Development of the Nexarium Store: Conceptual and Technical Solutions

by

Edwyn Batista

A report
presented to the University Lyon 2
in fulfillment of the
internship requirement for the degree of
Master 2
in
Programming and Development of Video Games

Lyon, Auvergne-Rhône-Alpes, France, 2018

Abstract

The Nexarium is a proposal of B2Expand to integrate games around the concept of cross-gaming information. Assets that are shared among games can carry a portion of the experience lived in a context to another. In this scenario, the emerging blockchain technology presents characteristics that respond to these needs: e.g. the concept of "ownership" allows users to hold their inventory wherever they go. This feature is reached due the blockchain's capacity to mapping values to indexes (addresses), such as a bank account. In addition, the history of this information is immutable and the control is decentralized, discarding the reliance on a central element. This last property is also important for reducing the possibility of attacks and unavailability. Then, the Ethereum blockchain is highlighted because it provides a powerful mechanism to build decentralized applications, by means the implementation of smart contracts.

The internship performed in B2Expand explored this scenario. The development of the Nexarium Store, an application to manage the commerce in the Nexarium ecosystem, proposed challenges related to the interaction with the blockchain and its implications. The solution must privilege the modularity and extensibility because it is aimed to aggregate shops and partners futurelly. As a product that should be oriented for gamers, it has to face some constraints imposed by the blockchain, defining an adequate user experience. This work was conducted starting from the requirements gathering and design of UX, until the conception/implementation of a custom architecture to deal with the requisites. The development of a modular plugin for authentication is valorized by the explanation of its technical solutions, besides the participation on other projects (e.g. the edition of smart contracts).

This report focus on how the previous knowledge was applied. And It also pays attention to the importance of the interaction with other domains and experienced professionals to the acquisition of new abilities. The product of this internship, a version of the Nexarium Store, includes the main functionalities that are necessary for the interaction with the Nexarium Ecosystem.

Résumé

Le Nexarium est une proposition de B2Expand visant l'intégration des différents jeux autour du concept d'informations *cross-gaming*. Quand un asset (un bien, en français) est partagé entre des jeux, il est capable d'apporter avec soi une partie de l'expérience vécue dans un contexte vers un autre. Dans ce scénario, la technologie émergente de la blockchain présente des caractéristiques qui répondent à ces besoins: e.g. le concept d'appartenance permet aux utilisateurs de conserver leur inventaire partout. Cette fonctionnalité est atteinte grâce à la capacité de la blockchain d'associer des valeurs à des indexes (adresses), telles qu'un compte bancaire. D'ailleurs, l'historique de cette information est immuable et le contrôle est décentralisé, enlevant la dépendance à un élément central. Cette dernière propriété est également importante pour réduire la possibilité d'attaques et d'indisponibilité. Ensuite, *la blockchain* Ethereum est mise en évidence car elle fournit un mécanisme puissant pour créer des applications décentralisées, par l'implémentation de *smart contracts*.

Le stage effectué dans B2Expand a exploré ce cadre. Le développement du Nexarium Store, une application pour la gestion du commerce dans l'écosystème Nexarium, a proposé des défis liés à l'interaction avec la blockchain et ses implications. La solution doit privilégier la modularité et l'extensibilité parce qu'elle vise à futurement regrouper d'autres magasins et partenaires. En tant que produit orienté pour les joueurs, il doit faire face à certaines contraintes imposées par la *blockchain*, définissant une expérience utilisateur adéquate. Ce travail a été réalisé en partant de la collecte des besoins et de la conception de l'expérience utilisateur, jusqu'à la conception / implémentation d'une architecture personnalisée pour répondre aux exigences du Nexarium. Le développement d'un *plugin* modulaire pour l'authentification est valorisé par l'explication de ses solutions techniques, en plus de la participation à d'autres projets (par exemple, l'édition de *smart contracts*).

Ce rapport met en avant la manière dont les connaissances pré-acquises ont été appliquées. Et il fait également attention à l'importance de l'interaction avec d'autres domaines et des professionnels expérimentés pour travailler de nouvelles compétences. Les produit de ce stage, une version du Nexarium Store, comprend les principales fonctionnalités nécessaires à l'interaction avec l'écosystème Nexarium.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The creation of video games is associated to an industry that is constantly in progress, presenting new tools, facing challenges and changing perspectives frequently. And this evolution leads to the release of several products that enchante people all around the world.

In order to get synchronized with the new trends, Gamagora (Communication Institute, University Lyon 2) offers a Master degree on Programming and Development of video games, focused on the exploration of recent technologies and methodologies related to the production of games. The students are stimulated to interact with the agents that compose this industry, benefiting from a multidisciplinary environment and experience of professionals. This is accomplished by the realization of an internship.

All this discussion about innovation settings can be amplified if we aggregate a new variable: the blockchain. A tool that presents new parameters of communication in order to supply decentralized control, offers a bunch of advantages that rapidly attracts the attention of the games. The possibility of holding an immutable history of transactions, security mechanisms and the capacity to ensure concrete ownership of virtual values are some examples. But, the adoption of such benefits has to deal with a counterpart: the cost and delay of transactions, for instance. Moreover, a factor that mess the games designers' ideas up is the need to incorporate the requirements for user interaction introduced by the blockchain.

Is this worth? Can the video games go beyond by using these features? These are the questions that B2Expand attempts to answer and it is being done it with a positive result. The integration of games and blockchain can give sense to an ecosystem of games

that share information among them. The players can own items that have a real exchange value.

So, during the period of May-September/2018, it was given the chance to integrate B2Expand's team with the objective of building this ecosystem: the Nexarium. The mission was to develop a store application that centralized the commerce, where the players could buy and sell assets shared by all the games. It required the evaluation of the user experience and the production of a extensible solution, that would be ready to welcome all the games from the ecosystem. The internship also opened the opportunity to develop competences on the Ethereum blockchain, the interaction with it and the concept of solutions involving smart contracts.

This document aims to describe the work developed during the internship, by focusing in its motivations and outcomes. Below, the introduction is followed by a section dedicated to clarify the terms and processes that characterize the context of work (chapter 2). The games and the blockchain are evidenced because they compose the ground subjects. And the relationship between them (the advantages and obstacles) is always highlighted. A brief description of the game industry helps to introduce the work environment (chapter 3): B2Expand is exposed by means the presentation of its organization, objectives and products. All these sections are useful to contextualize the importance of the performed tasks. The narrative of the executed activities takes the biggest portion of the content because it reports the details about the decision taking, its justifications and derived actions. The chapter 4 starts by showing the timeline for the internship and the strategy defined for the organization of the work. It is followed by the steps for the realization of the main tasks: the requirements gathering, implementation of UI, interaction/manipulation of smart contracts and the Nexarium plugin. The conclusion of this report brings the reflection on the results of the internship, including the knowledge acquisition and implemented products (chapter 5).

Chapter 2

Fundamentals

2.1 Digital Games

People are willing to dedicate a considerable part of their time to play games. And this statement can be applied to a very large demographic range: from children to elderly; from Asia to America; any gender etc. The duration of the game can vary; its subject or the context where it is applied; the number of platforms is huge; But people still get engaged by this tool.

The games' capacity of providing entertainment is clear. However, they are not reduced to the fun aspects. Prensky (2001) [47] points several characteristics that describe games and how they are capable to make them interesting and to produce engagement (Table 2.1).

These aspects listed by Prensky (2001) [47] evidence the variety of skills developed and experiences explored by the games. They allows us to understand their potential impact. Nevertheless it calls attention to the complexity of creating a product that correctly attends to all this features.

Alongside the development of computers and a related culture, the digital games assumed an important role in the society. But sometimes controversial: some of the first studies performed with the objective of evaluating the impact of video games were oriented to the verify their possible negative influence on children. This was derived from the violent content presented by some games (Anderson; Bushman, 2001 apud Jones et al., 2010) [38, 23]. However, recently the researches are directed to the opposite side: they focus on the positive outcomes of the games, in particular, the engagement as a means

Characteristics	**Result**
Fun	Enjoyment and Pleasure.
They are a form of play	Intense and passionate involvement.
Have Rules	Structure.
Have goals	Motivation
Interactive	Keep the action.
Adaptative	Flow (a balance between difficult and skills that involves the player and produces concentration).
Outcomes and Feedback	Learning.
Win States	Ego gratification.
Conflict / Competition / Challenge / Opposition	Adrenaline and Excitation.
Problem solving	Creativity.
Interaction	Social group work.
Representation and Story	Emotion.

Table 2.1: Characteristics of games and their importance (Adapted from [47]).

to produce learning (Prensky, 2001) [46]. The field of serious games is becoming a trend because it integrates the benefits of games in the teaching process in order to generate a game-based learning (Prensky, 2001) [46].

2.1.1 The Game Industry

The previous description about games clarifies the reasons why they are so fascinating. So this interest is converted into a lot of products that flood the market. The possibility of reaching a large range of public niches makes the adventure of producing games even more attractive. These are some reasons why 116 billion dollars were derived from the game industry in 2017 [24].

The numbers related to the games are capable to express their significance as a commercial product. For example, the Guinness book [48], based on informations from SuperData, registers League of Legends as the most played online game in 2016. It represents 100.4 million people per month, generating 1.9 billion dollars in virtual sales (also a record). It is important to mention that this money comes from microtransactions, since LoL is free to play.

However, the games industry is also sustained by the interaction with other medias. The game titles inspire the creation of accessory products, such as videos, toys etc. And sometimes the initiative for this kind of merchandising comes from the engaged audience, that helps to attract more fans (i.e. more profit).

The entertainment is still the subject that concentrates the great majority of interest of the game industry. Among the games, 43% is built for phones; 29% for consoles; and 28% for pcs [24].

The revenue reported by the game industry can also present a bad side. Kopster (2013) [39] establishes a reflection about the overvaluation of gameplay to the detriment of games' contents. It can be interpreted as that sometimes this industry forgets the real impact that games can produce. So they are designed with the main objective of being sold in large scale, creating a short-term satisfaction, and neglecting the long-term results.

2.2 Blockchain

The term "Blockchain" is a word that became world-spreaded due the cryptocurrencies (i.e. digital currencies), such as Bitcoin [3] and Ether [9]. People started to use their digital money to shopping or paying for services; the coins' value is tracked daily by investors and speculators. It is a market that represents 27 billion dollars in April/2017, as stipulated by [35]. This technology increases its importance not only as a computing tool, but also as a component that has a socio economic impact, by changing [or at least by stimulating the reflexion about] the way that people exchange information and value (in the context of cryptocurrencies, the money) [34].

The blockchain provides a distributed (and cryptographically safe) way to handle and store information, so the users do not need to trust in a central element, e.g. a server [36]. In the non-distributed paradigm, all information is hold by some agent in the sense that it is able to determine the validity of the values. So, it controls the system state by defining if some data corresponds to the current state or not. This context can be illustrated by a bank account, where the bank is the central institution, or a game account in a server-based game. Every time that we need to perform some action (transfer some money or use some game item), we need to interact with the central element to inform and validate this action. This method presents some risks, such as the vulnerability of the server, its unavailability or perhaps its malicious intent.

The blockchain proposal is based on distributing copies of the information among several elements that compose the network [26]. As the data is protected against modifications,

each element is able to verify the current state of the system. And when some element wants to modify the values, this change is informed to everyone. Then, the big amount of elements involved reduces the unavailability of the system; and if some element is attacked or has a suspicious behavior, the information is safe because it is stored by all the others (it is necessary that the majority of the elements be attacked in order to force a new value, which is difficult due the big amount of elements).

However, the blockchain is still under evolution. It means that some problems have to be faced in order to consolidate this technology. For the moment, the blockchains have to deal with problems like a high energy consumption, a speculative market over the digital money, complexity of usage [43] and a relative elevated cost for performing the transactions. All these aspects count for increasing the lack of belief.

In the following subsections, we will expose some technical details on how a blockchain works and how these features are actualized. Since the Ethereum blockchain was the environment used during the internship, a special focus will be oriented to the fundamentals of implementation using this tool. And finally, concrete examples of applications of the blockchain.

2.2.1 Interaction Among Blockchain Nodes

In short terms, the blockchain is a composition of transactions, which construct the state of the system. So, these transactions can be described as records of modifications on the state. It means that the order of the transactions is important because it can lead to different states.

The transactions are grouped in blocks, then linked among them, forming a chain: the blockchain. A well-illustrated representation of this process can be found at [4, 33].

An important tool for the formation of a blockchain is a hash function. This instrument, respecting the concept of a function, is basically the definition of a mapping between two sets. In the context of blockchain, it is applied to define "labels" for the blocks. But, some peculiarities are common to the hash functions that serve to the blockchain [27]: a) the target set (codomain) has a large amount of elements, representing the reduction of collision and a long-term disponibility (i.e. there is enough possibilities of result to deal with the needs); b) the inversion of theses functions is unfeasible, avoiding the regeneration of the source; c) they generate the same result for any identical values from the domain and a simple modification on the original value produces a big effect on the result. It is useful for validation of incorruptibility of values; and d) they do not consume a lot of resources to be calculated.

Based on the characteristics of hash functions for the blockchain, clarified above, we can mark that calculating a hash value establishes a "integrity certificate" for the block. It is possible because the calculation of a block's hash takes into account its entire content. In more details, each block of transactions is formed by:

- Index (Block #): block's id;
- Timestamp: the date when the block was created;
- Data: the information stored in the block (transactions);
- Nonce: a field that can be modified in order to "validate" the block, by satisfying a blockchain-specific condition. For example, in the blockchains based on the proof-of-work validation, the nonce must be adjusted to produce a hash value starting by a certain amount of 0 (zeros).
- Hash: block's hash value, that serves as "integrity label" of its content;
- Previous Hash: hash value calculated for the precedent block in the chain. It defines the link between blocks;

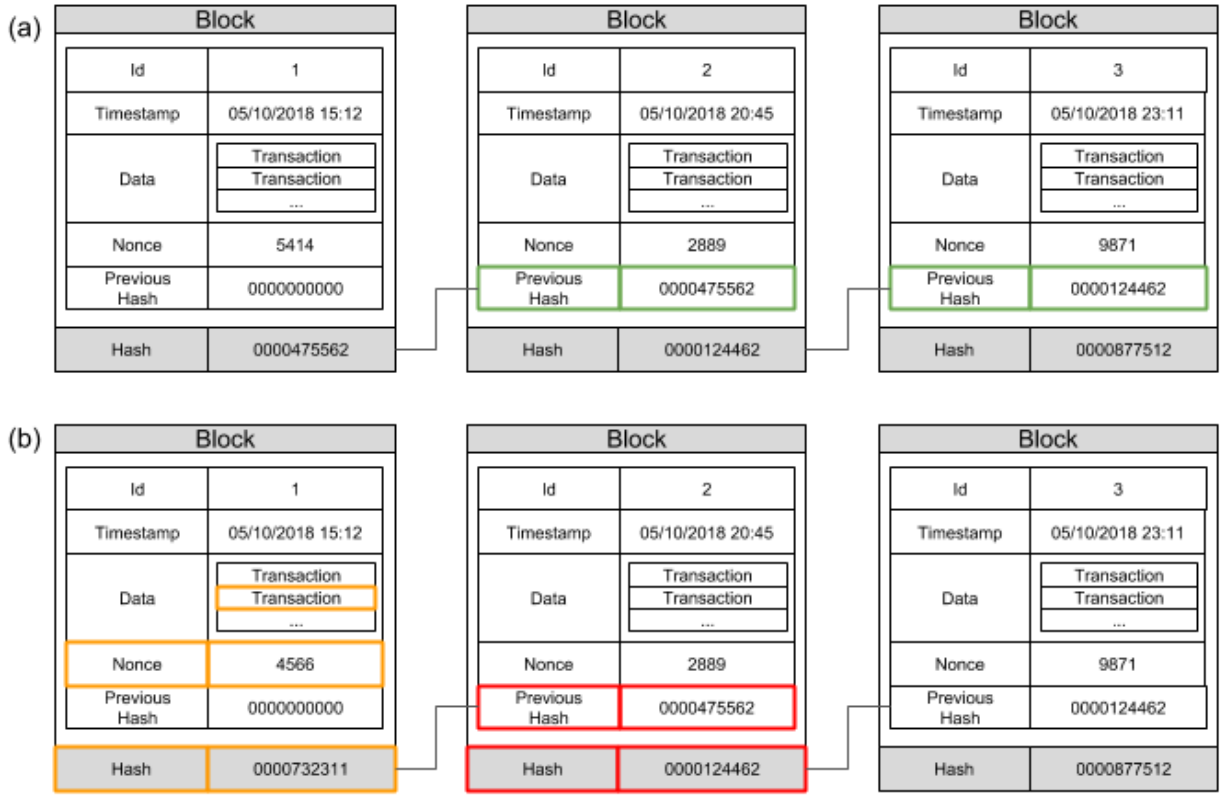
The fact that a block's hash value covers the reference to the previous block in the chain is necessary to ensure the security over the order of blocks. If a link is changed, the hash of respective block will not match its content. Then, this effect is replicated to the subsequent blocks, that cannot be validated. The same reasoning can be applied for the modification of the content of an intermediary block: the hash used as reference in the next block will lose its validity. This process contributes for the validation of the chain, collaborating for the immutability of the blockchain.

From a higher-level perspective, the blockchain is shared among nodes (Each node holds a copy of the blockchain). And their mission is to "mine" the blocks. In other words, they have to group transactions and validate the block. "Mining" is famous due the possibility of earning money, as a compensation to the devices that first accomplish it.

The nodes receive the transactions and create blocks. Once the block is done, the newly updated version of the blockchain is diffused and the nodes communicate among them to reach a consensus to accept the validity of the block and its insertion in the blockchain [41, 45]. Since the blockchain information is shared, the inclusion of the block is performed in all nodes.

The "mining" task is an arduous work, that requires a requires high-performance devices, sometimes application-dedicated. Among the activities executed, we can list:

Figure 2.1: The importance of a hash on simplified blockchains. (a) Presents a valid blockchain because the hash of the blocks are correctly reproduced on the next ones. (b) If someone attempts to change a transaction in block #1, the hash of the block will be affected and it is not more valid in the next blocks.



