University of Minho
Integrated Master in Informatics Engineering

TuiChain: A Blockchain Platform for Higher Education Financing

Technical Report

Projeto em Engenharia Informática

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

January 2021

Abstract

Although in Portugal the costs of higher education are not so high compared to less developed countries, there are still countless citizens who are unable to take the courses they want. In the case of less developed countries, the lack of support programs for education and higher education is an added problem to the lack of monetary possessions on the part of the inhabitants. This is particularly important now as more than ever higher education courses are gaining more and more important for entering the job market as companies are essentially looking for people who are highly specialized in a particular technical or scientific area.

Traditionally, students use classic loans provided by some banking institutions to try to obtain a higher academic degree, but these are always associated with interest payments, which often means in that students entering successive loan loops. Besides, these banks often reject students, considering them as a high-risk investment. Nowadays, some foundations work in this area of financing education through Income Share Agreements (ISAs) but these foundations are always limited by the capital of those who manage them. As soon as the capital is depleted, countless citizens remain without obtaining the financing they needed.

The TuiChain platform, based on Distributed Ledger Technologies and Ethereum's blockchain was developed to solve this problem. This platform allows the union between lenders and students who need funding. ISAs are established with the usual conditions that characterize it, allowing the student to obtain the total financing they need and investors a number of tokens proportional to the amount invested. In addition to the main market, investors will be able to trade tokens between them with the value of the tokens to be defined by the secondary market, based on existing demand and supply.

The existence of this platform brings numerous advantages to the area of education financing by removing the existing capital limit on foundations and also the numerous bureaucracies that often prevent students and banks from making contracts.

Contents

List of Figures

1 Introduction

1.1 Contextualizaton and motivation

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 [3]. Given these exorbitant figures, it is not expected that everyone will be able to bear such a financial burden. Also, several underdeveloped countries 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 labour 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* [1], 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, 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 funding requests, 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 number 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. Besides, 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, to enable the identification of more or less attractive offers (+/- risk, +/- reward), according to the desire of each user.

1.3 Report structure

The remainder of this document is organized as follows. Section 2 further motivates the platform and introduces relevant background concepts. Section 3 then provides a top-down description of the application, presenting its major functionalities and general architecture. Section 4 enumerates the application's requirements and describes its design. Section 5 specifies the development methodology followed by the team to build the platform. Sections 6, 7, and 8 then detail the design and implementation of the application's blockchain, backend, and frontend components, respectively. Section 9 describes how the application is deployed and discusses aspects relevant to its maintenance. Section 10 lastly concludes the report and identifies future development work.

2 Background

2.1 Cost of higher education

To analyze the impact of our solution on our target market, we conducted a survey and collected a considerable number of responses that allowed us to have an idea of how useful our solution would be.

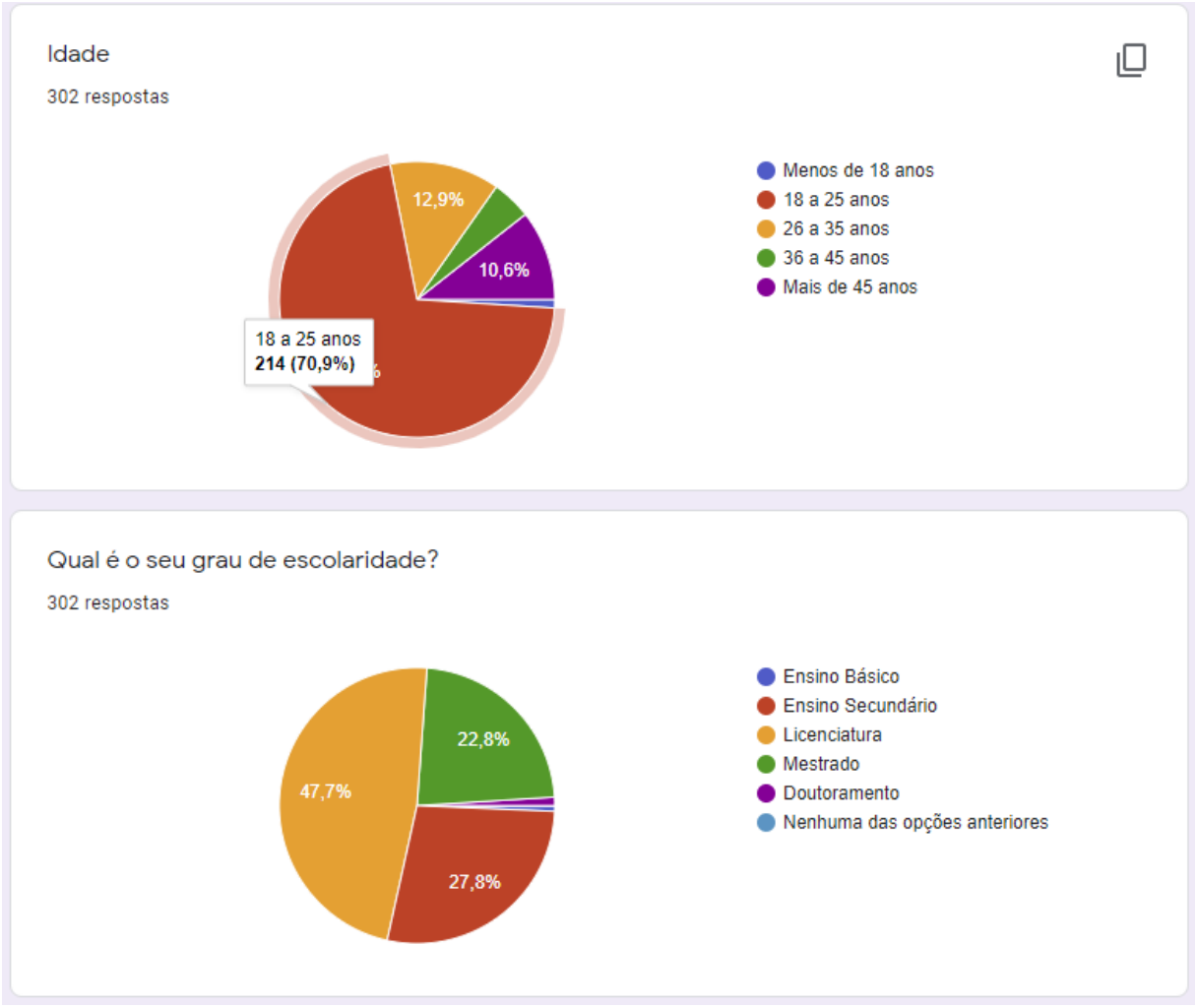

Figure 2.1: Survey results: demographics.

We obtained a total of 302 responses, of which 214 (about 71%) were in the 18-25 age group. Of these responses, half of them came from graduate students and the other half was divided into master's and high school students.

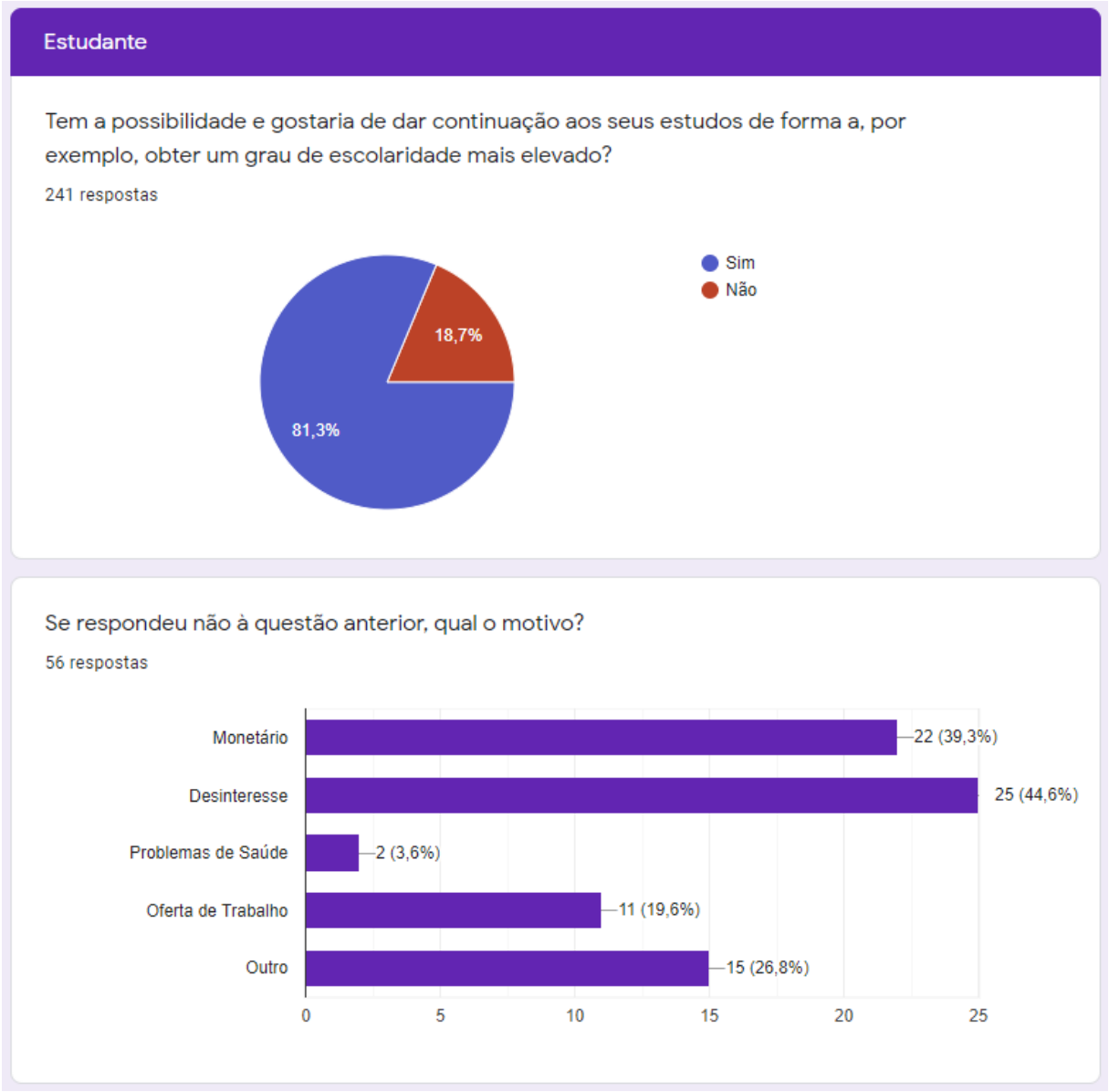

Figure 2.2: Survey results: data from current students.

We decided to divide the questionnaire into 2 large groups: students and alumni. In this way, it would be easier to understand the reasons that led people to stop studying.

In the group of students interviewed, 18.7% answered that they would not like to continue their studies. So to understand why they would not like to do that we lead them to the second question of this questionnaire that allowed us to draw some more conclusions:

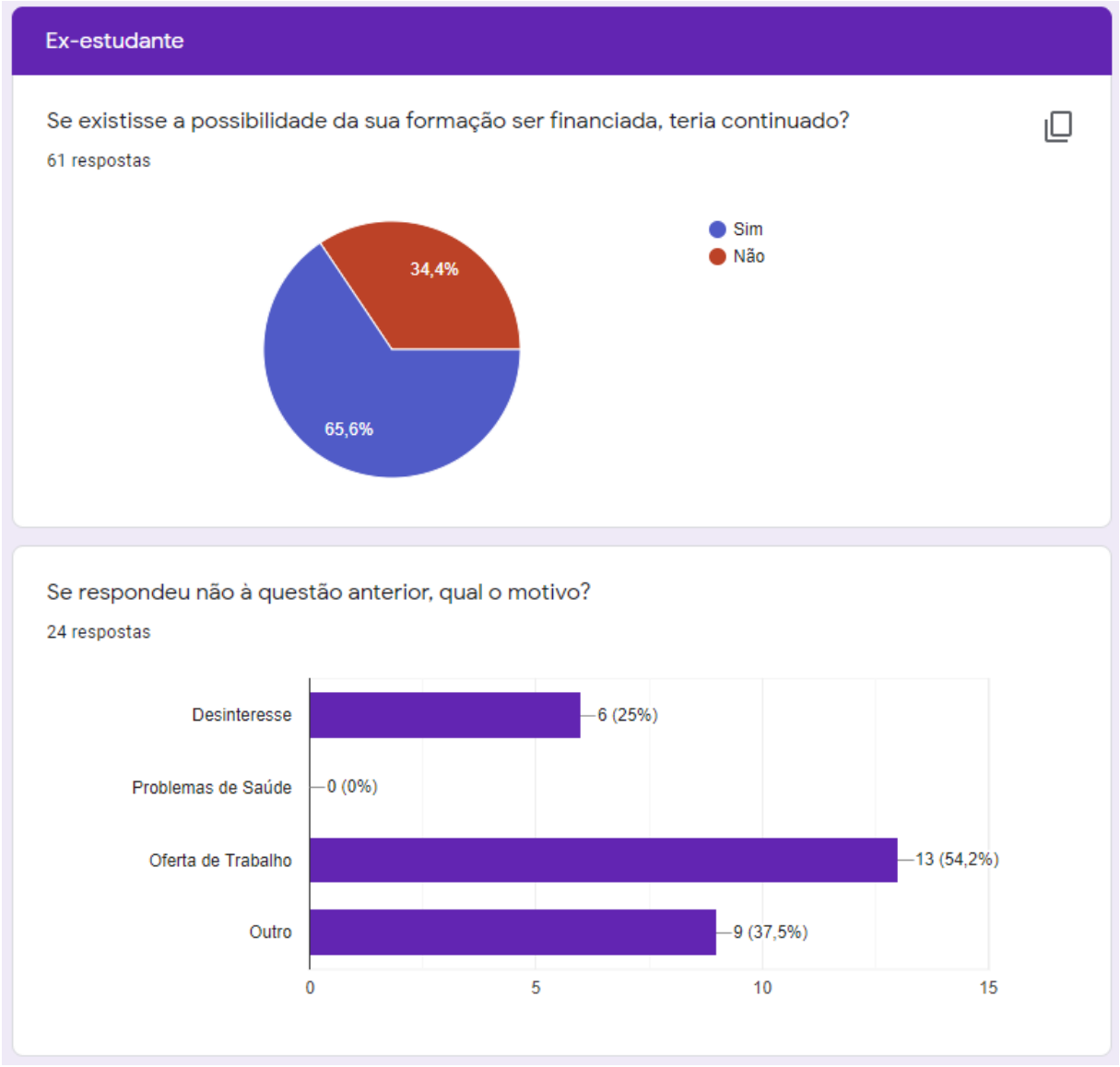

Figure 2.3: Survey results: data from former students.

This section showed that more than half of the former students (65,6%) would have continued their education if it could have been financed. In addition, a good percentage (54% of the "No" answers) said that they did not continue to study due to job offers, which leads us to assume that of these, some could have reconsidered that offer if they could increase their academic degree and even receive a better job offer.

2.2 Income share agreements

An ISA (Income Share Agreement) is a contract between an individual or organization and a student in which the student receives funding from them to complete the desired

course and agrees to pay them for their financial aid through pre-determined payments only after the course has ended based on a percentage of their wage after entering the labour market.

2.3 Blockchain and cryptocurrencies

A *blockchain* is simply a growing list of records, called *blocks*. The blocks in this list are organized sequentially and this relationship is encoded using cryptographic hashes of the block contents, which refer to the parent block. This makes the data structure resistant to direct modification of its data, as modifications to one block would require modification of all subsequent blocks in the chain. Due to these properties, it is used as a basis for the consensus protocols used by most cryptocurrency systems, where it functions as a distributed ledger.

The blockchain data structure and its application as the foundation of an open, decentralized cryptocurrency platform were introduced with the Bitcoin cryptocurrency. Today, many other cryptocurrencies and cryptocurrency platforms exist. Figure 2.4 depicts the evolution over the past 7 years of market capitalization of the 5 cryptocurrencies with the highest current market capitalization, 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.

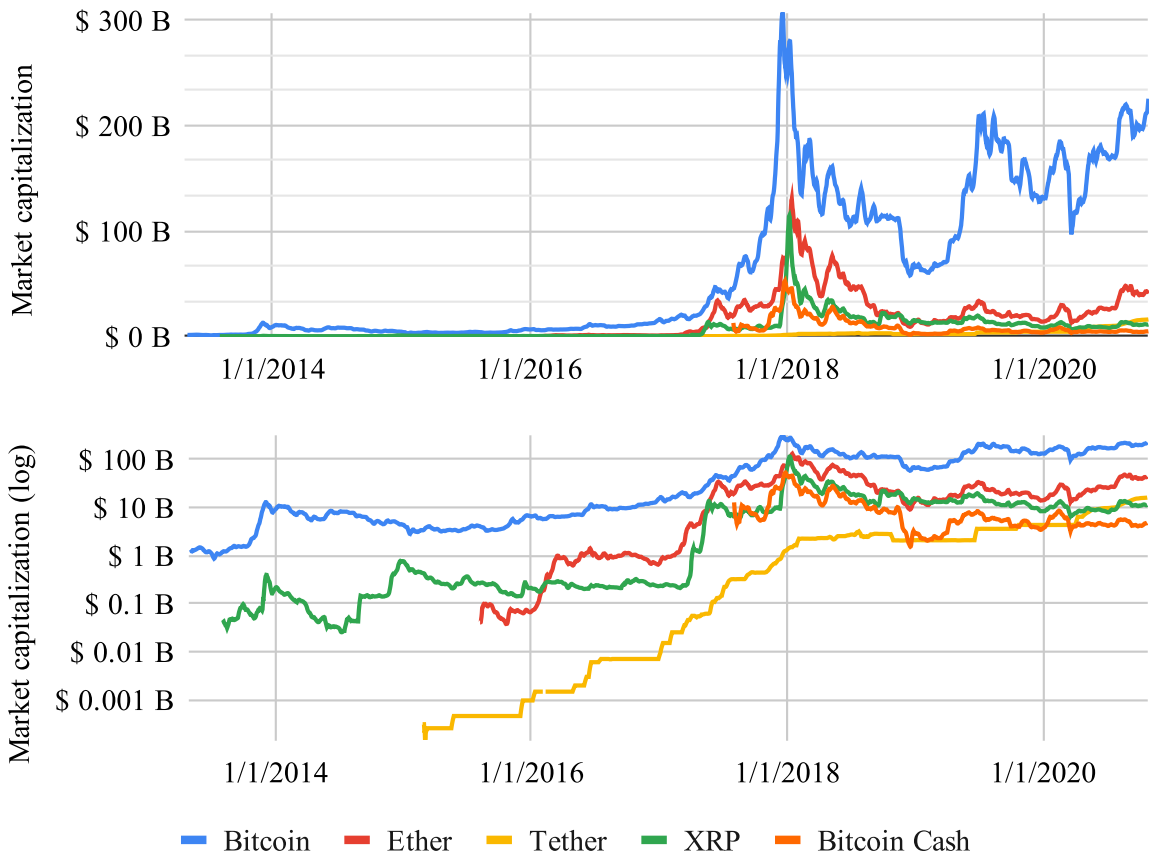

Figure 2.4: **Cryptocurrency market capitalization.** Market capitalization in USD of the 5 top cryptocurrencies from Apr. 28, 2013 to Oct. 23, 2020 (7-day moving average of daily values; top: linear y-axis scale; bottom: logarithmic y-axis scale).

3 Overview

The TuiChain platform enables students to request financial support to pay their tuition and allows investors to provide such funding through the Ethereum blockchain using the Dai stablecoin. Students are legally obliged to repay their investors if and only if their income is above a certain threshold, and the value of the payments is proportional to their income, similar to what is typically accomplished with income share agreements. Additionally, investors can trade shares of their loans with other investors, increasing liquidity.

3.1 Primary market

A student (the first kind of user) begins by applying for the chance to request a loan on the TuiChain platform. This application must include any required official documentation regarding their identity and financial or legal status and is reviewed by a human at TuiChain. If approved, a legally binding contract is then signed between the student and TuiChain. This contract specifies the terms under which the student must make the aforementioned payments. The student must also commit to notifying TuiChain of any changes to its academic and professional status and income.

For every approved loan request, two smart contracts are deployed on the Ethereum blockchain: (1) a standard ERC-20 token contract and (2) a loan management contract. Each token minted by the former corresponds to a predetermined amount of the loan value (e.g., 1 token = 1 Dai), and the total amount of tokens is set to match the full loan value (e.g., 5000 tokens total for a 5000 Dai loan request).

The loan management contract initially owns all tokens and allows interested investors (the second kind of user) to acquire any number of them by depositing their corresponding value in Dai plus a proportional fee (e.g., 1050 Dai for 1000 tokens with a 5% fee). Potential investors have access to details about the student's academic progress and income status. Once the loan is fully funded, the portion of deposited Dai corresponding to the fee is transferred to a TuiChain account and remaining Dai is transferred to the student. If, however, the loan is not fully funded after a predetermined amount of time, then it is cancelled and investors can recover all deposited Dai by returning their tokens.

Students make their payments directly to the loan management contract, which in turn transfers part of the value to a TuiChain account as an additional fee (e.g., 10% of the paid amount). Once the final payment is performed or the student is otherwise exempt from any further payments (as stipulated by the aforementioned legal contract), and once a human at TuiChain verifies that this is the case, the crowd sale contract begins allowing investors to redeem their tokens in exchange for a proportional amount of Dai deposited through the student's payments.

3.2 Secondary market

The TuiChain platform also provides a marketplace where investors can post token sell positions. Other investors can then browse through existing positions using specialized search features such as filtering by loan and student attributes.

Briefly, investors post sell positions (e.g., sell 90 tokens for 100 Dai) by transferring their tokens to a designated marketplace smart contract, which causes the position to appear in the platform's marketplace page. Other investors can then purchase those tokens through the same page using their wallets, sending the requested amount of Dai plus a proportional fee (e.g., 102 Dai for the full 90 tokens with a 2% fee) to the marketplace contract and obtaining ownership of the respective tokens. The fee amount is transferred to a TuiChain account and remaining Dai to the seller. Sell positions may also be removed by the seller, causing non-purchased tokens to be transferred back to them.

4 Requirements and Design

4.1 Concepts and terminology

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 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 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.

Income Share Agreement (ISA)
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.

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

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

4.2 Requirements

4.2.1 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. no: F01 **Type:** 9 (Functional)

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 first name, last name, email and password, then I'm registered on the system.

Originator: Brainstorming **Priority:** Must

Req. no: F01-b **Type:** 9 (Functional)

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:** Won't

Req. no: F02 **Type:** 9 (Functional)

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

Req. no: F02-b **Type:** 9 (Functional)

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:** Won't

Req. no: F03 **Type:** 9 (Functional)

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

Req. no: F04 **Type:** 9 (Functional)

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

Req. no: F05 **Type:** 9 (Functional)

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 that 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

Req. no: F06 **Type:** 9 (Functional)

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

Req. no: F07 **Type:** 9 (Functional)

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

Req. no: F08 **Type:** 9 (Functional)

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

Req. no: F09 **Type:** 9 (Functional)

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

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

Originator: Brainstorming **Priority:** Must

Req. no: F10 **Type:** 9 (Functional)

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

Req. no: F11 **Type:** 9 (Functional)

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

Req. no: F12 **Type:** 9 (Functional)

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

Req. no: F13 **Type:** 9 (Functional)

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

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

4.2.2 Non-functional requirements

Req. no: NF01 **Type:** 10 (Look and feel)

Description: The application shall have a simple and elegant feel, 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

Req. no: NF02 **Type:** 11 (Usability and humanity)

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 complete given list of tasks within 30 minutes.

Originator: Brainstorming **Priority:** Must

Req. no: NF03 **Type:** 12f (Performance: Capacity)

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

Req. no: NF04 **Type:** 12g (Performance: Scalability or extensibility)

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

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

Originator: Brainstorming **Priority:** Must

Req. no: NF05 **Type:** 13 (Operational and environmental)

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

Req. no: NF06 **Type:** 14 (Maintainability and support)

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

Req. no: NF07 **Type:** 14 (Maintainability and support)

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

4.3 Design models

This section includes the most generic models of the application, to summarize and group the components that will be necessary for the project to work.

