UNIVERSITAT POLITÈCNICA DE VALÈNCIA
School of Informatics

Development of a software application to tokenize and manage real estate

End of Degree Project

Bachelor's Degree in Informatics Engineering

Author: Marti Haynes, Eric David

Tutor: Letelier Torres, Patricio Orlando

External cotutor: Cramer Alkjaersig, Jens

Academic Year 2021-2022

Table of content

1. List of figures and tables

2. Abstract

Real estate has been a very important investment sector for a long time, but suffers from a high barrier to entry. This project aims to offer a solution, democratising access to the real estate market through the use of tokenization and blockchain technology. First of all, an in depth analysis of the domain of the problem will be presented. Then the design of the software solution will be considered, including a discussion on the different projectual choices and their impacts. Subsequently, the implementation of the solution will be explored, which is composed of the frontend, backend, and blockchain systems. Regarding the business analysis part, it consists of a business plan including a description of the company, a marketing analysis including environment and market study but also a marketing mix study. Finally, the business plan also includes a budget estimation. Succeeding this, the future of the project will be examined, determining what the next steps should be. Finally, a conclusion will provide some insight into the overall relevance and success of the project.

1 Introduction

Real estate has always been a preferred investment sector for many reasons. First of all, it is a sector of activity that goes back to the Middle Ages with the financing of castles and cathedrals, which is where the word "finance" comes from (finer = to find the means to finish) (Investopedia, 2022). The origin of the word informs about the main problem of the sector: the capital.

Indeed, if the real estate investment sector is one of the most stable and secure, it is not easy to access. Compared to other solutions such as investing in the stock market, banking solutions, and investing in goods such as wine and art, real estate is not easily accessible to the average consumer. This is because real estate is expensive, and even the cheapest properties are too expensive for a lot of people to afford, thus creating a barrier to entry.
The problem to address is the lack of accessibility to the real estate market in terms of capital to acquire a property.
As a team, we are interested in improving the accessibility of this market, while applying cutting-edge technology to lower the financial barrier that bargains small investors to enter and reduce the bureaucratic process to finalise an investment.

The challenges presented in this report can be divided into four areas: business, backend, frontend, and blockchain.
For the backend part we will focus on how to safely store and retrieve the data: we will present which solution we chose as well as the data structure. We will also present how the backend interacts with the frontend and the smart contracts. Regarding the frontend we will illustrate how the information is presented to the user and how we implemented the interaction with 3rd party applications. The blockchain section will focus on the functionalities implemented by each smart contract, how they have been tested and deployed.

For the business part we will focus on producing a business plan covering marketing and financial areas. The challenge is to give a clear overview of the company, the product and the different strategy in terms of marketing. Also, the second business challenge is to realise a realist and safe budget estimation.

1.1 Project delimitations

This report will not cover the following delimitations:

- Legal & Contracts: It will be assumed that the contract verifying system will be implemented in the future and will use a placeholder. This is because we need a fully dedicated legal team, and our capabilities and expertise in this field are very limited. We also think it will not add much to the prototype and this time will be better invested elsewhere.

- Helpdesk & faq: We assume that as far as the project is in a prototype phase we don't need any Helpdesk or FAQ section on the app.

- Deployment: We will make the prototype run on our local environment but will not deploy the app on the cloud. The smart contracts will not be deployed on the mainnet since it takes a small but considerable amount of money and time. They will be deployed to a testnet.

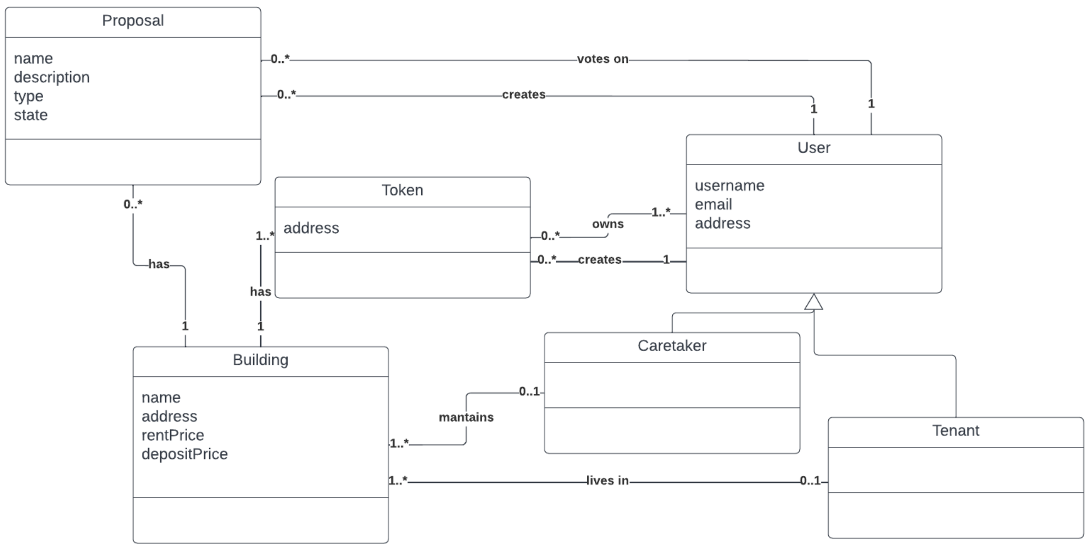
2 Software study

2.1 Analysis

The goal of this project is to lower the barrier to entry for the participants of the real estate market. In order to achieve this, the proposed solution allows for users to tokenize buildings into many parts, allowing other users to own and trade them.

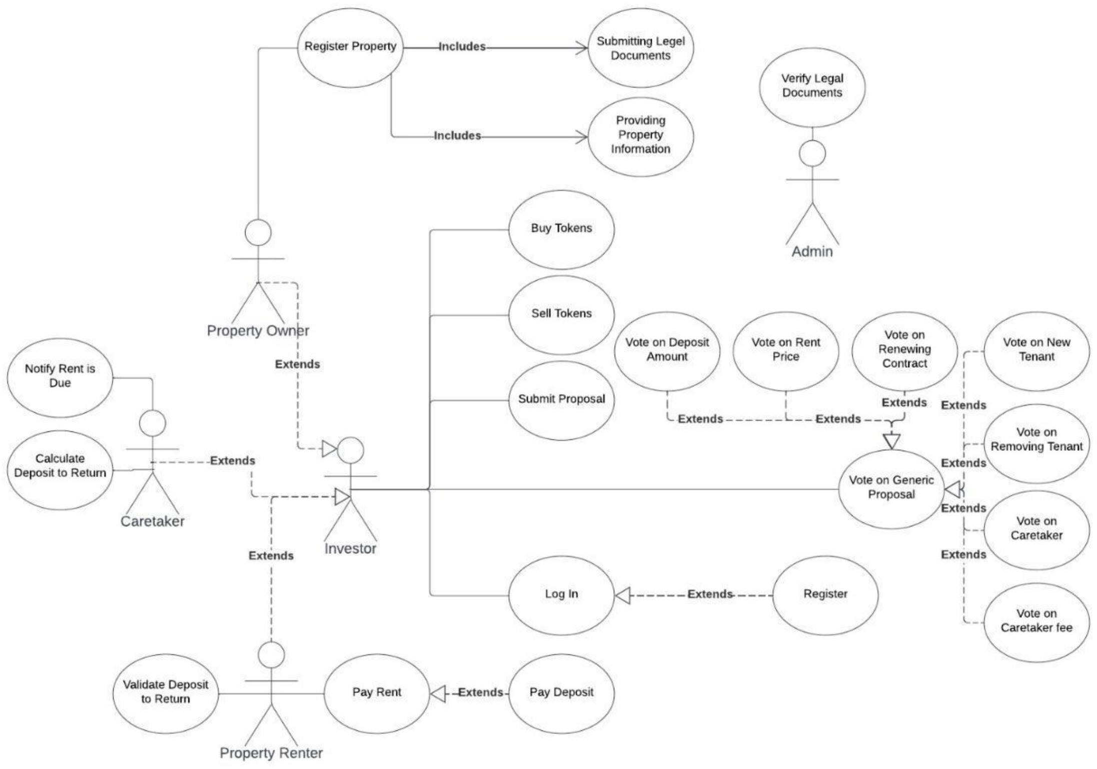
The domain regarding the solution is better understood with a domain model diagram, as shown below:

Figure 1: domain model



The main elements of the domain are users, buildings, and tokens. As for their interactions, a building has many tokens that serve to represent it. These tokens are created by a user, which at the same time can own many of them. There are also some special kinds of users, which are the caretaker and tenant, and they either maintain or live in the building respectively. Finally there are the proposals that a building has, and that can be created and voted on by users.

Figure 2: Use Case Diagram



The use case diagram is also very useful when thinking about how the different actors can interact with the proposed system. It allows an analyst to determine which functionality is shared, while also determining which functionality might have a higher priority to be implemented.

The aim of the project is to build a working prototype that includes a functional and elegant interface, full backend and database, and tested smart contracts. This will all be deployed in a local environment in the case of the app logic, and in one of the ethereum testnets (Rinkeby) in the case of the smart contracts, but will be functional.

2.1.1 Functional requirements

The following user stories will be implemented in this project:

Table 1: Functional Requirements

As a	I want	so that
User	to be able to login into the application without using a password	I can have an easy way to access the platform
User	to be able to visualise my data, tokens that I own, the buildings that I listed on the platform	I can be aware of my status on the platform and on the blockchain
User	visualise all the buildings listed and the information about them	I can make an informed purchase of tokens
User	list a building on the platform	I can create tokens to sell in the future to make profit
User	set a price for the tokens I own	I can sell my tokens on the blockchain
User	buy tokens related to a building	I get voting rights, part of the profits and privileges for that building.
Tenant	be able to pay the monthly rent	keep my tenancy status
Tenant	be able to see the deposit proposal after a rent period	be able to decide if accept it or not
Tenant	be able to accept or decline a deposit proposal after the rent period is over	to get my deposit back or get the proposal mediated by an external actor if I feel it is unfair

User	submit a proposal for a building	I can change the status of a building in terms of rent, tenant, caretaker or generic variables
User	vote for a proposal	I can decide if to change or not the status of a building
User	get all the existing proposals	I can see what changes have been done on a building since it has been listed and what are the proposed changes
Tenant	know who is the caretaker of the building that I rented	I can be supported when I need help during my tenancy
Caretaker	request the rent from the users every month and submit a deposit proposal after the rent period is over	I can be compensated for my duties

Also all the buildings listed in the application should follow the local rental laws and be verified by a dedicated legal team before the user can create tokens out of them. The application needs to be decentralised so that the users can access and interact with the core logic even if the backend and the app client are not working.

2.1.2 Non-functional requirements

The application also has the following non-functional requirements expressed in the SMART format

Table 2: Non-Functional Requirements

Specific	Measurable	Achievable	Relevant	Time-bound
The app will have a solid backend and frontend architecture	at least 95% uptime	Implement a backend architecture that is fault-tolerant and handles the errors without crashing. Deploy the application on a cloud infrastructure to balance the number of requests that the backend gets.	The platform can easily scale and manage more users over time.	3 months

2.2 Design

The system can be divided in three parts that interact with each other: the server, the frontend client, and the contracts.
The use of a hosted backend, database, and frontend makes it a semi-decentralized application.

The key difference from a centralised application is the resilience given by the smart contracts: by using scripts that are deployed and running on the blockchain it can be ensured that the core functionalities of the app will work even if the backend or the frontend are not running.

On the other end it differs from a centralised application because of the following factors:

- The frontend and the backend do not run on the blockchain but they interact with it
- The backend is used to interact with third-party services, such as mongo DB in this case. This is because modern applications cannot live in an isolated environment
- The backend holds all the business logic apart from the smart contracts.

2.2.1 Server

The server is based on a middleware architecture. A middleware is a layer of software that allows transmission of information between application and services. In this case, web middlewares are being used: the requests from the client go through a chain of middleware functions that handle the request and send a response to the client. The response could either contain the data that the client requested or an error.
To achieve this, multiple frameworks and languages were analysed, and it was chosen to implement the server using Express.

Express is a middleware-based framework for Node js that allows writing powerful APIs in javascript, it has many pre-built middlewares but it also allows developers to write custom middleware or to import third party middlewares.

An alternative approach would have been to use Django, a web development framework for python. Django also has a broad support for blockchain interactions and allows faster development since it has many pre-built functionalities. It is more scalable and secure.
On the other hand it has a slower evolution than express and it is not suitable for handling a large amount of requests.

Another tech backend framework that was considered was Hapi js. It is a plugin-based javascript framework for node js. The main advantage of this framework is that it has many standardised templates for testing, development and deployment that are ready to use: it gives the developers a boilerplate to build their own application. The main disadvantage is the plugin based logic that would have made the interaction with the frontend and the blockchain in this specific use case.

Express offers a greater customization, more efficient error handling and can manage a large amount of requests. It is easy to use and to install. It offers a good boilerplate to start building a web application. The middleware logic is used to easily interact with the frontend and the blockchain.

2.2.1.1 Middleware used

- **Authentication:** this middleware is implemented by the passport library. It is used to protect the routes that require user authorization.
- **Admin check**: Additional layer of security for the routes that require admin authorization
- **Building approval check**: An additional layer of security that requires a building to be verified and approved by an admin before performing actions on it
- **Caretaker check:** checks if a user is the caretaker of a building. This security layer ensures that only the designated caretaker of a building can perform certain actions.
- **Tenancy check:** checks if a user is the tenant of a building. This security layer ensures that only the designated tenant of a building can perform certain actions.
- **Balance check**: checks the balance that the user possesses for each token.
- **Controller handlers:** final software layer of the request handling process, performs operations and interacts with the database then formats the response and sends it back to the client.

2.2.2 Database

The server uses a NoSQL database for data storage. A NoSQL database is non-relational and often used in real-time web applications. It provides flexibility, scalability, high-performance, and highly functional APIs.

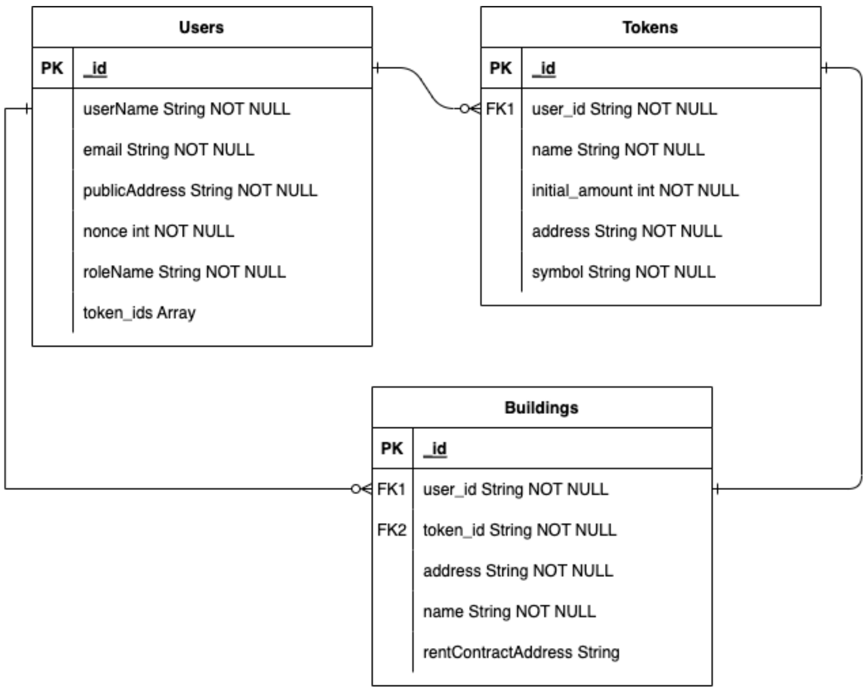
Adopting a non-relational database allows handling and storing large amount of data with minimum structure, it guarantees great performance and scalability at a reduced cost.
It was a goal to be able to easily share the data among the developer team, to achieve this goal a NoSQL database with a hosted web provider was needed.

Our choice was to adopt MongoDB as a database provider. Since it is often used together with Node js and Express it has an easy integration with the server. It is open source and document based which means that the information is stored in flexible documents instead of tables.

A document typically stores information about one object and any of its related metadata.
Documents store data in field-value pairs. The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects.

The ER schema is the one that follows.

Figure 3: ER Schema



2.2.2.1 Users

Users models the actors that interact with the platform.

- Each user is uniquely identified by the field _id.
- A user also has a unique email and username, these are used to send user notification by email.
- The public address is the field that identifies the user on the blockchain and makes the interaction with the contract possible.
- The field nonce is a random number mostly used for the authentication to prove that the user is the owner of the address
- The roleName is used to distinguish admins from normal users
- A User has an array of tokens they possess. This field implements the many-to-many relationship with the tokens table since it is an array of the unique primary keys of the tokens. A token id appears in the array only if the balance of the token returned from the blockchain call is greater than 0.

2.2.2.2 Tokens

A token models an instance of an ERC20 token deployed on the blockchain by a user from the platform.

- Each token is identified by a unique field called _id
- The field user_id implements the one-to-many relationship between users and tokens. A user can in fact create more than one token, one for each building.
- The name, initial amount and symbol represent the name of the token, the initial supply on the blockchain and the symbol of the token o the blockchain
- A token as a unique address that identifies it on the blockchain and makes the interaction with the smart contract possible.

The token table and the users table are related in two ways:

- many-to-many: many users can possess many tokens. A token is possessed from a user if the user balance of that token is greater than 0. Since users can trade among them a quantity of tokens that is less or equal than the total supply, a user can end up having a positive balance of more tokens, and many users can have a positive balance of the same token.
- one-to-many: a user can create many tokens, one for each building they wish to list on the platform.

2.2.2.3 Buildings

A building models a construction in the real world possessed by a user.

- Each building is uniquely identified by the field _id.
- The building is associated with the token by the field token_id. Buildings and Tokens are in a one-to-one relationship
- Name and address identify the building in the real world.
- The rentContractAddress stores the address of the rent contract associated with the building in the blockchain.

2.2.3 Interactions

There are three main interactions that occur in the application: frontend with backend, backend with contract, frontend with metamask.

The frontend interacts with the backend through a REST API protocol. An API, also known as application programming interface, is a set of definitions and protocols to integrate software communication. It can be imagined as a contract between the information provider (the server) and the information user (the client).
The contract states what information the server needs from the client and what data is given back to the client.

REST is a set of architectural constraints that can be implemented in a variety of ways. When a client makes a request to the server through a REST API, the server transfers the state of the resource it is managing to the client in a standard format through the HTTP protocol.

REST can be implemented by having a client-server architecture of a mobile app client and a Node js server that exchanges data in Json format. The communication between client and server is stateless, meaning that no client information is stored between two different get requests, there is no stored knowledge between two past transactions.

The routes are grouped by resource and there are three resources that the client can manage: users, tokens, buildings.

The backend interacts with the contracts through the Web3 js API. There are two kinds of interactions: the ones that modify the state of a contract and the ones that don’t. For the ones that change the state of the contract the Web3js creates a transaction object and gives back to the server the encoded ABI of the transaction. The ABI is an interface between the client that is going to interact with the contract and the bytecode stored in the blockchain.

When a contract is deployed its bytecode is stored in the blockchain but for high level programming languages to execute its functions the server needs to translate arguments and names into bytes representation. The response returned from the contract needs to be translated from bytecode into the tuple of return values defined in higher-level languages.

Languages that compile for the Ethereum Virtual Machine maintain strict conventions about these conversions, but in order to perform them, one must know the precise names and types associated with the operations. The ABI documents these names and types precisely: it defines the methods and structures used to interact with the binary contract, it tells the caller to encode the needed information in a format the EVM can understand, just like API does with the client server interaction but on a lower-level. It can be thought of as an adapter between the bytecode and the high level programming language. The backend returns the ABI to the client together with the nonce. The nonce (number only used once) is the number of transactions sent from a given address so that it can be determined in what order the pending transactions are executed.

For the interactions that don't change the state of the blockchain no transaction is generated thus the backend can directly call the function of the contract and get the result without having to go through the signing process.