- Transactions validity: each transaction requires a modification in the blockchain. So, it is necessary to verify the current state of the system and evaluate if the transaction can take place. During this phase, in blockchains like Ethereum, the applications can return errors or intentionally refuse the transaction. The balance of coins is consulted in order to verify if the user has means to pay for it. If the transaction is accepted, it becomes part of the block. Else, it is rejected.

- Collecting payment: each transaction includes an amount of coins for paying for its execution. So, the node collects its compensation by adding a special transaction into the block that indicates the transfer of this value towards it. Then, this trans-

action will be part of the block such as any other. The nodes are able to select the transactions that they will execute. It means that those ones which offer the best payment are privileged. A transaction that does not pay well can take longer to be processed, or even never be consumed.

- Block creation and diffusion in the network: the hardest work is the validation of a block. In some cases, Ethereum, for instance, this task consists in the variation of the block's nonce value until reaching a block's hash that matches a pattern (e.g. starts by zeros). Then the node must communicate with the network in order to inform the insertion of the block.

2.2.2 Wallet

In the blockchain, the agents are identified by ids (formally called public addresses). A money transfer, for instance, goes from someone (or something) that holds the Id1 to someone that holds the Id2. So the values correspond to mappings based on this information [28].

With the objective of performing the transactions safely, a mechanism that "signs" them ensure the origin of the transactions. This strategy consists on the generation of a pair of values, named private and public keys, that have this unique matching. The most important is that a message signed (or encrypted) by a private key can only be verified (or decrypted) by its public key and vice-versa. So, it brings the possibility to openly distribute the public key. Then, any agent can certify the origin of messages signed by the private key. In this case, the private key must remains undisclosed and under user's control [51].

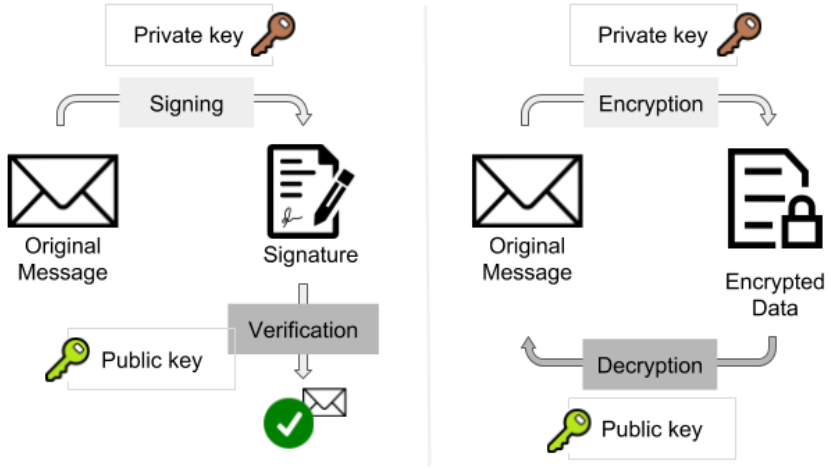
Ethereum benefits from the fact that the public key is reviewed to generate the public address that identifies an user.

All the elements described above, including the cryptographic keys and the public address, allow the secure association of the transactions to the user. This establishes the relationship of ownership with all values involved, characterizing a blockchain wallet.

Mnemonic

Handling the values of the cryptographic keys is normally difficult for human users. The reason is the numerical representation of theses values, sometimes requiring a conversion from hexadecimal base.

Figure 2.2: Possible applications of cryptographic keys. One key is capable to verify the data encrypted by the complementary one. It means that one key can attest the origin of a message signed by the other one. The same reasoning can be applied for the encryption.



With the aim of facilitating the manipulation by the user, Bitcoin developed the Mnemonic [20]. It provides a sequence of words that is capable to derive a set of cryptographic key pairs. Thus the user can deal with textual values, that are available in several languages.

The applications that interact with the blockchain bring functionalities that create/recover wallets from the Mnemonic.

2.2.3 Ethereum

The Bitcoin was the precursor of the blockchains. This is already consolidated but, as any other precursor, it opens gaps to innovation and improvement. And that is the ambition of Ethereum.

Ethereum is a blockchain based on its own currency, the Ether (ETH). And it differs from Bitcoin due its capability to host decentralized applications, known as DApps (In contrast to Bitcoin which is oriented exclusively to currency exchange) [21].

Smart Contracts

The environment proposed by Ethereum is composed by applications that are described as contracts (Smart Contracts). These contracts are written using the domain-specific language Solidity [17] and they are able to handle several complex computational elements, such as procedure calls, data mappings and contracts inheritance.

The contracts are inserted in the blockchain by a transaction (called deployment). They are stored as bytecodes and receive addresses (as pointers). Henceforth users or other contracts can interact with them by means a communication interface. An ABI (Application Binary Interface) brings the definition of all available methods. With the ABI, client applications are able to encode the requests by respecting the corresponding method's format.

Ethereum is capable to store variable values, such as a database. So, the transactions are interpreted as a way to change these values. They are evaluated by the code described in the contract and the effect applied over the current system state. Nevertheless any modification to this data must be done carefully since it costs some Ether. The reason for this cost comes from the fact that a change in values must be replicated through the whole network. The reading requests are executed with no charges.

The interaction with the Smart contracts can performed by means:

- Call functions: are reading requests. As each node holds the blockchain data, any one of them is able to answer this request.
- Transactions : requests to change the system state. After triggered, a transaction can be tracked by its hash code. Below we describe some required arguments that compose a transaction. A transaction is always signed by its sender in order to guarantee its integrity:
  - From: the sender's address;
  - To: the receiver's address;
  - Value: the amount of ether (ETH) exchanged by the transaction;
  - Data: some data that can be pertinent to the execution of the transaction.
  - Gas maximum: the maximum amount of gas units that can be spent by the transaction (Gas is the unit of measurement of effort demanded by the execution of the transaction). An estimate for the gas that will be required by a method can be requested from the contract.

  - Gas price: price for each unit of spent gas.

- Events: structures employed on the indication of the occurrence of some circumstance. The contract's code triggers events when some specific phenomenon occurs, e.g. a token transfer. Usually the client applications register watchers to track these events.

Besides the Ethereum main blockchain, there are some alternative environments dedicated to testing smart contracts. Since executing transactions in the main network has a considerable cost, smaller blockchains, such as Rinkeby, Kovan and Rospten are available to use fake coins. These test blockchains have less computing power than the main one and the ethers (ETH) are given by the community.

2.2.4 Tokens

The Cambridge dictionary defines a token as "something that you do, or a thing that you give someone, that expresses your feelings or intentions" [29]. In the blockchain, this statement can be translated to two distinct meanings, as described by [25]. They can be used as Usage Tokens: they are capable to represent a payment and they are required to perform actions (e.g. the Bitcoin and the Ether); or Work Tokens: represent the authorization to perform privileged tasks.

These concepts can also be categorized from another point of view, derivating the idea of Native Tokens and Asset-baked Tokens, among others [25]. The Native tokens are earned as a retribution for a work (mining, for example) and they are useful as a stimulus to performing these tasks. The native tokens are used to pay for transactions, as coins. Alternatively, the Asset-baked tokens mean the possession of a good (e.g. a game asset, a software license etc.).

Another famous term related to tokens is the ICO (Initial Coin Offering). It refers to the process of collecting funds based on the distribution of a token, triggered by a company that starts a project [37]. The investors give money (or digital currency etc.) and receive tokens as a compensation. As, the value associated to a token is low at that moment, the investors' expect to earn some profit as the project evolves (i.e. the token supposedly gets valorized).

2.2.5 Applications

Ethereum has established its importance by providing an environment for creation of decentralized applications. As a result, a countless amount of companies considered the opportunity to benefit from blockchain advantages and decided to develop solutions applied to this medium. The applications for token management and money transfer are instances, and they can be presented as the most commons. But, some initiatives go beyond and attempt to create more interesting projects, like social networks, video broadcasting etc. The games are not left aside, having to offer a new experience to the users.

In all cases, these adventurers are investing in a scenario that is growing, however still under development. It implies technical and conceptual challenges inherent to the incipient environment.

Some interesting Dapps (Decentralized apps) are listed below.

- Decentralized New Network [6]: this application proposes a hub for news inside the blockchain. By means this tool, writer users have the opportunity to create posts containing news; these posts are verified by reviewer users, that decide if the information will be published; and finally the reader users can rate the content, giving some tips to the writer. The reviewers are paid by their work and even the readers can earn some tokens by suggesting topics.

- LivePeer [14]: has the objective of building an infrastructure for decentralized video streaming In LivePeer, users can upload their productions and being compensated by turning their computers into processing nodes. For the developers, it offers an interface to integrate the streaming services to their applications and customize the experience.

  This Dapp wants to focus on the financial advantages: it provides more liberty in comparison to Social broadcasting services; and it is cheaper that Proprietary video services.

- Peepeth [16]: the main ambition of this proposal is to become a twitter-like social network integrated to the blockchain.

  This solution is based on the fact that the user posts will be permanently visible. As the interaction has a cost for the users, Peepth predicts the reduction of advertisements. Furthermore, the restrictions imposed by the blockchain (e.g. the delay to modify posts) can stimulate the reflexion on the contents of publications.

  This initiative encourages developers to produce side applications to interact with the main Dapp.

Its business model is grounded on charging users for first activities, but gradual reduction of costs based on engagement.

Games

The blockchain settings for games presents an key-factor: the transactions have a delay to be accepted and finally be integrated in the network. This way, the games cannot be based on a quick response time for the players. Another important aspect is the high cost of a frequent communication with the blockchain. All this context drives the designers to rethink the manner games are structured and propose innovative strategies to face these constraints. Some alternatives are oriented to the implementation of the following features:

- Part-centralization: it consists on the adoption of a server to concentrate some part of the control. In this case, the server takes charge of the repetitive interaction among the players. Then the blockchain is triggered in moments of greater importance, such as the storage of the final result of the interaction;

- Side chains [40]: it is the creation of a new blockchain as a branch of the main one. The side chains are normally dedicated to a specific purpose and therefore it reflects a gain in speed or processing power. The main blockchain is used as a safe point to save the information. However, the side chains have to implement alternative methods for dealing with security issues due the reduction of the number of nodes;

- Channels [42]: it is characterized by the opening of a means for communication between two users external to the blockchain. The transactions are performed inside this channel and, when the communication is completed, a history of the interaction is integrated into a blockchain. So the same result takes effect in the main blockchain.

The following projects illustrate the presence of games in Ethereum:

- CryptoKitties [5]: an application based on collectible non-fungible tokens. The token, which represents a cat (a kitty), is unique and belongs to the user. It cannot be replicated, but it can be breedable.

  The tokens have a very pleasant visual representation, putting in evidence its singularity, based on the token address.

  The interest for the game is improved by a market where tokens can be sold between users. Tokens' value increases according to factors like the tokens' rarity or beauty;

- Etheremon [8]: proposes the immersion in a world full of monsters that can be trained and evolve (such as Pokemons). The player must engage its monsters in battles in order to earn XP. When a Mon (the name of the creatures in Etheremon) reaches a certain level, it gains some abilities, like changing its appearance or lay eggs.

  We can find some similarities with the CryptoKitties: the laying process resembles the Kitties breeding and the commerce value is based on the monsters' characteristics.

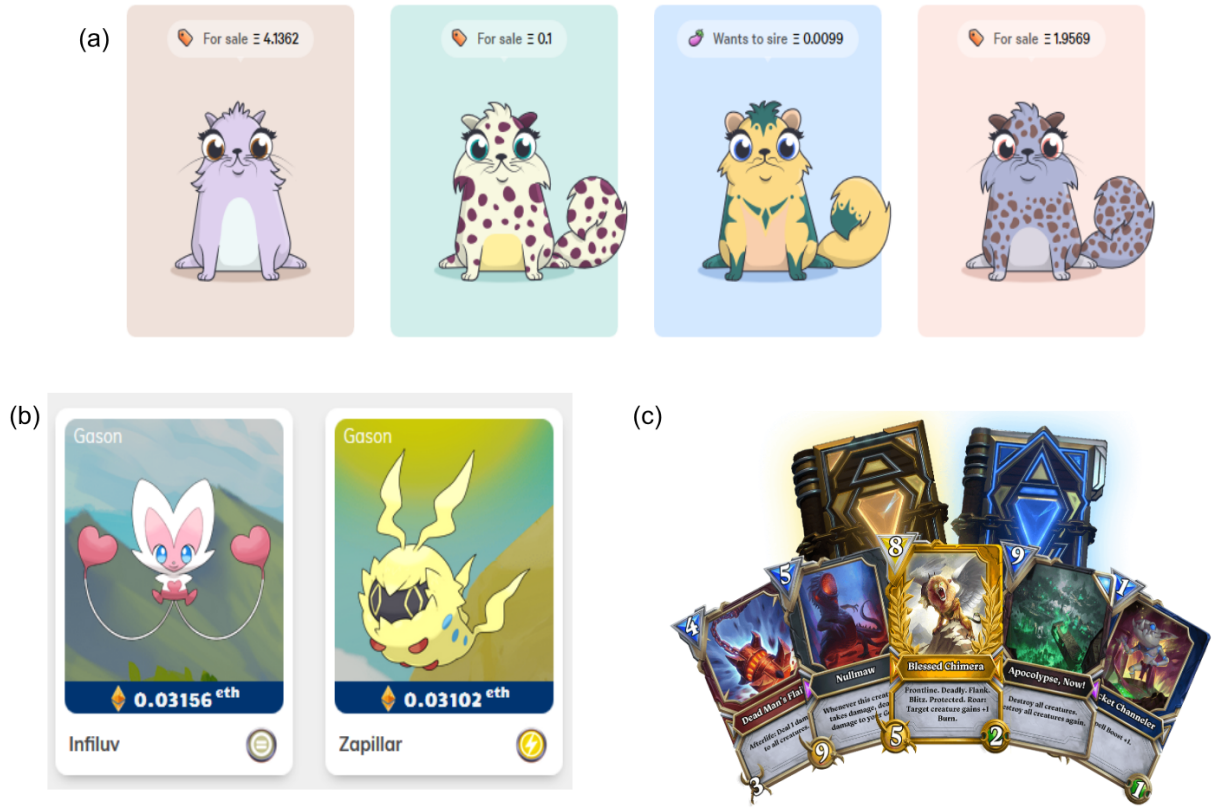
  The Etheremon roadmap establishes the future production of a virtual reality system to make even funnier.

- Gods Unchained [11]: it is a turn-based card game that uses the blockchain to ensure the ownership of the cards.

  The game provides more than 380 types of cards and highlights the fact that some of them will be sold in limited edition, increasing its value for the players. Once the selling of a limited edition card is over, they can be bought exclusively from other players.

  The Gods Unchained uses the blockchain immutability to stimulate the participation on tournaments, because the results will be irrevocable and visible forever.

Figure 2.3: Blockchain Games illustrations. (a) CryptoKitties catalogue [source: [5]]; (b) Etheremon store [source: [8]]; (c) Gods Unchained cards [source: [11]].



Chapter 3

Work Environment

The selection for a company to host the internship was oriented by the aim of applying and improving the knowledge acquired during the Master 2 formation. So, B2Expand proposed to engage the innovative development of games associated with the blockchain. This scenario allows the interaction with experienced game developers (among other professionals) and the introduction to the emerging world of the Smart Contracts. It represents the possibility to develop skills on Unity3D and the chance to get formed in a new technology, Ethereum. In addition, the proposal becomes even more attractive due the challenges added by the the not-well-established integration of these two subjects.

3.1 B2Expand

The B2Expand is a french company that uses its expertise in the blockchain domain to provide related solutions, such as the development of applications and consulting. The passion for, stimulated by the big revenue reported by this industry, made the video games the focus of interest of the company.

Created in January 2017, as a family project, B2Expand nowadays edits games that composes the proposal of an ecosystem based on its token, the Nexium.

3.1.1 General Information

The company is formed by a group of 3 co-owners, 13 employees and 4 interns. These people are distributed as following:

Figure 3.1: B2Expand's logo

- Chairman: Eric Burgel
- CEO: Manon Burgel
- CTO: Remi Burgel
- Director of Business Development: Marie Franville
- Lead Developer: Stephane Surget
- 3 software developers (+4 interns);
- 2 game designers;
- 4 graphics artists;
- 1 digital marketing manager;
- 1 administration/legal manager;

Some contact information:

- Webpage: http://b2expand.com
- Contact email: contact@b2expand.com
- Location: 19 rue Louis Guérin, 69100 Villeurbanne, France.

3.1.2 The Nexarium

B2Expand has the ambition to establish an ecosystem of games, called Nexarium, that turns around its token, the Nexium. In this scenario, a series of games is being designed to share a global context (the background history, for example) and they use the Nexium as coin. Then results from a game can have an effect in other one, such as a trophy (e.g. acquired by a very talented pilot in a race) can give some extra bonus in another game.

One of the main objectives of the Nexarium ecosystem is to share assets among games. This strategy of adopting cross-gaming assets allows the utilization of an once-bought asset in several games. So, the bought is registered in the blockchain (i.e. the asset belongs to the player), then this asset becomes available to use in any piece of the ecosystem.

B2Expand thinks about the Nexarium as an open environment where external companies can act, including their games. These partners would create assets that are trade in Nexium or they will use the Nexarium-natives assets in their games.

The Nexarium requires user authentication that is valid for all its components. The Nexarium account is hold by a server and these informations can be accessed any of the games.