4.3.1 Domain diagram

An analysis of the proposed requirements, and the operating logic of the system, we decided to summarize the product domain in the following diagram 4.1.

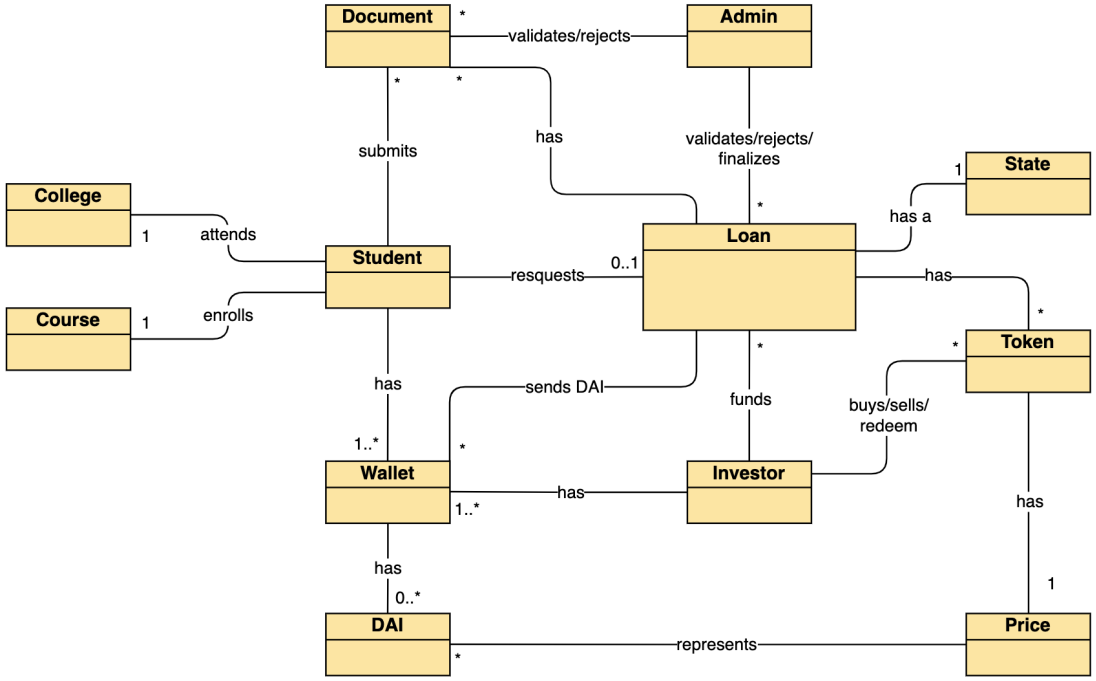

Figure 4.1: Domain diagram

The reading of this diagram can start at the student. The student attends a college and enrols a course of his/her choice. Since the student cannot afford to pay the fees for this course, he will have to apply for a loan. To supplement the request, the student may submit documents proving that the request is real, as well as during the course the student may submit documents proving the completion of subjects, or even good grades he has in some of them. Before the loan can start being financed, it must be approved by the admin.

A loan has associated with it a state that dictates whether the loan is pending, in the financing phase, in the active phase, in the finalized phase, etc., to have different actions on the loan depending on its state. Besides, a loan holds a number of tokens, which are transferred to an investor when he finances a student. And later, they can be bought or sold on the secondary market. In the last state of the loan, these tokens can be redeemed by whoever holds them to be exchanged for their monetary value, represented by the DAI virtual currency. Still, on the loan, it should be the admin who ensures that the payments are being made by the students, and should finalize the loan when the payments are finished.

For all this to be possible, both the student and the investor must have a wallet, which allows them to interact with the entire system.

4.3.2 Application structure

To meet the requirements, we proposed the following high-level architecture presented in figure 4.2, which allowed us to create teams to work independently on each component.

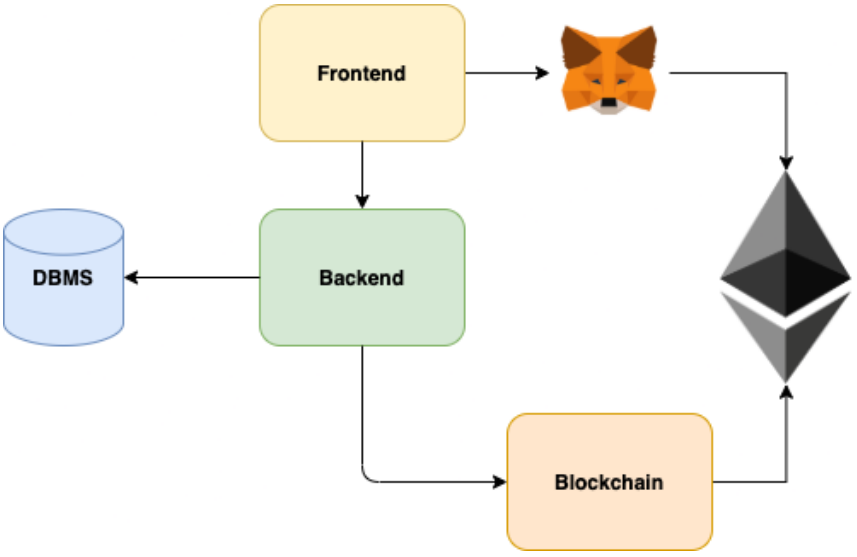

Figure 4.2: High-level application structure

Basically, this architecture, besides showing the necessary components for the whole system to work, also shows the relationship of dependencies that exists in the various components. The key component of the architecture is the external structure of Ethereum since that's where all the logic of smart contracts and data related to them will reside. The rest of the components are built also, to use the data residing in the Ethereum's Blockchain.

The Blockchain component contains the smart contracts that are deployed in the Ethereum network, as well as an entire library with access functions to the Ethereum blockchain, being therefore dependent on the Ethereum component.

The Backend component integrates and uses the functions of the blockchain component and is therefore dependent on it. Furthermore, it is in this component that the business logic and data access resides, dealing with customers requests, accessing the database if necessary, or the Ethereum network so that it can produce a response to the Frontend.

Finally, the Frontend is the data presentation component and is dependent on the Backend so that it can deliver the data to be shown to customers. This is the edge of the system, the one with which the end-user interacts. This component also depends on Metamask [1], which works as a crypto wallet and as a gateway for Blockchain applications.

These components will be covered in-depth in the following sections.

[1] https://metamask.io/

5 Development Methodology

To better organize and optimize the development process, the application code was stored in the version control system GitHub. GitHub allows contributors to interact, commenting and reviewing each others code, allowing cooperation and better programming practices.

5.1 Teams

The team was divided into groups, each focusing one of the Tuichain's components described in section 4, (one group was also dedicated to the application's specification and business model, which we called *Tokenomics*) as follows:

- **Frontend**: Pedro Moreira, Pedro Ferreira, Tiago Sousa
- **Backend**: Henrique Pereira, Ricardo Caçador, Rui Ribeiro
- **Blockchain**: Alberto Faria, Alexandru Domente, Nelson Sousa
- **Tokenomics**: João Silva, Nelson Sousa, Ricardo Milhazes

The Tokenomics team was lately divided and its members spread across the other teams.

5.2 GitHub organization

As said, GitHub was used as a VCS. There, we created an organization called Tuichain[2] where the members were assigned to teams (see figure 5.2) and the repositories created for each of the components, with an extra one for easier deployment configurations, as reflected on figure 5.1.

Later, repositories for the application's showcase (*stand*) and documentation were also added.

5.3 GitFlow

As a way of coordinating and guaranteeing code's quality, all the teams adopted a GitFlow[3] adaptation. This workflow allowed also the documentation of new features and a better workload division between team members.

The workflow for each new feature or bugfix followed the next steps:

1. **Issue Creation**: an issue should be created, describing the new functionality that needed implementation (or existing bug that needed fix)

[2] https://github.com/TuiChain
[3] https://www.gitflow.com/

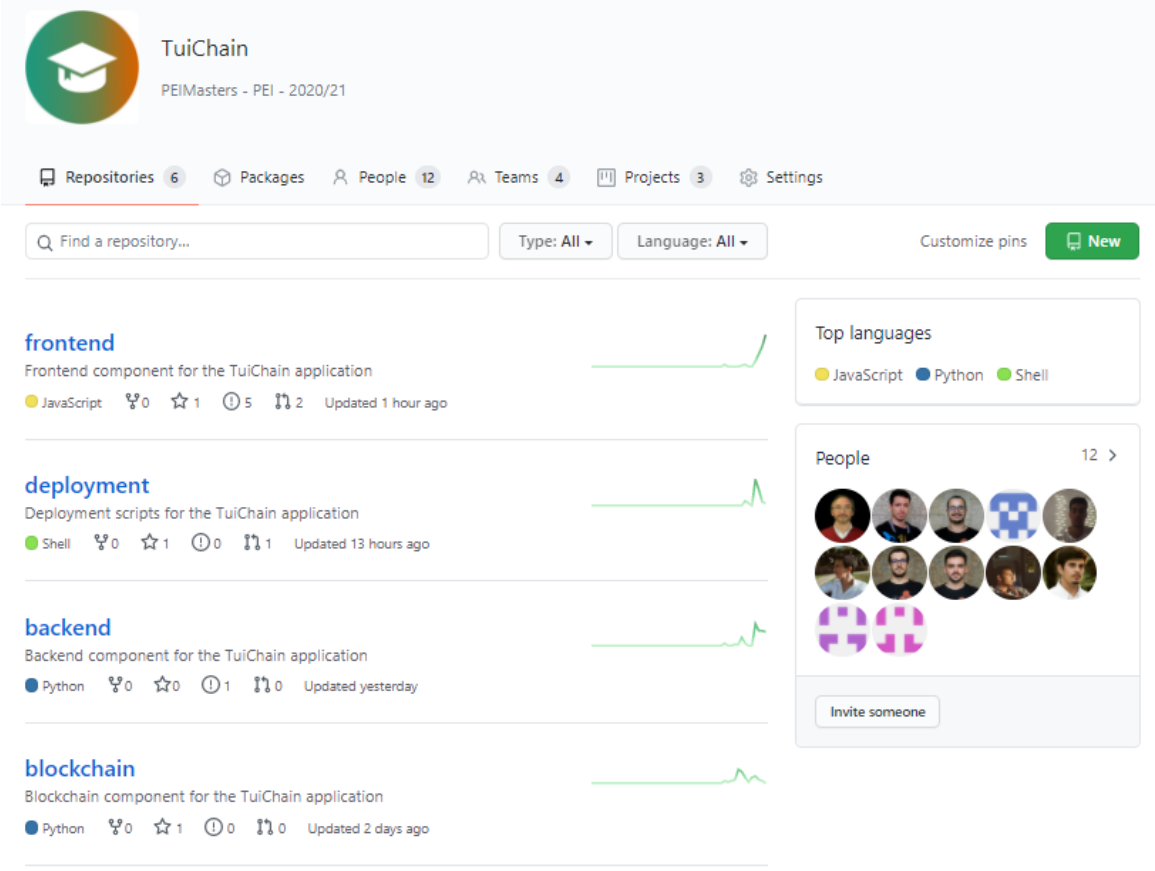

Figure 5.1: TuiChain GitHub organization page

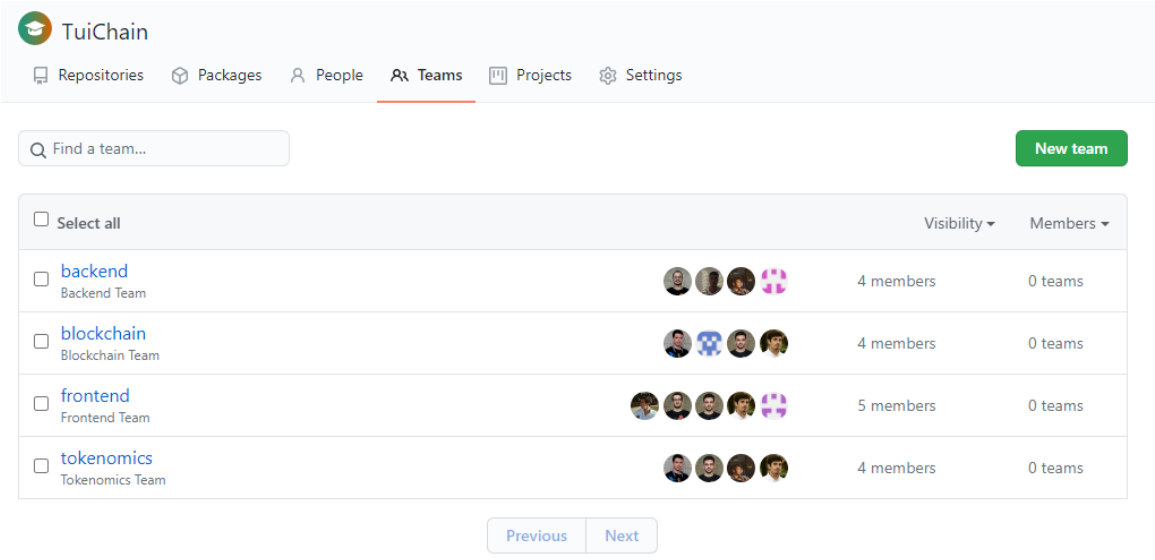

Figure 5.2: TuiChain GitHub teams

2. **Branch Creation**: a branch should be created for each issue, with the name "<first letter of name><first letter of surname>/<functionality or bugfix name>". For example, if John Doe wanted to add to the backend an image upload feature he would have to create a branch called "jd/image_upload", for example.

3. **Feature Implementation**: the developments made to address a given functionality should be committed only in the related branch, with meaningful messages describing each commit.

4. **Pull Request**: when the development is done, a pull request should be done to the *main* branch. Before the pull request, a rebase with the main branch needs to be done if any changes to the main branch were made, making sure the feature branch is updated with the latest version.

5. **Code Review**: before merging the code related to a new functionality or bugfix, the code should be reviewed by at least two other collaborators, improving this way the code's quality and allowing knowledge sharing.

6. **Squash and Merge**: when the code is reviewed and approved, the commits should be squashed and then merged into the main branch, leading to a cleaner evolution understanding of the repository.

This workflow is represented in figure 5.3, without the issue creation, pull request and code review steps.

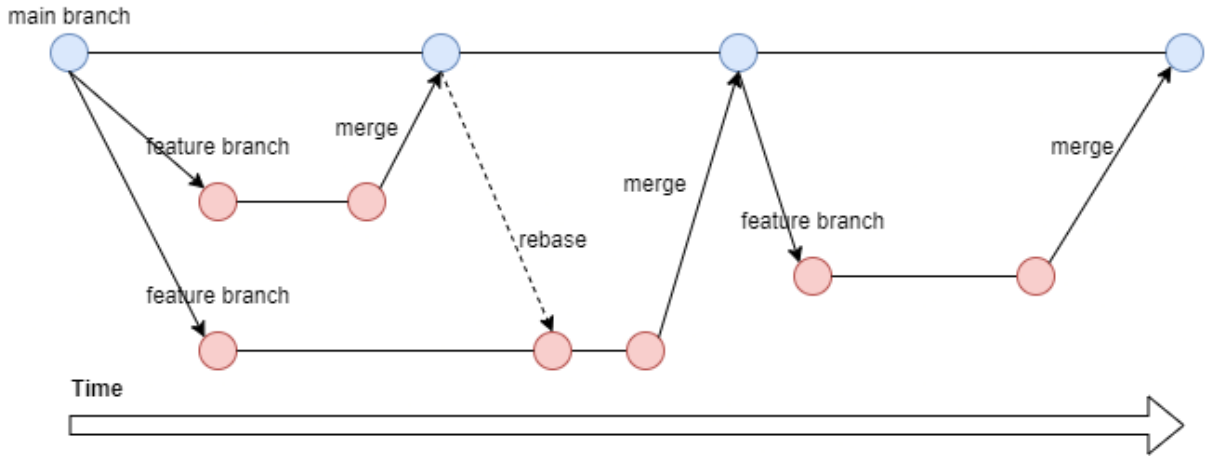

Figure 5.3: Git workflow used by the development teams

When in production, a development branch should be created and, from there, creating the feature branches and bug fixes, respecting the official GitFlow. The merges would then be made into the development branch and only then to the main branch when released to production. Thus, code in the main branch would correspond to the version available in the production environment. Hotfix branches could be created directly from the main

branch and merged once they are approved, as their objective is to provide a fast solution to an existing problem causing service disruption or malfunction.

5.4 Continuous integration

To guarantee code consistency and correctness, a set of rules were added as GitHub Actions, which were verified every time a pull request was done.

These rules consisted of checks done when compiling the code, for example, formatting checks according to predefined standards, running the written tests (logging how many have passed and failed) and checking the dependency versioning. These rules could also verify specific requirements, e.g. in the backend repository a check was made to verify if *makemigrations* was run in the Django component to verify there were no uncommitted changes to the database.

A log of builds is also available on GitHub, with the workflows specified for each commit after a pull request, with a description of each step's results. An example of such log is available on figure 5.4.

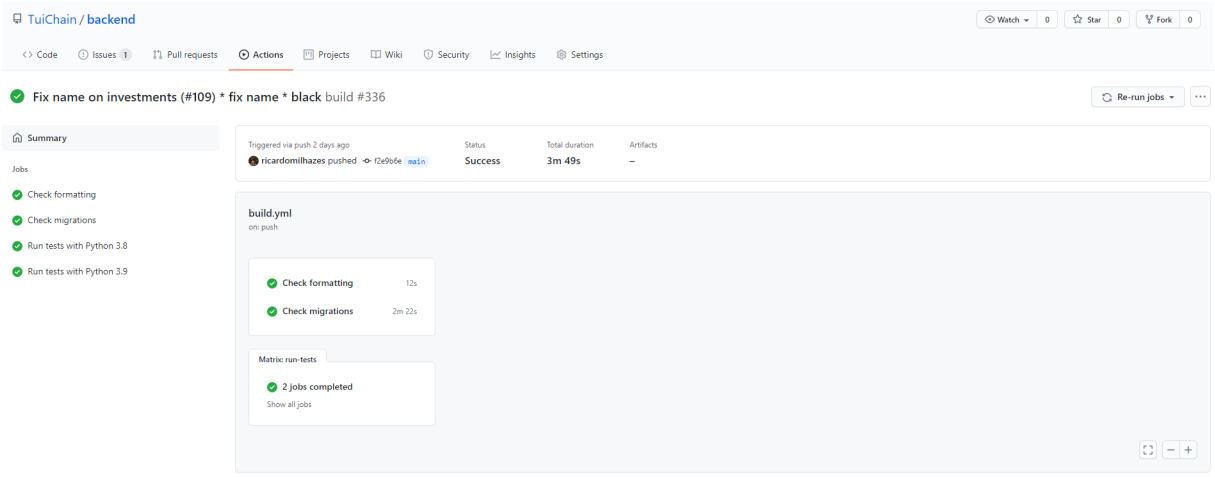

Figure 5.4: GitHub Actions: build checks log example

6 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 into 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; the last one being the actual implementation of the solution.

6.1 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 6.1 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 do 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.

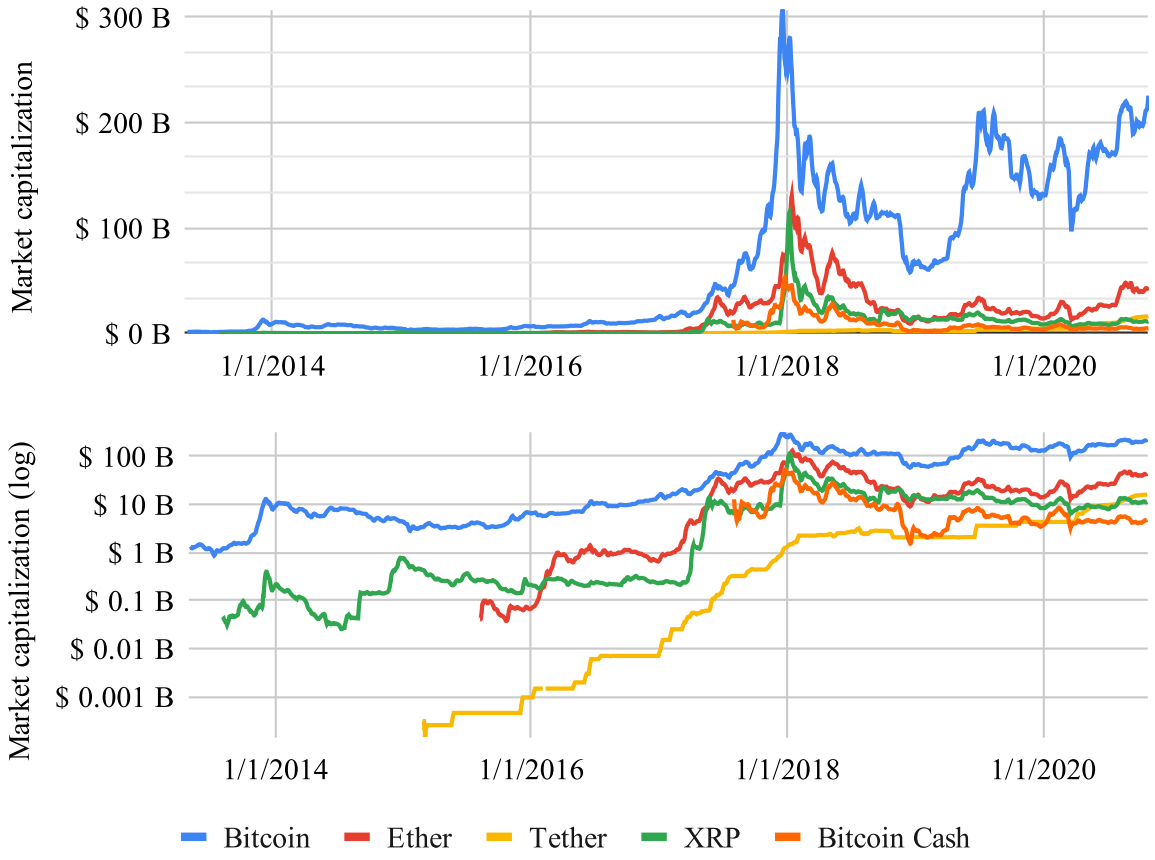

Figure 6.1: **Cryptocurrency market capitalization.** Market capitalization in USD of the 5 top cryptocurrencies from Apr. 28, 2013 to Oct. 23, 2020 (7-day moving average of daily values; top: linear y-axis scale; bottom: logarithmic y-axis scale).

6.2 Design and Specification

The Blockchain project will have a package and folder structure as the one represented in the Package Diagram (figure 6.2), with the clear distinction of the two different parts which are the access library, and the Smart Contracts.

According to figure 6.2, 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.

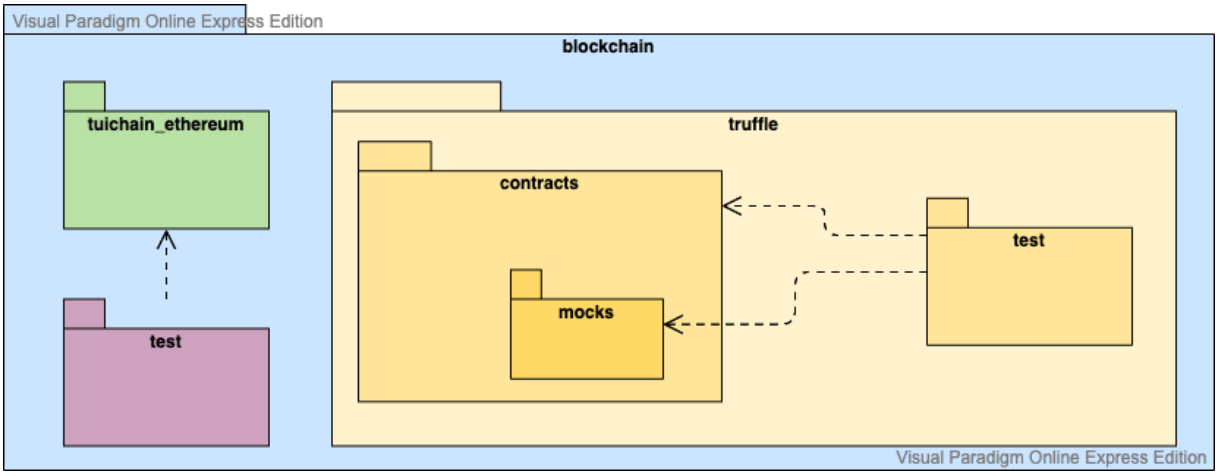

Figure 6.2: Package diagram

6.2.1 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:

- 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 6.3, 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.

