Decentralized mortgage market

Open market for real-estate crowdsourcing

K.C. Asmoredjo
A. Hovanesyan
S.M. To
C. M. Wong Loi Sing

TU Delft

Decentralized mortgage market

Open market for real-estate crowdsourcing

by

K.C. Asmoredjo
A. Hovanesyan
S.M. To
C. M. Wong Loi Sing

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Tuesday January 31, 2017.

Project duration: November 14, 2016 – January 31, 2017

TUDelft

Abstract

When taking out a mortgage from a Dutch financial institution, the financial institution will usually provide a fraction of the property value. When taking out a mortgage, it is expected that the person that is in need of a mortgage to provide a down payment for the remaining value. This means that people that need a mortgage will have to save up before they can buy a house. As saving up could be an obstacle if the required down payment is too high, crowdfunding the remaining value could be a solution.

Such crowdfunding platforms already exist. In fact, they are increasing in popularity. There are a variety of crowdfunding platforms for different needs, such as for entrepreneurs seeking investments for their projects. However, there is one thing all these platforms have in common. There is always a central point of trust, resulting in unneeded costs for the users of the platform.

This thesis presents the development process of researching and implementing a decentral market where mortgages can be crowdfunded. We have created a system that does not have a central point of trust, while it still connects people that need funding with investors and financial institutions.

An earlier bachelor project has already implemented a decentral marketplace to trade digital currencies. At first, the idea was to reuse their implementation. However, after researching we came to the conclusion that their implementation was no use to our mortgage market. As their marketplace is specifically designed for trading digital currencies, we developed our own market. To ensure some level of trust into our market, we used a blockchain to save the agreements that are made between the users. The blockchain ensures that the agreements made cannot be changed. Therefore, when an agreement is added to the blockchain, the agreement is final.

Our final product is a working prototype of a decentral mortgage market, which enables people that need funding to place mortgage and loan requests, receive mortgage offers and investment offers from financial institutions and investors. Financial institutions are able to receive mortgage requests and provide mortgages. Investors are able to search for investment opportunities, view campaigns that need funding and invest in these campaigns. The application can be run headless or using a graphical user interface, making it accessible for different types of users. It exposes a public API so third-parties can integrate it into their application. This is all possible without the risk of the mortgage market platform being shut down.

Preface

This thesis describes the development process of researching and designing a decentral market for real-estate finance. This project has been initiated as part of the course TI3806 Bachelor Project in the Computer Science program at the Delft University of Technology. The project took place from November 14, 2016 until January 31, 2017.

There are several people we would like to thank for their support during the project. First we would like to thank our coach, dr. ir. Johan Pouwelse and our client Stephan Hagens from ABN AMRO, for his valuable guidance and feedback. We would also like to thank Bart Gout from ABN AMRO and Leonard Franken from AFM for supporting us with their knowledge about mortgages. Finally, we would like to thank Martijn de Vos and the people from the Blockchain Lab for answering questions and helping us overcome challenges we had during the development phase of the project.

K.C. Asmoredjo
A. Hovanesyan
S.M. To
C. M. Wong Loi Sing
Delft, January 2017

Contents

1 Introduction

The mortgage market is a multibillion market that has seen some serious lows in the last couple of years. The high amount of borrowers defaulting their mortgage played a big role in the 2007-2008 financial crisis [45].

Taking a mortgage from Dutch financial institutions can be done as follows. First, the borrower needs to decide what the possibilities are (e.g. what are the total costs and how big can the loan be?). When the financial situation of the borrower has been assessed, a house can be chosen that fits the borrower's needs. When an agreement has been met with the seller of the house, a mortgage can be settled with a mortgage lender. Together with the lender, the borrower will be able to choose a mortgage type that fits their financial situation.

In the Netherlands, the maximum mortgage limit in 2016 was 102%. Each year it is being lowered with 1% until 2018, when the maximum mortgage limit will be 100% [22]. The CEO of the Dutch National Bank [14], Klaas Knot, wishes to see the maximum mortgage limit be reduced to 90% by 2028 [43]. While the maximum amount of the mortgage is a legal limit, financial institutions will usually only provide a fraction of the property value amount [27]. How much the financial institutions will exactly provide depends on the Loan-to-Value ratio, which differs from case to case. Instead, it is expected that the borrower provides a down payment of the remaining value. The large down payment means that people that are looking to buy a house will have to save up before they can buy a house.

To enable borrowers to get the capital they need but do not have, crowdfunding is a solution. Such crowdfunding platforms already exist and they are becoming more popular. Some give the opportunity to crowdfund mortgages, while others are more general and provide a place for entrepreneurs to find funding for their projects. These projects range from opening a new shop to projects like installation of personal solar panels.

What these platforms all have in common is that they are centralized, which means that there is a third party that connects the investors with the borrowers for a fee. Thus the crowdfunding services that these platforms provide are not free. A way to eliminate this unneeded cost is to make a decentralized platform, where no middleman is needed.

2
Problem description

In this chapter we define the problem, give a description of the client, and analyze related initiatives. Furthermore it contains the project requirements, the description of the initial screens we have designed based on the input of our client, and we describe the development process of the project.

2.1. Problem definition

It is currently difficult for an average citizen to get a mortgage [56]. Only borrowers with minimal risk of defaulting are given a mortgage. As the value of houses keep rising, it is becoming increasingly difficult for people to receive a mortgage. Additionally, because of the mortgages not covering the total cost of the estate, the only possibility is to make a large down payment. For a lot of people, and especially for young adults whom recently joined the working force, down payment is not an option. At the same time, interest on saving accounts is so low that it does not cover inflation. Investing could bring a solution to savings losing value over time.

There is need for a system that connects different parties together. These parties consist of the borrower, the investor and the financial institution. The three stakeholders have their own incentive to work with each other. The borrower needs a financial institution to settle a mortgage, and investors to fund the initial down payment. In return, the investors and the financial institution can receive interest on their respective investments. To solve the mentioned problems, we focus on creating a platform where mortgages can be partially crowdfunded.

Crowdfunding platforms are getting increasingly popular to create funds for projects in which financial institutions are not investing. Platforms on which it is possible to get your mortgage crowdfunded already exist, yet on all of these platforms there is a central party that connects the investors with the borrowers. The central party is not only unneeded, but also brings unnecessary costs and risks. Bringing down the middleman will bring down the entire system.

To avoid that the entire system collapses when one of the middlemen drops out, the system needs to be decentralized. During the project we will strive to create a platform where people can apply for mortgages that are funded by both financial institutions and investors. A person will be able to apply for a mortgage where the financial institution can fund a fraction of the house cost (e.g. 70% of the total cost). When the mortgage has been accepted, it will be transferred to an open market where investors can fund the remaining part of the the house cost. The interest rate on these investments will be negotiated between the borrower and the investor.

When we started the project, the problem description differed. The original project description can be found in appendix A. The aim of this project is to create a decentralized market where it is possible to crowdfund a mortgage. The system will not be controlled by a single party, but mostly by supply and demand.

2.2. Client description

The client for this project is ABN AMRO [1]. ABN AMRO is collaborating with the TU Delft Blockchain Lab [3], which is part of the Tribler team [51], on the development of blockchain [4] applications. They aim to create some complex blockchain applications that are suitable for use in practice, within half a year [28]. With the collaboration, ABN AMRO wants to improve their knowledge on blockchain technology, especially on the practical application of distributed digital ledgers.

TU Delft has been researching blockchain technology since 2007. The research at TU Delft focuses on developing next-generation blockchain technology that is trustworthy and scalable. Their scalable blockchain is based on tamper-proof secure timelines and a graph-based data structure. This project will be built upon earlier work that has been done at the Blockchain Lab [52]. All of the Blockchain Lab's work is shared with the open source community.

2.3. Required technologies

Being a decentral application, there can be no central server. All communication must be shared between actors in the system, with no central point of trust or authority. This can be achieved by creating a peer-to-peer network for the decentral mortgage market application. A part of the assignment is to extend the existing decentral market [29] to include crowdfunded mortgages. This market community was built upon the Distributed Permission System (Dispersy) [13] for peer-to-peer networking. That means that building upon Dispersy is part of our assignment. Furthermore, this being a project for ABN AMRO for development of blockchain applications, it is expected that the market uses blockchain technology in some way.

2.3.1. Dispersy

Dispersy is a fully decentralized platform designed to simplify the creation of distributed communities. It uses elliptic curve cryptography [26] in their simple identity and messaging system to securely and anonymously identify different nodes. It can run on systems consisting of a large number of nodes, without having the need for any server infrastructure. Each node is equally important and they run the same algorithm to perform the same tasks, which increases the robustness of the system. Data is forwarded between nodes and will eventually reach all nodes.

2.3.2. Blockchain

A blockchain is a distributed database that holds a steadily growing list of data called blocks. Each block contains, among other things, a timestamp and a link to the previous block. Due to its design, the data in the blocks is resistant to modification. Once the block is added to the blockchain, the block can not be altered without the blockchain becoming invalid. This makes the blockchain a great tool to store data that should never be modified.

2.4. Project requirements

To prioritize the requirements, we used the MoSCoW method [32]. The requirements are subject to change during the development phase.

2.4.1. Must Have

The Must Have requirements provides the Minimum Usable SubseT (MUST) of requirements that the project guarantees to deliver. These are the core functionalities and without these, the product cannot be delivered. These requirements have the highest priority.

- Borrower can place loan requests.
- Borrower can upload documents needed to apply for a mortgage.
- Borrower can see offers they get from investors and financial institutions.
- Borrower can accept offers they get from investors and financial institutions.
- Investor can see which campaigns are available.
- Investor can place an offer on a campaign.
- Financial institution can create a quote for a mortgage.
- Financial institution can see their pending loan requests.
- Financial institution can review pending loan requests.

- Financial institution can accept pending loan requests.
- Python 2 to be able to interface with Tribler and Dispersy.
- PyQt for the GUI.
- Tests and coverage.
- Full transparency. Everything open-source.

2.4.2. Should Have

The Should Have requirements are important but not vital. The product is still viable without these functionalities. It has the highest priority after the Must Have requirements.

- Borrower can reject offers they get from investors and financial institutions.
- Investor can see which campaign they have currently invested in.
- Financial institution can see which mortgages they currently have provided.
- Financial institution can reject pending loan requests.
- Scalable.
- Blockchain technology.

2.4.3. Could Have

The Could Have requirements are wanted or desirable, but less important than the Should Have requirements.

- Borrower can see which campaigns are available.
- Investor can resell their investment.
- Investor can invest passively.
- Financial institution can see which campaigns are available.
- Financial institution can determine the maximum interest rate that a borrower is allowed to pay an investor for the loan.
- Financial institution can recommend an interest rate that the borrower can pay to an investor for the loan.
- Secure storage and transfer of information.
- Encrypt and decrypt user documents.

2.4.4. Won't Have

The Won't Have requirements will not be implemented in the final solution, because they are out of scope for this project.

- Borrower can see how much has already been paid off.
- Investor can see how much has already been paid off.
- Financial institution can see how much has already been paid off.
- Regulator can see the total financial health of all the active loans.
- Transactions can be done through the system.
- Smart contracts to ensure binding agreements between stakeholders.

- The system can do a risk assessment.

Smart contracts are out of scope for this project and will be researched and implemented by a PhD student after we have finished this project. Besides smart contracts, transactions through the system are out of scope for this project as well. Transactions can not be done through the system, therefore borrowers, investors, and financial institutions will not be able to see how much has already been paid off. There will not be a regulator as our market is a decentralized market. We have decided that the system will not be able to do a risk assessment, as this would be too complex to be achieved in the duration of this project.

2.5. Screens & Workflow

Based on the requirements, a list of pages the program should have to achieve these goals has been compiled. The screen can be either for borrowers (B), investors (I) or financial institutions (F).

Profile (BI)
The Profile screen differs depending on the role of the person that is logged in. It is used to save and update personal information. Investors only need to enter their name, email address and IBAN. Borrowers need to enter more personal information and they can upload personal documents needed to apply for a mortgage, such as payslips and proof of identity.

Portfolio (BIF)
The Portfolio screen differs depending on the role of the person that is logged in. Borrowers can see, accept and reject the offers they received from financial institutions and investors. Investors can see the campaigns they have invested in, and their pending investment offers. Financial institutions can see which mortgages they currently have running.

Place Loan Request (B)
Borrowers can request a mortgage, which will then be sent to all financial institutions of their choice. This will include information on the house, its price, and the amount they want to borrow. They can enter a small description as well, so an investor can learn more about who they are investing in.

Pending Loan Request (F)
Financial institutions can see all pending mortgage requests that have been sent to them. They can review each of these requests individually and accept or reject them. When they accept a mortgage request, they have to enter the details concerning the mortgage, such as interest rate and duration.

Open Market (BIF)
In the Open Market screen all investment opportunities are presented to all roles, but investments can only be made by investors. Key information shown about campaigns are the property address, the amount needed, the interest, the duration of the mortgage, and the remaining time of the campaign. Search and filter functions will be included as well.

Campaign Bids (BIF)
The Campaign Bids screen can be reached from the Open Market screen by selecting a campaign, and shows the current bids that investors have made in the campaign. Additionally, from here it is possible for investors to place an investment offer of their own by entering the amount they want to invest, the duration of the loan and the desired interest rate.

2.6. Related initiatives

To get some more insight into the current crowdfunding market, it is necessary to analyze initiatives related to crowdfunding, crowdfunded mortgages and crowdfunded loans. We found a couple of these initiatives of which thirteen will be discussed in this section.

2.6.1. Jungo

Jungo [24] is a mortgage bank, with the ability to crowdfund your mortgage. All clients whom are accepted by the bank will be guaranteed to get a mortgage. But Jungo differs from normal mortgage banks by making it possible for third parties and institutional investors to provide a part of the mortgage in return for an attractive interest.

At the moment of writing, it is not yet clear what the application process is for a mortgage via Jungo as they have not launched their service yet. All borrowers will undergo a risk assessment based on the Code of Conduct for Mortgage Loans [7].

Investors can choose which project they want to finance, for an amount between €250 and €80.000. The provisional interest is between 3% and 3.5%. They pay the money via an iDEAL transfer to the person requesting funding, and will be paid back monthly for up to 8 years. The length of the investment varies per Jungo campaign.

However, only 20% of the mortgage sum can be crowdfunded. The remaining amount must be provided by Jungo itself. As this seemed as a constraint set by law, we reached out to Leonard Franken (supervisor InnovationHub) from The Dutch Authority for the Financial Markets (AFM) [49] to get some insight. According to him, the limit to the amount that can be crowdfunded seems to be set by Jungo itself, as no limit has been set by law. He then mentions that this could have been done to make it more attractive to investors. From the investor's perspective 30 years is a long time to lend out their money. The shorter the duration, the harder it becomes for the borrower to pay back the debt. By making a smaller sum that needs to be paid back in a shorter period of time, the deal becomes more attractive for both parties.

Investors are protected from borrowers, who are not able to live up to their obligation, by a fund controlled by Jungo to continue paying investors when a borrower misses a payment. This fund is limited however.

Thus Jungo can be seen as a mortgage bank that also functions as intermediary for individuals and institutional investors, who wish to invest in a mortgage. Finally they provide a guarantee (up to a certain point) that your investment will be returned.

2.6.2. Blandlord

Blandlord [2] is a platform for crowd ownership of homes. The idea behind the platform is that home owners can put their house up for sale via Blandlord, even if the home is inhabited (the new landlord of the renters will become Blandlord). These houses can then be bought by a number of investors who can buy a percentage of the house. The rent income is then shared among investors depending on the percentage that they own. Thus Blandlord is not only a platform for selling the houses, it also functions as a letting agent.

All homes must be 100% owned by the owner before being placed on Blandlord. To minimize the risk that the homes stay empty due to lack of renters, only houses in areas with a healthy house market are accepted, as judged by Blandlord. Furthermore, an extensive audit of the condition of the house is required as a prerequisite of being able to list the house on Blandlord, to prevent the investors being faced by unforeseen costs.

Investors can buy shares in a house, with the minimum investment amount being €100. The value of the shares are directly correlated to the value of the house. They are paid their share of the rent on a monthly basis, based on the expected rental income. The costs of using the service is a 1% transaction fee on all transactions (buying/selling shares).

2.6.3. Greencrowd

The focus of Greencrowd [21] is to provide a crowdfunding platform for sustainable energy projects such as installation of solar panels and wind power. These projects will in turn provide income by selling the power or service, or for instance lower the energy costs of a building that has installed solar panels.

Organisations can present a plan for a sustainable energy project, which will then be analyzed by Greencrowd to make sure the plan is realistic. They will analyze the financial risks and assign a grade to each project, ranging from A to G. The interest an investor can expect is tied to the grade. The riskier the project, the higher the return. A project has a certain amount of time to get financed, if the finance goal is not reached before the deadline, all investors will be returned their money.

Investors can invest in all running projects, with the amount varying per project. The running time of the loan also varies per project. The only constant is that the interest is always tied to the grade the project has received. Payments vary per project, which can be monthly or quarterly, and they can include special provisions for paying off a loan earlier (the investor then receives a bonus from the investee).

2.6.4. Fundrise

Fundrise [17] is an online platform where individuals, accredited or non-accredited, are able to directly invest in commercial real estate projects from real estate companies, without the middleman. The main difference between the Fundrise eREITs (electronic Real Estate Investment Trust) and other REITs, is that Fundrise is more transparent and has roughly 90% lower fees. Fundrise enables investors to choose in which eREIT they want to put capital in.

To submit a project for a loan, the companies need to indicate the amount to raise and the project type, and they have to upload relevant documents needed for the application. First the information will be screened to ensure investment is consistent with Fundrise standards. Then follows the in-depth analysis, which includes an on-site visit by a member of the Fundrise team. When the project is accepted, it will receive a Fundrise rating, a letter rating ranging from A to E. This way, investors can easily compare different investments.

The minimum investment is $1.000, and the maximum may not exceed 10% of the investor's gross annual income or net worth. The fund transfer is done via bank account. When investors get their return, is specific to each listing. When available, a distribution schedule can be found in the offering documents for that investment. Generally investors can expect to receive distributions quarterly. Asset management fees are between 1% and 1.5%.

In the event that a company is not able to pay back the loan, the eREIT will bear the risk. The eREIT will pay the investors to the extent of any deficiency between the value of the collateral and the amount of the loan.

2.6.5. RealtyShares

RealtyShares [42] is a real estate crowdfunding platform that lets accredited investors choose what they invest in. Investors have the choice to invest in one specific property or in a group of properties. RealtyShares offers commercial and residential real estate projects.

To apply for financing, relevant information and documents should be provided by the borrower. RealtyShares then reviews whether the borrower prequalifies based on their track record, financial strength and expertise. After the pre qualification, borrowers will be thoroughly reviewed based on their investment strategy, financials, legal standing, and property condition/location.

Investors can start investing from $5.000. The funds transfer is via bank account. Depending on the type of investment, investors get their return either quarterly or monthly. Investor fees have a maximum of 2% of the invested amount. RealtyShares offers investors to loan their money supported by a collateral, a specific real estate property or pool of properties.

2.6.6. Collin Crowdfund

Collin Crowdfund [9] offers linear loans through crowdfunding to established businesses and startups. Collin returns to the basics of banking, where the investor decides what they invest in. Collin cooperates with banks to demonstrate what is realistic, and give the business the opportunity they deserve. This combination ensures that the investor makes a good return at a reasonable risk, and the business can realize their ambitions.

To apply for a loan, the business has to indicate the desired loan amount and the purpose for the loan, and they have to upload relevant attachments for their loan application. Collin then provides the business with a Crowdfund Coach to assist with filling out the Collin Credit Score model, which is a risk assessment that tests the company's solvency, profitability and liquidity. The Collin Credit Score produces a D&B rating [11] and the boundaries of the interest rate the business is allowed to set. Collin either approves or rejects the loan application using all of that information.

Investors can choose which projects they want to finance. Private investors can invest for an amount between €500 and €80.000 and corporate investors can invest from €500 with no upper bound. They pay the investment via an iDEAL transfer to the business requesting funding and will be paid back monthly for the duration of the project, which can be anywhere between 6 months and 10 years.

To protect investors from businesses that are not able to live up to their obligation, Collin continues paying investors for one month. When the business is not able to pay the investors back after that month, there will be no payment of interest to investors.

Collin asks for a small fee from both the business and the investors, which makes it possible for Collin to take care of the payments for both sides. For every investment, a management fee is calculated annually at a rate of 0.85%. The business is asked for a one-time fee of 1.9% for a successful loan and a monthly fee of 0.05%.

2.6.7. Upstart

Upstart [54] is a platform that allows people to obtain loans with a fixed rate. Upstart does not only look at credit scores, because they believe people are more than just credit scores. They use a model that considers the school the borrower has attended, their field of study, their academic performance, and employment history in addition to their credit history. That allows Upstart to offer people the loans they have earned.

To apply for a loan at Upstart, borrowers have to fill in an online application which will include information about their academic credentials, work experience and what they plan on doing with the loan proceeds. Upstart also verifies the borrower's personal and credit information as part of the application process.

Investors cannot choose in which project they invest, because loans are not visible publicly on Upstart. Investors can invest for a minimum investment amount of $100. All loans are fixed-rate loans with a 36 or 60-month repayment period. Investors will get an investor account on Upstart, to which they can transfer money to invest, or to transfer their returns to their own bank account.

Upstart determines the interest rate using the borrower's education, credentials, work experience, and credit history. This means that the interest rates are different for each loan. Once the interest is determined, all loans have a fixed-rate interest. The loans are unsecured obligations of the borrower and are not supported by any collateral.

As compensation for the marketing and ongoing servicing of loans, Upstart receives servicing fees that are paid from the origination fees that are collected from borrowers. Borrowers pay a one time origination fee that is between 1% and 6% of the target loan amount. Investors pay an annual servicing fee of 0.5%.

2.6.8. CrowdAboutNow

CrowdAboutNow [10] is a crowdfunding platform that makes it possible for entrepreneurs to start a crowdfunding campaign. They will support the entrepreneurs whenever they need it. CrowdAboutNow makes it possible to invest safely, both legally and technically.

To apply for a loan, entrepreneurs have to take a test to determine if their plans are suitable for crowdfunding. If their plan is suitable, an account manager will get in touch with the entrepreneur to discuss their plans.

Investors can choose the projects they want to invest in themselves. Investors can invest for an amount up to €80.000. They pay the investment via an iDEAL transfer, for which a fee between €0.70 and €0.80 is asked. It is not clear in what period of time and at what interest rate the investors will get their investments back.