Nexarium Store

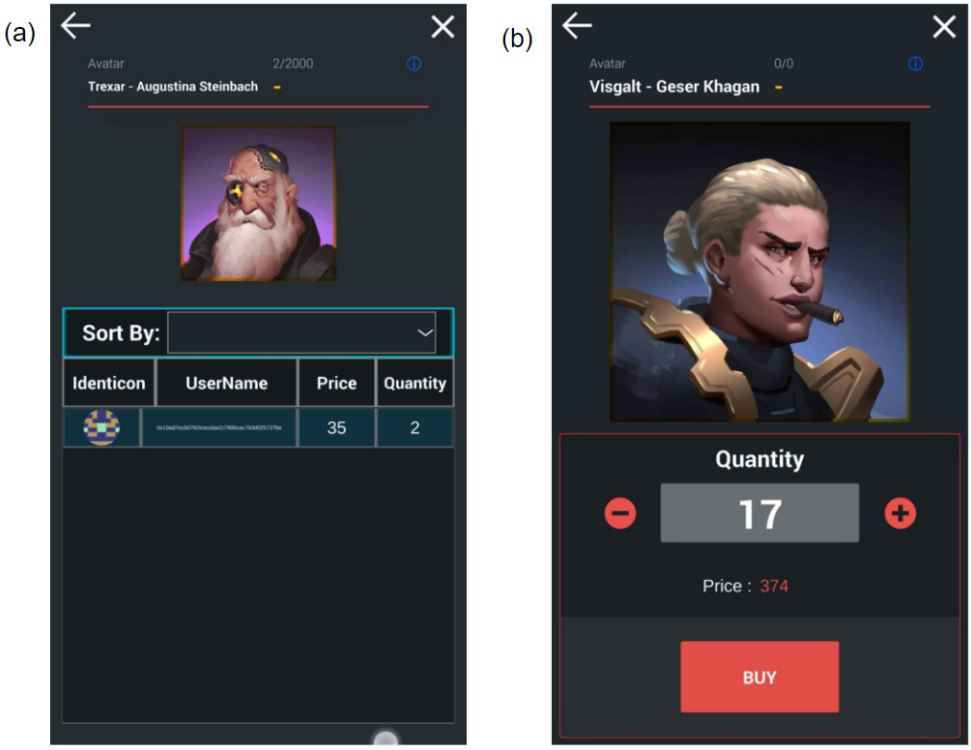
An important element of the Nexarium is the store that centralizes the assets trading. Normally, players can use standard assets available in the games. And If they want to acquire some extra assets, they must visit the Nexarium Store to do so.

Figure 3.2: The interaction among Nexarium elements. The games share information among them and the commerce is centralized in the store.

In this application, the players can perform some of the main actions mentioned below:

- Buy from The Corporation Shop: the corporation shop is the place where B2Expand publishes its assets to be sold. A type of asset can have a restriction on the amount of generated units, establishing some levels of rarity.
- Create a selling offer: once the assets is possessed by an user, this item can be exchange with others. So the user can create a selling offer that will be exposed in the Black Market. The user defines the amount of selling assets and the price to be payed. This offer is registered in the blockchain.
- Buy from the Black Market: The Black Market contains the list of offers related to each asset. When a buying is concretized in the Black Market, B2Expand does not take any part of the value involved in this transaction.

Figure 3.3: Nexarium Store screens for buying. (a) List of offers associated to an asset in the Black Market; (b) Screen to inform the amount of assets being bought.



The Nexarium store is responsible to handle player's Ethereum wallet information. In this case, this data is invisible from the games. The player registers its private key (that

is encoded by a wallet password and locally stored) and this key is used to launch the blockchain transactions.

Nexarium Games

Here, we list the games that have been developed by B2Expand and that composes the Nexarium ecosystem. Some other titles are arriving soon.

Beyond the void

The first game produced, Beyond the Void is a MOBA (Multiplayer Online Battle Arena) where the player has to conquest planets of a universe in perpetual motion. In a dispute 1 vs 1, it aggregates strategy and action.

The players controls a spaceship to beat the enemy and expand its frontiers. The planets can be upgraded with equipments that add more components to the game, such a military and economic rings.

Nexium Rush

Sharing the same universe of BTV, in the game Nexium Rush, the player assumes the role of a captain of a mining ship that explores the galaxy to collect minerals.

These resources can be sold in Nexiums. But, as the minerals have a variating value, according to market (like any other asset), the player has to choose the best moment to perform this action.

A ranking and badges reflect the players performance.

Star Scout & Hitch : Warp'Scape

The spaceships are again in evidence and they are the central elements of this game. Since the players can possess cross-gaming assets, they can use their ships to play this endless runner in their smartphones.

A warp tunnel is the scenario for this adventure. But navigating in hyper speed is not an easy task considering the asteroids.

The score is calculated based on the distance traveled by the player. Achievements and a leaderboard can be shared in social media in order to valorize players' victories.

Chapter 4

Executed Work

The main objective of the internship is to develop the Nexarium ecosystem. This means being part of a environment where new games are raising and the opportunity of improving skills in a emerging tool, the Ethereum blockchain.

B2Expand proposed to work in the development of the Nexarium Store mobile application. This software plays an important role for the company (explained in the Section Fundamentals 2). And the main missions were related to the redesign of the previous prototype (developed for web) and its implementation in Unity3D. Among the performed tasks, we can mention the upgrade of the user experience; the creation of an extensible architecture for the Store (that could integrate new games, for example); Manipulation/Interaction with Smart Contracts; the adaptation of the authentication plugin etc.

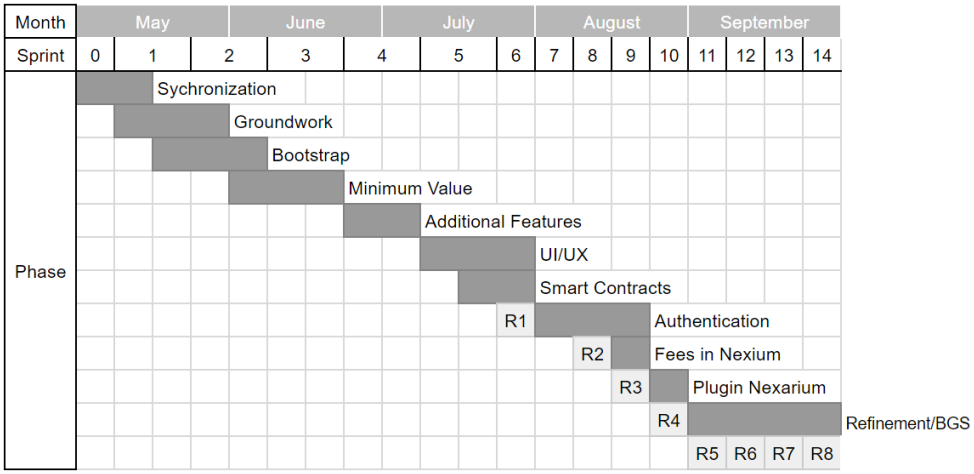
The initial objective established for the internship was to implement a new prototype until the end of September. Nevertheless, the good progress on the client application allowed us to integrate some other related projects, such as the exploration of a solution to pay the transactions by using the Nexium instead of Ether. The advancement of the product attracted the interest of the company to invite us to present an overview (followed by a live demonstration) during the Blockchain Game Summit (that will take place in 25-26 September).

In the next subsections, the organization of the executed work is explored. Next, we will introduce the tools applied during the internship, followed by the detailed description of the executed work.

4.1 Timeline and Methodology

In the starting weeks of the internship, we were able to define the scope of the project. Then the option for a development over sprints was inducted by the proposal of an evolutionary work. This evolution can be represented by the timeline for the internship, which is illustrated by the chart below.

Figure 4.1: Internship timeline.



Synchronization: this period is used to get introduced to the company's projects, objectives, internal process and tools. Some formation on these subjects, including concepts/practical issues of the blockchain, was conducted by colleagues.

Groundwork: during this phase, the internship's missions were clarified. The projects of the Nexarium Store was presented (among others) and we could define how the teams would be composed. In that moment, I was oriented to cooperate directly with Matthieu Eginard, as responsibles for the development of the store application. The interaction with Graphics Design and User Experience groups was also triggered.

The groundwork starts with the comprehension of the project requirements (section 4.3).

It is also important to establish some internal processes and tools that will facilitate the communication and knowledge diffusion during the development (section 4.2).

Bootstrap: the Nexarium project was already been started before the internship. A prototype of an authentication plugin has been developed, as well as a library containing the main tools for communicating with the blockchain. So the "Bootstrap" period was foremost useful to integrate and upgrade this previous work (subsection 4.5.1).

The first basic [but too significant for that period] result of this executed work was the obtention a list containing all B2Expand assets registered in the blockchain.

The central role played by the Nexarium Store in the Nexarium Ecosystem demanded some modularity and reusability from the proposed solutions. And that early moment was the best one to take those decisions in order to minimize the risks for the future of the project. Thus, some actions towards the long-term results were performed during this phase, for example:

1. the authentication plugin was transformed in a git submodule (like a software dependency). This way, it would be stored in an independent repository; and it could be integrated any other game of the ecosystem as an authentication manager.
2. A hierarchical architecture was designed and implemented for dealing with the category-based character of the store UI elements (section 4.4).

During the "Bootstrap", the work of specification of UI and UX was reinforced, as described in the (section 4.3.2).

Minimum value: for this period of development, there were established the core-features of the Nexarium Store. So, these elements should be prioritized with the objective of validating them in advance. The results from this proofing phase could be positive for developing other requirements. The key-features included: the exhibition of user's inventory (i.e. the assets owned by the user); the exhibition of Corporation Shop's and Black Market's offers; the user's response to an offer (i.e. buying); and the offer's creation by an user.

To achieve the missions of this period, a intense work on the comprehension of the smart contracts that manage the Nexarium Ecosystem had to be performed.

During this phase, we also started to put in practice the outcomes from the UI/UX validation started in the "Bootstrap". It means that the new UI Components and Panels had to be adjusted to the proposed architecture.

Additional Features: once the key-features were implemented, they let space for the development of accessory needs. The integration of a localization tool (that could

turn the Nexarium shop into a multi-language application) and the implementation of Filters are examples.

The refinement of previous products was always present through the phases. This is the case, for example, of the refreshing of panels based on blockchain events (section 4.4). It is a extension of the actions implemented in the "Minimum Value" phase, such as the buying of an asset .

UI/UX: until this phase, we had progressed on the development of several features. The specification of UI/UX, until that, followed the advancement of the project, focused on the features explored.

However, we had noticed that some of our objectives (notoriously, the "Authentication" phase, presented below) would be blocked by missing guidelines. That is derived from the need to upgrade the Nexarium authentication plugin (mentioned on the "Bootstrap" phase). This is due some new requirements that were raised during the development.

The decision that we have taken was to conduct a large work focused on these absent screens. The "UI/UX" phase was useful to clarify the user interaction, graphics resources and to produce prototypes etc. The executed work is better described in [30].

Smart Contracts: at the beginning, some subjects had to be considered out of scope of our Internship due time restrictions. This is the case of: the payment of transaction fees by using Nexium (instead of Ether); The new pricing system; and the KYC (acronym for Know Your Customer. It is a policy towards tracking money origins).

But based on the advancement of the project, we were able to extend our participation to these projects that are so important for the Nexarium Ecosystem. We were invited to collaborate on the design process, to implement the integration with the Nexarium Store and, in the case of the Pricing System, to edit the related smart contracts.

Furthermore, this period allowed the implementation of a transaction history in the Nexarium Store. This feature stores the transactions executed by the user and interrogates the blockchain about their status (section 4.3.2).

Authentication: as mentioned in the previous phases, the Nexarium Ecosystem had an authentication plugin that had to be upgraded. This plugin is basically a tool for communicating with the authentication server and to handle user's wallet information.

So, the "Authentication" phase was oriented to the adaptation of the plugin's UI and the implementation of some missing methods (Notoriously, the token management). A state machine was designed to control the user's session and respective screens (subsection 4.6.1).

Fees in Nexium: to contextualize, the development of the payment of transaction fees by using Nexium (instead of Ether), so-called Fees in Nexium, had became a priority. At that moment, its concept had been defined and its implementation was under progress.

The Fees in Nexium consists on the interaction between a client and a server, and then between the server and the blockchain (i.e. smart contracts). From the Nexarium Store's point-of-view, the request to the server must handle several factors, such as: the nexium price; the transactions' gas estimation; transactions' context arguments etc. In addition, this data must be encoded and signed.

This process required reflexion and exchange with the other agents. It is described in details in section 4.5.

Plugin Nexarium: a lot of work related to the plugin Nexarium was executed during the "Authentication" phase. However, at that moment, this work was directed to the Nexarium Store and the plugin must be useful in each piece of the ecosystem. So the "Plugin Nexarium" period was focused on the adaptation of the submodule to be used in other contexts.

In this case, the development was oriented by the integration of the plugin in the game Star Scout & Hitch [31]. So we could verify the external needs and adjust the plugin. These modifications vary from UI changes to reducing access to functionalities (section 4.6). A modular error handler for the plugin was also introduced during this iteration.

Refinement/BGS: the approach to the end of the internship represents the last opportunities to work on the pre-defined objectives. The tasks are oriented to the analysis/application of feedbacks and debugging.

The work must be presented at the BGS (Blockchain Game Summit) as a live demonstration. So, besides the production of the internship report, we add the preparation to this international event. This includes the acceleration of some key-features that must be presented as key-points for the company (even whether they will be presented as being under development, a small teaser must be ready).

So, September is all dedicated to these missions.

The distribution of the tasks for the internship was planned with the aim of frequently extract feedback from stakeholders. The iteration in small sprints, reproducing the application of an agile methodology [51], allowed to define short-term objectives that could be validated rapidly. Thus the product could be adapted to the needs (that are adapted in parallel).

The strategy involved some aspects from Scrum [22], such as artifacts and events. At every begin of sprint, a meeting for discussion on the previous work and planning the next sprint took place. The internship supervisor, the project supervisor (acting as a Product Owner) and some concerned agents were invited to participate. Sprint reports were produced to expose the results from the previous period and summarize the objectives. Besides the Jira tool (section 4.2), it served for documenting the progress. Daily meetings were conducted in order to share the progress and to exchange experiences (and questions).

The first sprints were executed in blocks of two weeks. This amount of time was beneficial to the learning phase. However, according to the evolution of the project, it was necessary to reduce this time span. At this moment, the software presented more functionalities and could tested more efficiently. So, the sprints became a one-week periods. The cycle of outcomes and respective feedback could be shorten.

The reasoning exposed in the previous paragraph agrees with the production of releases during the project. Once the software implemented functionalities that presented a certain level of maturity, this could be distributed for internal testing. These releases were accompanied by forms for user feedback, that included aspects like the general impressions, evaluation of UI, technical issues and questions that concerned the objectives of the specific release. In some moments, testing workshops took place in order to get a closer contact with users' responses.

The periodicity of releases is integrated in the Figure 4.1. The main aspects covered by them are divided as follows:

- Release 1 (R1): the basic functionalities were available. The user could visualize its inventory, buy assets and create offers. It counts also with the utilization of the UI architecture.

  The execution of R1 used Ropsten as testing blockchain network. However, the user could not change its wallet. As the wallet manipulation was not among the objectives of the sprint, a shared wallet holded Ethers and Nexiums to perform the transactions.

- Release 2 (R2): this version included the refactoring of screens related to the user profile and the basic interaction with the authentication server. It consisted on the first step for validation of the user path for the Nexarium accounts.

In this version, we provided a series of predefined wallets for the users. At the creation of its wallet, the user can select one of them from a list. So, the user does not have to ask for test Ethers or Nexiums. This alternative was implemented in order to allow the experience with the process of wallet creation.

- Release 3 (R3): contains extra functionalities involving the Nexarium account, such as "forgot password".

  The release 3 is also used to validate the gestion of the authentication token (see subsection 4.6.1).

- Release 4 (R4): the implementation of Nexarium plugin creates a new element to be observed in this release. The responsibility was splitted among the entities, creating conflicts between the main application and the submodule.

- Release 5 (R5): it brings the effective integration among the store and the games, via the Nexarium plugin. Until this phase, it was hard-coded because it permitted the quick validation of the previous aspects.

- Release 6-8 (R6-R8): these releases were not yet concretized before the writing of this report. But, in order to fulfill the planning, the last phases present releases devoted to the correction of errors and refinement.

4.2 Initiatives for Information Diffusion

Information is a very important resource for a project. So, maintaining a centralized base of information is a good practice during the development. Actions towards documenting and standardizing can save some time by answering frequent questions. Moreover, external or recently arrived agents can easily get introduced to the working processes.

In order to put this ideology in practice, some initiatives that were taken during the internship are listed below.

- Wiki page : a hub for all information related to the project, this webpage is fed with self-created tutorials corresponding to procedures that must be applied to install, integrate or use the tools that we developed; a list of links to external sources is also added to the wiki. This page is integrated to the project's git repository.

- Tasks Management : in order to control the progress of the project, the Jira tool [13] is used to manage the tasks. This tool provides a highly customizable scrum-oriented interface, that allows the integration of all company's projects.

Jira also offers a connection to the git repository. It improves the traceability of the tasks since each commit can be associated to its respective task.

- Code Documentation Generator : Doxygen [7] is a tool that is capable of producing code documentation from comments written in the project code. It creates a wiki page containing the documentation to guide the utilization of classes and methods.

  By using Doxygen, the programmer does not need to feed two different sources of documentation (code comments and wiki page). If no generator is employed, this situation can increase the information inconsistency between code, code comments and wiki, then leading to implementation issues. Furthermore, commenting the code is a task that requests a certain amount of work time (which is already normally short). So, the time that would be spent to write a second documentation is saved.

  A plugin for integrating Doxygen to Unity3D facilitates the document generation. However, Doxygen requires some training time because a specific format for the comments must be used.

- Blockchain Test Environment : the definition of a test environment creates a standard that makes easier the reproduction of tests and errors. When it is related to blockchains, it is even more important because it sets up the context of the blockchain, by handling the deployment of contracts, creation of accounts, distribution of Ethers etc. The adoption of a tool that manages all this factors can save time from development.

  In addition, executing transitions in the Ethereum blockchains requires the payment in Ethers. If it is applied to the main network, these payments mean money. So, the utilization of test blockchain networks (where the coins are worthless) or local setups is useful to save resources.

  On one hand, the deployment of a test environment in a test blockchain network reproduces all conditions that are applicable to the main one. On the other hand, we must collect Ethers from the community in order to pay for the transactions. In a local blockchain, the coins are automatically generated. In addition, the time to mine a block can be reduced. And if applied to a large number of tests, the amount of saved time is big. In summary, local blockchains are used in early-phase tests, when the implementation is not-mature and still needs a lot of validation. The more evolved the application is, it demands to get closer to the real context. So, the test blockchain networks are adopted.