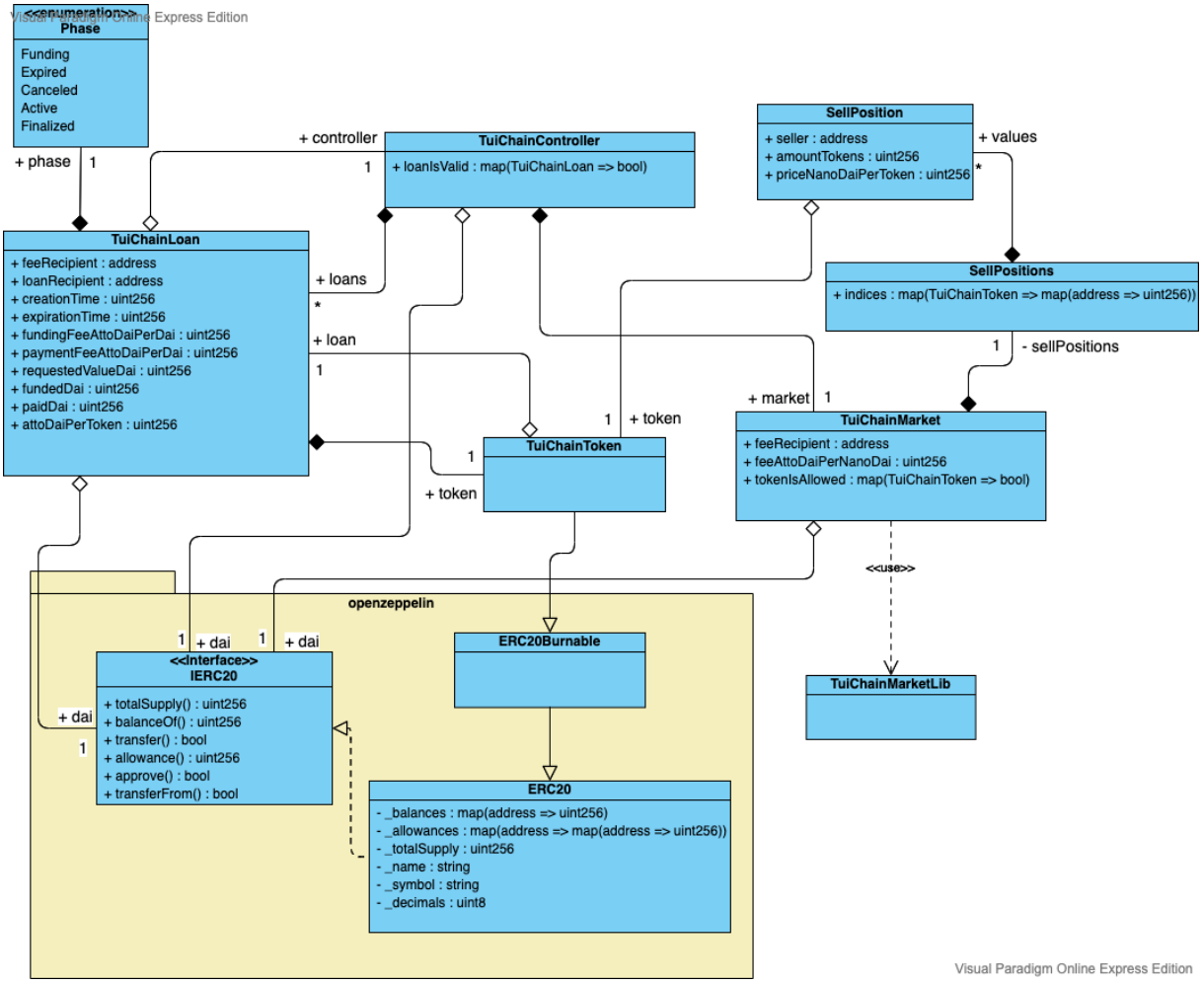

Figure 6.3: Simplified class diagram of smart contracts

The figure 6.4, represents a detailed class diagram that takes into account all the methods declared or implemented in each class, as well as library usages.

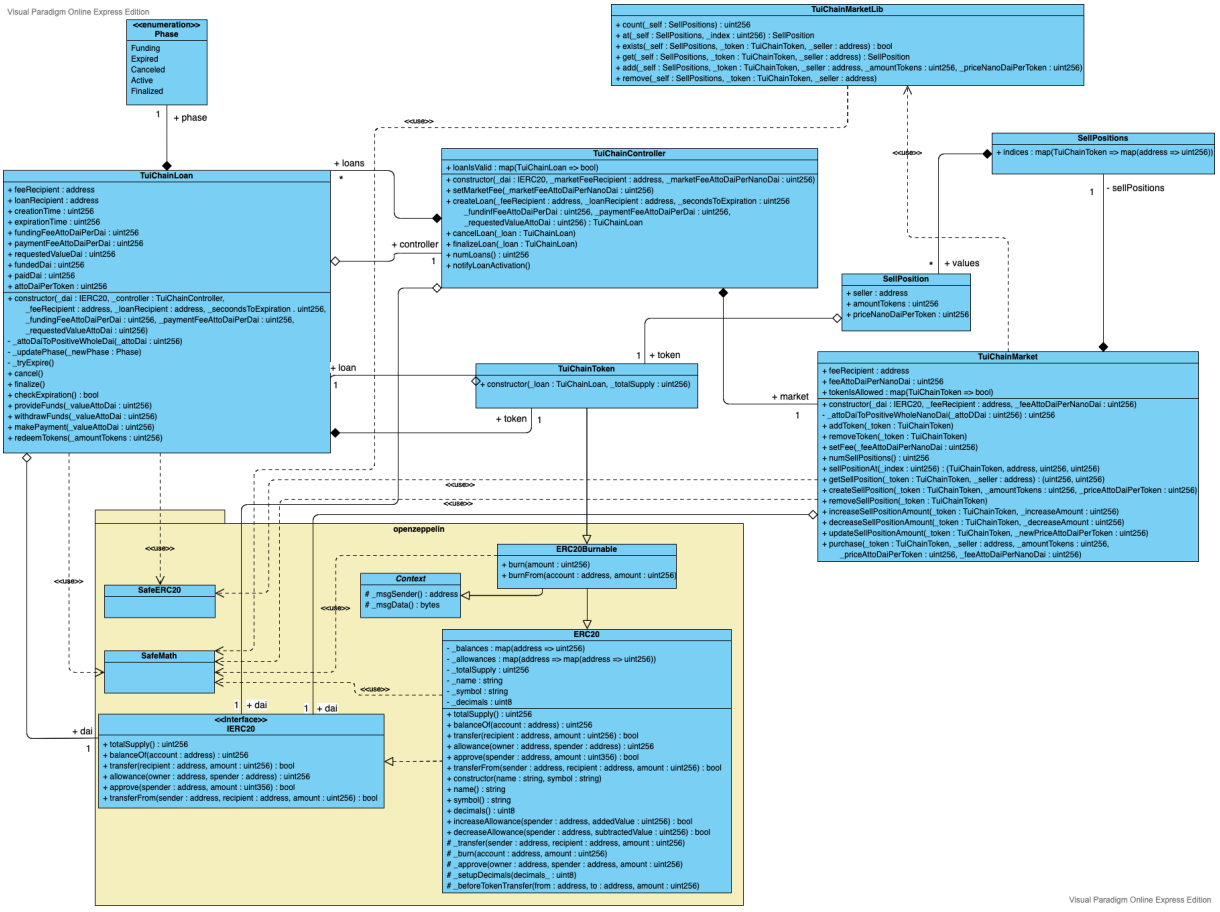

Figure 6.4: Class diagram of smart Contracts

When all the infrastructure is deployed in the Ethereum network, only three types of contracts will be accessible, and the public interfaces of them are represented in the figure 6.5.

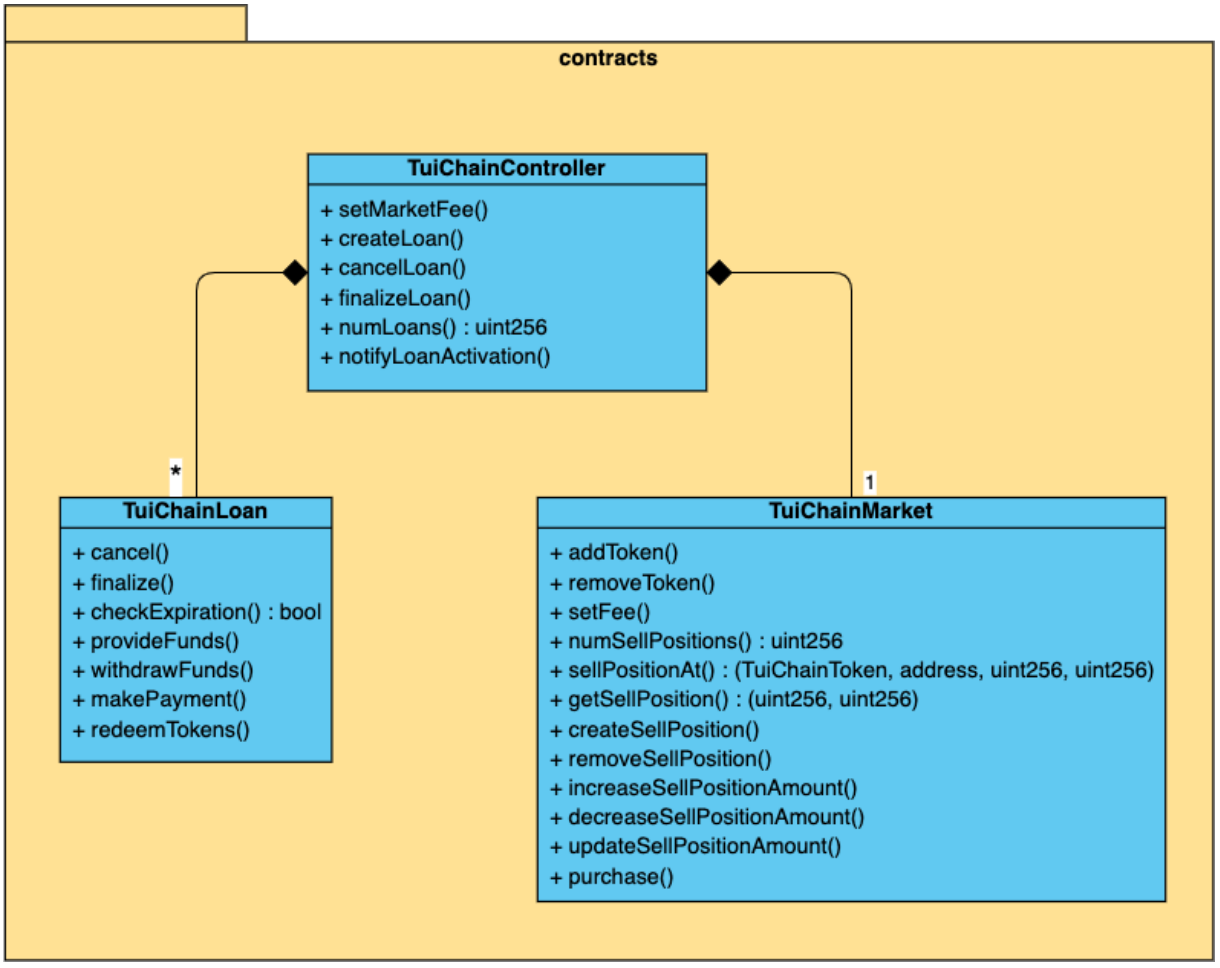

Figure 6.5: Class diagram of the public interfaces of smart contracts

In addition to structural diagrams, the Loan contract presents 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 6.6 presents a state machine diagram, which represents the Loan Contract behaviour, reduced to states, and also the events responsible for the transitions.

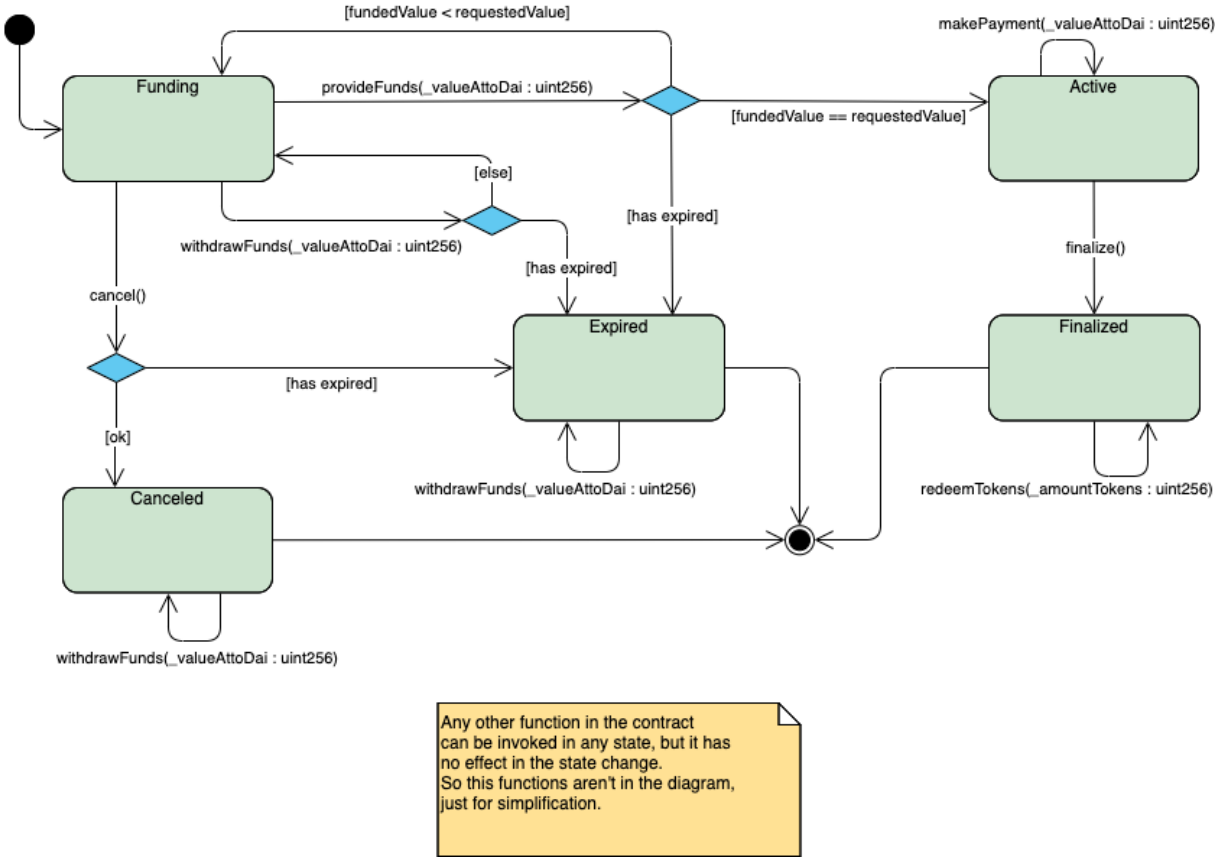

Figure 6.6: State machine diagram of the loan management smart contract

6.2.2 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.

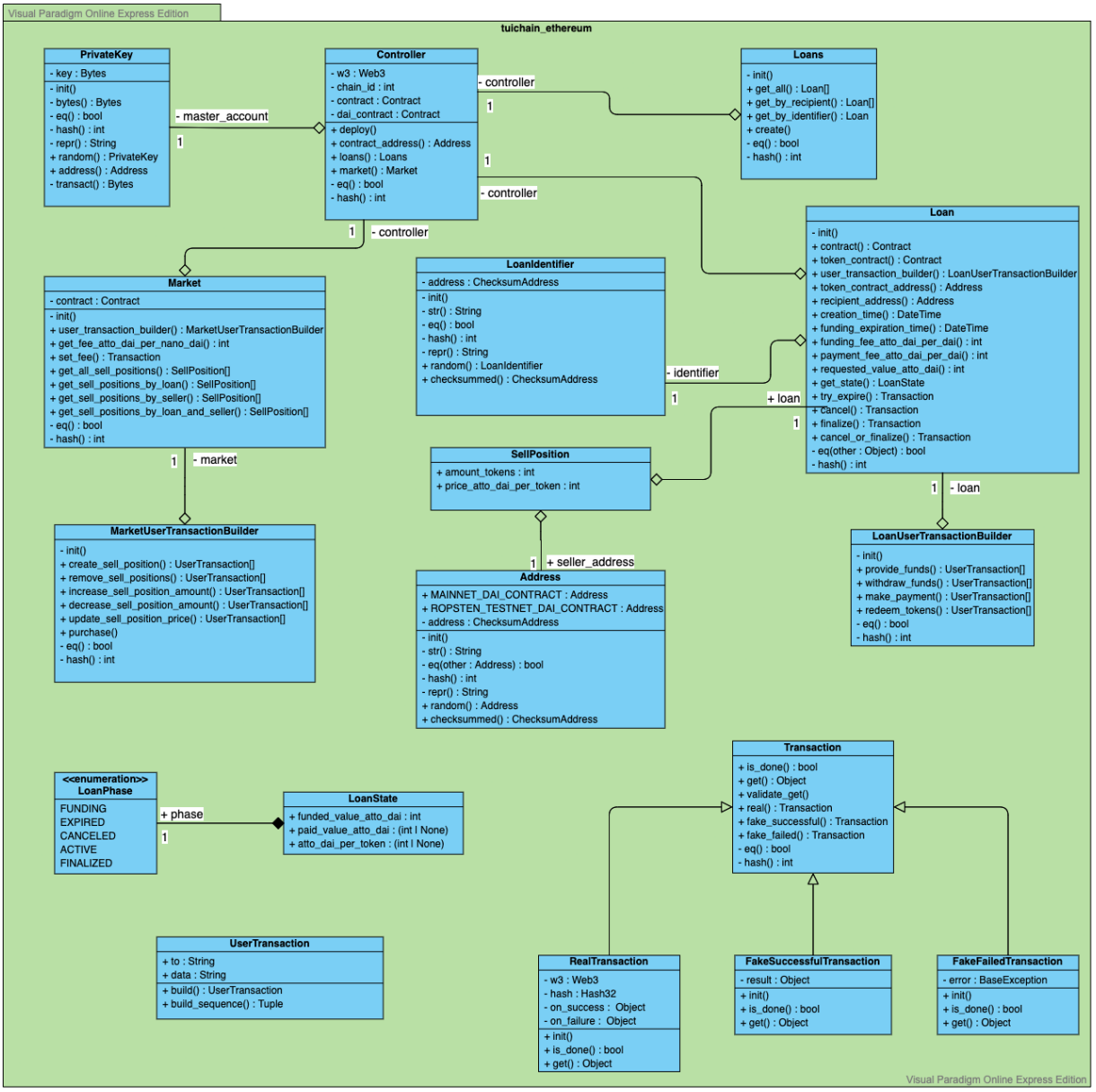

Figure 6.7: Class diagram of the access layer

This access library can be reduced to the class diagram presented in figure 6.7. 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 the same as the ones in contracts, but these 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.

6.3 Implementation

Standard safety and security guidelines were following during the 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 OpenZeppenlin's SafeMath library.

7 Backend

The development of this phase is also divided into 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 is 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 mid-course, and describing how the app runs with the backup of tests to support it.

7.1 Research

7.1.1 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 an admin panel which makes the job of managing models and test methods created much easier than other frameworks. Regarding security, it includes prevention of common attacks like Cross-site request forgery and SQL injections. Another advantage present is the 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 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 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 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 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, its asynchronous nature rises other problem which is callbacks. Since every task has a callback (a 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 the programming language and uses its latest features like namespaces, anonymous functions, interfaces, etc. It also offers great documentation making it 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 the 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 know 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 can 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 can 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 to develop 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, they have their differences and situations 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 a 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.

Selected framework. 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.

7.1.2 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 modelling 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 if 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 unrivalled by current NoSQL alternatives.

On the other hand, NoSQL databases do not a 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 into 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.

7.2 App development

7.2.1 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 set up 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 are 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.

7.2.2 Database models

After a team analysis of the project, we came up with a model for our project consisting of three entities: Students, Investors and Loans. This model was used in the first approaches and was our basis at 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 other students, so we grouped both entities into a single one, which we called User. When we began implementing the blockchain functionalities, using the component described in section 6, issues of data consistency were discussed and came into play, as investments on loans and operations related to them occurred asynchronously and thus, storing related information in the backend wouldn't allow us to

guarantee consistency. Therefore, another version of the database was developed, where we removed the investments and added another entity to store information of documents related to each loan.

This latest database model is presented on figure 7.1.

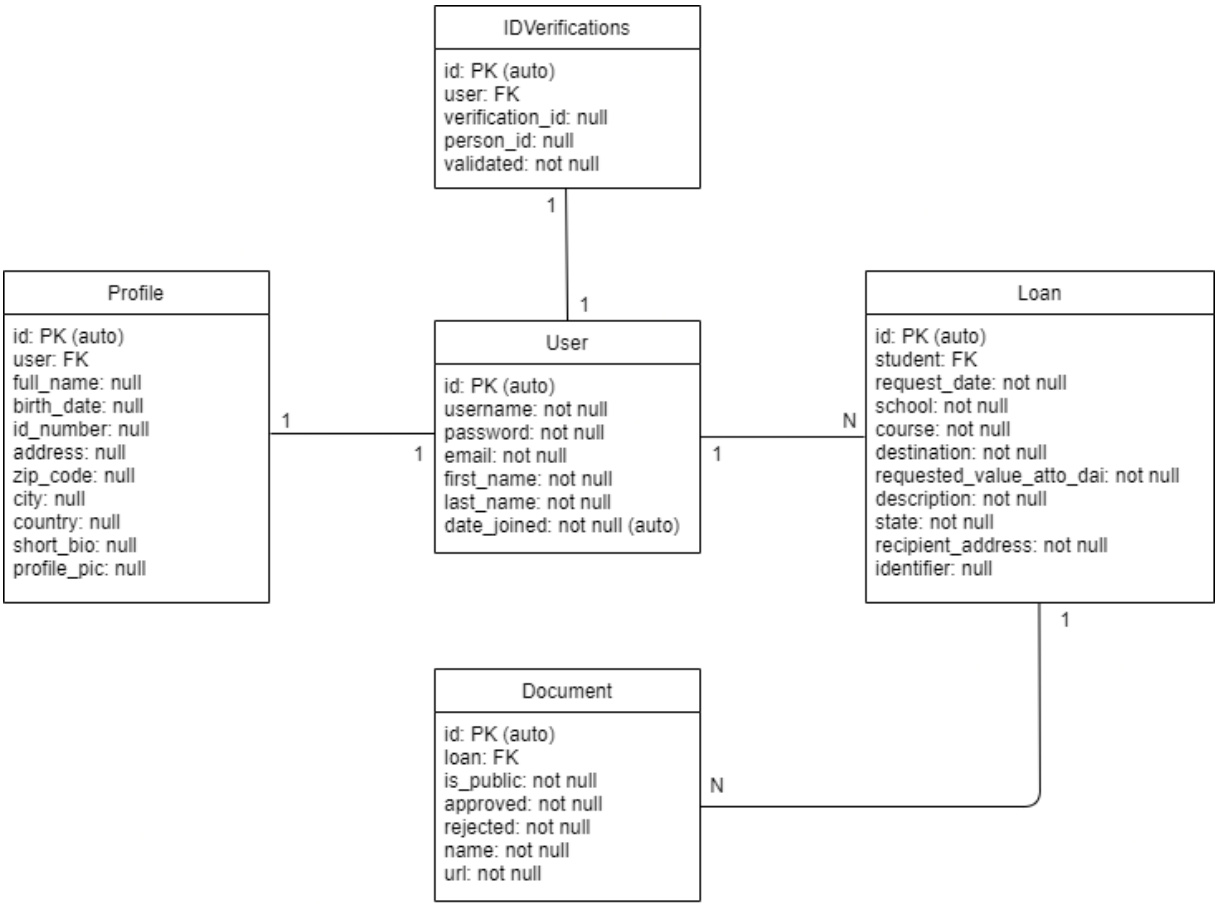

Figure 7.1: Database model

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. Users can be either normal users (students and investors) or admin users (identified by the *is_superuser* field of the predefined User Django's model).