To minimize the risk for investors, CrowdAboutNow verifies if the entrepreneur is willing to use its own network to invest its venture capital. CrowdAboutNow verifies the identity of the entrepreneur, the registration of the company, and its bank account. They also meet with the entrepreneur in person. If the entrepreneur has been registered for less than 3 years at the Chamber of Commerce [6], a partner that understands the industry will act as the nominating party. Should there not be enough investors to reach the loan target, all investors will get their money back.

CrowdAboutNow advertises entrepreneurs' crowdfunding campaigns, they give legal advice, and take care of the financial transactions. They do this for a one-time entry fee of 1% of the loan target and a 2% annual fee when the campaign is successfully funded.

2.6.9. Kapitaal op Maat

Kapitaal op Maat (KoM) [25] is a platform that makes it possible for different projects to get crowdfunded. Possible project funding sum can range from €25.000 to €500.000. Individual users can invest any amount between €100 and €50.000. Corporate investors do not have a hard ceiling, but both types of investors cannot invest beyond the amount that is needed to get the project fully funded.

To apply for a loan, the company or entrepreneur needs to present the website with a business plan of the venture. After that a fee of €395 is paid to KoM, to examine the presented offer and determine the risk of the venture. Additionally, if the venture is accepted, another fee of €350 must be paid to publish the venture on the website. The borrower will receive guidance with the right presentation of the offer. The calculated risk of the venture, which consists of a grade from A+ to C-, determines the interest. This will range from 5.5% for the lowest risk ventures (A+) and 9.5% for the highest risk ventures (C-).

When an investment is made, an initial fee of 0.9% of the investment is paid. All the investments are sent to the escrow account of KoM and stored until the funding goal is reached. If this goal is reached, a fee of 4% must be paid by the borrower to receive the funding. When the fees are paid and the transactions are made, the borrower will continue to pay the escrow a fixed rate every month until the investment and interest are completely paid off. After that, the escrow distributes the amount accordingly among the investors.

2.6.10. Geldvoorelkaar

Geldvoorelkaar [18] is a crowdfunding platform where Dutch corporations and individuals can invest in national, small to middle sized ventures.

The platform provides different types of loans which consist of: Fixed-rate, Interest-only and Convertible debt loans. The duration of the loan can range from 6 to 120 months. This can be as little as €1.000. There is no limit to how much an entrepreneur can ask, but no funding will be received until the loan goal is reached.

The fees for a funding request are €125 for private investors and between €350 and €500 for corporate investors. Additionally, when the loan goal is reached, the borrower pays a small fraction of the loan goal as success fee each year for the duration of the loan. This fee can range from 0.5% to 7.9% of the loan goal.

Geldvoorelkaar does background checks on everyone asking for a loan. Before a request can be made, the borrower has to present a business plan and a financial report. This is used to determine risk of the venture, which in turn will be used to determine interest of the loan. The higher the risk, the higher the interest. This is usually somewhere between 4% and 10% interest. After the transactions are made, the investors receive monthly payments by the borrower until the loan and interest are completely paid off.

2.6.11. Bouwsteen

Bouwsteen [5] is a new initiative that is going to be available in the first quarter of 2017. They want to provide a solution targeted at groups that are not served adequately in the market yet, namely starters (employees and self-employed) and seniors. Their goal is to create a financing product for the private housing market that allows for a reduction in the monthly payments, providing flexibility to the homeowner. Since Bouwsteen have not launched their service yet, it is not clear in what way the mortgages they will offer, will work and how they are planning on reducing the monthly payments.

2.6.12. MoneYou

MoneYou [31] is a bank that gives borrowers a lot of freedom with their mortgage. Unlike other banks, MoneYou does not give any advice in any form. It is up to the applier to decide what is best for them and their situation. The applier decides the specifics of the mortgages such as mortgage type, loan duration and loan amount.

What makes MoneYou special compared to other banks is that they facilitate the mortgage and all of the steps to get a mortgage online. In other words, it is possible for people to apply for a mortgage and go through most of the steps online. Before an application can be made, the applier needs to do a quiz on the MoneYou website. The quiz consists of multiple choice questions that should show that the applier knows what they are doing, and has the financial knowledge and assets to take out the mortgage. The quiz and other submitted documents, such as payslips and other financial documents are used by MoneYou to make a final verdict on giving out the mortgage or not. After the mortgage has been set, the definitive purchasing of the real-estate can be made with the help of a notary.

2.6.13. Gradefix

Gradefix [20] is a new initiative by ABN AMRO to automate credit checks. It is completely different than the other initiatives described in this section as it has nothing to do with crowdfunding, but credit checks are an integral part of any mortgage application. Gradefix calculates this score on the basis of the requester's bank transactions. By calculating the incoming and outgoing sums, and how much money is left at the end of each month, a credit rating is generated.

2.7. Development process

During the project we used the Scrum framework [44]. We worked in sprints of 2 weeks, and we focused on finishing new functionalities every sprint. Each chapter of this thesis reflects one sprint.

We held daily meetings within our group to discuss our progress, and weekly meetings with our coach to discuss our process and receive guidance. Additionally, several meetings were being held with our client to show our progress and receive feedback. These meetings ensured that we were able to stick to the wishes of the client and to incorporate feedback that they provided.

For revision control we used git, hosted on GitHub [19]. Our project code can be found at `https://github.com/Jumba/decentralized-mortgage-market`. During the the development of our software, we made use of Travis CI [50] and Jenkins [23] for continuous integration. Besides running the unittests on CI we elected to use Xenon [55], a monitoring tool that measures code complexity. It gives each function a grade based on their complexity, which ranges from A to F. Xenon has been configured to fail as soon as a function with complexity C or worse has been detected. The decision to use Xenon is explained in section 5.5.

We were also required to send our code two times during the course of the project to the Software Improvement Group (SIG) [46]. SIG reviewed our code and gave a rating representing how maintainable it is, according to their model. How we addressed the feedback can be found in section 5.6, and the full feedback can be found in appendix F.

3 The customer journey

After centering our first sprint around the orientation of the problem, we started working towards a demo for our client. We have designed the system, defined an application programming interface (API) and started working on visualising the core functionalities of the program in the graphical user interface (GUI).

The problem has three equally important stakeholders, as the system can not be functional without one of these stakeholders. These stakeholders are the borrowers, investors and financial institutions. In the following sections, the journeys of these stakeholders will be defined.

3.1. Researching the customer journey

Throughout the first sprint of the project, we concentrated on understanding the problem and related initiatives. Through analyzing the different initiatives, we came to understand more about crowdfunding real-estate.

Because our problem concerns people who need a mortgage, we had to understand how mortgages work as well. The two most common mortgages are Linear and Fixed-Rate [33], which are available for starters. Bart Gout, our real-estate/mortgage expert, provided us with a document that listed the required documents needed to apply for a mortgage and a scheme showing the process of getting a mortgage. These documents can be found in appendix B.

From the related initiatives, which were described in section 2.6, we found out that there are existing crowdfunding platforms for real-estate. A bank, MoneYou, that facilitates the mortgage and regulates all the steps to get a mortgage entirely online exists. A MoneYou representative explained that the required identity check can be delayed until the end of the application process when the borrower meets with a notary. Only the definitive purchase will have to be made with a notary. Thus the marketplace does not necessarily need to supply a way to verify the identity of a borrower.

3.2. Defining the customer journey

With the information acquired during the research into the problem in mind, we started defining the customer journeys. Each stakeholder has different requirements which the application has to accommodate. We have decided to describe these interactions, and visualize them using flowcharts. These descriptions can be found in the following sections, and the flowcharts can be found in appendix C.

3.2.1. The borrower's journey

The borrower's journey starts when the person opens the market software. The first screen the person sees is the Profile page. On this page they can add and edit all required personal information. The first time they fill in the profile they choose their role, which can be either borrower or investor. In this section, the chosen role will be borrower. Once the fields are filled in and all the required documents are uploaded, a prospective borrower can go on to apply for a mortgage. If they have already applied for a mortgage, they can check to see if any offers have been received from financial institutions or investors on the Portfolio page.

Applying for a mortgage is a fairly straightforward process, as all required documents will have been uploaded to the profile. The user will have to supply mortgage-specific information, such as house information, the real estate broker, and the amount wanted. They can choose which financial institutions to send the request to. Once sent, the only course of action is to wait for offers from the financial institutions.

Once a mortgage offer is received it will be shown to the borrower in their Portfolio. The borrower can then elect to accept the offer, or reject the offer. Once a mortgage offer is accepted the crowdfunding campaign will start. At this point, the mortgage offer is placed on the open market where investors can submit investment offers. As with the mortgage offer from financial institutions, offers from investors will be shown in the Portfolio where the borrower can accept or reject them.

Borrowers have the option to explore the Open Market. They will be able to see all current running campaigns and the offers that investors have placed on them, but they will not be able to place offers themselves.

3.2.2. The investor's journey

The investor's journey starts, like the borrower's journey, when the person opens the market software. The first screen the person sees is the Profile page. There they can add and edit their personal information, such as contact information and their IBAN. If it is their first time filling in the profile they have to choose their role, which can be either borrower or investor. In this section, the chosen role will be investor. After the required information is filled in they are able to see their portfolio, and search for investment opportunities on the open market.

On the Portfolio page, investors can monitor their investments. It lists all of the investor's current investments, and their pending investment offers.

If the investor wants to search for investment opportunities, they can do so at the Open Market. When an investment opportunity has been selected, they can see the current investment offers on that particular opportunity and place an offer of their own. Their investment offers and the status of the offers will be shown in their Portfolio.

3.2.3. The financial institution's journey

The financial institution's journey starts, like the borrower's and the investor's journey, when they open the market software. The first thing the financial institution sees is the Portfolio page. In their portfolio they can see the information on all of the mortgages that they have funded, and their pending mortgage offers.

To view mortgage requests they can go to the Pending Loan Request page. On this page there is a list of all the mortgage requests from people who want to buy a house. The financial institution is able to view every one of these mortgage requests individually.

When viewing an individual mortgage request, they will be able to view personal information about the borrower. They can see the documents that are needed to apply for a mortgage, and mortgage specific information, such as the house the borrower wants to buy and the amount of money they want to borrow. On this page they have the option to accept or reject the application. When they want to accept an application, they will have to fill in details on the mortgage they want to give out. Once the application has been accepted, it will be sent to the borrower who can then decide if they want to accept or reject the offer.

Financial institutions have the option to explore the Open Market. They will be able to see all current running campaigns and the offers that investors have placed on them, but they will not be able to place offers themselves.

3.3. Object-oriented analysis and design

In this section the object-oriented analysis will be documented. This analysis will use the problem description, as described in chapter 2, and the customer journeys as the source. Using them we will determine which objects will need to be represented in the system, how they will behave, and how they will interact with each other.

The first object identified is the User. This User can be either a borrower, investor or financial institution, which are immediately defined as a Role within the system. To simplify the system, a user can only have one role. Depending on the role, certain information will be required from a user. For instance, it is required to know the current address of a borrower, since the financial institution needs to know where to send mail. This information is not needed for the investor. Based on this, a Profile object has been created which includes all personal information of a user. To accommodate the extra information required for a borrower a BorrowersProfile has been created, which extends the Profile. For a mortgage request, an array of documents is required. These will be stored in a Document object.

When a borrower wants to request a mortgage, a House object is created with details of the house and

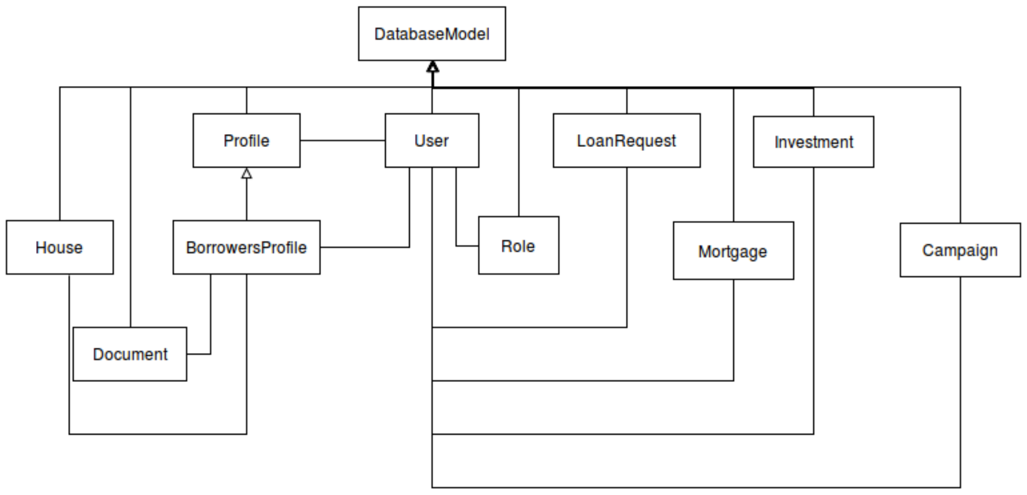


Figure 3.1: UML class diagram

its price, which will be sent to the financial institution along with a LoanRequest. In this LoanRequest, the borrower can specify the type of mortgage, and the amount wanted. As soon as a financial institution accepts the offer, this request is turned into a Mortgage which a borrower can the accept or reject. Once the mortgage offer has been accepted a Campaign will start, in which investors can invest by submitting Investment offers.

Since all objects will have to be encoded to be sent over the network, a DatabaseModel has been created from which all the other objects descend. This DatabaseModel implements functions to encode and decode the objects and generate unique IDs. The relations between these objects can be seen in the UML class diagram in figure 3.1.

3.4. Defining the API

To ensure that third-party programs can seamlessly access the decentralized mortgage market, an API has been designed. We call this API the Market API. The goal of this API is to provide all functions required to be able to make the customer journeys possible.

Based on the customer journey analysis, the following API functions have been created:

- create_user()
- create_profile(user, payload)
- create_loan_request(user, payload)
- create_campaign(user, payload)
- place_loan_offer(user, payload)
- login_user(private_key)
- load_profile(user)
- load_investments(user)
- load_open_market()
- load_borrowers_loans(user)
- load_borrowers_offers(user)
- load_borrowers_loan_status(user)
- load_all_loan_requests(user)
- load_single_loan_request(payload)
- load_bids(payload)
- load_mortgages(user)
- accept_mortgage_offer(user, payload)
- accept_investment_offer(user, payload)
- accept_loan_request(user, payload)
- reject_mortgage_offer(user, payload)
- reject_investment_offer(user, payload)
- reject_loan_request(user, payload)
- get_role(user)

The full API documentation has been included in appendix D.

3.5. Database structure

The market application stores all models in a key-value database. In this section this database will be defined, and implementation and design considerations will be argued.

During the research phase of the market application, it was made clear that there would be a meeting with the innovation department of ABN AMRO on December 5, 2016. We planned on showing them a basic clickable demo with the core functionalities, as described in section 3.6. With this date in mind, a simple database layer was constructed to feed the GUI with the necessary data.

Due to the rapid pace of development, the decision was made to make a key-value database, using the unique keys generated by the DatabaseModel as defined in section 3.3, along with the type of object as the keys. Thus the key of a User object is the user id and the type name is user. While a relational database would have been preferable due to the ability to search on all fields and the relations between tables, it was deemed too time consuming to implement and keep it up to date while working towards the demo. Furthermore the current implementation of the market application has no need for these searching abilities. However, future versions of the Market API could certainly benefit from advanced search functions, especially for the financial institutions to be able to construct SQL queries for their administrative needs.

As seen in figure 3.2, the database consists of two layers. The upper layer, Database, implements the database interface, which the Market API uses to communicate with the database. This database then stores the data in the given Backend, which can either be the MemoryBackend or the PersistentBackend. As the name suggests, the MemoryBackend stores all data in RAM and is flushed as soon as the application is closed. The persistent backend uses SQLite for persistence. The Dispersy library implements a generic SQLite database in `dispersy.database.Database`, which is used to implement the PersistentBackend. The purpose of the MemoryBackend is primarily for development and testing the code, whereas the PersistentBackend will always be used in production.

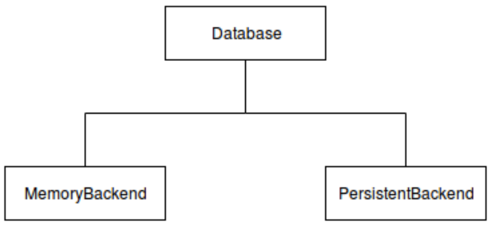


Figure 3.2: Database diagram

3.6. Visualising the core functionalities

Defining the journeys of the customers gave us a good understanding of the functionality that the GUI has to provide. The program's core functionality can be summarized by the following actions:

Placing a loan request
This can be done by a borrower when they want to apply for a mortgage. This action takes place in the Place Loan Request screen.

Reviewing loan requests
This can be done by financial institutions to assess a borrower's financial situation, using their personal information, and to determine if they are eligible for a mortgage. This action takes place in the Pending Loan Requests screen, which consists of two different sections. The first section shows a list of all pending loan requests, and the second section shows a detailed overview of a single loan request. Here they can accept or reject a loan request as well.

Accepting or rejecting a mortgage
This can be done by a borrower when they want to accept or reject a mortgage offer from a financial institution. This action takes place in the Portfolio screen.

Viewing campaigns
This can be done by all customers when they want to view the running crowdfunding campaigns. This action takes place in the Open Market and Campaign Bids screen, where they can view a list of all running campaigns, and all bids on a single campaign respectively.

Placing bids on campaigns
This can be done by investors when they want to make an investment on a campaign. This action takes place in the Campaign Bids screen.

Accepting or rejecting bids on campaigns
This can be done by borrowers when they want to accept or reject a bid that was placed on their campaign. This action takes place in the Portfolio screen.

A description of all of the screens that are mentioned above can be found in section 2.5. From these core functionalities, the first version of the GUI was made.

We created the screens using Qt Designer [39], which is a Qt [38] tool for designing and building GUIs. To control what information is shown in the screens, we used PyQt5 [37].

The screens that will be made in the next sprint exist to create a smooth user experience, and to make the core functionalities more accessible. The core screens, along with all other screens, can be found in appendix E.

4 Let's get connected

In the previous sprint we defined the customer journeys where we obtained an overview of the required information in our system. From this we were able to visualize the core functionalities in the GUI and define the API with the corresponding database structure and data models.

In this sprint we focused on maturing the remaining sections of the GUI, integrating Dispersy and the blockchain into the market, and implementing document transferring. These additions will connect all users, enable them to perform actions in a decentral network, and set the building blocks for a secure and robust application. We submitted our code to SIG on December 22nd, 2016 for a code quality check as well. In the end we had to extend this sprint by one week, due to not being able to complete the implementation of all functionalities within the two-week sprint time span.

4.1. Visualising the market

In the core screens of the GUI that were created last sprint, as described in section 3.6, not all functionalities of the market had been implemented yet. These missing functionalities can be summarized by the following actions:

Entering personal information
This can be done by borrowers and investors to make sure that the parties involved know where the funding or interest needs to go. This action takes place in the Profile screen. The personal information will be shown in the Portfolio screen once an offer has been accepted.

Uploading documents needed to apply for a mortgage
This can be done by borrowers before they can place a loan request. This action takes place in the Profile screen.

Viewing current loans
This can be done by borrowers and investors when they want to see their accepted loans. This action takes place in the Portfolio screen.

Viewing current mortgages
This can be done by financial institutions when they want to see their accepted and pending mortgages. This action takes place in the Portfolio screen.

Navigating between different screens
This can be done by all users when they want to view a different screen. This action takes place in all screens. All screens contain a navigation bar at the top of the page, which consists of links to other screens that the user has access to.

A description of all of the screens that are mentioned above can be found in section 2.5. With these functionalities, the full version of the GUI has been completed. All of the screens, including the core screens, can be found in appendix E.

4.2. Information flow design

There are three big stages to the market application, which are:

1. sending, accepting or rejecting a loan request;
2. sending, accepting or rejecting a mortgage offer;
3. and sending, accepting or rejecting an investment offer.

For each of these stages we have defined the information flow in the system. The stages each apply to a model, which are the LoanRequest, Mortgage, and the Investment models respectively. The model specification can be found in section 3.3.

When a borrower sends a loan request to a financial institution, the financial institution can respond by either rejecting the loan request or offering a mortgage to the borrower. The information flow for a loan request can be found in the sequence diagram in figure 4.1.

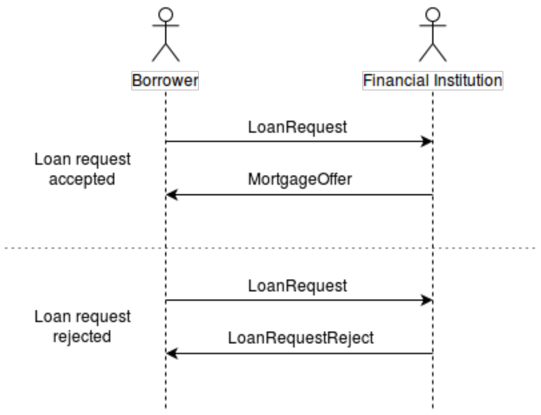


Figure 4.1: Information flow for loan requests

When a financial institution has sent a mortgage offer to a borrower, the borrower has the option to respond by either rejecting or accepting the mortgage. When the borrower accepts the mortgage, the borrower has to broadcast a message to other users to let them know that a new campaign has started. The information flow for a mortgage can be found in the sequence diagram in figure 4.2.

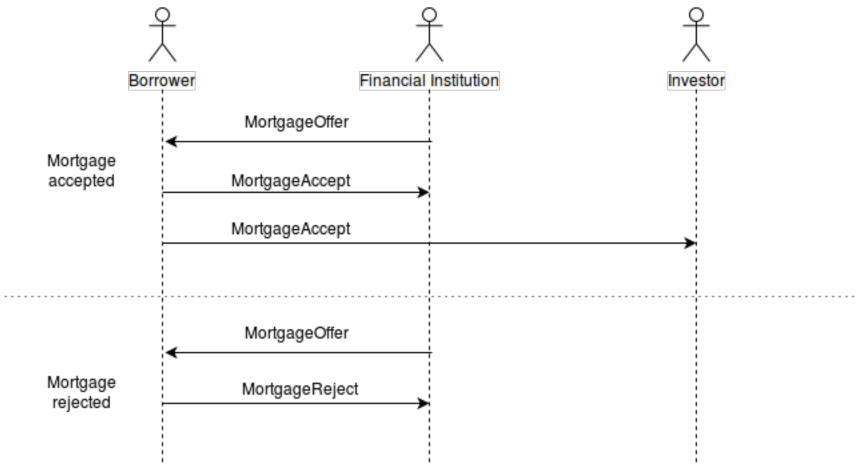


Figure 4.2: Information flow for mortgages

Once a borrower has accepted a mortgage, investors can place bids on their campaign by placing an investment offer on the borrower's campaign. The borrower can respond by either rejecting or accepting the investment. The information flow for an investment can be found in the sequence diagram in figure 4.3.

