SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER SCIENCE

MeltyFi

BLOCKCHAIN AND DISTRIBUTED LEDGER TECHNOLOGIES

Professors:
Claudio Di Ciccio

Students:
Vincenzo Imperati
Benigno Ansanelli
Andrea Princic

Academic Year 2022/2023

Contents

1 Preface

1.1 Brief presentation of the protocol

The **MeltyFi protocol** presented in the report proposes an innovative solution for **peer-to-pool lending and borrowing with NFT collateral**. Thanks to its structure and **independence from off-chain factors** such as the floor price of NFTs, MeltyFi allows borrowers to easily obtain loans **without the risk of involuntary liquidation of the NFT**, allowing them to obtain liquidity without risking the loss of their NFT. Additionally, the MeltyFi protocol allows lenders to use their capital to **provide liquidity for loans through a lottery system**. For this use of capital, lenders are obviously rewarded, and in the event that the loan they funded is not repaid, they have the opportunity to win the NFT used as collateral proportional to the capital they provided for that loan. If the loan is repaid, the lenders are obviously returned the capital they invested. Those who also repay a loan are rewarded with an amount equal to the interest paid to the protocol.

In summary, MeltyFi guarantees for all users an easy access to a decentralized system of lending and borrowing with NTFs collateral, independent of external factors, and encourages users to provide liquidity and repay loans in order to be rewarded for their correct behavior in proportion to their personal capital invested in the protocol. MeltyFi represents an effective solution for **"making the illiquid liquid"** and offers **benefits to both borrowers and lenders**.

1.2 Outline of the report

- **Background**: an introduction to all concept upon which MeltyFi is based is given (section 2).
- **Presentation of the context**: contains the aim of the protocol, use cases and a neat explanation (section 3).
- **Software Architecture**: this section describes the software architecture including the tools used, an overview of the protocol components, and a detailed examination of each component, as well as reflections on design choices (section 4).
- **Implementation**: here the frontend and the logics behind it are presented (section 5).
- **Know issues and limitations**: limitations of the protocol are discussed in this section (section 6).

1.3 Team members and main responsibilities

- **Vincenzo Imperati**: Project manager, back-end developer, [project, web, report, brand] designer.
- **Benigno Ansanelli**: Developing and maintaining the frontend user interface, with a focus on lottery-related functionality. Preparing and delivering reports and presentations on lottery-related topics.
- **Andrea Princic**: Developing and maintaining the frontend user interface, with a focus on profile-related functionality. Designing use cases and sequence diagrams.

2 Background

2.1 Blockchain technology

Blockchain is a technology that enables the creation of a **distributed, immutable and secure ledger of transactions**. It relies on a network of nodes that collaborate to **maintain a coherent and up-to-date record of transactions**, known as the "blockchain". Each transaction is inserted into a "block" (with others transactions) that is added to the existing chain, creating a history of all transactions made on the network.

The security of the blockchain is ensured through the use of public key cryptography to sign transactions and a **decentralized consensus system** to validate transactions and add new blocks to the chain. The consensus system used depends on the specific blockchain implementation, but the most common ones are Proof of Work (PoW) and Proof of Stake (PoS).

One of the main features of the blockchain is its **transparency**, as all transactions are public and visible to all network participants. However, it is possible to use cryptography to maintain user anonymity and ensure the privacy of transactions.

The blockchain has a wide range of applications, from cryptocurrencies to supply chain management, from electronic voting to decentralized finance (DeFi) applications. The flexibility of blockchain technology allows the creation of smart contracts, which enable the automation of complex processes through the execution of secure and tamper-proof code on the network.

In addition, the blockchain offers greater security compared to centralized systems as it does **not rely on a single entity or authority** for its operation and **is not subject to single point of failure**. This makes the blockchain particularly suitable for scenarios where it is important to ensure the security of transactions and the immutability of data.

2.1.1 Ethereum blockchain

Ethereum is a Proof of Stake blockchain that uses the Ether ($ETH) as its native coin [1]. It is the second-largest blockchain per market cap, with more than 180 billion US dollars [2], just below Bitcoin. One of the reasons for its success is the introduction of smart contracts, pieces of code that are decentralized and execute directly on-chain. Smart contracts allow decentralized applications (DApp) and are fundamental to the so-called Web 3.0.

2.1.2 Smart contracts

Smart contracts are computer programs that run on the blockchain and can automate the process of negotiating. Due to their decentralized nature, smart contracts are considered **trustworthy and secure**, as their **code runs uncensorably** and **cannot be changed** once posted on the blockchain. This makes them particularly suitable for handling transactions and forging contractual relationships where transparency and impartiality are important.

Another advantage of smart contracts is their ability to **eliminate intermediaries**, thus reducing costs and time to negotiate. Furthermore, smart contracts can be used to create incentives for correct user behavior, as the conditions of the contract can be automated in order to penalize users who do not comply with their obligations. Furthermore, smart contracts can be used to create decentralized governance systems, where decisions are made transparently and democratically by all network users.

2.1.3 Token ERC standard

Ethereum smart contracts can be used to create different types of tokens, each of which has specific functions and uses. Let's examine the three main types of Ethereum tokens: ERC-20, ERC-721 and ERC-1155, and explain their characteristics and applications.

ERC-20 ERC-20 tokens are the most popular Ethereum token and are the norm for token creation on the platform. They are designed to be **interchangeable** and to adhere to an interoperability standard, which makes them easy to implement in different DApps and to trade on decentralized marketplaces. ERC20 tokens are typically used to represent units of a certain resource, such as loyalty points or gaming tokens.

ERC-721 ERC-721 tokens, by contrast, represent **unique, non-fungible assets**, such as works of art or real estate. These tokens are used to represent goods that have intrinsic value or are rare or unique. We will analyze them better later.

ERC-1155 ERC-1155 tokens represent **a combination of the first two types of tokens**, ERC-20 and ERC-721, and allow you to manage both fungible and non-fungible assets within a single contract. These tokens are particularly suitable for building decentralized games or representing a collection of assets. Furthermore, they are often used as tickets and are extremely functional because they adapt to the needs of both ERC-20s and ERC-721s.

2.2 Application domain

This section describes the basic concepts and phenomena that will allow us to more clearly understand the purpose of the MeltyFi protocol and the problems it solves. Here we will analyze the phenomenon of NFTs, their value and the manipulations of the latter. We will define in detail what a lending and borrowing platform on blockchain is, one of the most famous use cases of decentralized finance (DeFi); and we will analyze platforms of this kind that make use of NFTs to allow you to obtain loans.

2.2.1 NFTs

Non-fungible tokens(NFTs) have become popular as unique and non-interchangeable units of data that signify ownership of associated digital items, such as images, music, or videos. **Token "ownership" is recorded and tracked on a blockchain** [3][4].

The market for NFTs, transferrable and unique digital assets on public blockchains, has received widespread attention and experienced strong growth since early 2021. Prominent examples of NFTs, such as the **artist Beeple** selling a piece of digital art for $69 million [5] or **Twitter CEO Jack Dorsey** auctioning off his first-ever tweet for $2.9 million [6], show that NFTs have received mainstream attention and represent a popular application in FinTech and the cryptocurrency ecosystem.

NFTs are **unique certificates of authenticity on blockchains** that are usually issued by the creators of the underlying assets. These assets can be digital or physical in nature. Fungible goods such as money or trade goods can be exchanged for goods of the same kind. By contrast, non-fungible items cannot be exchanged for a similar good because their value exceeds the actual material value. Examples from the analogue world include items of artistic or historical significance, or rare trading cards, all of which have a long history of trading in auctions and other marketplaces. In the digital world, it has so far been difficult to trade and auction non-fungible goods, as their authenticity was hard to verify. NFTs now pave the way for the digitization and trade of unique values on the internet [7].

The Etherum blockchain has welcomed the phenomenon of NFTs and it is precisely on this blockchain that the most famous and valuable NFTs reside [8].

2.2.2 NFT floor price

Above all due to the non-tangibility and novelty of this type of asset, it is difficult to decipher the true value and therefore define the right price. The floor price is the most used and most established price indicator to determine the value of an NFT. In general, NFT floor prices are an attempt by market participants to gather information about the fair market value of an NFT project at the collection level. This helps focus an NFT buyer's decision-making and analysis by eliminating factors in the collection such as rarity, traits, and more. [9].

The floor price, however, is a **parameter external to the blockchain**, and therefore as such it is **subject to manipulation** for various purposes. This makes it even more difficult to correctly determine the true value of NFTs, contributing more to increasing their volatility'. **Two phenomena** of floor price manipulation are described below. Both of these methods intentionally mislead prospective NFT buyers into believing that the fair market value of an NFT is the new floor price when in reality the price is not a result of natural demand. This is harder to factor in compared to other NFT floor price factors, and requires NFT buyers to do their due diligence on NFT ownership metrics, market sales, project community, and more.

Sweeping the floor Some NFT collections, often those which are low-priced, can be subject to price manipulation in the form of mass buying. Referred to as ”sweeping the floor” in NFT communities, groups or wealthy individuals can make a concentrated effort to raise the floor price. In this scenario, the floor price is defined as the lowest-priced NFT in a collection as priced by marketplaces.

Wash trading Another way to manipulate the price is through wash trading, where an individual trades their own NFTs. Simply put, a group or individual with enough NFTs can manipulate the price by listing their own NFTs on a marketplace for a more expensive price, and then buying them to artificially inflate the price [10].

2.2.3 Lending and borrowing platform on blockchain

Blockchain lending and borrowing platforms are the main driver of decentralized finance (**DeFi**) [11]. These platforms allow users to lend and borrow assets in a **decentralized manner**, without the need for a central intermediary. These platforms typically use smart contracts to facilitate lending and borrowing transactions, and they often use a variety of different assets as collateral.

Before describing the two largest lending and borrowing platforms in Ethereum (Aave and Compound), a few words are necessary to describe the concept of collateralization, loan-to-value, liquidation and liquidity pool [12]. Finally we mention the possible risks in using lending and borrowing platforms on blockchain.

Collateralization Collateralization is a fundamental concept in the financial industry. It simply refers to something you put up as a guarantee when borrowing money. For example, if you take out a bank loan to buy a house, the house will serve as collateral. If you fail to repay your loan, the bank will repossess your home [13].

It’s the same in Defi. If you want to borrow some assets from the liquidity pool, you must provide the pool with some other assets as collateral. If you do not repay your loan, the protocol will not return your collateral to you. The collateral will be used to repay your debt to the liquidity pool.

Loan-to-value Loan to value (LTV) [14] is the ratio of the loan’s value to the value of collateral. In a typical financial market, credit scores determine the risk involved in a loan. The lower the credit score, the higher the risk for lenders. Instead of credit scores, the crypto lending process offers asset-backed loans.

LTV determines the amount of cryptocurrency one would need as collateral before one could get a loan. The lender holds on to this collateral until the loan is fully paid back.

The main benefit of LTV in crypto lending is that it helps minimize the risk on the lender's part. The user also benefits from LTV in that they can access larger loans at lesser interest rates.

Liquidation In traditional finance, liquidation occurs when a company or group must sell some of its assets at a loss to cover a debt. DeFi liquidations are similar in that users take out debt from a protocol and provide crypto assets as collateral to back the debt. Thus, DeFi liquidation is the process by which a smart contract sells crypto assets to cover the debt [15].

Liquidity pool A liquidity pool is a crowdsourced pool of cryptocurrencies or tokens locked in a smart contract that is used to facilitate trades between the assets on a decentralized exchange (DEX). Instead of traditional markets of buyers and sellers, many decentralized finance (DeFi) platforms use automated market makers (AMMs), which allow digital assets to be traded in an automatic and permissionless manner through the use of liquidity pools [16].

Aave Aave is a decentralized non-custodial liquidity protocol where users can participate as depositors or borrowers [17]. Depositors provide liquidity to the market to earn a passive income, while borrowers are able to borrow in an overcollateralized (perpetually) or undercollateralized (one-block liquidity) fashion. As of the time of writing (Jan. 12, 2022), the total value locked (TVL) in Aave's smart contracts stands at \$3.78 billion [18].

Compound Compound (COMP) is an Ethereum-based lending and borrowing protocol that algorithmically sets interest rates based on the activity in its liquidity pools [19]. As of the time of writing (Jan. 12, 2022), the total value locked (TVL) in Compound's smart contracts stands at \$3.59 billion [20]. As a Compound user, you can lend and borrow some of the most popular cryptocurrencies instantly without having to go through a traditional financial intermediary. It's one of the largest and oldest lending and borrowing apps in the crypto world.

Compound is used extensively by DeFi developers, who programmatically integrate it into their DApps and use the protocol for dynamic borrowing and lending. Many yield aggregation protocols and other DeFi apps make use of Compound. The protocol is also widely used by the general crypto public, not just developers. Non-technical users can borrow funds from Compound by supplying a different crypto coin as collateral.