2. Each User has a **Profile** with the personal details, such as full name, birth date, geographical information (country, city, address and zip code), ID number, short biography and URL to the profile picture.

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 **Loans**, described by the request date, the school and the course they want to attend (along with the school's location), the requested value, a loan description, its state (described next), the recipient address (wallet address of the user who requested the loan) and the loan identifier in the blockchain.

5. When requesting a loan, users can also upload **Documents**, to provide more information to the investors or the admins. These documents are related to a specific loan and can be either public or private and in both cases are approved or rejected by the admins. Only public approved documents are visible to other users. Documents are described using their names and URLs.

7.2.3 Loan states

Loans can have the following states:

- Pending: waiting for approval
- Creating: after approval, waiting for Loan to be created by the blockchain component
- Approved: after creation, it becomes approved and all states can be collected on the blockchain component
- Withdrawn: has been withdrawn by the user
- Rejected: has not been approved by an admin

Once they are approved, its state is obtained from the blockchain.

7.2.4 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 information and resources.

7.2.5 Documentation

For documenting Django API we needed a tool that could do just that so we initially had into account four of them, Swagger, ReDoc, Sphinx, and Django Rest Framework built-in documentation.

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 helps 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 this said, we tried to use Swagger since it's one of the most if not the most popular tools when it comes to API documentation, having a package called 'drf-yasg' which is an automatic Swagger generator for Django REST Framework. However, we couldn't implement Swagger due to some errors we couldn't solve related to authentication specifications so we decided to use Django Rest Framework built-in documentation. Every function has a *docstring* with the respective description, parameters and responses so all of them can be understood by everyone.

7.2.6 API Routing

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 to do so.

These endpoints can be divided into 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 include admin routes and user routes. The first ones can only be accessed by admins and are mainly approve/reject routes. The others can be used to fetch information about loans and their documents, other users, secondary-market investments, and others.

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, as seen in Figure 7.2.

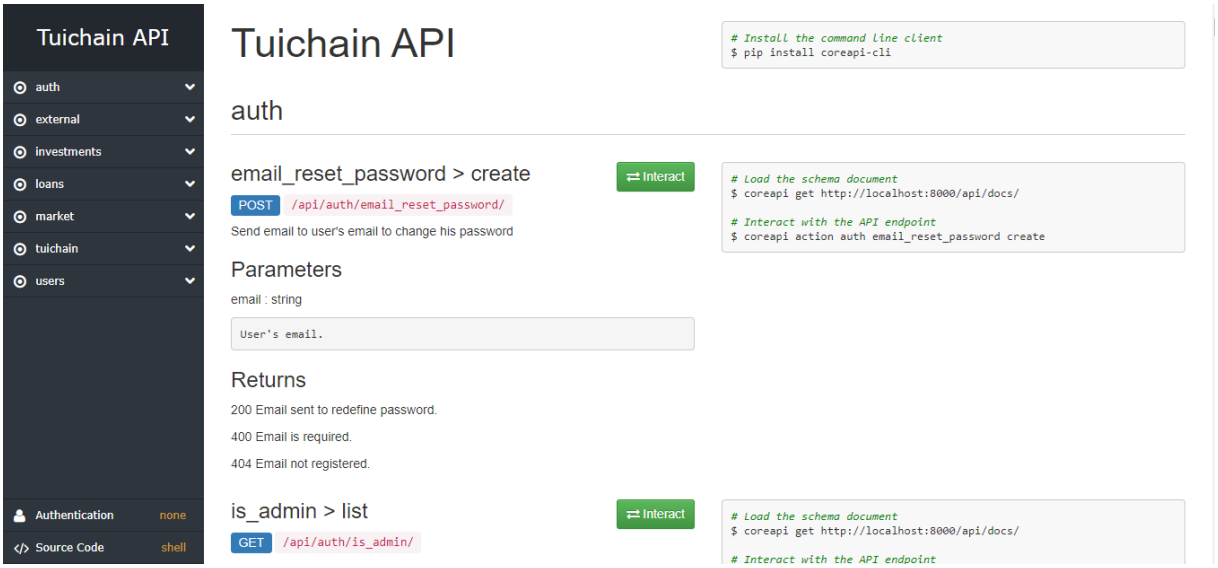

Figure 7.2: API routing documentation example

The complete routing is listed in Appendix A.

7.2.7 Storage

Uploaded profile pictures and documents need to be stored to later be provided to the frontend. Therefore, several options were discussed to solve this problem. Using local storage was discarded as it was not so easily extended and thus reducing scalability, so we opted for a cloud storage solution. Given the fact that Google Cloud Platform offers a 350$ voucher to new users, we decided to make use of it and created a storage bucket in that platform and proceeded to upload the mentioned documents to it, storing in the database the URL made available to access them.

7.2.8 Know Your Customer (KYC)

A requirement for Tuichain was the confirmation of a person's identity since this involves legal contract processes and so the application must keep track of the people and their respective accounts. The tool used for this intent is called Stripe[4]. Stripe is a payments platform, that provides an API for ID verification as a *beta* product[5], which we applied for and access was given to us. We chose Stripe given the fact that it was properly documented and allowed us to use it for free (while we are on the development and testing phase).

Stripe's API works by first asking for a picture of an ID document and a face picture. Then, it verifies the authenticity of the ID documents and compares its picture with the

[4] https://stripe.com/
[5] https://stripe.com/docs/identity

one taken. After that process, the API gives the verdict if the user is verified or not. Finally, the user will have access to the operations granted by the application depending on the previous outcome.

7.2.9 Blockchain access

As said before, the access to the blockchain was done using the Python package developed by the Blockchain team in section 6. The package made available an object called *Controller*, which provided the connection between the backend and the blockchain components and information.

8 Frontend

The development of the interface was divided into two parts: the first involved a research process with subsequent Hierarchical Task Analysis (HTA) [5] and prototypes, while the second was the development of the user interface itself. This chapter describes the process mentioned before in Section 8.1 and the best practices used for developing a good UI/UX in Section 8.3.

8.1 Research

8.1.1 Personas

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

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

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

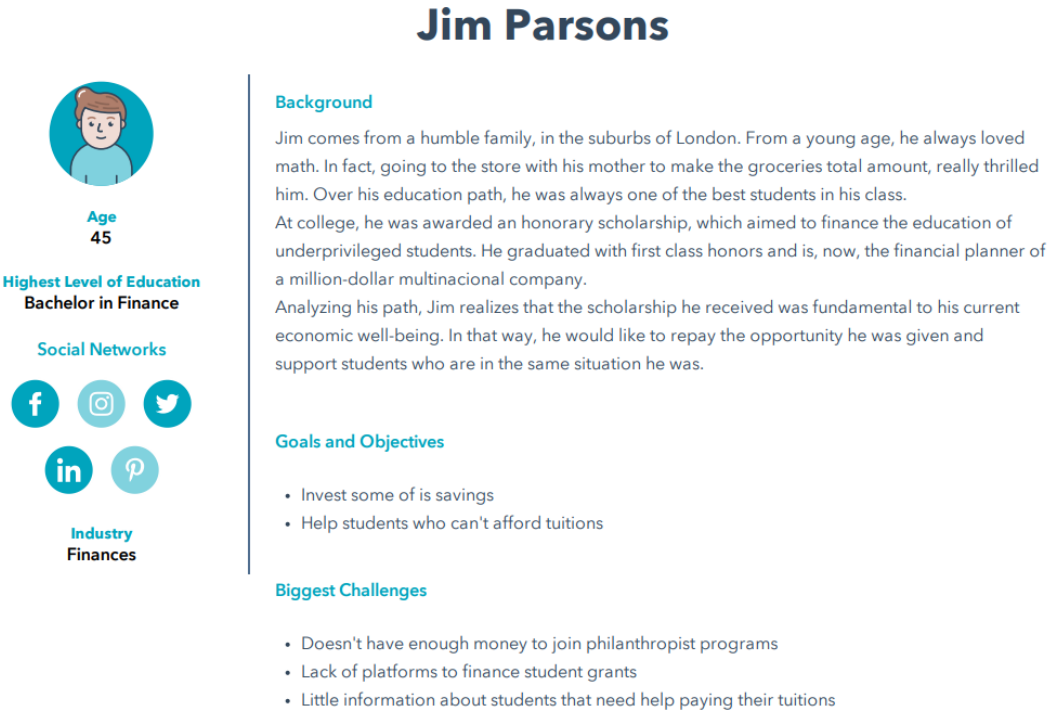

Figure 8.1: First persona

Mayim Bialik

Age
18

Highest Level of Education
High School

Social Networks

Industry
Tecnhology

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 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 8.2: Second persona

Johnny Galecki

Age
33 years

Highest Level of Education
Master's degree

Social Networks

Industry
Technology

Background

Johnny is a technology enthusiast. He is a Backend Developer at a startup that operates in the area of cryptocurrencies. From an early age, he started to follow the development of blockchain technologies and, therefore, cryptocurrencies. He has always believed in the potential of these technologies and faithfully believes that one day, fiat money will be replaced by virtual currency. In addition to having a substantial amount of virtual currency, he has recently replaced several tradicional banking services for decentralized finances services.
DeFi is one of the themes that has most captivated his attention lately. He seeks to support projects in which he believes. TuiChain is the perfect opportunity for him to invest his money in a noble cause.

Goals or Objectives

- Invest in cryptocurrencies
- Support DeFi projects

Biggest Challenges

- Find a DeFi project with a supportive purpose

Figure 8.3: Third persona

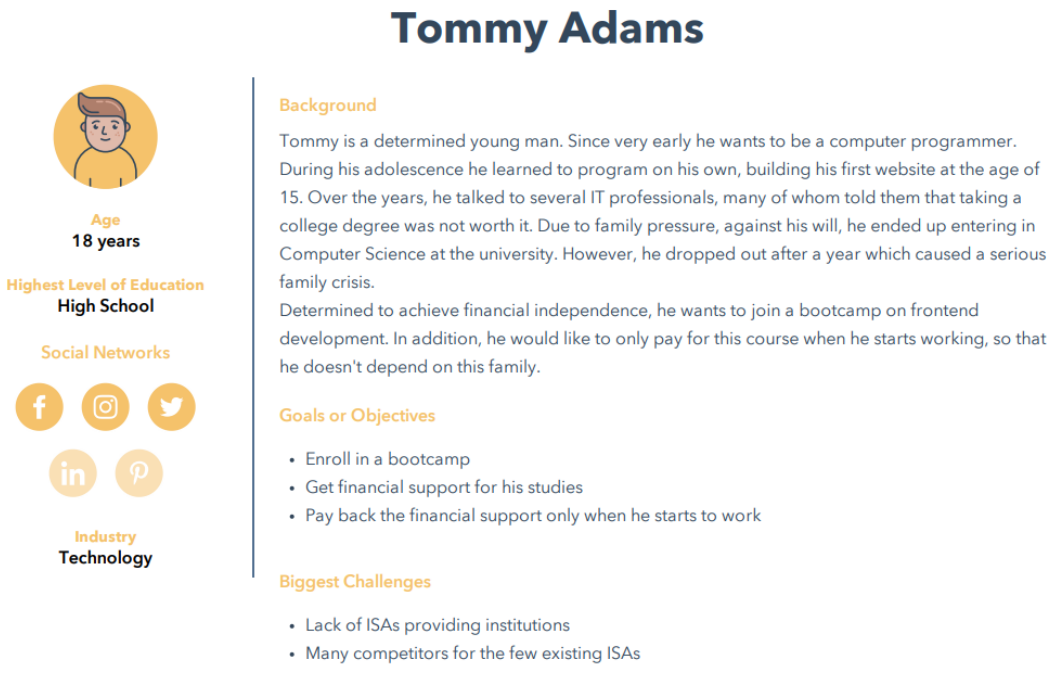

Figure 8.4: Fourth persona

8.1.2 HTA

A task [5] is the human activity that makes it possible to achieve a goal. Analysing possible tasks makes it is possible to understand the behaviour of our users in the context of our system, giving us the information needed 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 8.5) and financing a student (Figure 8.6).

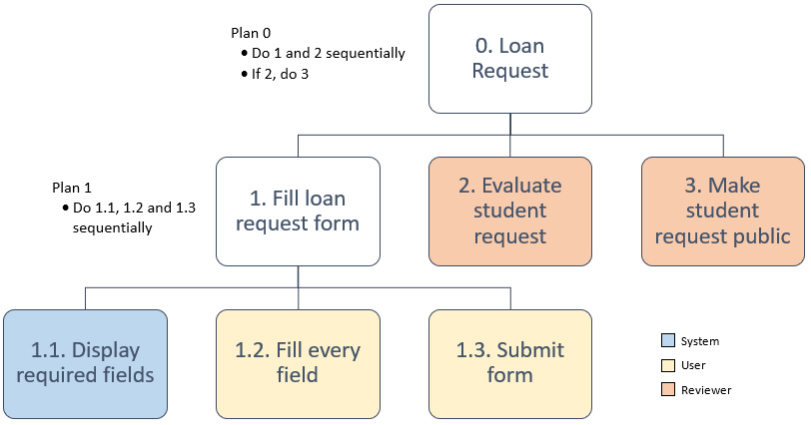

Figure 8.5: Loan request (HTA)

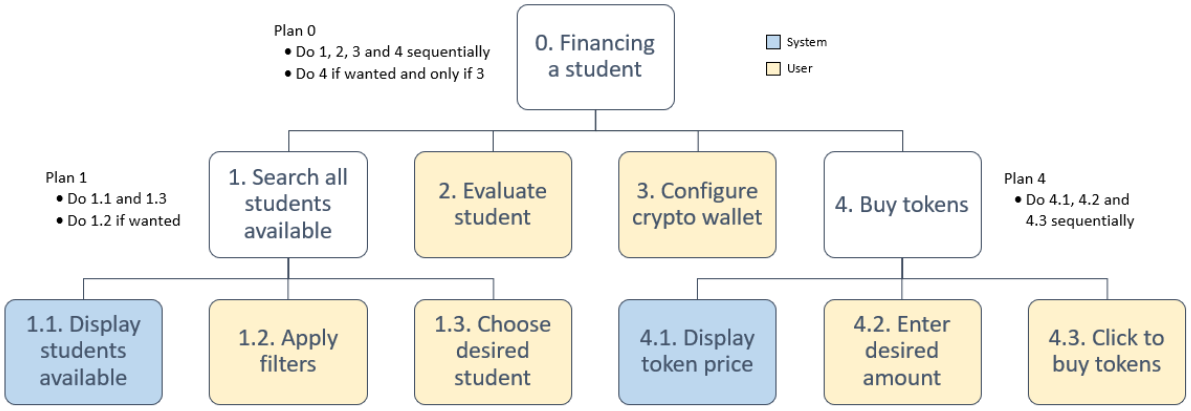

Figure 8.6: Financing a student (HTA)

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 8.7 to Figure 8.12 are, respectively, the following prototypes:

- Student signup
- Student loan request

– Catalogue of students asking for a loan
– A student page (with a loan in progress)
– Secondary market
– A student page (with load finished)