The frontend interacts with metamask each time a transaction needs to be signed. After the transaction information is received in the frontend, it is used to create a new transaction. A transaction is composed of the following elements:

- from
- to
- value
- data
- gasLimit
- maxFeePerGas
- maxPriorityFeePerGas
- nonce

The *from* field is the address of the user sending the transaction. The *to* field is the address receiving the transaction. In the case of deploying a contract, the value is set to the null address (0x0000000000000000000000000000000000000000). The *value* field is the amount of ether to send with the transaction, specified in wei (wei is the smallest unit of ether, one ether equals 10^{18} wei). The *data* field is used for encoding information. Usually it is used to encode function calls to a contract, or to encode a contract in order to deploy it. It is encoded in hexadecimal format. The *gasLimit, maxFeePerGas,* and *maxPriorityFeePerGas* are fields related to the gas of the transaction. Gas is related to the fees paid to the miners/validators of the blockchain in order to confirm a transaction. The *nonce* field as described previously describes the number of the transaction

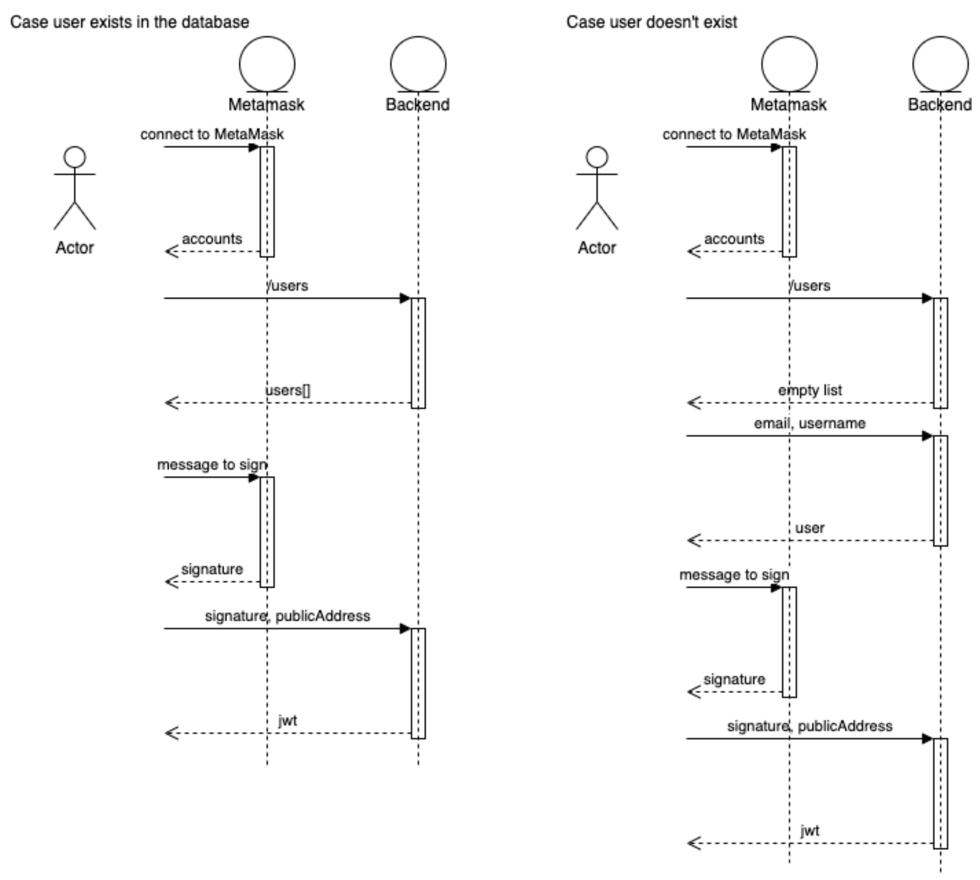
All this data is encoded into a single transaction, which is then sent to Metamask. The user is then able to sign this transaction, which is then submitted to the blockchain by Metamask, making use of a node provider (in this case Metamask uses the node provider Infura). The transaction is then confirmed by miners/validators. After the transaction is submitted, Metamask returns the transaction hash to the frontend, which is then sent to the backend. This transaction hash can then be used to get information on the transaction.

2.2.3.1 Authentication

The login process involves the backend, the database, the frontend, and the wallet provider in order to generate a JWT token for the users to authenticate themselves in all the requests. JWT (Json Web Token) is an open standard that defines a compact and self-contained way to securely transmit information between parties as a JSON object. Each JWT token has a header that holds the encryption algorithm and the token type. The payload is a JSON object holding the desired information. The last part is the encrypted signature. The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way. To create the signature, the Base64-encoded header and payload are taken, along with a secret, and signed with the algorithm specified in the header.

The whole authentication process is explained visually by the following interaction diagram.

Figure 4: Authentication Interaction Diagram

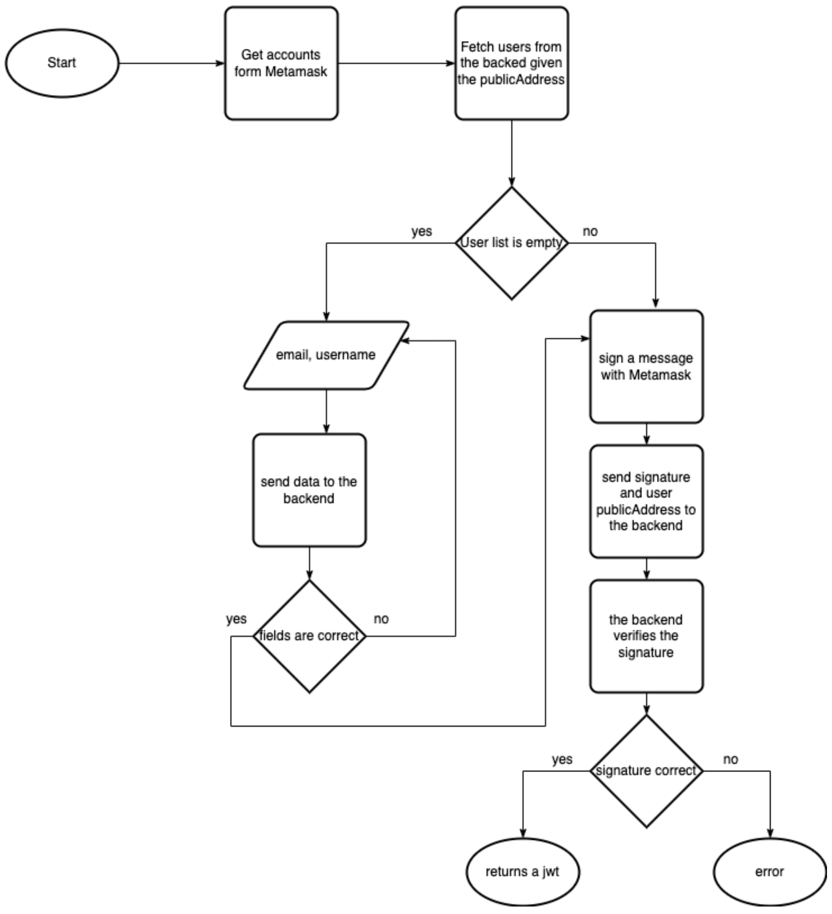


The client application connects to the wallet provider which in this case is Metamask. Metamask gives back to the client a list of accounts. The client takes the first account and passes it to the backend. The backend queries the database and returns a list of users that have that public address. If the list is empty the client prompts the user to insert a nickname and an email and passes those fields to the backend. The backend then creates a new user in the database and returns the user to the client application. The client makes the user sign a unique message through the wallet provider that proves that They are the owner of that public address. The message and signature then are passed to the backend together with the public address of the user that signed the message. The backend retrieves the public address passing the signature and the message to the web3 API. If the given public address and the one retrieved from web3 are the same then the backend issues a new JWT containing the user id as a payload.

If the user does exist then the client skips the call to the create user route and directly prompts the user to sign the message on the wallet provider.

All the process is better explained by the following flux diagram.

Figure 5: Authentication Flux Diagram


Start
Get accounts form Metamask
Fetch users from the backed given the publicAddress
yes
User list is empty
no
email, username
sign a message with Metamask
send data to the backend
send signature and user publicAddress to the backend
yes
fields are correct
no
the backend verifies the signature
yes
signature correct
no
returns a jwt
error

2.2.4 Contracts

Smart contracts are computer programs that live on the blockchain, and can execute certain logic on command. This simple concept opens up lots of possibilities, due the openness and decentralisation of public blockchains, like ethereum. In this case this is perfect for the main logic of this system, so that any user can interact and make use of the platform in an open, secure, and trustless environment. In this case there are 3 main contracts that will serve different functions:

- NewToken.sol
- BuySell.sol
- Rent.sol

It is important to note that for these purposes, only one BuySell contract will exist, that will manage all buying and selling of all the different tokens that will exist on the platform. Meanwhile, a new version of NewToken and Rent will be deployed every time a new building is tokenized, creating a specific token for that building, the system for managing rent for that building, and the system for governance for that building.

2.2.4.1 NewToken

This contract serves a dual purpose. It serves as a token creator, but at the same time, it also serves as a platform for the governance of the buildings. Normally having a contract that has more than one responsibility can be a bad practice, but in this case it makes sense, as will be explained further on. For the token creation, the contract gets most of its logic from the ERC20.sol contract, a widely used library and standard for tokens developed by Openzeppelin. By extending this contract not only does NewToken inherit lots of useful token functionalities, such as transferring, minting, and more, but it also makes the tokens compatible with many blockchain protocols. This opens up lots of possibilities, as it allows for users to create derivative products using their tokens, or for them to use them in already existing applications. For instance, a user who wants to keep their tokens for a long time but wants additional profit could participate in an external lending pool, only needing to transfer their tokens. This kind of composability is one of the promising aspects of Decentralized Finance, and often why it is sometimes referred to as “Money Legos”. Another reason to use the

ERC20 standard is that it has been audited and has been “battle tested” for many years.

In addition, NewToken contract also extends ERC20Votes.sol, another contract by Openzeppelin. This contract is an extension in itself of ERC20, and it basically allows for the tracking of votes using tokens. This is the main reason why the contract serves this dual function, as having one vote equals one token simplifies the architecture and reduces complexity. That being said, one might wonder why this extension is needed at all, being that one could simply query the balance of tokens a user has and then use that number as the number of votes. This might seem like a simple and efficient solution, but it has great security flaws, mainly regarding what is commonly called the “double spend/vote problem”. Let's imagine a situation in which the balance of a user is used in such a way to vote on a proposal. Even if the system checks that a user can’t vote twice, nothing stops them from transferring the tokens to a new account and voting again from the new account. To solve this issue, the ERC20Votes extension provides a way of tracking votes historically. That way, a snapshot of balances is taken for the block number the proposal was created, and then each user gets the corresponding number of votes. It doesn't matter if the tokens are transferred after the proposal was created, as the balance in that instant is what counts. This way there is a secure and reliable governance system for voting on the proposals submitted by users. There is also a way of checking historical balances, something that is also useful for the Rent contract that will be explained later.

In addition to extending these two libraries, the NewToken contract also has some custom logic. The first part is the minting of tokens, as it is only allowed during the creation of the contract, and never again. The second is the voting for proposals, allowing users to submit a proposal, and also to vote on existing proposals. How the system works is that proposals start with 0 votes, and if they get more than 50% of the votes, then they are accepted. A proposal being accepted has different meanings, but they can be separated into two types. The first type would be a generic proposal, where its outcome has an impact on the token holders but not directly on the blockchain. An example of this would be a proposal would be “Proposal to paint the walls green”. This proposal being approved means that some organisation will be required outside of the platform to carry it out (of course it is not possible to paint a room through the blockchain), but the proposal itself and its outcome will be registered on the blockchain. The second type of proposal can address particular issues specifically, which can be chosen when creating it, like for example renewing the tenant's contract for additional months, or changing the price of rent. This type of proposal is different because once it is accepted, it will automatically interact with the Rent contract in order to achieve the desired effect, without any additional input from users. The third custom element of this contract is the deployment of the Rent contract. As both these contracts need to communicate, there is the issue of how they will know the address of the other contract once they are deployed. Of course this could be done manually, but it would introduce complexity, time, and is prone to human error. Another more complex but sometimes used solution is to use special techniques in order to precompute the address that a certain contract will be deployed at. This would have worked, but a simpler solution was decided, which was to have the NewToken contract itself deploy the Rent contract once it was deployed. This way the NewToken contract can store the address of the Rent contract, and can interact with it without issue. This is also an advantage to the users, as they are able to tokenize a building in one single transaction, making it easier and less confusing, while preventing a situation where one contract was deployed but the other one was not.

2.2.4.2 BuySell

This contract hosts all the logic of buying and selling tokens. This is a very important functionality, as users need to have a way to trade their tokens. This way the users who tokenize buildings can sell part or all of them, while small investors can buy small quantities. For the implementation of these trades, there are two main possibilities. The first one is an order book system, one commonly used in traditional finance. The second one is the liquidity pool system, one that has been gaining popularity recently, especially with DeFi protocols such as Uniswap. They both have their advantages and disadvantages.

Order book systems require a system for storing buy or sell orders, or in some cases both. How the system works is that a user can submit a price at which they want to sell their tokens, and create an order. Then, when someone is willing to buy the tokens at that price, they can fill the order, paying the price and receiving the tokens in exchange. This approach has been used for centuries, and is simple to implement. The main disadvantage is that an order book system cannot be instantaneous for both buyers and sellers, only for one of the two parties at best. If someone puts tokens on sale, they need to wait until a willing buyer appears. This problem is often mitigated by having lots of users and lots of trades.

On the other hand liquidity pool systems do not require an order book, but instead rely on a totally different mechanism. A "pool" is created, where the tokens being traded are deposited. Then users can trade these tokens simply by depositing one of them, and withdrawing the other one. The price of the tokens relative to each other is calculated using the ratio of the amount of tokens in the liquidity pool, so each time a user swaps one token for another, the price can be affected. The amount of tokens deposited in the pool also affects the price, as a pool with high liquidity will react a lot less to an individual trade than a small pool. This system has the advantage over order books in that both buyers and sellers can trade instantly at market price. That being said, the main disadvantage is that the pool must have liquidity in order to function. This is normally achieved by allowing users to deposit their own assets into the pool, and earn proportional fees from all the trades in the pool. Still, there is the drawback that someone has to provide liquidity, and that can be complicated for markets with a small volume of tokens being traded.

Having measured both pros and cons, the BuySell contract implements an order book system. Building a liquidity pool system from scratch would have been too complex and out of the scope of this project, and even if an already existing one was implemented, there is the huge downside that someone would have to provide liquidity for every building that is tokenized in two currencies, the building tokens and ether. For this reason a simple order book system makes the user experience simpler, and fits the needs of the problem better. It is also a good compromise, as advanced users can create their own liquidity pools for their tokens using existing solutions like Uniswap.

The BuySell contract order book system is implemented only for the selling of tokens. This makes sense for reducing complexity and simplifying the process of users buying the token, making it an instant process. A user can put their tokens on sale, and select a price at which they want to sell them. Then users have the option to buy the tokens with the cheapest price directly, filling that order and replacing it with the next cheapest set of tokens on sale. There is also the option for sellers to withdraw their tokens, in case they do not want to sell them anymore or wish to set a different price.

2.2.4.3 Rent

Like mentioned previously, the Rent contract is the contract that manages the rent. The Rent contract gets deployed by the NewToken contract, which sends changes whenever a non-generic proposal is accepted. The Rent contract then applies these changes automatically. The contract has additional functionality for managing the rent, like storing information regarding it. The price of rent, the price of the deposit, the contract duration are stored and updated accordingly. The tenant and caretaker information is also stored. The tenant is the person who is living in the building, while the caretaker is the person responsible for the upkeep of the building, and plays the role a traditional landlord would play.

One of the main functionality of the Rent contract is the ability to pay rent. The tenant is able to pay the rent and deposit, which get locked up in the contract. The caretaker automatically gets sent a percentage of this rent, as compensation for their responsibilities. The token holders then have the ability to withdraw their share of rent, proportional to the amount of tokens they own. This is where the previously mentioned ERC20Votes extension comes in handy again, helps to avoid the problem of users transferring their tokens and claiming rent multiple times. At the time the rent is paid, a snapshot is taken in the same manner as with the proposal creation, and the share of rent that corresponds to each user is calculated. When a user claims their share of rent, it is unlocked from the contract and sent to their wallet.

The ethereum blockchain and other EVM compatible blockchains have some limitations that impede scheduling events to occur automatically at a certain point in time. Every smart contract action must be triggered by a call to a function from a user directly, or from another smart contract which in turn has the same limitations. This makes it hard to design where for example, the option to pay rent is activated once a month, like is needed in the Rent contract. There are two main options in this scenario.

The first one is to rely on some external source, like a bot that periodically makes a call once a month to the contract to activate the rent. The downside is that this sacrifices decentralisation as if the bot does not perform for whatever reason, the whole system stops working. An alternative is to have a function available for any user to call, and have them call it at the right time. Obviously this can be risky as there can be an issue if no one calls it, but in most situations it would be resolved quickly by anyone interested in using the application. This can be further improved by applying game theory concepts, in this case adding some incentive to calling the function in order to ensure it will be called. This approach was chosen, giving this responsibility to the caretaker as part of their duties, and in case they do not do as required, they can be voted out by the token holders through a proposal, and another one can be selected.

Once the rental contract of the tenant has ended regardless of the circumstances, there is the need to manage what to do with the deposit. This can be complicated, especially to do on the blockchain. A two step approach was chosen for the Rent contract. First, after the rental contract ends, the caretaker is responsible for proposing the percentage of the deposit to keep and to return. Ideally, if there have been no damages or fines, then the complete deposit will be returned to the tenant. The caretaker is incentivised to submit a proposal as soon as possible in order to claim their part of the deposit if they have one, but also because they can be voted out if they do not complete their duties. Once this proposal is submitted, the second step begins, where the tenant can either accept or reject the proposal the caretaker submitted. If the proposal is accepted, then the deposit is split between the caretaker and the tenant in the proposed ratio, and sent to both. On the flip side, if the proposal is rejected, then the entire deposit gets sent to Dstate. Dstate will then act as a mediator, review the case and decide what percentage of the deposit belongs to the caretaker and to the tenant, although this falls outside the delimitations of this project.

2.2.5 UI

The user interface of the application is developed using flutter. This framework allows for the creation of responsive multi platform apps, while following the material design guidelines. Flutter was chosen for its many advantages.

The most important is the platform, as having a mobile app allows for many users to interact with the system from their smartphones in an intuitive and simple manner. Being multi platform is also a big advantage, as even though the focus was on the android version, the flutter app can easily be ported to ios and web versions, opening up the system to even more potential users. Another big advantage is responsiveness. Users can not only use the app on different platforms, but also on devices with different screen sizes, without it being a detriment to the user experience. Finally, using material design further improves the user experience, as users are accustomed to the “look and feel” of material design, and it makes things like icons, widgets, and tools, easy to use and understand without having any previous experience with the app. It also provides a pleasant interface, which can arguably lead to higher user satisfaction while using the app, creating a more positive experience overall. Flutter achieves all these things with similar to native app performance, while still providing additional functionality through its own rendering engine.

The design for the app starts with a login screen which the user can use to then access the main functionality of the app. To log in, they connect through their metamask wallet and then they cryptographically sign a message to prove who they are. They can also register if it is their first time using the app. After that, the user can access their portfolio page, where they can see information on their ether balance, and also on the different tokens they own and their respective balances. From this page they can also access different pages, such as the building tokenization page or the buildings page. From the building tokenization page, a user can first register a new building in the system. Once that is done and the building has been approved, they can move on to the next step, creating the tokens that represent it, in the token creation page. Once all the parameters are decided, the contracts are deployed and the building has been successfully tokenized. Going back to the main portfolio page, the user can also access the buildings page. In the buildings page, all the currently tokenized buildings are listed with their basic information, so the user can browse through them. The user can also select a particular building, which will bring them to the trade page for the token of that building. In this page it is possible to trade the tokens, while also being able to navigate to the rent and governance pages. In the rent page all the rental management options are available. Similarly, in the governance page, the user can see the current proposals and create new ones. They can also select a particular proposal to see more information, vote on it if desired.

2.3 Implementation

In this section, how the core processes were implemented will be explained, starting from the backend, moving to the contracts and then the frontend.

2.3.1 Backend

2.3.1.1 App initialization

The first noticeable process happening in the backend is the server initialization.
The following code refers to the www file in the appendixes.

```
var app = require('../app');
var debug = require('debug')('dstate-be:server');
var http = require('http');
```

The app is initialised when the backend is launched from the terminal through the `npm start` command in the command line. This command is registered in a file called package.json and executes the command `nodemon .bin/www.` Nodemon is a library that *demonises* a process, meaning it is going to watch for any change in the codebase and restart the app whenever a change occurs. The www file first imports the app instance created in the app.js file from the express library. It also imports the HTTP library that will create the actual Node server.

```
var port = normalizePort(process.env.PORT || '3001');
app.set('port', port);


var server = http.createServer(app);

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
```

The database connection is realised by the following snippet in the app.js file. It takes a standardised URI like this

```
const                                      uri                                      =
`mongodb+srv://${process.env.MONGO_DB_USER}:${process.env.MONGO_DB_PS}@clus
ter0.cbsn2.mongodb.net/dstate?retryWrites=true&w=majority`;
```

The `MONGO_DB_USER` and `MONGO_DB_PS` variables are stored in a .env file that is loaded into
the app through a library called dotenv

```
mongoose.connect(uri, {
   serverSelectionTimeoutMS: 5000
}).catch(err => console.log(err.reason));
mongoose.Promise = global.Promise;
mongoose.connection.on('error', (err) => {
   console.log('We have an error with the database: ' + err);
})
```

The connection is realised by the mongoose library. It is important to notice that the connection can only be done from an IP that is whitelisted on the database cluster so, when the app gets deployed on its own static IP, the address should be whitelisted.