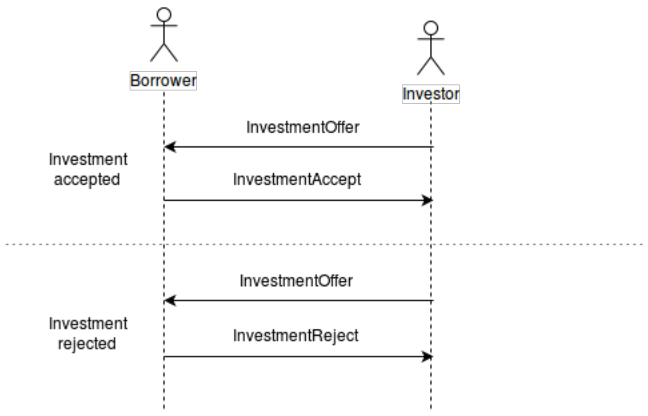


Figure 4.3: Information flow for investments

4.3. Integrating Dispersy

Dispersy lies at the heart of the decentralized mortgage marketplace. Integrating Dispersy to the market community proved to be a considerable challenge. It introduces a major paradigm change, since the program must work properly across different computers with each a different state. At the heart of Dispersy lies a simple identity and messaging system. These two systems have been fully integrated into the marketplace. In the following two subsections these integrations will be discussed.

4.3.1. Dispersy and Market Identity

Dispersy implements an identity system based on public and private keys. A Member is someone who is part of a Dispersy community, and is identified by a unique public key. As soon as the member is connected, a Candidate is created. A Candidate is the object representation of the network address of the Member. The Member answers the question of who, and the Candidate the question of where.

Having read the Dispersy documentation prior to designing the API, the decision was made to identify the users in the market using their public and private keys as well. To ensure that the identity of a user is always known offline as well as online, a key pair is generated and used for the creation of the Dispersy member and the market user. This key pair is then saved in the local database to be retrieved for future use. This process is completely transparent to the user.

The Candidate however is not saved to any persistent database, but rather to a memory location in the Market API. Candidates are volatile, their IP address can change often, and connections get dropped. Thus as soon as a user closes the application, the candidates list, which associates API User objects with candidates, is lost and has to be rebuilt at the next connection to the community.

4.3.2. Dispersy message sharing

Dispersy implements a variety of ways messages can be sent between peers. The method of sending a message is called a Distribution in Dispersy, and the recipient of a message is determined by the Destination. There are multiple implemented distributions and destinations, but for this project only two destinations and distributions will be used.

The CommunityDestination specifies that a message must be sent to a (specified) maximum number of (randomly selected) online members of the Dispersy community. To send a message to specific peers, the CandidateDestination must be used, which expects a list of peers to whom the message must be sent.

These destinations can be reached using two distribution methods. Using the DirectDistribution the message is sent directly to a peer, after which it will be discarded. This distribution is used in the market application alongside CandidateDestination to handle direct messages to peers. As mentioned above, the CommunityDestination sends the message to a list of randomly selected peers. To ensure that all peers receive the message, DirectDistribution is not advisable in this case, as after the randomly selected peers receive the message, it is discarded. This is solved by using the FullSyncDistribution.

The FullSyncDistribution instructs Dispersy to store the message for further propagation in the future. Furthermore, newly connected peers can ask online peers to send them all messages that they missed since they were last online. This message type is used for certain messages which ought to be shared to as many peers as possible, such as open campaigns and investments. This distribution method will be used alongside the CommunityDestination to share all public data.

When using FullSyncDistribution, models are constantly sent around in the network. Thus it is possible for two versions of the same model to be in circulation. Take for instance the Campaign object, which is updated every time an investment is accepted to set the new remaining amount that needs to be crowdfunded. To ensure that newer models are not overwritten by older versions, the notion of signing a model is introduced. By signing a model, a user signs the hash of the content of the model, and sets the signing time. Thus by comparing this time, it can easily be determined if a model is newer. This signing step takes place on all models before they are sent over the network.

4.4. Communication protocol specification

Defined in this section are the Dispersy messages and payloads, the communication between the Market API and the Dispersy Community, and the implementation of signed Dispersy messages. We will refer to two types of messages, Dispersy messages and Market API messages. Dispersy messages refer to the implementation of the Dispersy messages as defined in `dispersy.message.Message`. Market API messages on the contrary are defined by the market application to specify which handler should process the content of a specific incoming Dispersy message. This was done in an attempt to limit the amount of Dispersy messages needed and prevent duplicate code. First the Dispersy messages will be defined, followed by the Market API messages.

4.4.1. Dispersy Messages

As described in section 4.3, messages lie at the heart of Dispersy. Four messages have been defined for the mortgage market community. In Dispersy a message must always have a Payload, which is a class that specifies which data will be carried in the message. These payloads must have a decoder and encoder to serialize the data for transfer over User Datagram Protocol (UDP). To prevent having a host of payloads, encoders and decoders, an effort was made to make the payloads as generic as possible to enable reuse. The payloads that are implemented are:

DatabaseModelPayload
A payload with two attributes: fields and models. Where fields is the list of keys pertaining to DatabaseModels in the models attribute.

APIMessagePayload
This payload is similar to the DatabaseModelPayload, but with an extra attribute request. This request attribute is used to specify which API message handler needs to be used.

SignedConfirmPayload
This payload holds the following fields: benefactor, beneficiary, agreement_benefactor_encoded, agreement_beneficiary_encoded, sequence_number_benefactor, sequence_number_beneficiary, previous_hash_benefactor, previous_hash_beneficiary, signature_benefactor, signature_beneficiary, time. These hold the keys of the two signing parties, the agreements of both parties (which should be equal), the sequence number of the block in the blockchain as defined in section 4.6, the hash of the previous item in the blockchain, and the signature of the message. Finally, the time is saved.

The messages using these payloads are defined below.

introduce_user
Sent as a reply to the built-in Dispersy message to introduce oneself to other nodes (`Community.on_introduction_response`). Contains a DatabaseModelPayload with the User object.

api_message_community
This is an API message sent to the entire community using the FullSyncDistribution type. FullSyncDistribution enables the gossiping of messages, which will ensure that users who log in at a later time can get the message with a dispersy-sync request. Example usage is for instance the propagation of a new campaign. These messages are saved in the Dispersy database for further gossiping.

api_message_candidate
This is an API message sent directly to one or more explicitly given candidates. All communication between a borrower and a financial institution happens through direct messages. These messages are discarded directly after being received.

signed_confirm
The signed_confirm message is specifically created to seal an agreement between two parties. Using the DoubleMemberAuthentication provided by Dispersy, it requires that both parties sign the message if they agree with the message content. The message content being either a Mortgage or Investment. Thus before signing the message the system checks if the offer received is consistent with what has been agreed with. Only agreements which have passed through the signed_confirm process are treated as final and binding. More information about how signed_confirm is implemented can be found in section 4.6.2.

4.4.2. Market API Messages

In this section we describe the messages that are needed for the Market API, as described in section 3.4, and its communication protocols, which were described in section 4.2. These messages are different from the Dispersy messages described in section 4.4.1. The Market API messages will be shared across the network in the same Dispersy payload. This was done to remove the amount of duplicated code, since all API messages have the same payload format.

LoanRequest
A LoanRequest message is sent by a borrower to a financial institution when they create a loan request. The payload of the message contains the LoanRequest, House and BorrowersProfile objects.

LoanRequestReject
A LoanRequestReject message is sent by a financial institution to a borrower, when it rejects a loan request. The payload of the message contains the LoanRequest object.

MortgageOffer
A MortgageOffer message is sent by a financial institution to a borrower, when it accepts a loan request. The payload of the message contains the LoanRequest and Mortgage objects.

MortgageAcceptSigned
A MortgageAcceptSigned message is sent directly by a borrower to a financial institution, when they accept a mortgage offer. The payload of the message contains the Mortgage and Campaign objects.

MortgageReject
A MortgageReject message is sent by a borrower to a financial institution, when they reject a mortgage offer. The payload of the message contains the Mortgage object.

InvestmentOffer
An InvestmentOffer message is sent by an investor to a borrower, when they place an investment offer. The payload of the message contains the Investment and the Profile object.

InvestmentAccept
An InvestmentAccept message is sent by a borrower to an investor, when they accept the investment offer. The payload of the message contains the Investment and the BorrowersProfile object.

InvestmentReject
An InvestmentReject message is sent by a borrower to an investor, when they reject the investment offer. The payload of the message contains the Investment object.

CampaignBid
A CampaignBid message is sent by either an investor or a borrower, when they place a bid on a campaign or accept or reject a bid respectively. The payload of the message contains the Investment, Campaign, LoanRequest, House, and Mortgage objects.

4.5. Document Sharing

Dispersy is the platform that we use to host the distributed market. The program uses the Dispersy message packages to transfer different payloads between users. While that works well for small payloads, its maximum payload size is too small to hold a document. While theoretically it is still possible to send documents over Dispersy by splitting the messages, it is more practical to use a protocol like Trivial File Transfer Protocol (TFTP) [47] to transfer the documents.

To transfer the documents the program uses a module named tftpy [48]. With this module files can easily be uploaded to a server or downloaded from a server. In this case we want to upload the documents of a borrower to the financial institutions that were chosen during the placement of a loan request.

For simplicity, we make the assumption that the financial institutions are always online and are hosting a TFTP-server. Another assumption is that every user needs to upload the same set of documents, of which a list can be found in appendix B. Further we assume when the transfer of the documents fails, we will be able to try to transfer them again.

When the user places a loan request, all files are added to a queue and uploaded to all chosen financial institutions one by one. If something goes wrong with transferring the documents, the jobs are saved to be re-sent.

4.6. Blockchain

To ensure that no one can attempt to illegitimately change an agreement after it has been finalized, a blockchain is used. We reused the TU Delft MultiChain [34], but changed the body to save specific information relevant to our market. Another change is that we use limited sharing in our blockchain. Every block in the blockchain stands for an agreement between two users of the market. Only these two users have this block in their blockchain. The blockchains of these two users will not be synced with each other to form a larger blockchain for both users.

4.6.1. Architecture

The blockchain consists of individual blocks. Every block contains the fields in the SignedConfirmPayload listed in section 4.4.1 and two other fields: previous_hash and sequence_number. The contents of a block can be seen in figure 4.4. The agreement data is the same for both users that have the block in their blockchain, and this is the part that gets hashed. The last two fields are different for every user. The previous_hash contains the hash of the latest block in their blockchain and the sequence_number is the number of the block to be added, which should be the previous sequence number plus one.

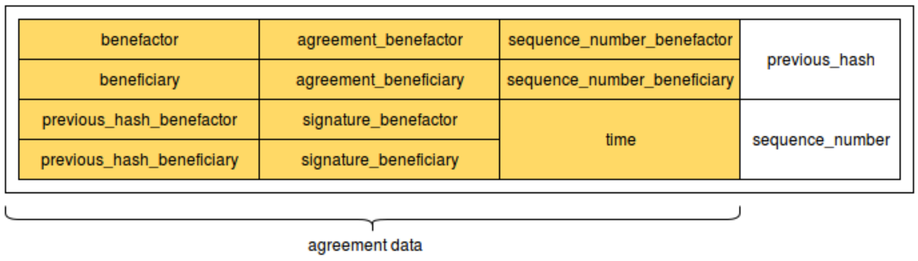


Figure 4.4: The fields in a block

The blocks of a blockchain are linked to each other through the hashes of the blocks. As each block contains the hash of the block before them, an ordered chain of the blocks is created. The first block added to the blockchain is the genesis block, which has sequence number zero. The previous_hash of the next block to be added is the hash of the genesis block. Another way to order the blocks to get the whole chain is by the block's sequence number. By using the sequence number of a block, we do not need to traverse through the hashes to get the chain. A graphical representation of a part of a blockchain can be found in figure 4.5. This figure shows how the blocks are linked to each other.

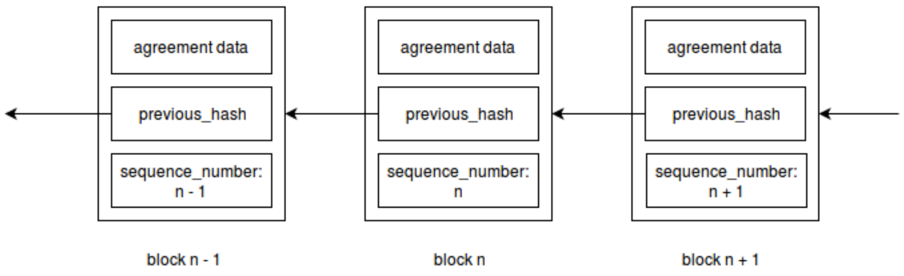


Figure 4.5: A chain of blocks in a blockchain

4.6.2. Signed messages and the blockchain

Dispersy implements signed messages, which are messages that are signed by two or more parties. Each party signs the payload using their private key. The purpose of these messages is to prove that the parties all agree over the content of a message.

When the beneficiary (borrower) accepts a mortgage offer or an investment offer, the benefactor (financial institution or investor) will create and send out a Dispersy signature request message to the beneficiary, with only the request part filled in. This message contains the SignedConfirmPayload listed in section 4.4.1, but not all fields are filled in by the benefactor. The fields beneficiary, agreement_beneficiary, sequence_number_beneficiary, previous_hash_beneficiary and signature_beneficiary will be filled in by the beneficiary. We call this group of fields the response. The benefactor persists the message and the corresponding block. A hash will be created from this message.

Upon receiving the signature request message, the beneficiary decides whether to accept or drop the message. This decision is made by comparing the agreement sent by the benefactor with the agreement the beneficiary has in its local database. When it has been decided to accept the message, the beneficiary will create and sign the response part of the message, and send it back to the benefactor. The message that is being sent back is called the signature response message. The beneficiary persists the message and the corresponding block. A hash will be created from this message.

Once the benefactor receives the signature response message, a final check will be performed. If the agreement_benefactor and agreement_beneficiary are the same, and there are no inconsistencies in the payload, the benefactor will accept the message. At this point the signing process is complete. The benefactor updates the message and the corresponding block. A new hash will be created from this message. A sequence diagram of the signed messages protocol can be found in figure 4.6

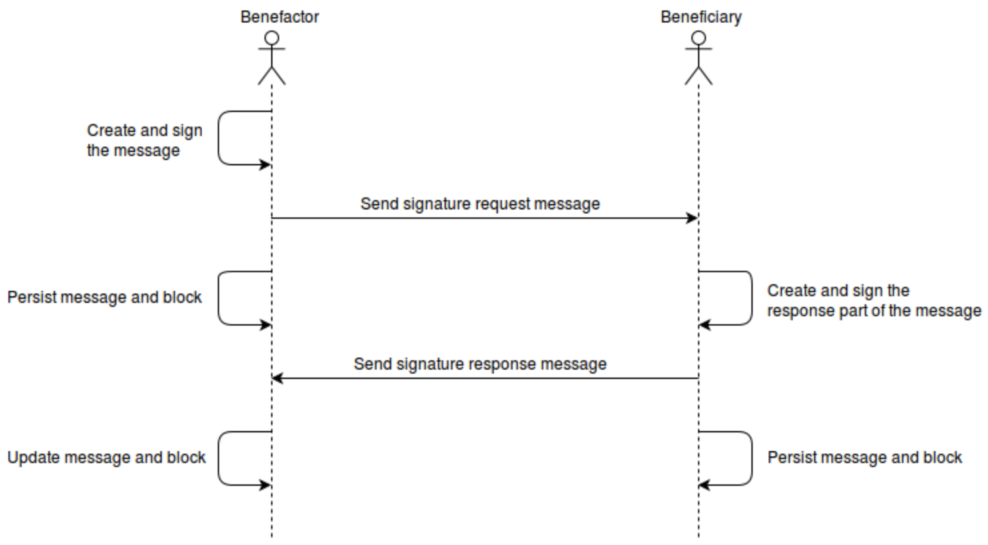

Benefactor
Beneficiary
Create and sign the message
Send signature request message
Persist message and block
Create and sign the response part of the message
Send signature response message
Update message and block
Persist message and block

Figure 4.6: Signed messages protocol sequence diagram

4.6.3. Blockchain storage model

The blockchain is stored in a SQLite database. The same database used for the market application, which is described in section 3.5, is used for this purpose, with a table specifically for the blockchain. An interface BlockChain has been defined in `market.database.backends.BlockChain`, which must be implemented by all backends supporting the blockchain. Currently only the PersistentBackend supports the blockchain. Since the blockchain could be tested using the PersistentBackend, there was no need to implement the BlockChain interface in the MemoryBackend.

The BlockChain interface defines the following public functions:

add_block
: Persist a block to the blockchain. This can be an incomplete block, which is only completed once it is updated with the information from the beneficiary.

update_block_with_beneficiary
: Update an existing block with information from the beneficiary. Only the relevant fields are updated with the new data.

get_latest_hash
: Get the hash of the latest block in the chain.

get_latest_sequence_number
: Get the sequence number of the latest block in the chain.

get_next_sequence_number
: Get the sequence number of the latest block in the chain plus one. If there is no latest block, the next sequence number is zero.

get_by_hash
: Returns a block saved in the chain with the matching hash.

get_by_public_key_and_sequence_number
: Returns a block saved in the chain where the given public key is equal to either benefactor or beneficiary, and the given sequence number is equal to sequence_number_benefactor or sequence_number_beneficiary

create_genesis_block
: Creates the genesis block and returns it.

check_add_genesis_block
: If there are no blocks in the blockchain, persist the genesis block to the blockchain.

5 Quality assurance

We have created a functional decentralized mortgage market prototype. In the final phase of the project, we focused on extending the test suite of the market and maturing the code. This resulted in about 4500 lines of application code and 3250 lines of test code. We had our client do an acceptance test as well, and showed multiple groups of colleagues of our client a demo of our prototype.

Last sprint we have sent our code for evaluation to SIG. In this chapter we will discuss the outcome of the SIG review, and how we have addressed their feedback. After processing the feedback and maturing the code, we sent our code to SIG again on January 24, 2017.

5.1. Unit testing

During the development of the market, we wrote unit tests to ensure that every feature of the market worked according to specification, and keeps working as intended during the development process. Unit tests test the functionality of small units of code, in our case a unit test tests a single function. That way the unit tests are independent of each other, and a function has to be tested only once. For the unit tests we used the built-in unittest library and the nose library [35], with nose being used to run the test and generate reports.

In some tests it is necessary to use certain objects that do not need to be tested themselves, or that require substantial amount of data and parameters to initialize, such as Dispersy. To simplify the tests and to be sure that only the function that we wanted to test is being tested, we mocked other objects that were needed in the test. To create mocks we used the mock library [30].

Ideally all tests are run locally before an update is pushed to the revision control server, but this is not always the case. To make sure test-breaking changes are caught as soon as possible, we employed continuous integration. For continuous integration Travis CI was used, along with Jenkins which was provided by the Blockchain Lab. Not all tests were able to run on these continuous integration platforms, because the requirements for running the GUI tests using PyQT's testing framework could not be installed. Nose was configured to ignore GUI tests to get around this issue.

To measure the extent of code tested by our tests, we took code coverage into account. Code coverage reports show what percentage of the code has been tested. If all of the lines of code in a file have been executed during the tests, the code coverage will be 100% for that specific file. These reports were generated by nose which generated raw .xml files. These were then parsed by Jenkins's cobertura plugin or pycobertura. Our final implementation consists of about 4500 lines of application code, about 3250 lines of test code, and our code coverage is 94%. Our coverage report can be found in table 5.1.

5.2. Integration testing

In this stage of the development process, we developed some integration tests to check the functionality of the whole program at once.

To perform the integration tests we set up some scenarios. Scenarios are functions which are executed every 5 seconds that perform tasks that mimic a role. We created different scenarios for each role, which means that we have scenarios for borrowers, financial institutions and investors.

The borrower's scenario creates a profile and then it sends a loan request to all financial institutions. When a loan request has been sent it waits until it gets mortgage offers, and when mortgage offers have been

Module	**Statements**	**Missing**	**Excluded**	**Coverage**
market.py	4	0	0	100%
market/api.py	1	0	0	100%
market/api/api.py	447	2	0	99%
market/api/crypto.py	10	0	0	100%
market/api/messages.py	12	0	0	100%
market/community.py	0	0	0	100%
market/community/community.py	330	14	0	96%
market/community/conversion.py	74	16	0	78%
market/community/encoding.py	165	3	0	98%
market/community/payload.py	95	4	0	96%
market/community/queue.py	64	3	0	95%
market/controllers.py	0	0	0	100%
market/controllers/banks_portfolio_controller.py	44	2	0	95%
market/controllers/borrowers_portfolio_controller.py	75	1	0	99%
market/controllers/campaign_bids_controller.py	36	0	0	100%
market/controllers/investors_portfolio_controller.py	43	1	0	98%
market/controllers/main_window_controller.py	81	1	0	99%
market/controllers/navigation.py	78	0	0	100%
market/controllers/openmarket_controller.py	33	0	0	100%
market/controllers/pending_loan_requests_1_controller.py	33	0	0	100%
market/controllers/pending_loan_requests_2_controller.py	65	1	0	98%
market/controllers/place_loan_request_controller.py	35	0	0	100%
market/controllers/profile_controller.py	88	2	0	98%
market/database.py	0	0	0	100%
market/database/backends.py	231	17	0	93%
market/database/database.py	53	4	0	92%
market/market_app.py	148	82	0	45%
market/models.py	86	1	0	99%
market/models/document.py	22	0	0	100%
market/models/house.py	21	0	0	100%
market/models/loans.py	165	6	0	96%
market/models/profiles.py	45	0	0	100%
market/models/role.py	6	0	0	100%
market/models/user.py	38	0	0	100%
Total	*2628*	*160*	*0*	*94%*

Table 5.1: Coverage report

received it accepts a random mortgage. After accepting a mortgage it waits until it gets investment bids, and accepts random investment bids when they have been received. The investor's scenario creates a profile and then waits for campaigns to be available on the open market. When it has found some campaigns, it sends investment offers to random campaigns. The financial institution's scenario waits for loan requests to come in. When loan requests has been received it accepts them randomly.