Risks [21]

- **Liquidation risk**: If the pledged collateral is a volatile crypto asset (such as ETH), and the value drops too far, then the ETH may get liquidated. This is a very undesirable result, as it means that the ETH is sold off after a price drop [22].
- **Crypto volatility**: Cryptocurrency is inherently volatile, and using crypto assets to pledge collateral for a loan can lose a user a significant amount of money. First, the funds are locked into the contracts and cannot be accessed until the loan is paid off. Second, the rules for required liquidations mean losing those funds when the value drops.
- **Liquidity risk**: Users that deposit crypto may not be able to withdraw funds if the liquidity drops too far. This means that they would need to wait until more crypto is deposited by other users to be able to withdraw funds.

2.2.4 NFTs as loan collateral

Nowadays NFTs can be used as collateral to secure a loan. The loan can work like any other DeFi loan, with the difference that the collateral is the NFT itself. The NFT, by its nature, is comparable to an illiquid asset, so for loans of this type, the lender must agree on the value of the collateral and decide on a loan-to-value in agreement with the borrower. This dynamic is the main aspect that allows the execution of the loan and any repayment or liquidation. As with DeFi loans, those with NFT collateral are also managed by smart contracts, which hold the NFT for the entire duration of the loan.

Advantages The advantage of NFT collateral loans is that of being able to obtain liquidity from an illiquid asset. In this way it is possible to benefit from the market value of the NFTs held, without the need to sell them. This last aspect is very advantageous if one thinks of the unicity of the NFTs. The nfts, being unique in fact, are not fungible and therefore selling an NFT to then buy back another one does not in any way guarantee that you are buying back exactly the same one that was sold.

Disadvantages The disadvantages of NFT collateral are many. The cause of all the disadvantages (according to our personal interpretation) is linked to the difficulty of transforming illiquid assets into liquid assets without incurring any repercussions or compromises. In fact, the blockchain is a closed system and the DeFi that is built on top of it inherits this property. This means that, more specifically in our case, if in a closed system such as DeFi an asset that is illiquid by nature such as an NFT is made illiquid, the system suffers an imbalance, favoring the borrower who wants to benefit of this action (make liquid your NFT). This imbalance in a closed system must necessarily be rebalanced by showing disadvantages for some agents of the system. The latter may be the lenders or the borrower himself in the event that his own action ultimately penalizes him. This explanation just made is an attempt to define the root of all the causes which then generate all the disadvantages that can be encountered by operating with DeFi loans with NFT collateral.

Some of the main possible disadvantages and risks are summarized below. However, this topic has not yet been analyzed in detail in the literature and there are many possible attacks that can undermine loans with NFT collateral. We have collected some articles to help the reader develop a personal idea on this topic [23][24][25][26].

- **Market Conditions**: Considering the NFT market is extremely volatile, the value of NFTs fluctuates by the second. With that, lenders risk being stuck with an overvalued NFT if the project plummets in value.
- **Risk of Default**: If you can't pay back the loan, you could lose your NFT. Considering many lenders lend less money than a particular NFT might be worth, you could end up losing money in the long run.
- **Improper Valuation**: An improper proper valuation of an asset means that you run the risk of losing money as a lender. Also, borrowers risk losing the ability to borrow more money if their valuation is undervalued.

We close the discussion by recalling that in order for a lending and borrowing platform to work well with NFT collateral, it is necessary to defend oneself against all possible negative scenarios that disadvantage the user of the platform itself.

Existing protocols

- Arcade [27]
- BendDAO [28]
- DropsDAO [29]
- JPEG'd [30]
- LiquidNFTs [31]
- nftfi [32]
- NFTGo [33]
- Pine [34]
- reNFT [35]
- X2Y2 [36]

2.2.5 Peer-to-peer lending and borrowing with NFTs collateral

Definition In peer-to-peer lending and borrowing, the lender and borrower agree on the NFT's value, the length of the term, and the amount of interest. At the end of the expiry date, if the borrower can't repay the loan in time, the NFT is sent to the lender's wallet as collateral for the unpaid amount.

Pro It eliminates the need for a floor price, as the value of the NFT is agreed upon by the two parties involved in the loan. It also eliminates the risk of sudden liquidation.

Con Achieving agreement between the lender and borrower can be difficult, and a single lender must provide the entire loan amount rather than drawing from a pool of funds provided by multiple lenders. This can limit the availability of loan capital and increase the risk for the lender.

2.2.6 Peer-to-pool lending and borrowing with NFTs collateral

Definition In Peer-to-Pool lending and borrowing, multiple lenders provide liquidity into a pool, and the borrower takes the liquidity from that pool. These pools algorithmically set a threshold for the floor price of the NFT. If it falls below the threshold, is automatically liquidated.

Pro Facilitates access to loan capital for borrowers.

Con The market value of NFTs is highly volatile and can fluctuate rapidly, which can result in the unexpected liquidation of an NFT.

3 Presentation of the context

3.1 Aim of MeltyFi protocol

The aim of MeltyFi is to define a new protocol for lending and borrowing with NFT collateral based on peer-to-pool design, solving some of the problems that plague these types of DeFi platforms.

MeltyFi infact is builted in different way. It is a peer-to-pool design-based lending and borrowing protocol in which the loan collateral is NFT and the funds are raised through a lottery ticket foundraising mechanism. There is no floor price dependence, so no liquidation risk. This is because the borrower creates a lottery and puts up an NFT. The lottery tickets, called WonkaBars, are sold to multiple lenders so that the borrower gets liquidity.

If the borrower repays the loan before the set maturity date, the NFT is returned to him or her and the lenders are repaid. If the borrower does not repay, on the maturity date the NFT is sent to the lottery winner.

It is also easy for the borrower to get the loan because of the peer-to-pool design, in which there are multiple lenders and not just one. MeltyFi rewards with ChocoChips ($CHOC) borrowers who repay their loans and all lenders who merge their WonkaBars.

With this mechanism, as we will see, it is possible to obtain a protocol resistant to many problems and which provides benefits to all users who use it.

3.2 The chocolate factory inside MeltyFi

In the MeltyFi protocol, each NFT is comparable to a **chocolate bar**. When the borrower applies for a loan, they decide how many squares of chocolate to divide the bar (their NFT) and what the price is for each piece. In doing so, the chocolate bar (the NFT) is actually breaking down into small squares of chocolate (the **WonkaBars**), smaller than the bar itself. In this way, therefore, the borrower requests the loan and the illiquid and inseparable NFT is starting to lose these two properties (**it is melting**). Once this process is complete, lenders can decide whether and how many WonkaBars to buy. By buying a WonkaBar, the lender is financing the loan requested by the borrower.

In the event that the borrower repays the loan, the lender can melt the purchased WonkaBars to receive back the capital employed, plus the premium for having employed this capital (the **ChocoChips**, the fragments produced by the chocolate bar when dividing into chocolate squares).

In the event that the borrower does not repay the loan, all WonkaBar holders compete to win the NFT collateralized by the loan because the NFT is inseparable and only one WonkaBar holder can win it. Just like in the chocolate factory [37], at this point the WonkaBar holders open their WonkaBars with the hope of finding the only winning ticket (the **golden ticket**) capable of authorizing them to receive the collateral NFT as a prize. At this point, the winner can melt their WonkaBars to receive the NFT as a prize, but also the ChocoChips for using their own capital in financing the loan. Those who are not winners can still melt their WonkaBars to receive only the ChocoChips.

3.3 Use cases

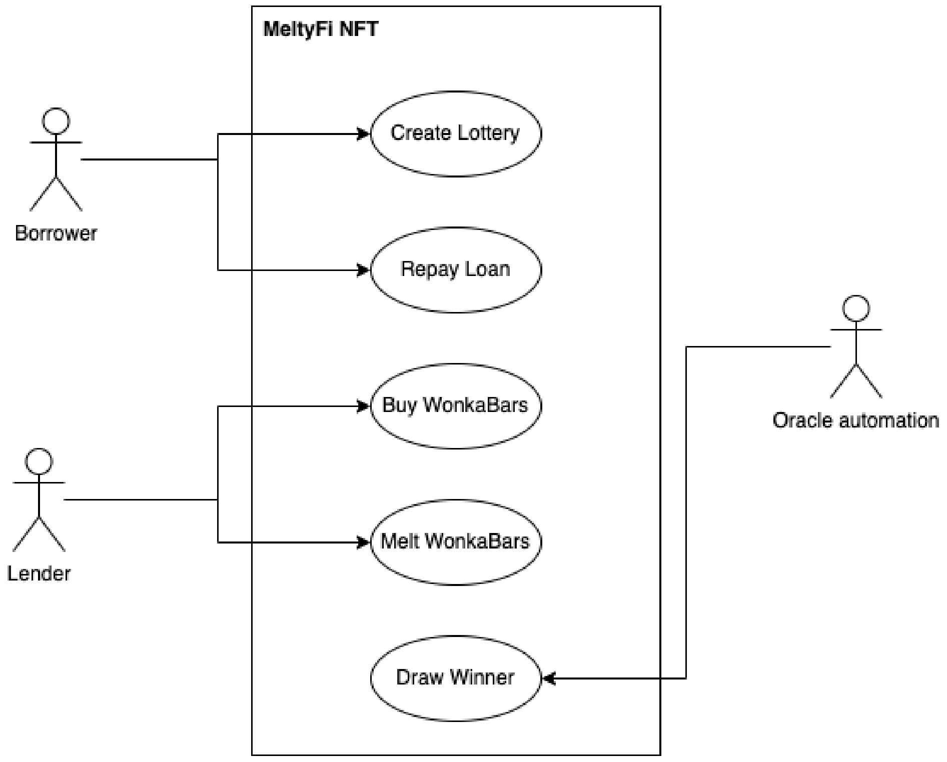


Figure 1: Use cases of the protocol

In Figure 1 we present the use cases, five in total. There are 3 actors:

- The **Lender** that can **Create Lottery** (Figure 2) and **Repay Loan** (Figure 3).
- The **Borrower** that can **Buy WonkaBars** (Figure 4) and **Melt Wonkabars** (Figure 5).
- The **Oracle automation** (Figure 6) that can draw a winner for an expired lottery.

As follows, the only two possible scenarios that can arise within the protocol are described.

3.3.1 Scenario 1: Borrower creates a lottery and repays the loan

After a borrower creates a lottery, lenders can buy WonkaBars and fund the borrower. The borrower immediately receives 95% of the price for each WonkaBar sold, while the remaining 5% is kept by the MeltyFiDAO. After some time, but before the lottery expires, the borrower can repay the loan to cancel the lottery. They are awarded some $CHOC and the NFT is given back to them. At this point, lenders can melt their WonkaBars and be refunded of their investment. They are also awarded with some $CHOC.

3.3.2 Scenario 2: Borrower creates a lottery but doesn't repay the loan

If a borrower doesn't repay the loan before the expire date of their lottery, a winner is drawn among the lenders and the lottery is marked as concluded. Now lenders can melt their WonkaBars and still be awarded with $CHOC, but they won't be refunded of the investment since the lottery took place. The winner lender also receives their prize NFT after melting their first WonkaBar.

3.4 Why using a blockchain

Blockchain provides a decentralized and distributed system for recording and tracking transactions. This means that there is no single point of failure or control, making the system more resilient and secure. Additionally, the use of cryptography and consensus algorithms ensures that transactions are immutable and tamper-proof, providing a high level of security and trust for users of the protocol, very much needed in case of lending and borrowing to people we don't know and don't trust.

Ethereum was the blockchain of choice. The use of NFTs as collateral also aligns well with Ethereum, as it is currently the leading blockchain platform for NFTs. Additionally, Ethereum's large and active developer community would likely provide the necessary support for the development and maintenance of the protocol. Another key feature of Ethereum leveraged by MeltyFi is smart contract functionality. It is seamless to think of MeltyFi as a smart contract, since it has to manage things already on blockchain. Using smart contracts it is possible to automate processes such as loan origination, collateral management, and loan repayment, reducing the need for intermediaries, and possible preferences that they could make, beside increasing efficiency.

In summary, the MeltyFi protocol utilizes blockchain technology to provide a secure, transparent and decentralized platform for lending and borrowing with NFTs as collateral. The decentralized nature of blockchain ensures that the platform is resilient and the smart contract functionality ensures that the process is efficient. Together, these capabilities make blockchain an essential technology for the MeltyFi protocol.