  The elected environment for local tests was the Truffle Framework [18]. By means one of its tools, Ganache, the developer can instantiate a local Ethereum blockchain

and interact with it via javascript code. So, the Nexarium project has a set of scripts that deploys the contracts and add some test data (such as, the available assets and offers). These scripts are often extended to represent emergent testing needs.

4.3 Requirements Gathering

The work of defining the objectives of the project starts by the requirements gathering. A series of procedures was performed with the aim of collecting the needs of Nexarium, translate them into features and filter them according to the scope of the internship.

Although the initial period of project's timeline is dedicated to this phase, the work is conducted through the whole work. This represents the ambition to clarify the view over the project, at the beginning, and after execute a iterative evolution based on feedbacks.

The initial requirements gathering can be categorized in 3 major axes : interviews, previous version review and mockup. To these actions, we can also aggregate the prototyping (in the sense of producing functional code) as the tool for later catching opinions.

4.3.1 Interviews and Review

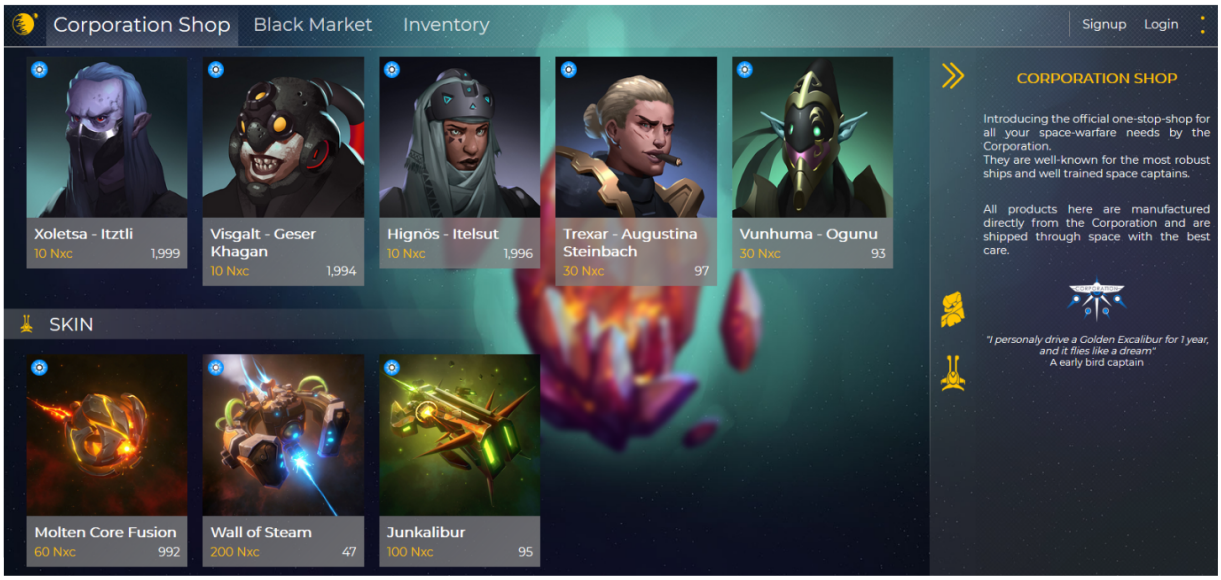
The interviews involved people who designed the idea of Nexarium, who could express their intentions and wishes for the ecosystem. This procedure was strongly connected to the evaluation of the previous prototype of the Store, which had two versions : web interface and mobile interface.

As general requirements for the Nexarium Store, we can mention that, despite the potential of utilization by traders, it is thought to populated by gamers. This means that the user experience must be adapted to this public (specially by dealing with the implications brought by the blockchain technology).

B2Expand wants to valorize the concept of ownership related to the assets (the possibility to sell the asset is an example). This comes together with the presence of several games in the ecosystem, that allows the cross-gaming aspects. This plurality brings the requirement of extensibility to the store. It must be able to accept new games and assets, from B2Expand or its partners.

Among the collected information, besides the topics already discussed, the following points show some of the prerequisites:

Figure 4.2: The old version of Nexarium store: web interface.



- The application must handle the creation/management of the Nexarium account, as well as the wallet. And this should derive a module to be integrated in each associated game, as an authentication tool;
- Mobile and web versions;
- It must allow the interaction with Metamask (a tool that manages the wallets and manipulates blockchain transactions);
- Multi-language application;

The necessities were converted into Use Cases. This brief documentation was useful to a preliminary validation. At this moment, we could determine that some requirements would be considered out of scope due time restrictions (For example, the web version remains as a stipulation, but it did not advance in implementation. The structure for the multi-language UI is implemented, but there was not enough time to produce the translation to other languages but the english).

Blockchain Implications

Among the requirements of this project, a special attention must be devoted to the restrictions/conditions imposed by the blockchain. The option for this technology is due a lot of positive sides, but, for some of them, a counterpart must be afforded in order to assure its good application. These barriers are highlighted because they can produce a disaffection on the regular game player (the target public):

1. Time to Buy : users are used to have an immediate return of their actions. But in the case of the blockchain, the transactions are not instantaneously proceeded. E.g. if someone buys an item, they must wait until transaction to be accepted in order to confirm it;

2. Different Accounts (Game and Wallet) : a actual decentralized interaction with the blockchain requires that the user owns a wallet. It can be translated into the need of dealing with distinct account information at specific moments (e.g. two different passwords: one for the Nexarium account and another one for the wallet);

3. No central storage: the client-server paradigm made users get used to have a safe point for their authentication data. If they lose their password, for example, they can recovery it with no big obstacles. However, in a decentralized blockchain application, the user is asked to keep it own credentials (the private key or mnemonic). This create a risk of loss of information (that almostly every time means loss of money).

4. Two currencies : in the case of Nexarium, the Nexium plays like a coin (as a exchange token). Moreover, Ethereum has its native coin, the Ether. When a transaction is triggered, for instance, the user can be asked to send some Nexiums (to pay for the asset buying) and some Ether (to cover the transaction fees). In this context, the user can get confused about which coin is being applied, value conversion etc. It also creates the obligation to maintain a budget containing both currencies.

5. KYC (Know Your Customer) : the companies that work by selling items in the blockchain are asked to justify the origin of those funds. So they are forced to request to users some information in order to keep a track of the money that is spent in their applications.

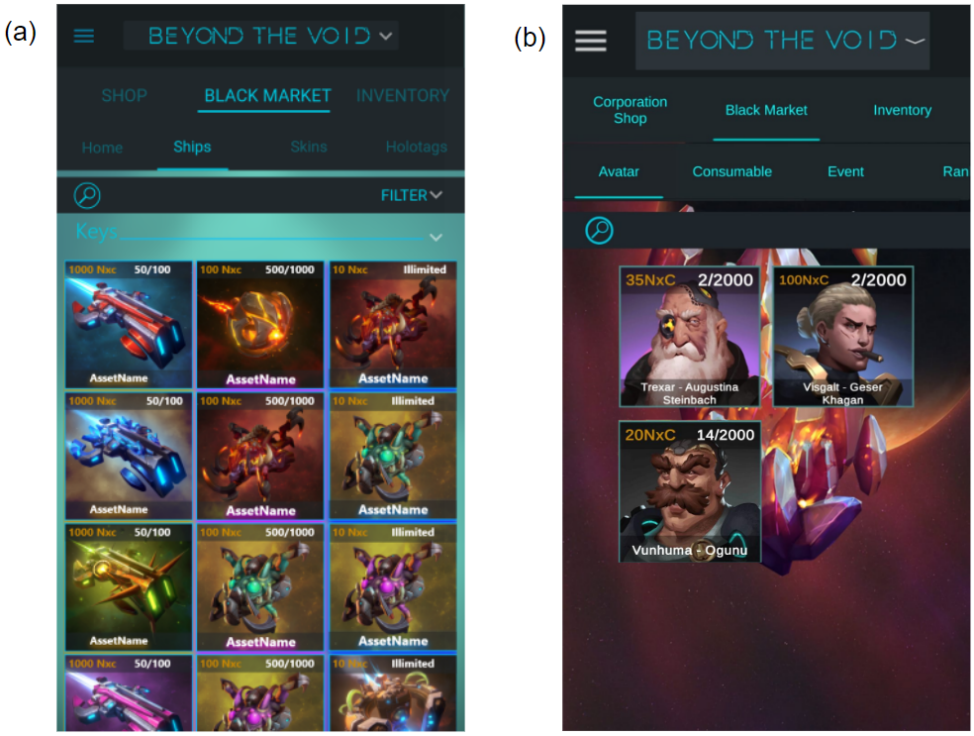
4.3.2 Mockup

Once we could establish a todo-list, we were able to apply a tool for graphically introduce our perception of the product: the mockup.

Since we are "drawing" the screens, by means the mockup, we could avoid wasting time with the implementation of unnecessary functionalities on Unity3D. This is useful to rapidly collaborate with the stakeholders in order to achieve a consensus. The outcome from this process serves as documentation for the implementation in Unity3D.

In this phase, Adobe XD [1] was chosen as the software for prototyping because it allows the construction of a user path associated to the user interface.

Figure 4.3: Comparison between the mockup version (a) and its respective screen in Unity3D (b).



Design of the User Experience

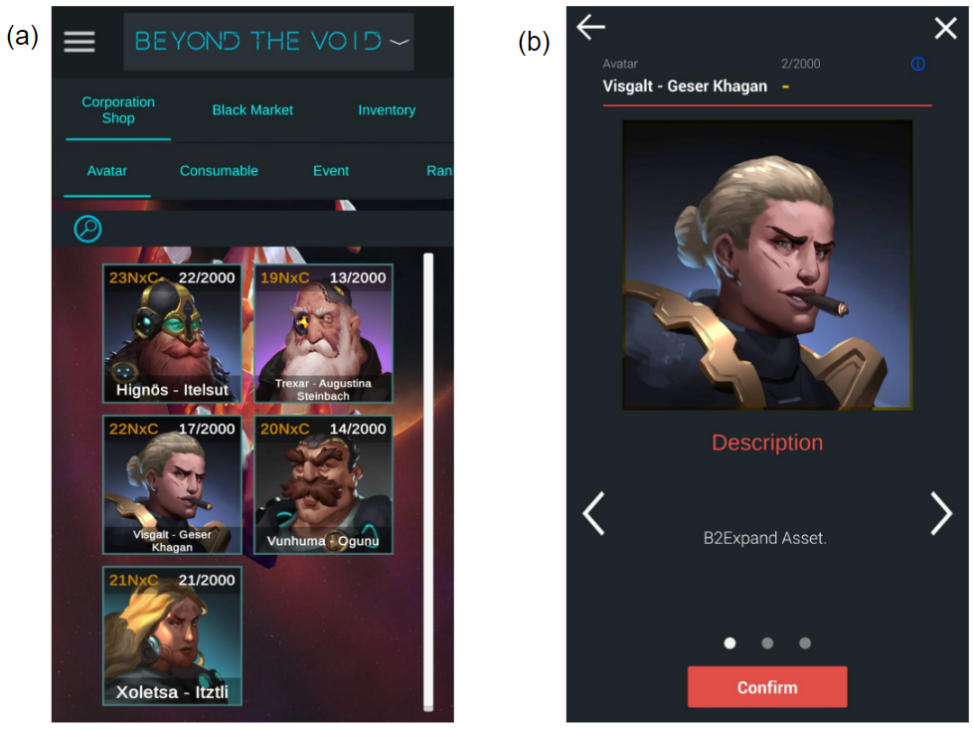
During the elaboration of the mockup, we had time to search for the experience that could answer some of the questions from the requirements. This proposal of experience is illustrated by the application screens and described in the following topics:

- Gamer Oriented : the mission of orienting the experience towards the gamers can be visualized as a two-way effort : adapting and teaching.

  In one sense, the UI must reproduce the environment that the gamers are used to (such as the usual game stores). This is reached by abstracting blockchain terms, for example.

  The asset is presented as in any other shop; Informations like its address are skipped (even if those informations are always available for expert users);

Figure 4.4: Nexarium Store screens. (a) The Corporation Shop; (b) Screen containing the asset's description.



In the opposite, for the things that cannot be "hidden", it must be offered to the gamer the smoothest possible way to perform the action. And here enters the teaching

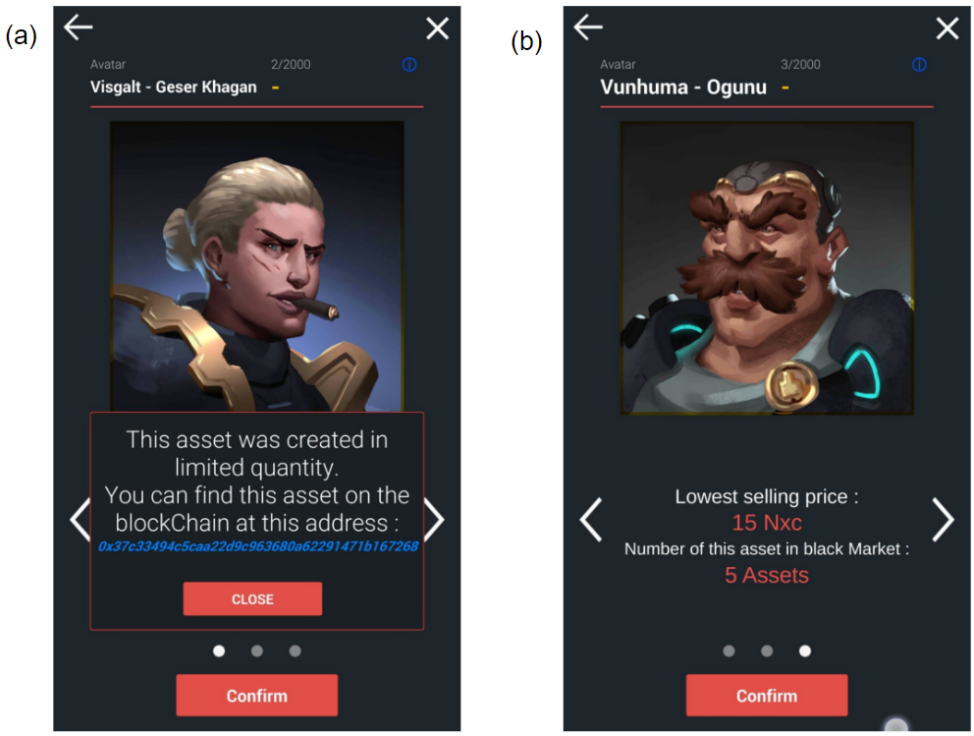
role, that will be explained in more details in the topics related to the "Blockchain Implications". An example of this scenario is the explanation about the need to pay for transaction fees, even if B2Expand does not take any profit on some transactions.

Figure 4.5: Explanation about the need to pay transaction fees.

Anyway, the "low-level" blockchain content is maintained as accessory functionalities. This is useful for reaching another part [not mutually exclusive] of the public: the traders (i.e. people that use the blockchain as stockage exchange) or people that are used to manipulate the blockchain. They have the opportunity to visualize the transaction history and assets' addresses, for example, as well as recover their previous wallets. Such features can also be seen as stimulation to the knowledge acquisition by the blockchain novices.

- Ownership : the presence of "Inventory" and "Black Market" tabs are aiming to integrate the concept of ownership to the Nexarium Store. Their functionalities express the possession and the capacity of exchange. The "Inventory" shows all assets owned by a user, while it gives the opportunity to publish an offer containing

Figure 4.6: Functionalities for expert users. (a) A "info" button opens a popup with the information; (b) Extra tabs can be added to the asset description in order to exhibit useful information.



this assets.

The description of each asset always bring the amount of existing instances of the same type. This information aggregates the concept of rarity to an asset, by adding a variable to the ownership issue.

- Cross-Gaming : this aspect can be splitted into two factors. The first one is the exposition of several games in the same application; The second one is the exploitation of the presence of the same asset in different games.

  In order to expose the multiple Nexarium games, we defined the possibility to navigate among them. The user scrolls the list of games and choose one to open its store. The selection adapts the interface according to the game context. For instance, the fonts, colors and background are adjusted. The adopted UI Architecture allows an even more intense adaptation, by changing the format of UI elements, their disposi-

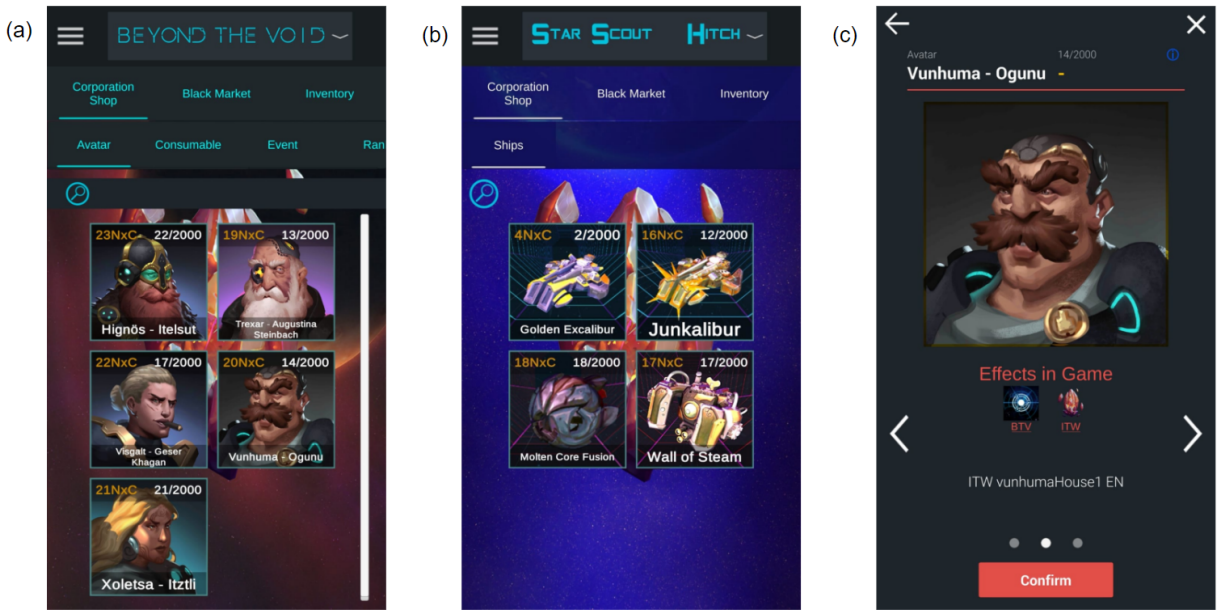
Figure 4.7: Representation of an offer in the Corporation Shop. This image highlights the amount of available instances of this asset.

tion etc. However, the examples produced during the internship were reduced to the basic characteristics.