To run the scenarios, data has to be inserted. We generate fake data to put into the scenarios using the Faker [15] Python package. By running the scenarios at the same time we could verify the functionality, performance, and reliability of the market. We were able to view the GUI to see if the right information was displayed, and we were able to check the database to see if objects have been saved correctly.

5.3. Acceptance testing

To determine if the client's requirements are met, we conducted an acceptance test. We had our clients use our program as the different stakeholders, using tasks we gave them. These tasks can be found in appendix G. While they had the role of one stakeholder, we performed the other roles in the background. In this section we will discuss the feedback they gave us.

First, we had them try out the borrower role. They had to create a profile, create a loan request, accept a mortgage offer, and accept and reject some investment offers. Filling in the profile page went smoothly. When they had to switch to the page to place a loan request, they mentioned that the order of the navigation bars was a bit confusing because it did not follow the order of the process. When filling out a loan request, they noticed that the house number is a separate field. We originally did that, because we intended to use a postcode API to retrieve the address automatically. They mentioned that almost all houses that are for sale in the Netherlands are on funda [16], and it would be nice if we were able to automatically fill in the property information by retrieving the information from funda. They thought that a borrower should be able to decide the mortgage duration themselves. There was some confusion about the mortgage types. The fixed-rate mortgage type was confused with a fixed interest rate. After sending the loan request, the user did not realize that files were being sent in the background. Accepting a mortgage offer, and accepting and rejecting investments were also performed without problems. They did notice that they went over the wanted amount when accepting investments. They would have liked to see a feature, where the borrower could challenge an offer by sending a counter offer.

After trying out the borrower role, we had them try the financial institution role. They had to accept a mortgage. They had no problems opening a loan request. When entering the mortgage offer information to accept a mortgage, they were not sure if they had to enter a '%' sign for the interest rates, and they also did not know whether to use a '.' or a ','. When they entered a '%' sign or a ',' instead of a '.', a pop-up opened which said 'You didn't enter all of the required information'. It was not clear that it meant that they entered the information in the wrong way, as they filled in all of the required fields. Further they thought that it would be nice to be able to see the loan request details once a mortgage offer has been sent.

Lastly, they tried out the investor role. They had to create a profile, find a campaign to invest in, and place some investment offers. They had no issues entering the profile information, but they did mention that more personal information is needed when an investor wants to invest more than €500. When trying to place an investment bid, they thought that the open market was the screen in which they could place bids. They mistook one of the filter fields for a field in which they could enter the investment amount. Once we told them that they had to select and view a campaign to be able to place a bid, they had trouble finding the right screen because they did not see the button to view the campaign. When they were at the right page to place an investment offer, they had no problems placing one. They did mention that it would be nice to be able to see the campaigns once they are completed.

The acceptance test was conducted near the end of our project, which resulted in us not having enough time to process their feedback in our application. We decided to conduct the test and their feedback to the report for future reference.

5.4. ABN AMRO demo

After the acceptance test we took some time to demo the program on January 23, 2017 to people who work at the Innovation Centre at ABN AMRO, that were interested in our project. We showed a demo to multiple groups with different backgrounds. Some were more familiar with the technical side of the field, while others concentrated more on the business aspects. They were all enthusiastic about the market and the possibilities for in the future. During these demos they gave us valuable feedback, which is documented in this section.

One of the developers noticed that the market will have scaling problems due to the way nodes introduce themselves to each other. Introduction requests happen randomly, thus there is no way to ensure that a node finds a financial institution within an acceptable time span on a network with a large amount of nodes. We have not done any research on this topic, thus we can not with any certainty say how many nodes large refers to. But on a network with 1000 nodes, having 4 financial institutions would mean that there are at worst 995 nodes that can be discovered before a financial institution is found. Recognizing the severity of this issue, we have documented it and proposed a solution in section 6.3.

Nearly all the colleagues asked how the market application handles identity verification. The current market is fully trust based, no identity verification occurs at any point. From these questions we concluded that identity verification will be required for the market to be considered a viable product.

5.5. Code metrics

To ensure that we leave behind a maintainable code base, we have decided to generate code metrics results for the entire mortgage market application. There exist various opinions on using code metrics to determine maintainability. Coleman [8] for instance acknowledges that while no model is perfect, they can provide much needed feedback to developers to help guide their effort in making the code maintainable, while Deursen [12] mentions a few concerns with the Maintainability Index. His key argument is that there is no clear explanation for the formula behind the Maintainability Index. The only explanation is that all underlying metrics are directly correlated to line of code, and as such measuring lines of code and taking the average per module is a much simpler metric.

We elected to use Radon [41] to generate code metrics to highlight sections which could be refactored to decrease the complexity. Radon was used to great effect in our code base, identifying targets for refactoring during this sprint. An example of such a change can be found in appendix H. The large if-statement was removed and replaced with a simple dictionary, and the message names have been replaced with an Enum. Furthermore, after the refactor it became possible to test if all messages have a handler, by iterating through the APIMessage enum and ensuring that all messages have a handler connected to them.

5.6. SIG code quality

In this section we will describe the feedback we received from SIG on the quality of our code, and describe how we addressed it. The full feedback can be found in appendix F.

The market application was sent to SIG for the first review on December 22nd, 2016. The overall score for the system at this point was 4 out of 5 stars. Overall the code scored 5 stars for its small volume, lack of duplication and balance of components. 4 stars for the simple units, and loosely coupled modules and 3 stars for component independence.

The main issues identified by SIG were related to unit size and unit interfacing. Unit size refers to the length (lines of code) of functions, with each function longer than the average length lowering the score. The rationale behind having multiple shorter functions as opposed to one long function is to limit the amount of tasks a function has. This makes the function easier to understand and test. Unit interfacing relates to the amount of functions with an above-average number of parameters. Having an above-average amount of parameters usually indicates a lack of abstraction.

The unit size issues were primarily in the GUI related code. SIG noted that most of the longer functions were generated files by PyQt5. This has however already been solved as the GUI is now generated during run-time. A few classes have been refactored to split the functions into smaller units.

As for unit interfacing, the only function identified with an excessive number of parameters is the SignedConfirmPayload, defined in section 4.4.1, with 13 parameters. The problem with refactoring this payload to accept an object consolidating related parameters is that this would complicate the encoders and decoders of the payload. As such we have decided to leave this constructor as it is. To improve readability however, all instances where this constructor will be called have been changed to use named parameters.

6 Conclusion and evaluation

In the past ten weeks, we have fulfilled all the Must Have and Should Have requirements. In this chapter we will discuss the fulfillment of the project requirements listed in section 2.4 and discuss the challenges we have faced during the project. Finally we suggest recommendations for future work, and we conclude our project.

6.1. Fulfillment of requirements

In this section we look at the project requirements, and discuss which requirements have been fulfilled. The requirements that have been fulfilled are marked green, and the requirements that have not been fulfilled are marked red.

6.1.1. Must Have

- Borrower can place loan requests.
- Borrower can upload documents needed to apply for a mortgage.
- Borrower can see offers they get from investors and financial institutions.
- Borrower can accept offers they get from investors and financial institutions.
- Investor can see which campaigns are available.
- Investor can place an offer on a campaign.
- Financial institution can create a quote for a mortgage.
- Financial institution can see their pending loan requests.
- Financial institution can review pending loan requests.
- Financial institution can accept pending loan requests.
- Python 2 to be able to interface with Tribler and Dispersy.
- PyQt for the GUI.
  We used PyQt to control the GUI.
- Tests and coverage.
  We achieved this by creating unit tests and generating a coverage report every time we ran the tests.
- Full transparency. Everything open-source.
  All of our code is publicly accessible at `https://github.com/Jumba/decentralized-mortgage-market`.

6.1.2. Should Have

- Borrower can reject offers they get from investors and financial institutions.
- Investor can see which campaign they have currently invested in.
- Financial institution can see which mortgages they currently have provided.
- Financial institution can reject pending loan requests.
- Scalable.
  It is scalable, because we implemented the market upon Dispersy. Tribler is proof for the scalability of Dispersy.
- Blockchain technology.
  We implemented a blockchain into the market to save agreements that have been made between two parties.

6.1.3. Could Have

- Borrower can see which campaigns are available.
- Investor can resell their investment.
- Investor can invest passively.
- Financial institution can see which campaigns are available.
- Financial institution can determine the maximum interest rate that a borrower is allowed to pay an investor for the loan.
- Financial institution can recommend an interest rate that the borrower can pay to an investor for the loan.
- Secure storage and transfer of information.
- Encrypt and decrypt user documents.

6.1.4. Won't Have

- Borrower can see how much has already been paid off.
- Investor can see how much has already been paid off.
- Financial institution can see how much has already been paid off.
- Regulator can see the total financial health of all the active loans.
- Transactions can be done through the system.
- Smart contracts to ensure binding agreements between stakeholders.
- The system can do a risk assessment.

6.2. Challenges

During the development of the market we faced some challenges. In this section, those challenges and how we worked through them will be discussed.

The first challenge we came across was understanding the problem. The original project description was an issue on GitHub, of which a copy can be found in appendix A, which was not clear. On December 5, 2016, we held a demo for our client. It was not entirely clear what we had to show, and we had only one week to start developing. After the demo we were more confused about what the final product should be. As the project description was to create a decentralized blockchain-regulated open market for crowdsourcing real-estate, the final product would have to be be decentralized. During the demo, however, it became clear that our client wanted to use oracles and regulators which would make the system centralized, and they wanted us to use smart contracts as well. After more research and discussing with our coach, we narrowed down the scope and the direction of the project became more clear.

One of the challenges was integrating Dispersy into the market. Getting Dispersy to work was challenging due to its documentation. While it was helpful, we had trouble getting it to work because the example code was incorrect. With the help of Martijn from the Blockchain Lab this was resolved. We had trouble understanding how Dispersy worked at first. Due to the lack of in-depth documentation, a lot of Dispersy workings had to be deduced from Dispersy unit tests and the implementation of its functions.

We had trouble with Qt5 during the first couple of weeks. While it has proven to be a great framework, some of the functionalities were not working without some unintuitive tweaking. Another problem encountered with Qt, and specifically PyQt5, was that the Python documentation was missing. Instead, the official website referred to the C++ implementation, which in some cases had functions that were unknown in the Python version. Again, we solved this with the help of Martijn from the Blockchain Lab.

Getting Dispersy to work alongside Qt was more challenging than expected, since both Qt5 and Dispersy have an event loop that were competing for attention. The solution was to start twisted [53], which is an event-driven networking engine that Dispersy uses, using a qt5reactor [40] for twisted which handles the Qt5 event loop.

Sending files proved to be an issue, since we did not account for the maximum message size of 1500 bytes employed by Dispersy since it is based on UDP, and they are using the maximum Maximum Transmission Unit (MTU) on the internet of 1500 bytes to prevent fragmentation. The solution we used was to send the files using TFTP.

6.3. Known issues and future work

Currently there are a few known issues present in the market application. As mentioned in section 6.2, the maximum message size in Dispersy is 1500 bytes. It is currently possible for some messages to exceed this size, at which point they get dropped. While this is a rare occurrence, it needs to be resolved to ensure consistent and reliable messaging.

Furthermore, there are some significant security risks with the current implementation of the models, which are described in section 3.3, and the way they are propagated throughout the network. Any malicious user can modify their market application to send models throughout the network with incorrect information. The integrity of models is not ensured by the program, and as such there is no way of knowing if a model that is being distributed through the network has been changed by a peer.

The market is susceptible to attacks, due to its current implementation placing a significant amount of trust in the goodwill of the users. Take for instance a borrower looking for investors for their mortgage. Anyone can be an investor, pledge a certain amount to a campaign, and then disappear forever. By doing this repeatedly for all campaigns, an attacker can easily disrupt the system by not giving genuine investors the chance to invest in a campaign. A solution to this is to move away from trusting the users, and implement a blockchain with smart contracts. Smart contracts can be used to enforce a contract by making sure that nobody can back away from an agreement. However, because smart contracts are currently a work in progress and have proven to be a complex matter, the functionality has been deemed as out of scope.

There can be an issue when transferring documents. In the event that documents fail to be sent during the creation of a loan request, the program will retry multiple times to resend these files and send out the loan request even if not all documents were sent. The issue here is that the program will retry to send the documents only five times, which means that in some cases the financial institution does not receive all of the documents that belong to a loan request.

At the moment agreements between two parties can only be finalized if both parties are online. The mortgage request and investment flows described in section 4.2 use the CandidateDestination with DirectDistribution, which are explained in section 4.3.2, meaning that the messages are only sent to the concerning parties, and can not be propagated throughout the network. Thus if one of the parties goes offline during the making of an agreement, it will fail.

During the January 23, 2017 demo, that can be found in section 5.4, a developer identified an issue with the user discovery method used by the market application. At the moment each node connects to the community and waits until they receive an introduction request from the bank. This works fine for a small network with less than 20 peers. However, it is not optimal. Consider a network with 1000 nodes with 4 banks. It will take a considerable amount of time to find the bank, since there is no way to ensure that the bank gets connected to the node in a timely manner. A solution would be for the banks to send a Dispersy message using FullSyncDistribution and high priority with their IP address as payload, which is then gossiped around. As soon as a node receives this message, it creates a Candidate, as described in 4.3.1, pointing to the bank and add it to their candidates list. This does open the possibility for malicious nodes to lie about who the bank is. A possible solution for this is to check who signed the User object sent in the introduce_user message and confirming that it has been signed by the bank using their public key.

6.4. Ethical issues

One of the target audiences of our application are borrowers who can not receive mortgages from a financial institution, due to them not having enough money for the down payment. Using crowdfunding they can raise the money for the down payment. This implies that borrowers, who have been deemed risky investments for the financial institution, are now being financed by third-parties. Despite that the final implementation according to the requirements of the client will include a method to assess the risk, the borrower will still be borrowing the same amount of money. In this case the risk is shared between the financial institution and a number of investors, instead of only the financial institution. When a borrower defaults, it will have a larger effect on the investors' financial situation than on that of the financial institution.

This market application will enable the influx of foreign money into the Dutch real-estate market. By providing funds for borrowers through an unregulated market, foreign investors can invest in Dutch property, while taking the interest generated straight out of the Dutch economy.

6.5. Conclusion

The goal of this project was to create a prototype of a decentralized market in which mortgages can be crowdfunded. Users should be able to get a mortgage crowdfunded, and they should be able to invest in other users' crowdfunding campaigns.

To design the market, some related initiatives were researched first. Using that research, customer journeys were defined for the different stakeholders using flowcharts. Using these flowcharts we started designing the GUI, the database, and we created an API to be able to get the right information from the database to show in the GUI.

After creating the core functionalities we defined the information flow in the system. Using the information flow we started integrating Dispersy into the market, to be able to identify users and send messages between users. Due to Dispersy having a maximum message size, the documents that are needed to apply for a mortgage could not be send through Dispersy. To be able to transfer those documents, we made use of TFTP. To ensure that agreements made between two users (e.g. a financial institution giving a borrower a mortgage) can not be altered, a blockchain has been implemented to save these agreements.

The market that is developed for this project is a working prototype of a decentralized platform that can connect users with different goals. Users that are in need of funding to buy a house are able to get a mortgage quote from financial institutions, and they are able to receive investment bids from users that want to invest. Financial institutions can receive mortgage applications from users that want to buy a house, and partially fund those users' mortgages. Users that are looking for an investing opportunity are able to view open crowdfunding campaigns, and they are able to place investment bids on these campaigns.

Bibliography

[1] ABN AMRO. ABN AMRO. URL https://www.abnamro.nl/.

[2] Blandlord. Blandlord. URL https://blandlord.com/.

[3] Blockchain Lab. Blockchain Lab. URL http://www.blockchain-lab.org/.

[4] Blockchains. Blockchains: The great chain of being sure about things, 2015. URL http://www.economist.com/news/briefing/21677228-technology-behind-bitcoin-lets-people-who-do-not-know-or-t

[5] Bouwsteen. Bouwsteen. URL http://www.bouwsteen.info/.

[6] Chamber of Commerce. Kamer van Koophandel. URL https://www.kvk.nl/.

[7] Code Of Conduct For Mortgage Loans. Gedragscode Hypothecaire Financieringen / Code Of Conduct For Mortgage Loans, 2011. URL https://www.nvb.nl/publicaties-standpunten/publicaties/1671/gedragscode-hypothecaire-financieringen-code-of-conduct-for-mortgage-loans.html.

[8] Don Coleman, Paul Oman, Dan Ash, and Bruce Lowther. Using metrics to evaluate software system maintainability. *Computer*, 27(undefined):44–49, 1994. ISSN 0018-9162. doi: doi.ieeecomputersociety.org/10.1109/2.303623.

[9] Collin Crowdfund. Collin Crowdfund. URL https://www.collincrowdfund.nl/.

[10] CrowdAboutNow. CrowdAboutNow. URL https://crowdaboutnow.nl/.

[11] D&B Rating. D&B Rating. URL http://www.dnb-nederland.nl/dnb-data/rating.

[12] Arie Deursen. Think twice before using the maintainability index, 2017. URL https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/.

[13] Dispersy. Dispersy. URL https://github.com/Tribler/dispersy.

[14] Dutch National Bank. De Nederlandsche Bank. URL https://www.dnb.nl/.

[15] Faker. fake-factory 0.5.5. URL https://pypi.python.org/pypi/fake-factory/0.5.5.

[16] Funda. Funda. URL http://www.funda.nl/.

[17] Fundrise. Fundrise. URL https://fundrise.com/.

[18] Geldvoorelkaar. geldvoorelkaar. URL http://www.geldvoorelkaar.nl/.

[19] GitHub. GitHub. URL https://github.com/.

[20] Gradefix. Gradefix. URL https://gradefix.com/.

[21] Greencrowd. Greencrowd. URL https://greencrowd.nl/.

[22] Hypotheek Rijksoverheid. Hypotheek. URL https://www.rijksoverheid.nl/onderwerpen/koopwoning/inhoud/hypotheek.

[23] Jenkins. Jenkins. URL https://jenkins.io/.

[24] Jungo. Jungo. URL https://jungo.nl.

[25] Kapitaal op Maat. Kapitaal op Maat. URL https://www.kapitaalopmaat.nl/.

[26] Vivek Kapoor, Vivek Sonny Abraham, and Ramesh Singh. Elliptic curve cryptography. *Ubiquity*, 2008 (May):7, 2008.

[27] Loan to Value Ratio. Loan-To-Value Ratio - LTV Ratio, (n.d.). URL `http://www.investopedia.com/terms/l/loantovalue.asp`.

[28] Roy Meijer. TU Delft and ABN AMRO to collaborate on the development of blockchain applications, 2016. URL `http://www.tudelft.nl/en/current/latest-news/article/detail/samenwerking-tu-delft-en-abn-amro-voor-ontwikkeling-blockchain-toepassingen/`.

[29] J. Winter M.J.G. Olsthoorn. Decentral market: self-regulating electronic market. Master's thesis, Delft University of Technology, 2016.

[30] Mock. Mock. URL `https://pypi.python.org/pypi/mock`.

[31] MoneYou. MoneYou. URL `https://www.moneyou.nl`.

[32] MoSCoW Prioritisation. MoSCoW Prioritisation. URL `https://www.agilebusiness.org/content/moscow-prioritisation`.

[33] NHP Types of Mortgages. NHP Soorten hypotheken. URL `https://nhp.nl/page/?permalink=hypotheken/informatie/hypotheekvormen/soorten-hypotheken&piranha-culture=nl`.

[34] Steffan D Norberhuis. *MultiChain: A cybercurrency for cooperation.* PhD thesis, TU Delft, Delft University of Technology, 2015.

[35] Nosetests. Nosetests. URL `http://nose.readthedocs.io/en/latest/`.

[36] Project Description. Project Description issue on Github, 2016. URL `https://github.com/Tribler/tribler/issues/2606`.

[37] PyQt. PyQT. URL `https://riverbankcomputing.com/software/pyqt/intro`.

[38] Qt. Qt. URL `https://www.qt.io/`.

[39] Qt Designer. Qt Designer. URL `http://doc.qt.io/qt-5/qtdesigner-manual.html`.

[40] Qt5reactor. qt5reactor. URL `https://github.com/sunu/qt5reactor`.

[41] Radon. Radon. URL `https://github.com/rubik/radon/`.

[42] RealtyShares. RealtyShares. URL `https://www.realtyshares.com/`.

[43] RTL Z / Erik Rezelman. DNB-topman Klaas Knot: aftrek hypotheekrente versneld afbouwen, 2016. URL `http://www.rtlnieuws.nl/geld-en-werk/dnb-topman-klaas-knot-aftrek-hypotheekrente-versneld-afbouwen`.

[44] Scrum. Scrum Alliance. URL `https://www.scrumalliance.org/`.

[45] Manoj Singh. The 2007-08 Financial Crisis In Review, (n.d.). URL `http://www.investopedia.com/articles/economics/09/financial-crisis-review.asp`.

[46] Software Improvement Group. Software Improvement Group. URL `https://www.sig.eu/`.

[47] TFTP. RFC 1350 - The TFTP Protocol (Revision 2). URL `https://tools.ietf.org/html/rfc1350`.

[48] TFTPy. TFTPy - A Pure Python TFTP Protocol Implementation. URL `http://tftpy.sourceforge.net/`.

[49] The Dutch Authority for the Financial Markets (AFM). Autoriteit Financiële Markten. URL `https://www.afm.nl/en`.

[50] Travis CI. Travis CI. URL `https://travis-ci.org/`.

[51] Tribler. About Tribler. URL `https://www.tribler.org/about.html`.

[52] Tribler Wiki. Tribler GitHub Wiki. URL `https://github.com/Tribler/tribler/wiki`.

[53] Twisted. Twisted. URL `https://twistedmatrix.com/trac/`.

[54] Upstart. Upstart. URL `https://www.upstart.com/`.

[55] Xenon. Xenon. URL `https://github.com/rubik/xenon`.

[56] Zonder geld van je ouders is het onmogelijk een huis te kopen. Zonder geld van je ouders is het onmogelijk een huis te kopen, 2016. URL `http://nos.nl/op3/artikel/2115595-zonder-geld-van-je-ouders-is-het-onmogelijk-een-huis-te-kopen.html`.