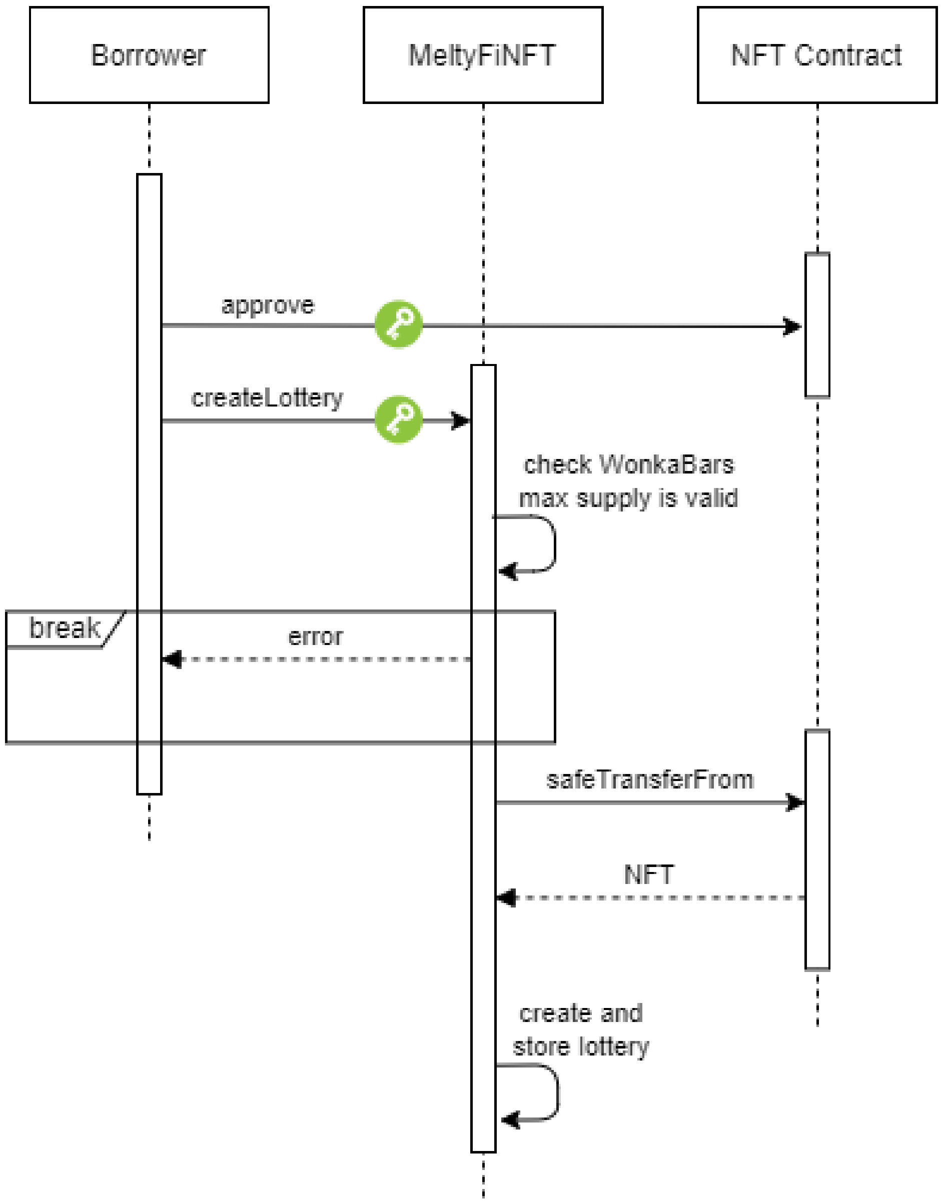


Figure 2: CreateLottery sequence diagram

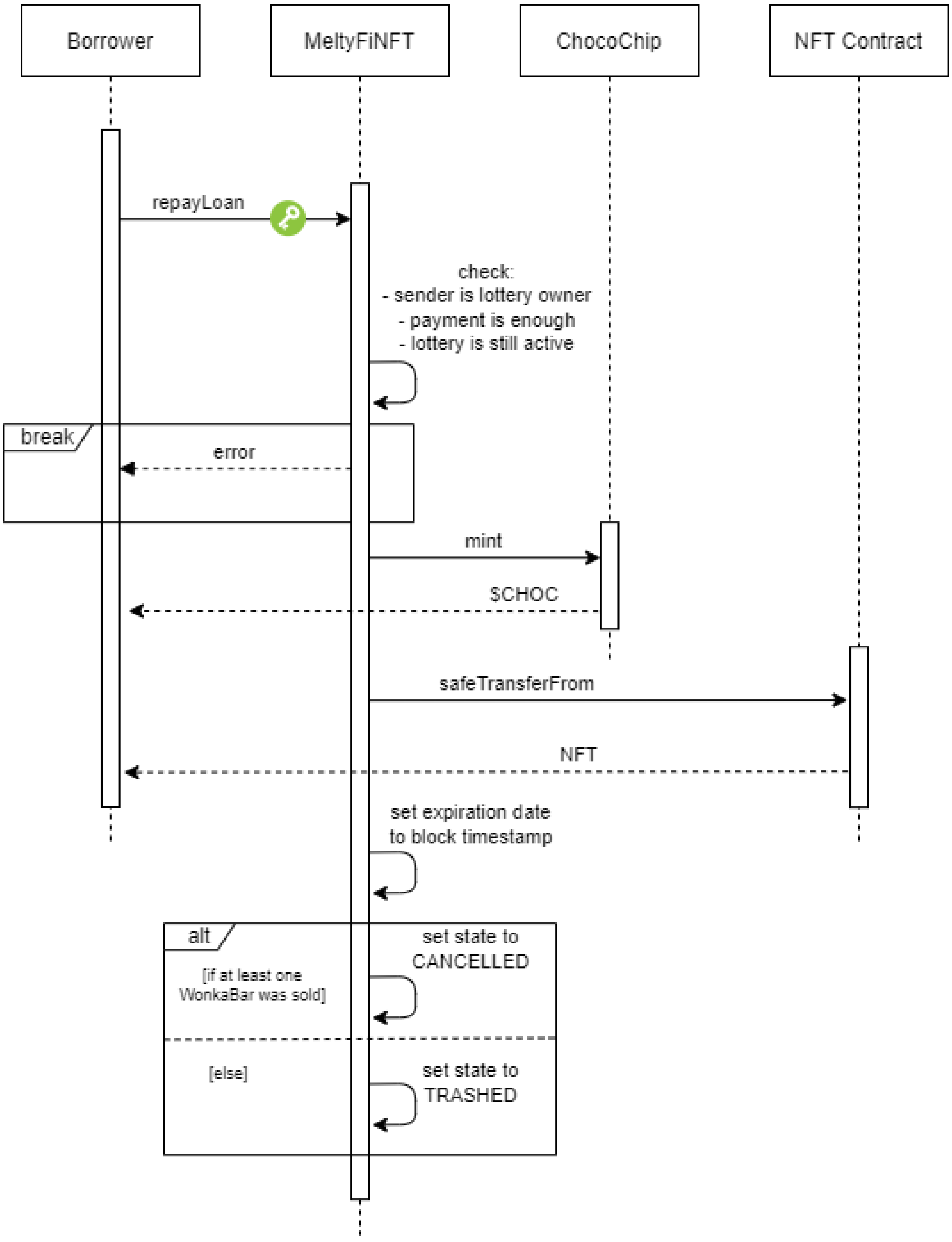


Figure 3: RepayLoan sequence diagram

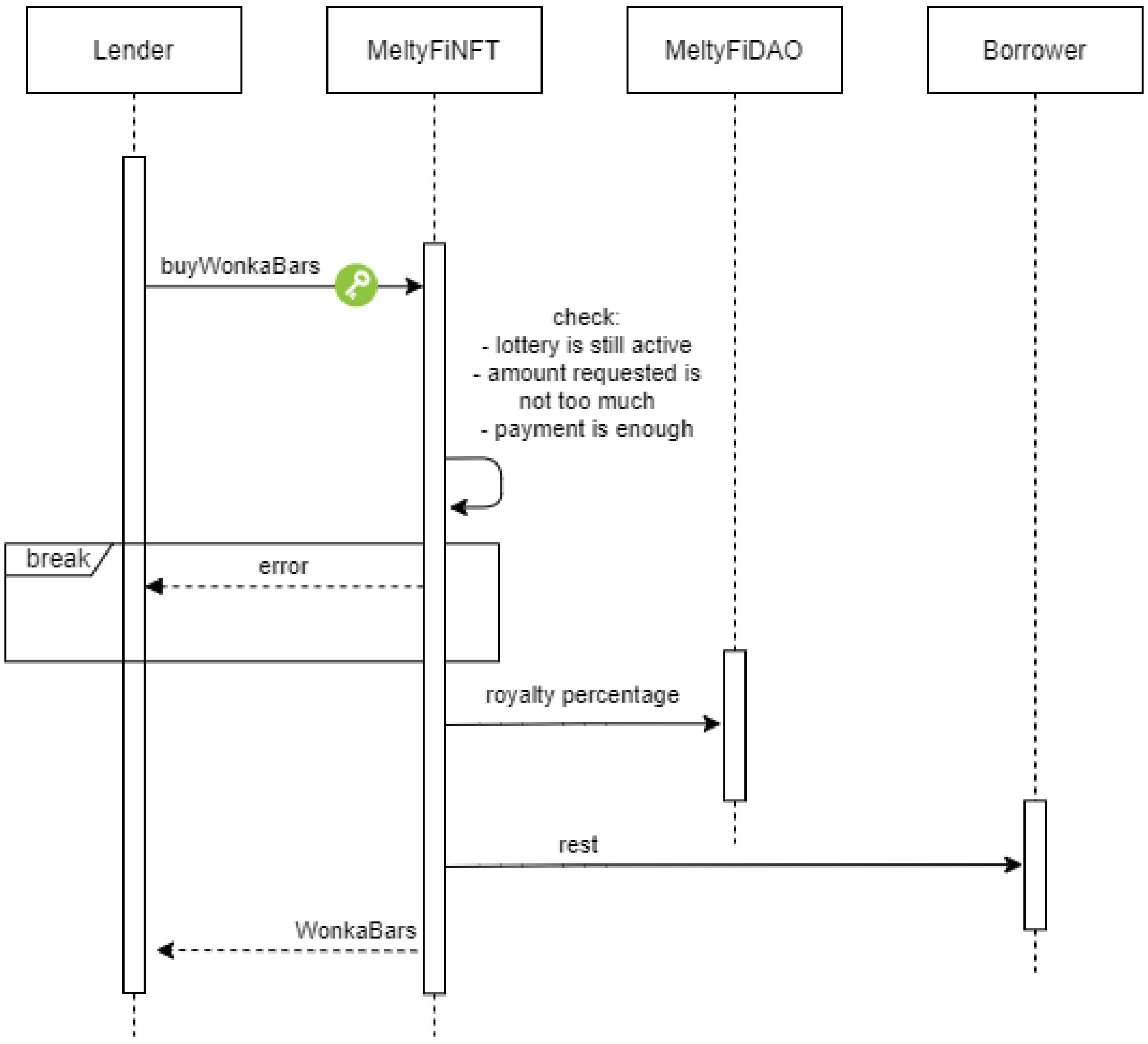


Figure 4: BuyWonkaBars sequence diagram

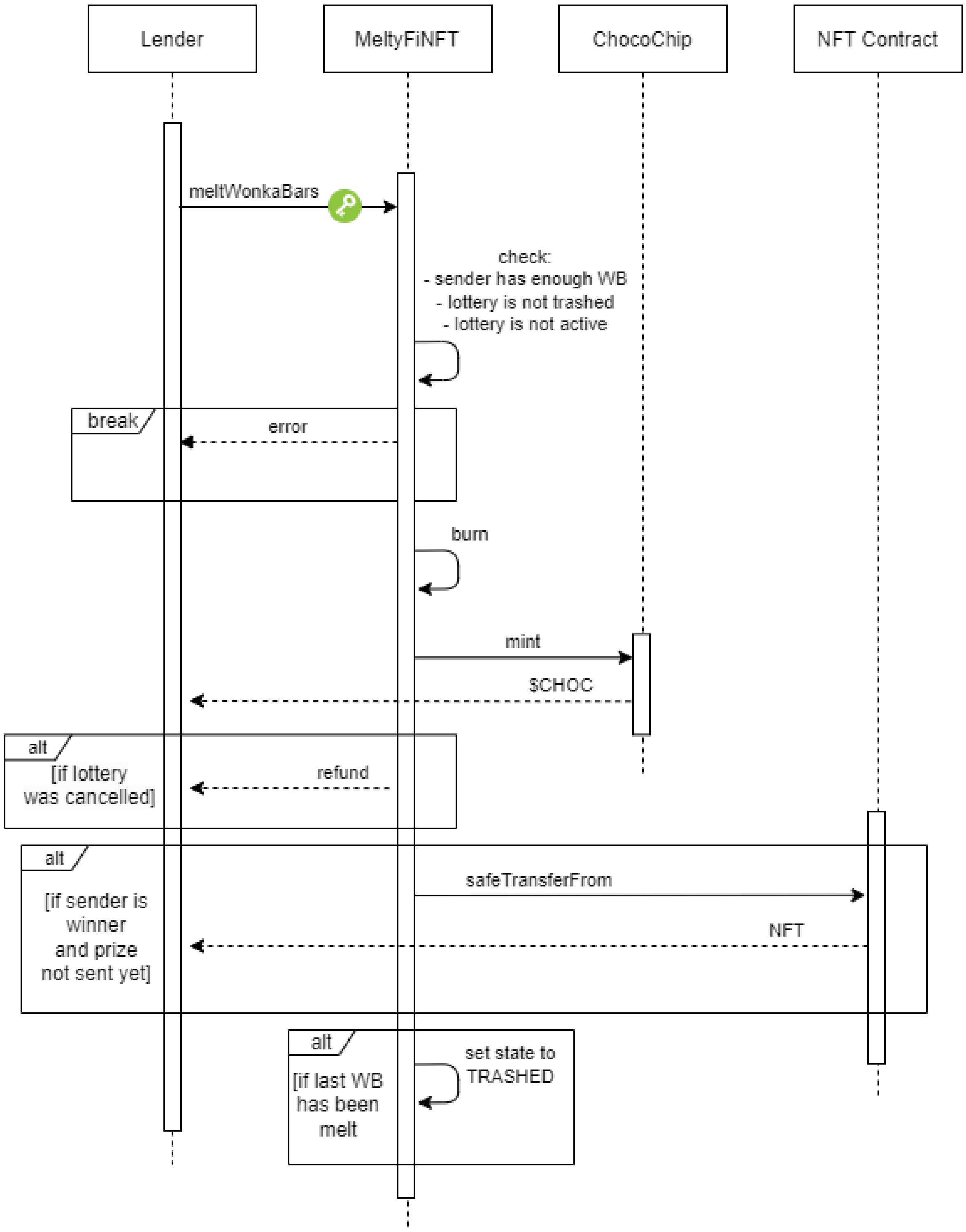


Figure 5: MeltWonkaBars sequence diagram

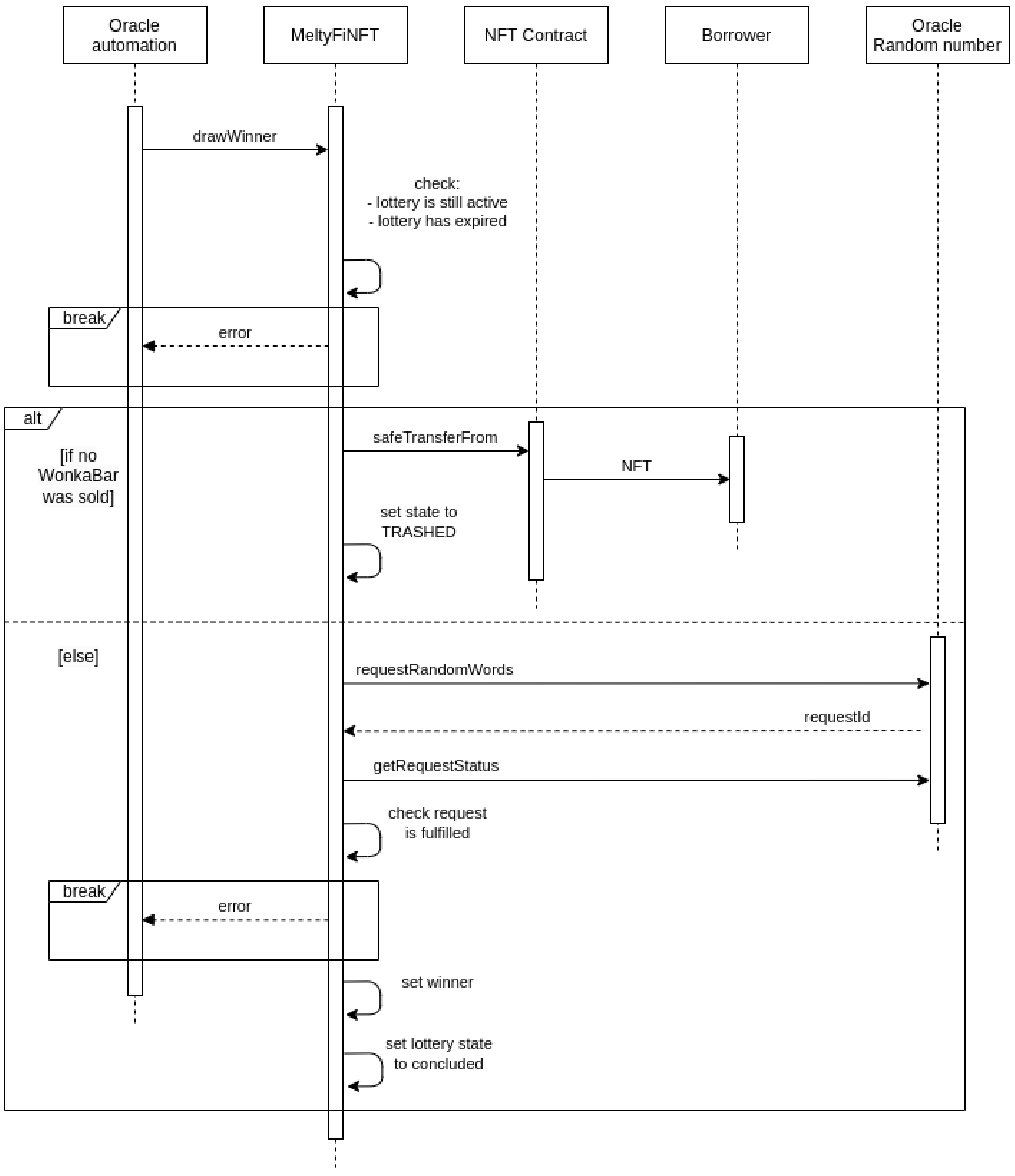


Figure 6: DrawWinner sequence diagram

4 Software architecture

In this section, the software architecture of MeltyFi is shown in detail. All the tools used are mentioned, an overview of all the protocol components is given, and then they are explained individually. Finally, reflections are given regarding the design choices made.

The entire protocol is open-source and is searchable on GitHub [38]. It is also possible to interact with MeltyFiNFT via Goerli Etherscan interface or MeltyFi.NFT DApp. Equally it is possible to interact with MeltyFiDAO via Goerli Etherscan interface or MeltyFi.DAO DApp. MeltyFiNFT encapsulates the core of the MeltyFi protocol while MeltyFiDAO encapsulates the DAO management of the MeltyFi protocol.

4.1 Tools used

For the realization of the MeltyFi protocol we relied on the environment **Node.js** [39] in order to use the following npm packages:

- **@openzeppelin/contracts**: used for draw on all the already audited code that we would use when necessary [40]
- **@chainlink/contracts**: used for the realization of the whole part that makes use of oracles [41]
- **hardhat**: used as development environment [42]
- **@nomicfoundation/hardhat-toolbox**: used to deploy on goerli testnet and verify contract on etherscan [43]
- **hardhat-docgen**: used for automatic generation of code documentation [44]
- **dotenv**: used to load sensible datas in the code such as private keys and API keys [45]

The **Alchemy API** [46] was used to make deployment possible, and the **Etherscan API** [47] was used to make contract verification possible. **Remix** [48] was used as an editor while writing the contracts. The latter were compiled with **solidity compiler 0.8.17**. Overall, all the MeltyFi protocol is capable of running on the same or higher versions of **solidity 0.8.9**. This choice made avoids overflow problems and allows us to benefit from all the audit code that was intended to be used.