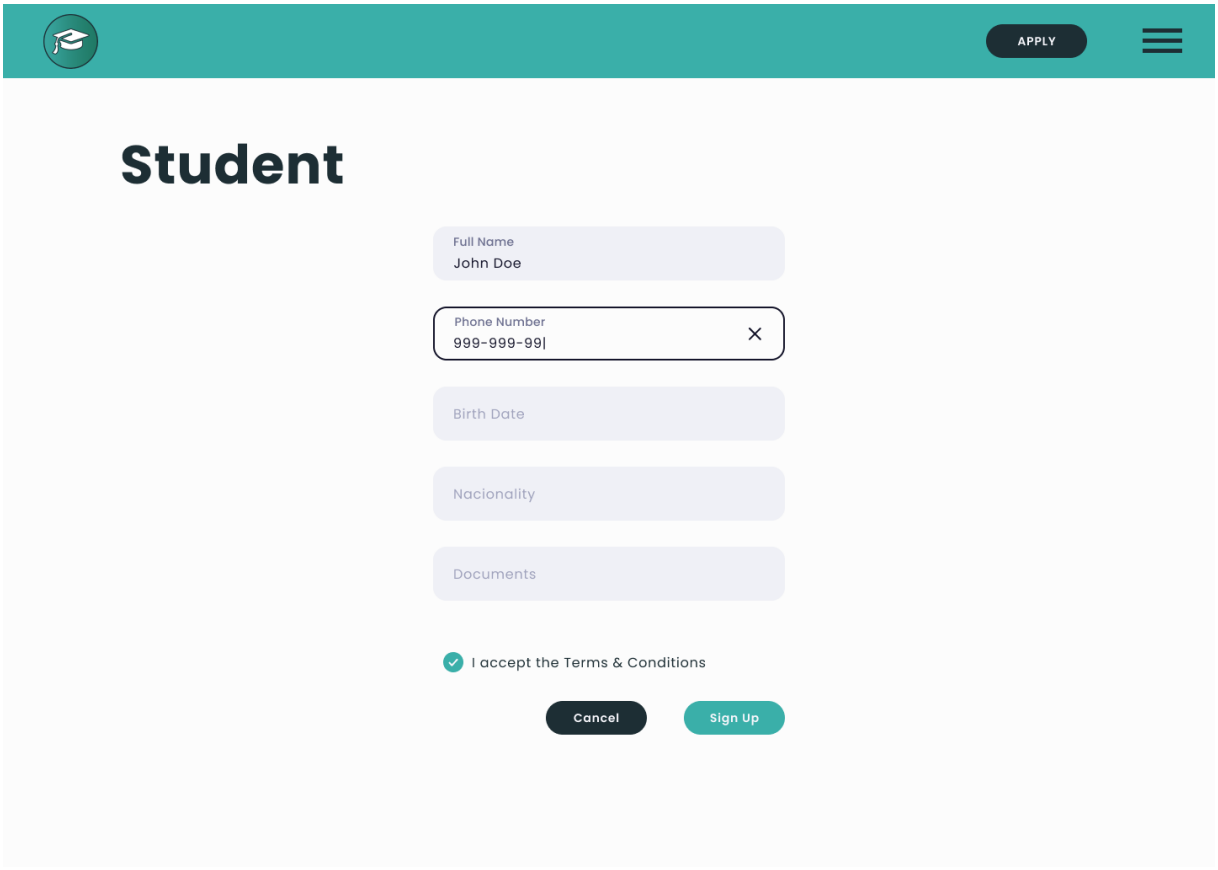

Figure 8.7: Sign up page

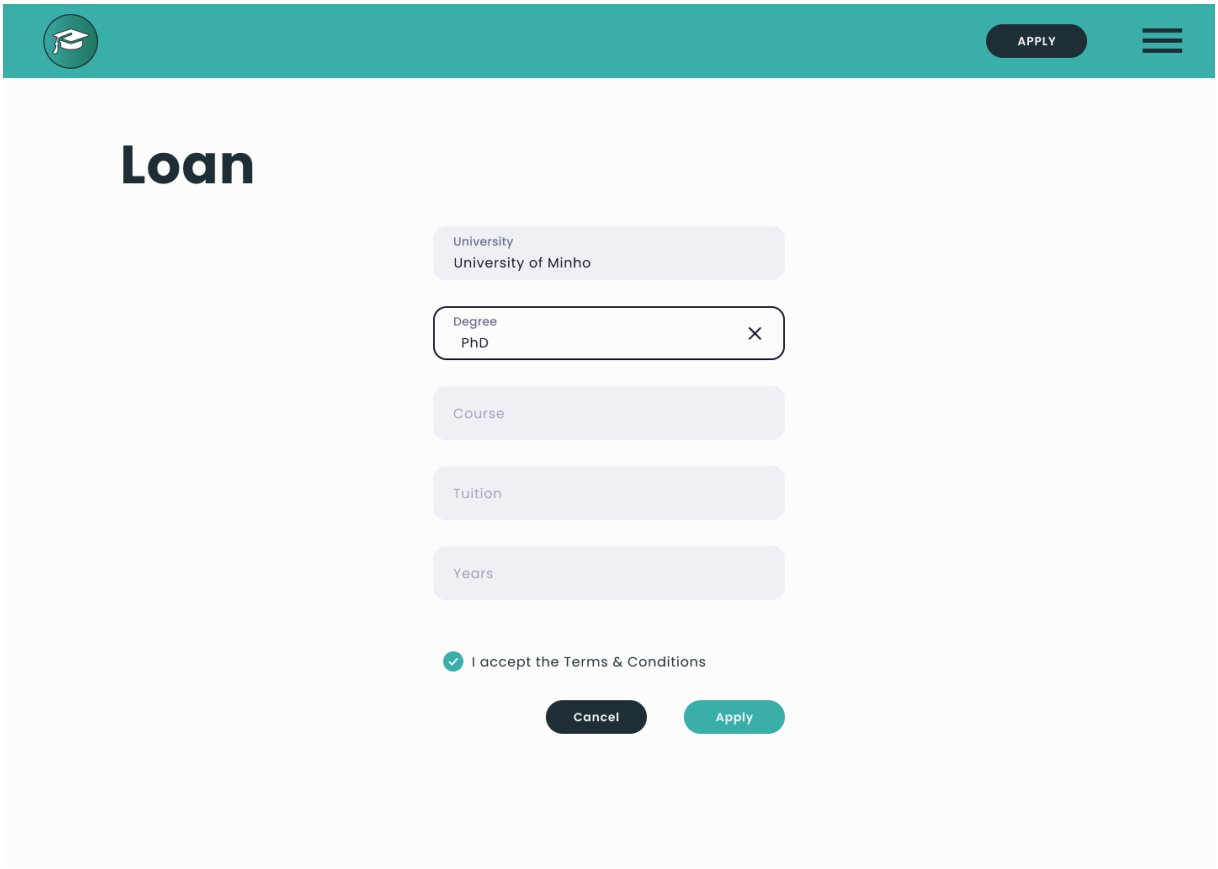

Figure 8.8: Loan request page

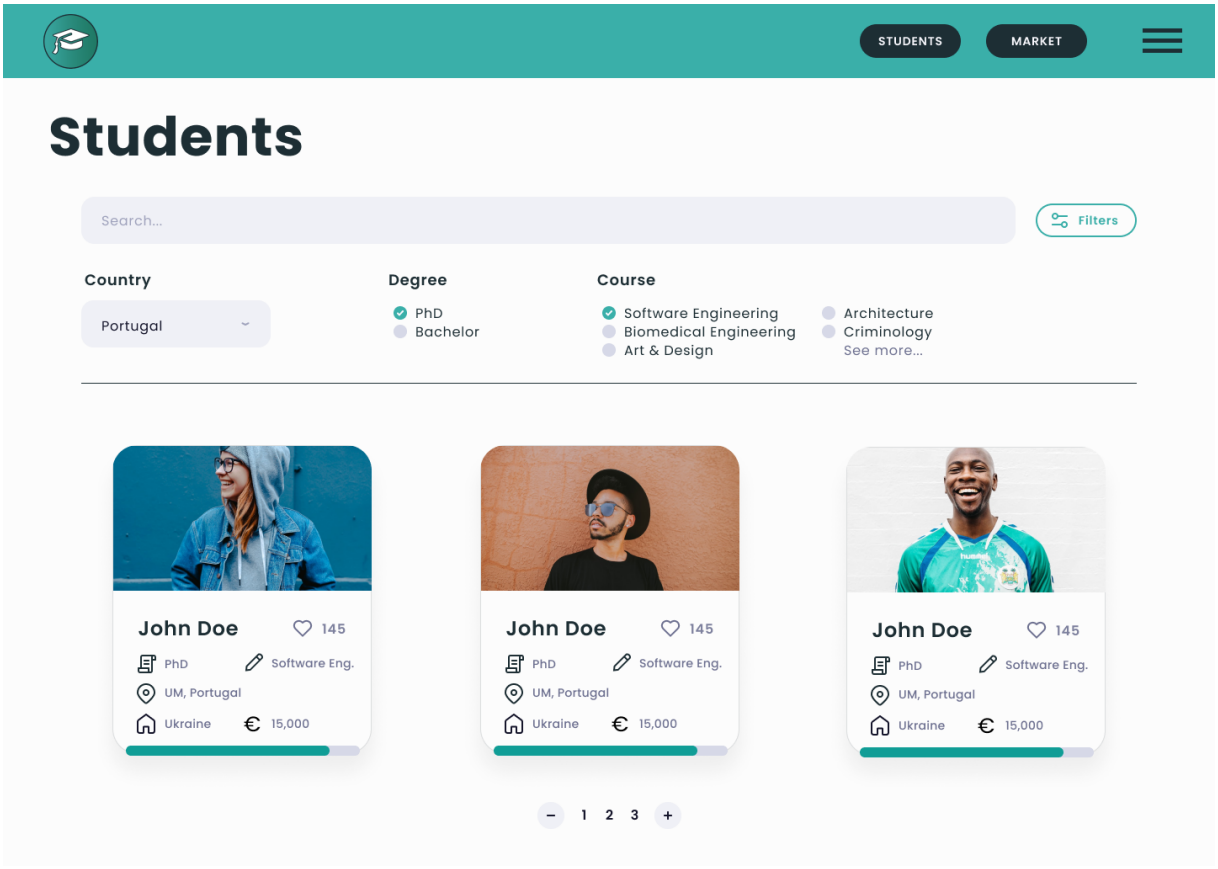

Figure 8.9: Students search page

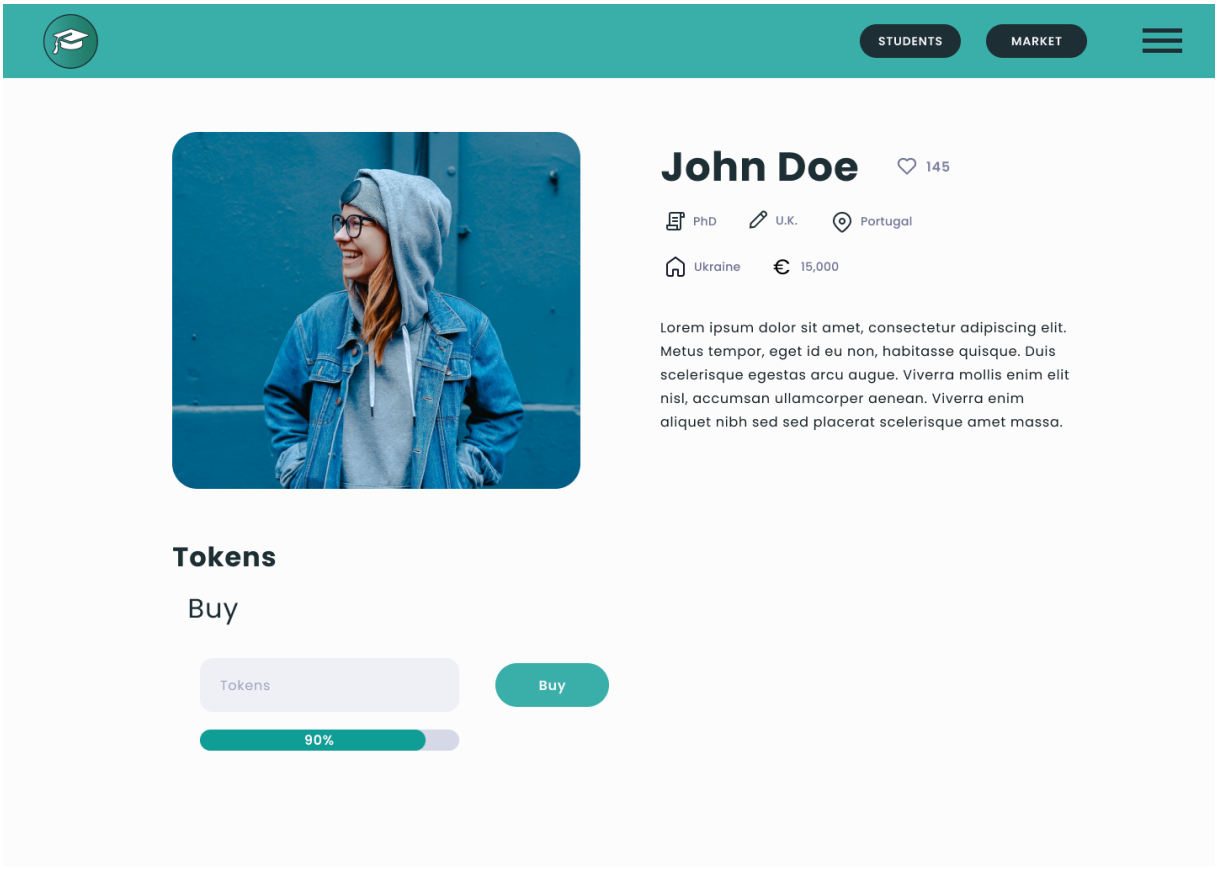

Figure 8.10: Student page

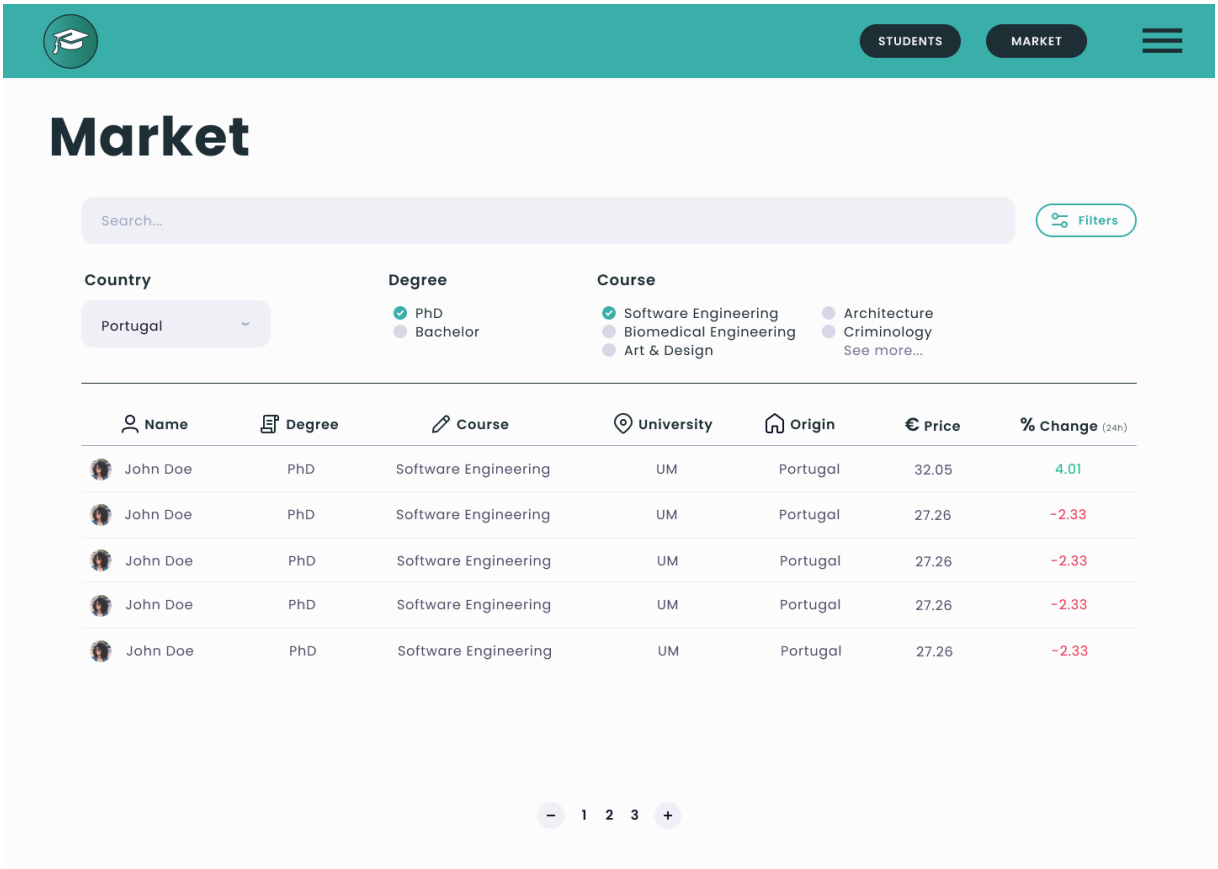

Figure 8.11: Secondary market page

Figure 8.12: Active student page

8.2 Metamask

MetaMask[6] is an essential partner of our frontend since it works as a crypto wallet, and as a gateway to Blockchain applications. It also allows us to buy, save, send and exchange tokens, being available as a browser extension, making it desirable for our system to use it.

Fundamentally, MetaMask injects a global API into websites visited by its users at *window.ethereum*. This API allows websites to request users' Ethereum accounts, read data from blockchains the user is connected to, and suggest that the user sign messages and transactions. The presence of the provider object indicates an Ethereum user.

This way, the frontend can use the *window.ethereum* to get user information from MetaMask or invoke methods at some blockchain node since MetaMask wraps an API of RPCs that aims to facilitate communication with Ethereum nodes.

The way our application interacts with this tool as for the operations done through it is described in the section below.

[6] https://metamask.io/

8.3 UI/UX

Throughout the building process of the user interface, some principles of usability [2] were put in practice. Alongside with these principles, the Nielsen's heuristics [4] were used to evaluate the UI. This chapter will discuss how these principles and evaluation techniques were used in the different stages of our application interface.

8.3.1 Student and investor

Landing page

The landing page is destined to new users of our application. Here is presented the Tuichain platform, its value proposition (Figure 8.13) and how to use it. Since the platform has two types of users, each with its own *Getting started* guide, as seen in Figure 8.14, to achieve the ***Help and documentation*** heuristic.

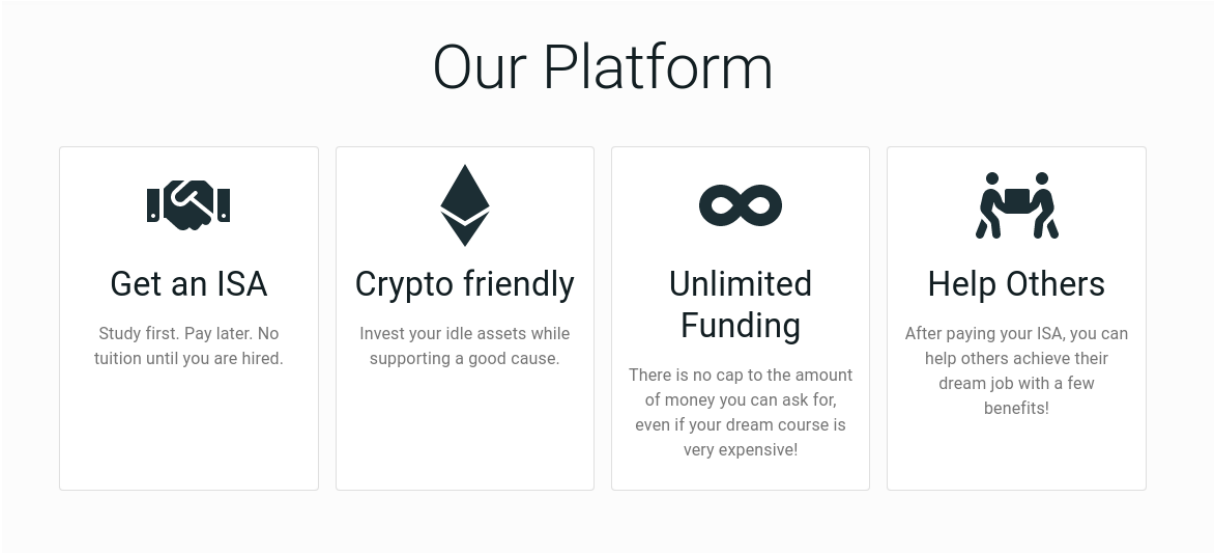

Figure 8.13: TuiChain's value proposition

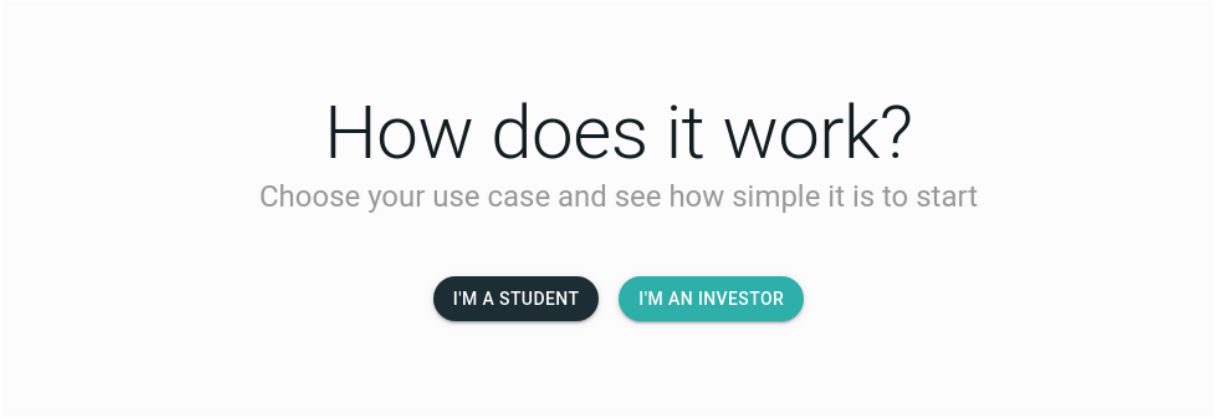

Figure 8.14: Getting started guide

User authentication

To use our application, a user must register. To realize the user authentication (sign up and login), the user is presented to a regular form, illustrated in the following figure.

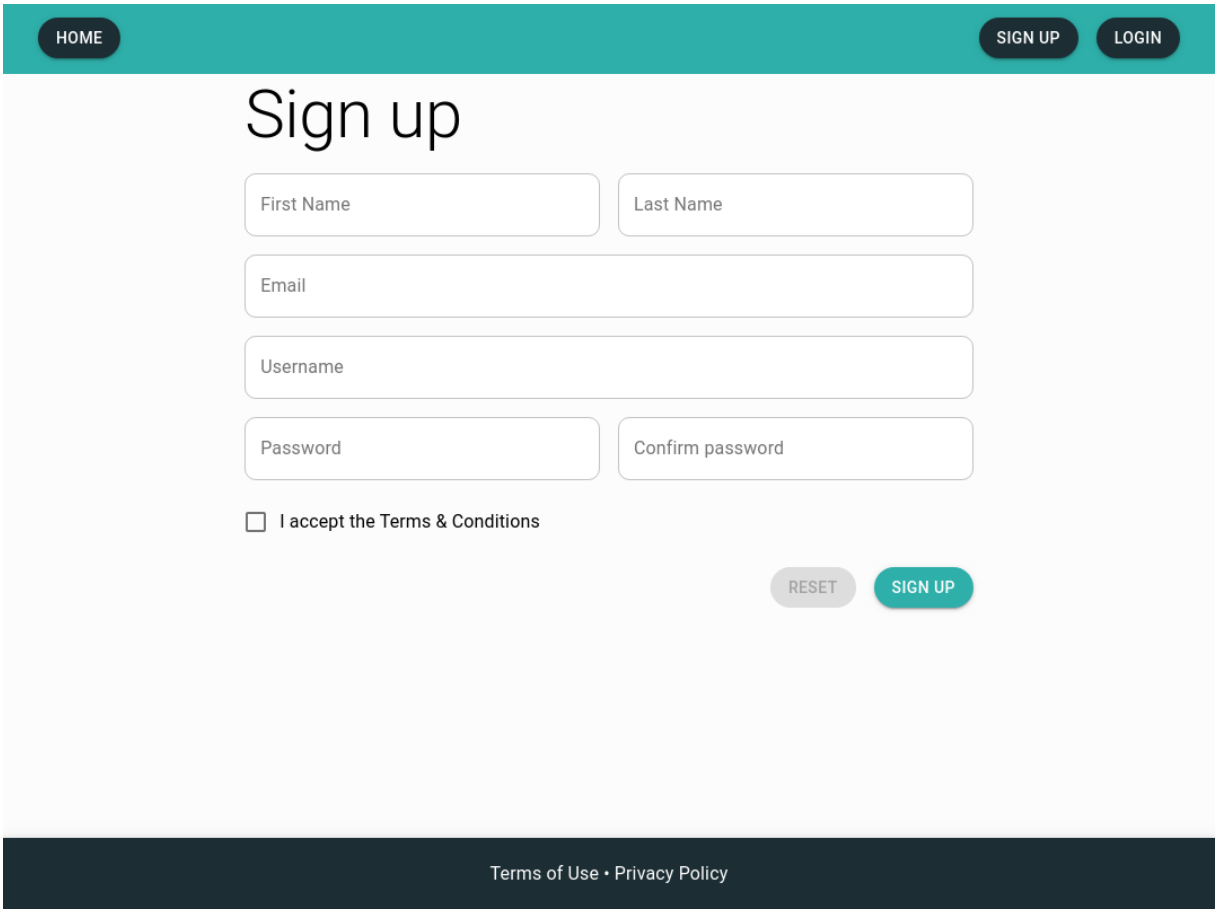

Figure 8.15: Sign up page

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 8.16.

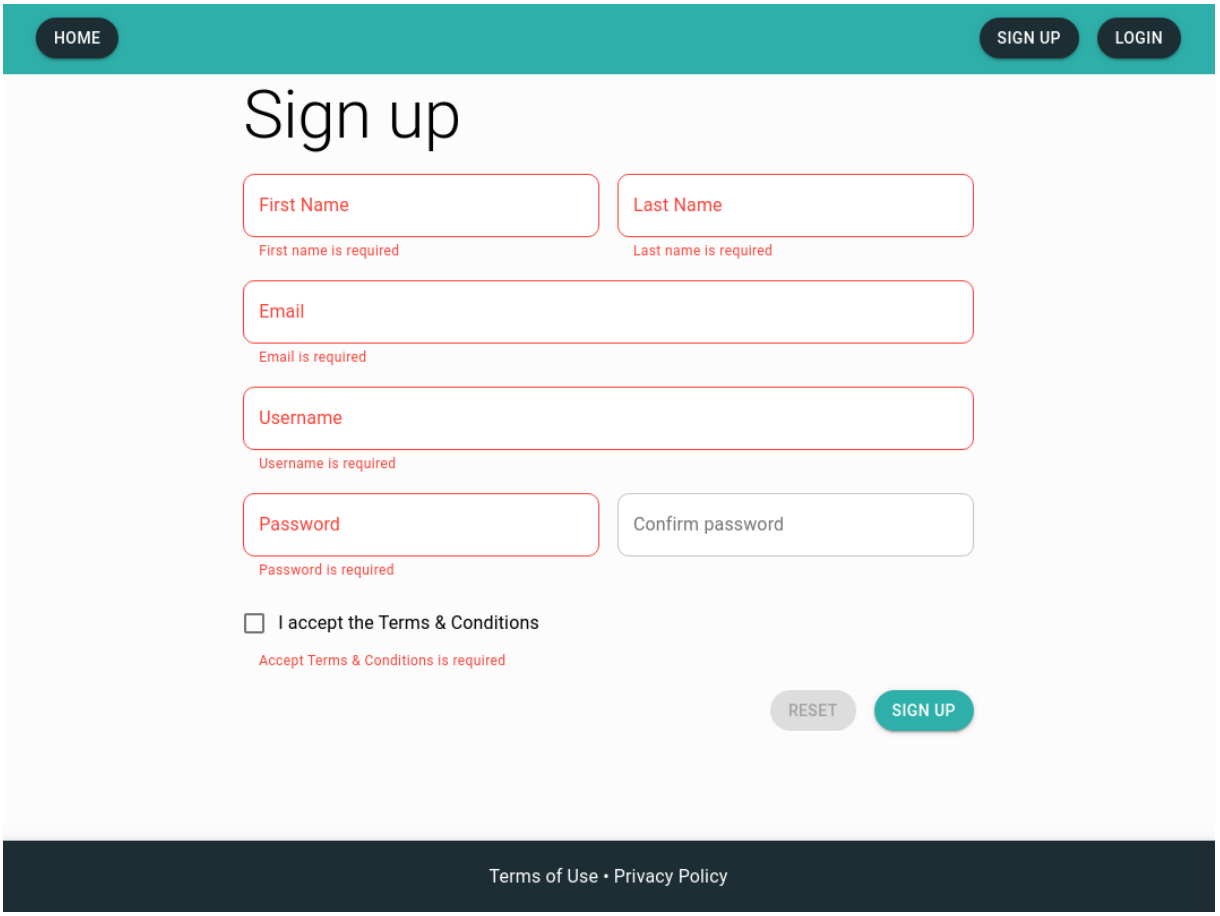

Figure 8.16: Sign up page validation errors

The fact that an error shows up when the passwords don't match, visible in Figure 8.17, 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.

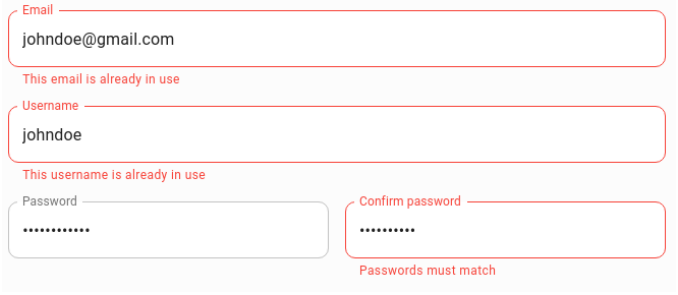

Figure 8.17: Sign up page validation errors (2)

The login follows the same principles, therefore achieving ***Consistency and Stan-***

dards, but not to be repetitive, only the login form is presented in Figure 8.18.

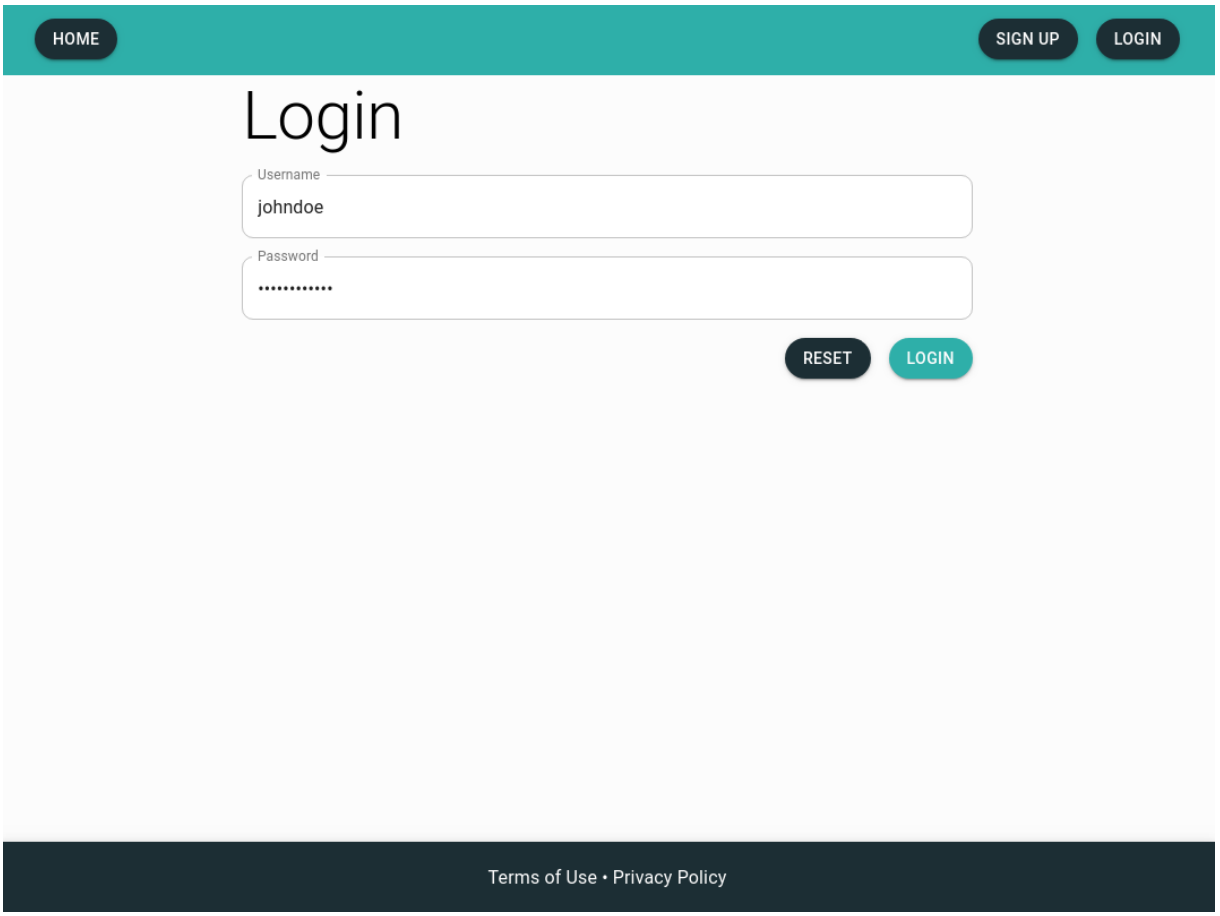

Figure 8.18: Login page

Dashboard

The first card visible in Figure 8.19 represents the active loan a user has at the moment. Since he can only have one active loan at a time, this card resumes the state of that loan.