If an asset is present in a game, it will be available in the game's store. Further, when the asset is selected in any context, a tab containing its related games is displayed. A description of the importance of the asset in each game is also presented.

From this feature, the user can have a fast comprehension of the cross-gaming aspect of the assets. In addition, all this interactions can facilitate the advertising of newly integrated games, increasing the advantages of being part of the ecosystem.

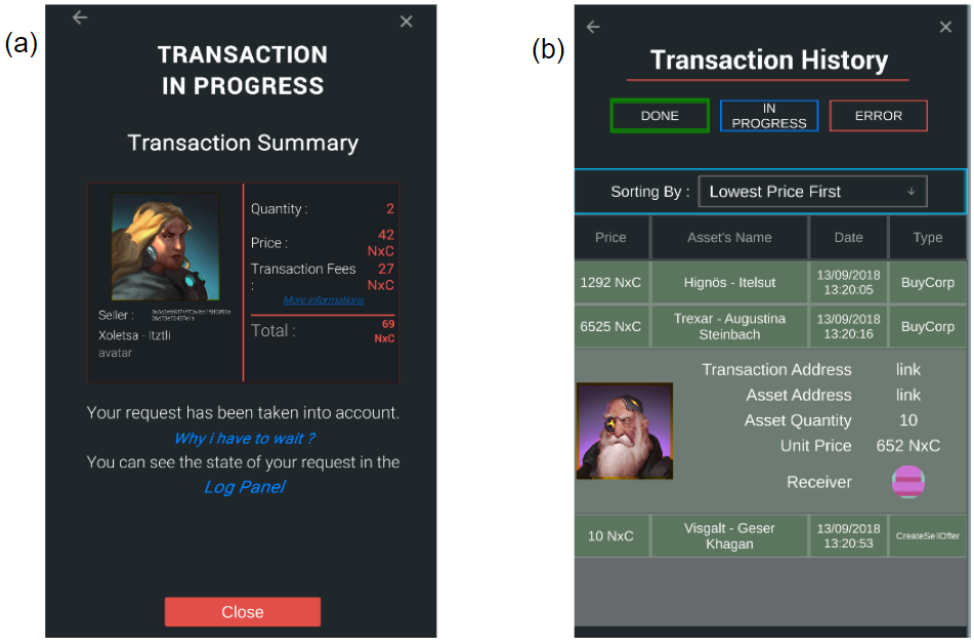
Figure 4.8: The distinction between two shops: (a) Shop BTV and (b) Shop Star Scout & Hitch. (c) The game-related description of an asset.



- Time to Buy : the fact of waiting for the concretization of a transaction is unavoidable. So, the strategy adopted in this case is the explanation about how the process is being proceeded and what is your role. The education is a key to answer the question "why this 'weird' process is taking place?" and "what must i do now?". This approach is focused on the concientizacion, based on the advantages that the blockchain can bring, even if there is a "cost" associated. Actually, this methodology is generally expanded to the "Blockchain implications", such as the transactions fees.

  For the specific case of the "Time to Buy" problem, besides the description of "what is happening" (as we can verify in the Figure 4.9), the user is oriented to visit the Transaction History to follow its transactions. This tool tries to contextualize the transaction's information as part of the store, with the objective of facilitating the comprehension.

Figure 4.9: (a) Screen containing orientations about the triggered transaction. (b) Transaction History.



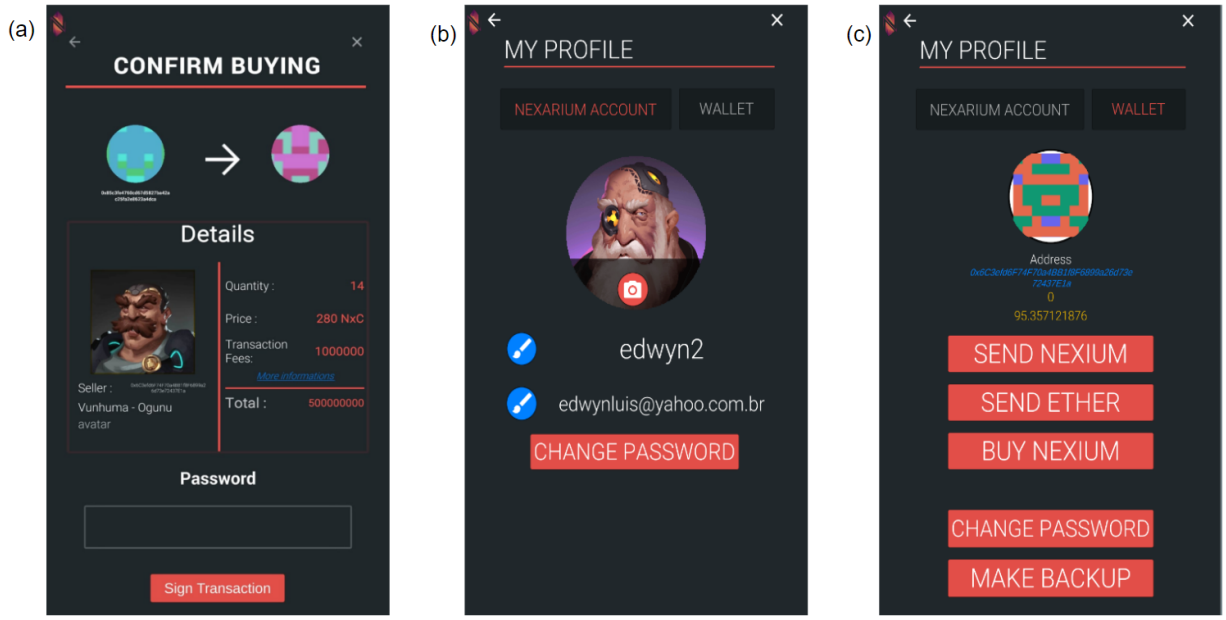
- Different Accounts (Game and Wallet) : this aspect of the "Blockchain implications" must be handled by well defining the action perimeter of each account. The Nexarium credentials is relative to the access/manipulation of profile information. It has a larger application, since it will be useful in all the elements of the ecosystem. The

employment of the wallet is restricted to the interaction with the blockchain, in actions like the buying/selling assets.

The objective of the user experience is to distantiate the use cases of each account, in order to clarify their functions. In this context, the authentication plugin centralizes all interactions with the profile data, serving for the Nexarium Account. It contains a specific graphic interface, that helps the distinction to other store functions. Moreover, this graphic identity will be carried into the games, which will help to fasten this concept. Meanwhile, the usage of the wallet is drastically less recurrent. Notwithstanding, it is usually followed by an explicit explanation of its context and importance. These messages also serve to warn about the critical characteristic of the action, that normally involves money spending.

It is always important to mention that the wallet is not mandatory for executing or playing the games. So, the wallet creation is posed as an optional step on the account creation process. In accordance with the fact that the wallet is facultative, a functionality to create it at any moment is available.

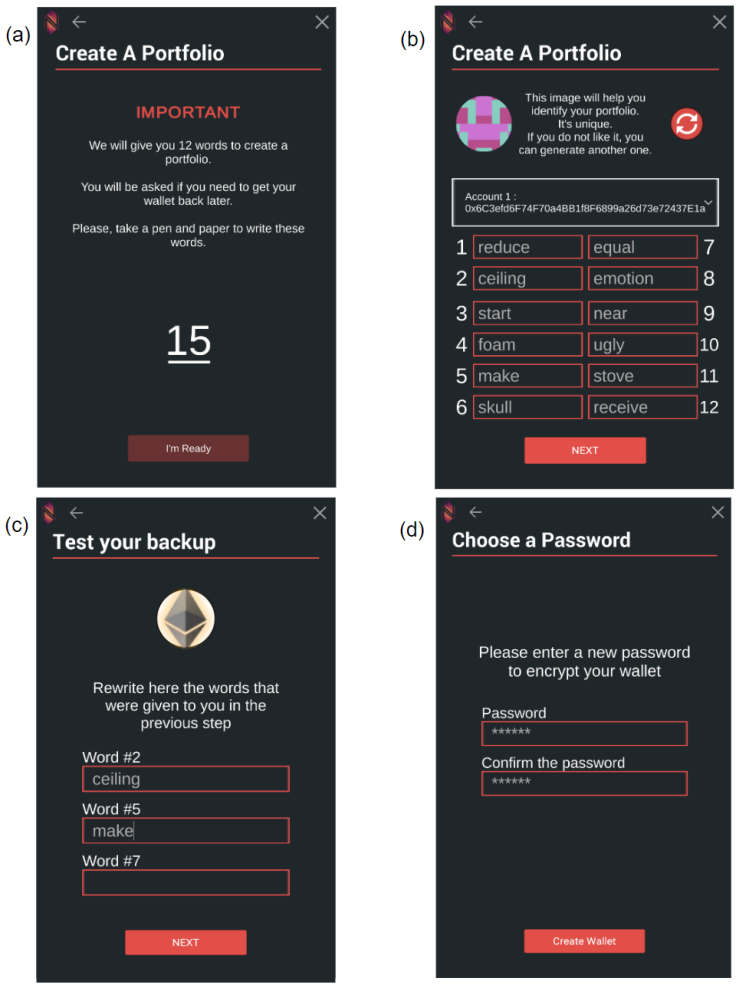
Figure 4.10: The wallet password is demanded to each transaction (a). So, the distinction between the entities must be clear. The profile screen splits the concepts: (b) the Nexarium account and (c) the wallet information.



The presence of a wallet can disclose another interesting UX issue: the gamer users are normally not used to create a wallet to manage its assets. So they do not master the related concepts, nor the procedures to create one.

Based on this problematic, the Nexarium Store guides the users through the creation of their wallets. Analogously, informative messages try to elucidate the context.

Figure 4.11: Orientations for the process of wallet creation. (a) Warning message; (b) Creation of Mnemonic; (c) Testing for reinforcement; (d) Wallet password.



- No central storage : an option for trusting in the users is done when application asks them to hold their own wallet credentials (private key or mnemonic). So, the role of

Nexarium is to educate about the advantages of this option and the importance of keep the information safe. It must be clear that the misuse can leads to theft or loss of values. However, we have to be careful in order to prevent from fear of utilization.

The idea that the application does not keep the integrality of wallet information must be reinforced. With this in mind, one of the steps of wallet creation is a test to verify if the user has written down its mnemonic (Figure 4.11(c)). This is put into practice by requesting some random words among the ones that compose the mnemonic. Not only as a test, this method is interesting because it fortifies the alert.

- Two currencies : allowing the payment of the transaction fees in Nexium was one of the most important requirements for the store. Looking from the perspective of the user experience, this feature represents a step towards the facilitation of the buying process, since the user does not need to buy two different tokens and convert them frequently in the store.

  To deal with this problem, the solution involves the integration with smart contracts and other intermediary elements. The technical role played by the Nexarium Store application will be described later in this report.

  However, in the context of user interface, we can mention that all informations related to prices are described in Nexium: prices for buying assets, prices indicated for creating selling offers, transaction fees, discounts etc. This can be noted, for instance, in the Figure 4.9(a), where the transaction fees are shown in Nexiums. The presence of two tokens would request more "screen space" and a special attention to differentiate them.

  This tool requires the participation of a very important element that makes it even more powerful. The fact that the Nexiums could be bought in actual money (Euros or Dollars) allows the whole buying/selling cycle be involved by the Nexium. This feature is also in progress, mainly because it depends on external agents.

- KYC (Know Your Customer) : thinking about this mandatory process, we proposed a tool that handles the amount of money spent by the users, categorizing them according to their expenses. The more information the user provides, the more money it can spend.

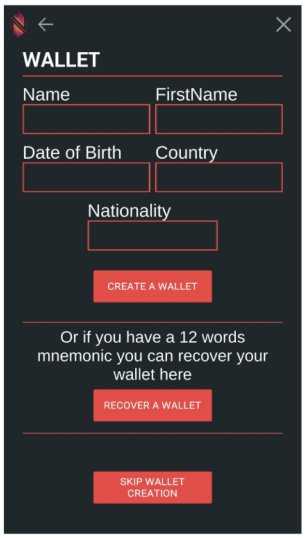
  The acquisition of KYC information is generally integrated to actions on Nexarium account or wallet. Basically, the first phase is the creation of the Nexarium account. The user informs its initial information (such as email address and nickname). After this phase, the user is considered at level 0. This level actually does not allow the user to buy on the store. This allowance is granted when the user registers its

wallet (create a new one or recover it) [Anyway, he could only perform blockchain transactions after having a wallet]. This action is preceded by the insertion of some data, such as its name, birthdate and address. Now being at level 1, the user can buy in Nexarium store, however, it has a value limitation per month. To go further, the user can include more information (e.g. a passport photo) and then have the budget expanded.

According to the user's KYC level, messages will be prompted when transactions are not permitted.

The implementation of KYC was not fully integrated to the internship, but as a important requirement, its needs were considered during the UI/UX validation.

Figure 4.12: Screen requesting KYC information before creating a wallet.



4.4 User Interface

The implementation of the User Interface is the concretization of the design defined for the User Experience. But it is not the only requirement for the UI. As a complement, the proposed solution must be ready to engage new games, shops, assets and others.

The overall development was performed by using Unity3D UI Elements [19]. Unity3D provides a series of predefined widgets to handle UI, such as text fields, images, scrolling

panels etc. The adaptation to the distinct devices' screen size is facilitated by means a "Anchoring" system, that modifies the position and dimensions of UI elements.

Unity3D serves also to produce builds for multiple platforms. With this option, the application can reach a larger range of users.

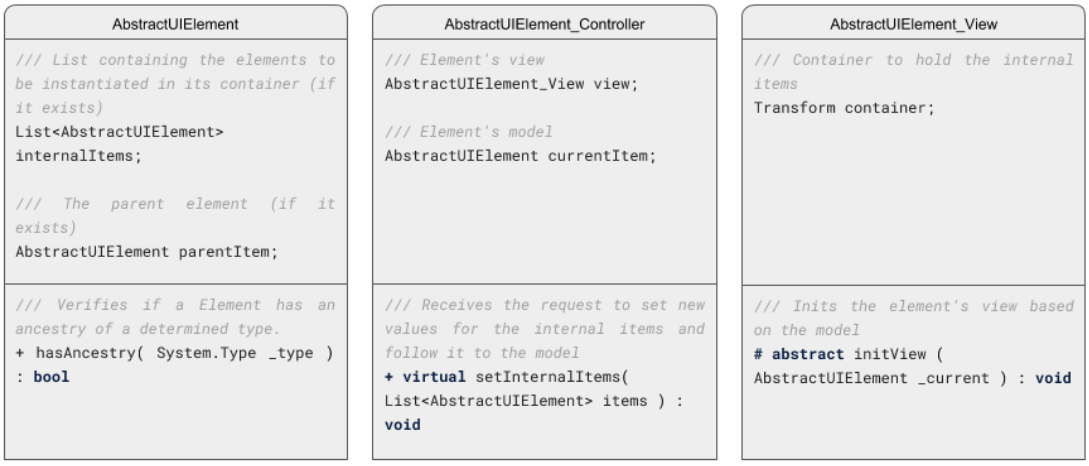
UI Architecture

The description of the solution can start by the application of the architectural pattern MVC (Model-View-Controller) [50]. This pattern aims to isolate the logic (Controller), data structure (Model) and representation (View). So the general implementation is facilitated since the responsibilities of each component is clearer. We can also mention that the independence among the parts allows the modification of a piece without prejudice to others.

The issues of extensibility and modularity are explored in the conception of an architecture based on a hierarchical arrangement of components. This construction is grounded on a abstract class that represents a UI Element (name AbstractUIElement).

Basically, the AbstractUIElement class determines the form of relationship between the UI elements, where they can have children (also possibly one or none) and a parent. This characterizes the structure of a tree of AbstractUIElements.

Figure 4.13: The fundamentals of AbstractUIElement.

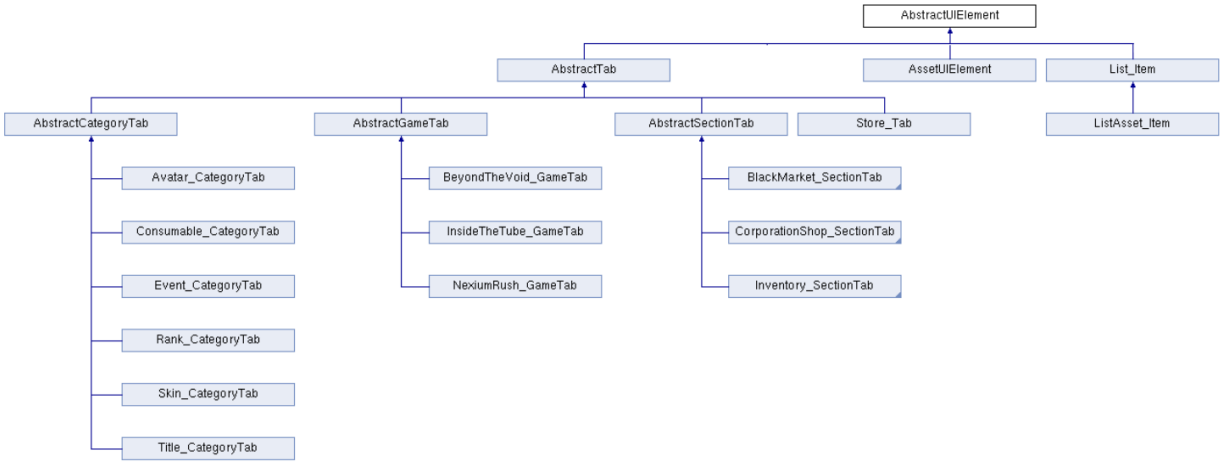


The concept of class specialization allows the creation of a variety of elements that

respect the hierarchical principle. And the resulting structure can contains nodes that implement different behaviors (even if they are in the same level of the hierarchy).