Finally, to allow MeltyFi to function properly, which requires interaction with oracles, the entire protocol was deployed on **Goerli testnet** [49]. **Goerli faucet** [50] and **Link faucet** [51] were used to enable this.

4.2 Protocol blueprint

The blueprint of the entire protocol is shown in Figure 7. As already mentioned, **MeltyFiNFT** is the contract that handles all the main lending and borrowing part of the protocol. MeltyFiNFT is also a contract that handles an ERC-1155 token, this token is none other than WonkaBar (the utility token). MeltyFiNFT is owner of **LogoCollection** (the meme token) to be able to mint at the user's request, it is owner of **ChocoChip** (the governance token) to be able to mint when due, and finally it is owner of **VRFv2DirectFundingConsumer** to make a request for a random word to be able to elect the winner of a completed lottery. **MeltyFiDAO** is the contract that manages the DAO of the protocol, so it needs a **TimelockController** and CochoChip as the governance token.

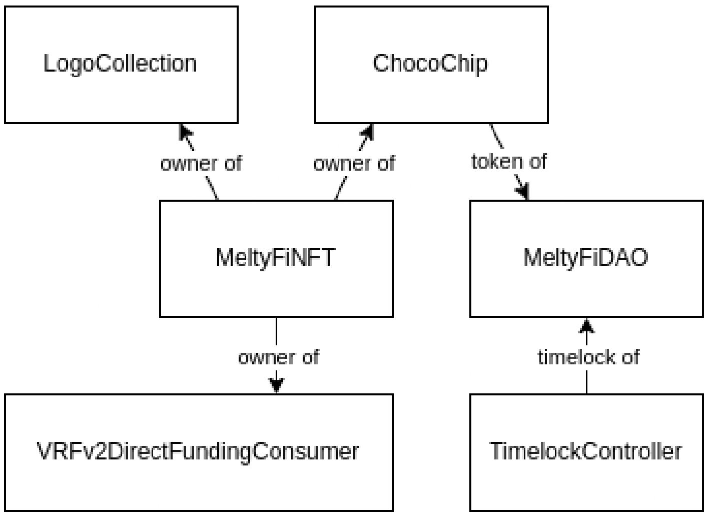


Figure 7: Protocol blueprint

4.3 LogoCollection

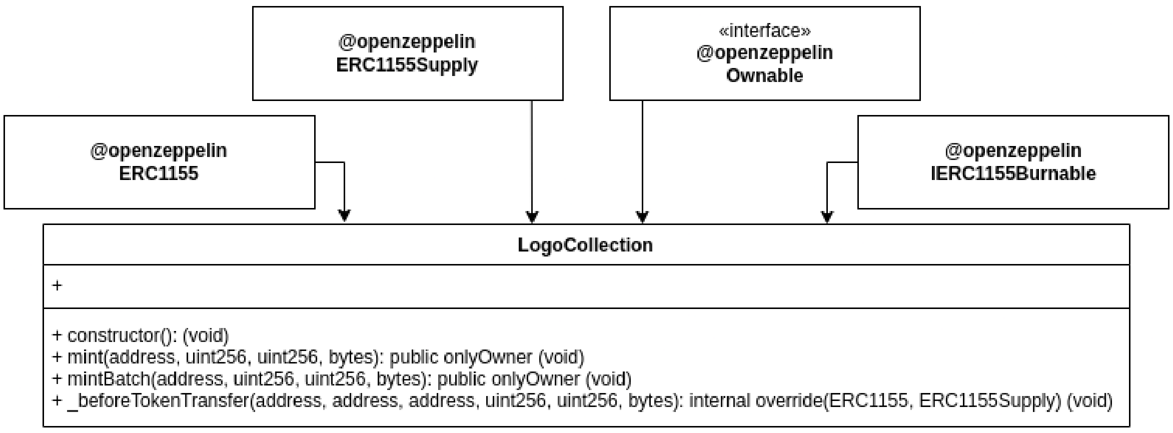

@openzeppelin
ERC1155Supply
«interface»
@openzeppelin
Ownable
@openzeppelin
ERC1155
@openzeppelin
IERC1155Burnable
LogoCollection
+
+ constructor(): (void)
+ mint(address, uint256, uint256, bytes): public onlyOwner (void)
+ mintBatch(address, uint256, uint256, bytes): public onlyOwner (void)
+ _beforeTokenTransfer(address, address, address, uint256, uint256, bytes): internal override(ERC1155, ERC1155Supply) (void)

Figure 8: LogoCollection.sol class diagram

LogoCollection is an **ERC-1155** token whose supply can be monitored, minable only by the owner and burnable by anyone. This token has no utility, in fact it is intended to be the **meme token**, created to publicize the protocol logo. So that is why it is minable by anyone via the `mintLogo` function found in MeltyFiNFT. The ERC-1155 standard was used to take advantage of the fungibility of ERC-20 tokens and the ability to integrate metadata as is done in ERC-721 tokens. The Figure 8 shows the class diagram of the LogoCollection contract, while the Figure 9 shows the image attached to the token.