2.3.1.2 Sample request

The routes are grouped in files that export a router object. Whenever a new request is received, the request event is called, providing two objects: a request (an HTTP.IncomingMessage object) and a response (an HTTP.ServerResponse object). Those 2 objects are essential to handle the HTTP call. The first provides the request details. In this simple example, this is not used, but you could access the request headers and request data. The second is used to return data to the caller.
A sample request looks like this:

```
router.post("/createProposal",passport.authenticate("jwt",{session:
false}),tokenController.createProposal);
```

The first parameter is the request uri, the other parameters are the middlewares a request goes through.

A sample middleware looks like this:

```
exports.checkCaretaker = async (req, res, next)=>{
    …
   }
```

A middleware is an asynchronous function, meaning that it can be executed in parallel and the server doesn't have to wait for one middleware to finish its execution to execute another. In the context of a single request, however, middlewares need to be executed in order. For this reason, a middleware also takes the function `next` as a parameter. When the function is called the `next` middleware on the chain is called.
The data sent from the client can be accessed from the request (`req`) object.

There are two ways a client can send data to the server:

- Through the body of the request: this is the case of POST requests and data can be accessed using `req.body.parameter_name`
- Through the query string: this is the case of GET request but can also be used for POST requests, in this case data can be accessed using `req.query.parameter_name`

Data can be sent back to the client through the response (`res`) object. To do so the `res.send()` function is called, which takes up to two parameters. The first one is the status code, the second is the data in a JSON format.

2.3.1.3 Authentication

Decoding the JWT

The authentication process is handled by a library called passport js. Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped into any Express-based web application. In order to use this library a strategy had to be configured. A strategy is a standard process to authenticate the user implemented through a middleware. The JWT strategy was chosen since there is a need to decode JWT tokens.

```
const options={
   jwtFromRequest:ExtractJWT.fromAuthHeaderAsBearerToken(),
   secretOrKey:process.env.PASSPORT_SECRET,
}
const strategy=new JWTStrategy(options, async (payload,done)=>{
   await User.findOne({_id:payload.id}).then((user)=>{
       if(user){
           return done (null,user)
       }
       else{
            return done(null,false)
        }
    }).catch(err=>done(err,null))
})

passport.use(User.createStrategy());
passport.use(strategy)

passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());
```

First the options which are the secret key used in the encryption and decoding algorithm and the field of the header from which to extract the JWT that will be decoded are declared.

Then a strategy object is initialised, which is an instance of the JWTStrategy of passport js. The constructor takes the options and a callback function. The callback is used to tell passport what to do after the JWT is decoded. In this case, it takes the id from the payload of the decoded JWT and queries the database. If a user with that id exists it returns it to the next middleware in the chain. If not it fails with a 401 error.

Issuing the token

The function that issues the token is the login function in authController. This function is called from the auth route after the user has signed the message in the wallet provider.

The first thing that the function does is to retrieve the signature and the public address from the body and check if the fields are not null. If they are, it returns an error.

```
const {signature, publicAddress} = req.body;
if (!signature || !publicAddress)
   return res
       .status(400)
       .send({error: 'Request should have signature and publicAddress'});
```

Then creates a javascript promise that queries the database and retrieves the user with the given public address. If the user doesn't exist it returns an error. If not, it returns the user to the next callback.

```
User.findOne({publicAddress: publicAddress}).exec()
   .then((doc) => {
       if (!doc) {
           res.status(401).send({
                   error: `User with publicAddress ${publicAddress} is not
found in database`,
           });

           return null;
       }

       return doc;
   })
```

The next step is to verify that the user that is being authenticated is the one that signed the message in the wallet provider. To do so the same message that the user signed together with the signature is sent to the `ethSigUtil.recoverPersonalSignature()` of the ethSigUtil library. The function returns the public address that signed the message. Then the address returned is compared with the one the client sent. If they are the same the next callback is executed, if not an error is returned.

```
.then((user) => {

   const msg = `I am signing my one-time nonce: ${user.nonce}`;

   const msgBufferHex = ethJsUtil.bufferToHex(Buffer.from(msg, 'utf8'));

   const address = ethSigUtil.recoverPersonalSignature({
       data: msgBufferHex,
       sig: signature,
   });

   if (address.toLowerCase() === publicAddress.toLowerCase()) {
       return user;
   } else {
       res.status(401).send({

           error: 'Signature verification failed',
       });

       return null;
   }
})
```

Since the user nonce was used to sign the message, another one needs to be produced.

```
.then((user) => {

   user.nonce = Math.floor(Math.random() * 10000);
   return user.save();
})
```

The last step is to issue and return the JWT together with the user id.

```
.then((user) => {
    let accessToken=jwt.sign(
        {
            id: user.id, publicAddress,
        },
        process.env.PASSPORT_SECRET,
    )
    return {accessToken, user_id:user.id};
})
.then(({accessToken,user_id}) => res.json({accessToken,user_id}))
```

The jwt library is used to issue the token. The sign function takes the payload object as a parameter and the secret key and returns a JWT string.

2.3.1.4 Sample transaction

Some of the routes in the backend involve calling functions from the contracts that modify the status of the blockchain, which means that they will create a transaction that the user has to sign with the wallet provider.

As an example to explain the whole process the following request is useful:

```
router.post("/deploy",passport.authenticate("jwt",{session:
false}),middlewares.checkForBuildingApproval,
buildingController.deployToken );
```

This request creates the transaction to deploy the NewToken.sol contract. When the request is handled it first goes through the passport middleware that decodes the JWT token in the header. The next middleware it has to go through is the deployToken function in the building Controller.

```
exports.deployToken = async (req, res, next) => {
    const pathToFile = path.join(__dirname, '../solidity/build/contracts',
'NewToken.json')
```

```
   var data = JSON.parse(fs.readFileSync(pathToFile));
   var myContract = new web3.eth.Contract(data.abi);
   let encodedABI = await myContract.deploy({
       data: data.bytecode,
        arguments: [req.body.initial_amount, req.body.name, req.body.symbol,
BigInt(req.body.rentPrice),                  BigInt(req.body.depositPrice),
req.body.remainingMonths,    req.body.caretakerShare,    req.body.caretaker,
req.body.tenant]
   }).encodeABI()
                               const         nonce          =          await
web3.eth.getTransactionCount(req.user.publicAddress);
   res.send({abi: encodedABI, nonce: nonce})

}
```

The first thing that the function does is to get the path to the JSON interface of the contract. This is located in the build folder after a contract is compiled. All the contracts are compiled before the application startup. Since it is a JSON file the function reads from it using the JSON library.

The next step is to create a contract object passing the ABI to the constructor. The function gets the ABI from the previously parsed JSON interface. By calling `myContract.deploy`, giving it the arguments for the contract constructor and the bytecode from the JSON interface a new transaction is created. With the `encodeABI()` function, the encoded ABI of the just-created transaction is received. The function is asynchronous thus waiting for it to finish using await in front of the function call is necessary.

The function then calculates the nonce using the web3 js library passing the public address of the user. The user in the req object is the one that was decoded from the passport middleware.

The final step is to send the ABI and the nonce back to the client using `res.send()`

The process of calculating the ABI and the nonce for a transaction is standard for all the requests that involve creating a transaction. The only difference is the method called from the contract.

Retrieving the transaction receipt

This specific route also involves retrieving the transaction receipt from the blockchain and performing actions after the transaction has been confirmed.
The route that starts this process is

```
router.post("/createToken",passport.authenticate("jwt",{session:
false}),middlewares.checkForBuildingApproval,
buildingController.createToken );
```

The middleware that handles the request is the following

The first thing that the createToken function does is to retrieve the transaction from the blockchain using web3 js library. The `getTransaction` function takes a parameter which is the transaction hash sent from the frontend.

```
let tx = await web3.eth.getTransaction(req.body.transactionHash)
```

Since the transaction could still be pending the next step is to poll the blockchain until the block number is not null anymore. If the block number has a definite value it means that the transaction is completed and the receipt is obtained.

```
while (tx.blockNumber == null) {
   tx = await web3.eth.getTransaction(req.body.transactionHash)
}
let                    receipt                    =                    await
web3.eth.getTransactionReceipt(req.body.transactionHash)
```

If the transaction fails for any reason, an error is returned to the frontend.

```
if(receipt.status==false){
   res.send(500, "Transaction failed")
}else{
…
}
```

If the transaction is successful on the other end, a token is created on the database. The building related to the token is updated, setting the token_id to the id of the just created token.

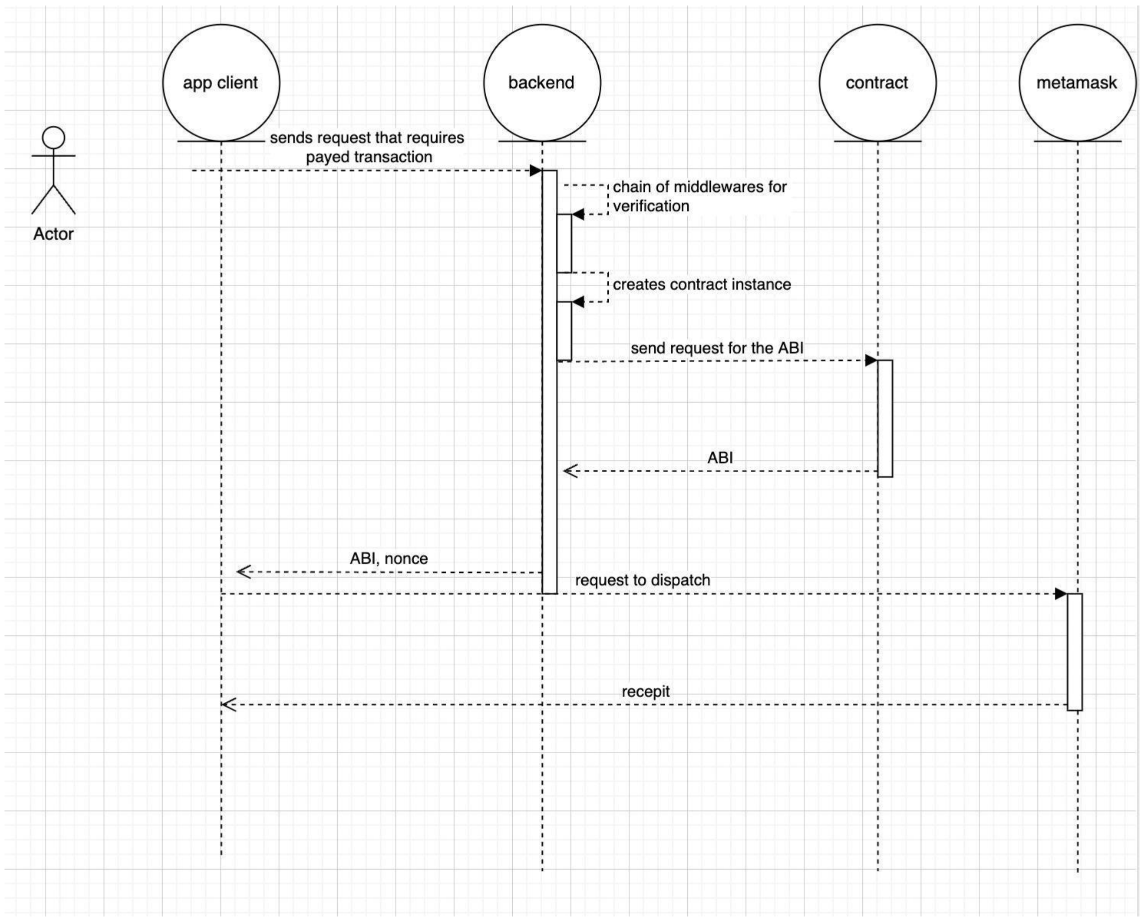
```
try {
   let token = await Token.create({
       name: req.body.name,
       symbol: req.body.symbol,
       initial_amount: req.body.initial_amount,
       address: receipt.contractAddress,
       user_id: req.user._id
   });
   let building = await Building.findOneAndUpdate(
       {"_id": req.body.building_id},
       {"token_id": token._id, "rentContractAddress": rentAddress},
   )
   let user = await User.findOneAndUpdate(
       {"_id": req.user._id},
       {$push: {"token_ids": token._id}}
   )
   return res.send({building: building, token: token})

} catch (error) {
   console.log(error)
   res.send(500, error)
}
```

The rentContractAddress querying the newly deployed contract is retrieved. Since the function that returns it is a view function, creating a transaction is not necessary. Finally, the token id is added to the arrays of tokens that the user owns.

The entire process can be graphically described by this interaction diagram.

Figure 6: Transaction interaction Diagram


app client
backend
contract
metamask
Actor
sends request that requires payed transaction
chain of middlewares for verification
creates contract instance
send request for the ABI
ABI
ABI, nonce
request to dispatch
recepit

2.3.2 Contracts

The smart contracts are all implemented using solidity, the default programming language used for Ethereum and other EVM compatible blockchains. Solidity is an object-oriented high-level programming language, with influences from C++, Javascript, and python. Solidity is a compiled language, and when a .sol file is compiled, generates a set of opcodes. These opcodes are similar to an assembly language, and then get encoded into bytecode, which the EVM can process in order to modify the state of the blockchain.

2.3.2.1 NewToken

Probably the most important smart contract in this system is NewToken.sol, because it is used for the tokenization of the buildings.

```
pragma solidity ^0.8.0;

import "./Rent.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import
"@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";
```

What is seen above is the beginning of the contract. In solidity it is necessary to specify the solidity compiler version to use. This is important because even if the language gets updated, a smart contract will always be run with the version it was compiled in. In this contract the version 0.8.0 or above is being used.

As for the imports, this contract has to communicate with Rent.sol, so it is being imported. Both the ERC20.sol and ERC20Votes.sol from Openzeppelin are also being imported, as described previously.

```
contract NewToken is ERC20, ERC20Votes {

    Rent public rent;

    struct proposal {
        string title;
        string description;
        uint proposalType;
        uint id;
        uint blockNumber;
        uint uint0;
        uint uint1;
        uint uint2;
        address address0;
    }
```

Here the contract NewToken is declared, while also specifying that this contract inherits from ERC20 and ERC20Votes. The public variable rent is declared, which will be used to store a reference to the rent contract. Then a struct called proposal is defined, which contains all the attributes a proposal will have.

```
uint public proposalNumber = 0;
mapping(uint => proposal) public proposals;
mapping(uint => uint) public votes;
mapping(uint => bool) public votingResult;
mapping(uint => mapping(address => bool)) voters;
```

In the code above, proposalNumber is declared. This variable will be used to keep track of the current proposal, and will also be used as the id of the new proposals. After that, a few mappings are defined. Mappings are very useful in solidity in order to store values and access them efficiently without having to iterate through an array. The first mapping maps the number of a proposal to the proposal itself, to be able to retrieve them easily. The second mapping maps the number of a proposal to the number of votes it has. The third mapping maps the number of a proposal to the votingResult, so if the proposal was accepted or not. Finally the fourth and most important mapping maps the proposal number to another mapping, which in turns maps a user address to a bool. This allows the contract to store if an address has voted or not, for every different proposal.

```
	constructor(uint tokenNumber, string memory name, string memory
symbol, uint _rentPrice, uint _depositPrice, uint _remainingMonths, uint
_caretakerShare, address payable _caretaker, address payable _tenant)
ERC20(string(abi.encodePacked("dstate-", name)), symbol)
ERC20Permit("Governance") {
		_mint(msg.sender,tokenNumber*10**18);
		delegate(msg.sender);

			rent = new Rent(_rentPrice, _depositPrice,
_remainingMonths, _caretakerShare, _caretaker, _tenant, address(this),
address(this));

	}
```

In this code the constructor for the contract is defined. In a solidity smart contract the code inside a constructor is only executed once, when the contract is deployed. This constructor takes many arguments, some are used for the creation of the tokens but the majority are passed on to the rent contract constructor. Here the name and symbol of the ERC20 token are also defined. After that, the tokens are minted using the _mint() function and sent to the address that deployed the contract, msg.sender. It is worth mentioning that the number of tokens created is multiplied by 10^{18}. This is because solidity has no floating point numbers, only whole numbers. In order to represent a number with decimals, a big integer is chosen and a specific value is chosen as the reference for the decimal point. In this case 10^{18}represents 1.0 token, which is the standard for ERC20 tokens, and gives 18 decimal points of precision. This is useful in order for users to trade very small amounts.

After the minting, the delegate() function is called. This must be done because of the way the ERC20Votes library works if it is desired that all tokens are always tracked as votes. The same delegate function is also applied to the _afterTokenTransfer() function as can be seen in the appendix. Finally, the rent variable is initialised with the instance of the Rent contract, which is deployed at the end of the constructor.

```
            function createProposal(string memory _title, string memory
_description, uint _proposalType, uint _uint0, uint  _uint1, uint _uint2,
address _address0) public returns(uint _theId){
              ERC20Votes token = ERC20Votes(address(this));
               require(token.getVotes(msg.sender) > 0, "You must own some
tokens to create a Proposal");
                  proposal memory prop = proposal(_title, _description,
_proposalType, proposalNumber, block.number, _uint0,   _uint1, _uint2,
_address0);
              proposals[proposalNumber] = prop;
              votes[proposalNumber] = 0;
              proposalNumber++;
              return proposalNumber - 1;
          }
```

The createProposal() function allows users to submit a new proposal for the building. The parameters it takes include information about the proposal itself, as well as some generic fields that are used to store the different variables needed in the different kinds of proposals once they are accepted. The function also requires that the user owns more than 0 tokens in order to submit a proposal, and any call to this function reverts if this condition isn't met.

When a new proposal is created, the function arguments are used, as well as the current block number, in order to keep track of the instant the proposal was created. After that the new proposal is stored in the proposals mapping, using proposalNumber as a key, which has the same value as the proposal id. Then the votes of this proposal are set to zero, and finally the proposalNumber is increased by one.

```
function vote(uint _proposalId) public returns(bool _result) {
    require(!votingResult[_proposalId], "Voting Ended");
    require(!voters[_proposalId][msg.sender], "Already Voted");
    proposal memory prop = proposals[_proposalId];
    ERC20Votes token = ERC20Votes(address(this));
        votes[_proposalId]  +=  token.getPastVotes(msg.sender,
prop.blockNumber);
    voters[_proposalId][msg.sender] = true;

    if(votes[_proposalId] >= (this.totalSupply() / 2)){
        votingResult[_proposalId] = true;
```

The voting function allows users to vote on a proposal, and only takes the id of a proposal as a parameter. First the function checks that the proposal has not been accepted already, and then checks that the user calling the function has not voted already. If any of these two conditions are not met, any call to the function reverts.

After these checks, the number of votes for the proposal is increased, and the amount of votes being added is determined by the function getPastVotes() from the ERC20Votes library. It takes the address of the user calling the function and the block number at which the proposal was created, and returns the number of votes the user had at that moment. Then the voters mapping is updated to reflect that the user has already voted.

Finally a check is made to determine if the current number of votes is enough to accept the proposal. The number of votes is compared to the total supply of the token divided by two, and if it is larger or equal to it, the proposal is passed. This then results in the votingResult mapping being updated. In the case of non-generic proposals, this also triggers a function that will perform the required change in the Rent contract. The function in question is different for every kind of proposal.