The following class diagram (Figure 4.14) shows some of the classes derived from AbstractUIElement.

Figure 4.14: The hierarchy of AbstractUIElement.



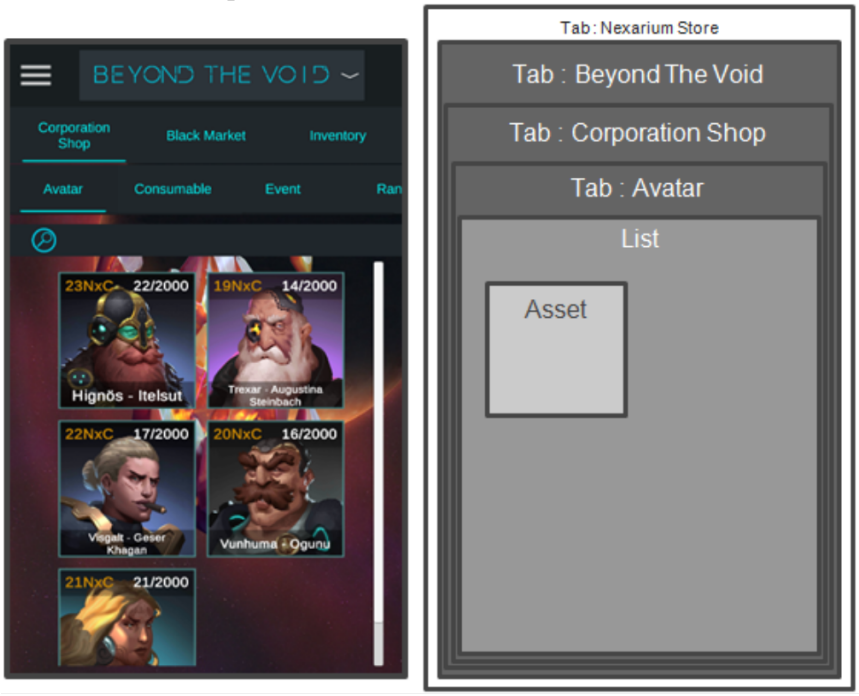
To better understand the application of the current architecture, the subclasses present in the class diagram (Figure) can be described as follow:

- Asset (AssetUIElement): the representation of an asset (or offer) in the application, it can be considered as the terminal node of the tree.
- List (ListItem): represents a list of objects. In the implementation of the Nexarium Store, it serves, for example, to hold AssetUIElements>.
- Tab (AbstractTab): the mission of a Tab is to offer a list containing a reference to its children, by allowing that the user selects one of them. So, it will show the chosen one as its child.

However, the schema presented by the Figure 4.15 is not the exclusive way to use these elements. The structure can be rearranged and adapted in order to confront several issues. Below, we describe the possibilities that this architecture proposes:

Cascade construction: the tree structure permits the user interface be generated in cascade, reacting to modifications in the model. In short, the UI represents the current

Figure 4.15: The correspondence between the user interface and UI elements.



state of the model tree. So, the architecture can deal with multiple depth, by adding new levels or removing them.

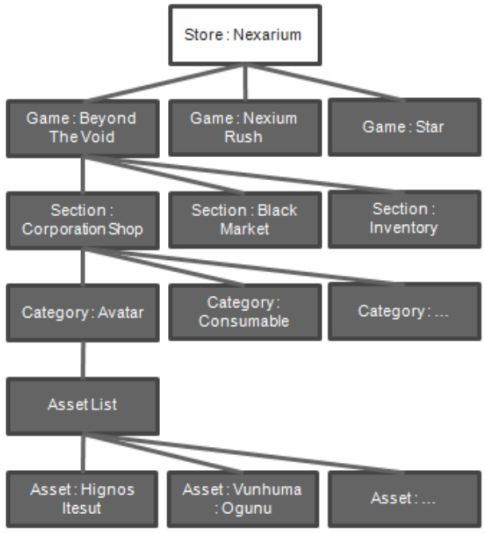
Other factors can affect the UI construction. Since each element of the tree can implement its behavior, they can present their own perception of the structure. This means that an intermediary node can decide to expose just a part of its children. So, the user interface will be composed of a tree branch.

This concept is put into practice by the tabs, for example. They are able to display the selected child and can wait to generate the other ones when necessary. The Store Tab, for instance, is a Tab that shows the selected Game Tab, providing the **multi-game** aspect of the Nexarium Store.

Dataflow based on propagation: although the structure can be simply described as a top-down hierarchy (the higher component is the more important), the way that the information is propagated deserves a topic devoted to. Not restricted to the steps of propagation, it is also directed to the implications and applications of this method.

The basic issue that is treated by this solution is where the UI elements can find

Figure 4.16: The tree that represents the interface of Figure 4.15 (i.e. its model).



information in the case that they do not own it. The answer is based on the responsibilities of the nodes. So, there are two options : 1) A node is able to generate the information; 2) A node can ask its parent for the data.

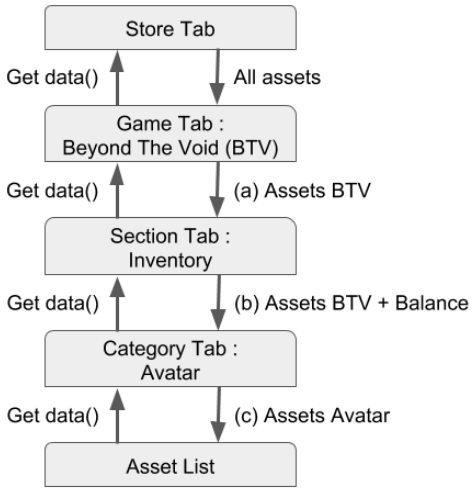
To illustrate this, we can use the Figure 4.17. It brings the example of a recently created List component that needs a list of assets to work with. Then, the List demands the information to its parent. In the List parent's turn, it searches for the data in its respective parent and so on. These steps are repeated until some entity is able to provide the information (In this case, the Store Tab).

The definition of the UI Elements fonts/colors is also executed by applying this path: the element asks for someone who is able to define the values. In Nexarium Store, as we want that all elements reproduce the game's environment, the Game Tab holds this information. So, all its children will use these characteristics. Each subclass indicates its own values, creating the distinction between the games (Beyond the void, Nexium Rush etc.). However, the most important is the fact that this reasoning can be applied at any level of the hierarchy.

It is necessary to mention that the process shown by the Figure 4.17 allows that each intermediate node applies its own perception to the data, according to its function. So the data that is transferred can be modified, creating **context-based information**.

The UI elements can perform several kinds of operations over the data. Actions like filtering and sorting are the simplest ones. Typifying this (Figure 4.8(a)), a Category

Figure 4.17: Data Flow. A element at the bottom of the tree can obtain information from the top.



Tab, such as a Avatar Tab, has the role to select only the assets that correspond to the "Avatar" type. So, they can decide to forward the ones which are convenient.

The Lists have a similar mechanism because they provide user-input filters, such as a searching field or ordering dropbox (Figure 4.19). But as these kind of filters need to be easily replicated in different settings, the IModifierHandler interface (Appendix A) defines the needs to any List to handle such filters (called List Modifiers).

The Section Tab presents a distinct behavior by applying a more complex operation (Figure 4.18). This type of Tabs uses a superclass of Asset in order to add some extra information to them. So it instantiates a new list containing the extended data and send it to their children. In the case of Inventory Tab, the extra information is the balance of each asset; The Corporation Shop Tab and Black Market Tab aggregate a list of offers related to each asset.

Listeners

The sense of the interaction can also be inverted. An element at the top of the structure can send data to its children. This is useful, for example, when it wants to inform modifications on the data. A notification about an asset that has been bought or a user-input filter that has been changed are situations where this procedure has to be executed because it requests the update of the UI.

For this context, we decided to apply the Observer design pattern [32] with the aim of employ the notion of dependency. This process is described as: 1) An UI element subscribes for notifications using its parent's Listener Handler (Appendix C) (or other source of data); 2) The parent will spread the data when it is necessary, by calling the listeners; 3) So, the current element interpret the call and forward the "message" to its listeners. So, this procedure can make the data reach the tree leaves. The IGetData interface (Appendix B) establishes the needs for the interaction between elements.

The listener calls require that the elements verify any modifications on the model. Based on it, we can consider that the structure is projected to create/update itself based on changes on its model.

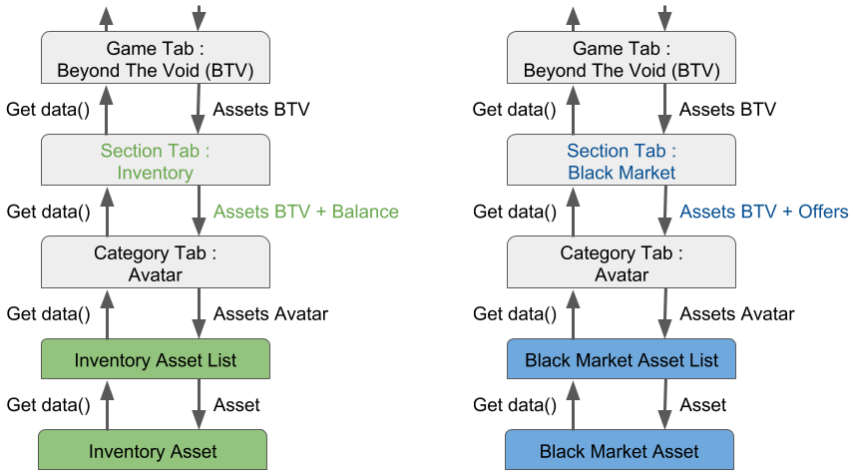
In this scenario, a new entity appears: the Data Manager. It is posed as the highest element in the listeners tree. The Store Tab (which is the highest one in the UIElements tree) is subscribed directly to the Data Manager in order to diffuse the assets information through the UIElements tree. Some other independent components can also subscribe to the Data Manager. So it is a hub for data, that avoids inconsistency among different contexts. UIElements that are intermediary nodes can also directly subscribe to Data Manager in order to obtain different information, like the Inventory Tab that listen for changes on the balance of assets.

The interaction performed through the listeners is also useful for keeping the panels updated. A panel, such as the one which shows assets' details or related offers, must be informed when some event occurs. So, these panels (that are not subclasses of AbstractUIElement) register themselves as listeners of the assets (which are the tree leaves). If some modification on the tree model is related to the asset represented in the panel, this information will also be repercuted in the panel's context.

Diversity of Views: the role that each component plays in the UIElements tree can create context-based information, as mentioned in the last topic. This kind of information can be illustrated by Figure 4.18. There, each Section Tab acts differently from the others, by adding extra information to the list of assets. It creates two distinct types of content: in one side, information oriented for the "Inventory"; in the other side, information for the "Black Market". This interference changes the data data perceived by its children. Now, the offspring have to deal with modified assets. In some situations it does not mean too much, like the Category Tabs, that do not need the specific-created information. However, for the Lists, the variety of data require the creation of new filters and specific asset representations, for example.

Actually, the context-based information is generated in each step of the data flow.

Figure 4.18: The context-based information. Focused on the participation of the Section Tabs.



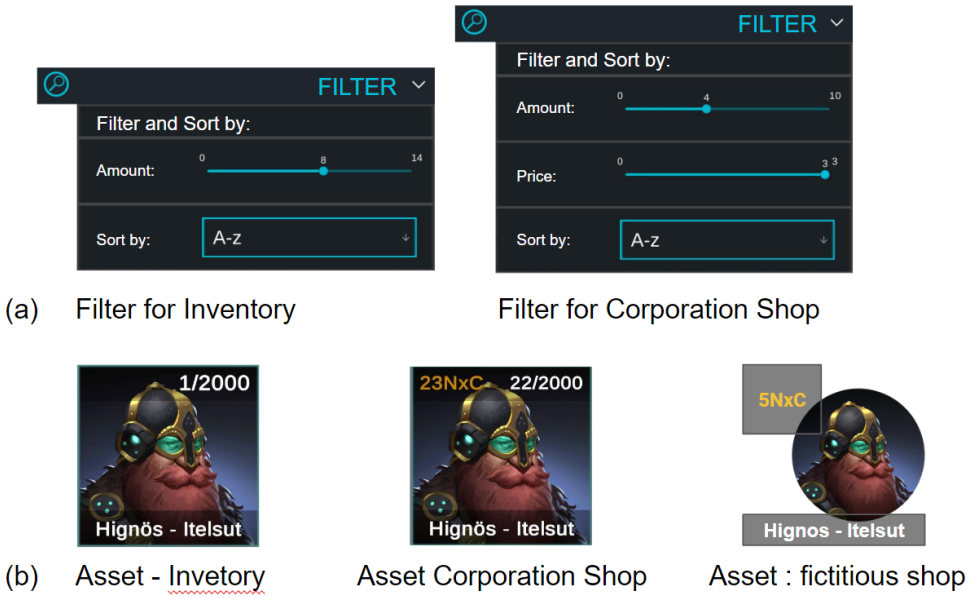
This happens because each component can contribute to the generation of a new context, modifying the data according to this perception. The simple task of filtering or sorting a list is a modification of context for the children.

So the context-based information is the basis for the diversity of Views. Since each tree branch and the data in transit characterize a different context, the layout that represents each UIElement can vary, be customized, adapted to the context.

The implementation of this feature can be reached by two methods: 1) a unique UIElement is capable to express all possible contexts. So it adapts itself to the current needs; 2) We can apply an adaptation of the Factory method, that put into practice the modularization. It is described as useful to "eliminate the need to bind application-specific classes into your code" [32]. In this option, the parent calls a Factory in order to obtain an instance of the UIElement that is adapted to the context. In Unity3D, this strategy allows to manipulate different prefabs, each one containing a View for the component.

If a variety of Views is not very important to the solution, the unique component is easier to implement. It centralizes the code, keeping the implementation simpler. In the case of Nexarium Store, the user experience specification requires a uniformity on asset representations in order to ensure the coherence among them. Although the Factory method was not too much necessary for manipulation of Views, it remains as an alternative for customization or future shops that will integrate the ecosystem.

Figure 4.19: The distinct views of the same type of UIElement based on the context. (a) The user-input filters associated to a List. They vary, for example, in the amount of fields. (b) The assets can present different information or a different layout disposition [The example of a fictitious game is merely illustrative. It aims to expose possible applications].



However, the Factory method procedure demonstrates its important participation on the "Cascade Construction". The principle behind the AbstractUIElement is to call a Factory in order to populate itself with respective children. The Factory takes charge of deciding which UI Element will be instantiated and lets the AbstractUIElement independent of context-specific behavior.

Multi-language

The context-based information can also applied in another circumstances. As a more specific application of the Views adaptation, explained previously, the process of multi-language manipulation benefits from this feature to determine which term must be shown.

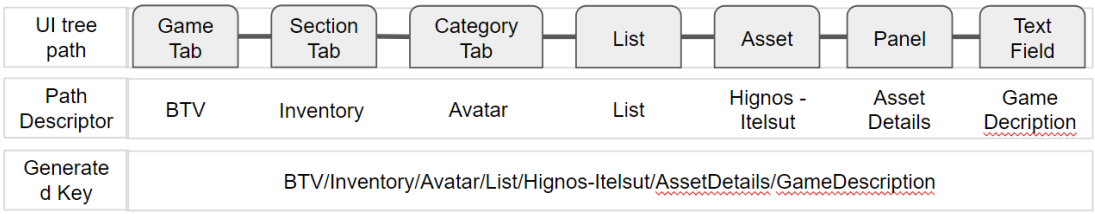
To describe the scenario, the implementation of Nexarium Store uses I2 Localization [12] as a tool to manage the multi-language exhibition. I2 includes a plugin to Unity3D, where we can define terms (as dictionary keys) that represent entries in

each selected language. It provides an easy procedure to define terms, translate them and apply it all to Unity3D UI components. Some of its most important features are the integration with Google Drive spreadsheets (to store the dictionary of terms, and that makes possible the collaborative work over the data) and with Google Translate (To perform automatic translation of terms, if desired).

I2 requires the association of a term to each text component in order to define what will be shown when the software is executed. Normally, this link is established by means a simplified interface with Unity3D, where the developer selects a term in a list of possibilities. I2 is capable to detect the device's current language (and make use of it, if the language is defined in the dictionary) and we can define a default language to be used in the case of missing term translation.

But, the definition of the term to be applied in a text field is not very simple in the Nexarium Store, due the different possible contexts. The Figure 4.8 brings an example of this setting. It is illustrated by the text to be shown as the game-related description of an asset. This value can vary according to its ancestry: the asset "" can have distinct effects in different games, for example. So, the key for the text is obtained at runtime, by constructing a string with the elements that compose the path in the tree.

Figure 4.20: Construction of a key for the game-related description of an asset. The path is obtained at runtime based on the ancestry of a UI Element.



A searching mechanism is implemented to split the key and look for substrings in the dictionary. With this feature, a key can be generalized and reduce the amount of entries. Then, the generated key "BTV/Inventory/Avatar/List/Hignos-Itelsut/AssetDetails/GameDescription", from Figure 4.8, can be found as "BTV/Hignos-Itelsut/GameDescription", with a small key manipulation (remove the panel id).

4.5 Smart Contracts

The Ethereum smart contracts are the pieces of a decentralized application, coded using the Solidity [17].

To start implementing softwares that interacts with Ethereum, the first step must be towards the comprehension of its capabilities and related constraints. It serves to extract benefits and to foresee critical points. Some of the most interesting requirements for the implementation of smart contracts concern the cost of execution. The code must optimize the amount of commands because each one of them is payed. This value is used to compensate the device that is performing the calculation. Moreover, the deployment of a contract or creation/modification of variables that remains in the blockchain have an associated fee because the nodes must be updated. These factors brings the stimulus for reflection about each algorithmic decision.