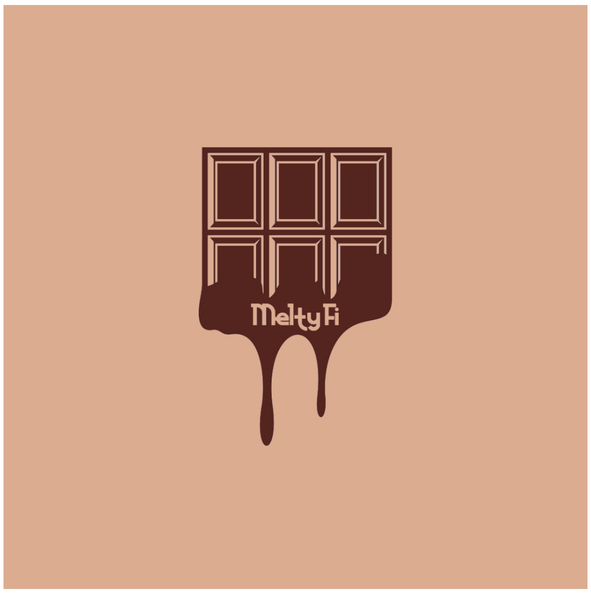


Figure 9: LogoCollection

4.4 ChocoChip

The Figure 10 shows the class diagram of the ChocoChip contract. ChocoChip ($CHOC) is an **ERC-20** token whose supply can be monitored, minable only by the owner, and burnable by anyone. ChocoChip has the property of allowing approvals to occur through signatures. Being a **governance token**, ChocoChip must necessarily also have the snapshot and voting property. ChocoChip starts with a supply of zero units and can only be minted if capital is employed in the protocol.

This is possible by buying WonkaBars or by repaying a loan. In the first case, buying WonkaBars is equivalent to buying tickets of active lotteries. In fact, when a lottery is concluded or cancelled, it will be possible to call the `meltWonakBars` function of MeltyFiNFT to melt your WonkaBars related to the concluded or cancelled lottery. Doing so will result in receiving 1 $CHOC for each Finney spent in the purchase of the WonkaBars being melted. In fact, this is equivalent to receiving a prize equal to the capital spent in the protocol.

In the second case, it is possible to receive ChocoChips if you repay a loan taken

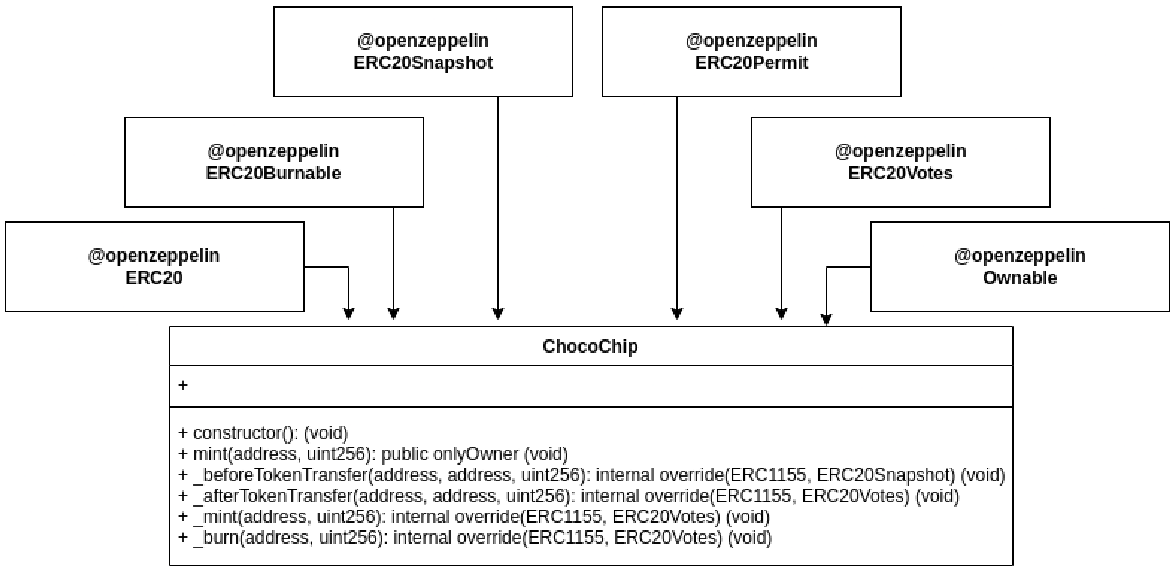

@openzeppelin
ERC20Snapshot
@openzeppelin
ERC20Permit
@openzeppelin
ERC20Burnable
@openzeppelin
ERC20Votes
@openzeppelin
ERC20
@openzeppelin
Ownable
ChocoChip
+
+ constructor(): (void)
+ mint(address, uint256): public onlyOwner (void)
+ _beforeTokenTransfer(address, address, uint256): internal override(ERC1155, ERC20Snapshot) (void)
+ _afterTokenTransfer(address, address, uint256): internal override(ERC1155, ERC20Votes) (void)
+ _mint(address, uint256): internal override(ERC1155, ERC20Votes) (void)
+ _burn(address, uint256): internal override(ERC1155, ERC20Votes) (void)

Figure 10: ChocoChip.sol class diagram

out with MeltyFiNFT. In fact, the borrower who creates a lottery, and then takes out a loan, will receive all earnings from the WonkaBars sold from that lottery, excluding a 5% royalty that is earmarked for MeltyFiDAO's treasury. To repay the loan, the borrower will have to pay back an amount equal to the value of all WonkaBars sold (doing the math, the interest is 5.26%). The difference between the full amount he pays back and the money he was actually loaned equals the value the borrower spent on the protocol and thus he will receive 1 $CHOC for every Finney spent on the protocol. So the borrower receives a prize equal to the capital spent in the protocol.

ChocoChip holders should be interested in holding the token because possession of the token itself demonstrates a kind of proof of work in having employed their capital in the protocol. However, a new ChocoChip is minted with each Finney employed in the protocol, and this feature allows the token to be backed in some sense to the value of $ETH. In addition, ChocoChip is the governance token of MeltyFiDAO, the protocol's decentralized autonomous organization. This means that $CHOC holders have decision-making power over protocol changes and updates based on the amount of $CHOC they hold. Obviously, since $CHOC is a token that demonstrates the holder's use of the protocol, it is correct to weigh votes based on the balance of $CHOC to intrapreneur decisions about protocol changes. ChocoChip, besides being the perfect token to employ in the DAO of the protocol, is also the perfect token to distribute the value of the treasury among all ChocoChip holders. In fact, for every circulating ChocoChip, there is one Finney in MeltyFiDAO's treasury. After these considerations, it's possible to see that DAO could decide to make the treasury worthwhile and distribute the revenue to the ChocoChip holders. In short, those who hold $CHOC will benefit proportionately from the proper use of the treasury and can make decisions about how to properly employ the treasury or how best to modify the protocol in a healthy way.

4.5 TimelockController

The Figure 11 shows the class diagram of the TimelockController contract. This contract is useful for the proper construction of MeltyFiDAO. Indeed, in a governance system, the TimelockController contract is responsible for introducing a delay between a proposal and its execution.

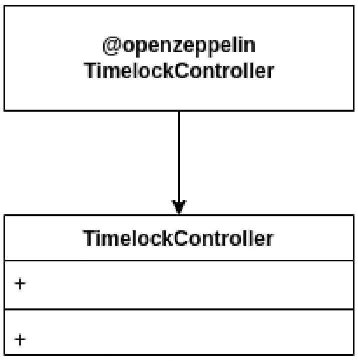
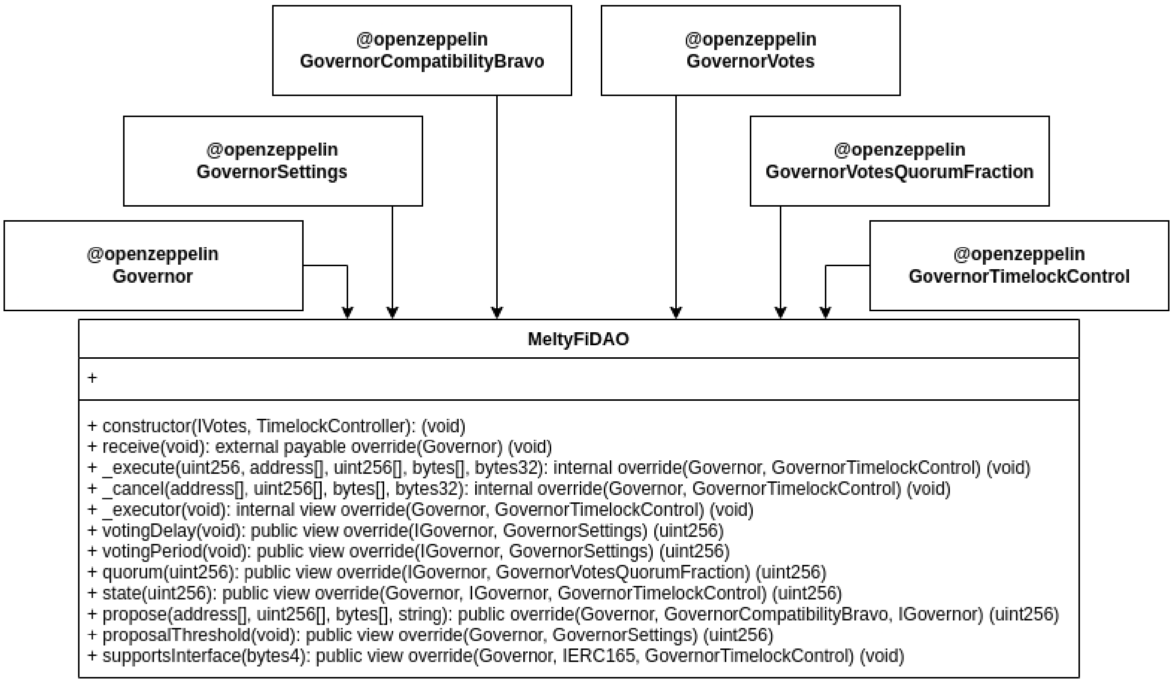


Figure 11: TimelockController.sol class diagram

4.6 MeltyFiDAO



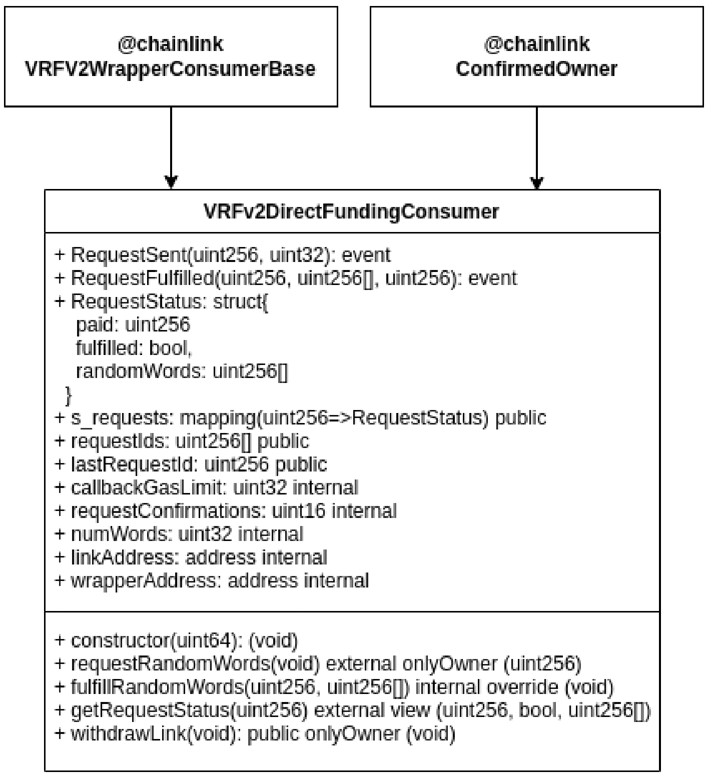
```
+ constructor(IVotes, TimelockController): (void)
+ receive(void): external payable override(Governor) (void)
+ _execute(uint256, address[], uint256[], bytes[], bytes32): internal override(Governor, GovernorTimelockControl) (void)
+ _cancel(address[], uint256[], bytes[], bytes32): internal override(Governor, GovernorTimelockControl) (void)
+ _executor(void): internal view override(Governor, GovernorTimelockControl) (void)
+ votingDelay(void): public view override(IGovernor, GovernorSettings) (uint256)
+ votingPeriod(void): public view override(IGovernor, GovernorSettings) (uint256)
+ quorum(uint256): public view override(IGovernor, GovernorVotesQuorumFraction) (uint256)
+ state(uint256): public view override(Governor, IGovernor, GovernorTimelockControl) (uint256)
+ propose(address[], uint256[], bytes[], string): public override(Governor, GovernorCompatibilityBravo, IGovernor) (uint256)
+ proposalThreshold(void): public view override(Governor, GovernorSettings) (uint256)
+ supportsInterface(bytes4): public view override(Governor, IERC165, GovernorTimelockControl) (void)
```

Figure 12: MeltyFiDAO.sol class diagram

The Figure 12 shows the class diagram of the MeltyFiDAO contract. MeltyFiDAO was programmed following Openzeppelin's best practices, taking advantage of Openzeppelin's audit code for the most part precisely. MeltyFiDAO uses ChocoChip as the governance token, is Bravo compatible and has a TimelockController.