Appendices

A
Project description: Open market for real-estate crowdsourcing

Market platforms such as AirBnB, eBay, Uber, and Blandlord.com bring supply and demand together. Based on our ongoing "blockchain-regulated markets" we now explore usage in various domains. This issue addresses an open market for crowdsourcing in real-estate.[36]

Outcome of initial brainstorm sketches:

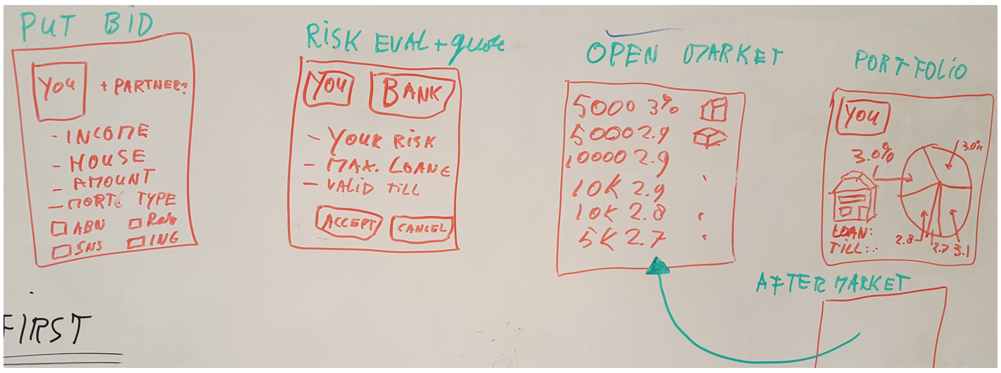


Figure A.1: Brainstorm sketches

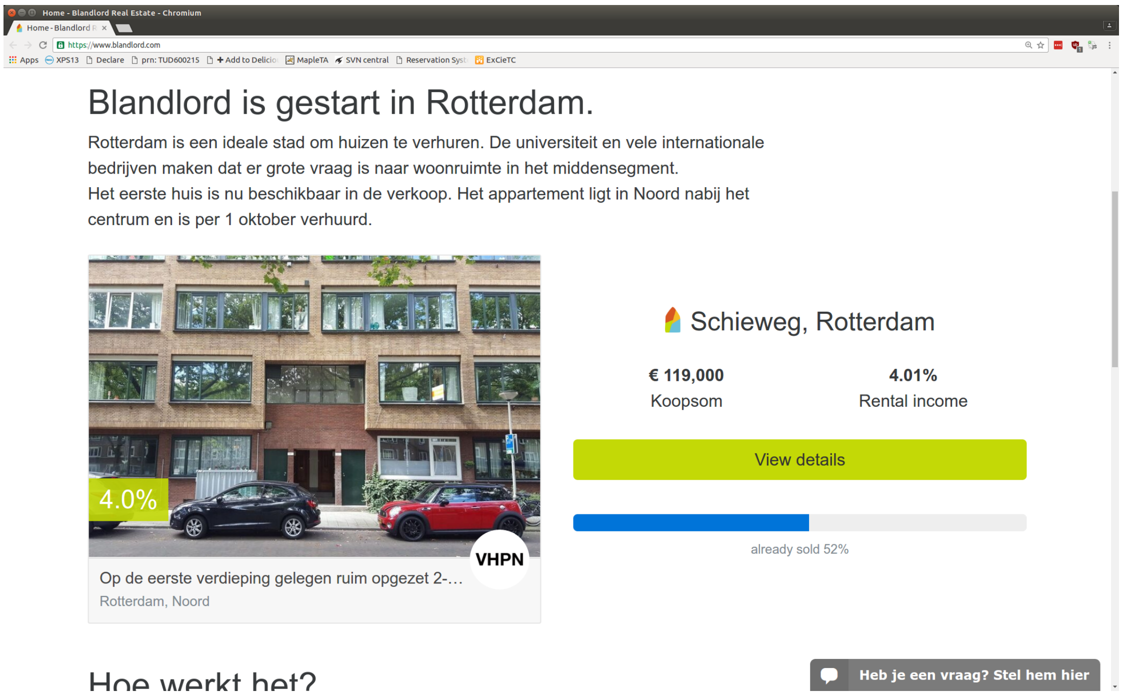
Sprints:

- Week 1-2: Write-up of context and understanding of problem, understand related initiatives. Describe at least 10 initiatives. Learn Python.
- Week 3-4: Apply Python. Visualize the concept using PyQT designer with 4 operational screens. Feedback round from bank.
- Week 5-6:
- Week 7-8:

Roadmap:

- Feb/Jan: we have 3 components ready in draft: operational market + GUI implemented in QT + business case
- till April: define API between market and business logic, create business logic core, integrate the market, business logic, and GUI.
- April: present MvP results

Various markets for real-estate already exist, such as:



Blandlord is gestart in Rotterdam.

Rotterdam is een ideale stad om huizen te verhuren. De universiteit en vele internationale bedrijven maken dat er grote vraag is naar woonruimte in het middensegment.
Het eerste huis is nu beschikbaar in de verkoop. Het appartement ligt in Noord nabij het centrum en is per 1 oktober verhuurd.

4.0%

VHPN

Op de eerste verdieping gelegen ruim opgezet 2-…
Rotterdam, Noord

Schieweg, Rotterdam

€ 119,000
Koopsom

4.01%
Rental income

View details

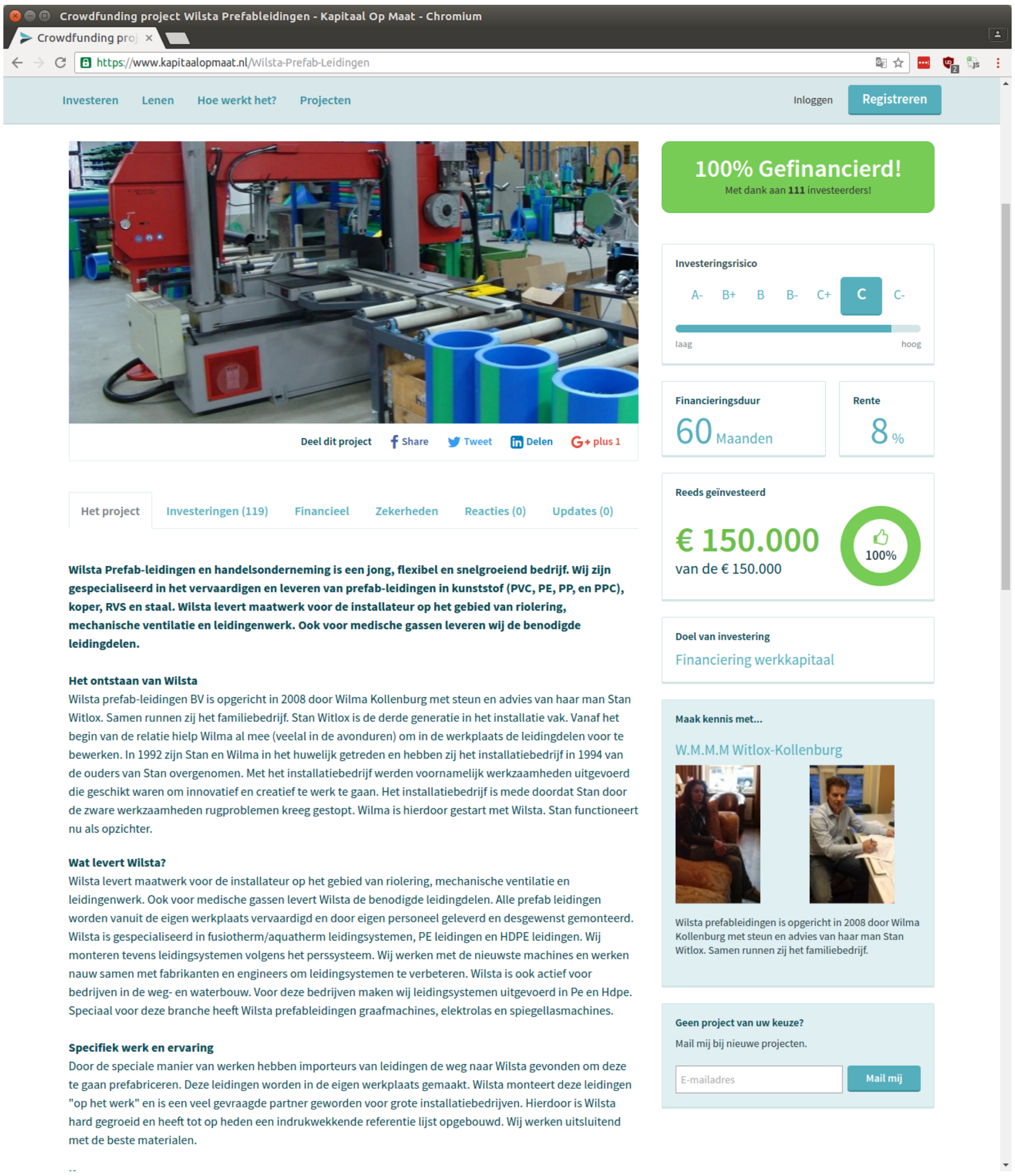
already sold 52%

Hoe werkt het?

Heb je een vraag? Stel hem hier

Figure A.2: Blandlord is gestart in Rotterdam

Plus an operational crowdsourcing market, running locally which TUDelft indirectly has access to:



Wilsta Prefab-leidingen en handelsonderneming is een jong, flexibel en snelgroeiend bedrijf. Wij zijn gespecialiseerd in het vervaardigen en leveren van prefab-leidingen in kunststof (PVC, PE, PP, en PPC), koper, RVS en staal. Wilsta levert maatwerk voor de installateur op het gebied van riolering, mechanische ventilatie en leidingenwerk. Ook voor medische gassen leveren wij de benodigde leidingdelen.

Het ontstaan van Wilsta
Wilsta prefab-leidingen BV is opgericht in 2008 door Wilma Kollenburg met steun en advies van haar man Stan Witlox. Samen runnen zij het familiebedrijf. Stan Witlox is de derde generatie in het installatie vak. Vanaf het begin van de relatie hielp Wilma al mee (veelal in de avonduren) om in de werkplaats de leidingdelen voor te bewerken. In 1992 zijn Stan en Wilma in het huwelijk getreden en hebben zij het installatiebedrijf in 1994 van de ouders van Stan overgenomen. Met het installatiebedrijf werden voornamelijk werkzaamheden uitgevoerd die geschikt waren om innovatief en creatief te werk te gaan. Het installatiebedrijf is mede doordat Stan door de zware werkzaamheden rugproblemen kreeg gestopt. Wilma is hierdoor gestart met Wilsta. Stan functioneert nu als opzichter.

Wat levert Wilsta?
Wilsta levert maatwerk voor de installateur op het gebied van riolering, mechanische ventilatie en leidingenwerk. Ook voor medische gassen levert Wilsta de benodigde leidingdelen. Alle prefab leidingen worden vanuit de eigen werkplaats vervaardigd en door eigen personeel geleverd en desgewenst gemonteerd. Wilsta is gespecialiseerd in fusiotherm/aquatherm leidingsystemen, PE leidingen en HDPE leidingen. Wij monteren tevens leidingsystemen volgens het perssysteem. Wij werken met de nieuwste machines en werken nauw samen met fabrikanten en engineers om leidingsystemen te verbeteren. Wilsta is ook actief voor bedrijven in de weg- en waterbouw. Voor deze bedrijven maken wij leidingsystemen uitgevoerd in Pe en Hdpe. Speciaal voor deze branche heeft Wilsta prefableidingen graafmachines, elektrolas en spiegellasmachines.

Specifiek werk en ervaring
Door de speciale manier van werken hebben importeurs van leidingen de weg naar Wilsta gevonden om deze te gaan prefabriceren. Deze leidingen worden in de eigen werkplaats gemaakt. Wilsta monteert deze leidingen "op het werk" en is een veel gevraagde partner geworden voor grote installatiebedrijven. Hierdoor is Wilsta hard gegroeid en heeft tot op heden een indrukwekkende referentie lijst opgebouwd. Wij werken uitsluitend met de beste materialen.

--

100% Gefinancierd!
Met dank aan **111** investeerders!

Investeringsrisico
A- B+ B B- C+ [✓ C] C-
laag hoog

Financieringsduur
60 Maanden

Rente
8 %

Reeds geïnvesteerd
€ 150.000
van de € 150.000
100%

Doel van investering
Financiering werkkapitaal

Maak kennis met...
W.M.M.M Witlox-Kollenburg

Wilsta prefableidingen is opgericht in 2008 door Wilma Kollenburg met steun en advies van haar man Stan Witlox. Samen runnen zij het familiebedrijf.

Geen project van uw keuze?
Mail mij bij nieuwe projecten.

Figure A.3: Kapitaal op Maat

B

Mortgage Process

Welke documenten heb je nodig voor een hypotheek?
Als je een hypotheek afsluit, heb je nogal wat documenten nodig. Welke documenten je nodig hebt voor een hypotheek, ligt aan de fase van het hypotheekadviestraject. En aan jouw situatie. Sommige documenten heb je al nodig bij het oriëntatiegesprek. Andere documenten heb je pas nodig bij het afsluiten van je hypotheek.

Documenten voor hypotheek per fase
In onderstaande tabel zie je welke documenten je voor je hypotheek nodig hebt in welke fase. Dit zijn de 4 fasen van het hypotheekadviestraject waarin verschillende documenten nodig zijn:

1. Oriëntatie: berekening maximale hypotheek en voorbeeld maandlasten.
2. Hypotheekadvies: hypotheek samenstellen en producten vergelijken.
3. Hypotheekofferte aanvragen.
4. Hypotheek afsluiten: getekende hypotheekofferte naar de geldverstrekker.

Documenten voor hypotheek per situatie
In onderstaande tabel zie je welke documenten je voor je hypotheek nodig hebt in welke situatie. De algemene documenten heb je altijd nodig. De overige documenten alleen als die situatie voor jou geldt.
Alle documenten mogen een scan of kopie zijn, tenzij er "(origineel)" achter staat.

Identificatie	**1**	**2**	**3**	**4**
Paspoort of ID-kaart	x	x	x	x
Paspoort of ID-kaart (origineel)*	x	-	-	-
* Origineel moet gecontroleerd worden.				

Werk	**1**	**2**	**3**	**4**
Loonstrook	x	x	x	x
Werkgeversverklaring	x	x	x	-
Jaaropgaven afgelopen 3 jaar *	x	x	x	x
Arbeidsovereenkomst **	-	-	-	x
Rekeningafschrift met bijschrijving netto salaris	-	-	-	x
Arbeidsverleden via Mijn UWV	x	x	x	-

* Als je geen vast contract én geen intentieverklaring hebt.
** Als ingangsdatum minder dan 6 maanden geleden is.

Uitgaven	1	2	3	4
Budgetoverzicht *	-	x	-	-
* Bijvoorbeeld via Nibud Persoonlijk Budgetadvies.				

Vermogen	1	2	3	4
Afschrift spaarrekening(en)	-	x	x	x
Opgave beleggingen	-	x	x	x

Voorzieningen werkgever	1	2	3	4
Uniform Pensioen Overzicht (UPO)	-	x	x	x
Opgave MijnPensioenOverzicht.nl	-	x	x	x
Verzekering arbeidsongeschiktheid	-	x	x	x

Woning	1	2	3	4
Getekende koopovereenkomst	-	-	x	x
Taxatierapport *	-	x	x	-
* PDF van een gevalideerd taxatierapport.				

Verbouwing	1	2	3	4
Specificatie verbouwingskosten	-	x	x	-

Schenking ouders	**1**	**2**	**3**	**4**
Getekende overeenkomst	-	-	-	x
Paspoort of id-kaart ouders	-	-	-	x

Lening ouders	**1**	**2**	**3**	**4**
Getekende overeenkomst	-	-	-	x
Paspoort of id-kaart ouders	-	-	-	x

Lening	**1**	**2**	**3**	**4**
Overeenkomst	-	x	x	x
Bankafschrift met maandbedrag	-	x	x	x
Bewijs aflossing *	-	-	-	x
Bewijs afmelding BKR *	-	-	-	x
Bewijs eigen geld *	-	-	-	x

* Als je jouw lening of krediet moet opzeggen voor je hypotheek.

Studielening	**1**	**2**	**3**	**4**
Opgave DUO begin schuld	-	x	x	x
Opgave DUO huidige schuld	-	x	x	x

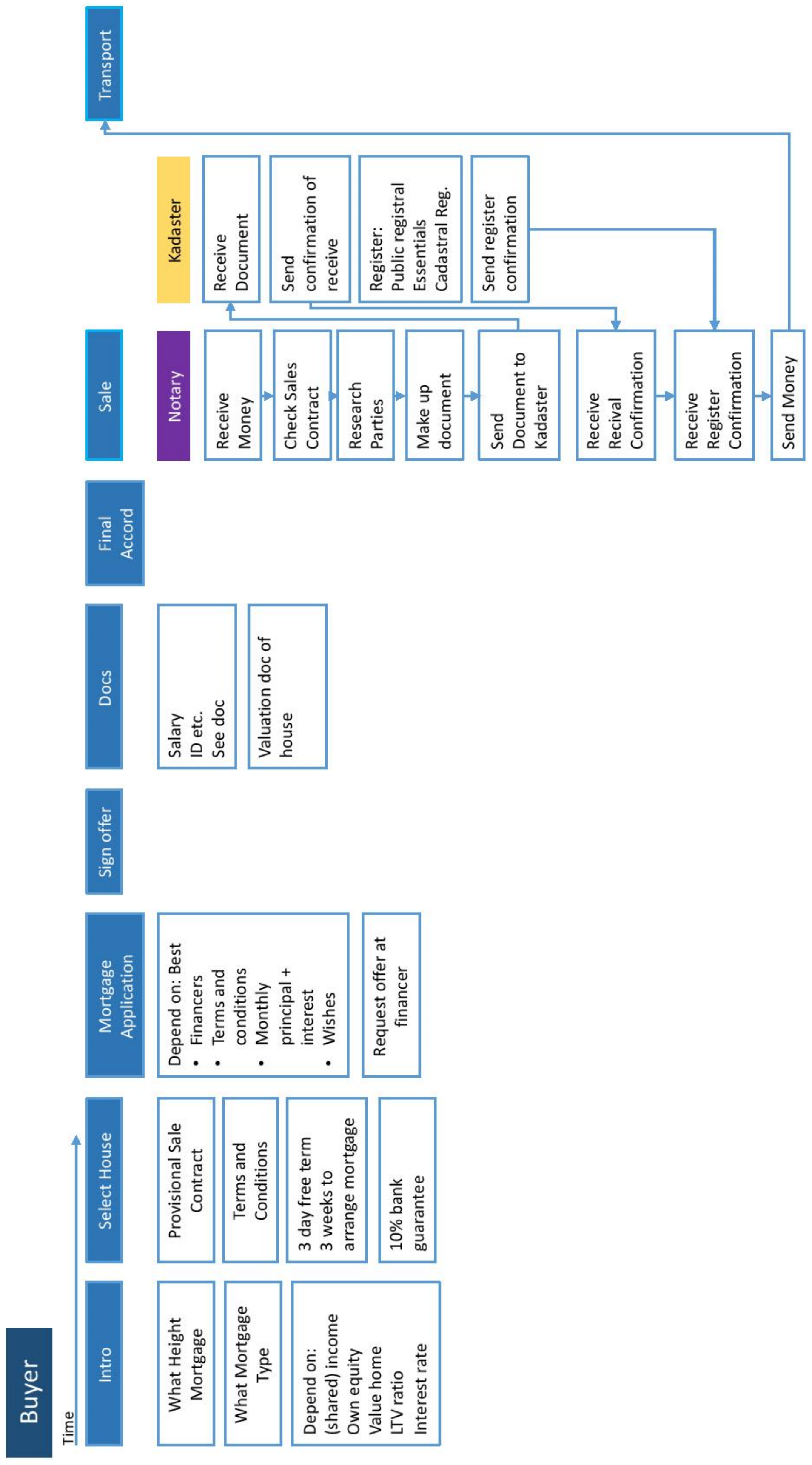
Gescheiden	**1**	**2**	**3**	**4**
Convenant	-	x	x	x
Inschrijving gemeente	-	x	x	x
Akte van verdeling *	-	x	x	x

Vonnis rechtbank **	-	x	x	x
Inschrijving vonnis **	-	x	x	x
* Als je partner de woning overneemt.				
** Als jouw scheiding via de rechtbank gaat.				