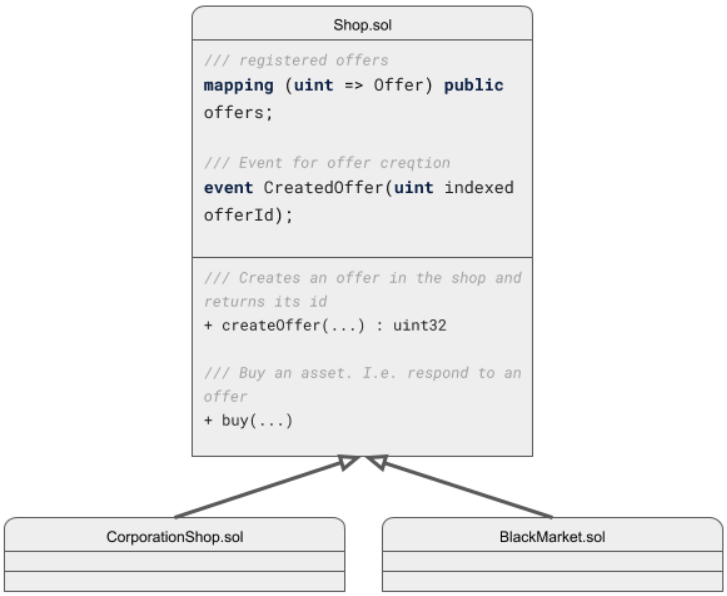
So, the early phase of the internship was oriented to collect knowledge from the experts in blockchain. Orientations about the usage of Solidity and related tools were conducted during teaching sections (some outcomes are described in the section "Initiatives for Information Diffusion" 4.2).

Then, the learning period was complemented by the implementation of small functionalities to put into practice the explored subjects. The first feature is the retrieval of assets from the Nexarium Store. It is basically a request for informations performed by a Call function to a contract (see the "Fundamentals" Section 2.2). This introductory task opened the opportunity to advance towards the use of Transactions, applied in the basic functionalities of the Nexarium Store: buying assets and creating offers.

However, the execution of Transactions is not as simple as the Call functions. To correctly compose the arguments for a Transaction, it is necessary to understand the behavior of the contracts. That was a motivation to read their code and clarify its requisites. The basic Store Transactions are illustrated by the Figure 4.21. We can notice that implementation of the Shop's behavior is centralized, but the context-specific interactions are handled by the "subcontracts" (informal term for the contracts that act as subclasses of another contract, inheriting its methods): the Corporation Shop and the Black Market.

In parallel to the Nexarium Store, other initiatives were being developed in order to extend the contracts' functionalities. So, the good progress of the Nexarium Store application, including the fast integration of the blockchain interactions, allowed the participation on some of these projects. This collaboration with experient professionals permitted the improvement of competences related to the development of smart contracts. The performed contribution can be reported as follows:

Figure 4.21: The Shop contract holds the basic functionalities of the Nexarium Store. It is specialized into two different contracts that handle context-specific interactions: the Corporation Shop and the Black Market.



- Pricing system : B2Expand wants to define a new dynamics to the price of an asset in the Corporation Shop. So, the price associated to an offer decreases along the time.

  In this contract, we have implemented a mechanism that takes into account the date of publishing of an offer to decide its price.

- Fees in Nexium : the payment of transaction fees by using Nexium is splitted into several entities. The client (the Nexarium Store, for instance) contacts an intermediary entity. This "broker" receives the Nexium sent by the client and interacts with the blockchain to perform the transaction (by paying it in Ether).

  The collaboration in this project is summarized to the validation of the concept, by implementing and testing the client-side in the Nexarium Store.

4.5.1 Interface for Interaction

As Unity3D is the main development environment adopted in this internship, the coding starts with the integration of the Nethereum (a .Net library for Ethereum) [10]. It brings

the basic methods for the interaction with the blockchain. That reflects its application in the several points of the code.

However, the standard code for executing Nethereum methods in Unity3D is a bit large, since it handles the creation of the request data and the return of an asynchronous call. So, the frequent utilization of these functions lead to the emergence of the need of an interface adapted to Nexarium needs. This layer provides an extra level of abstraction, facilitating the application of Nethereum in the Nexarium softwares. By "hiding" the original code, it also reduces the length of the code.

The solution is simplified in the functions declarations below. It is based on a subclass of Nethereum's Contract, hereby called NxContract, that offers simple-to-use methods to interact with the blockchain via Nethereum.

- NxContract's method interface for a Call function to the blockchain:

```
/// Calls the contract's function identified by "_methodName" in the ABI.
/// "_methodName" : Name of the method (based on the contract's ABI)
    that must be executed.
/// "decodeResult" : Function that decodes the result from the call.
/// "successCallback" : Function to be executed if a successful result is obtained.
/// "errorCallback" : Function to be executed if an error is obtained.
/// "_extraData" : values to be transferred to the callbacks.
/// "_methodArgs" : Input for the call function.
/// <typeparam name="T">Type of the result of the call function</typeparam>

protected IEnumerator Call<T>(
    string _methodName,
    Call_Result_Decode<T> decodeResult,
    Call_Result_Success_Callback<T> successCallback,
    Call_Result_Error_Callback errorCallback,
    object[] _extraData,
    params object[] _methodArgs
)
```

- NxContract's method interface for a Transaction to the blockchain:

```
/// Triggers the contract's transaction identified by "_methodName" in the ABI.
/// ‘‘_methodName" : Name of the method (based on the contract's ABI)
```

```
    that must be triggered.
/// "_from" : Address of the entity which is executing the transaction.
/// "_gas" : Maximum amount of gas that can be spent by this transaction.
/// "_gasPrice" : Price to be paid for each gas unit spent.
/// "_valueAmount" : The amount of ether to send.
/// "_transactionInfo" : Data that is related to the transaction.
/// "_methodArgs" : Input for the transaction function.

public void Transaction(
    string _methodName,
    Address _from,
    BigInteger _gas,
    BigInteger _gasPrice,
    BigInteger _valueAmount,
    TransactionInfo _transactionInfo,
    params object[] _methodArgs
)
```

- NxContract's method interface for watching a Event in the blockchain.

```
/// Watches the event. Once the ‘‘_eventName’’ is triggered,
    the ‘‘listener’’ will be called.
/// "_eventName" : Event to be watched (based on the contract's ABI).
/// "listener" : Listener for the event.
/// <typeparam name="T">Class that holds the event data.</typeparam>

public void RegisterEvent<T>(
    string _eventName,
    NxEvent<T> listener
) where T : new()
```

4.6 Nexarium Plugin

The main proposal of Nexarium is being a game ecosystem. So many games interact among them and it is represented by sharing information (e.g. assets).

It is a fact that the Nexarium Store is a central component of this ecosystem because it is a hub for assets transactions. However, all this information cannot be monopolized by one entity. The other constituents of the ecosystem (the games) have to be in touch with this content. Then they can interpret the blockchain state managed by the Store and reproduce the effects of this in their contexts.

Finally, the Nexarium plugin has the mission of being the link among the components of the Nexarium Ecosystem. It is a module that can be integrated into each game (or into the Store) and provides a toolkit for interaction with the ecosystem. This means also making available the tools to visualize the blockchain state. Then the games do not need to implement their own mechanism to do so. This represents the standardization of the obtention of blockchain data.

The Nexarium plugin uses a centralized server to store data that is not handled by the blockchain. An account (surprisingly called Nexarium Account) is created to handle this. This account holds, for example, the user's Ethereum public address, which is set by the Store when the wallet is created and is useful for recovering user's assets from blockchain. To access this information, the user must inform its specific credentials for the Nexarium Account: its name (or email address) and Nexarium password (A discussion about this account is conducted in Section 4.3).

As mentioned, this account is an opportunity to preserve game-specific information (the ones that cannot be included in the blockchain). This kind of data is kept away from the blockchain because, among other factors, 1) Sometimes they can consist of frequently updatable data, that would cost too much to maintain in the blockchain due transaction fees; 2) Some games need a quick interaction with data, that does not match with blockchain's time to accept a transaction; 3) They do not need the strong "true-ownership" label, that is granted by immutability of the blockchain.

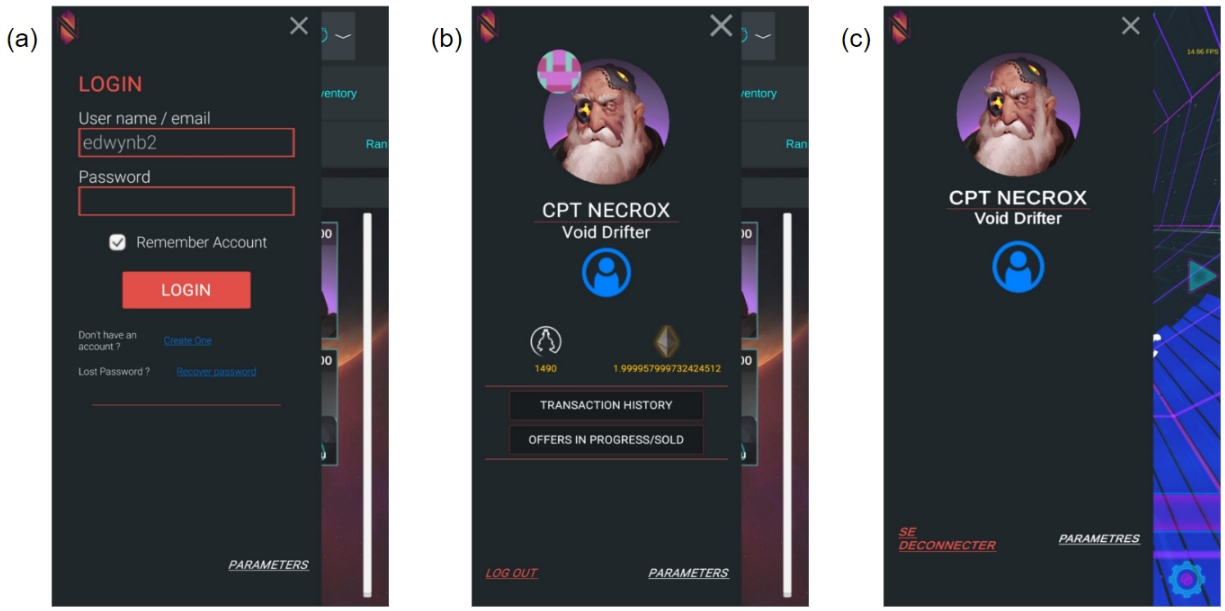
As a toolkit, the Nexarium plugin brings different functionalities according to the context where they will be applied. These methods are presented in Table 4.1. The main point is that, as the Store has the responsibility of perform blockchain transactions, it must hold the user's wallet. So, it makes necessary the addition of several extra features.

In order to ensure the modularity of the Nexarium plugin solution, its code is maintained as a separated repository and it is included as a git submodule [10] in other projects. This strategy allows that updates to the submodule can be easily applied to its dependents. In addition, each project can keep track of the submodule's commit that is currently in use.

Functionality	Description	Context
Signup	The user creates its Nexarium account. It informs its nickname, email address and password. The email is then verified by a validation message sent to the informed address.	Games / Store
Sign-in	The user informs its nickname/email and password to get access to its Nexarium account.	Games / Store
Recover Nexarium Password	A message is sent to their email address, containing a code that is used to recover the Nexarium account's password.	Games / Store
Profile	A screen exposing several informations related to the user.	Games / Store
Change Nexarium Info	The user is allowed to modify some of their information, such as the nickname, email address etc.	Games / Store
Nexium Balance	The interface shows the user Nexium balance.	Games / Store
Retrieve Inventory	The plugin provides an interface to retrieve user's assets from the blockchain.	Games / Store
Read Legal Policies	Before confirming the login, the user must read updates to the legal policies. This procedure is performed by default during the signup.	Games / Store
Create Wallet	The interface orients the user through the creation of a wallet. The user's public Ethereum address is associated to their Nexarium account.	Store
Recover Wallet	The interface orients the user through the recover of a wallet. The user's public Ethereum address is associated to their Nexarium account.	Store
Update KYC Info	The user can update their KYC information.	Store
Perform Transaction	The user can perform blockchain transactions, such as buying or creating an offer.	Store
Transaction History	A list containing the history of triggered transactions is exhibited.	Store

Table 4.1: Functionalities of the Nexarium plugin. The context lists where these functions can be used.

Figure 4.22: Nexarium plugin screens. (a) The login screen is common to all contexts. However, the screen for logged user's information is an example of distinction of functionalities, between the Nexarium Store (b) and other games (c).



4.6.1 State Management

The previous description clarifies the idea that the context were the plugin runs influences its content. But, in order to produce the correct user experience, the evaluation of a series of conditions must orient the screens displayed. I.e. the implementation must consider the status of the Nexarium Account. For example, users are only permitted to create wallets if they have signed-in.

This setting conducts to the adoption of a state machine [44] to manage the software. In the Nexarium plugin, the implementation of this strategy brings some advantages, such as:

Control centralization: if several entities hold the control over the information, it is difficult to determine the correct data in case of disagreement, leading to information inconsistency.

So this mechanism was built as a centralized point (in our case, a class named "ProfileManager") that has all decision-taking responsibility. It is capable to determine the

current state and exhibit the respective screen. This class also provides an interface for receiving external events that aim to trigger a transition.

Ensure execution conditions: the requirements of the domain forces some conditions to the execution of actions. For instance, the user has to read the legal policies updates before signing-in; and if we step back, it is mandatory to have a Nexarium account to do so.

The state machine provides a solution to this issue based on a simple reasoning: 1) the actions are performed inside a stats; 2) A state is only reached if all the conditions are satisfied. So, we establish a tool for controlling the conditions of the use cases.

Easy to understand: the state machine is accompanied by a standard for its representation via a diagram. So, the adoption of this tool is a good practice in order to explain its behavior.

This diagram can have multiple levels of abstraction. Then, not only the developer team can have an easy understanding of the complete operation.

The Figure 4.23 presents a description oriented to developers. It includes states, decision taking and events. We can focus on the existence of interest zones, where some functionalities are grouped. It is the case for the "Token validation", "Wallet" and "Policies".

Easy to implement: the construction of a state machine is based the manipulation of the current state based on the transactions, that can demand a state modification. When a state is reached, its behavior is executed. So, events or guard conditions trigger new states (or a self-loop).

In the case of Nexarium plugin, the states are categorized into "Logical" and "Panel" ("Profile" are an special instance of a "Panel"). "Panel" is a term to indicate that the state is associated to a screen, and its behavior is executed by this object (E.g. the state "Login" is linked to the panel where the user perform their login). The "Logical" ones have their operation run internally by the Profile Manager.

Easy to maintain : as the Nexarium plugin will be integrated in many applications, frequent updates will be supposedly required. So, it is important that this solution presents an easy maintainability.

This characteristic is achieved as result from the association of the previous topics ("Easy to understand" and "Easy to implement"). For including or removing a behavior, the developers must think about adding/deleting a state to the machine. So they find a clear correspondence between the visual representation and the code.

We can notice that the "Wallet" zone (Figure 4.23) is reachable only by Nexarium Store events. It means that this part of the mechanism can be completely disconnected when necessary. It represents the modularity of the state machine.

The state machine is also a powerful tool for debugging because its execution can be tracked. The path of condition verifications can be derived from the states and transitions used until reaching the bug point. Actually the simple observation of the current state gives a lot of evidences about the error.

A very important function of Nexarium plugin's state machine is the validation of the authentication token. This token is provided by the server and represents that the user has the correct credentials for signing-in. It is hashed, containing information about the user's session. The client application stores this data once the user tries to perform login in the server (by informing their password). This strategy is used to avoid the frequent communication using the user's password.

The token validation involves a lot of steps (Figure 4.23, zone "Token Validation") and plays an primordial function by defining whether a user is logged-in or not. So, a Session Manager was implemented with the objective of dealing with the token, by offering an abstraction layer over the token for the Profile Manager. It stores, decodes and extracts token's information.

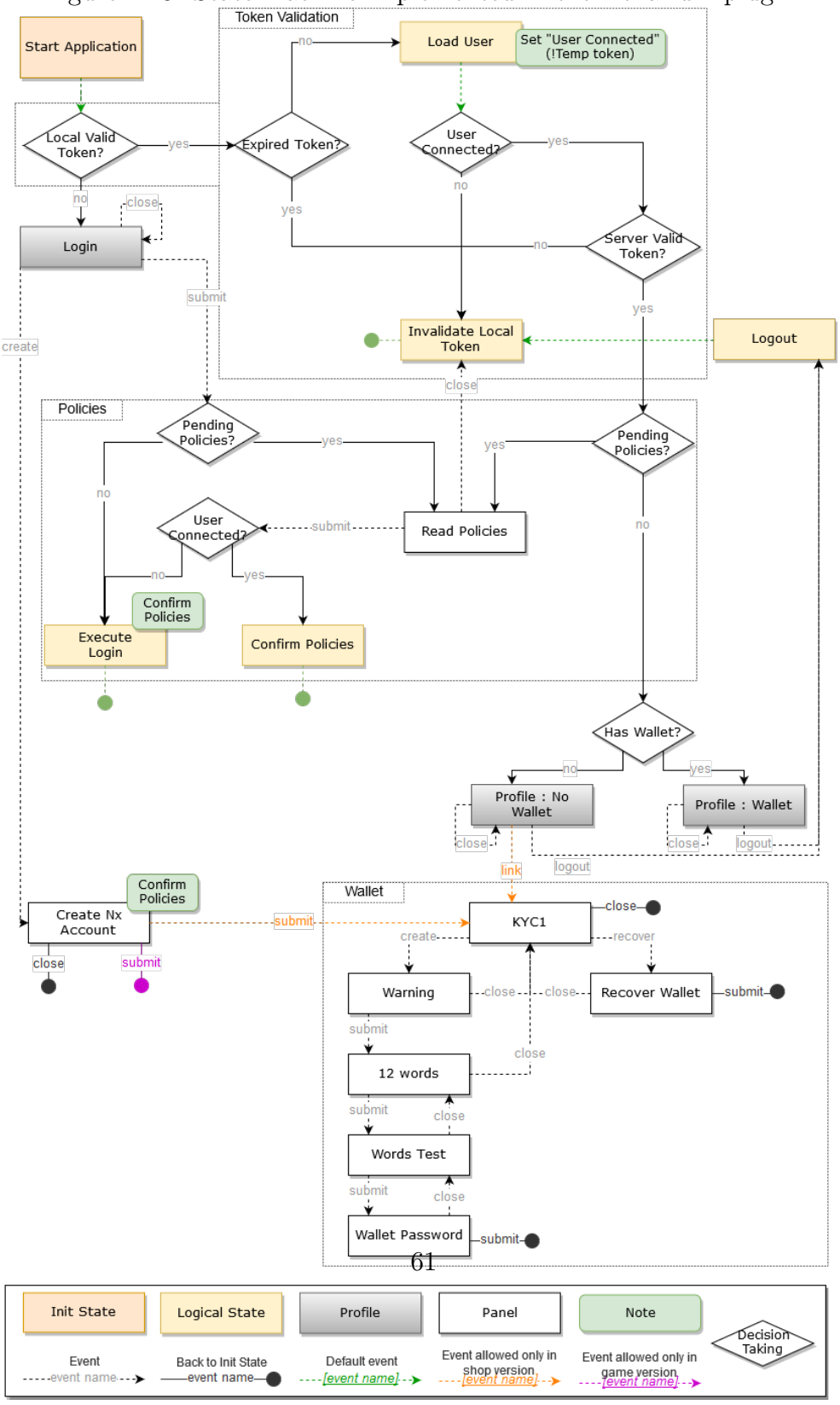
4.6.2 Error Handling

The Nexarium plugin establish an intense communication with the server. One of the results of this interaction is the reporting of errors. So, from the client-side, the software must be ready to interpret them and friendly expose these messages to the user.