Next to it is the investments card. Here the user is presented with a portion of his investments.

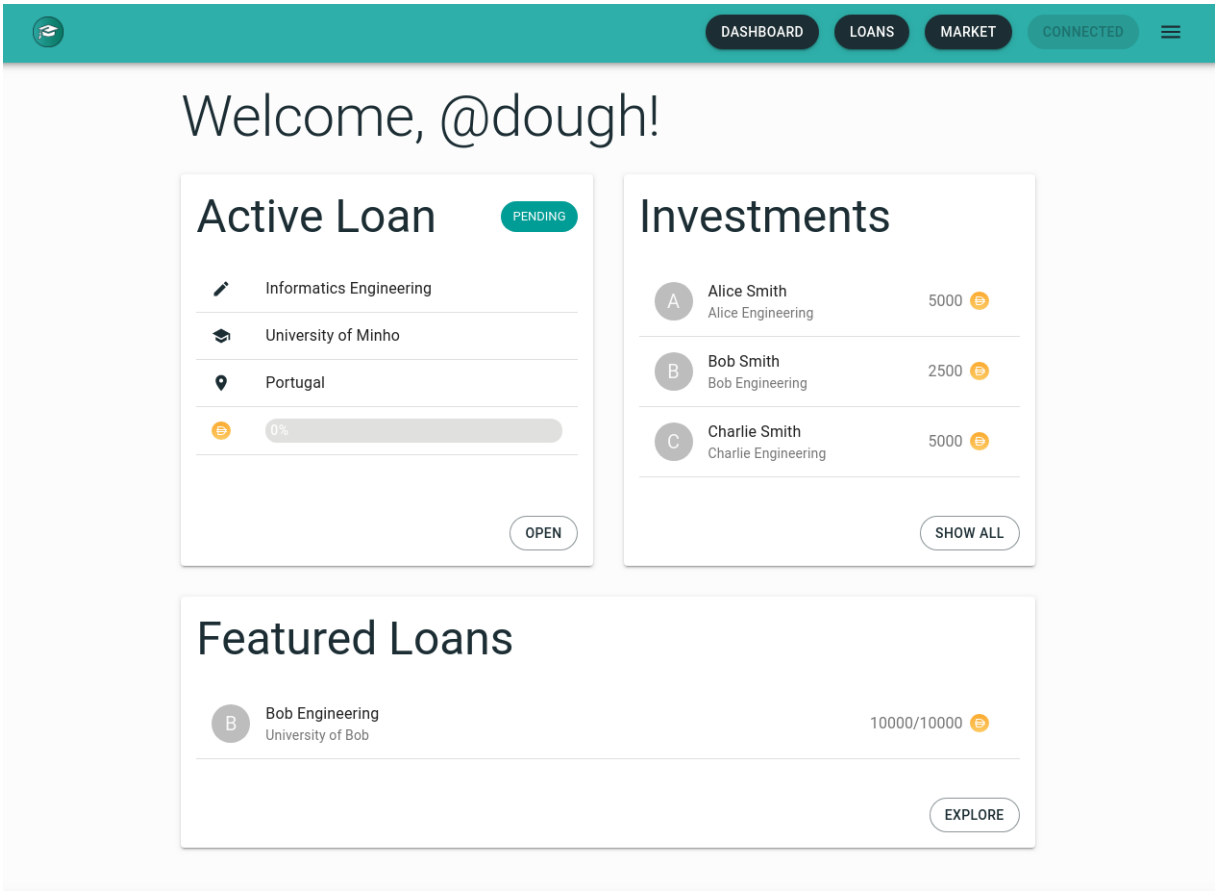

Figure 8.19: Dashboard page

Finally, the bottom card shows the user some active loans from other students that he can invest in. The objective of this dashboard is to group critical information to the user and display it in a simple manner.

Request a loan

A student to request a loan in our application, he has to fill the form illustrated in Figure 8.20. The amount that the user has to enter is in DAI.

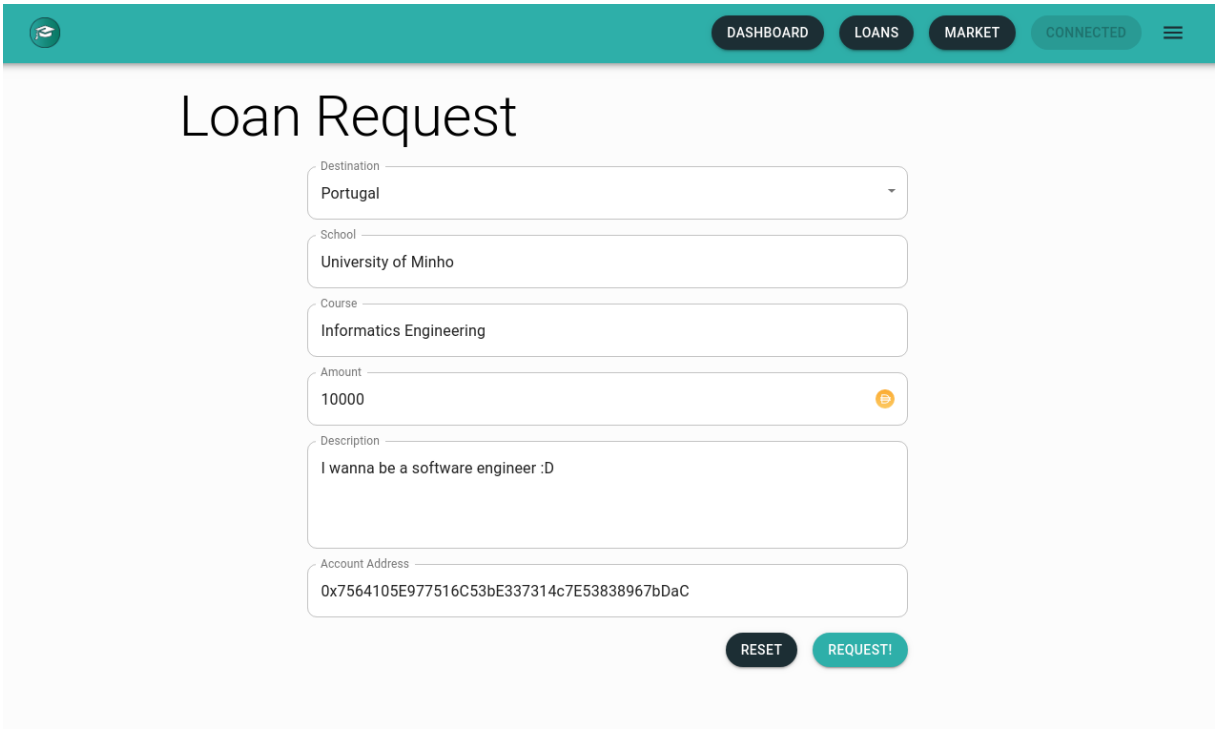

Figure 8.20: Loan request page

If the user has its wallet connected, the wallet address is filled out automatically, helping the user in this process. However, the student has requirements he has to meet to be able to complete his request: he needs to have his wallet connected (via MetaMask) and he has to have his KYC process fulfilled (this process will be discussed below). Otherwise, the following error messages are shown, and the button to make a request is disabled, guaranteeing not only the ***Predictability*** principle but also the ***Help users recognize, diagnose and recover from errors*** heuristic.

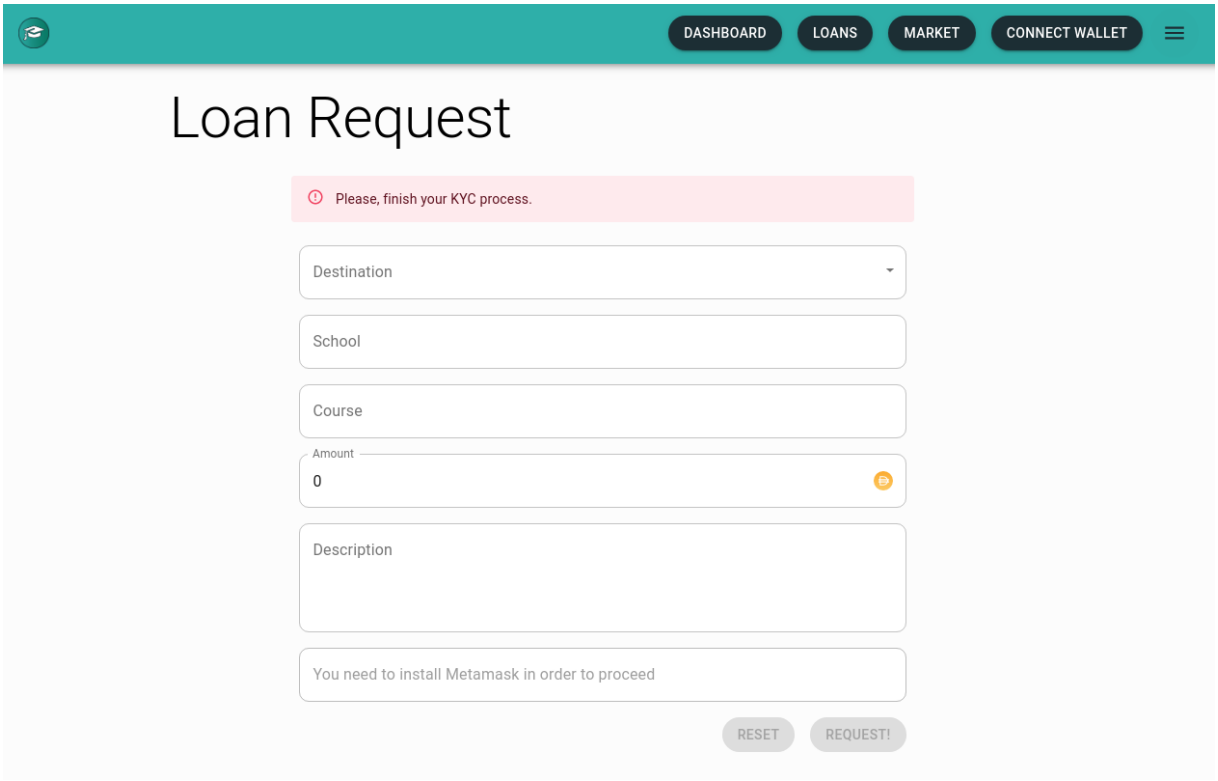

Figure 8.21: Loan request page validation errors

Manage Loan

After creating the request, the student is taken to the page where he can manage that request. Here he can submit public documents to enhance his loan, private documents so our administration team can manage the terms of the contract, or even cancel the request. Each operation depends on the current state of the loan. For instance, he can only submit documents after the funding phase is over, or can only cancel if it is not in the active phase. Figure 8.22 shows the behavior mentioned above.

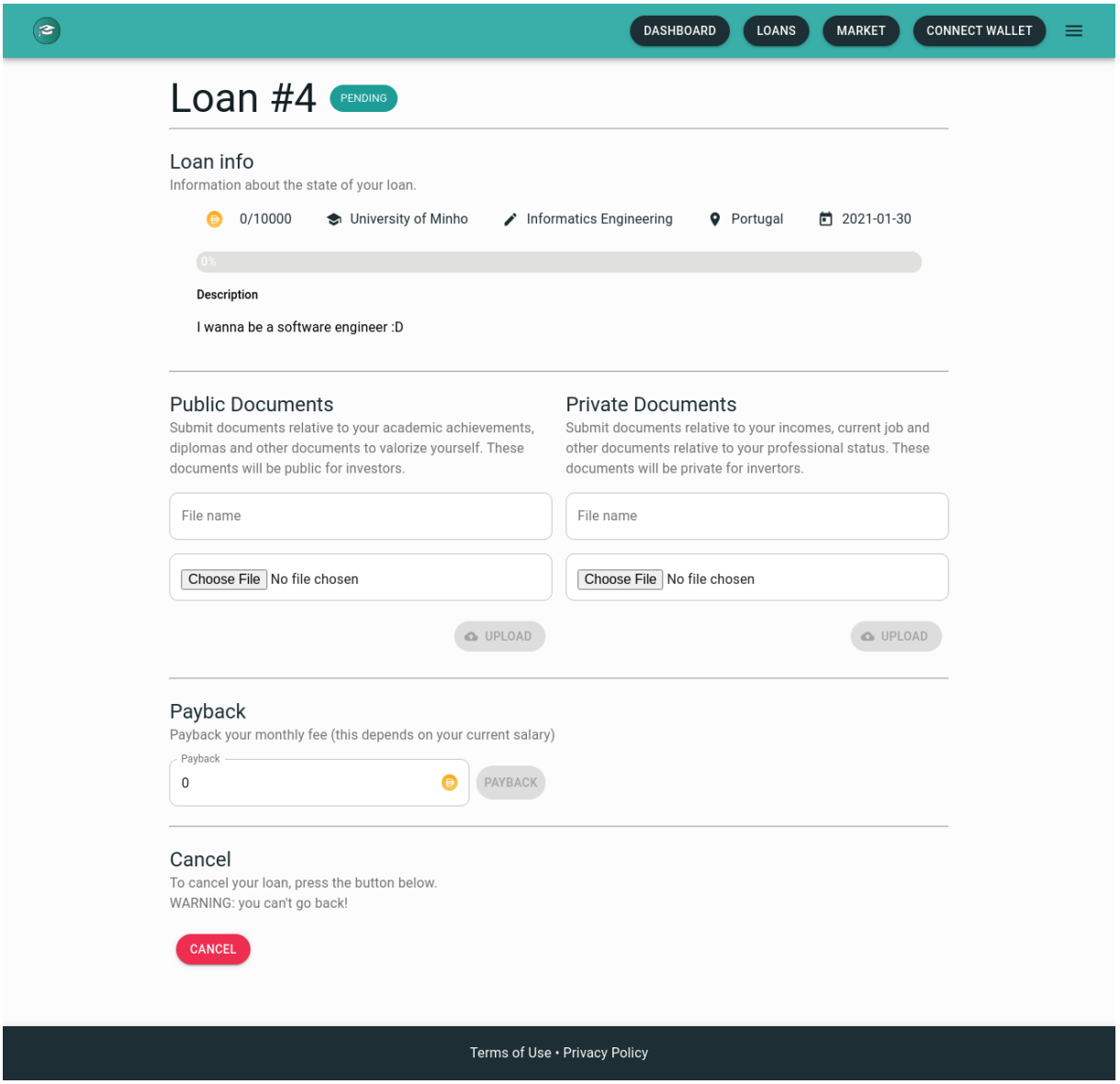

Figure 8.22: Loan management page

Furthermore, it is on this page that the user pays back the amount for the loan he request. After the amount is paid, an admin can change the loan state to *finalized* and the respective investors redeem their money.

Personal Loans

Going back a level, the student has a list of all of his loans, organized by each state possible, visible in Figure 8.23.

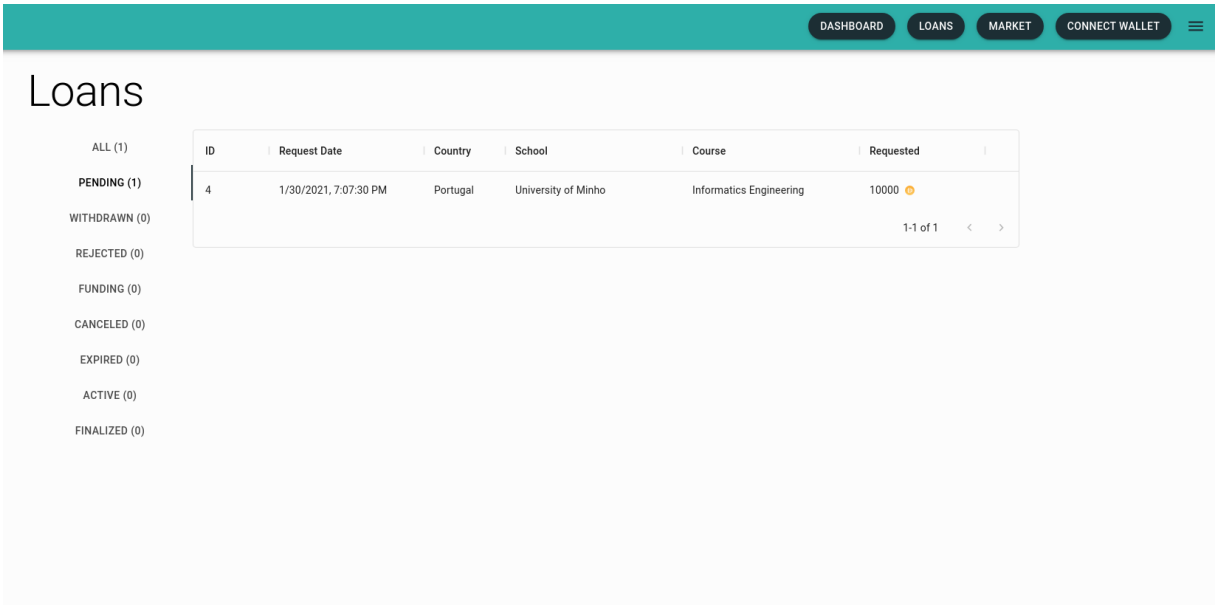

Figure 8.23: Personal loan page

This is one of the simpler pages of our application as it only intends to organize the information so the user can easily search for it.

Loans

Turning to the investor perspective, there is the page where all loans in the *funding* phase are listed. Figure 8.24 shows this page, where each student has a card with relevant information about his/her loan, for example, which course he/she wants to take and how much does it cost. There is also a filters section on this page. It allows for more experienced users to restrict what they want to fetch because they have an investment vision, or for people to search for results that they relate with. This mechanism is related to the ***Flexibility and efficiency of use*** heuristic.

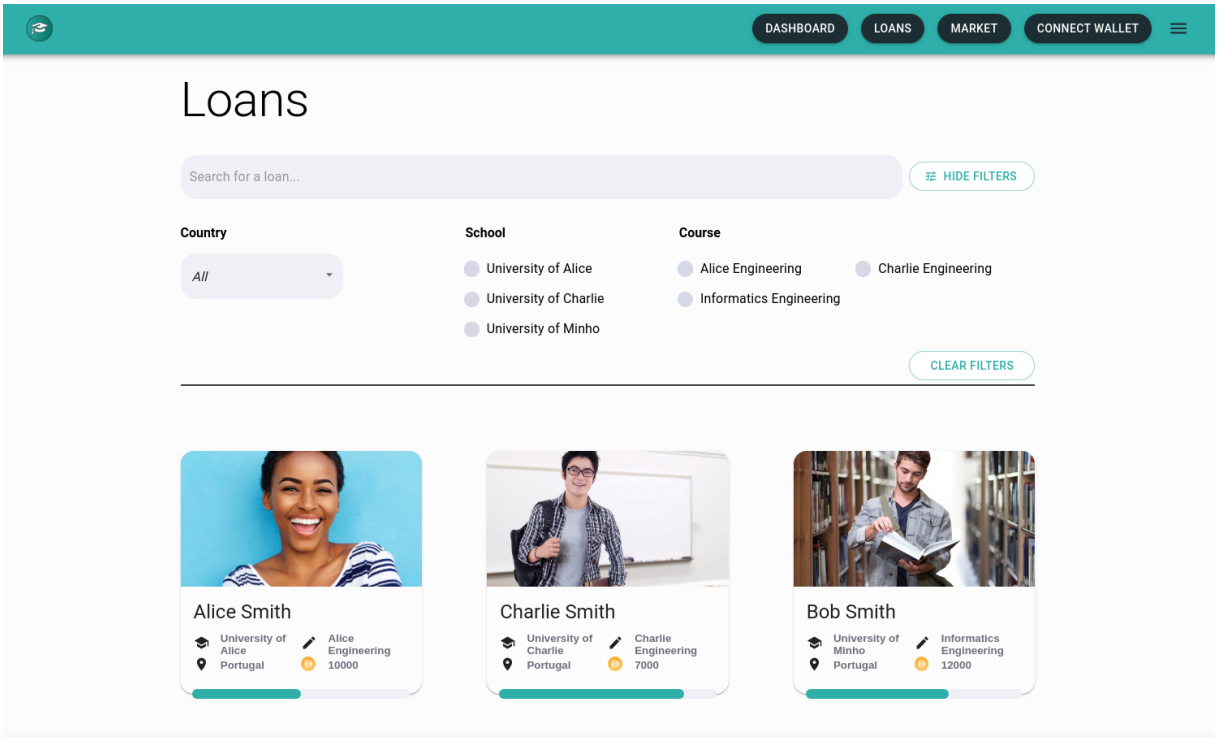

Figure 8.24: Funding loans

When entering this page, there is a need to fetch the funding loans. To represent that state, a spinner is used to indicate the user that information is being fetched instead of an empty page. This follows the ***Visibility of system status*** heuristic and it is visible in Figure 8.25.

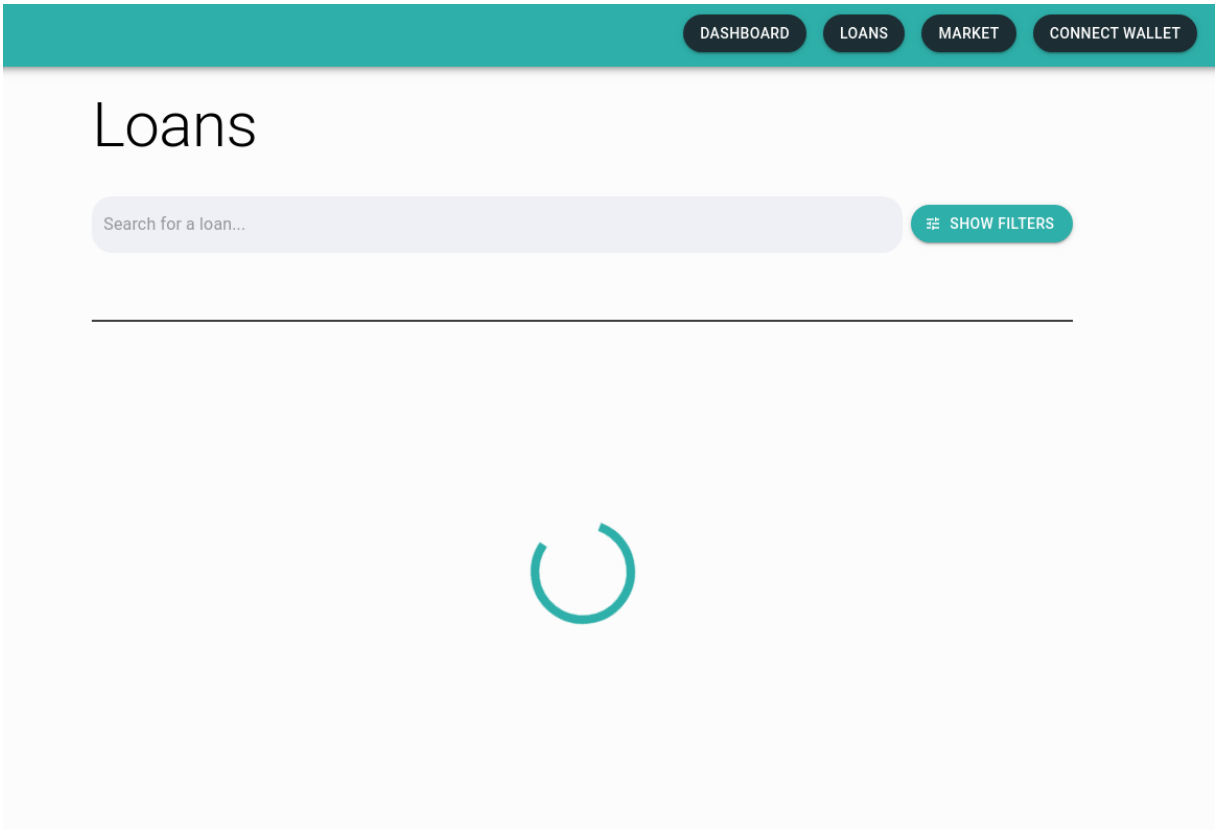

Figure 8.25: Fetching loans

Loan

Perhaps the most important page of our application, the loan page illustrates the different loan phases. Depending on the phase, this page shows different information and changes which actions a user can make within it. Figures 8.26, 8.27 and 8.30 display the some different phases a loan can be in.