Flexwerker	**1**	**2**	**3**	**4**
Jaaropgaven afgelopen 3 jaar	-	x	x	x
Arbeidsovereenkomst	-	x	x	x

Zelfstandig ondernemer	**1**	**2**	**3**	**4**
Uittreksel KvK	-	-	x	x
Jaarcijfers 3 jaar	x	x	x	x
Aangiftes IB 3 jaar	x	x	x	x
Aanslagen IB 3 jaar	x	x	x	x
Voorlopige cijfers	x	x	x	x

Uitkering	**1**	**2**	**3**	**4**
Specificatie uitkering	x	x	x	x
Jaaropgaaf	x	x	x	x


Buyer
Time
Intro
What Height Mortgage
What Mortgage Type
Depend on: (shared) income Own equity Value home LTV ratio Interest rate
Select House
Provisional Sale Contract
Terms and Conditions
3 day free term 3 weeks to arrange mortgage
10% bank guarantee
Mortgage Application
Depend on: Best Financers Terms and conditions Monthly principal + interest Wishes
Request offer at financer
Sign offer
Docs
Salary ID etc. See doc
Valuation doc of house
Final Accord
Sale
Notary
Receive Money
Check Sales Contract
Research Parties
Make up document
Send Document to Kadaster
Receive Recival Confirmation
Receive Register Confirmation
Send Money
Kadaster
Receive Document
Send confirmation of receive
Register: Public registral Essentials Cadastral Reg.
Send register confirmation
Transport

C
Customer journeys

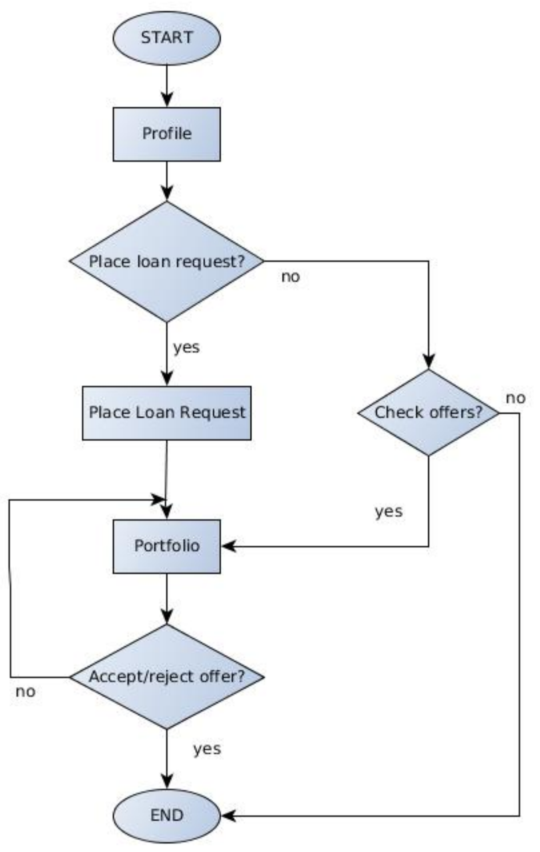


Figure C.1: Borrower's journey flowchart

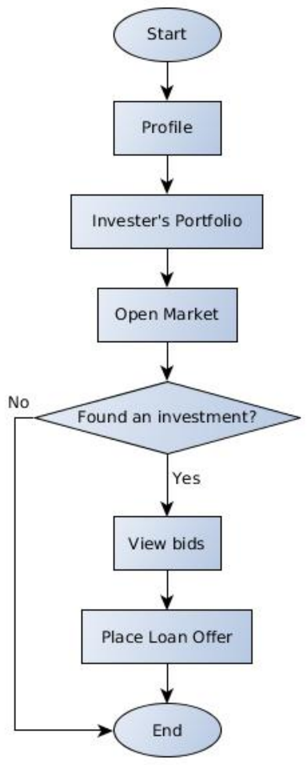


Figure C.2: Investor's journey flowchart

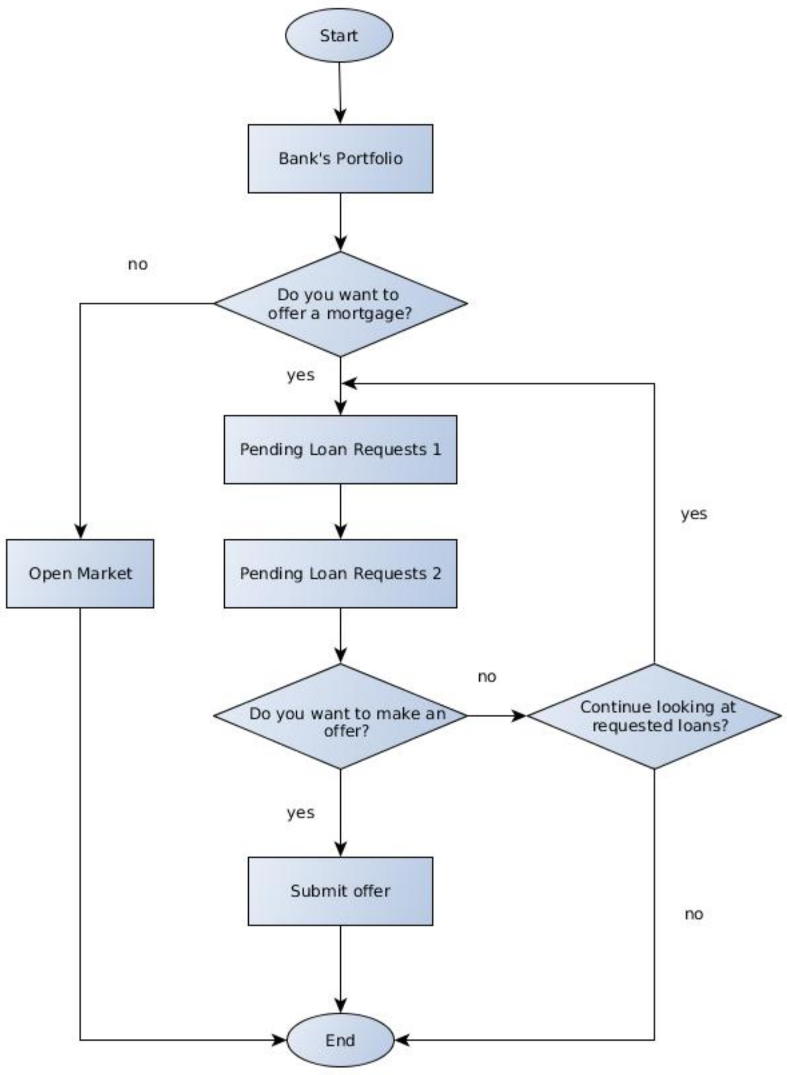


Figure C.3: Financial institution's journey flowchart

D

Market API

Market API

class `market.api.api.MarketAPI`(*database*) [source]

Create a MarketAPI object.

The constructor requires one variable, the *Database* used for storage.

`accept_investment_offer`(*user, payload*) [source]

Accept an investment offer for the given user.

The payload dictionary has the following composition

Key	Description
investment_id	The id of the investment

Parameters:
- **user** (`User`) – The user accepting an investment offer.
- **payload** (*dict*) – The payload containing the data for the `Investment`, as described above.

Returns: Returns True if successful, False otherwise.

Return type: bool

Raise: AssertionError if the user does not have a campaign assigned.

`accept_loan_request`(*bank, payload*) [source]

Have the loan request passed by the payload be accepted by the bank calling the function.

The payload is as follows:

Key	Description
request_id	The `LoanRequest` id
amount	The amount the bank is willing to finance
interest_rate	The interest rate to be paid over the financed amount (float)
default_rate	The default rate (float)
max_invest_rate	The maximum investment interest rate (float)
duration	The duration of the mortgage
risk	The risk associated with the loan

Parameters:
- **bank** (`User`) – The bank accepting the loan request.
- **payload** (*dict*) – The payload containing the data for the `Mortgage` , as described above.

Returns: Returns the loan request and the mortgage objects, or None if an error occurs.

Return type: tuple(LoanRequest, Mortgage) or None

`accept_mortgage_offer`(*user*, *payload*) [source]

Accept a mortgage offer for the given user.

This action automatically rejects all other mortgage offers.

The payload dictionary has the following composition

Key	Description
mortgage_id	The id of the mortgage

Parameters:
- **user** (`User`) – The user accepting a mortgage offer
- **payload** (*dict*) – The payload containing the data for the `Mortgage` , as described above.

Returns: Returns True if successful, False otherwise.

Return type: bool

`create_campaign`(*user*, *mortgage*, *loan_request*) [source]

Create a funding campaign with crowdfunding goal the difference between the price of the house and the amount requested from the bank. #TODO: Should it be more flexible?

Parameters:
- **user** (`User`) – The `User` for who the mortgage is being made
- **mortgage** – The `Mortgage` pertaining to the house being financed
- **loan_request** (`LoanRequest`) – The `LoanRequest` created prior to the mortgage being accepted.

Type: mortgage: `Mortgage`

Returns: True if created, False otherwise.

Return type: bool or False

`create_loan_request`(*user, payload*) [source]

Create a Loan request for the given user using the payload provided.

The payload dictionary has the following composition

Key	Description
postal_code	The postal code of the house that is the target of the mortgage
house_number	The house number of the house that is the target of the mortgage
address	The address of the house that is the target of the mortgage
price	The total price of the house
seller_phone_number	The phone number of the seller
seller_email	The email of the seller
mortgage_type	The mortgage type: 1 = linear2 = fixed-rate
banks	List of banks the request should be sent to
description	Free text (unicode)
amount_wanted	The amount the borrower wants financed

Parameters:
- **user** (`User`) – The user creating a loan request
- **payload** (*dict*) – The payload containing the data for the `House` and `LoanRequest` , as described above.

Returns: The loan request object if successful, False otherwise

Return type: `LoanRequest` or False

`create_profile`(*user, payload*) [source]

Creates a new profile and saves it to the database. The profile can either be a normal Profile or a BorrowersProfile, depending on the role given in the payload. Overwrites the old profile. The role 'FINANCIAL_INSTITUTION' can not have a profile, but the role will be set. Thus the function will return *True*.

The payload contains the following data. If the role is *1* for *BORROWER* then the last three fields must also be pushed.

Key	Description
role	The role id uit of the following tuple: ('NONE','BORROWER', 'INVEST
first_name	The user's first name
last_name	The user's last name

Key	Description
email	The user’s email address
iban	The user’s IBAN
phonenumber	The user’s phone number
current_postalcode	The user’s current postal code
current_housenumber	The user’s current house number
current_address	The user’s current address
documents_list	A list of documents

Parameters:
- **user** (`User`) – The user for whom a profile has to be made
- **payload** (*dict*) – The payload containing the data for the pofile, as described above.

Returns: The Profile if a borrower or investor role was set, True if a bank role was set, False if the

payload is malformed :rtype: `Profile` or True or False

`create_user()` [source]

Create a dispersy user by generating a key public/private pair.

Returns None if the user creation process fails.

Returns: A tuple with the User object, the public ey and the private key. The keys encoded in HEX.

Return type: (`User` , Public, Privat) or None

db

Returns the database object

`get_role`(*user*) [source]

Get the role of the user from the database.

Parameters: **user** – The `User` whose role you want to retriee

Returns: : Returns the role or None.

Return type: `User` or *None*

`load_all_loan_requests`(*user*) [source]

Display all pending loan requests for the specific bank

Parameters: **user** (`User`) – The bank `User`

Returns: A list of lists containing the :any: ‘LoanRequest’s and the :any: ‘House’s

Return type: list

`load_bids`(*payload*) [source]

Returns a list of all bids on the selected campaign.

The payload dictionary has the following composition

Key	Description
mortgage_id	The id of the selected mortgage

Parameters: **payload** (*dict*) – The payload containing the data for the `Investment` , as described above.

Returns: A list :any: ‘Investment’ objects, a :any: ‘House’ object, and a :any: ‘Campaign’ object.

Return type: list, House, Campaign

`load_borrowers_loan_status`(*user*) [source]

Get the borrower’s campaign if it exists or loan request if it exists :param user: User-object, in this case the user has the role of a borrower :return: Campaign if it exists, else LoanRequest if it exists, None otherwise

`load_borrowers_loans`(*user*) [source]

Get the borrower’s current accepted loans :param user: User-object, in this case the user has the role of a borrower :return: list of the loans, containing the current accepted loans, and the investor’s profile

`load_borrowers_offers`(*user*) [source]

Get all the borrower’s offers(mortgage offers or loan offers) from the database. :param user: User-object, in this case the user has the role of a borrower :return: list of offers, containing either mortgage offers or investment offers :rtype: list

`load_investments`(*user*) [source]

Get the pending and current investments list from the investor.

Parameters: **user** (`User`) – The user whose investments need to be retrieved.

Returns: A list containing lists with the investments, the house, the campaign, and the borrower’s profile

Return type: list

`load_mortgages`(*user*) [source]

Display all pending and running mortgages for the bank

Parameters: **user** (`User`) – The bank `User`

Returns: A list of lists containing the :any: 'Mortgage', the :any: 'House', the :any: 'Campaign',

and the :any: 'BorrowersProfile' :rtype: list

`load_open_market`() [source]

Returns a list of all mortgages that have an active campaign going on.

Returns: A list containing lists with `Mortgage` objects, :any: 'House' objects, and :any: 'Campaign'

objects. :rtype: list

`load_profile`(*user*) [source]

Load the given users profile.

Depending on the user's role, it will return a `Profile` for an investor or a `BorrowersProfile` for a borrower. None for a financial institution

Parameters: **user** (`User`) – The user whose profile has to be loaded.

Returns: `Profile` or `BorrowersProfile` or None

`load_single_loan_request`(*payload*) [source]

Display the selected pending loan request

Returns: A list of lists containing the :any: 'LoanRequest', the :any: 'Profile' of the borrower that wants a

mortgage, and the :any: 'House' that the borrower wants :rtype: list

`login_user`(*private_key*) [source]

Login a user by generating the public key from the private key supplied, and searching the user object in the database using the generated key.

Parameters: **private_key** (*str*) – The private key of the user encoded in HEX

Returns: The logged in User if successful, None otherwise.

`place_loan_offer`(*investor, payload*) [source]

Create a loan offer by an investor and save it to the database. This offer will always be created with status as 'PENDING' as the borrower involved is the only one allowed to change the status of the loan offer.

The payload contains the following data:

Key	Description
amount	The amount being invested
duration	The duration of the loan in months
interest_rate	The interest due to be paid over the loan
mortgage_id	The id of the mortgage being financed

Parameters:
- **investor** (`User`) – The investor wishing to invest in a mortgage by placing a loan offer.
- **payload** (*dict*) – The payload containing the data for the `Investment`, as described above.

Returns: The loan offer if successful, False otherwise.

Return type: `Investment` or False

`reject_investment_offer`(*user, payload*) [source]

Decline an investment offer for the given user.

The payload dictionary has the following composition

Key	Description
investment_id	The id of the investment

Parameters:
- **user** (`User`) – The user rejecting an investment offer.
- **payload** (*dict*) – The payload containing the data for the `Investment`, as described above.

Returns: Returns True if successful, False otherwise.

Return type: bool

`reject_loan_request`(*user, payload*) [source]

Decline an investment offer for the given user.

The payload dictionary has the following composition

Key	Description
request_id	The id of the loan request

Parameters:
- **user** (`User`) – The bank rejecting a loan request.
- **payload** (*dict*) – The payload containing the data for the `LoanRequest`, as described above.

Returns: Returns the rejected :any: 'LoanRequest' if successful, None otherwise.

Return type: LoanRequest or None

`reject_mortgage_offer`(*user, payload*) [source]

Decline a mortgage offer for the given user.

The payload dictionary has the following composition

Key	Description
mortgage_id	The id of the mortgage

Parameters:
- **user** (`User`) – The user rejecting a mortgage offer
- **payload** (*dict*) – The payload containing the data for the `Mortgage`, as described above.

Returns: Returns True if successful, False otherwise.

Return type: bool

`reject_pending_campaign_bids`(*user, campaign*) [source]

Checks if the campaign is completed. If so, rejects all pending bis on the campaign.

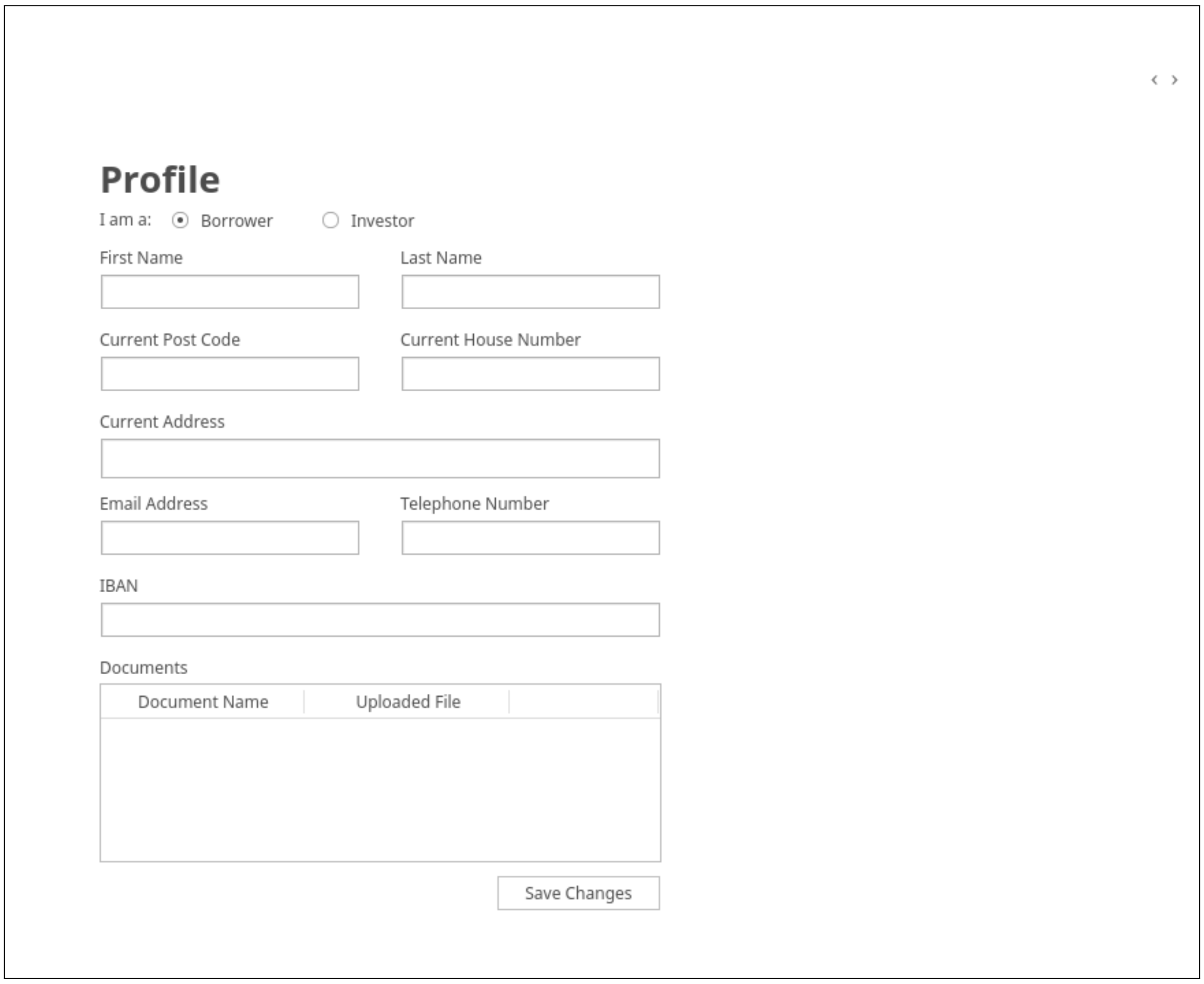
Parameters:
- **user** (`User`) – The user accepting an investment offer.
- **campaign** – The campaign that needs to be checked

`user_key`() [source]

Returns the user key the API communicates as.

E

GUI



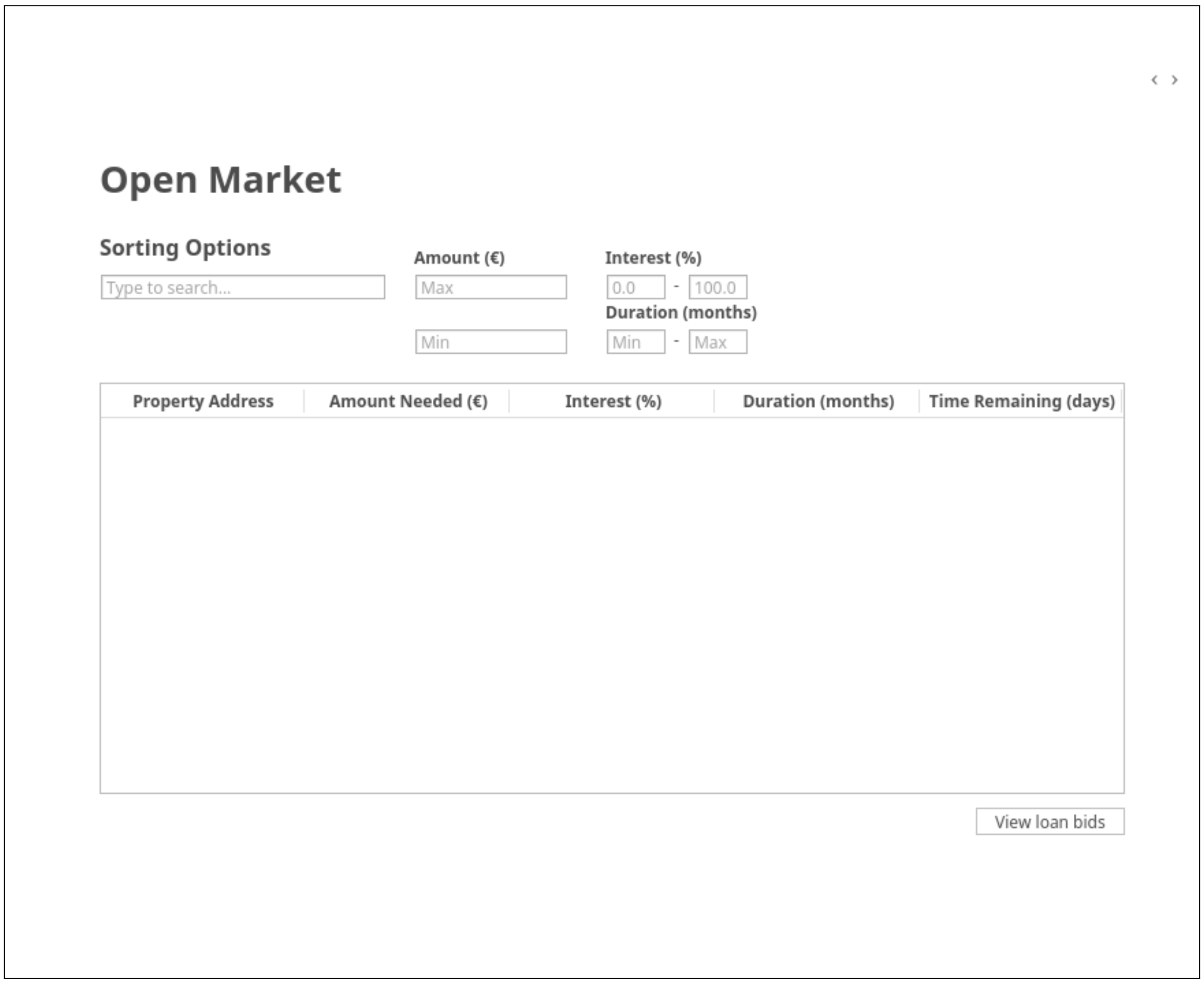
Profile

I am a: [✓ Borrower] Investor

First Name	Last Name
Current Post Code	Current House Number
Current Address	
Email Address	Telephone Number
IBAN	

Documents

Document Name	Uploaded File	



Open Market

Sorting Options

Type to search...

