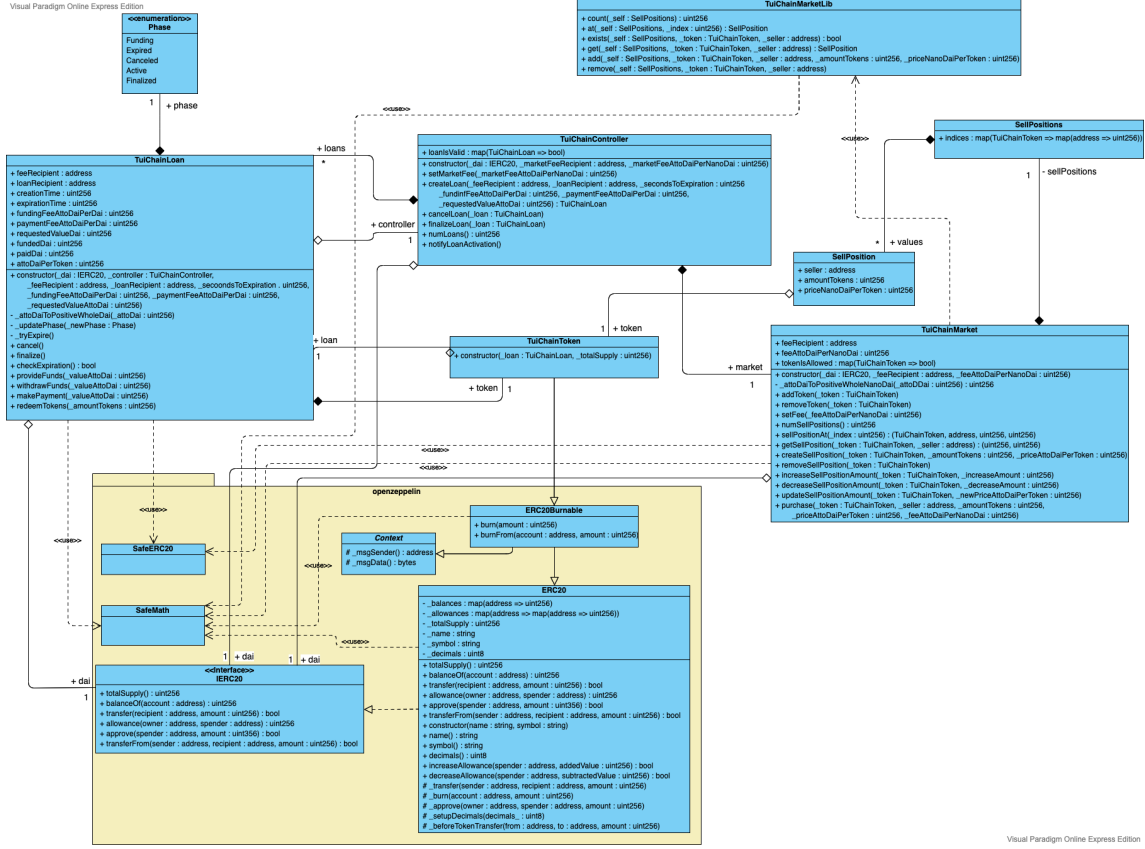


Figure 22: Class Diagram of Smart Contracts.

When all the infrastructure is deployed in the Ethereum network, only three types of contracts will be accessible, and the public interfaces of them is represented in the figure 23.

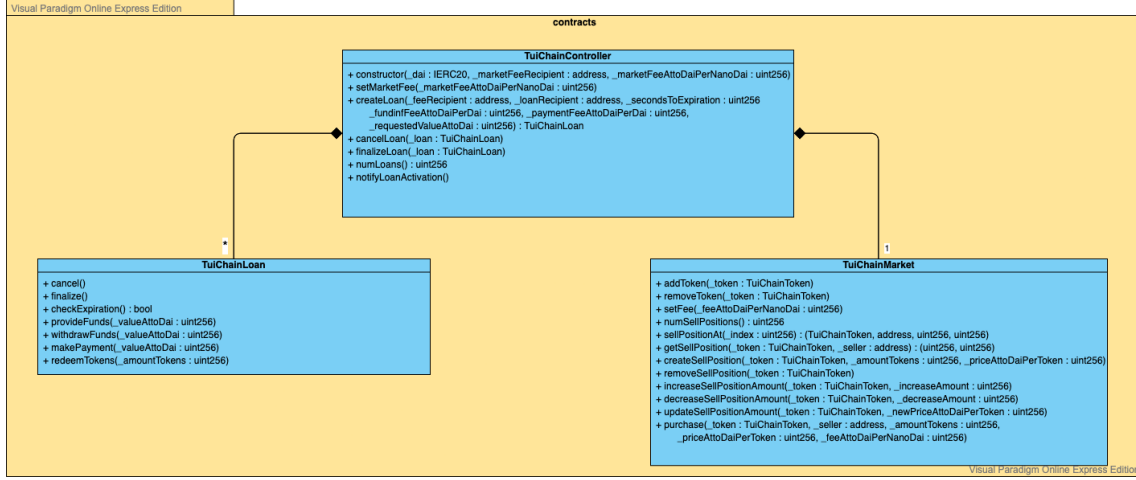


Figure 23: Class Diagram of the Public Interfaces of Smart Contracts.