2.3.2.2 BuySell

The BuySell contract manages all the trades users make through the application.

```
pragma solidity ^0.8.4;

import "./NewToken.sol";

import "@openzeppelin/contracts/access/Ownable.sol";
```

The NewToken contract is imported, which allows the BuySell contract to access its functionality. The Ownable.sol contract is also imported, it is another library provided by Openzeppelin. The functionality of this library will be explained in a later section, as it is much more relevant in the Rent contract.

```
    struct sellingInstance {
        address payable seller;
        address tokenAddress;
        uint256 amountOfETH;
        uint256 id;
        uint amountToSell;

    }

    struct variables{
        int firstCheapestIndex;
        int nextCheapestIndex;
        bool sent;
        uint price;
        int remainingToBuy;

    }
```

The first thing that the contract does is to declare two structs that will be needed in the future functions. A struct is a way used in solidity to group variables. The selling instance struct models a selling order in an order book system.

```
    sellingInstance [] sellingInstances;
    uint256 idCount=0;

     event  BuyTokens(address  buyer,  uint256  amountOfETH,  uint256
amountOfTokens);
```

Then the contract declares an array of selling instances. This array models a group of orders of the order book system.
The idCount variable keeps track of the last id given to a selling instance in the array.
The contract also initialises an event that will be emitted by the buy function.

```
     function  setPrice(uint256  amountOfETH,  uint256  tokenAddress,  address
tokenAddress) public returns (sellingInstance memory instance){
        ERC20 newToken = ERC20(tokenAddress);
        newToken.transferFrom(msg.sender,address (this),tokenAmount);
          sellingInstance memory s = sellingInstance(payable(msg.sender),
tokenAddress, amountOfETH, idCount,tokenAmount);
        sellingInstances.push(s);
        idCount= idCount+1;
        return s;

    }
```

The setPrice function creates a new order (sellingInstance) in the order book system and stores it in the array of sellingInstances. The order is initialised with an id which is the value of the idCount global variable, the address of the sender, the address of the token to sell, the amount the user wants to sell and the price in wei (the smallest unit of ETH) for one token. The contract transfers the amount of tokens the user wants to sell from the user address to the contract address. In order for the contract to do the transfer the user needs to call the approve function in the token contract to authorise the BuySell contract to spend their tokens. Normally it would be dangerous to authorise another address to spend user tokens because the authorised address can spend the tokens however they want. But since the contract is a script running on the blockchain the only ways it can spend the tokens is by the functions it has inside. The user can see the functions the contract has therefore can know in advance if to trust the contract or not.

```
    function getNextCheapest(address tokenAddress, sellingInstance memory
previousCheapest,  sellingInstance[]  memory  sortedArray)  private  pure
returns (int instanceIndex){

        for (uint j = 0; j < sortedArray.length; j++) {
                          if  (sortedArray[j].tokenAddress  ==  tokenAddress  &&
sortedArray[j].amountOfETH>=  previousCheapest.amountOfETH  &&  int(j)  >
indexOf(sortedArray, previousCheapest.id
) {
                    return int(j);
                }
        }
        return -1;
    }
```

The getNextCheapest function gets the next cheapest sellingInstance given a token address and the previous price. It takes as a parameter a sortedArray of selling instances, the previous cheapest selling instance of the same token address and the token address. It scans the sorted array until it finds a selling instance whose token address is the same as the one given as an argument and the price is greater or equal than the one of the firstCheapest instance. If an instance is found then the index associated with it is returned, if not it returns -1.

```
    function getPriceForTokens(address tokenAddress, uint amount) public
view returns(uint pri){
        sellingInstance[] memory sortedArray =
injectionSort(sellingInstances);
        int firstCheapestIndex = getFirstCheapest(tokenAddress,
sortedArray);
        int nextCheapestIndex;
        sellingInstance memory nextCheapest;
        uint remainingToBuy =amount;
        uint price;
        if(firstCheapestIndex == -1){
            return 0;
        }
        sellingInstance memory firstCheapest=
sortedArray[uint256(firstCheapestIndex)];
        if(firstCheapest.amountToSell>=amount){
            price= (amount * firstCheapest.amountOfETH) / (10 ** 18);
            return price;
        }else{
            price= (firstCheapest.amountToSell *
firstCheapest.amountOfETH) / (10 ** 18);
            remainingToBuy= remainingToBuy - firstCheapest.amountToSell;
            nextCheapestIndex =
getNextCheapest(tokenAddress,firstCheapest, sortedArray);
        }
        while(remainingToBuy >0 && nextCheapestIndex!=-1){
            nextCheapest=sortedArray[uint256(nextCheapestIndex)];
            if(nextCheapest.amountToSell>=remainingToBuy){
                price= price + ((remainingToBuy *
nextCheapest.amountOfETH) / (10 ** 18));
                return price;
            }else{
                price= price + ((nextCheapest.amountToSell *
nextCheapest.amountOfETH) / (10 ** 18));
                remainingToBuy= remainingToBuy -
firstCheapest.amountToSell;
                nextCheapestIndex =
getNextCheapest(tokenAddress,nextCheapest, sortedArray);
            }
        }
        return price;
    }
```

The function returns to the user the promised price for the amount of tokens they wish to buy of a given token address. It first sorts the sellingInstances array in memory. Then it calls the getFirstCheapest function to get the selling instance that has the lowest price in ETH per token of the given token address. From this point there can be two cases: the amount requested from the user could be greater than the amount of tokens on sale by that sellingInstance, or the amount requested is less or equal than the amount on sale.

If the amount requested is less or equal than the amount on sale then the amount to buy multiplied by the ETH per token is returned to the user.

If the amount requested is greater than the amount to buy multiplied by the ETH per token is added to the price variable, previously initialised to zero. The amount on sale for that sellingInstance is then subtracted from the remainingToBuy variable, previously initialised as the amount the user wants to buy in total.
After this first step the next cheapest instance is found using the getNextCheapest function. From then on the function enters into a loop that ends if one or more of the following are verified:

- There is no more nextCheapest in the array
- the remaining to buy gets to zero

At each cycle one of the two cases analysed above with the fistCHeapest can occur. At the end of each cycle a new NextCheapest is found.
When the function exits the loop it returns the price to the user.

```
function buyTokens(address tokenAddress, uint promisedPrice, uint amount)
public payable {
        uint actualPrice = getPriceForTokens(tokenAddress,amount);
        require(msg.value >= actualPrice, "Send ETH to buy some tokens");
        ERC20 newToken = ERC20(tokenAddress);
        uint256 vendorBalance = newToken.balanceOf(address(this));
         require(vendorBalance >= amount, "Vendor contract has not enough
tokens in its balance");
        require(actualPrice == promisedPrice, "Price mismatch");
                          sellingInstance   []   memory   sortedArray   =
injectionSort(sellingInstances);
        sellingInstance memory nextCheapest;
                                        variables    memory    vars    =
variables(int(getFirstCheapest(tokenAddress, sortedArray)), -1, false, 0,
int(amount));
        (vars.sent) = newToken.transfer(msg.sender, amount);
        require(vars.firstCheapestIndex != -1, "No tokens available");
                               sellingInstance   memory   firstCheapest=
sortedArray[uint(vars.firstCheapestIndex)];
        int firstCheapestRealIndex = int(firstCheapest.id);
        if(firstCheapest.amountToSell>=amount){
            sellingInstances[uint(firstCheapestRealIndex)].amountToSell =
sellingInstances[uint(firstCheapestRealIndex)].amountToSell      -amount;
sellingInstances[uint(firstCheapestRealIndex)].seller.transfer((amount   *
firstCheapest.amountOfETH) / (10 ** 18));
            vars.remainingToBuy = 0;
        }else{
                                vars.price=  (firstCheapest.amountToSell  *
firstCheapest.amountOfETH)          /          (10          **          18);
sellingInstances[uint(firstCheapestRealIndex)].seller.transfer(vars.price
);
                                vars.remainingToBuy  =  vars.remainingToBuy-
int(sellingInstances[uint(firstCheapestRealIndex)].amountToSell);

sellingInstances[uint(firstCheapestRealIndex)].amountToSell=0;
                                                vars.nextCheapestIndex    =
getNextCheapest(tokenAddress,firstCheapest, sortedArray);
        }
        while(vars.remainingToBuy >0 && vars.nextCheapestIndex!=-1){
            nextCheapest=sellingInstances[uint(vars.nextCheapestIndex)];
            int nextCheapestRealIndex = int(nextCheapest.id);
            if(int(nextCheapest.amountToSell)>=vars.remainingToBuy){

sellingInstances[uint(nextCheapestRealIndex)].amountToSell              =
sellingInstances[uint(nextCheapestRealIndex)].amountToSell
-uint(vars.remainingToBuy);
sellingInstances[uint(nextCheapestRealIndex)].seller.transfer((uint(vars.
remainingToBuy) * nextCheapest.amountOfETH) / (10 ** 18));
                vars.remainingToBuy = 0;
            }else{
                                vars.price=  (nextCheapest.amountToSell  *
nextCheapest.amountOfETH) / (10 ** 18);

sellingInstances[uint(nextCheapestRealIndex)].seller.transfer(vars.price)
;
                                vars.remainingToBuy  =  vars.remainingToBuy-
int(sellingInstances[uint(nextCheapestRealIndex)].amountToSell);
sellingInstances[uint(nextCheapestRealIndex)].amountToSell=0;
                    vars.nextCheapestIndex = getNextCheapest(tokenAddress,
nextCheapest, sortedArray);
            }
        }
        require(vars.sent, "Failed to transfer token to user");
        emit BuyTokens(msg.sender, msg.value, amount);
    }
```

The buyTokens function works in a similar way to the getPrice function. Firstly it calls the getPriceForTokens function again and checks if the user has sent enough ETH to buy the requested amount of tokens. Another check that the function does is if the promised price that the user passes as an argument is equal to the actual price. The function then initialises an instance of the NewToken and checks that if the balance of tokens of the given token address that the contract holds is greater or equal than the one the user wants to buy.

It sorts the array of sellingInstances in memory, gets the first cheapest and transfers the tokens to the buyer. From this point there can be two cases as explained in the previous function. The key difference is that since the sellingInstances array in the contract will be modified the firstCheapest index referring to the sorted array is transformed into the index referring to the real array. This process also happens for each of the nextCheapestInstance.

The main differences in the loop are that instead of adding up the remaining to buy or the amount of tokens on sale multiplied by the price in ETH per token, the function sends that value in ETH to the seller of each instance. Also it subtracts the amount that has been bought from each instance from the amount on sale for each instance.

```
function cancelSale(address tokenAddress, uint amount) public {
```

The cancelSale() function works in a very similar manner to the buyTokens() function described previously. The key difference is that only the selling instances of the user that calls the function will be considered. Also, no ether needs to be sent to this function in order for the user to withdraw their tokens. It is relevant to mention that the tokens are removed from sale starting with the cheapest ones first. This is intentional, as sellers might not want to keep their cheaper tokens on sale if the price is increasing.

2.3.2.3 Rent

The Rent contract houses all the functionality around the rent management, and also implements the functions that change things once certain proposals have been accepted.

```
pragma solidity ^0.8.4;

import "./NewToken.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";

contract Rent is Ownable{
```

The NewToken, ERC20, ERC20Votes, and Ownable contracts are imported. Ownable is a very important extension for this contract, as it allows for certain functions to only be accessed by a certain address, defined as the owner.

```
struct rent {
        ERC20 token;
        uint rentPrice;
        uint depositPrice;
        uint currentDeposit;
        uint depositProposal;
        uint caretakerShare;
        uint rentNumber;
        uint remainingMonths;
        uint rentBlockTime;
        uint8 status;
        mapping(uint => uint) rentBlock;
        mapping(uint => uint) rentAmountPerToken;
        address payable caretaker;
        address payable tenant;
        address payable previousTenant;
    }
```

The struct rent is created, to store all the variables that represent the state of the rent for the current building. Most of these variables are self explanatory, but some like status or rentBlock might need some explaining. The status variable stores the current state of the rental building, such as if rent is due or if rent has already been paid among others. The rentBlock mapping maps the rentNumber to the block number at the time the rent was paid.

```
    rent rentInfo;
                          address       payable       nullAddress       =
payable(0x0000000000000000000000000000000000000000);
                        address       payable       disputeAddress       =
payable(0x7176bd09199068E21bE4137d1630fb8712633445);
    address tokenAddress;
    mapping( uint => mapping(address => bool)) tokenHolders;

    event RentPaid(address tenant, uint amountOfETH, uint blockNumber);
```

Some more variables are created, with some notable ones being the dispute address which will be used for mediation, and the tokenHolders mapping. This mapping maps the rent number to a mapping, which in turn maps a user address to a bool. This mapping is used to store the information of which users have already claimed their share of rent every given month the rent has been paid. After that the RentPaid event is defined, which will be emitted whenever rent is paid.

```
        constructor(uint _rentPrice, uint _depositPrice, uint _remainingMonths, uint _caretakerShare, address payable _caretaker, address payable _tenant, address _votingContract, address _tokenAddress)
{
        rentInfo.token = ERC20(_tokenAddress);
        rentInfo.rentPrice = _rentPrice;
        rentInfo.depositPrice = _depositPrice;
        rentInfo.currentDeposit = 0;
        rentInfo.caretakerShare = _caretakerShare;
        rentInfo.caretaker = _caretaker;
        rentInfo.rentBlock[0] = block.number;
        rentInfo.rentNumber = 0;
        rentInfo.rentAmountPerToken[0] = 0;
        rentInfo.tenant = _tenant;
        rentInfo.previousTenant = nullAddress;
        rentInfo.remainingMonths = _remainingMonths;
        rentInfo.depositProposal = 101;
        rentInfo.rentBlockTime = block.timestamp;
        tokenAddress = _tokenAddress;
        if(rentInfo.tenant != nullAddress) { rentInfo.status = 2; }
        else { rentInfo.status = 0; }
        transferOwnership(_votingContract);
    }
```

In the constructor of the contract, all of the rentInfo variables are initialised to default values, to be used and modified later on. The value of the status variable depends on the tenant argument; if there is a non-null tenant address then the status is set to 2 (rent due), while it is set to 0 (unlisted) in the opposite case.

After initialising all the variables, the transferOwnership() function from the

Ownable.sol contract is called. This function sets a new owner, in this case the owner is set to the NewToken contract which deployed this contract. The reasoning for this will be explained later on, but this means that functions with the onlyOwner modifier can only be called by the NewToken contract.

```
    function getRentAndDepositPrice() public view returns(uint _price) {
        if(rentInfo.currentDeposit != 0){
            return rentInfo.rentPrice;
        }
        else {
            return (rentInfo.rentPrice + rentInfo.depositPrice);
        }
    }
```

The next section is filled with a lot of getter functions that are simple and not really worth talking about, but one in particular is relevant. The getRentAndDepositPrice() function returns the amount the tenant has to pay in rent. The function automatically adds the deposit price to the rent price if it is the tenant's first month, while only returning the rent price for all the next months.

```
      function newTenant(address payable _tenant, uint _remainingMonths,
uint _rentPrice, uint _depositPrice) public onlyOwner returns(address
_newTenant, uint _newMonths) {
           require(rentInfo.status < 2, "Current tenant must be removed
first");
        rentInfo.tenant = _tenant;
        rentInfo.remainingMonths = _remainingMonths;
        rentInfo.rentPrice = _rentPrice;
        rentInfo.depositPrice = _depositPrice;
        rentInfo.status = 2;
        return (rentInfo.tenant, rentInfo.remainingMonths);
    }
```

The next section contains the functions that change the state of the Rent contract after certain proposals have been accepted. They are all pretty similar, but newTenant() is the most interesting from an implementation point of view. The function implements the onlyOwner modifier, like mentioned previously, only allowing the NewToken contract to call it. This is extremely important, as without this modifier this function would be a huge security flaw, allowing any user to accept a new tenant. The function also checks that there is no current tenant, as a new tenant can only be added once the previous one has been removed or their contract has ended. After these checks the function updates the values of the rentInfo variables to the information for the new tenant.

```
    function requestRent() public {
         require(block.timestamp - rentInfo.rentBlockTime > 2629743, "One
month has not gone by yet");

        rentInfo.rentBlockTime = block.timestamp;
        rentInfo.rentNumber += 1;
        rentInfo.status = 2;
    }
```

The requestRent() function has to be called by the caretaker in order to initiate the process of paying rent for the new month. The reasoning for doing this was discussed previously in the design section. The function is not limited to only the caretaker, as there is no downside in allowing any user to call it if they wish to. There is a check that determines if at least one month has gone by since the previous rent was requested, and that limits how often this function can be called. It can be noticed that the value of 2629743 is used to determine how much time has passed. These are the average number of seconds in a month, considering all months having the same duration and the same for every year. While this is slightly inaccurate, in this case it was decided it was the best decision. This is because checking the number of seconds for every different month, and for leap years, leads to a lot of complexity that could potentially introduce vulnerabilities, something that was deemed unnecessary when considering the inaccuracy is small and generally has no major implications.

After this check the current block timestamp is registered in order to calculate when the requestRent() function can be called next month. Then the rentNumber is increased by one, and the status set to 2 (rent due).

```
    function payRent() public payable {
        require(rentInfo.status == 2, "Rent is not due yet");
        require(rentInfo.remainingMonths > 0, "Rental contract is over");
        if(rentInfo.currentDeposit != 0){
               require(msg.value >= rentInfo.rentPrice, "Not enough eth to
cover rent");
            rentInfo.rentBlock[rentInfo.rentNumber] = block.number;
                  rentInfo.caretaker.transfer((rentInfo.caretakerShare  *
msg.value) / 100);
             rentInfo.rentAmountPerToken[rentInfo.rentNumber] = (msg.value
-   ((rentInfo.caretakerShare   *   msg.value)   /   100))   /
(rentInfo.token.totalSupply() / (10**18));
            rentInfo.remainingMonths -= 1;
            rentInfo.status = 3;
            if(rentInfo.remainingMonths == 0) {
                rentInfo.previousTenant = rentInfo.tenant;
                rentInfo.tenant = nullAddress;
                rentInfo.status = 1;
            }
```

The payRent() function allows the tenant to pay rent. It has the payable modifier, which makes it so that a function can receive ether. This function is divided in two parts, which behave slightly differently depending on if the user is paying rent or is also paying the deposit. The logic is very similar so the implementation of the first one will be explored. There are three main checks that prevent the function from being called. The first one checks that the rent is due. The second one checks that there are still months remaining in the rental contract. The third one checks that the tenant has sent enough ether to cover the cost of rent for the current month.

After these checks, the current block number is stored. Then the caretaker gets sent the ether that corresponds to them according to the caretakerShare variable. Afterwards, the rentAmountPerToken is calculated. This value reflects how much ether corresponds to a token, and will then be used in order to fairly distribute the rent to token holders. Next the values of some variables are updated, such as the remainingMonths which is decreased by one, and status which is changed to 3 (rent paid). If it is the last month of the contract, more variables are updated to reflect it. Like mentioned previously, a very similar process is followed for the user paying the deposit. After all this, the RentPaid event is emitted.

```
    function withdrawRent() public returns(uint _rent) {

        require(!tokenHolders[rentInfo.rentNumber][msg.sender], "Rent already claimed");
        require(rentInfo.rentAmountPerToken[rentInfo.rentNumber] > 0, "Rent not available yet");
        ERC20Votes token = ERC20Votes(tokenAddress);
        uint votes = token.getPastVotes(msg.sender, rentInfo.rentBlock[rentInfo.rentNumber]);
        require(votes != 0, "No delegated votes");
        uint rentPayment = (rentInfo.rentAmountPerToken[rentInfo.rentNumber] * votes) / (10 ** 18);
        payable(msg.sender).transfer(rentPayment);
        tokenHolders[rentInfo.rentNumber][msg.sender] = true;
        return rentPayment;
    }
```

The withdrawRent() function allows token holders to withdraw part of the rent, proportional to the tokens they hold. There are a few checks in place, the first one makes sure the user has not claimed rent already this month, while the second one makes sure the rent has already been paid by the tenant. Following these checks, the ERC20Votes function getPastVotes() is used in order to determine the user's balance at the instant the rent was paid. Then another check is performed, to make sure the user had a balance greater than 0 at this time. Subsequently the amount that corresponds to the user is calculated and sent to them. Finally it is stored in the tokenHolders mapping that the user has already claimed rent for the current month.

It is also relevant to mention that there is a very similar function to this that allows the user to withdraw rent from a previous month in the case they did not withdraw it in time.

```
    function returnDepositProposal(uint _depositProposal) public {
            require(_depositProposal < 101, "Please enter a valid % of the
deposit");
          require(msg.sender == rentInfo.caretaker, "Only the caretaker can
call this function");
         require(rentInfo.status == 1, "Rental contract is not over");
            require(rentInfo.currentDeposit > 0, "Deposit has already been
withdrawn");

         rentInfo.depositProposal = _depositProposal;
    }
```

The returnDepositProposal() function is used when a tenant ends their contract, and has to have their deposit returned or taken, or some combination of both. The checks for this function require that the caretaker is the one that calls it, while having a valid number for the deposit proposal. There are additional checks that make sure the rental contract has not ended yet, and that the deposit has not yet been withdrawn. After all these checks, the deposit proposal is stored in the depositProposal variable of rentInfo.

```
    function returnDepositAcceptance(bool _depositAcceptance) public {
          require(msg.sender == rentInfo.previousTenant, "Only the previous
tenant can call this function");
         require(rentInfo.depositProposal != 101, "No deposit proposal has
been created yet");
        uint deposit = rentInfo.currentDeposit;
        rentInfo.currentDeposit = 0;
        if(_depositAcceptance) {
                                  rentInfo.caretaker.transfer((deposit   *
rentInfo.depositProposal) / 100 );
                  rentInfo.previousTenant.transfer(deposit - ((deposit *
rentInfo.depositProposal) / 100 ));
        }
        else{
            disputeAddress.transfer(deposit);
        }
        rentInfo.previousTenant = nullAddress;
        rentInfo.depositProposal = 101;
    }
```

The returnDepositAcceptance() function is used by the leaving tenant to get back their deposit. There are some checks that make sure the leaving tenant is the one calling the function, and that a deposit proposal has already been submitted by the caretaker. The function then has two possible results depending on if the leaving tenant agrees with the proposal from the caretaker or disagrees. In the first case, the deposit is split and sent to both of them. In the second scenario, the complete deposit gets sent to a disputeAddress, managed by Dstate, for mediation.