4.7 VRFv2DirectFundingConsumer

VRFv2DirectFundingConsumer is a fair and **verifiable random number generator**, provided by chainlink, that allows MeltyFiNFT to access random values without jeopardizing security. VRFv2DirectFundingConsumer is called to generate a random word to elect the winner of a lottery when it concludes. For each request MeltyFiNFT makes to VRFv2DirectFundingConsumerFor some $LINK is spent from MeltyFiNFT's account. This is not a problem, it is the cost of the service. The Figure 13 shows the class diagram of the VRFv2DirectFundingConsumer contract.



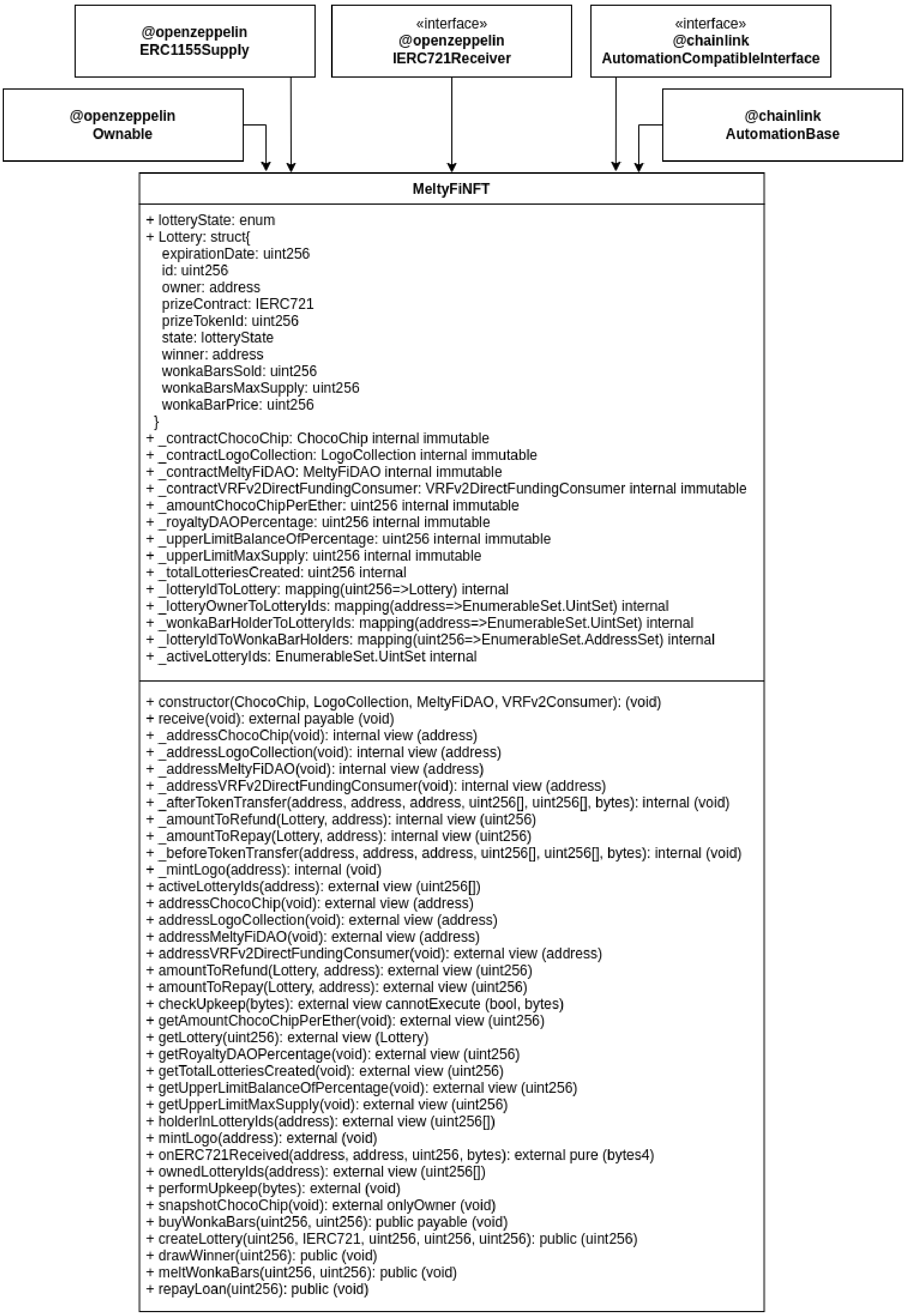
```
+ RequestSent(uint256, uint32): event
+ RequestFulfilled(uint256, uint256[], uint256): event
+ RequestStatus: struct{
    paid: uint256
    fulfilled: bool,
    randomWords: uint256[]
  }
+ s_requests: mapping(uint256=>RequestStatus) public
+ requestIds: uint256[] public
+ lastRequestId: uint256 public
+ callbackGasLimit: uint32 internal
+ requestConfirmations: uint16 internal
+ numWords: uint32 internal
+ linkAddress: address internal
+ wrapperAddress: address internal

+ constructor(uint64): (void)
+ requestRandomWords(void) external onlyOwner (uint256)
+ fulfillRandomWords(uint256, uint256[]) internal override (void)
+ getRequestStatus(uint256) external view (uint256, bool, uint256[])
+ withdrawLink(void): public onlyOwner (void)
```

Figure 13: VRFv2DirectFundingConsumer.sol class diagram

4.8 MeltyFiNFT

MeltyFiNFT is the contract that runs the core functionality of the MeltyFi protocol. It manages the creation, cancellation and conclusion of lotteries, as well as the sale and refund of WonkaBars for each lottery, and also reward good users with ChocoChips. The contract allows users to create a lottery by choosing their NFT to put as lottery prize, setting an expiration date and defining a price in Ether for each WonkaBar sold. When a lottery is created, the contract will be able to mint a fixed amount of



```
+ lotteryState: enum
+ Lottery: struct{
    expirationDate: uint256
    id: uint256
    owner: address
    prizeContract: IERC721
    prizeTokenId: uint256
    state: lotteryState
    winner: address
    wonkaBarsSold: uint256
    wonkaBarsMaxSupply: uint256
    wonkaBarPrice: uint256
  }
+ _contractChocoChip: ChocoChip internal immutable
+ _contractLogoCollection: LogoCollection internal immutable
+ _contractMeltyFiDAO: MeltyFiDAO internal immutable
+ _contractVRFv2DirectFundingConsumer: VRFv2DirectFundingConsumer internal immutable
+ _amountChocoChipPerEther: uint256 internal immutable
+ _royaltyDAOPercentage: uint256 internal immutable
+ _upperLimitBalanceOfPercentage: uint256 internal immutable
+ _upperLimitMaxSupply: uint256 internal immutable
+ _totalLotteriesCreated: uint256 internal
+ _lotteryIdToLottery: mapping(uint256=>Lottery) internal
+ _lotteryOwnerToLotteryIds: mapping(address=>EnumerableSet.UintSet) internal
+ _wonkaBarHolderToLotteryIds: mapping(address=>EnumerableSet.UintSet) internal
+ _lotteryIdToWonkaBarHolders: mapping(uint256=>EnumerableSet.AddressSet) internal
+ _activeLotteryIds: EnumerableSet.UintSet internal

+ constructor(ChocoChip, LogoCollection, MeltyFiDAO, VRFv2Consumer): (void)
+ receive(void): external payable (void)
+ _addressChocoChip(void): internal view (address)
+ _addressLogoCollection(void): internal view (address)
+ _addressMeltyFiDAO(void): internal view (address)
+ _addressVRFv2DirectFundingConsumer(void): internal view (address)
+ _afterTokenTransfer(address, address, address, uint256[], uint256[], bytes): internal (void)
+ _amountToRefund(Lottery, address): internal view (uint256)
+ _amountToRepay(Lottery, address): internal view (uint256)
+ _beforeTokenTransfer(address, address, address, uint256[], uint256[], bytes): internal (void)
+ _mintLogo(address): internal (void)
+ activeLotteryIds(address): external view (uint256[])
+ addressChocoChip(void): external view (address)
+ addressLogoCollection(void): external view (address)
+ addressMeltyFiDAO(void): external view (address)
+ addressVRFv2DirectFundingConsumer(void): external view (address)
+ amountToRefund(Lottery, address): external view (uint256)
+ amountToRepay(Lottery, address): external view (uint256)
+ checkUpkeep(bytes): external view cannotExecute (bool, bytes)
+ getAmountChocoChipPerEther(void): external view (uint256)
+ getLottery(uint256): external view (Lottery)
+ getRoyaltyDAOPercentage(void): external view (uint256)
+ getTotalLotteriesCreated(void): external view (uint256)
+ getUpperLimitBalanceOfPercentage(void): external view (uint256)
+ getUpperLimitMaxSupply(void): external view (uint256)
+ holderInLotteryIds(address): external view (uint256[])
+ mintLogo(address): external (void)
+ onERC721Received(address, address, uint256, bytes): external pure (bytes4)
+ ownedLotteryIds(address): external view (uint256[])
+ performUpkeep(bytes): external (void)
+ snapshotChocoChip(void): external onlyOwner (void)
+ buyWonkaBars(uint256, uint256): public payable (void)
+ createLottery(uint256, IERC721, uint256, uint256, uint256): public (uint256)
+ drawWinner(uint256): public (void)
+ meltWonkaBars(uint256, uint256): public (void)
+ repayLoan(uint256): public (void)
```

Figure 14: MeltyFiNFT.sol class diagram

WonkaBars (setted by lottery owner) for the lottery. These WonkaBars are sold to users interested in participating in the lottery and money raised are sent to the lottery owner (less some fees). Once the expiration date is reached, the contract selects a random WonkaBar holder as the winner, who receives the prize NFT. Plus every WonkaBar holder is rewarded with ChocoCips. If the lottery is cancelled by the owner beafore the expiration date, the contract refunds WonkaBars holders with Ether of the lottery owners. Plus every WonkaBar holder is rewarded with ChocoCips.

The Figure 14 shows the class diagram of the MeltyFiNFT contract. MeltyFiNFT extends the IERC721Receiver interface to allow the reception of ERC721 tokens. MeltyFi also extends the AutomationCompatibleInterface interface and the AutomationBase contract to enable the cunstom logic automation offered by chainlink, which is useful for ending a lottery when the expiration date is reached and then electing the NFT winner, i.e., the prize of the concluded lottery. MeltyFi also extends Ownable and ERC1155Supply for managing the WonkaBar token.

4.8.1 WonkaBar

WonkaBar ($WKB) is a token **ERC-1155** managed by MeltyFiNFT to generate **lottery tickets** for each lottery. The ERC-1155 standard is perfect for generating lottery tickets, in fact to each WonkaBar token ID we match a lottery with the same token ID so that for each lottery we get specific WonkaBars. In fact for each token ID, WonkaBar stores the balance of all users who own the respective WonkaBars to that token ID, thus to the lottery with the same token ID as the WonkaBars. The Figure 15 shows the image attached to the token.

Figure 15: WonkaBar

4.8.2 Lottery

Lottery is a structure maintained by MeltyFiNFT to hold all the useful information for each lottery. The structure is shown in detail below. Note that the lottery can have **four possible states**, described in detail in Figure 16 and commented on below.

A lottery is **active** when it has been created, the lottery owner has not yet cancelled it (i.e., has not yet executed `repayLoan` on the lottery), and the expiration date has not yet passed. When a lottery is active, it is possible to buy WonkaBars of the said lottery.

A lottery from active can become **cancelled** when the owner executes `repayLoan` before the expiration date and someone has bought WonkaBars from the lottery. If no one has bought lottery WonkaBars, then after `rapayLoan` the lottery is trashed. Of a cancelled lottery, holders of WonkaBars from that lottery can call `meltWonkaBars` to get back the money used in the purchase of WonkaBars and to get ChocoChips.

A lottery from active can become **concluded** if at the expiration date the owner has not yet repaid the loan and someone has bought WonkaBars from the lottery. If no one has bought lottery WonkaBars, then at the expiration date the lottery is trashed. Of a concluded lottery, holders of WonkaBars from that lottery can call `meltWonkaBars` to get ChocoChips. If `meltWonkaBars` is executed by the winner of the lottery, in addition to the ChocoChips, that winner also receives the NFT put in for the lottery.

A lottery is **trashed** when it has finished being active and there are no more (or never were) WonkaBar holders for that lottery.