Starting with the funding phase, an investor can buy tokens to fund the student. If he already has tokens, he can even withdraw those tokens if the loan is still in the funding phase.This can be seen in Figure 8.26. The success of this act is followed by the progress bar filling up (***Synthesizability*** principle).

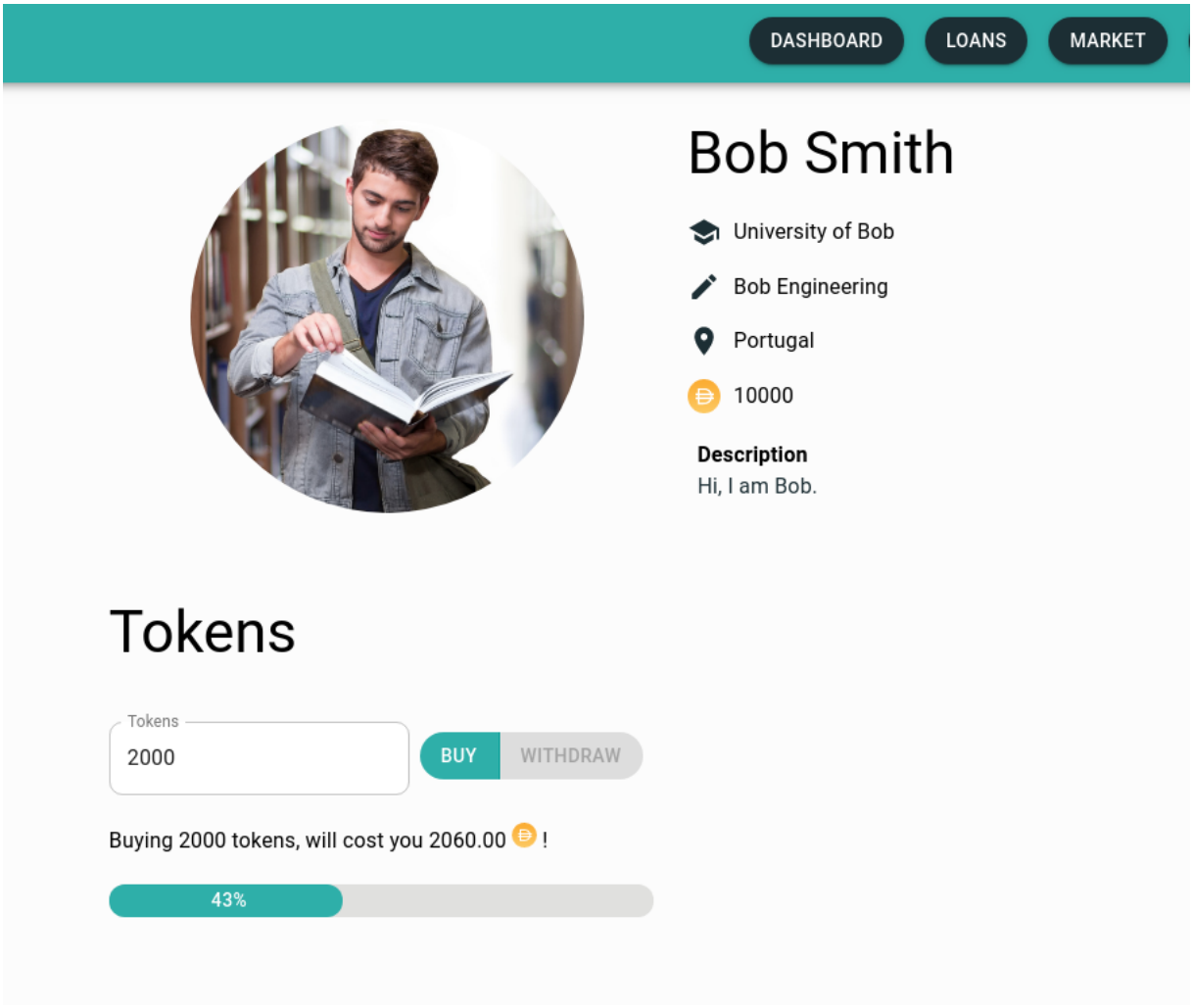

Figure 8.26: Loan - phase Funding

The acts of buying or withdrawn tokens are connected to the blockchain and therefore some actions on the wallet are needed. This will be shown in Figure 8.28.

Moving to the active phase, visible in Figure 8.27, the user, as an investor, can buy tokens if they are available on the market. By introducing the number of tokens that he intends to buy, the user is presented with pop-ups as in Figure8.28.

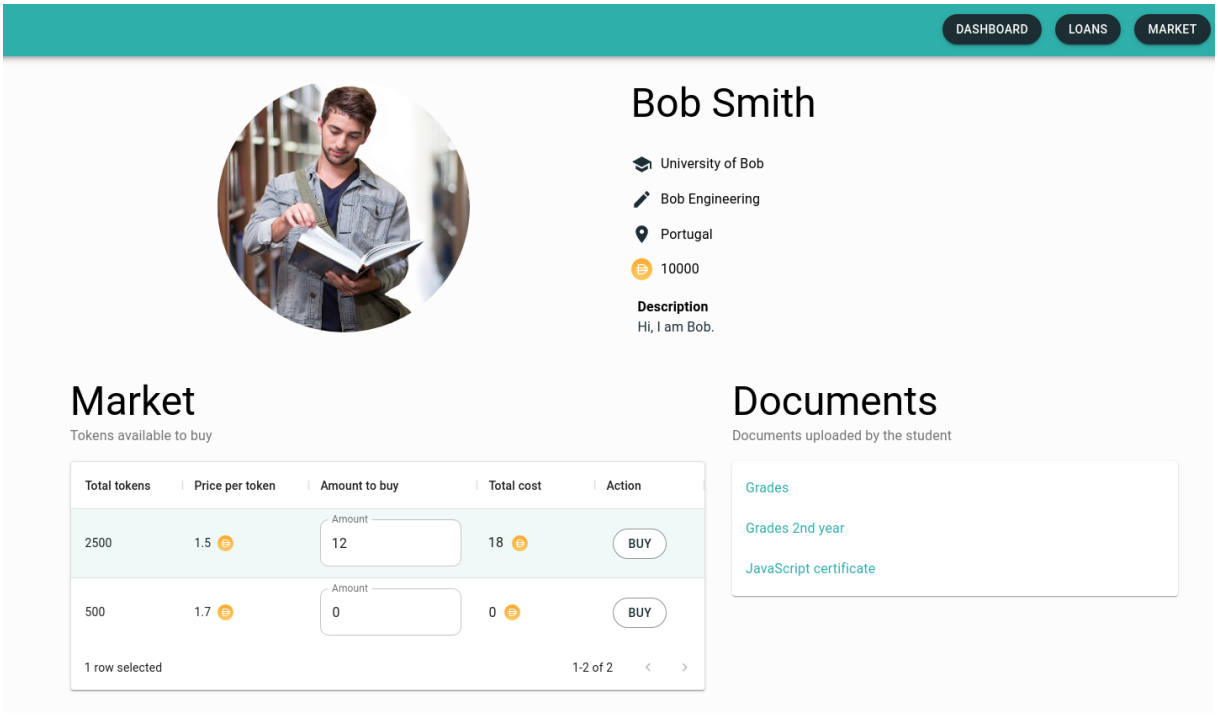

Figure 8.27: Loan - phase Active

In this page, the documents uploaded by the student are also displayed. their objective is to show the development of the student throughout the course.

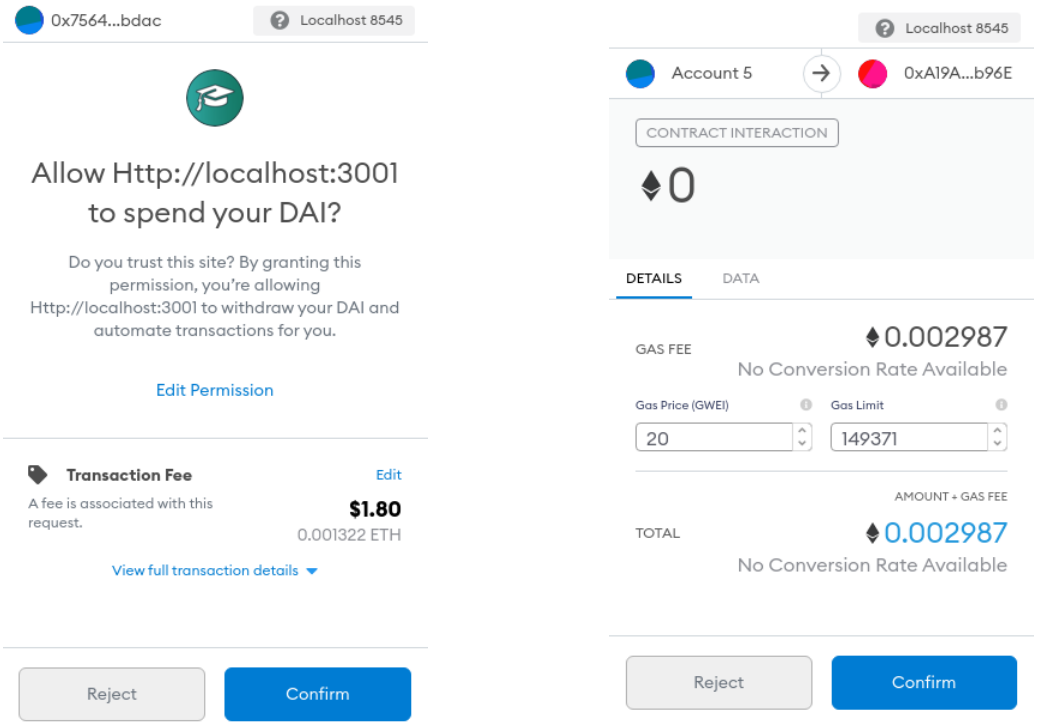

Figure 8.28: MetaMask pop-ups

This popups are from MetaMask and represent two transactions that are needed to execute an operation that involves some amount of tokens. The first one is to allow the contract to move a number of tokens, the second one is the transaction that moves that amount.

If this operation is successful, the toast presented in Figure 8.29 shows up. This mechanism reflects the ***Visibility of system status*** heuristic. This heuristic is present through all of the application.

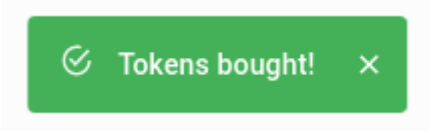

Figure 8.29: Tokens bought with success

Finally, in the cancelled phase, there are no actions available for the user, as seen in Figure 8.30. The other states have similar behaviour, so there is no need to describe them here. One last thing that is worth mentioning is that if the investor had investments in a student that either as seen his loan cancelled for some reason, or it is in the finalized phase, he can withdraw or redeem the invested tokens.

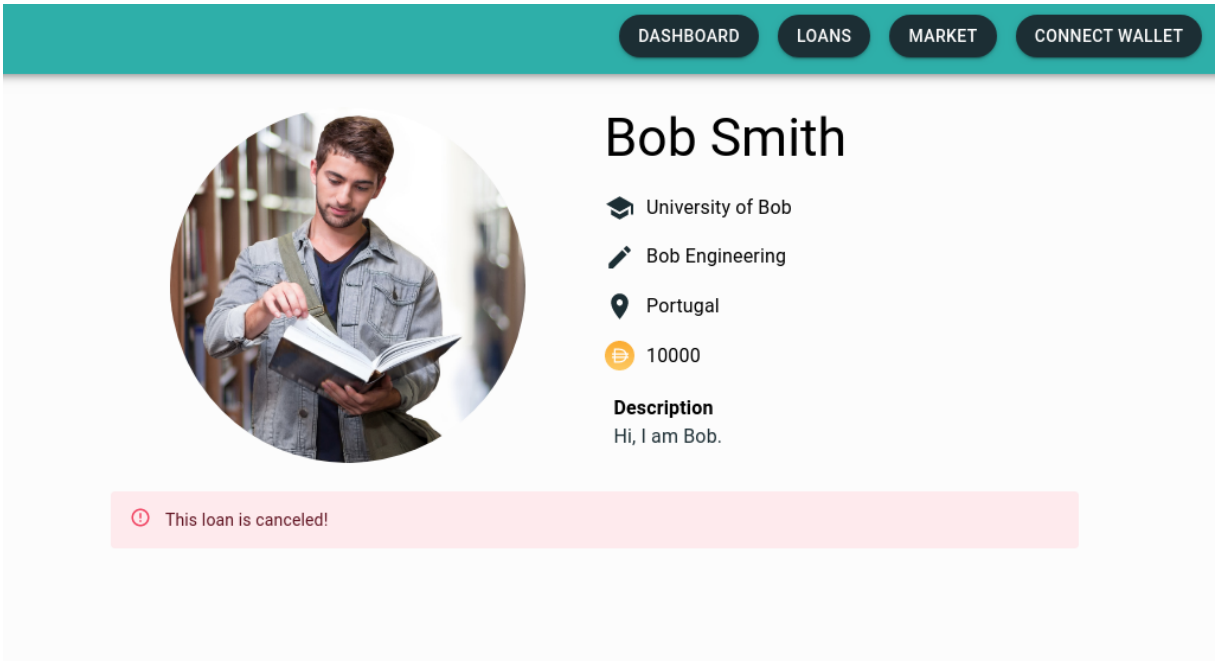

Figure 8.30: Loan - phase Canceled

Despite being optimized to be used on a desktop, the application is also responsive. The mobile version of the previous figure is shown in Figure 8.31.

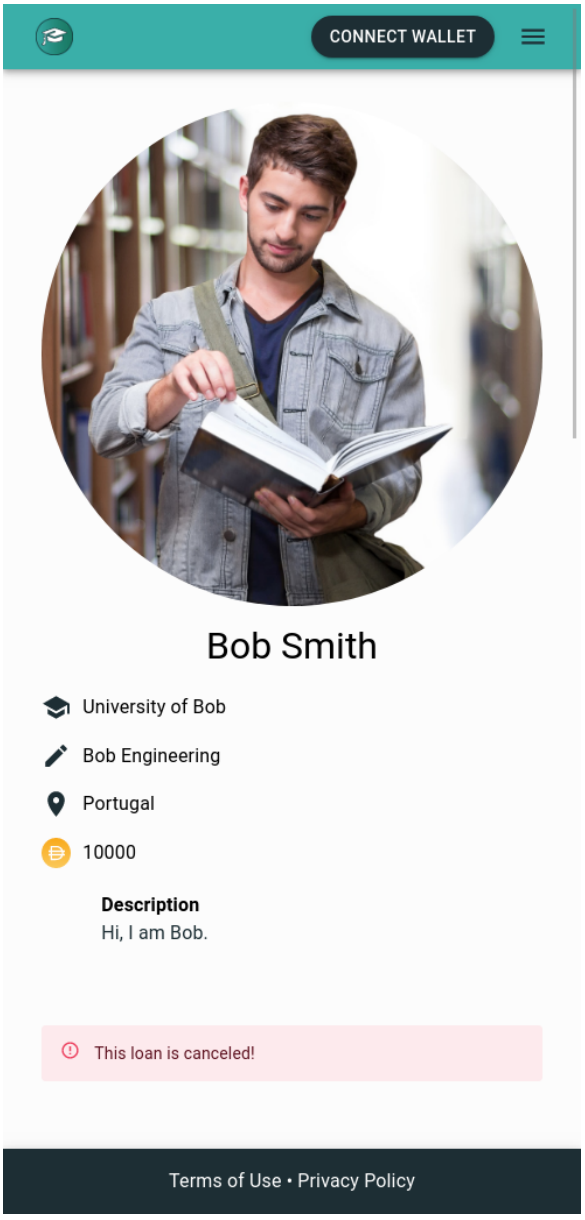

Figure 8.31: Loan - phase Canceled (Mobile)

One more thing to worth mentioning is that all the fonts are responsive.

Market

To check which loans are in an active phase, the investor can go to the market page (cg. Figure 8.32). To select a loan, the user simply has to click on the respective loan.

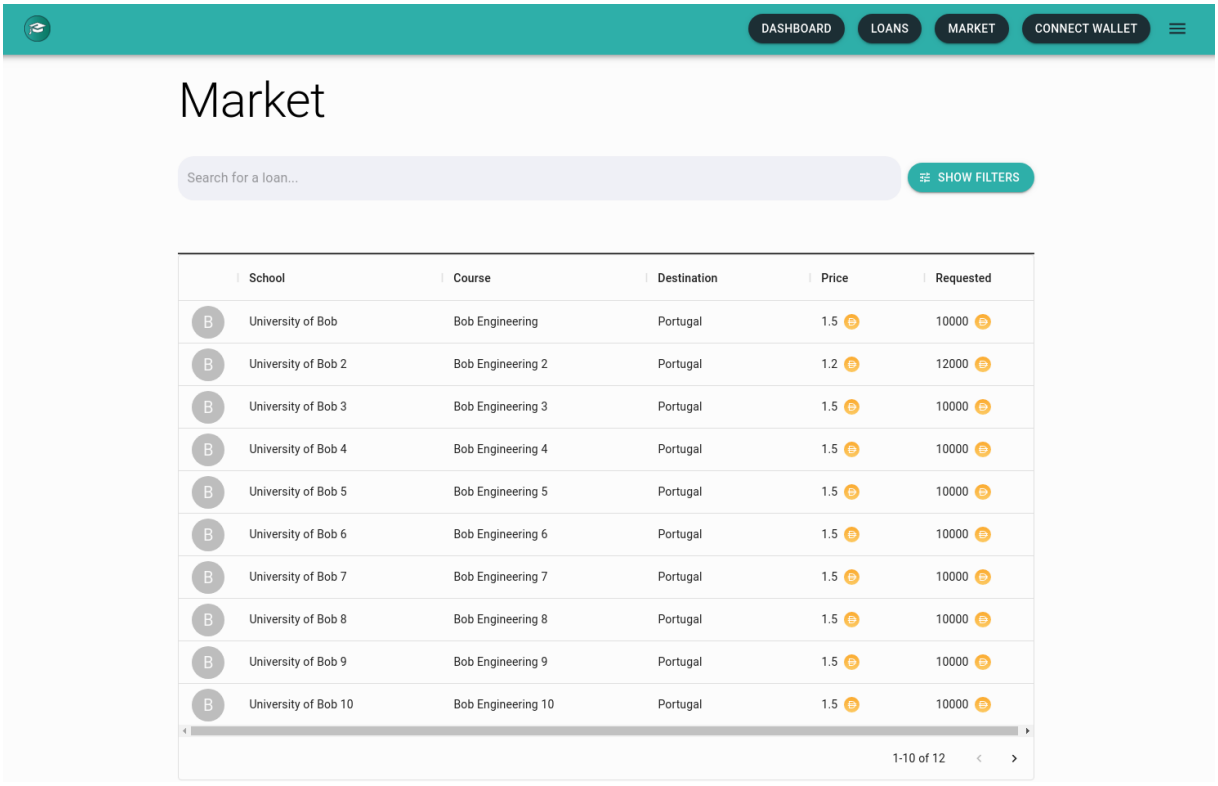

Figure 8.32: Market

Investments

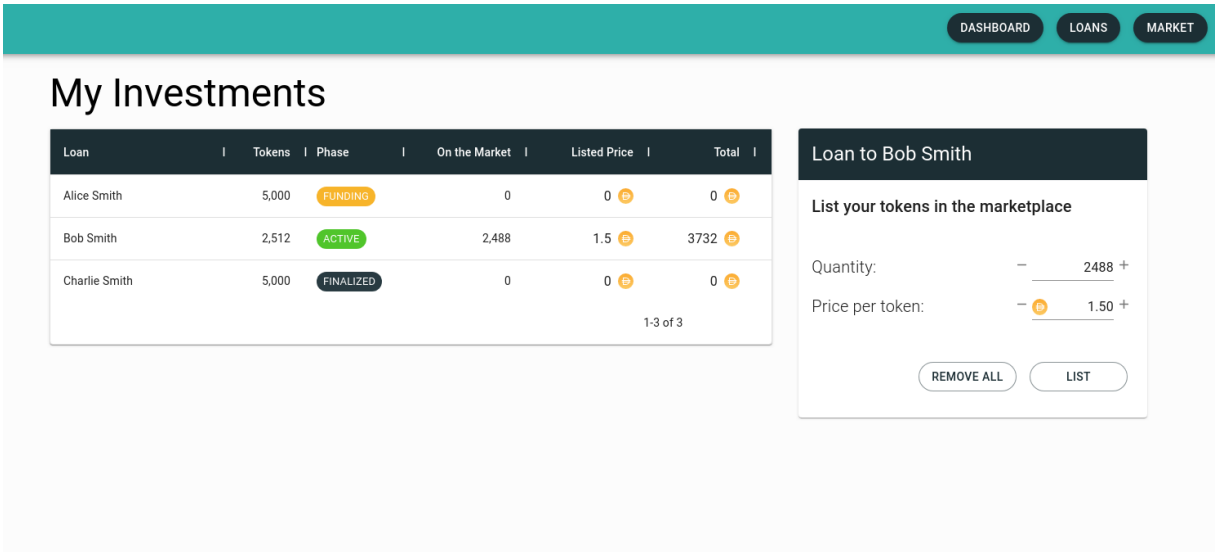

Figure 8.33: Investments

The figure 8.33 shows the investment page, accessible via the "Show All" button on the Dashboard. The objective of this page is to gather all the investments of a user so that they can see their assets and interact with them. For each investment in the table it is possible to see the number of tokens a user holds, the number of tokens listed in the secondary market, the price at which they are listed, the number of tokens listed, and the stage the loan is at.

When a user selects an investment from the table, a set of actions is available in the card on the right side of the view page. The content of the card and the actions available are different depending on the phase in which the investment is currently at. In the active phase, the user can cancel his funding and retrieve the money he or she had given to the student. When the loan is active is possible to list tokens on the secondary, and edit any existing sell positions. Once in the "Finalized" phase, the user can redeem their tokens and collect the payments that the student has made over the years.

Account

The last page, as a normal user of our application, is the account page. Here, the user can edit his basic information, as seen in Figure 8.34.