In addition to structural diagrams, the Loan contract presents a behaviour that deserves clear emphasis and specification, because this is where the whole logic of state transition lies in an application for funding. Thus, figure 24 presents a state machine diagram, which represents the Loan Contract behaviour, reduced to states, and also the events responsible for the transitions.

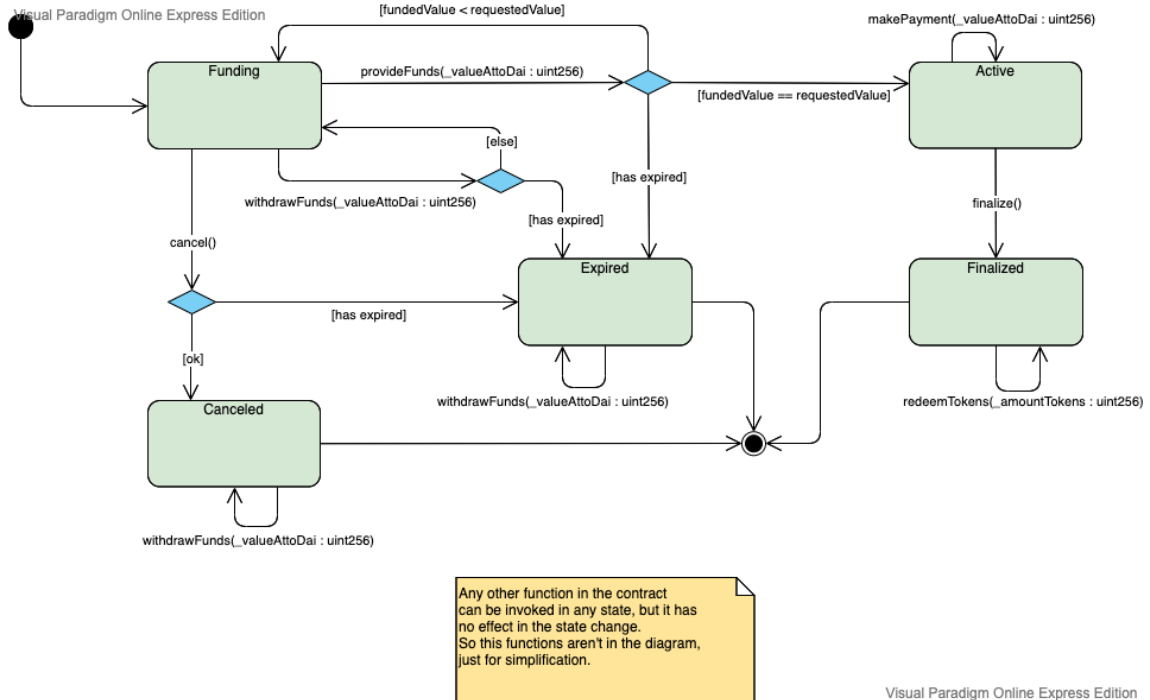


Figure 24: State Machine Diagram of the Loan Smart Contract.

10.2.2 Access Library

After specifying the contract infrastructure to be deployed, it remains the specification of the access library to interact with the Ethereum network, and to be used by the Backend component.