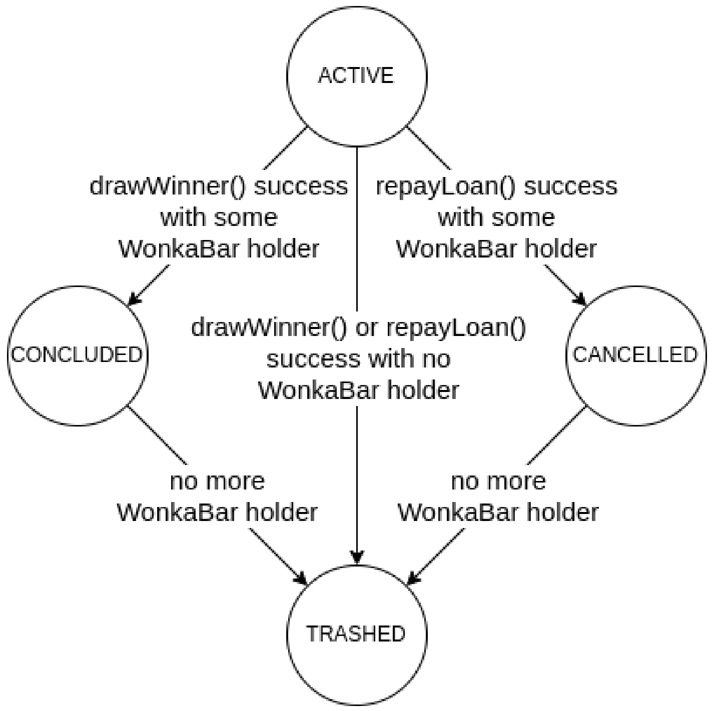

ACTIVE
drawWinner() success with some WonkaBar holder
repayLoan() success with some WonkaBar holder
CONCLUDED
drawWinner() or repayLoan() success with no WonkaBar holder
CANCELLED
no more WonkaBar holder
no more WonkaBar holder
TRASHED

Figure 16: Lottery state diagram

```
/// Data type representing the possible states of a lottery
enum lotteryState {
    ACTIVE,
    CANCELLED,
    CONCLUDED,
    TRASHED
}
/// Struct for storing the information of a lottery
struct Lottery {
    /// Expiration date of the lottery, in seconds
    uint256 expirationDate;
    /// ID of the lottery
    uint256 id;
    /// Owner of the lottery
    address owner;
    /// Prize NFT contract of the lottery
    IERC721 prizeContract;
    /// Prize NFT token ID of the lottery
    uint256 prizeTokenId;
    /// State of the lottery
    lotteryState state;
    /// Winner of the lottery
    address winner;
    /// Number of WonkaBars sold for the lottery
    uint256 wonkaBarsSold;
    /// Maximum supply of WonkaBars for the lottery
    uint256 wonkaBarsMaxSupply;
    /// Price of each WonkaBar for the lottery, in wei
    uint256 wonkaBarPrice;
}
```

4.8.3 Data structures

The following are descriptions of the data structures used by MeltyFi for proper operation. `_lotteryIdToLottery` is used whenever you want to retrieve the data of a lottery starting from the lottery ID. `_lotteryIdToWonkaBarHolders` is used in `drawWinner` to retrieve all the WonkaBar holders of the lottery whose winner you want to draw. `_activeLotteryIds` is useful in `checkUpkeep` to figure out which active lotteries reach expire date, so as to perform the conclusion of these and elect the winner. `_lotteryOwnerToLotteryIds` and `_wonkaBarHolderToLotteryIds` are useful for retrieving hard-to-find data immediately, useful only for those using MeltyFiNFT or interfaces that implement MeltyFiNFT.

```
/// maps a unique lottery ID to a "Lottery" object containing
↪  information about the lottery itself
mapping(uint256 => Lottery) internal _lotteryIdToLottery;
/// maps the address of a lottery owner to a set of lottery IDs that
↪  they own
mapping(address => EnumerableSet.UintSet) internal
↪  _lotteryOwnerToLotteryIds;
/// maps the address of a WonkaBar holder to a set of lottery IDs for
↪  which they have purchased a ticket
mapping(address => EnumerableSet.UintSet) internal
↪  _wonkaBarHolderToLotteryIds;
/// maps a lottery ID to a set of WonkaBar holder addresses that have
↪  purchased a ticket for that lottery
mapping(uint256 => EnumerableSet.AddressSet) internal
↪  _lotteryIdToWonkaBarHolders;
/// set that stores the IDs of all active lotteries
EnumerableSet.UintSet internal _activeLotteryIds;
```

4.8.4 createLottery function

This public function creates a new lottery. The function raises error in the following cases:

- Error if the caller is not the owner of the prize.
- Error if the maximum number of WonkaBars for sale is greater that the upper bound.

The function parameters are the following:

- **duration**: The duration of the lottery, in seconds.
- **prizeContract**: The contract that holds the prize for this lottery.
- **prizeTokenId**: The token ID of the prize for this lottery.
- **wonkaBarPrice**: The price of a WonkaBar in this lottery.
- **wonkaBarsMaxSupply**: The maximum number of WonkaBars that can be sold in this lottery.

The function return the ID of the new lottery.

```
function createLottery(
    uint256 duration,
    IERC721 prizeContract,
    uint256 prizeTokenId,
    uint256 wonkaBarPrice,
    uint256 wonkaBarsMaxSupply
) public returns (uint256)
{
    /// The maximum number of WonkaBars for sale must not be greater
    ↪  than the upper bound
    /// The maximum number of WonkaBars for sale must not be lower
    ↪  than the lower bound
    /// transfer the prize to this contract
    /// create a new lottery
    /// update internal state
    /// return the ID of the new lottery
}
```

4.8.5 buyWonkaBars function

This function allows a user to buy a specified amount of WonkaBars for a lottery. The caller must send the correct amount of Ether along with the transaction. A percentage of the total spending will be transferred to the MeltyFiDAO contract and the rest will be transferred to the owner of the lottery. The caller's balance of WonkaBars for the specified lottery will also be updated. The function raises errors in the following cases:

- Error if the lottery is not really active.
- Error if after this purchease the total supply of WonkaBars will exceed the maximum supply allowed.
- Error if the caller's balance of WonkaBars for this lottery, after the purchase, will exceed the `_upperLimitBalanceOfPercentage`.
- Error if the value sent is not enough to cover the cost of the WonkaBars.

The function parameters are the following:

- **lotteryId**: The ID of the lottery for which the WonkaBars are being purchased.
- **amount**: The number of WonkaBars to be purchased.

```
function buyWonkaBars(
    uint256 lotteryId,
    uint256 amount
) public payable
{
    /// retrieve the lottery with the given ID
    /// calculate the total spending for the WonkaBars
    /// The lottery must be really active
    /// After this purchease the total supply of WonkaBars must not
↪   exceed the maximum supply allowed
    /// The caller's balance of WonkaBars for this lottery, after the
↪   purchase, must not exceed the _upperLimitBalanceOfPercentage
    /// The caller must sent anough amount of Ether to cover the cost
↪   of the WonkaBars
    /// transfer _royaltyDAOPercentage of the total spending to the
↪   MeltyFiDAO contract
    /// transfer the rest of the total spending to the owner of the
↪   lottery
    /// mint the WonkaBars for the caller
    /// update the total number of WonkaBars sold for the lottery
}
```

4.8.6 repayLoan function

This function repays the loan for the given lottery ID. The caller of the function must be the owner of the lottery. The function raises errors in the following cases:

- Error if the caller is not the owner of the lottery.
- Error if the value sent is not enough to repay the loan.
- Error if the lottery is not more active.

The function parameters are the following:

- **lotteryId**: The id of the lottery to repay the loan for.

```
function repayLoan(
    uint256 lotteryId
) public payable
{
    /// retrieve the lottery with the given ID
    /// Calculate the total amount to be repaid
    /// The caller must be the owner of the lottery
    /// The caller must sent anough amount of Ether to repay the loan
    /// The lottery must be active
    /// Mint Choco Chips to the owner of the lottery
    /// Transfer the prize to the owner of the lottery
    /// Remove the lottery from the active lotteries
    /// set the expiration date to the current block timestamp
    /// If the total supply of WonkaBars is 0
        /// set the state to TRASHED
    /// else
        /// set the state to CANCELLED
}
```

4.8.7 drawWinner function

This function draws the winner of a lottery. The function raises errors in the following cases:

- Error if the lottery state is not active.
- Error if the lottery expiration date is not passed.
- Error if the VRF request for random words is not fulfilled.

The function parameters are the following:

- **lotteryId**: The ID of the lottery.

```
function drawWinner(
    uint256 lotteryId
) public
{
    /// retrieve the lottery with the given ID
    /// The lottery state must be active
    // The lottery expiration date must be passed
    /// remove lottery from the active lotteries
    /// if there are no WonkaBar sold transfer prize to the owner,
 ↪  otherwise set lottery winner
        /// transfer prize to the owner if no tokens were sold
        /// set lottery state to trashed
    /// else
        /// The VRF request for random words must be fulfilled
        /// set lottery winner
        /// set lottery state to concluded
}
```

4.8.8 meltWonkaBars function

This function allows a user to melt their WonkaBars of a specific lottery when this is no longer active. If the lottery is canceled the sender will receive the refund and ChocoChips. If the lottery is concluded the sender will receive ChocoChips, and the lottery prize if he is the winner. The function raises errors in the following cases:

- Error if the user does not have enough WonkaBar balance to melt the given amount.
- Error if the lottery is trashed.
- Error if lottery is really active.
- Error if lottery is waiting to be concluded by the oracle.

The function parameters are the following:

- **lotteryId**: The ID of the lottery from which the WonkaBars will be melted.
- **amount**: The amount of WonkaBars to be melted.

```
function meltWonkaBars(
    uint256 lotteryId,
    uint256 amount
) public
{
    /// retrieve the lottery with the given ID
    /// calculate the total refound for the WonkaBars
    /// the user must have enough WonkaBar balance to melt the given
↪   amount
    /// the lottery must not be trashed
    /// lottery must not be really active or waiting to be concluded
↪   by the oracle
    /// Burn the WonkaBars for the caller
    /// Mint Choco Chips to the caller
    /// if lottery state is cancelled, also refound the caller
    /// if the caller is the winner and he does not already receive
↪   the price
        /// transfer prize to the caller (the winner)
    /// if all lottery's WonkaBars are melted, trash the lottery
}
```

4.9 Deepening of the design choices

Delegate call issue Delegate call was not used to transfer the ownership of LogoCollection, ChocoChips, VRFv2DirectFoundingConsumer to MeltyFiNFT, but it was done manually. It would have been convenient to use delegate call at the time of the constuction of MeltyFiNFT so that the ownership of all three contracts could be transferred to meltyFiNFT, but it was not able.

Merge dilemma between MeltyFiNFT and WonkaBar It was considered several times during the outlining of the code structure whether or not to merge the contract that handled MeltyFiNFT with the one that handled WonkaBar. In the end, the solution was that they necessarily needed to be merged because otherwise there would be data inconsistency when WonkaBars were sent to address zero or simply when they were passed from one wallet to another MeltyFiNFT would store the old owner and not update this information. It was necessary to make these two contracts talk tightly if they were separate, so the decision that was made was to merge the two contracts to easily have consistency of all the data that the protocol needs to work properly. So WonkaBar and MeltyFiNFT effected the merge. Note also that with the merge it is then possible to easily update the data managed by MeltyFiNFT related to the WonkaBars hand in hand with the data itself of the WonkaBar token, this just in the `_beforeTokenTransfer` and `_afterTokenTransfer` functions.

WonkaBar sold and WonkaBar supply are different things For each lottery it is necessary to know both the WonkaBar supply of the relevant lottery and the WonkaBars sold. These two data may not have the same value when, for example, some WonkaBars are sent to address zero. In this case the WonkaBars sold would appear greater than the supply, and if the supply was used to calculate the WonkaBars sold, after some WonkaBars were sent to address zero, it would result in the borrower paying less than he should. One therefore needs both pieces of data. WonkaBar sold is useful for calculating exactly how many WonkaBars were sold (thus how many were mint) while WonkaBar supply is useful for figuring out whether a lottery is to be trashared or not. WonkaBar sold and WonkaBar supply are two data therefore that we need, they play two different roles and can be altered differently.

Looking for the best data structures The EnumerateSet data structures provided by Openzeppelin were used to rely on as much audited code as possible when possible, also making programming easier without having to think about building basic data structures for storing information. so only more complex, non-standard structures were built. These structures are all MeltyFi maps and they are all necessary, each essential for at least one function.

There had also been thought of creating mapping mappings etc. To partition well all lottery related info by doing lottery state discrimination. but this was not good as an approach because solidity is not possible to do swaps of mappings but only of type date base (it would have been very complex to effect a lottery state change). So we focused more and more on formulating simple mappings that are indispensable to the operation and use of the protocol without storing derivable data.