Amount (€)

Max

Min

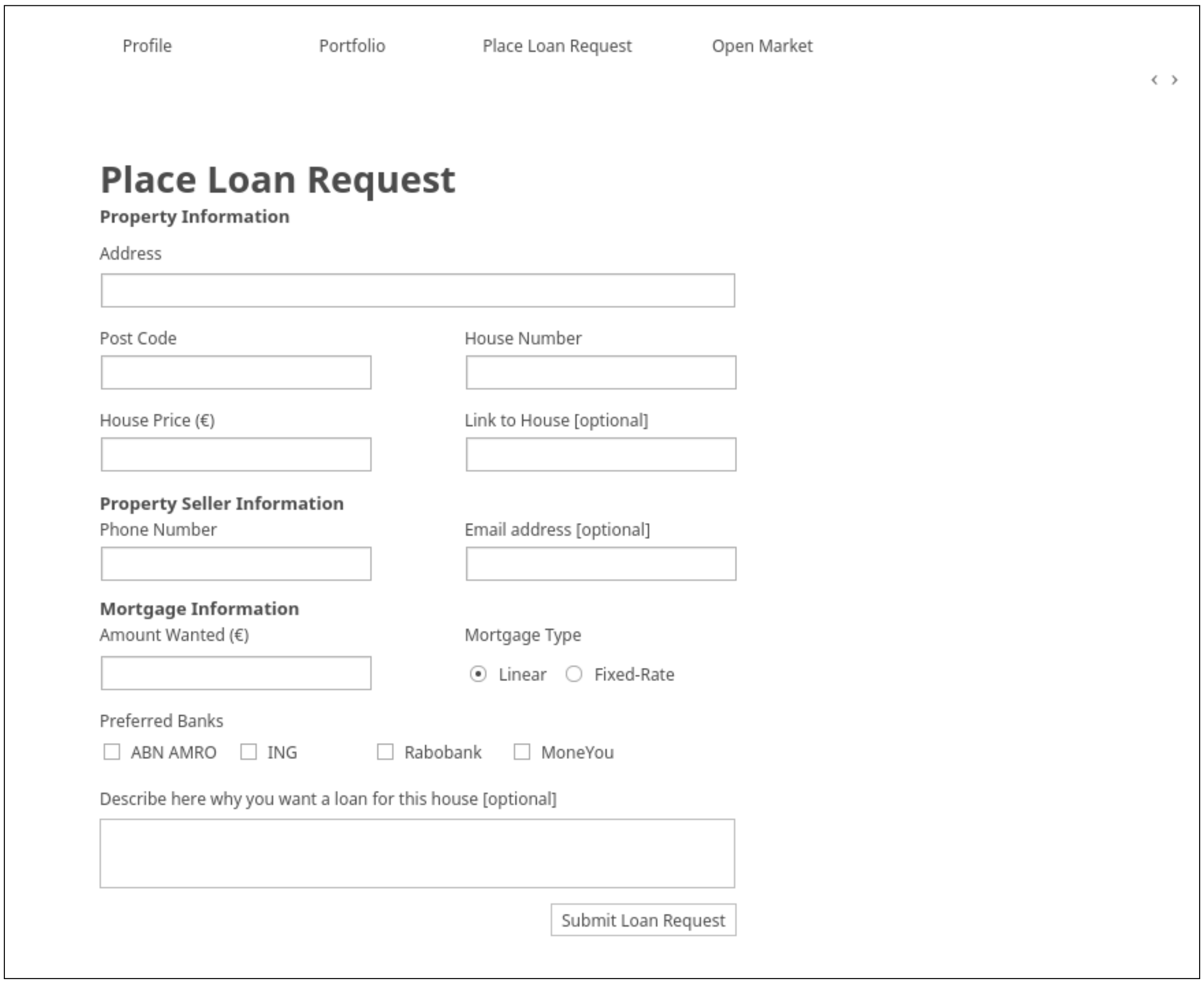
Interest (%)

0.0 - 100.0

Duration (months)

Min - Max

Property Address	Amount Needed (€)	Interest (%)	Duration (months)	Time Remaining (days)



Profile Portfolio Place Loan Request Open Market

‹ ›

Place Loan Request

Property Information

Address

Post Code

House Number

House Price (€)

Link to House [optional]

Property Seller Information

Phone Number

Email address [optional]

Mortgage Information

Amount Wanted (€)

Mortgage Type

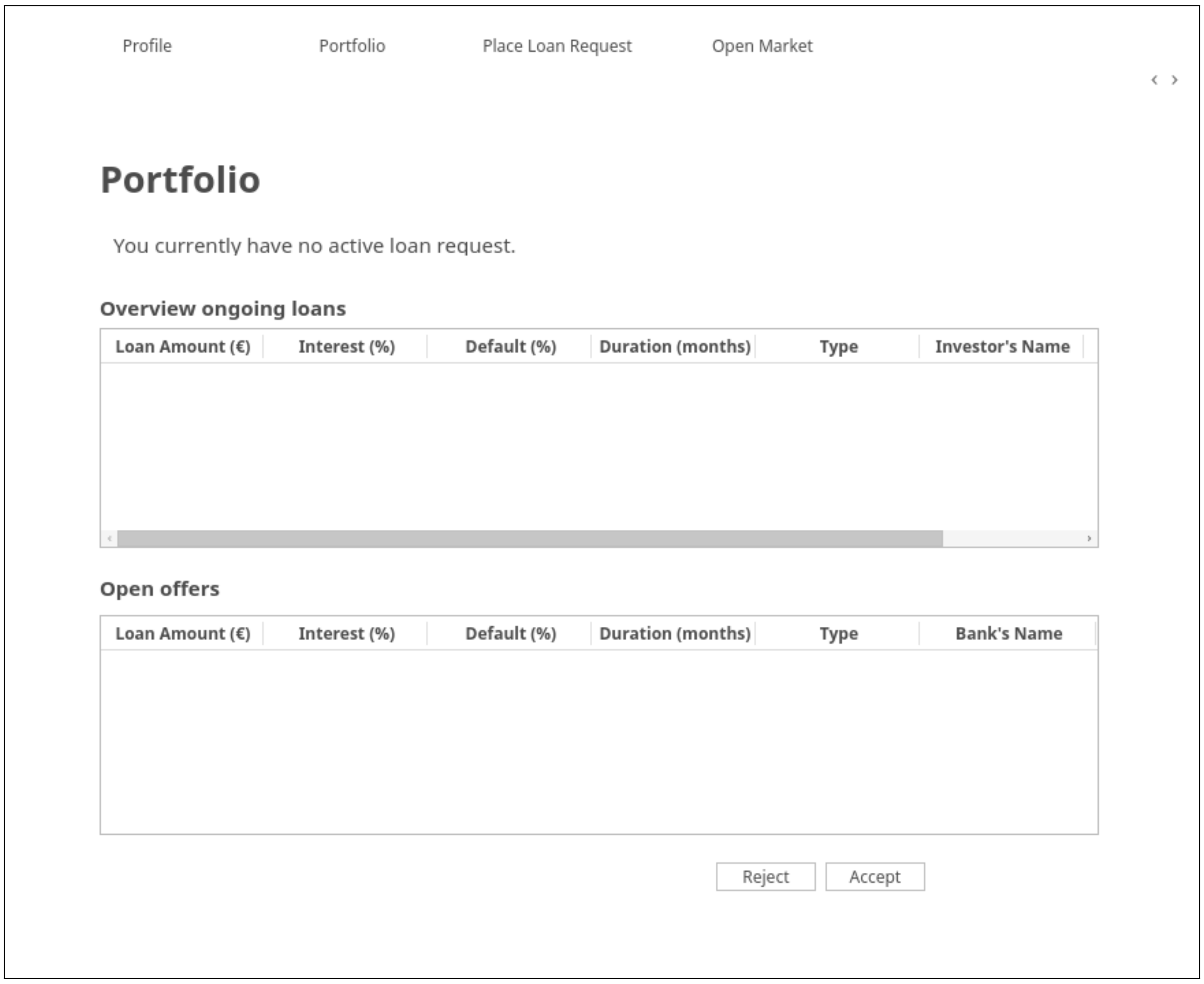
[✓ Linear] Fixed-Rate

Preferred Banks

☐ ABN AMRO ☐ ING ☐ Rabobank ☐ MoneYou

Describe here why you want a loan for this house [optional]

Submit Loan Request



Portfolio

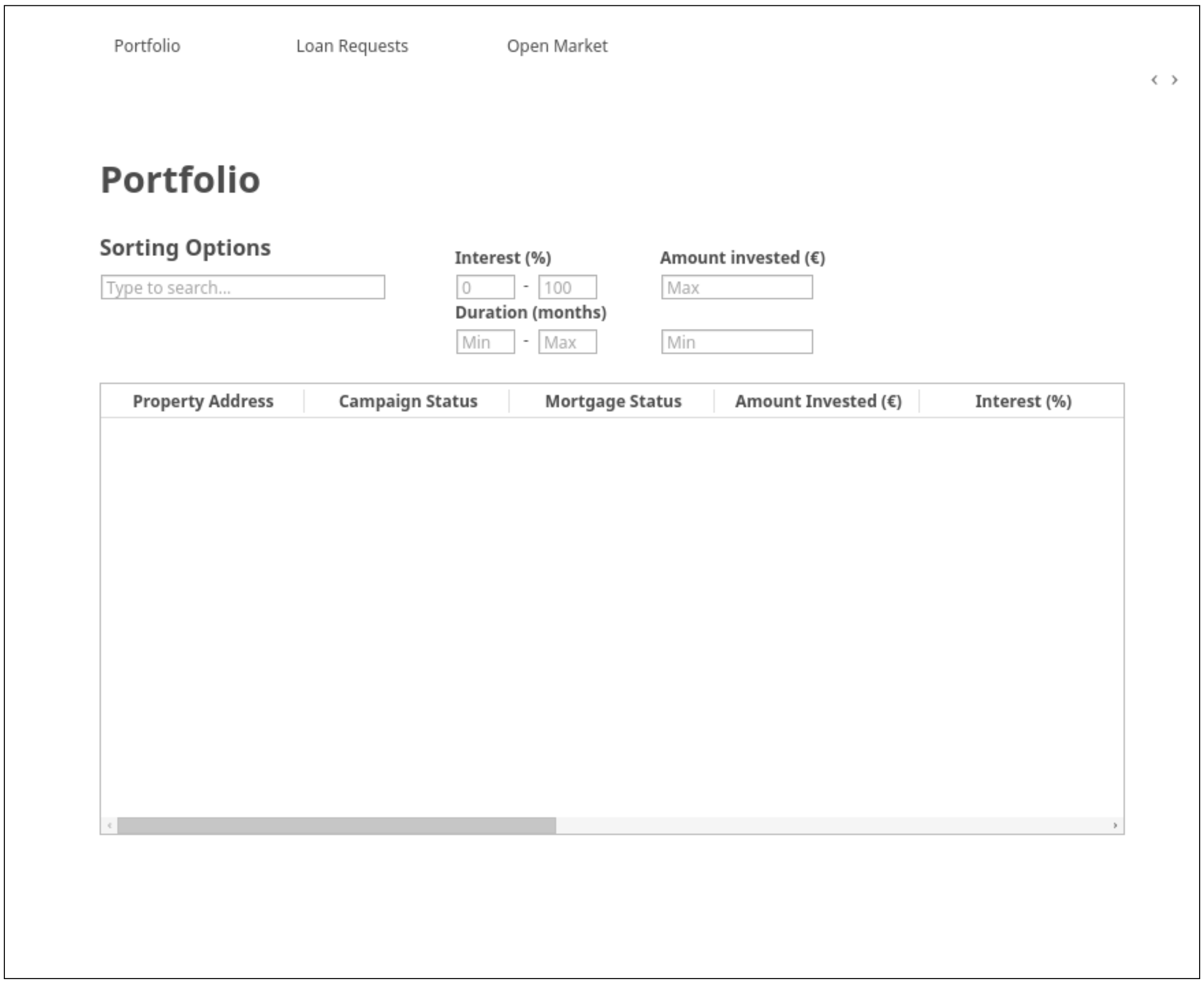
You currently have no active loan request.

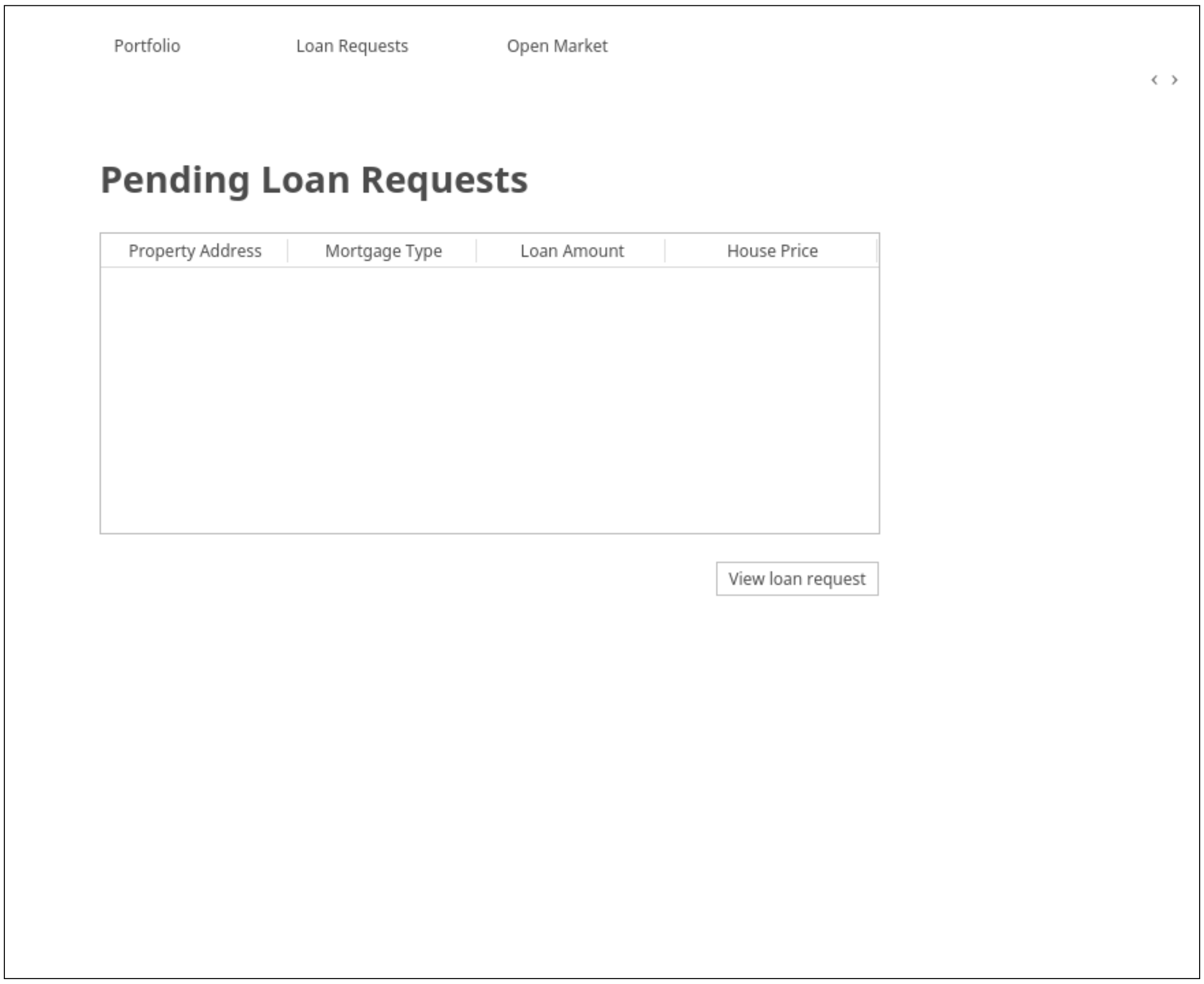
Overview ongoing loans

Loan Amount (€)	Interest (%)	Default (%)	Duration (months)	Type	Investor's Name

Open offers

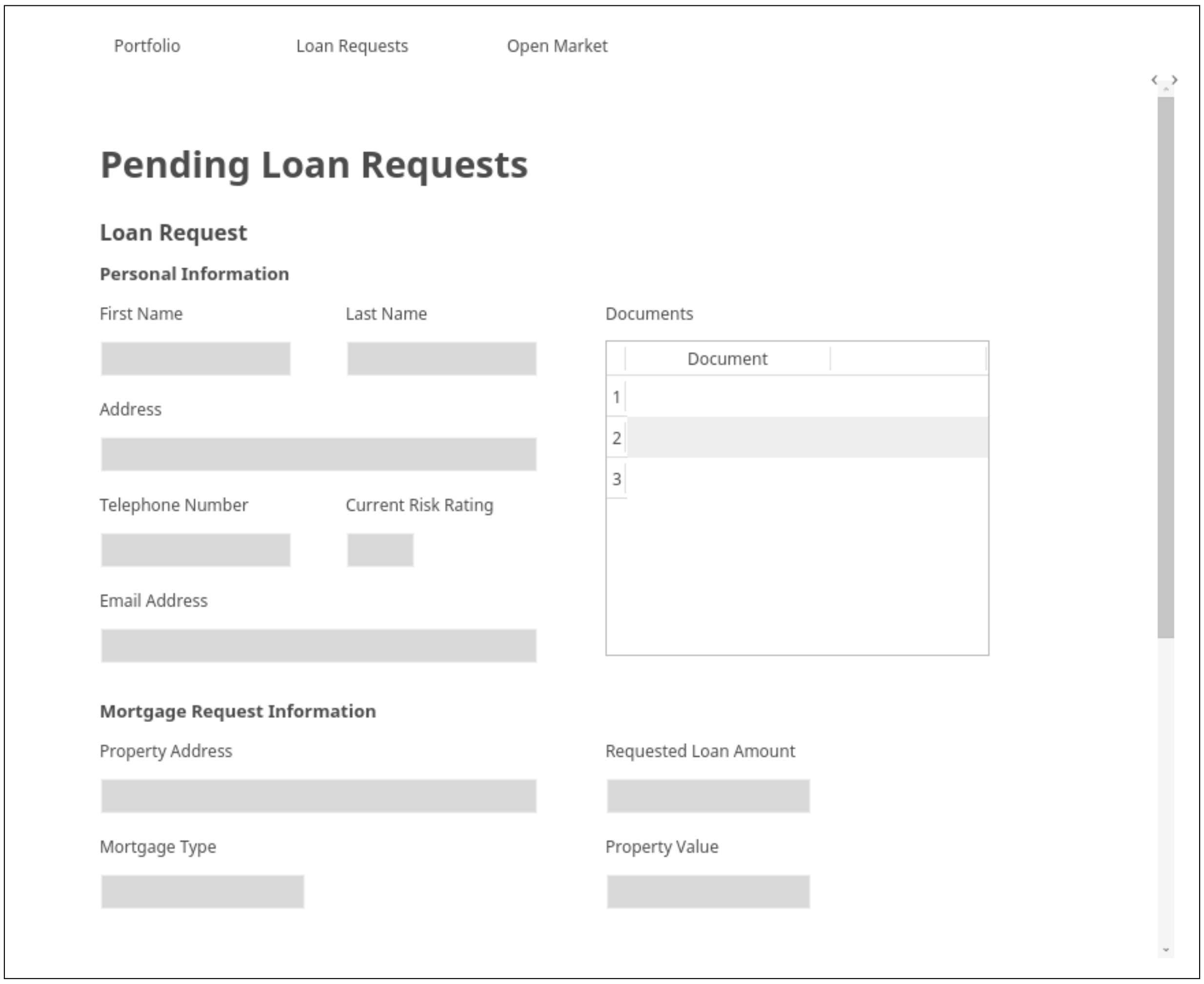
Loan Amount (€)	Interest (%)	Default (%)	Duration (months)	Type	Bank's Name


Portfolio
Loan Requests
Open Market
‹ ›
Portfolio
Sorting Options
Type to search...
Interest (%)
0
-
100
Duration (months)
Min
-
Max
Amount invested (€)
Max
Min
Property Address
Campaign Status
Mortgage Status
Amount Invested (€)
Interest (%)



Pending Loan Requests

Property Address	Mortgage Type	Loan Amount	House Price



Pending Loan Requests

Loan Request

Personal Information

First Name

Last Name

Address

Telephone Number

Current Risk Rating

Email Address

Documents

	Document
1	
2	
3	

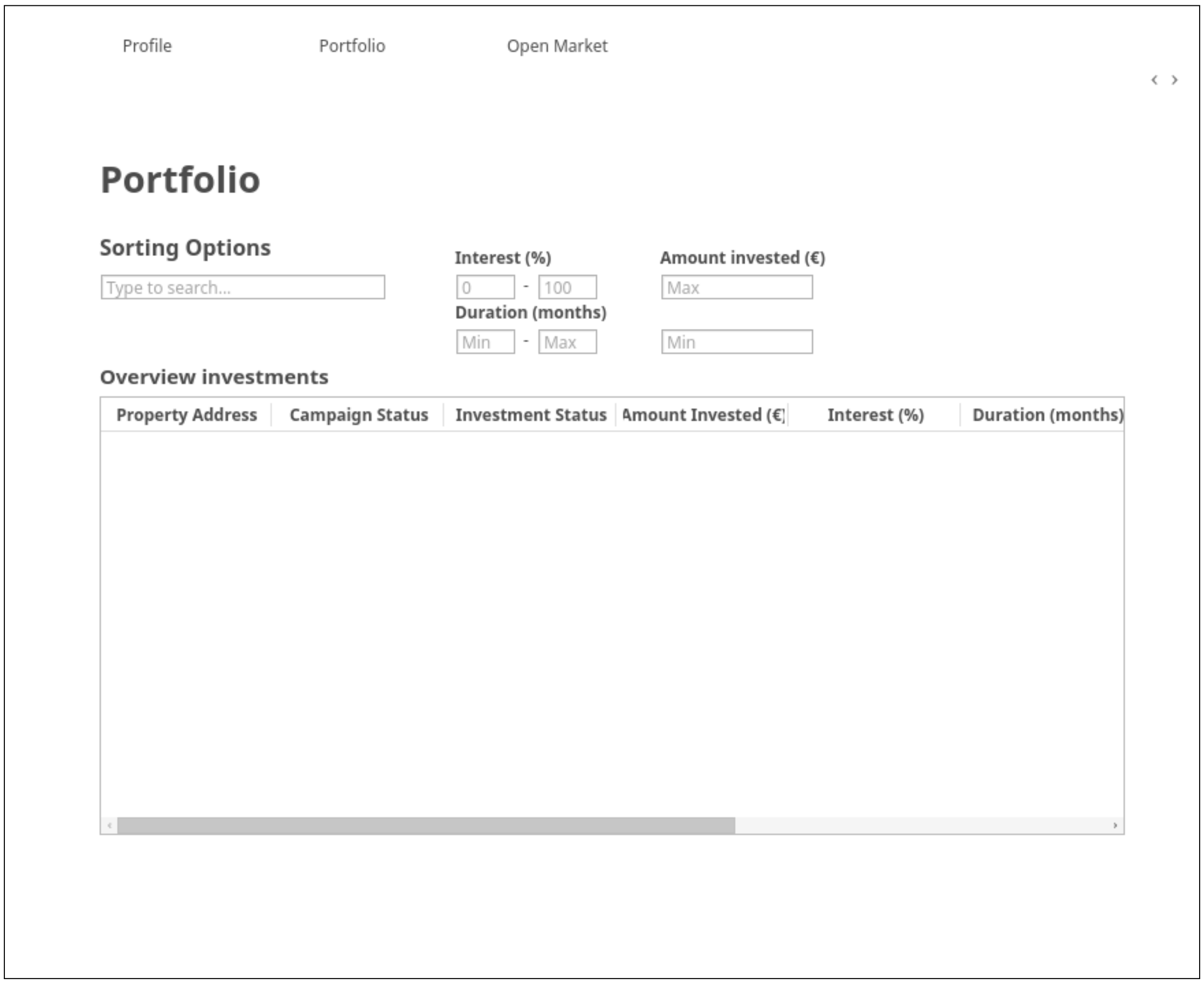
Mortgage Request Information

Property Address

Requested Loan Amount

Mortgage Type

Property Value



Portfolio

Sorting Options

Type to search...