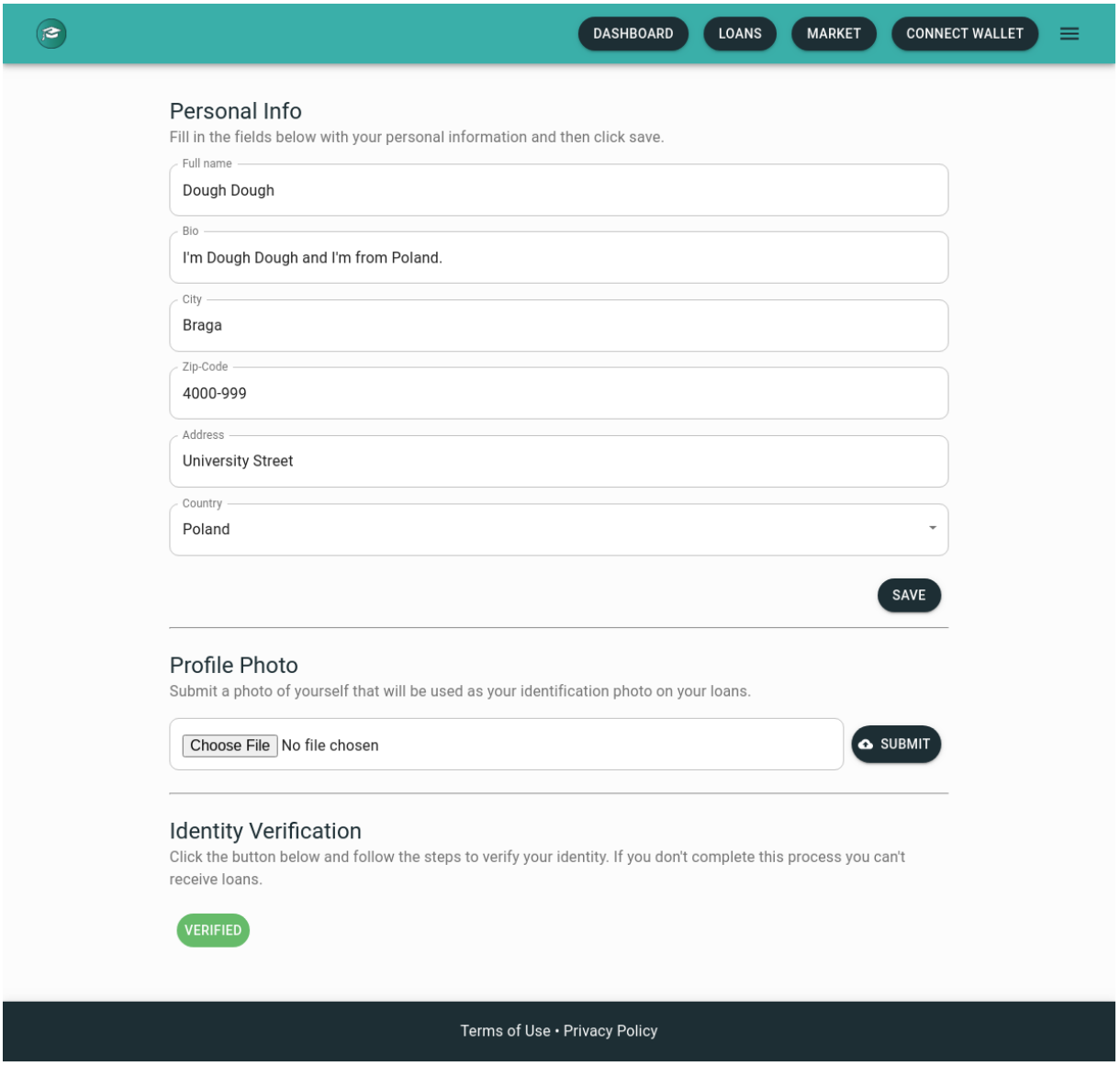

Figure 8.34: Manage Account

As mentioned before, the user needs to verify his ID (KYC). To perform this, we use an external software, Stripe. This process is mocked by Stripe itself, as the full version is paid. Figure 8.35.

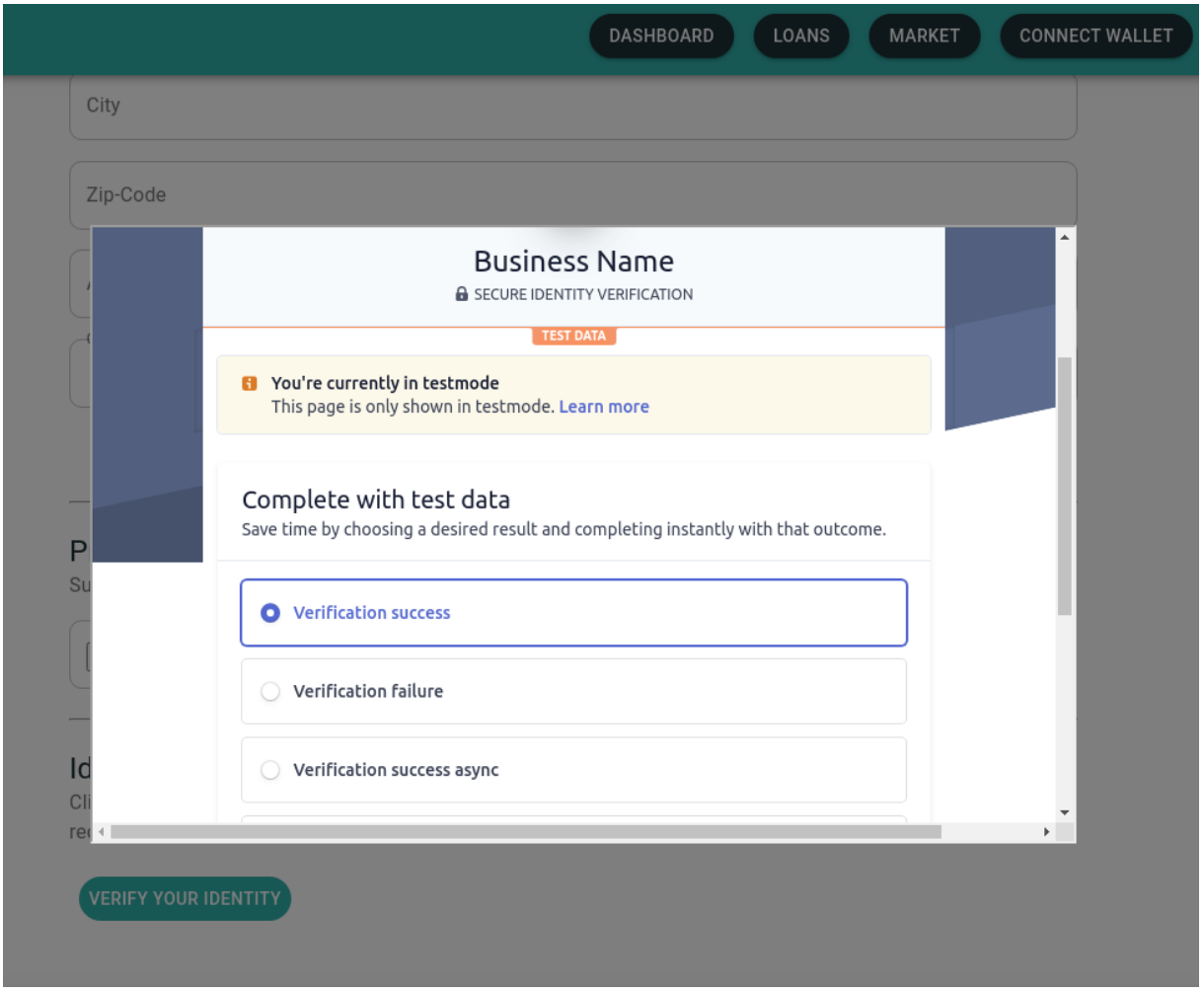

Figure 8.35: KYC (Stripe)

8.3.2 Administrator

The last type of user of our application is the administrator. This user has the responsibility to accept loan requests, validate documents for active loans and to determine a loan as finalized. To fulfil the ***Consistency*** principle, every page has the same structure: the title and the respective table with all entries that correspond to the title. Figures 8.36, 8.38 and 8.39, respectively, illustrates the different pages above mentioned.

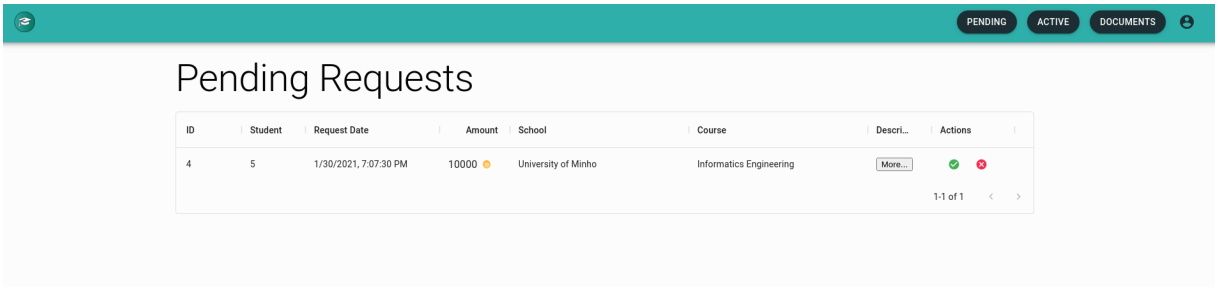

Figure 8.36: Pending loan requests (Admin)

The previous figure shows yet other two principles: ***Observability*** and ***Browsability***. This is due to the fact that the user can observe how many pending requests, in this case, are to accept or deny, Besides that, the table has pagination to help the user browse through the multiples loans. This notion of pagination and browsability is applied to all of the tables present in our application, either for admins, students or investors. These tables also allow the user to filter, order or hide columns.

If the admin thinks that the information displayed in the table is sufficient, it can simply accept or deny the loan request. If not, clicking in a specific request opens up a more detailed version of it. Here, the admin can even specify parameters of the loan such as *days to expiration*, *funding fee* and *payment fee*. Figure 8.37 illustrates the described behavior.

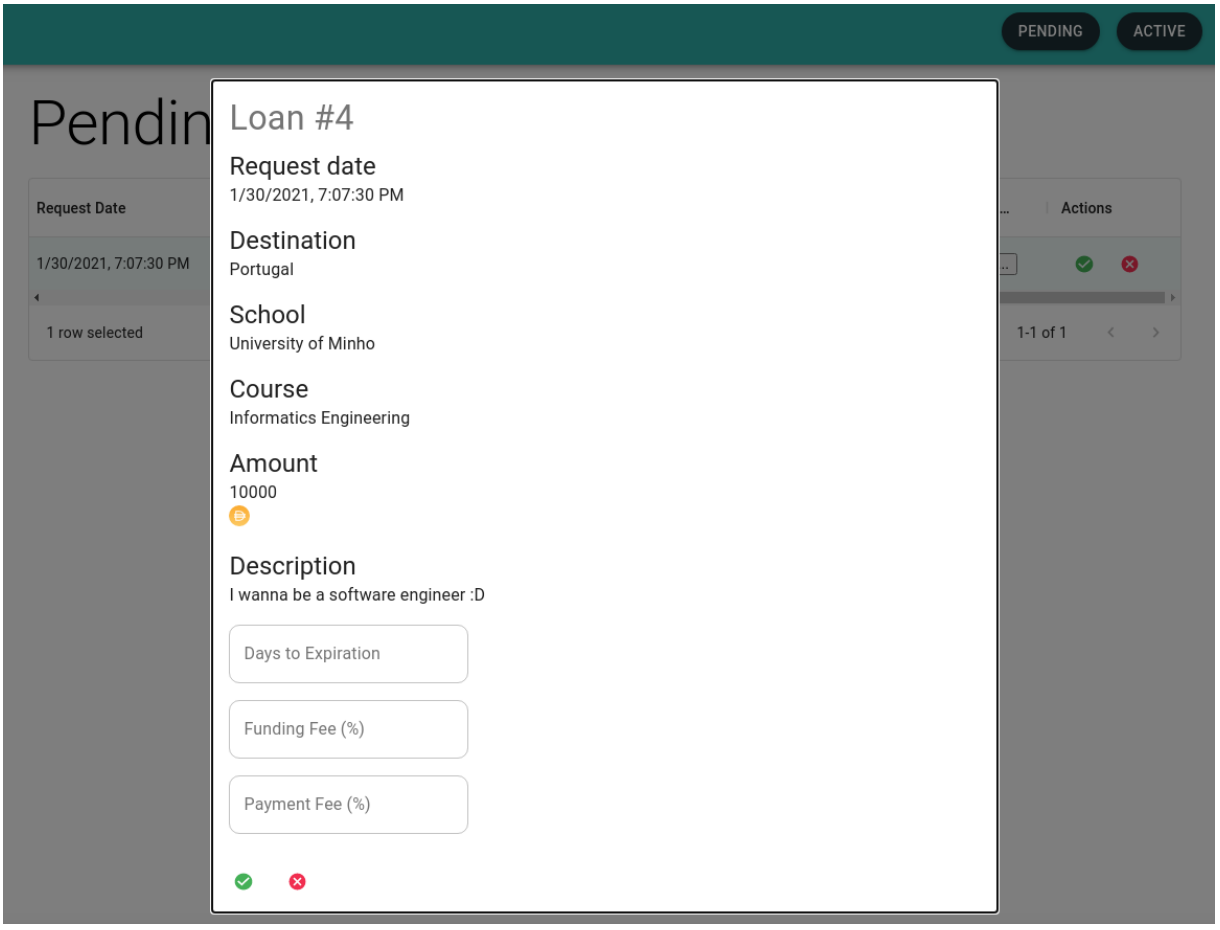

Figure 8.37: Pending loan requests dashboard (Admin)

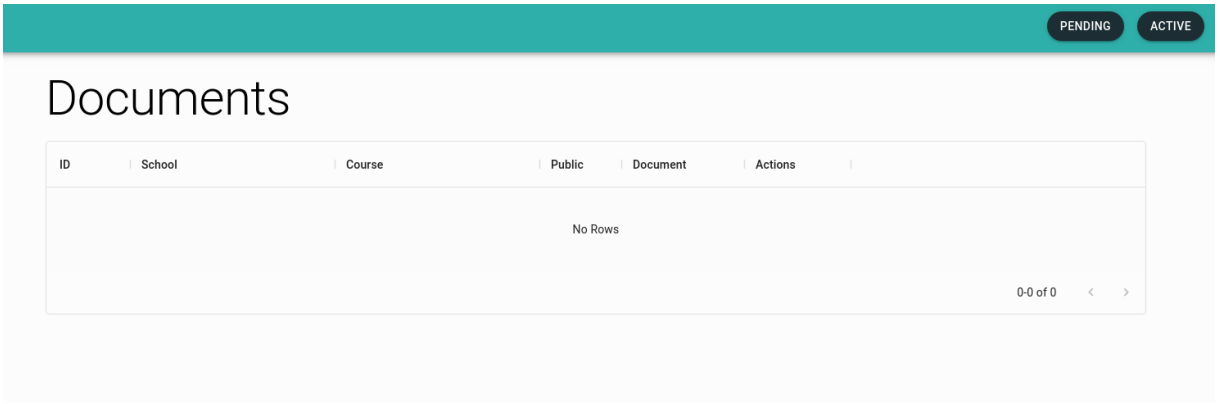

Figure 8.38: Documents dashboard (Admin)

Finally, Figure 8.39 shows how simply the admin can mark a loan as finalized, by clicking the check button. This operation has the prerequisite that the user of that loan

has paid the investment made in it.

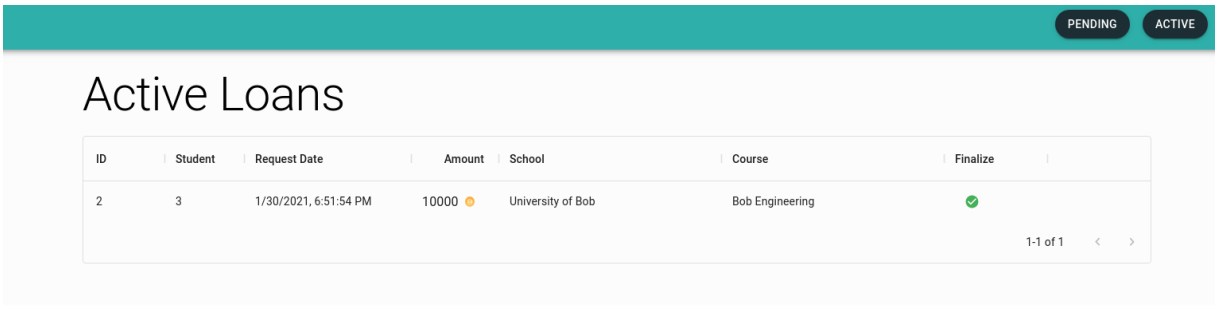

Figure 8.39: Active loans dashboard (Admin)

9 Deployment and Maintenance

A production deployment of the TuiChain application consists of the following major 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 MySQL or PostgreSQL;
- A storage service to persist documents and other assets uploaded by users;
- One or more Ethereum clients.

Several choices can be made on to how 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.

Figure 9.1 illustrates a possible production deployment featuring these components. Dependencies on the Stripe service and the use of Metamask as a wallet by the client browser are also depicted. Note that the shared-nothing architecture followed by the backend component makes the application easily scalable.

Two kinds of 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 production setting.

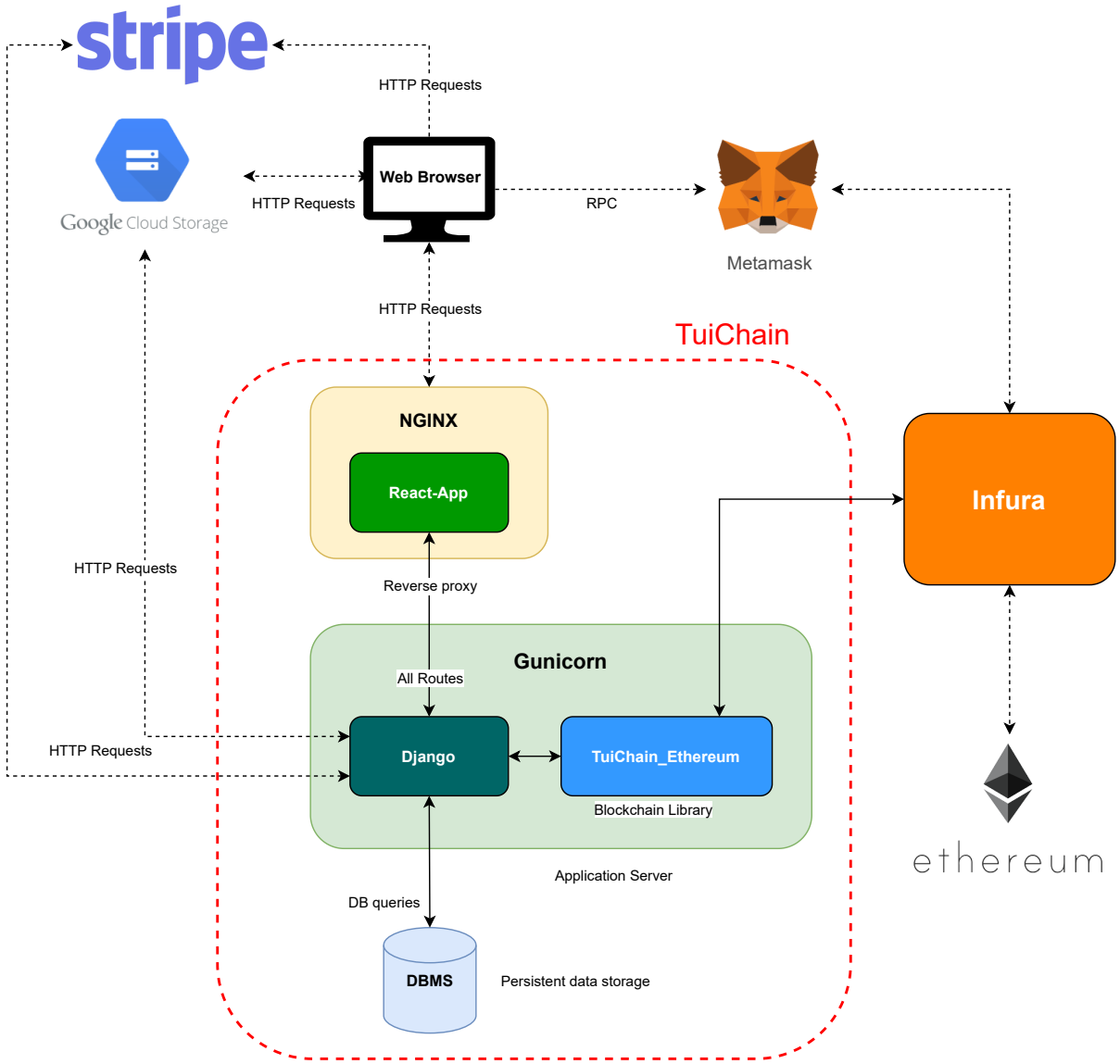

Figure 9.1: Example production deployment.

10 Conclusion

As we said in the beginning, our intent with this project was to propose a platform based on blockchain technology to bring together students who need financing and lenders, and we aim to reduce the number of people that can't attend higher education due to financial reasons. This jibes with the fact that having a masters, MBA, PhD is increasingly important in today's labour market. Besides, cryptocurrency and blockchain are growing at a very rapid pace, which indicates that these solutions may be the answer for the future ahead when it comes to education financing or any other platform with payments involved.

Having said this, we feel that we have made an important first step to make that objective a reality. We, as a team, feel like we have gained knowledge on many topics we didn't know at the present date like blockchain, cryptocurrencies or ethereum network. For some, it was the first experience working in a big project where many people have to articulate between themselves to put out a viable product and others, even though it was not their first time, gained enriching experience with tools that can be useful later on their professional life.

10.1 Future work

One of the main things we couldn't implement and that we considered adding was specialized search features whether it's searching given the student's past education, loan's value, attending university, etc. We also had the idea of implementing an algorithm that calculated the risk of investing in a certain student that could be a part of a premium plan that could be implemented or not, depending on the purpose of the application.

Even though this application is only implemented as a web app, we think it would be beneficial to have a native app since it's widely known that people nowadays also own and use smartphones so this implementation could potentially bring more investors, students and allow users to check their accounts more easily.

Finally, a major topic not related to software development is legal issues. Since all this process involves contractual and financial aspects there is a necessity to approach legal issues more seriously and professionally. Having said this, we see realize the importance of working with specialists in law and economy to bring legitimacy and confidence in our work as well as the application. This would also enable process automation since it enables some tasks done by the admin to be automated which include loan approvals, return rate calculation, among others.

References

[1] Home – Fundação José Neves, 2020. Retrieved Jan. 13, 2020 from https://joseneves.org/

[2] Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. *Human–Computer Interaction*. Pearson Education, 3rd edition, 2003.

[3] HSBC Holdings. The Value of Education: The price of success. Technical report, 2018. Retrieved Jan. 13, 2020 from https://www.us.hsbc.com/content/dam/hsbc/us/en_us/value-of-education/HSBC_VOE5_USA_FactSheet_508r2.pdf

[4] Jakob Nielsen. 10 Usability Heuristics for User Interface Design, 2020. Retrieved Jan. 13, 2020 from https://www.nngroup.com/articles/ten-usability-heuristics

[5] Andrew Shepherd. *Hierarchical Task Analysis*. CRC Press, 1st edition, 2000.

A API Routing

A.1 Authentication

api/auth

- /login/
- /signup/
- /verify_email/
- /verify_username/
- /reset_password/<int:id>/<str:token>/
- /email_reset_password/
- /is_admin/

A.2 User

api/users

- /get/<int:id>/
- /get/
- /get_all/
- /update_profile/

A.3 Blockchain

api/tuichain

- /get_info/

A.4 External

api/external

- /create_verification_intent/
- /get_verification_intent/<slug:verification_intent_id>

A.5 Loans

api/loans

- /new/
- /validate/<int:id>/
- /finalize/<int:id>/
- /reject/<int:id>/
- /get_personal/
- /get_all/
- /get_operating/
- /get_state/<str:state>/<int:user_info>/
- /get/<int:id>/
- /cancel/<int:id>/
- /user_withdraw/<int:id>/

A.6 Loan Transactions

api/loans/transactions

- /provide_funds/
- /withdraw_funds/
- /make_payment/
- /redeem_tokens/

A.7 Loan Documents

api/loans/documents

- /get_unevaluated_docs/<int:id>/
- /get_approved_public_docs/<int:id>/
- /get_personal_docs/<int:id>/
- /get_all_unevaluated/

- /approve_document/<int:id>/
- /reject_document/<int:id>/
- /upload_document/<int:id>/

A.8 Investments

api/investments

- /get_personal/<str:user_addr>/
- /get/<int:id>/<str:user_addr>/
- /get/<int:id>/

A.9 Market

api/market/transactions

- /create_sell_position/
- /remove_sell_position/
- /increase_sell_position_amount/
- /decrease_sell_position_amount/
- /update_sell_position_price/
- /purchase/

A.10 Documentation

api/docs/