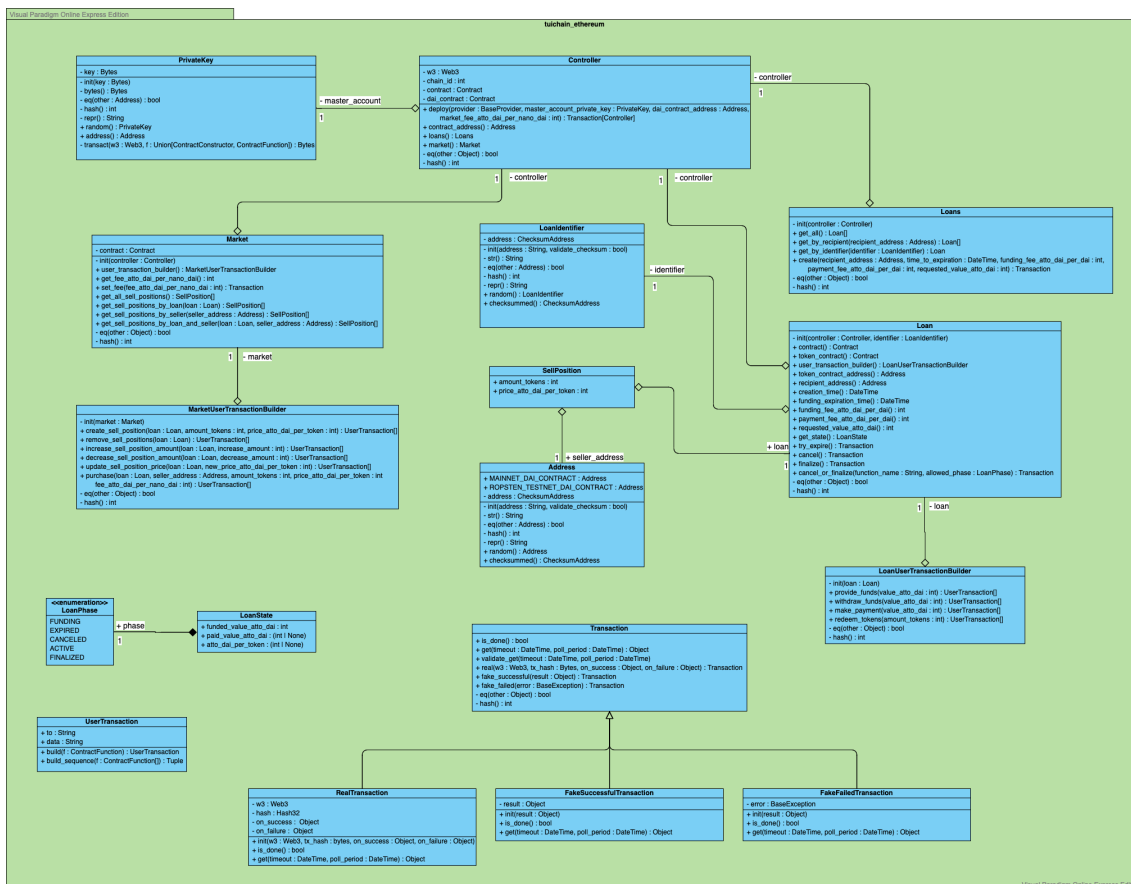


Figure 25: Class Diagram of the Access Layer.

This access library can be reduced to the class diagram presented in figure 25. The specification of this library is pretty basic once the contracts are already specified, being the ones where the logic is. In this diagram, the classes are basically the same as the ones in contracts, but this ones have a different purpose. For example, the Controller class in contracts class diagram represent the logic which the controller must have, on the other hand, the Controller class in this diagram represents the logic behind the Ethereum access to the functions that the Controller contract has, and can be invoked.

10.3 Implementation

Standard safety and security guidelines were following during development of the Solidity contract implementations. These include:

- All functions in all contracts are $O(1)$ (have constant complexity). This prevents denial-of-service attacks that attempt to increase the gas consumption of certain functions beyond what is allowed by Ethereum's block gas limit;
- The checks-effects-interactions pattern is followed wherever possible, preventing reentrancy bugs (cf. https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html);
- Battle-tested, widely-used contract implementations and libraries provided by the OpenZeppelin project are reused wherever possible, such as its ERC-20 token implementation.
- All arithmetic is checked for overflow and underflow using OpenZeppelin's SafeMath library.

11 Deployment and Maintenance

A production deployment of the TuiChain application consists of the following components or services:

- A reverse proxy and static content provider such as NGINX;
- One or more instances of the Django-based web server;
- A (possibly replicated) relational database management system such as PostgreSQL;
- One or more Ethereum clients.

Several choices can be made as to in which way to deploy these components, such as whether to collocate several services in the same hosts. Deployment of Ethereum clients, in particular, may be delegated to a third-party service such as Infura, which provides interfaces to already-deployed clients. These matters are yet to be decided.

Two kinds of development deployment are used for the development of the TuiChain application, both deploying a local Django server that also serves static content. In one of those deployments, the blockchain component is backed by a local test Ethereum network provided by the Ganache tool. The other deployment is similar but relies on a public Ethereum test network, such as Ropsten, Kovan, Rinkeby, or Goerli, thus providing an environment closer to the a production setting.

References

- [1] HSBC Holdings. The Value of Education: The price of success. Technical report, HSBC Holdings, 2018. Retrieved Nov. 8, 2020 from https://www.us.hsbc.com/content/dam/hsbc/us/en_us/value-of-education/HSBC_VOE5_USA_FactSheet_508r2.pdf.
- [2] Home – Fundação José Neves. Retrieved Nov. 8, 2020 from <https://joseneves.org/>.
- [3] Home – lambda school. Retrieved Nov. 8, 2020 from <https://lambdaschool.com/>.
- [4] Andrew Shepherd. *Hierarchical Task Analysis*. 2000.
- [5] Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. *Human-Computer Interaction*. 01 2004.
- [6] 10 usability heuristics for user interface design. Retrieved Nov. 20, 2020 from <https://www.nngroup.com/articles/ten-usability-heuristics/>.