2.3.3 UI

2.3.3.1 Connection to Wallet

The connection from the wallet to the frontend of the application is of big importance, as it allows a user to log in and sign transactions. In order to achieve this connection, the protocol WalletConnect is being used, which allows for apps to integrate with different wallets. In this case the focus was on the Metamask wallet, as it is the most widely used one, but other wallets can also be used.

```
final session = await connector.createSession(
   chainId: 4,
   onDisplayUri: (uri) async =>
   {print(uri), await launchUrl(
     Uri.parse(uri),
     mode: LaunchMode.externalApplication,
   )});

setState(() {
 final account = session.accounts[0];
});
```

The dart code above calls the createSession() method of the connector, which is an object of the WalletConnect class. The chain id is selected (in this case 4 represents the Rinkeby Testnet, while the ethereum mainnet is 1) and a special url that connects the app with the wallet is created and launched. After the session is established, the account variable is set with the user account.

```
String nonce = user["nonce"].toString();
String msg = "I am signing my one-time nonce: " + nonce;
isDialogShown = true;
_showDialog(context);
String signature = await provider.personalSign(message: msg,
address: accountAddress, password: "test password");
if(isDialogShown){Navigator.pop(context);}
```

This code is used in the next step of the login process. A msg is produced using a randomly generated nonce received from the backend, and the personalSign() function is called so that the user can sign this message using their wallet. After the message is signed, it gets returned to the backend in order to verify that the user is truly in possession of their private key.

```
String data2 = decodedRsp["abi"];
int nonce = int.parse(decodedRsp["nonce"].toString());
data2 = data2.substring(2);
const Utf8Encoder encoder = Utf8Encoder();
List<int> value = hex.decode(data2);
Uint8List encodedData = Uint8List.fromList(value);
var tx;
isDialogShown = true;
_showDialog(context);
tx      =      await      widget.provider.sendTransaction(from:
widget.accountAddress,
   to: tokenAddress,
   data: encodedData,
   nonce: nonce,
   gas: 1500000);
if(isDialogShown){Navigator.pop(context);}
```

Finally the wallet is also used to sign transactions and send them to the blockchain. This is an example for the vote() function call on the NewToken contract. The transaction arguments are obtained from the backend, including the ABI and nonce, and are encoded into a transaction. The transaction is then sent to the wallet to be signed, and from there it is propagated.

2.3.3.2 API Calls

The frontend and the backend communicate using REST API calls.

```
Response rsp = await post(
       Uri.parse('http://'        +        widget.localIp        +
':3001/building/getPriceForTokens'),
 headers: <String, String>{
   'Content-Type': 'application/json; charset=UTF-8',
   'Authorization': 'Bearer ' + widget.authToken,
 },
 body: jsonEncode(<String, dynamic>{
   'building_id': buildingId2,
   'tokenAmount': 1,
   'tokenAddress': tokenAddress,
 }),
);
Map<String, dynamic> decodedRsp =json.decode(rsp.body);
String   price   =   (double.parse(decodedRsp["price"])      /
(pow(10,18)) ).toString();
```

There are many API calls, this example is for obtaining the price of a token. A post request is created and submitted to a specific route in the backend server, passing the authentication token in the header, and the information required in the body, in JSON format. After a response is obtained, its body is decoded into a map, and then the relevant variable (in this case price) is extracted from the map.

2.3.3.3 Dynamic & Conditional Rendering

Flutter is a very powerful tool, and one of its very useful features is being able to render items conditionally and dynamically. This is very useful to show users the information as it exists in the database, and to adapt the current user experience to the actual state.

```
 for(dynamic building in list) {
   print(building);
   final String name = building["name"];
   final String address = building["address"];
   final String buildingId = building["_id"];
                                             String          token            =
"0x0000000000000000000000000000000000000000";
   String rent = "0x0000000000000000000000000000000000000000";
   try{ token = building["token_id"]["address"];} catch(e){}
   try{ rent = building["rentContractAddress"];} catch(e){}
   buildingCard = Padding(
     padding: const EdgeInsets.all(8.0),
     child: Card(
       clipBehavior: Clip.antiAlias,
       shape: RoundedRectangleBorder(
         borderRadius: BorderRadius.circular(26),
       ),
       child: Column(
         children: [
           Stack(
             children: [
               Ink.image(
                 image: NetworkImage(
                   'http://placeimg.com/640/480/arch',
                 ),
                 child: InkWell(
                       onTap:  ()  => beforeBuySell(token,  rent,
buildingId),
                 ),
                 height: 240,
                 fit: BoxFit.cover,
               ),
               Positioned(
                 bottom: 16,
                 right: 16,
                 left: 16,
                 child: Text(
                   name,
                   style: TextStyle(
                     fontWeight: FontWeight.bold,
                     color: Colors.white,
                                                   backgroundColor:
Colors.grey.withOpacity(0.25),
                     fontSize: 24,
                   ),
                 ),
               ),
             ],
           ),
           Padding(
             padding: EdgeInsets.all(16).copyWith(bottom: 0),
             child: Column(
               children: [
                 Padding(
                         padding: const EdgeInsets.only(bottom:
8.0),
                   child: Text(
                     address,
                     style: TextStyle(fontSize: 16),
                   ),
                 ),
                 Text(
                   token,
                   style: TextStyle(fontSize: 12),
                 ),
               ],
             ),
           ),
           ButtonBar()
         ],
       ),
     ),
   );
   buildings.add(buildingCard);
   dev.log(decodedRsp.toString());
 }
 Navigator.push(context, MaterialPageRoute(builder: (context) {
       return  BuildingsPage(title:  "Dstate",  provider:
widget.provider,  authToken:  widget.authToken,  localIp:
widget.localIp,                              accountAddress:
widget.accountAddress,buildings: buildings);
 }));
}
```

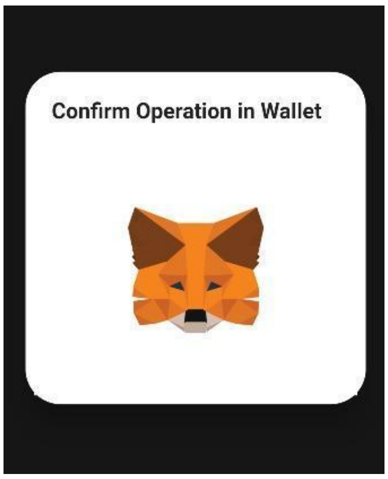
For showing the list of existing buildings to the user, the process is the one above. The widgets are of the Card class, and are created in a loop with the values obtained from the backend, which in turn have been fetched from the database. Each widget is created with different values, and also with the function that will allow users to tap on them to open the pages inside the building. These widgets are then stored in a list, which gets passed on to the next page, which then loads every widget inside a ListView widget and renders them on-screen.

Widgets can also be shown dynamically, making them appear and disappear from view depending on the state of the app (this is what is referred to as stateful). This is used in a few locations in the app, but the code is too spread out to show it as a code snippet. It is used in the token creation page, in order to dynamically load the TextFields relating to rent depending on if the building is for rent or not. It is also used in the create proposal page, in order to dynamically load the TextFields that need to be filled in for the selected kind of proposal.

2.3.3.4 Visual Elements

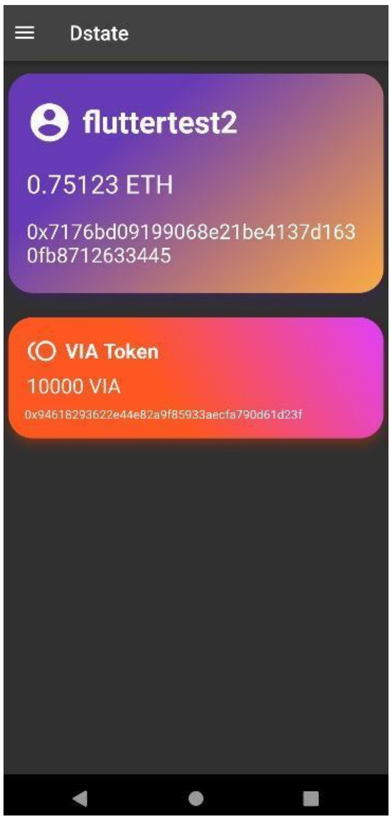
Visual elements are an important part of interacting with most software systems, and the way they are implemented can greatly affect the experience the user has.

Figure 7: Confirm Operation Widget



In the figure above, the Confirm Operation in Wallet widget can be observed. It consists of an alert that cannot be exited, a simple instruction to the user, and a Metamask animated icon. While this might be simple, this might be one of the most important choices of the visual design. Switching from one application to another can be confusing to users, but is necessary in the context of the application. This negative effect is limited by giving the user very clear instructions, and not allowing them to do anything else until they are finished. The metamask logo serves as a visual reminder of which application they need to open, while the looping animation reminds them that the application is waiting for them.

Figure 8: Portfolio Page



fluttertest2

0.75123 ETH

0x7176bd09199068e21be4137d163
0fb8712633445

VIA Token

10000 VIA

0x94618293622e44e82a9f85933aecfa790d61d23f

The figure above shows the user portfolio page. This is the first page they are greeted with when they open they log into the application, and provides useful information at a quick glance. The user can see their username, ether balance, and wallet address. They can also observe a list of the tokens they own, together with their quantities and addresses. The gradient color scheme offers some refreshing visuals while also being appealing and simple.

Figure 9: Building Card

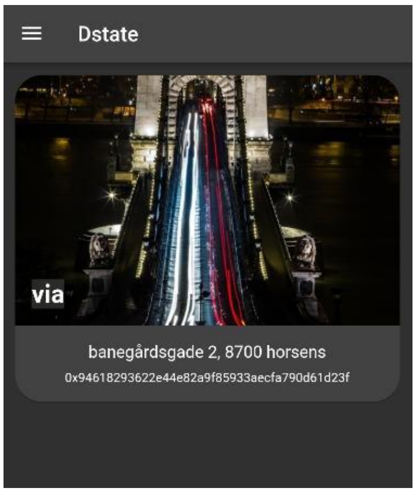
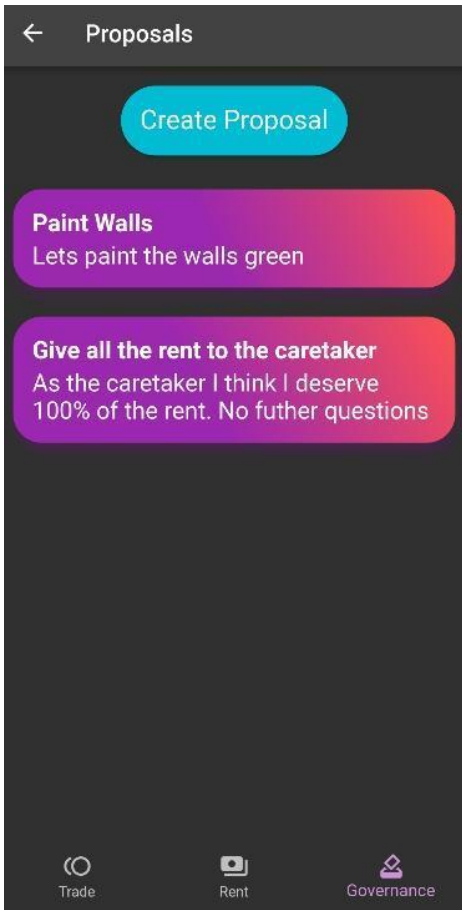

Dstate
via
banegårdsgade 2, 8700 horsens
0x94618293622e44e82a9f85933aecfa790d61d23f

Figure 9 shows how the user is provided with a view of the building list, having access to an image, name, address, and token address. This information is useful to provide information quickly while making it easy to distinguish between buildings. The card design is also elegant and functional.

Figure 10: Proposals Page


Proposals

Create Proposal

Paint Walls
Lets paint the walls green

Give all the rent to the caretaker
As the caretaker I think I deserve 100% of the rent. No futher questions

The proposals page interface displays a list of the current proposals with their name and description. This is done to allow more proposals to fit on-screen, and further details can be seen when accessing a specific proposal. The bottom navigation bar is also present in the Rent and Trade pages, and allows the user to quickly and comfortably navigate the screens related to a specific building. Finally it is relevant to note how many buttons and cards in the UI follow a rounded style. This is done because of multiple reasons, the main ones being that it is becoming a popular choice for material design, and also it is similar to a style Metamask uses. This makes the transition from the app to the wallet and back more seamless and less jarring.

2.4 Tests

In order to make sure the main functionality of the application was working correctly, a process of manual testing was followed. While a more standardised approach would have been ideal, this was sacrificed in order to reach the scope of the envisioned project. As this software product is a Minimum Viable Product, this is acceptable.

Through the testing process of different components and their integration, various undesirable behaviours were detected. Some of these were trivial, and promptly resolved, while others required in depth analysis and evaluation. Some notable examples include some bugs in the BuySell contract, which caused it to misbehave when putting tokens on sale, in a price decreasing order. This blocked any users from buying tokens.

The ethereum Ropsen testnet was also very useful in the testing process, allowing for the deployment of contracts in simple manner, in order to analyse their behaviour and interaction.

3 Business study

3.1 Methods

3.1.1 Business Plan

The business plan is one of the main tools to evaluate a business creation project. It is composed of many sections covering the project's foundations, its management, the marketing study, and the financial study. A business plan is not a fixed model, it must be adapted to the project according to its characteristics as well as to the expectations in terms of results of the analysis. The other models used for the analysis of the business part are in fact sections of the business plan. It is therefore a whole that allows one to have a global vision of the project and its analysis.

Table 3: Business Plan guideline

1. Executive summary	a) A brief summary of the key information in the business plan.
2. Idea & background **Your WHY**	a) Describe the problem b) Describe how you solve the problem c) Define briefly your customers d) Revenue - Explain how you are going to earn money – what are people paying for and how? e) People - Describe briefly the team behind this idea f) Passion – Highlight the passions in the team g) Explain your WHY – your deep why
3. Vision, mission, values & goals **G**	a) Your vision b) Your mission statement c) Your company values d) Your goals
4. Product & concept	a) Core product b) Add on products c) Customer values (features, benefits, values) d) Pricing e) Intellectual property rights - IPR f) Development plan - Outlining a future product portfolio g) Production (own production or outsourcing)
5. The MACRO environment – regulations, etc. **G**	a) Political & legal factors b) Economic factors c) Ecological/physical environment (global warming, etc.) d) Social/cultural factors e) Technological factors

6. The MICRO environment	6.1 The market potential a) Customers & buying behaviour b) Market size & growth rates c) Trends 6.2 The industry d) Industry structure & environment e) Entry barriers f) Competitors & your competitive advantages g) Substitution h) Distributors i) Suppliers
7. SWOT	a) Strengths b) Weaknesses c) Opportunities d) Threats
8. Sales & marketing	a) Your brand - your brand story, your brand values b) Sales & distribution channels c) Sales activities d) Core messages & positioning e) Marketing mix activities
9. Management & Organisation	a) Legal structure and ownership b) Management c) Board & advisors d) Partnerships e) Key activities f) Key resources
10. Action & Development plan	a) Goals b) Milestones c) Actions
11. Control & evaluation	a) Control & evaluation of your marketing activities, your business idea, your products, etc.

12. Risk analysis	a) Take the most serious and the most likely risks into consideration and think about what you could do to avoid them and how you could react if they become real.
13. Budgets	a) Establishing budget b) Operating budget c) Cash Flow budget d) Funding (part 8)
14. Appendix	

Source: VIA Entrepreneurship, Lene O. Sørensen, February 2022

The business plan template above is the one that is used for this project. It comes from the VIA Entrepreneurships class. The template used is complete and contains sections that are not useful for this project given the list of sub-problems and delimitations defined in the project description. Therefore the sections related to management, risk analysis and development plan are not used.

3.1.2 Financial plan (budget realisation):

The financial plan is an analysis that aims to evaluate the financial stability of a company. In the context of this project, it takes the form of a budgetary estimate that allows to account for the first years of the project's life. It should be noted that this analysis is only an estimate and allows only with the help of data from research on costs to report on how the finances of the company should be during the first years. It is an analysis that allows us to understand the growth and the different spending choices of the company.

The models used for this budgeting are the establishment budget, the operating budget and the cash flow budget. The templates used for this analysis are from the VIA entrepreneurship course.

3.1.2.1 Establishment budget

The purpose of the establishment budget is to organise the different costs related to the creation of the company and its installation. These are costs that occur before the start-up of the company and allow the smooth running of the company.

3.1.2.2 Operating Budget

The operating budget is a budget that accounts for the various revenues and expenses anticipated by the company during the first five years. It also includes three different scenarios that allow the project to be better prepared for possible management difficulties. Indeed, although the estimate of the receipts and the costs made upstream for the realisation of the budget is the most rigorous possible, it remains an estimate and the reality can thus not concord with the budget. Therefore, the use of three scenarios, including a realistic business development scenario, a pessimistic scenario (revenues divided by three) and an optimistic scenario (revenues multiplied by three) allows one to prepare for possible difficulties.

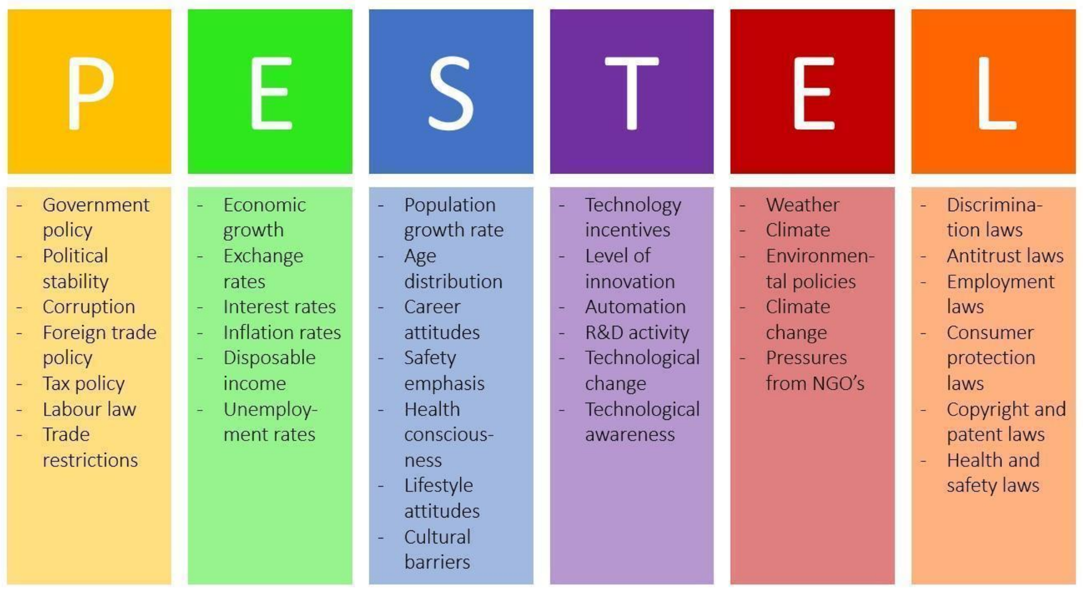
3.1.2.3 Cash Flow budget

The cash flow budget is similar to the operating budget but is focused on the company's cash flow. It covers the first year of the company's life, which is one of the most critical in terms of risk. This budget is used to estimate whether the company's cash flow will be in difficulty or not. To do this, it is necessary to consider the different flows, revenues and costs.

3.1.3 PESTEL Model

The PESTEL model is a marketing tool to analyse the macroeconomic environment of a company or a project. In this model, the environment is broken down into six different aspects: political, economic, social, technological, ecological and legal. By analysing the environment in this way, it is possible to better understand the strengths and weaknesses of the company related to this macroeconomic environment.

Figure 11: PESTEL Model

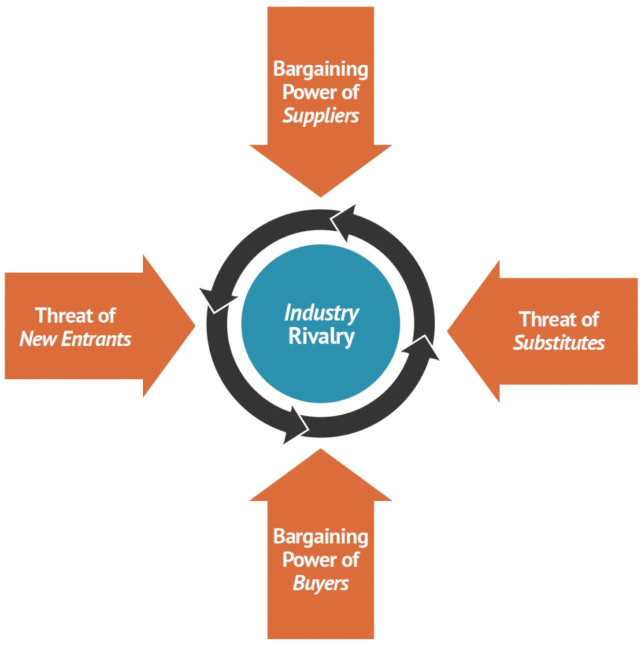


P	E	S	T	E	L
- Government policy - Political stability - Corruption - Foreign trade policy - Tax policy - Labour law - Trade restrictions	- Economic growth - Exchange rates - Interest rates - Inflation rates - Disposable income - Unemploy-ment rates	- Population growth rate - Age distribution - Career attitudes - Safety emphasis - Health conscious-ness - Lifestyle attitudes - Cultural barriers	- Technology incentives - Level of innovation - Automation - R&D activity - Technological change - Technological awareness	- Weather - Climate - Environmen-tal policies - Climate change - Pressures from NGO's	- Discrimina-tion laws - Antitrust laws - Employment laws - Consumer protection laws - Copyright and patent laws - Health and safety laws

3.1.4 Porter's 5 forces

The five forces model of porter model is a marketing tool to analyse the macro-economic environment of the company. It also allows us to analyse the state of the competition on the market.