Interest (%)

0 - 100

Duration (months)

Min - Max

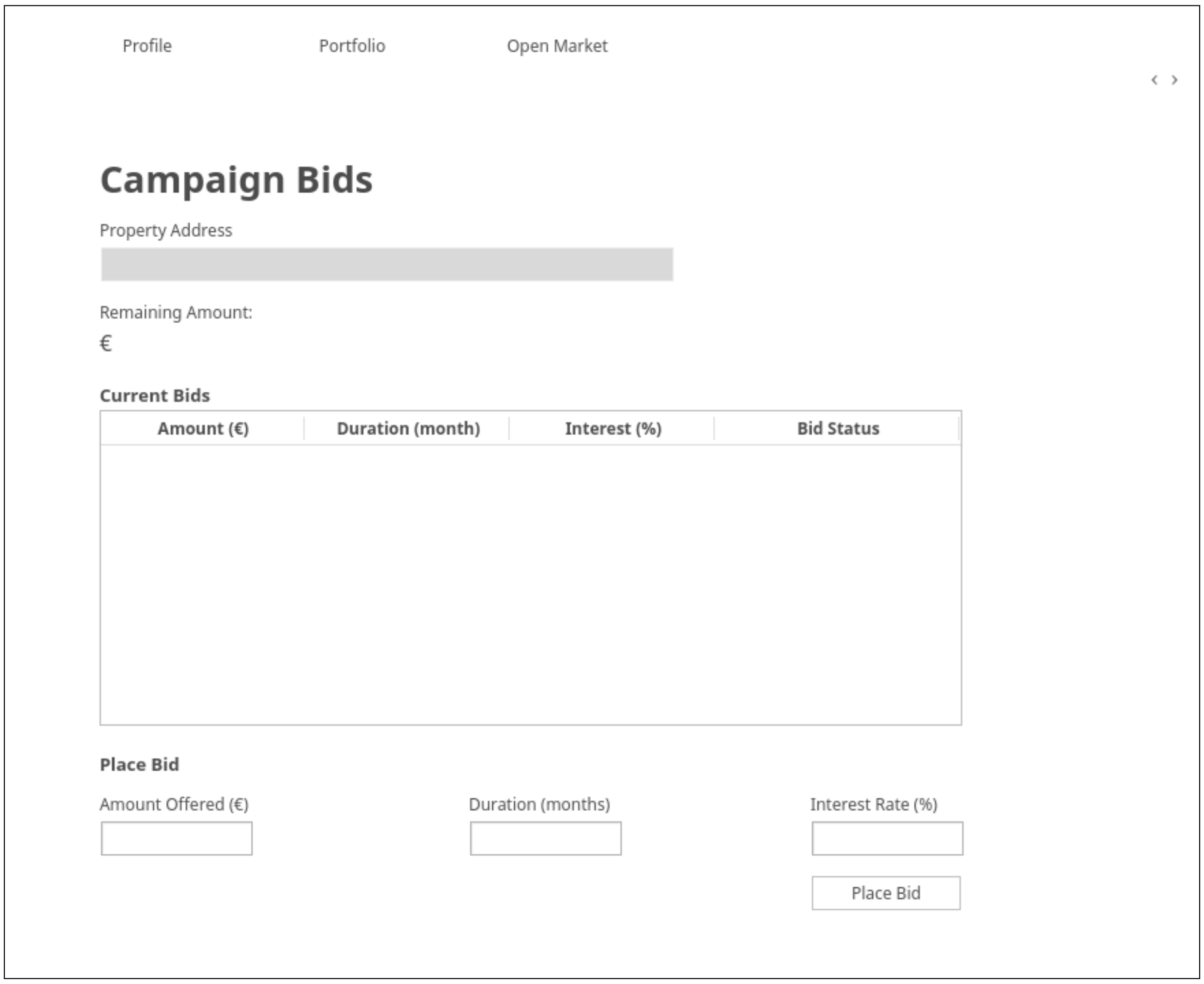
Amount invested (€)

Max

Min

Overview investments

Property Address	Campaign Status	Investment Status	Amount Invested (€	Interest (%)	Duration (months)



Campaign Bids

Property Address

Remaining Amount:

€

Current Bids

Amount (€)	Duration (month)	Interest (%)	Bid Status

Place Bid

Amount Offered (€)	Duration (months)	Interest Rate (%)

F

SIG feedback

We had to submit our code to SIG twice for a code quality review. We sent our code in on December 22nd, 2016 and on January 24th, 2017. Below you can find their feedback.

First feedback

The code of the system Decentralized Mortgage Marketplace scores 4 stars (3.6 out of 5.5) on our maintainability model, which means that the code is maintainable above average (the score ranges between 1 to 5 stars).

Overall, your code scored 5 stars for its small volume, the lack of duplication and its balance of components, 4 stars for its simple units and the loosely coupled modules and 3 stars for component independence. The lowest scores where in unit size and unit interfacing.

Components

Looking at the structure of the project, it looks like you have 4 different components on top level: configuration, test, front end and back end. Diving further on the market folder you have a proper componentization. The 5 stars you received for the component balance is due to the fact that we considered these inner components inside market.

Unit Size

For Unit Size we look at the percentage of code that is above average in length. Breaking down such methods into smaller pieces makes each component easier to understand, test, and is therefore easier to maintain. In the submitted code there are generated files by pyQt5 that they belong to the longer methods in the system, such as, the Ui_MainWindow(object). This unit is not hand written code, it doesn't need to be maintained and it can be created by the source .ui file. It really affected your score and you can avoid committing all of them. Another improvement could be to separate big units with many responsibilities into other reusable units like ProfileController.save_form(self). Comment lines such as "# Get the data from the forms" and "# Check if the user can switch roles" are a good indication that the code following fits in an autonomous unit. Therefore, it is advisable to take a critical look at the longer methods within the system and divide them where possible and avoid committing generated code.

Unit Interfacing

Unit Interfacing is depended on the percentage of code consists of units with an above-average number of parameters. An above-average number of parameters usually indicates a lack of abstraction. In addition, a large number of parameters often leads to confusion in calling the method, and in most cases also to longer and more complex methods. An example of a unit that needs refactoring is the SignedConfirmPayload.Implementation.__init__ which has 13 parameters that could be extracted in a object related (or even named) to beneficiary.

Conclusion

Overall your code scores above average. Hopefully, you can manage to maintain this level for the rest of the development or even reach the 5 stars.

Finally, the presence of 150 (unit) tests that are uploaded is in any case very promising. Hopefully, the volume of the test-code will also grow at the time that new functionality will be added.

Second feedback (Dutch)

In de tweede upload zien we dat zowel het codevolume als de score voor onderhoudbaarheid licht zijn gestegen.

Als we naar de aspecten kijken die bij de analyse van de eerste upload als verbeterpunten werden genoemd, zien we dat jullie bij Unit Size een aantal methodes hebben aangepast waardoor de deelscore iets hoger is geworden. Bij Unit Interfacing zien we minder verbetering. De constructors met grote aantallen parameters zijn nog steeds een probleem, en er zijn sinds de eerste upload ook nog een aantal van dit soort constructors bijgekomen. Dit is een typisch probleem met onderhoudbaarheid op het moment dat de hoeveelheid functionaliteit blijft groeien: jullie datamodel wordt groter, waardoor je objecten meer velden nodig hebben, maar als dit oneindig blijft groeien zit je op een gegeven moment met een onwerkbare situatie. Voor de toekomst is het goed om te kijken naar hoe je die parameters kunt groeperen, en voor die groepen kun je vervolgens weer nieuwe objecten maken.

Tot slot is het positief om te zien dat jullie naast de nieuwe productiecode ook veel nieuwe testcode hebben geschreven.

Uit deze bevindingen kunnen we concluderen dat de aanbevelingen grotendeels zijn meegenomen in het ontwikkeltraject.

G
Acceptance test tasks

Tasks for the borrower

After opening the market application, you are going to fill in your profile as a borrower. Perform the following tasks to create a profile.

1. Identify yourself as borrower.
2. Fill in first name and last name: Alexander Un.
3. Fill in the current address, house number and postcode: Mekelweg 4, 2628 CD.
4. Fill in email and phone number: alexanderun@gmail.com, 0612345678.
5. Fill in IBAN: NL78INGB0009364027.
6. Upload a document.
7. Save profile.

Now you are going to send a loan request to a bank. Navigate to the Place Loan Request page and perform the following tasks.

1. Fill in the address, house number and postcode of the desired house: Rapenburg 4C, 1011TX.
2. Fill in the price of the house: 425000.
3. Fill in the phone number of the seller: 061245679.
4. Fill in the amount wanted: 400000.
5. Select the fixed-rate mortgage type.
6. Select ABN AMRO as the preferred bank.
7. Submit the loan request.

Now that the loan request has been sent to ABN AMRO, all have to do is wait for the loan request to be accepted. Refresh the Portfolio page till you see an offer from the bank. Continue with the following task when the loan request has been accepted and a mortgage offer has been sent to you.

1. Accept the mortgage offer you have just received.

The mortgage you accepted, will now appear in the Ongoing loans table. If you go to the Open Market page, you will see your campaign that has started. Investors are now able to place bids. Go to the Portfolio page and keep refreshing till you get an investment offer. Continue with the following task when you have received two investment offers.

1. Accept an investment offer of your choice.
2. Reject an investment offer of your choice.

The accepted investment will now appear in the Ongoing loans table and the rejected investment will not.

Tasks for the investor

After opening the market application, you are going to fill in your profile as an investor. Perform the following tasks to create a profile.

1. Identify yourself as investor.
2. Fill in first name and last name: Alexis Deux.
3. Fill in email and phone number: alexisdeux@gmail.com, 0612345675.
4. Fill in IBAN: NL78ABNAB0072964837.

5. Save profile.

Now you are going to look for investment opportunities on the Open Market page. Navigate to the Open Market and keep refreshing till a campaign appears. Perform the following tasks after a campaign has appeared.

1. Select the campaign and view the loan bids.
2. Place a bid. Offer an amount of 2000 euros for a duration of 36 months with an interest rate of 1.5%.
3. Place the bid.

Navigate to the Open Market page. And perform the following tasks to place another bid.

1. Select the campaign and view the loan bids.
2. Place a new bid. Offer an amount of 100 euros for a duration of 12 months with an interest rate of 62.5%.
3. Place the bid.

Navigate to the Portfolio page. You will now see the bids you just placed in your overview of investments. When your bid has been accepted, the Investment Status will change from Pending to Accepted.When your bid bas been rejected, it will disappear from the overview. When you navigate back to the Open Market and view the bids on the campaign, you will see which bids are still pending or have been accepted or rejected.

Tasks for the financial institution

After opening the market application you are already identified as a bank, in this case as ABN AMRO. Navigate to the Loan Requests page and keep refreshing till a loan request appears in the table, then perform the following tasks.

1. Select the loan request and view the loan request.
2. View a document. When done viewing, close the document.
3. Make a mortgage offer with the amount offered being 325000 euros, interest rate 2%, default rate 2.5%, and loan duration 360 months.

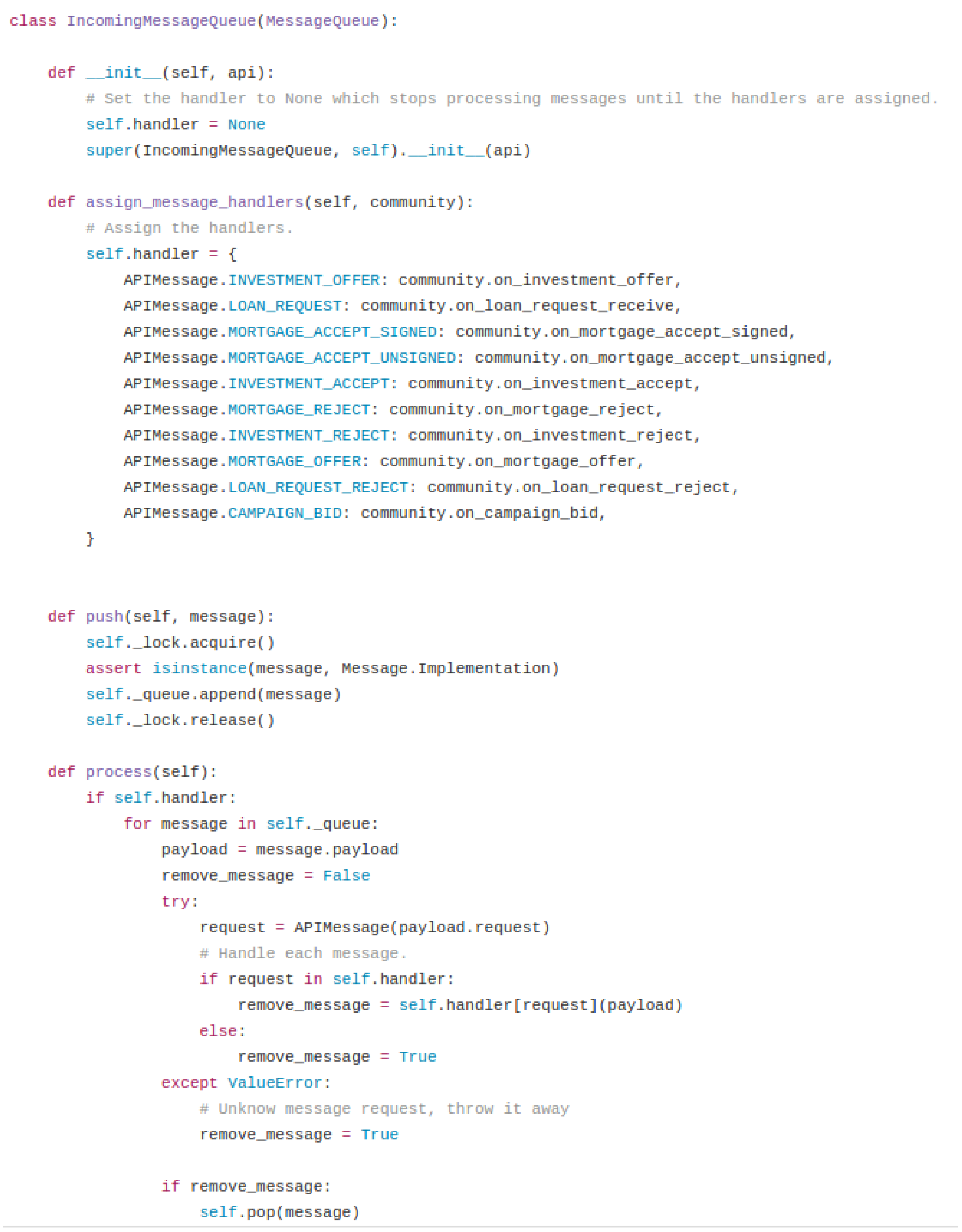
Now that you have accepted the loan request and have sent a mortgage offer, you will have to wait for a response from the borrower. Navigate to the Portfolio page. Here you will see the mortgage offer you have just made, marked as Pending. Keep refreshing till you see that the mortgage offer has been Accepted. Then if you navigate to the Open Market, you will see that a campaign has been started for the borrower to crowdfund the remaining amount. You will also be able to see the loan bids placed on the campaign.

H

Code metrics result

```
class IncomingMessageQueue(MessageQueue):

    def push(self, message):
        self._lock.acquire()
        assert isinstance(message, Message.Implementation)
        self._queue.append(message)
        self._lock.release()

    def process(self):
        community = self._api.community

        for message in self._queue:
            payload = message.payload

            request = payload.request
            remove_message = False
            # Handle each message.
            if request == u"investment_offer":
                remove_message = community.on_investment_offer(payload);
            elif request == u"loan_request":
                remove_message = community.on_loan_request_receive(payload)
            elif request == u"mortgage_accept_signed":
                remove_message = community.on_mortgage_accept_signed(payload)
            elif request == u"mortgage_accept_unsigned":
                remove_message = community.on_mortgage_accept_unsigned(payload)
            elif request == u"investment_accept":
                remove_message = community.on_investment_accept(payload)
            elif request == u"mortgage_reject":
                remove_message = community.on_mortgage_reject(payload)
            elif request == u"investment_reject":
                remove_message = community.on_investment_reject(payload)
            elif request == u"mortgage_offer":
                remove_message = community.on_mortgage_offer(payload)
            elif request == u"loan_request_reject":
                remove_message = community.on_loan_request_reject(payload)
            elif request == u"campaign_bid":
                remove_message = community.on_campaign_bid(payload)
            else:
                # Unknow message request, throw it away
                remove_message = True

            if remove_message:
                self.pop(message)
```

Figure H.1: A problematic function identified using radon before being refactored



```python
class IncomingMessageQueue(MessageQueue):

    def __init__(self, api):
        # Set the handler to None which stops processing messages until the handlers are assigned.
        self.handler = None
        super(IncomingMessageQueue, self).__init__(api)

    def assign_message_handlers(self, community):
        # Assign the handlers.
        self.handler = {
            APIMessage.INVESTMENT_OFFER: community.on_investment_offer,
            APIMessage.LOAN_REQUEST: community.on_loan_request_receive,
            APIMessage.MORTGAGE_ACCEPT_SIGNED: community.on_mortgage_accept_signed,
            APIMessage.MORTGAGE_ACCEPT_UNSIGNED: community.on_mortgage_accept_unsigned,
            APIMessage.INVESTMENT_ACCEPT: community.on_investment_accept,
            APIMessage.MORTGAGE_REJECT: community.on_mortgage_reject,
            APIMessage.INVESTMENT_REJECT: community.on_investment_reject,
            APIMessage.MORTGAGE_OFFER: community.on_mortgage_offer,
            APIMessage.LOAN_REQUEST_REJECT: community.on_loan_request_reject,
            APIMessage.CAMPAIGN_BID: community.on_campaign_bid,
        }


    def push(self, message):
        self._lock.acquire()
        assert isinstance(message, Message.Implementation)
        self._queue.append(message)
        self._lock.release()

    def process(self):
        if self.handler:
            for message in self._queue:
                payload = message.payload
                remove_message = False
                try:
                    request = APIMessage(payload.request)
                    # Handle each message.
                    if request in self.handler:
                        remove_message = self.handler[request](payload)
                    else:
                        remove_message = True
                except ValueError:
                    # Unknow message request, throw it away
                    remove_message = True

                if remove_message:
                    self.pop(message)
```

Figure H.2: IncomingMessageQueue after being refactored

Infosheet

Decentralized mortgage market

Name of the client organization: ABN AMRO
Date of the final presentation: January 31, 2017

Description

For this project we were tasked with creating a scalable decentralized market for crowdfunding mortgages. Our client is ABN AMRO, who is collaborating with the TU Delft Blockchain Lab on the development of blockchain applications.

In the research phase we looked at related initiatives to understand more about crowdfunding real-estate and we had to find out how applying for a mortgage works, as our application involves people who want to apply for a mortgage. We also had to research Dispersy and blockchain as we needed these technologies in our implementation of the product.

We used the scrum framework to keep the process in check and we tried to finish a new functionality every sprint. We held daily meetings within our group and weekly meetings with our coach. During the project we found that the initial scope was too big for this project, so we had to change the project description to fit the project duration.

The product that we developed for this project is a working decentralized platform that can connect users with different goals. Users that are in need of funding to buy a house are able to get a mortgage quote from financial institutions, and they are able to receive investment offers from users that want to invest. Financial institutions can receive mortgage applications from users that want to buy a house, and partially fund those users' mortgages. Users that are looking for an investing opportunity are able to view open crowdfunding campaigns, and they are able to place investment offers on those campaigns.

It was important for us to make the program easily extendable. We have made various recommendations for future work. Due to time constraints and complexity of certain functionalities we were not able to implement all wishes of the client. These functionalities might be implemented by future teams working on this project. At the moment the application is a proof of concept and can not yet be used by consumers. As such, we advise that our program is passed to MSc and PhD students to further develop the market. Our client has indicated that they would like to continue working on our market.

Members of the project team

Name	Katia Asmoredjo	Arthur Hovanesyan
Main contribution	API and GUI	GUI and document transfer
Name	Seu Man To	Calvin Wong Loi Sing
Main contribution	API and blockchain	API, database, and Dispersy

All team members took on the role of developer and contributed to writing the thesis and preparing the final presentation.

Client	**Coach**
S.E. Hagens	Dr. Ir. J.A. Pouwelse
Innovation Center at ABN AMRO	Distributed Systems at Delft University of Technology

Contact

Johan Pouwelse, J.A.Pouwelse@tudelft.nl
Katia Asmoredjo, katia.asmoredjo@gmail.com
Arthur Hovanesyan, a.hovanesyan@gmail.com
Seu Man To, seumanto@gmail.com
Calvin Wong Loi Sing, calvin@idea-factory.nl

The final report for this project can be found at: `http://repository.tudelft.nl`.