The solution adopted in this case implements a class that "translate" the server errors (json data) into Nexarium exceptions. An algorithm for selecting the correct multi-language key for the exception helps to generate a feedback for the user. The Nexarium plugin offers a standard message panel for showing them.

However, there is another part that is equally important in this process: each client application can have its own error exhibition system. I.e. maybe the host application needs these exception being exposed in a specific local panel layout, adapted to its artistic guidelines, for example. So, we take advantage of the Singleton design pattern [32] to solve this problem. This mechanism restricts the existence of a single instance of an object. So, by implementing the abstract class "MessagePanel" (Appendix D), a local error exposition system can handle the Nexarium exceptions.

Figure 4.23: State machine implemented in the Nexarium plugin.



Chapter 5

Conclusions

The objective of an internship is to develop skills based on finding solutions for real problems. It is the opportunity to contextualize the knowledge acquired during the academic courses.

Performing an internship in an company counts with the positive side of the integration into the working process of experient professionals, which improves the learning.

In B2Expand, these concepts were put into practice by the engagement in a key-project. The Nexarium Store plays a central role in the company's objectives, so it stimulates the interest of the whole team in participating. This represent the active collaboration of different production domains (Game Design and Graphics design, for the UX; Business Management for requirements gathering etc).

The internship presented a large variety of sources of learning. Besides the interaction with people, mentioned before, the effort to deal with an emerging technology exercises the adaptability to new tools and the capacity of proposing solutions based on constraints (such as those mentioned on Section 4.3).

The knowledge acquisition was also possible due the intervention of colleagues that produced courses, orienting the subjects to the our work context. So, during this period, we had classes on game design, Unity3D and blockchain. The training to deal with the blockchain included lessons on Solidity (a domain-specific language to build smart contracts). The opportunity to benefit from the coaching of experts in the development for Ethereum is a factor that must be valorized due its rarity.

From a private perspective, the internship gave the chance to get deeper into subjects of personal interest. The conception/implementation of an architecture that was capable

to deal with several requirements was one of the main matters. As well as working in projects that had explored a large range of subjects (E.g. user experience conception, smart contracts etc). Furthermore, the liberty for taking decisions [and mainly the feeling that suggestions are being taken into account] encouraged the abilities of reflection. And all this gains even more motivation due to the conscience of construction of a game ecosystem, that will entertain an immeasurable amount of people.

The main barriers faced during this period concern the challenges imposed by the blockchain. It is an environment that brings advantages, but also obstacles. Above all, the fact that this tool exiges changes on users' mindset about interaction, for example. It requires hours of discussion on the possible solutions. In addition, related to the work environment, the execution of multiple parallel projects creates dispute for resources, such as graphic artists or game designers. A good schedule must be conducted to plan the distribution of means.

Respecting the objectives established, the Nexarium Store project achieves a useful version at the end of the internship. It includes the main functionalities that are necessary for the interaction with the Nexarium Ecosystem. Some iterations still must be performed, in order to integrate in-progress B2Expand initiatives (such as the KYC, or the buying of Nexium in euros/dollars). But, as described by this report, the implementation is ready for extensions, by providing a modular and customizable structure.

A pleasant outcome of the internship, being considered as an acknowledgement of the executed work, is the invitation to present the solution during the BGS (Blockchain Game Summit). The event that takes place in Lyon (September, 25-26) and introduces the tendances for the domain and exposes the existing solutions. As an important showcase for B2Expand, the Nexarium Store will be presented, focusing on the solutions it can provide (such as those from Section 4.3.2), as an important component of the Nexarium Ecosystem.

Finally, an expression of gratitude to the people that contributed to the satisfactory progress obtained. Gamagora formation (Lyon 2), by means its professors and staff, formed competences that grounded the work. Then, the good reception, environment and collaboration provided by B2Expand, that derived good results for both parts.

References

[1] Adobe XD. https://www.adobe.com/pt/products/xd.html. [Online; accessed 08-September-2018].

[2] B2Expand. http://b2expand.com/. [Online; accessed 22-August-2018].

[3] Bitcoin. https://bitcoin.org. [Online; accessed 08-September-2018].

[4] Blockchain Demo - a visual demo of blockchain technology. https://blockchaindemo.io. [Online; accessed 25-August-2018].

[5] CryptoKitties. https://www.cryptokitties.co. [Online; accessed 01-September-2018].

[6] DNN - Decentralized News Network. https://dnn.media/. [Online; accessed 08-September-2018].

[7] Doxygen. http://www.doxygen.nl/. [Online; accessed 08-September-2018].

[8] Etheremon - Decentralized World of Ethre monsters. https://www.etheremon.com/. [Online; accessed 01-September-2018].

[9] Ethereum Project. https://www.ethereum.org/. [Online; accessed 08-September-2018].

[10] Git Submodules. https://git-scm.com/book/en/v2/Git-Tools-Submodules. [Online; accessed 08-September-2018].

[11] Gods Unchained. https://godsunchained.com. [Online; accessed 01-September-2018].

[12] I2 Localization. http://inter-illusion.com/tools/i2-localization/. [Online; accessed 08-September-2018].

[13] Jira. https://www.atlassian.com/software/jira. [Online; accessed 08-September-2018].

[14] LivePeer. https://livepeer.org/. [Online; accessed 01-September-2018].

[15] Nethereum. https://nethereum.com/. [Online; accessed 08-September-2018].

[16] Peepeth. https://peepeth.com/. [Online; accessed 01-September-2018].

[17] Solidity. https://solidity.readthedocs.io/en/v0.4.24/. [Online; accessed 08-September-2018].

[18] Truffle Suite. https://truffleframework.com/. [Online; accessed 08-September-2018].

[19] Unity UI. https://docs.unity3d.com/Manual/UISystem.html. [Online; accessed 08-September-2018].

[20] Mnemonic documentation. https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki, [2013] 2018. [Online; accessed 01-September-2018].

[21] How Is Ethereum Different From Bitcoin? https://www.forbes.com/sites/quora/2017/09/14/how-is-ethereum-different-from-bitcoin/#19a24994502b, September 2017. [Online; accessed 08-September-2018].

[22] Altexsoft. Agile Project Management: Best Practices and Methodologies. https://www.altexsoft.com/wp-content/uploads/2016/04/Agile-Project-Management.-Best-Practices-and-Methodologies-AltexSoft-Whitepaper.pdf. [Online; accessed 08-September-2018].

[23] Craig A. Anderson and Brad J. Bushman. Effects of Violent Video Games on Aggressive Behavior, Aggressive Cognition, Aggressive Affect, Physiological Arousal, and Prosocial Behavior: A Meta-Analytic Review of the Scientific Literature. *Psychological Science*, 12(5):353–359, September 2001.

[24] James Batchelor. GamesIndustry.biz presents... The Year In Numbers 2017. https://www.gamesindustry.biz/articles/2017-12-20-gamesindustry-biz-presents-the-year-in-numbers-2017, December 2017. [Online; accessed 08-September-2018].

[25] Blockchainhub. Cryptographic Tokens. https://blockchainhub.net/tokens/. [Online; accessed 03-September-2018].

[26] BlockGeeks. What is Blockchain Technology? A Step-by-Step Guide For Beginners. https://blockgeeks.com/guides/what-is-blockchain-technology. [Online; accessed 28-August-2018].

[27] BlockGeeks. What Is Hashing? Under The Hood Of Blockchain. https://blockgeeks.com/guides/what-is-hashing/. [Online; accessed 03-September-2018].

[28] Michele D'Aliessi. How Does the Blockchain Work? https://medium.com/@micheledaliessi/how-does-the-blockchain-work-98c8cd01d2ae, June 2016. [Online; accessed 28-August-2018].

[29] Cambridge Dictionary. Token meaning in Cambridge English Dictionary. https://dictionary.cambridge.org/dictionary/english/token. [Online; accessed 05-September-2018].

[30] Matthieu Eginard. Rapport de Stage. September 2018.

[31] Gregory Gailliot. Rapport de Stage. September 2018.

[32] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[33] Sean Han. How does blockchain really work? I built an app to show you. https://medium.freecodecamp.org/how-does-blockchain-really-work-i-built-an-app-to-show-you-6b70cd4caf7d, September 2017. [Online; accessed 25-August-2018].

[34] David Hessekiel. The Future Of Social Impact Is...Blockchain. https://www.forbes.com/sites/davidhessekiel/2018/04/03/the-future-of-social-impact-is-blockchain/#53de4f0ec3fd, April 2018. [Online; accessed 04-September-2018].

[35] Garrick Hileman and Michel Rauchs. 2017 global cryptocurrency benchmarking study. April 2017. [Online; accessed 08-September-2018].

[36] IBM. What is blockchain? https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=XI912346USEN&, 2018. [Online; accessed 28-August-2018].

[37] Investopedia. Initial Coin Offering (ICO). https://www.investopedia.com/terms/i/initial-coin-offering-ico.asp. [Online; accessed 27-August-2018].

[38] Chris Jones, Ruslan Ramanau, Simon Cross, and Graham Healing. Net generation or Digital Natives: Is there a distinct new generation entering university? *Computers & Education*, 54(3):722–732, April 2010.

[39] Raph Koster. The ethics of entertainement. *Theory of Fun for Game Design*, pages 164–175, 2013.

[40] Sherman Lee. Explaining Side Chains, The Next Breakthrough In Blockchain. https://www.forbes.com/sites/shermanlee/2018/02/07/explaining-side-chains-the-next-breakthrough-in-blockchain/#17cf06e252eb, February 2018. [Online; accessed 25-August-2018].

[41] Lisk. Nodes. https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/nodes. [Online; accessed 01-September-2018].

[42] Antonio Madeira. What are State Channels. https://www.cryptocompare.com/coins/guides/what-are-state-channels/, September 2018. [Online; accessed 08-September-2018].

[43] Bernard Marr. The 5 Big Problems With Blockchain Everyone Should Be Aware Of. https://www.forbes.com/sites/bernardmarr/2018/02/19/the-5-big-problems-with-blockchain-everyone-should-be-aware-of/#95fb561670c3/, February 2018. [Online; accessed 28-August-2018].

[44] Karleigh Moore. Finite State Machines. https://brilliant.org/wiki/finite-state-machines/. [Online; accessed 08-September-2018].

[45] Maryanne Murray. Blockchain explained. http://graphics.reuters.com/TECHNOLOGY-BLOCKCHAIN/010070P11GN/index.html, June 2018. [Online; accessed 28-August-2018].

[46] Marc Prensky. The digital game-based learning revolution. *Digital game-based learning*, 2001.

[47] Marc Prensky. Fun, play and games: What makes games engaging. *Digital game-based learning*, pages 11–16, 2001.

[48] G.W. Records. *Guinness World Records 2018 Gamer's Edition: The Ultimate Guide to Gaming Records*. Guinness World Records, 2017.

[49] Ken Schwaber and Jeff Sutherland. The Scrum Guide. November 2017. [Online; accessed 08-September-2018].

[50] Ian Sommerville. Architectural design. *Software Engineering*, 2010.

[51] SSL2Buy. Symmetric vs. Asymmetric Encryption. https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences. [Online; accessed 08-September-2018].

APPENDICES

Appendix A

IModifierHandler

```
/// A modifier generates a new list based on some modification criteria.
/// <param name="_values">List of elements that will be treated.</param>
/// <returns>The modified list.</returns>
public delegate List<T> ListModifier<T>( List<T> _values );

/// Holds modifiers and applies them.
public interface IModifierHandler<T> {

    /// Adds the modifier to the list of current modifiers.
    /// <param name="_modifier">Modifier to be applied.</param>
    void addModifier( ListModifier<T> _modifier );

    /// Removes the modifier from the list of current modifiers.
    /// <param name="_modifier">Modifier to be removed.</param>
    void removeModifier( ListModifier<T> _modifier );

    /// Checks if the modifier is present in the list of current modifiers.
    /// <param name="_modifier">Modifier to be verified.</param>
    bool containsModifier( ListModifier<T> _modifier );

    /// Clears the list of current modifiers.
    void clearModifiers ();

    /// Forces the application the modifiers
```

```
    void applyModifiers ();

    /// Gets the original items that are affected by the Modifier.
    /// <value>The original items that are affected by the Modifier.</value>
    List<T> Items{get;}
}
```

Appendix B

IGetData

```
/// Interface that defines the interactions with an entity that obtains data.
/// <typeparam name="T">Data type.</typeparam>
public interface IGetData<T> where T : new(){

    /// Gets the listener handler. Then the child can register/remove its listeners
    /// <returns>The listener handler.</returns>
    ListenerHandler<T> ListenerHandler();

    /// Gets the current value
    /// <returns>The current value.</returns>
    T CurrentValue();
}

/// Interface that defines the interactions with an entity that obtains data.
/// Includes a type for identification of searched element.
/// <typeparam name="T">Data type.</typeparam>
/// <typeparam name="ID">Id type.</typeparam>
public interface IGetData<T,ID> where T : new(){

    /// Gets the listener handler. Then the child can register/remove its listeners
    /// <value>The listener handler.</value>
    /// <param name="_id">Value's unique identification.</param>
    ListenerHandler<T> ListenerHandler( ID _id );
```

```
    /// Gets the current value
    /// <returns>The current value.</returns>
    /// <param name="_id">Value's unique identification.</param>
    T CurrentValue( ID _id );
}
```

Appendix C

ListenerHandler

```
/// Manages the listeners.
/// <description>The user class registers/removes its listeners and
    the ListenerHandler calls them when necessary.</description>
/// <typeparam name="ListenerValue">The type of some data
    that is transmitted by the listener.</typeparam>
public class ListenerHandler<ListenerValue> {

    /// <summary>Definition of the function to be called
        when the handler has a success result.</summary>
    public delegate void ListenerCall( ListenerValue _value );
    /// <summary>Definition of the function to be called
        when the handler has an error result.</summary>
    public delegate void ListenerError( System.Exception _exception );

    /// <summary>Registered listeners for a success result.</summary>
    List<ListenerCall> Listeners = new List<ListenerCall>();
    /// <summary>Registered listeners for an error result.</summary>
    List<ListenerError> ListenersError = new List<ListenerError>();

    /// Initializes a new instance of the ListenerHandler class.
    /// <description>The lists of registered listeners will be empty.</description>
    public ListenerHandler(){}

    /// Adds the listeners to the handler.
```

```
/// <param name="_listener">Listener to be called
    when the handler has a success result.</param>
/// <param name="_listenerError">Listener to be called
    when the handler has an error result.</param>
public void addListener( ListenerCall _listener, ListenerError _listenerError ){
    if (_listener != null) {
        Listeners.Add (_listener);
    }
    if (_listenerError != null) {
        ListenersError.Add (_listenerError);
    }
}

/// Removes the listener.
/// <param name="_listener">Listener to be removed
    from the list of success listeners.</param>
/// <param name="_listenerError">Listener to be removed
    from the list of error listeners.</param>
public void removeListener(
    ListenerCall _listener,
    ListenerError _listenerError
){
    Listeners.Remove (_listener);
    ListenersError.Remove (_listenerError);
}

/// Calls the listeners registered on success listeners list.
/// <param name="_value">Value to be transmitted
    to the success listeners.</param>
public void callListeners( ListenerValue _value ){
    foreach( ListenerCall listener in Listeners ){
        listener (_value);
    }
}

/// Calls the listeners registered on error listeners list.
/// <param name="_exception">Exception to be transmitted
    to the error listeners.</param>
```

```
    public void callListenersError( System.Exception _exception ){
        foreach( ListenerError listener in ListenersError ){
            listener (_exception);
        }
    }
}
```

Appendix D

Message Panel

```
// Responsible for showing messages
public abstract class MessagePanel : MonoBehaviour {

    #region SINGLETON
    protected static MessagePanel instance;

    // Assume the place of instance and destroy the old one
    protected virtual void Awake(){
        if( instance == null ){
            instance = this;
        }
        if( instance != this ){
            Destroy ( MessagePanel.Instance.gameObject );
            instance = this;
        }
    }

    public static MessagePanel Instance {
        get {
            return instance;
        }
    }
    #endregion
```

```
    /// Request to show an Error Message.
    /// <param name="_error">The Exception to be shown.</param>
    public abstract void AddError( System.Exception _error );

    /// Request to show a specific Message.
    /// <param name="_messageType">Type of the message to be shown.</param>
    public abstract void AddMessage( MessageItem.MessageDataType _messageType );
}
```