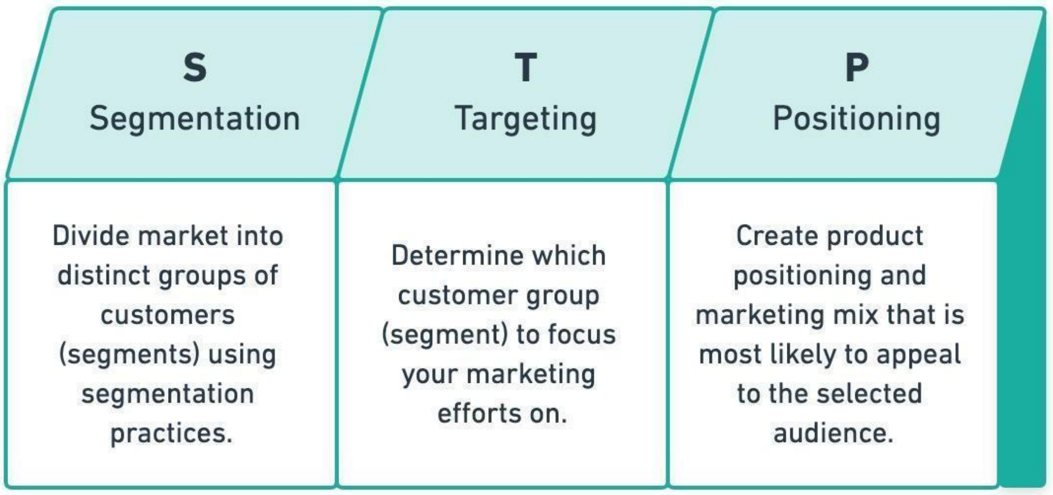
Figure 12: Porter's 5 Forces Model



3.1.5 STP model

The STP model is used to segment a market, target the best segment and position the company according to this segment. It is a positioning tool that takes place within the market analysis and before the definition of the marketing strategy. It is useful to better understand the market and respond to the situation with the best positioning, allowing the company to increase its results.

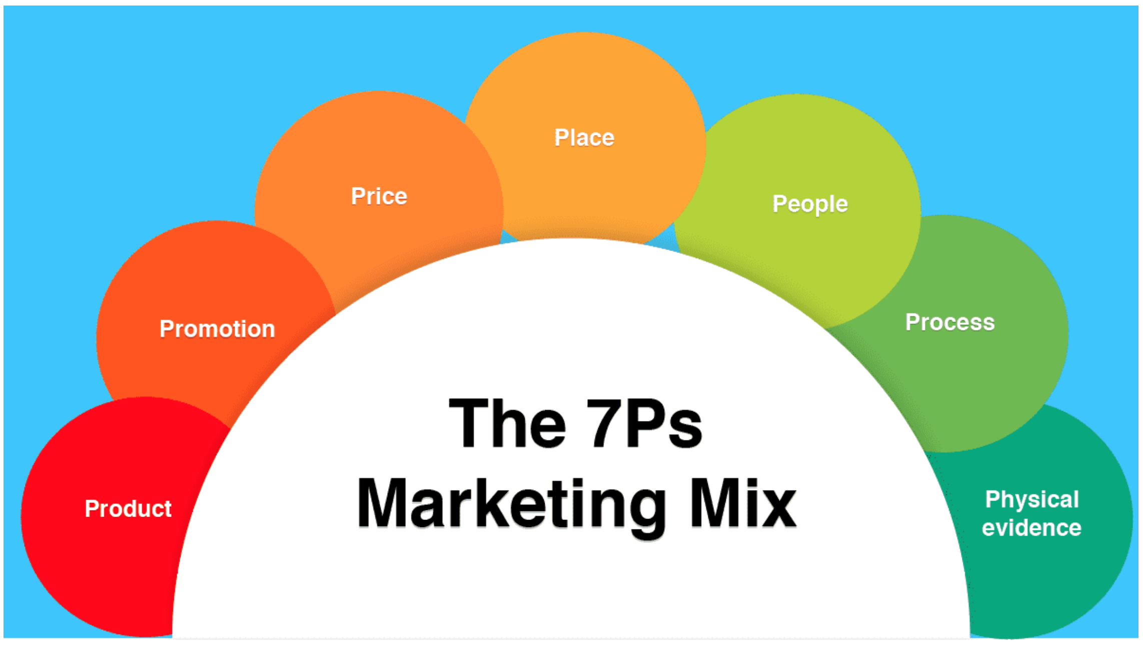
Figure 13: STP Model


S
Segmentation
T
Targeting
P
Positioning
Divide market into distinct groups of customers (segments) using segmentation practices.
Determine which customer group (segment) to focus your marketing efforts on.
Create product positioning and marketing mix that is most likely to appeal to the selected audience.

3.1.6 7P’s Model

The 7p's model is a tool to define a marketing strategy in line with the characteristics of the company and its market. It is the last step of the marketing analysis. The study is based on the other marketing analysis seen previously. There are two versions of this model, one with only four sections, and this one with 7 sections. This is a model that covers more factors than the original version. Within the framework of the Dstate project, the study of the 7p's allows to define a marketing mix relevant to the goals and objectives of the company.

Figure 14: 7P's Model



3.2 Results/ Business Plan

3.2.1 Executive summary

In an era where technological advances are becoming more and more numerous, there are still markets, such as real estate, where the evolution of solutions used seems to be slower, leaving inequalities and barriers for some market players. That's where Dstate comes in, offering new solutions to make access to the real estate market easier for everyone. The goal of the project is to change the way people access and interact in the real estate market. In order to carry out this project, it is necessary to conduct a complete analysis of the different components of the project, such as marketing and finance. This is how the business plan becomes useful.

3.2.2 Idea & background

3.2.2.1 Description of the problem

The problem to address is the lack of accessibility to the real estate market in terms of capital. This market has historically and to this day had a large barrier to entry, preventing many people from participating in it. In other words: How can small retail investors participate in the real estate market?

3.2.2.2 How Dstate solve the problem

The solution provided by Dstate is to reduce the cost of entering the real estate market. For this, the use of the recent blockchain technology is an asset. Indeed, thanks to this technology, it is possible to create, exchange and divide many services or assets. The case of real estate is not excluded, and it is already possible to sell or buy a property via this technology. However, it would be of little interest to use blockchain to simply buy or sell a property. What is interesting on the other hand is the possibility of being able to fragment these goods in the form of tokens. A token coming from the division of a property represents then a fraction of the global property.
In this way, instead of having to buy a whole property to start investing in real estate, it is now possible to buy a fraction of a property. In addition, blockchain technology brings other possibilities such as a decentralised and secure management of real estate. This can facilitate the management for the owner(s) as well as for the tenants. The tenants also benefit from this system. It also means that a tenant who does not have enough capital to become an owner can use his savings to buy a share of the property in which they reside. This would have two consequences. The first one is that they would have access to savings that could be more advantageous than a traditional bank investment. And the second being that part of the rent they pay will be directly transferred to them representing the part of the rent that is due to them, reducing their expenses.

3.2.3 Description of the customers

Our customers can be firstly divided into three categories:
-Property owners
-People interested in acquiring one or more properties
-Tenants whose landlords use Dstate services

Property owners

Property owners can be interested in Dstate services for different reasons. The first is to generate liquidity without selling the entire property. For a homeowner, acquiring liquidity can be useful when major purchases are to be made, such as an unexpected daily expense, but also such as financing a new property. Why would it be interesting to sell only a part of a property? The main reason is that some properties are more valuable than others. For example, an owner having bought a property with high profitability or whose location allows it to gain in value over time has no interest in selling the property and losing control over it. Thus it will be preferable to sell a part of the property which will be possible to repurchase more easily later on.

People interested in acquiring one or more properties

Our largest potential customer base is represented by people looking for a way to invest in real estate. These investors are looking for a way to invest in relatively safe savings and have a barrier to entry that does not represent an obstacle for them.
These investors may also have a larger capital that would be sufficient to purchase a property in the traditional way. In this case, the advantage of Dstate is that they can diversify their investment by buying tokens of multiple properties, thus reducing the risk of their investment.

Tenants whose landlords use Dstate services

The last customers are the tenants of properties that are already tokenized. If the owner of the property in which they reside decides to use Dstate, they will then begin using the Dstate management service.

3.2.3.1 How do Dstate generate revenue

To generate revenue, Dstate applies fees to various operations on the platform.

The tokenization of a property:

When a property is tokenized by Dstate the tokens are then put on the market. It represents a workload to verify the authenticity of the property, perform the operations via the blockchain and therefore represents a cost that is passed on to the person wishing to perform this process. There are therefore fees that the owner of the property must pay.

The purchase of property tokens:

When a customer wants to buy a token on the marketplace, they are charged a fee for the transaction. These fees are used in part to pay for the costs of using the blockchain, but also for Dstate to generate revenue.

Subscription for renting management services:

If the owner(s) of a property on the platform wishes to use the management tool provided by Dstate, it will incur a fee in the form of a service charge to be withdrawn each month from the revenue generated by the property.

3.2.3.2 The team

The Dstate team is composed of three members who are the project leaders. There is Eric Marti Haynes and Gloria Desideri who compose the software team, as well as Gaspard HUGOT managing the commercial and financial part.

3.2.3.3 Motivation

The main reason why the Dstate project was conceived is to enable a real profound change in the real estate market. It is about having a different vision and approach to conceive what real estate investment is. Indeed, although the investment aspect of the market is the most emphasised, real estate is above all living, working or cultural places. This means that everyone is linked to it with more or less involvement. There are two aspects on which Dstate wants to have an impact. The first is market accessibility: allowing everyone to access and invest in real estate. And the second is to make all market-related transactions more secure.

3.2.4 Vision, mission, values & goals

3.2.4.1 Vision

"We believe that the future of the real estate market is in the new technologies related to blockchain"

Dstate's vision is positive and optimistic and is based on a simple observation: the real estate market is a fossil, and it must evolve in line with the rest of the world. With a keen eye for new technologies that promise to revolutionise the modern world, Dstate believes that solutions are possible to improve and reshape the real estate market.

3.2.4.2 Mission statement

Dstate is dedicated to offer new ways to be part of the real estate market, including more accessibility, security, and independence.

3.2.4.3 Company values

Independence:

The value of independence carried by the Dstate company means that the company seeks to give customers greater autonomy, a way for them to choose how they wish to participate in the real estate market. Within the investor framework the players are sometimes limited due to centralization, most often low capital individuals. This is why Dstate seeks to change this by offering more opportunities to investors.

Transparency:

Transparency is an important value for Dstate and generally common to most companies in the crypto currency world. Indeed, it is first and foremost a desire to be honest with the customer and allow a certain amount of trust to be built up. Transparency is also an aspect of the companies using blockchain which is partly induced by this technology because all the operations carried out on a blockchain are public and thus transparent.

Liberty:

The value of freedom refers to the freedom of clients but also the freedom of people in general. For example, Dstate will not have offices at first, which allows employees to work remotely, offering more freedom in managing their time. In another context, the services offered by Dstate are accessible by all, whether through the company or directly by accessing the features via the Blockchain.

Security:

Security is an important value when it comes to investment and large amounts of money are involved. Dstate is committed to this value and seeks to make its services as safe as possible.

3.2.4.4 Your goals

Table 4: Goals and Objectives

Goals and Objectives chart	
Goals	**Objectives**
Be sustainable	Reach the break-even point
Improve accessibility to the market	Make our service effective
Create a strong and loyal community	Acquire a large user base and fidelyze it
Generate Profit	Go beyond the break-even point

3.2.5 Product & concept

3.2.5.1 Core product

Dstate's core product is the real estate tokenization service. This service is open to anyone who can prove ownership of a property.

Dstate's tokenization service is composed of different steps. First, the owner who wants to use the service registers on the platform and passes a first step of identity verification. In a second step, he proposes an asset to be divided into tokens. For this, a second verification phase is required, including the analysis of the property documents of the proposed asset. Finally, if all the preliminary verifications are positive, the property is divided into tokens which are then put on sale on the platform.

When someone is purchasing a token, it has consequences. Firstly, the buyer acquires a part of the property ownership in the proportion of the property tokens that he owns. This gives him/her rights to the income from the property and to the management of the property according to the contractual terms of the property. The second is that the previous owner is paid the current value of the tokens he sold, thus providing him with liquidity.

3.2.5.2 Add on products

Dstate add on Product is the property management service which is a complement to the tokenization. Indeed, the fact that there are several owners can make the management of the property more complicated. Whether it's the distribution of the rent or the decision-making process such as renovations or rent increases.

There are two main cases. Firstly, it is possible that one person owns most of the shares, in which case he or she has the power to make decisions. The second case is that there is no majority and that the distribution of the power of proposal and decision must be established in the contract of tokenization. The person who uses the Dstate service for his property can then choose to keep the control of his property and thus be the person who makes the decisions. But he can also choose to make the management democratic. Thus, each owner will have a power of decision to the extent of its possession of tokens of the property. It is in this case that the Dstate management tool is used because it automates all the votes, proposals, and transactions.

3.2.5.3 Pricing

As seen earlier in the section on how Dstate generates revenue, there are three sources of revenue for the company. The prices set are broken down as follows:
Tokenization price: 1000€ + 0.5% of the total value of the asset.

Comment: The fees for tokenization can be suppressed for the first six months of the launch of the company considering that it could be a barrier to the growth.

Fees on token purchase: 0.5% of the total amount

Management service fees: 1% if the rent (Deduce before the distribution to the investors)

These prices are based on two criteria: to be able to generate a sufficient income for the company to be sustainable but also not to have a pricing too high that would scare away potential customers.

3.2.5.4 Production

In terms of production, the company only produces a service. This service is based on a computer tool that must first be developed.

It is envisaged that the development of Dstate's computer system will be subcontracted, and that its maintenance and possible improvement will be managed by a team of software developers. The subcontracting for the realisation of this system represents a cost which is estimated at 50.000€. Indeed, in France for example, the median salary of blockchain developers per year is €50,000. The development time for two developers is 6 months which amounts to 50,000€ of development costs.

As for the costs of maintaining or improving the service, it is necessary to consider hiring a blockchain developer in a sustainable way. The cost would therefore be €50,000 per year. Cost to be included in the budgetary estimation.

3.2.6 The MACRO environment

3.2.6.1 Political & legal factors

The political and legal environment in which Dstate operates is quite complex. Indeed, the company and its activities are subject to regulations related to the real estate market, cryptocurrency and blockchain technologies. Moreover, these regulations may differ depending on the country in which the service is sold. There are currently no legal limitations on the principle of tokenizing real estate and then selling these tokens. The same goes for cryptocurrency exchanges and markets that allow their exchange. However, the European community, because it is mainly in Europe that Dstate plans to be established, is beginning to look into the subject. Indeed, already today with the project MICA (Market in Crypto Assets) regulations could be put in place soon to create a framework for the use of different tokens. This indicates a desire on the part of the European authorities to provide the best possible framework for the sector. The consequence of such regulations would be greater consumer confidence, but also, conversely, a risk of reducing the possibilities of the market.

3.2.6.2 Economic factors

The economic environment is relatively favourable to the project. First of all, the real estate market is a stable market with relatively constant growth. Indeed, despite the crisis of 2008 and the pandemic of 2020, the real estate market has seen its prices increase. On the other hand, the crypto-currency and blockchain technology market is booming. The only major risk is the youth of this technology which can worry potential investors and also experience more frequent crisis episodes.

3.2.6.3 Ecological/physical environment

The ecological and physical aspect of the environment in which Dstate operates does not have much impact. The only point to note is that challenges to the energy consumption of this technology are beginning to emerge. However, this remains insignificant and moreover, the market players are working to reduce this consumption to make the technology more sustainable.

3.2.6.4 Social/cultural factors

First, demography is an important measure for understanding the social and cultural aspect of the environment. In Europe the active population (between 15 and 64 years old) represents 243 million people. Secondly, the fact that in Europe the population has a strong tendency to save and invest is an asset for the project. Finally, the general attraction for new technologies, especially among 18-35 year olds, makes the environment favourable to the development of the project.

3.2.6.5 Technological factors

The technological aspect of the environment is one of the most important given the field in which Dstate operates. In general, since the arrival of the Internet, the technological advances have not ceased to be more and more rapid. The world is now totally connected and this represents an asset for a digital project. Moreover, the technologies related to blockchain are democratising all over the world and the number of people who are interested in it is constantly increasing.

3.2.7 The MICRO-environment

3.2.7.1 The market potential

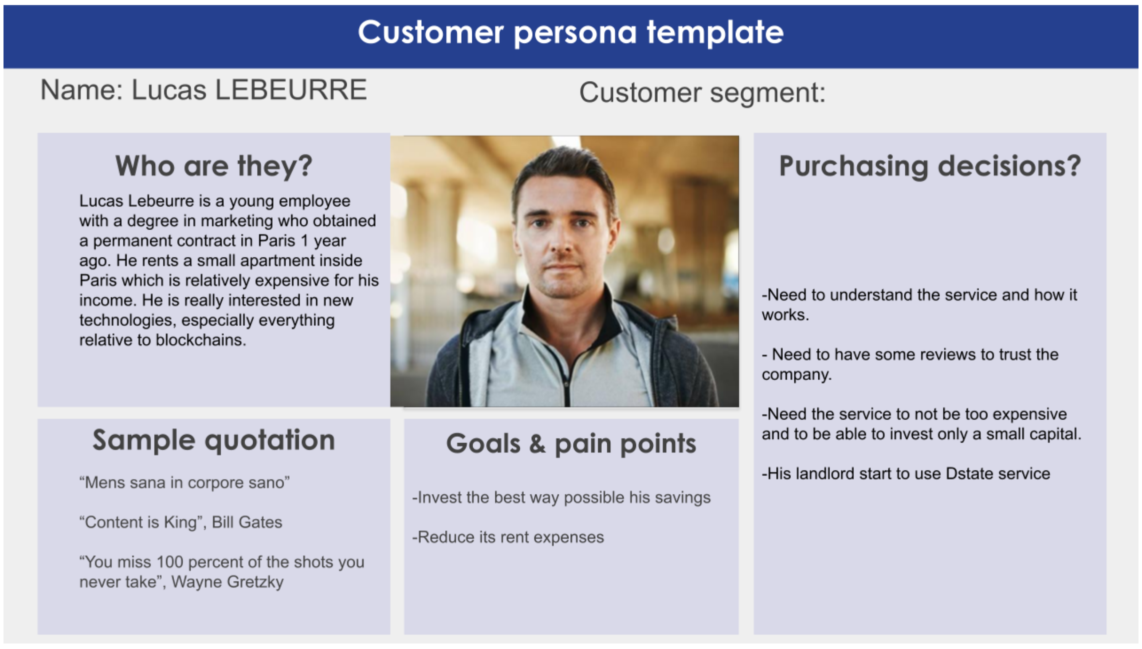
Customers

Potential customers:

The potential users of Dstate are therefore grouped into three categories:
-Property owners
-People interested in acquiring one or more properties
-Tenants whose owners use Dstate services
In the context of the consumer study, although Dstate addresses anyone who owns property or is a tenant. In other words, the majority of the working population. It is important to distinguish the fact that homeowners are generally older people, between 35 and 65 years old. In addition, even fewer homeowners are familiar with the blockchain world.

Personas and values proposition:

Figure 15: Persona n°1



Customer persona template

Name: Lucas LEBEURRE

Customer segment:

Who are they?

Lucas Lebeurre is a young employee with a degree in marketing who obtained a permanent contract in Paris 1 year ago. He rents a small apartment inside Paris which is relatively expensive for his income. He is really interested in new technologies, especially everything relative to blockchains.

Purchasing decisions?

-Need to understand the service and how it works.

- Need to have some reviews to trust the company.

-Need the service to not be too expensive and to be able to invest only a small capital.

-His landlord start to use Dstate service

Sample quotation

“Mens sana in corpore sano”

“Content is King”, Bill Gates

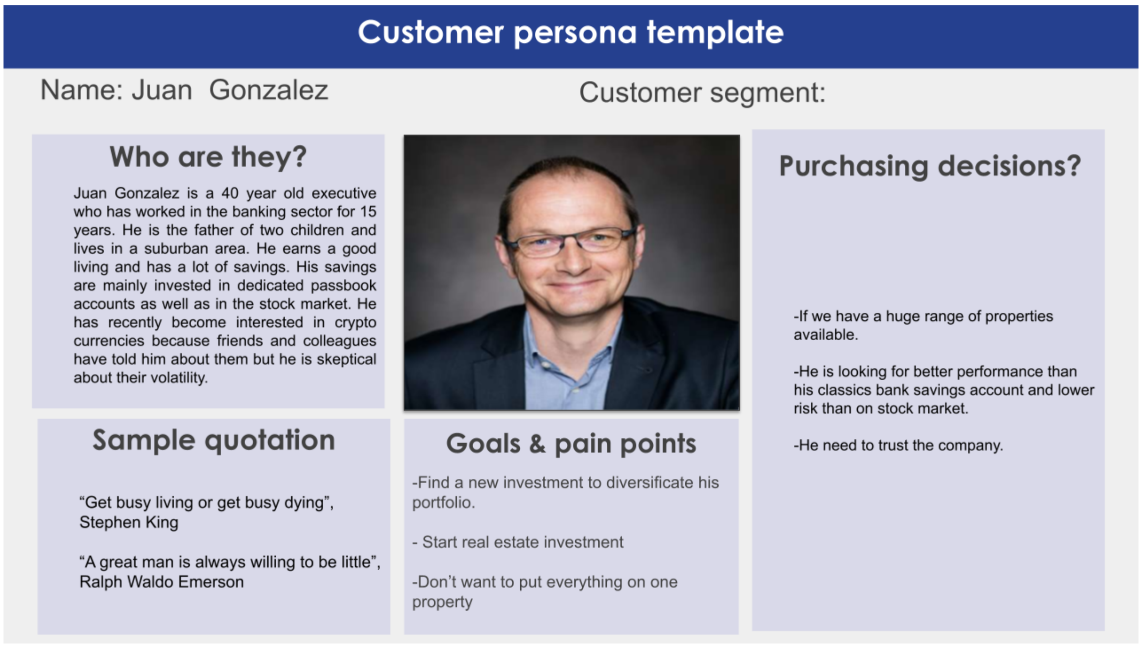
“You miss 100 percent of the shots you never take”, Wayne Gretzky

Goals & pain points

-Invest the best way possible his savings

-Reduce its rent expenses

Figure 16: Persona n°2



Customer persona template

Name: Juan Gonzalez

Customer segment:

Who are they?

Juan Gonzalez is a 40 year old executive who has worked in the banking sector for 15 years. He is the father of two children and lives in a suburban area. He earns a good living and has a lot of savings. His savings are mainly invested in dedicated passbook accounts as well as in the stock market. He has recently become interested in crypto currencies because friends and colleagues have told him about them but he is skeptical about their volatility.

Purchasing decisions?

-If we have a huge range of properties available.

-He is looking for better performance than his classics bank savings account and lower risk than on stock market.

-He need to trust the company.

Sample quotation

“Get busy living or get busy dying”, Stephen King

“A great man is always willing to be little”, Ralph Waldo Emerson

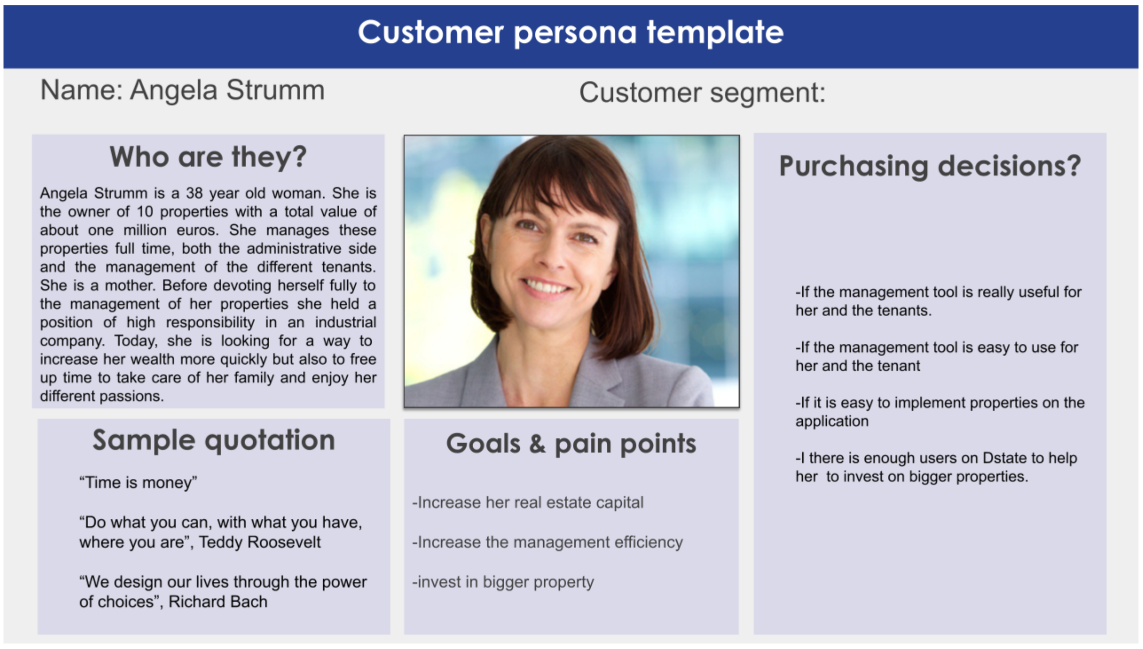
Goals & pain points

-Find a new investment to diversificate his portfolio.

- Start real estate investment

-Don’t want to put everything on one property

Figure 17: Persona n°3



Customer persona template

Name: Angela Strumm

Customer segment:

Who are they?

Angela Strumm is a 38 year old woman. She is the owner of 10 properties with a total value of about one million euros. She manages these properties full time, both the administrative side and the management of the different tenants. She is a mother. Before devoting herself fully to the management of her properties she held a position of high responsibility in an industrial company. Today, she is looking for a way to increase her wealth more quickly but also to free up time to take care of her family and enjoy her different passions.

Purchasing decisions?

-If the management tool is really useful for her and the tenants.

-If the management tool is easy to use for her and the tenant

-If it is easy to implement properties on the application

-I there is enough users on Dstate to help her to invest on bigger properties.

Sample quotation

"Time is money"

"Do what you can, with what you have, where you are", Teddy Roosevelt

"We design our lives through the power of choices", Richard Bach

Goals & pain points

-Increase her real estate capital

-Increase the management efficiency

-invest in bigger property

Market size

To define the market size, it is necessary to calculate how many people represent the three categories of potential consumers.

First, it must be admitted that the third category, the tenants whose landlords use Dstate, are by and large not potential users that can be sought out, but rather users induced by their landlord's membership.

That being said, and in order not to make an overly optimistic estimate, it is prudent to consider only one tenant per landlord using Dstate. Therefore, the result will be a low estimate.

Second, for the group of customers looking to invest in real estate. It is difficult to find European-wide data on customer interest in real estate investment. However, given that Dstate uses blockchain and the use of its services leads to the use of crypto currencies or at least encourages it. It is realistic to say that it is mainly European crypto-currency users or potential crypto-currency users who will be part of this target group. In Europe, the global average adoption rate sits at approximately 23 percent. Additionally, among non-users, 35% have doubts about the security it represents and

34% say they don't know how to buy or hold cryptos. These non-users are potential customers because the brakes that prevent them from using crypto currencies may disappear in the coming months, years.

Finally, the number of owners of rental properties in Europe is difficult to estimate because the only known data are the share of the population owning their homes (70%) and the share renting their homes (30%). Therefore, in the context of the study of potential customers, another less precise method is to be used. Assuming that France is a country like the main European countries where Dstate will be launched, and that 7.3 million households in France are multi-owner, it is possible to conjecture that this represents the share of homeowners who would be likely to use the Dstate service. So, the ratio must be calculated between these 7.3 million compared to the number of households in France, that is 29.2 million.

Calculation: 7.3/29.2*100 = 25

In France, 25% of the households are therefore multiple owners. It is this figure that will be used with the European active population in the calculation of the market size.

Calculation (in millions):

(243*0.23) +243*(0.35+0.34) = 223.56

There are 243.56 million potential customers for the group of customers looking to invest in crypto and real estate.

243*0.25 = 60.75

There are 60.75 million potential customers in the group of customers that are landlords who rent properties.

This number of potential customers may seem unrealistic, but these are just the people that Dstate might touch. In the case of a company offering investor and blockchain-related services, it is certainly impossible to reach everyone. However, a large market size is still an advantage because even if you reach a very small portion of that market, it will still represent many customers.

Trends

As far as market trends are concerned, overall the market is constantly increasing. First of all, the real estate market is constantly increasing, even though it is only slight. And secondly, because the market for crypto-currency and blockchain related services has been growing steadily for the last 5 years.

3.2.7.2 Porter 5 forces

Competition in the industry

This part refers to the number of competitors on the market. After doing researches, it is established that Dstate have few competitors. There are only five competitors that exist on the market : Stobox, RealT, digishare and Solidblock, Omni PSI and. The first important point to note is the limited number of competitors. Indeed, the market being young or even new, few are those who have already developed a company similar to Dstate. This is an advantage because there is less competition and it is easier to stand out.

The second important point is related to the service offered by these competitors. These three competitors do not offer exactly the same service as Dstate. While they do offer a marketplace of tokens from tokenized assets, the tokenization service offered to the asset owner is different. Indeed, Dstate offers a service that allows any owner to offer his property to tokenization in a simple and fast way. On the other hand, Dstate's competitors offer a consulting service that takes more time and therefore leads to a smaller number of properties on the market.

Potential of new entrants into the industry

About the potential of new entrants into the industry, because the market is new and there are few competitors, the risk of potential new entrants is high. It is possible to divide these potential new entrants into two groups because although there is a high risk of new entrants, they do not all have the same advantages.

The first group, the least dangerous in terms of competition, is that of the new players who, like Dstate, would be new projects without large existing capital and with fewer means to differentiate themselves from Dstate.

The second group is made up of large companies already existing on the crypto-currency market like Binance and FTX or already present on the real estate market. The latter could seek to diversify and offer a real estate tokenization service and would therefore become competitors for Dstate. This second group represents a bigger risk because the competing company would already have a lot of capital, customer base and brand image.

Power of suppliers

Dstate does not have any suppliers since the company offers an IT service. The only vendor that can be considered is the development company that is outsourced during the development tokenization process. However, this is a one-time operation that takes place at the launch of the project. It is therefore not relevant to take it into account in the analysis.

Power of customers

The bargaining power of customers in this case is very low, for several reasons. First of all, this is a service that companies offer. Therefore, customers do not have much choice. Secondly, during the sales process, the customer does not meet the different members of the company, which prevents negotiation. Finally, the service is mainly for individuals and not for other companies, so it is rare to try to negotiate in this kind of case.

Threat of substitute products

The risk of substituting the product proposed by Dstate is mitigated. Indeed, potential Dstate customers are looking for several criteria, namely low cost of access, security and a certain rate of return. However, some substitutes, such as traditional real estate investments, may represent a risk if the potential customer decides to abandon certain criteria. More importantly, there are similar investment systems that do not use blockchain. These substitutes have the same advantages in terms of market entry cost and return. However, they do not allow easy access to liquidity. Also, it should be noted that these are substitutes only for the investment principle. As for the liquidity advantage that Dstate's service offers to owners and the management solution for properties held by token owners, there are no substitutes.

3.2.8 SWOT

Table 5: SWOT Analysis

Strengths	**Weaknesses**
Innovative service Easy to use service Secure process	Complexe technology Hard to understand for customers Need a strong customer mass
Opportunities	**Threats**
Recent market Diversification huge potential customer mass Few competitors	Potential new competitors Cyber-attack risks

3.2.8.1 Strengths

Innovative service:

One of the major strength of Dstate is that the service that it propose is innovative. Indeed, even if some competitors are already proposing similar services, Dstate is bringing a solution to make it easier for the customers. This is a strength because the customers can really find a new solution to their needs.

Easy to use service:

The process of the service is made in a way that users can have a good and easy experience. The simpler it is the better. The easy-to-use aspect of Dstate is a strength because it permits to keep a bigger part of the leads that are dragged to the website and so increase the customer base, increasing the revenus and the stability of the company.

Secure process:

The security offered by the utilisation of blockchain is a strength for the company considering the fact that a lot of customers can be afraid of putting their money on new or unknown web-site. With this technologie the user can be reassured. This way, Dstate will lose less leads because of security. Also, it is a strong marketing advantage that can be used against competitors.

3.2.8.2 Weaknesses

Hard to understand for customers:

The complexity of the technology used by Dstate to tokenize and manage the multi-owning systeme can also be hard to understand for customers. This can be a weakness, scaring potential customers to invest on the platform. To reduce the potential negative effects of this weakness, it is necessary to focus on the transparency of the process and make it easy for users to find information on the site that will help them understand how the service works.

Need a strong customer mass:

The main weakness of Dstate is the need for a large user base to operate. The principle of tokenization induces the need for investors to buy the tokens. In case the number of users is too low, for example not enough owners to offer goods on the platform or not enough investors to buy tokens, it would ruin the experience for all users. The launch of the company and its first months or years are therefore a decisive moment, the one that carries the most risk. If the company does not manage to exceed the minimum number of users required for the service to function properly, the project will be frozen.

3.2.8.3 Opportunities

Recent market:

A good opportunity for Dstate is the fact that the market in which it operates is new, if not totally new. Although this market is actually the union of two existing markets, the real estate and crypto markets, it is still an opportunity in terms of competition and diversification.

Diversification:

Dstate offers three services on its platform that are all linked and can be considered as a single service bundle. However, given the ever-evolving technology used, there are great opportunities to diversify by adding new services to the Dstate catalogue. These new services could complement the existing ones or stand out while remaining in line with the company's values and objectives.

huge potential customer mass:

Another opportunity for the project is the large pool of potential customers. Indeed, as it is detailed in the market study, the potential customers are numerous as the service is able to interest a wide variety of people. This represents an opportunity to grow Dstate's customer base quickly and not suffer from a loss of momentum as the company expands.

Few competitors:

Finally, as mentioned earlier, Dstate has little competition because of the market in which it operates and the service it offers. This represents an opportunity because it means less effort for the company to distinguish itself or to fight on the marketing level against its competitors. These efforts can be placed in other actions participating in the development of the company.

3.2.8.4 Threats

Complex technology:

The technology used by Dstate and its tokenization service is a recent technology which can represent a weakness. Indeed, although the process works, there may very well be bugs over time, development difficulties and therefore increased costs. This represents a weakness.

Potential new competitors:

The youth of the market and its expansion also brings its share of risks such as the risk of new competitors on the market. This risk is common to all companies, but in this case, large companies with more capital and experience could choose to diversify by creating a branch dedicated to the tokenization of real estate. This represents a significant risk because Dstate as a new company does not have the financial and marketing means to fight against this kind of competitor.

Cyber-attack risks:

Finally, a low risk is always present in the digital world and even more so when large amounts of capital are at stake, the risk of computer attacks is present. Indeed, despite the security provided by technology, the risk of theft is present and fighting against it has a cost. Security represents a financial cost but also a cost in terms of image. Indeed, the image of the company could be tainted and the confidence of the customers and the potential customers would be lost.

3.2.9 Sales & marketing

3.2.9.1 Segmentation, Targeting and Positioning

Segmentation

When using an STP model, the first step is to identify and create customer segments. Various criteria are used for that.

Demographic:

From a demographic point of view, although potential clients are between the ages of 15 and 65, real estate investment is of more interest to 25–40-year-olds. In France, for example, the average age of the first real estate purchase is 31. It should also be noted that this advanced age is partly the result of a high barrier to entry into the market. With Dstate's service, this age could be reduced.

In addition, Dstate has three categories of potential customers: Property owners, investors, and tenants whose property is tokenized. Tenants are not a target because they are customers induced by the owners on the platform. The two categories to differentiate in terms of age are investors and owners. As far as investors are concerned, there is no precise age range because the minimum investment on Dstate is low and does not represent a break for a young segment of the population. An age range for investors from 15 to 65 years old can be established.

On the other hand, for the owners who wish to submit a property on the platform, it should be considered that few young people have real estate assets. In this case, the segment would be composed of people between 30 and 65 years.

It should also be noted that Dstate's service uses new technologies, is accessible only on the Internet. This reduces the project's ability to reach an older population that may not be familiar with new technologies and even less with the notion of blockchain. This reduces the target age range of owners to 30-50 years.

Geographic and online presence:

For the segmentation from a geographical point of view, the main information that could be used is the difference in the proportion of people who own property according to the countries of the European Union. Indeed, in the Eastern European countries the percentage of the population which owns his property is more important. Moreover, the real estate market in the richest and most developed countries of the European Union is more important and more dynamic. It is therefore logical to privilege a segment of the European territory composed of developed countries, those of the West and the North, as well as the Capital of the other less developed countries.

In terms of online presence, the most appropriate segment is that of people using social networks such as reddit and twitter. These are generally the networks most used by those interested in crypto-currencies and blockchain.

Finally, in terms of employment, it should be considered that people whose job is related to finance, investment or IT development are more likely to be interested in the project.

Psychographic:

Dstate must target the part of the population whose values and interests are in line with what the company offers. In terms of values, people who are close to values such as ambition, a spirit of discovery and dynamism are more likely to be interested in the project. In terms of interests, people interested in finance, real estate, investment and new technologies.

Conclusion:

Table 6: Segmentation Table

Segments	Age	Geographic Area	Interests
Owners	30-50years old	Western and Northern Europe, Eastern Europe important cities	Real estate, investment, new technologies
Young Investors	15-29 years old	All Europe	Investment, blockchain, real estate
Older investors	30-50 years old	Western and Northern Europe, Eastern Europe important cities	Investment, real estate, finance, new technologies, blockchain.

8.1.2. Targeting

Table 7: Targeting Table

Segments		Main owners	Young Investors	Older investors
Measurability	What is the size of the segment?	60.75 million	Around 75 million	Around 100 million
	Purchase behaviours	Analytical Buyer	The Driver Buyer	Analytical Buyer
	Needs of the segment	Security, Trust	Potential, Communication	Potential, Security
Accessibility	Can we communicate (advertising, for example) with the segment?	Medium With email campaign and social media	High With socials media and dedicated websites	Medium With email campaign and social media
	How often we can communicate	Daily	Daily	Daily
	How costly is it? The required marketing channels	Medium cost, So-Me marketing strategie	Medium cost, So-Me marketing strategie	Medium cost, So-Me marketing strategie
Sustainability	Is the segment profitable enough to sustain marketing efforts?	The segment is essential	No	Yes
	Does the segment align with our business goals?	Yes	yes	yes

	Can we realistically offer value to the segment sustainably?	yes	yes	yes
Actionability	Can we maintain a competitive edge for the segment?	yes	No	Yes
	Can we communicate the way the segment wants?	No	Yes	Yes

8.1.3. Positioning

Dstate's positioning of its prospects is based on the use of the product and its application. This is the most appropriate positioning strategy. Indeed, the service offered by Dstate is new and innovative and it is on this point that it is most interesting to develop a marketing strategy. It is not relevant to position the company in terms of price since there are no competitors offering the same service.

Regarding the market segments chosen as being the most relevant, namely homeowners and elderly investors, it makes sense to position yourself as a new alternative to what they already know.
The important points to emphasise to attract members of these segments are security, innovation, and simplicity.

3.2.9.2 Marketing mix activities

Product

As seen before, Dstate offers a property tokenization service. This allows individuals to invest in real estate without having a large capital. On the other hand, it allows owners to free up liquidity quickly.

The service offered by Dstate meets the different needs of customers:
-The need to generate liquidity
-The need for a real estate investment for small portfolios

-The need for an automated and secure platform

-The need for a tool to easily invest with others.

The company addresses these needs by creating a platform where users can submit an asset to the tokenization process, buy or sell tokens on the site's marketplace and manage the assets they own. In this way Dstate is a complete tool that meets the needs of customers.

Competition in the market is low because the technology and the market in general is new. Moreover, with its service, Dstate differentiates itself as the only company offering ease of use and speed of process. Indeed, the few potential competitors offer a service of advice and support, or only a marketplace to invest. As a result, Dstate has a comparative advantage.

Price

The pricing study for the various Dstate services was done in such a way as to be sustainable while not being a hindrance to the users.

Tokenization price: 1000€ + O.5% of the total value of the asset.

Fees on token purchase: 0.5% of the total amount

Management service fees: 1% if the rent (Deduce before the distribution to the investors)

The price of the service cannot be the core of the marketing strategy since there are no competitors whose prices could be used as a comparison. Therefore, the price variable should not be an obstacle but not a marketing asset to stand out.

Dstate's pricing strategy is to be neither below nor above the current prices in the real estate and crypto markets. Indeed, to place below is not relevant considering the weak competition and to place above is not relevant either because the service is meant to be easy to access.

Also, it is envisaged that an offer will be created at launch to help the platform generate a large catalogue of properties. The strategy is to waive the tokenization fee for the first six months to encourage owners to come to the Dstate site. This would represent a cost to the company but would ultimately allow for a more aggressive launch. This strategy is not included in the budget estimate, however, as it is still a project and a marketing strategy such as this would require extensive study and does not represent a permanent strategy for the company.

Place

The place where the service is visible, available and used is the company's dedicated platform. It is therefore a digital service accessible anywhere in the world. However, it should be noted that Dstate is only being deployed in Europe at the moment to address legal issues. Indeed, it is risky to try to reach the whole world as soon as the company is launched. It is therefore preferable to start in Europe where regulations differ little between each country and seek to export once the company is stable.

Dstate is therefore an online service that is easily available. This is part of the marketing strategy and is in line with the values and needs of users for ease of access and use. This way Dstate can reach many potential customers. The presence on the internet also allows a better use and efficiency of digital advertising tools.

Promotion

The entire promotion of Dstate will be digital because of the service offered and the place where it is accessible. For this purpose, several tools are used:

- Social Media Marketing (SMM)

Social Media Marketing is the use of social media to promote a company, a product or a service. In the case of Dstate it is a good way to reach customers, especially on social media like twitter where there are strong communities of customers which are easy to target via advertisement. This is mainly about using the advertising tools provided by social media to effectively reach the customer target. In terms of content, it is generally short presentation videos with links to the Dstate site to generate leads. Then, it is also a matter of creating accounts representing the company on the different social media to first gain visibility but also to have a link with the users, allowing them to interact with them and thus strengthen the image of the company.

- Search Engine Optimisation (SEO)

SEO is the process of improving the visibility of a website in a search engine. For this purpose, there exist various methods such as the use of keywords that allow the greatest number of searches possible. For example, in the case of Dstate the keywords real estate, blockchain, investment, tokenization and token must be present. It is also necessary to create as many connections as possible between the Dstate site and other linked sites to improve the leads leading to the sites.

- Search Engine Marketing (SEM)

Close to SEO the SEM is the method that consists of paying for being better placed in the result on the search engine. This method can cost a lot but generally give good results. To do that the differents ads that can be made and buyed are:

- Search Ads
- Shopping Ads
- Display Ads
- Gmail Ads
- YouTube Ads
- Partnerships with influencers

Finally, the last method of promotion used by Dstate is the creation of partnerships with influencers. These latter may be important figures in the real estate and crypto markets, who share content on youtube, has a very influential twitter or facebook account using partnerships remuuner it is a whole part of their communities that Dstate can reach. It should be noted that this strategy is mainly relevant to reach the investor segment, especially young people, even if today social networks are not only used by a young population.

People

The people aspect of Dstate company isn’t relevant for the marketing mix study considering that it is an online company so the consumer is not supposed to meet any of the workers of the company. However, speaking about business to business meetings and partnerships with external collaborators such as marketing agency or developing company employed by Dstate the question of the workforce of Dstate become relevant. Indeed, it is important to show maturity, professionalism and seriousness to get collaborators to trust you. In this case Dstate workforce isn’t the most experience considering that the project holders are young but it is balance by the fact that they are motivated. Therefore, this lack does not represent a major pain point for the marketing mix.

Process

From the customer perspective, the process of buying and using the Dstate platform and services is easy to use. That is one of the main purposes of the service. The website concentrates all the services offered by the company in one place. About the payment it is the same standards as most of the web-sites such as credit card payments and bank transfer but the customer can also use a crypto wallet. The interface of the website is intended to be clear and user-friendly.

In terms of client support they will be of course a whole section dedicated to frequently asking questions, company description and support team contacts.

Physical evidence

The physical evidence must be in adequacy with the company values, a clear and smooth design for the website, the logo, and the social media content.

Figure 18: Physical Evidence n°1

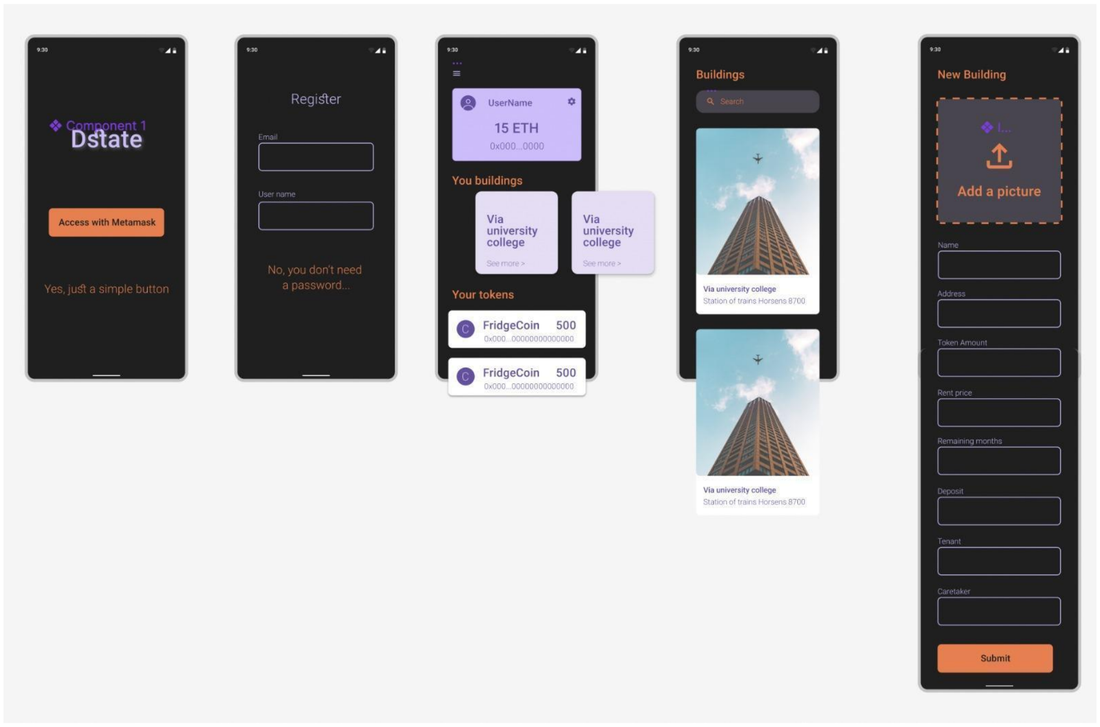
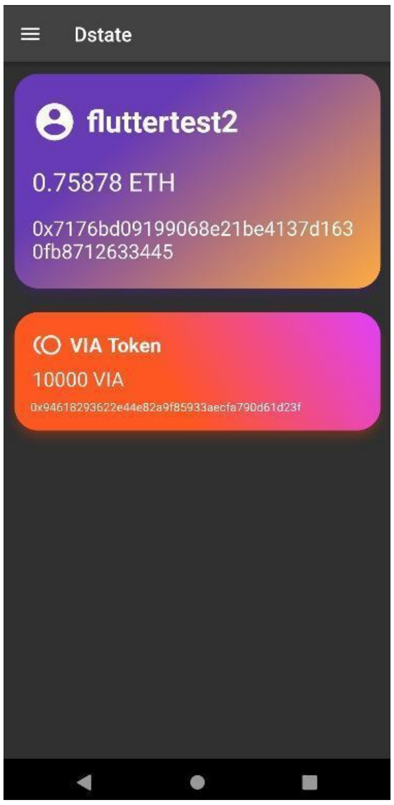


Figure 19: Physical Evidence n°2



fluttertest2

0.75878 ETH

0x7176bd09199068e21be4137d163
0fb8712633445

VIA Token

10000 VIA

0x94618293622e44e82a9f85933aecfa790d61d23f

3.2.10 Budgets

The budgeting of a project is an important step to evaluate the financial needs of the project but also more generally to evaluate the feasibility of the project. For this purpose, there are different budgets to dissociate which represent different aspects of the financial life of the company. It should also be noted that a budget is only an estimate of the needs and does not correspond to the reality of the course of the project which is impossible to foresee with exactitude. Within the framework of the budget study for the Dstate project, the estimates are as realistic as possible within the limits of the data collected during the research.

3.2.10.1 Establishing budget

The establishing budget is the amount of money the project holders need to establish the company and how it is distributed among the different cost categories. In the case of Dstate the establishing budget isn't huge considering differents points. First The company doesn't need any offices because it is an online application and staff can remotely work. This represents savings on the budget and gives more liberty for the employees regarding their work. The main cost of the budget is the cost of contracting a company to develop the Dstate software. This cost was estimated from the average salary of a developer specialised in blockchain in Europe. It was estimated that it would take 3 months of work for four developers, which amounts to an annual salary of one developer, or 50,000€. Then the costs are divided between the purchase of computers, printers, photocopiers and office equipment. Finally, there is a certain budget for the costs of creating the company and consulting legal professionals.

3.2.10.2 Operating budget

The operations budget is the most important budget, both in size and utility. It is the budget that gives an idea of the different costs and revenues that the company expects during its first 5 years of life. To be more accurate, there are three scenarios proposed.

The first scenario is the most realistic, the company plans to tokenize 10 properties the first month whose average value is 250000€. The company also assumes that each property subscribes to the management service and that there are on average 100 investors who own each property.

For the following months, the overall increase in revenues is first estimated at 10% and then decreases to stabilise at 1%. It is difficult to estimate the growth of a company, especially when the market is new. It was therefore preferable to choose a relatively slight growth for more security.

The estimated costs concerning the salaries correspond to the employment of two developers costing 5000€ gross per month as well as three times 4000€ gross per month to pay the project leaders who will also work full time.
The main cost is the marketing budget, which plays a major role. Indeed, it is an important expense for Dstate because it is one of the main factors to grow the number

of users. In the second scenario, the marketing expenses are greatly reduced because it is not possible to reduce the human resources expenses. It should be noted that these reductions could be the cause of a decrease in revenue growth, but this cannot be estimated.

Concerning the second scenario, it represents the worst case scenario, where all revenues are divided by three. By necessity, in order not to bankrupt the company, the costs are readjusted, mainly the marketing expenses.
The third scenario is the exact opposite of the second, where all revenues are multiplied by three. This scenario is less realistic than the first two and is therefore less interesting to consider.

After five years of life, all three scenarios end in profit. For the first scenario the estimated profit is 511 433€. For the second and third scenarios the results are respectively 149 852€ and 14 475 072€. In all three cases, this is a positive estimate for the project.

3.2.10.3 Cash Flow budget

The cash flow budget is a tool that complements the operating budget. In fact, it allows you to see the company's cash flow each month. The budget has only been made for the first year and following the estimates of scenario number 1 because it is the most realistic scenario and the first year is the most critical for a company. In general, Dstate's cash flow is relatively good. During the first few months, it sinks into the red but consolidates very quickly to finally be in constant growth. According to these estimates, Dstate should not suffer any major cash flow problems during the first year.

3.2.10.4 Funding

The financing of Dstate is made up solely of the contributions of the three project leaders, each up to 25 000€. This way, there is no need to take out a loan that would later destabilise the cash flow. Also, this brings the total financing to 75 000€ which covers the expenses foreseen by the establishing budget as well as an initial liquidity for the treasury.

4 Future of the Project

Through the process of software development, a Minimum Viable Product has been reached. This product, even if not in its final form, shows how the proposed solution will operate, while also serving as a basis for future versions using an incremental approach. Future versions of the backend could include better and more extensive error handling, automated unit testing, and CI/CD process. When actual legal contracts are being managed, the database can host a new table for the uploaded files, relating them to specific users and buildings. It could also include an AWS S3 bucket to host images. Newer versions of the smart contracts could benefit from expanded functionality, while also emitting more events that can be used to store useful information. They could also implement some optimizations to reduce gas cost, both for deployment, and for users interacting with them. As for the frontend, a big and simple improvement could be porting the interface to more platforms, such as ios and web. User experience could also be improved, in order to achieve a greater usability, accessibility, and user satisfaction. While the current iteration of the products satisfies the proposed challenges, it has room to grow and improve, but still serves as a good example and starting point for a more improved version.

5 Conclusions

Regarding the business study, it shows that the company is able to generate revenue by applying fees on the different services that it offers. This revenue is estimated as enough for the company to be sustainable and profitable. Also, considering the marketing part of the study, it appears that the company is in a new market and has only a few competitors. This brings an advantage to the company and allows it to position itself as the only provider of the specifical service that it is offering. Finally after a study about which marketing mix the company needs to use, the results are showing that it is composed of digital marketing including SEO and SAM. Considering the market, the price of the service is not impacted by the competitors.

This is because the company is providing a service that is well differentiated from the competition.

In regards to the software development, a robust implementation of the design was achieved, especially with the smart contracts, which lie at the core of the solution. All the proposed user stories have been considered, covering the needs of the stakeholders. A decentralised, secure, and functional set of smart contracts have been developed and tested. Furthermore, a fast, flexible, and lightweight backend server was created. Moreover, a consistent, reliable, and rapid database has been built. Finally, a portable, responsive, and user-centric frontend interface was implemented. To conclude, a Minimum Viable Product that covers all the subproblems is the result of the software engineering process.

6 Sources of information

ec.europa.eu. (n.d.). *Overview - Housing price statistics - Eurostat*. [online] Available at: https://ec.europa.eu/eurostat/web/housing-price-statistics [Accessed 10 May 2022].

stobox.consulting. (n.d.). *Stobox | Security Token Offering and Tokenized Assets Ecosystem*. [online] Available at: https://stobox.consulting/real_estate_asset_tokenization [Accessed 10 March 2022].

solidblock.co. (n.d.). *Best Real Estate Tokenization - SolidBlock*. [online] Available at: https://solidblock.co/real-estate-tokenization/ [Accessed 20 March 2022].

ec.europa.eu. (n.d.). *Overview - Housing price statistics - Eurostat*. [online] Available at: https://ec.europa.eu/eurostat/web/housing-price-statistics.

Talent.com. (n.d.). *Salaire, France - Salaire moyen*. [online] Available at: https://fr.talent.com/salary?job=d%C3%A9veloppeur+blockchain#:~:text=Le%20salaire%20m%C3%A9dian%20pour%20les [Accessed 27 April 2022].

Statista. (n.d.). *Topic: European Union*. [online] Available at: https://www.statista.com/topics/921/european-union/#dossierKeyfigures.

Schmitz, R. (2022). *Rise of Crypto Europe*. [online] Storm2. Available at: https://storm2.com/storm2-voice/blockchain/crypto-europe/#:~:text=Whilst%2C%20in%20Europe%2C%2040%20percent
[Accessed 27 April 2022].

BFM BUSINESS. (n.d.). *7,3 millions de ménages multipropriétaires possèdent plus de la moitié des logements de particuliers*. [online] Available at: https://www.bfmtv.com/immobilier/7-3-millions-de-menages-multiproprietaires-possedent-plus-de-la-moitie-des-logements-de-particuliers_AV-202111260173.html#:~:text=Immobilier- [Accessed 27 April 2022].

www.insee.fr. (n.d.). *Ménage – Famille − Tableaux de l'économie française | Insee*. [online] Available at: https://www.insee.fr/fr/statistiques/4277630?sommaire=4318291#:~:text=En%202016%2C%20la%20France%20compte [Accessed 26 April 2022].

Scott, G. (2020). *Porter's 5 Forces*. [online] Investopedia. Available at: https://www.investopedia.com/terms/p/porter.asp.

RealT, Inc. (n.d.). *RealT, Inc.* [online] Available at: https://realt.co/.

solidblock.co. (n.d.). *SolidBlock*. [online] Available at: https://solidblock.co/tokenization-form [Accessed 10 March 2022].

omni-psi.com. (n.d.). *Omni $ORT - Decentralized NFT Real Estate Marketplace: Buy Property With Crypto*. [online] Available at: https://omni-psi.com/.

digishares.io. (n.d.). *Services*. [online] Available at: https://digishares.io/services [Accessed 5 March 2022].

Auth0 Docs. n.d. *JSON Web Token Structure*. [online] Available at: <https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-structure> [Accessed 1 June 2022].

Auth0 Docs. n.d. *JSON Web Tokens*. [online] Available at: <https://auth0.com/docs/secure/tokens/json-web-tokens> [Accessed 1 June 2022].

Mongoosejs.com. n.d. *Mongoose ODM v6.3.5*. [online] Available at: <https://mongoosejs.com/> [Accessed 1 June 2022].

Savchenko, N., 2019. *Decentralized Applications Architecture: Back End, Security and Design Patterns*. [online] freeCodeCamp.org. Available at: <https://www.freecodecamp.org/news/how-to-design-a-secure-backend-for-your-decentralized-application-9541b5d8bddb/> [Accessed 1 June 2022].

Mindtools.com. n.d. *SMART Goals: How to Make Your Goals Achievable*. [online] Available at: <https://www.mindtools.com/pages/article/smart-goals.htm> [Accessed 1 June 2022].

Atlassian. n.d. *User Stories | Examples and Template | Atlassian*. [online] Available at: <https://www.atlassian.com/agile/project-management/user-stories> [Accessed 1 June 2022].

Web3js.readthedocs.io. n.d. *web3.eth.Contract — web3.js 1.0.0 documentation*. [online] Available at: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth-contract.html> [Accessed 1 June 2022].

https://www.redhat.com/. 2020. *What is a REST PI?*. [online] Available at: <https://www.redhat.com/en/topics/api/what-is-a-rest-api> [Accessed 1 June 2022].

Quicknode.com. n.d. *What is an ABI? explained - step-by-step beginners guides | QuickNode*. [online] Available at: <https://www.quicknode.com/guides/solidity/what-is-an-abi> [Accessed 1 June 2022].

Kakkad, R., 2021. *Django vs. Express: Which Backend Framework to Choose?*. [online] Weboccult.com. Available at: <https://www.weboccult.com/blog/django-vs-express-which-backend-framework-to-choose> [Accessed 1 June 2022].

hapi.dev. n.d. *hapi.dev*. [online] Available at: <https://hapi.dev/> [Accessed 1 June 2022].

Fastapi.tiangolo.com. n.d. *FastAPI*. [online] Available at: <https://fastapi.tiangolo.com/> [Accessed 1 June 2022].

freeCodeCamp.org. 2020. *What is Middleware? Definition and Example Use Cases*. [online] Available at: <https://www.freecodecamp.org/news/what-is-middleware-with-example-use-cases/> [Accessed 1 June 2022].

Introduction to Node.js. n.d. *Introduction to Node.js*. [online] Available at: <https://nodejs.dev/learn> [Accessed 1 June 2022].

Business.qld.gov.au. 2022. Running a business | Business Queensland. [online] Available at: <https://www.business.qld.gov.au/running-business> [Accessed 1 April 2022].

Investopedia. 2022. Investopedia. [online] Available at: <https://www.investopedia.com> [Accessed 1 April 2022].

ethereum.org. 2022. Home | ethereum.org. [online] Available at: <https://ethereum.org/en/> [Accessed 1 April 2022].

Uniswap Protocol. 2022. Home | Uniswap Protocol. [online] Available at: <https://uniswap.org/> [Accessed 1 April 2022].

OpenZeppelin. 2022. OpenZeppelin. [online] Available at: <https://openzeppelin.com/> [Accessed 1 April 2022].

Hapipal.com. 2022. hapi pal. [online] Available at: <https://hapipal.com/> [Accessed 1 April 2022].

Docs.metamask.io. 2022. Introduction | MetaMask Docs. [online] Available at: <https://docs.metamask.io/guide/> [Accessed 1 April 2022].

Trufflesuite.com. 2022. Truffle Suite - Truffle Suite. [online] Available at: <https://trufflesuite.com/> [Accessed 1 April 2022].

Docs.flutter.dev. 2022. Flutter documentation. [online] Available at: <https://docs.flutter.dev/> [Accessed 1 April 2022].

Dart packages. 2022. Dart packages. [online] Available at: <https://pub.dev/> [Accessed 1 April 2022].

Scrum.org. 2022. Home. [online] Available at: <https://www.scrum.org/> [Accessed 1 April 2022].

Dart packages. 2022. *walletconnect_dart | Dart Package*. [online] Available at: <https://pub.dev/packages/walletconnect_dart> [Accessed 1 June 2022].

Docs.soliditylang.org. 2022. *Solidity — Solidity 0.8.14 documentation*. [online] Available at: <https://docs.soliditylang.org/en/v0.8.14/> [Accessed 1 June 2022].

Docs.walletconnect.com. 2022. *Mobile Linking | WalletConnect Docs*. [online] Available at: <https://docs.walletconnect.com/mobile-linking> [Accessed 1 June 2022].

ethereum.org. 2022. *Transactions | ethereum.org*. [online] Available at: <https://ethereum.org/en/developers/docs/transactions/#:~:text=An%20Ethereum%20transaction%20refers%20to,takes%20place%20within%20a%20transaction.> [Accessed 1 June 2022].

GitHub. 2022. *card_example/main.dart at master · JohannesMilke/card_example*. [online] Available at: <https://github.com/JohannesMilke/card_example/blob/master/lib/main.dart> [Accessed 1 June 2022].

GitHub. 2022. *Connect Metamask with a native mobile app built with Flutter · Issue #3735 · MetaMask/metamask-mobile*. [online] Available at: <https://github.com/MetaMask/metamask-mobile/issues/3735> [Accessed 1 June 2022].

Relevant Software. 2022. *Top 8 Flutter Advantages*. [online] Available at: <https://relevant.software/blog/top-8-flutter-advantages-and-why-you-should-try-flutter-on-your-next-project/> [Accessed 1 June 2022].

7 Appendices

Business part budget excel table:

Business part Appendix.xlsx

All the software files referenced in this document can be accessed in the following public repositories. One contains the backend and the smart contracts, while the other contains the frontend:

Backend & Smart Contracts: https://github.com/gloriadesideri/dstate-be

Frontend: https://github.com/ericmartihaynes/dstate-frontend