Lottery constraints spiegare perche occorre _upperLimitBalanceOfPercentage per evitare disinteresse nel comprare wonkabar in una lotteria se un solo address detiene troppa probabilita di vittoria e _upperLimitMaxSupply per evitare cicli troppo lunghi nel drawWinner, serve anche un limite minimo di wonkabar per permettere almeno un minimo di un biglietto per utente, quindi min e 5 con 25% di _upperLimitBalanceOfPercentage

MeltyFi is safe without oracles MeltyFi does not rely on any external factors except the interaction with the Chainlink oracles. What happens if the oracles do not do their duty? Maybe because they are late or maybe because the $LINKs are missing in the accounts that fund the oracles. What happens is that you do not compromise the protocol because the consistency of the data remains intact and you would only delay the execution of the draw winner of the concluded lotteries. As to who is to fund link the oracles, the cunstom logic automation [52] must necessarily rely on an address that actively sends funds to chainlink, so it must necessarily be the address of the protocol deployer. Note that even if the protocol deployer does not fund the oracle with the $LINKs, it is still possible to call `drawWinner`, because precisely cunstom logic automation is not strictly necessary, since it is just an automation of a call of `drawWinner`, which however can be done by anyone when the time comes to be able to do it. For the generation of the random number (for the drawing of the lottery winner) the oracle is necessarily needed, and a direct funding [53] was specifically chosen so that if the deployer does not fund the oracles, anyone can fund it in order to be able to enable the execution of `drawwinner`. So the protocol even if the deployer lacks funding or if the oracles delay execution, the protocol is not compromised.

Deploy best practice Obviously to deploy all the protocol contracts we chose to deploy them singly following the order dictated by the inter-contractual dependencies. Of course, it was not chosen to deploy a single contract that would deploy others; it would have been too costly unnecessarily.

No snapshot for WonkaBar token at the lotteries conclusion A snapshot of the WonkaBars is not taken when the lotteries expire (conclude) because it is not essential anyway and the way the protocol is structured it is not necessary.

5 Implementation

5.1 Tools used

React is a JavaScript library for building user interfaces. It is useful because it allows developers to build web applications with a component-based architecture, which makes it easy to manage the state and logic of different parts of the application separately. React also uses a virtual DOM, which optimizes the performance of updates and improves the overall speed of the application. React also has a large and supportive community and a wide range of available resources and third-party libraries, making it a popular choice for developers building web applications.[54]

React-Bootstrap is a library of pre-built components for React that allow developers to easily add Bootstrap styling to their projects. The use of pre-built components can help to improve the consistency and maintainability of the codebase. [55]

Ethers.js is a JavaScript library for interacting with the Ethereum blockchain. It provides a simple and easy-to-use interface for developers to interact with smart contracts, send transactions, and manage their digital assets on the Ethereum blockchain. Ethers.js is built on top of the Ethereum JSON-RPC API and supports both the web3.js and Web3.eth APIs. It is compatible with both Node.js and browser environments. Ethers.js is useful for developers who want to build decentralized applications (DApps) on the Ethereum blockchain. It abstracts away the complexities of working directly with the Ethereum JSON-RPC API, making it easy for developers to interact with smart contracts, send transactions, and manage digital assets. Additionally, Ethers.js has a lot of functionalities that are not present in web3.js, such as contract events, contract transactions, and contract deployment. [56]

thirdweb is a development framework that allows you to build web3 functionality into your applications. [57]

MetaMask is a browser extension and cryptocurrency wallet that allows users to easily interact with decentralized applications (DApps) on the Ethereum blockchain. It acts as a bridge between the user's browser and the Ethereum network, providing users with a secure and user-friendly way to manage their cryptocurrency and interact with DApps. MetaMask is useful because it eliminates the need for users to run a full Ethereum node, making it easy for anyone to access and interact with the Ethereum blockchain. Additionally, it provides a simple and secure way for users to manage their cryptocurrency and interact with dApps, making it a powerful tool for users and developers alike. [58]

OpenSea Testnets API is used for browsing non-fungible assets on the Ethereum Goerli test network. We used it for retrieving all the asset for a certain address. [59]

5.2 MeltyFi.NFT DApp

In this section, we will delve into the research and development process that led to the implementation of specific solutions for the frontend application of MeltyFi. We will examine the difficulties encountered during the process and how the tools and technologies cited above were utilized to overcome these challenges. The purpose of this analysis is to provide a comprehensive understanding of the decision-making process and the rationale behind the chosen solutions.

5.2.1 Home page and website fruition

The placement of the protocol description on the homepage serves as an effective introduction for users who may be unfamiliar with the technology. This approach allows them to quickly grasp the purpose and functionality of the protocol. As the protocol becomes more widely adopted, this information may be relocated to an ”Info” section, accompanied by tutorials to enhance the user’s understanding and utilization of the DApp. As best practice we inserted a navbar that linked to the various pages, so that in each point of the DApp the user could easily move around the website. At the bottom we inserted a footer that contains useful addresses such as MeltyFi.

A fundamental thing is that thirdweb has a standard implementation of the login button that is really nice, and well integrated with MetaMask.

5.2.2 Lotteries page

The layout of this page features two tabs, ”Browse Lotteries” and ”Create Lottery”. In order to access either of these tabs, it is necessary to connect a wallet. In principle, it is only necessary to have Metamask installed to view active lotteries, however, we opted to log in with a MetaMask account at the beginning of the session, rather than when purchasing tickets, as this will facilitate a faster buying process (crucial in case of a quick lottery).

Browse Lotteries tab Allows users to view lotteries created by other users. In designing this feature, we conducted research on other lottery websites and aimed for the best user experience. This led us to the use of cards for each lottery. Since they feature an NFT as a prize, the actual image of the NFT is vital, as users are highly influenced by what they see when purchasing NFTs. In addition, the card displays relevant information about the lottery, such as the date, number of tickets sold, and the price. The only button present on each card allows users to purchase tickets for that lottery.

The recurring principle for the front end development was to optimize the user experience as much as possible. To achieve this goal, various subtle enhancements were implemented. For instance, the date is displayed according to the browser's language settings. Additionally, prices that are too low to be significant are not displayed, and an indicator of the number of tickets sold is provided to give users an indication of the popularity of a given lottery.

When a user decides to purchase a lottery ticket, they enter into a modal view, where the background is blurred to focus attention on the information for that lottery. This view includes a link to the user contract and another link to the NFT contract, allowing users to conveniently decide if the owner and contract are worth trusting. The Wonkabar price is also displayed no matter how small, and a selector allows users to choose the number of tickets to purchase, with the final price automatically computed. The recurrent idea is to prevent users from making mistakes in order to avoid the user (and the chain) failing transactions. The user can insert only positive integers numbers for the tickets. Additionally, the protocol forbids buying more than 25 percent of the tickets, so the user can't insert a number of tickets that makes this property false. The front end prevents many errors that could occur, but in the event of problems, an error message will be displayed. When the user clicks on buyWonkaBar, a MetaMask form will appear, and the modal view will close once the transaction is completed (unless there's an error). Another way in which the front-end helps the backend is in the case of the oracles failure (that probably due to insufficient funds, don't declare lotteries as closed), so the frontend filters the active lotteries given by the backend adding also a filter for lottery with an expiry date that is expired. In one case, the fronted actually modified the backend: at the beginning there were many calls to know the information about a lottery. We realized that this really slowed the frontend, so we instead did only one call that returned all the information about a lottery. The loading speed of the frontend was one of the reason for adopting Goerli, a fairly fast testnet.

Create Lottery tab Allows users to view and select from their available ERC721 NFTs to use as collateral for a loan. To retrieve a list of NFTs associated with a given address, the OpenSea API was utilized, as there was no existing method for easily retrieving NFTs using Ethers library, we reluctantly made this choice as it added yet another dependency. We filtered ERC721 type NFTs, the only that could be selected as collateral. The card displayed for each NFT in the "Create Lottery" tab includes the collection name, token ID, and a button to initiate the lottery creation process. Once the user click on createLottery, they are presented with a modal view to input the value of each lottery ticket in Wei. The choice to use $Wei was made as it allows for more flexibility in selecting values, particularly with limited resources. The option to use $ETH in the future cannot be excluded. A date picker is also provided for the user to select the expiration date of the lottery, rather than requiring manual input of seconds. The total revenue from the sale of all lottery tickets is also displayed for the user's reference. To create a lottery, the user must first approve the transfer of the selected NFT to the MeltyFi protocol's address. The frontend first waits for the approval transaction (that authorizes the transfer of the NFT from the owner to MeltiFy) to be logged on the blockchain before initiating the lottery creation process. This is done to prevent failed transactions. If at any point the transaction fails, an error message is displayed to the user.

5.2.3 Profile page

The Profile page can only be accessed by a logged user. After logging in it will immediately show, in the top right corner, the balance in ChocoChips. In the middle of the page, two containers will show the owned lotteries and the lotteries for which the user has bought at least one WonkaBar.

In this page, as in the Lottery page, lotteries are shown as cards in which the NFT image is the most important part. Card descriptions and buttons are different based on the state of the lottery.

Owned lotteries The expire date is the first information shown (expressed in the browser's language format), followed by the number of WonkaBars sold over the total supply of WonkaBars, and the amount of ETH to repay. In this section, the amount to repay is shown in its entirety, no matter how small it is, in order to give the user an accurate overview of their lotteries' state.
Here the button can be used to repay the loan: when clicked, Metamask will ask the user to sign the transaction, in which the amount to repay is already set. In case an error occurs, an alert box is prompted.
An example transaction that uses this function can be found at `0xc559...b984`.

WonkaBars bought Lotteries for which the user has bought at least one WonkaBar share some common information with the owned lotteries (expire date, WonkaBars sold over the total supply) but the button's function is completely different: here the button is used to melt all the owned WonkaBars to receive a reward, based on the state of the lottery. Also some additional information is shown:

- **Active lotteries**: the state of the lottery (Active) and the win probability for the user: this value is computed as $wonkaBarsOwned/wonkaBarsSold * 100$ (rounded to the second decimal digit), since no matter how much the total supply of WonkaBars is, what counts is how many WonkaBars have been sold and how many of them are owned by the user.
  In this case the button is disabled, since WonkaBars can only be melted after the lottery is no more active.

- **Cancelled lotteries**: the state of the lottery (Cancelled) is followed by the winner, which in the case of a cancelled lottery is None. Instead of the win probability (useless now since the lottery is already over) the user is told what they will receive when melting WonkaBars: for cancelled lotteries the user will receive the refund and some ChocoChips.
  Clicking on the button will open a Metamask window in which the user will be asked to sign the transaction.
  An example transaction that uses this function can be found at `0xd957...7c3f`

- **Concluded lotteries**: the state of the lottery (Concluded) is now followed but the address of the winner. The address, too long to fit in a card, is shortened in the form `0x0123....4567` and is directly linked to the address page on Etherscan[60]. For lotteries in this state the user will only receive ChocoChips, since the loan has not been repaid.

5.3 Deepening in the implementation

A significant amount of time and effort was dedicated to carefully evaluating and selecting the most appropriate frameworks for our project. The rationale behind this approach is that investing time in researching and selecting the right frameworks can ultimately save time in the long run.

One such framework that proved to be worthwhile was React-Bootstrap. Despite its steep learning curve, it allowed us to create a visually pleasing website that would have been difficult to achieve using plain JavaScript alone. However, the documentation for this framework proved to be a challenge, requiring a significant amount of time to fully understand and utilize it effectively.

Unfortunately, not all frameworks were as successful. ethers-react, for example, promised to be highly compatible with React, but ultimately failed to function. For connecting to the backend, we ultimately chose to use the ethers.js library, which is known for its lightweight characteristics compared to web3. Despite initial difficulties in implementation, we were ultimately able to make it work.

We also explored the Thirdweb framework, which promised to simplify the development process. While it did simplify certain aspects, such as the login button, it ultimately proved to be ineffective for other aspects of the project. Despite initial efforts to make it work, we ultimately returned to using the Ethers framework for rendering NFTs, which proved to be a slower (but functioning) process.

5.4 MeltyFi.DAO DApp

Although the MeltyFiDAO smart contract has been successfully deployed on the Goerli Testnet, we have not yet implemented a user interface to facilitate interactions with the contract. This was a conscious decision as we wanted to provide a foundation for developers to build upon, leaving the MeltyFiDAO contract as a skeleton structure for future development and customization. While the user interface for interacting with the MeltyFiDAO smart contract has not yet been developed, it is still possible for users to interect with the contract through the use of Goerli Etherscan interface.

6 Known issues and limitations

For the borrower is hard to get a fast loan, since funds are to be retrieved from many lenders who buy WonkaBars. Moreover lenders are uncertain about the future of the lottery, because there are three different endings:

- first one, if the borrower repays the loan, lenders will be awarded with ChocoChips and refunded of their investment;
- second, if the borrower doesn't repay the loan and the WonkaBar does not have the golden ticket, the lender will only be awarded with ChocoChips;
- but if the the WonkaBar has the golden ticket (third case), the lender will be awarded with ChocoChips and the NFT of the borrower.

7 Conclusions

We presented MeltyFi, a new type of lending and borrowing protocol based on the peer-to-pool design, in which the loan collateral is NFT and the funds are raised through a lottery ticket foundraising mechanism. MeltyFi's design allows borrowers not to risk seeing their NFT liquidated unexpectedly because there is no dependence on external factors such as floor price. Borrowers also have full control over how much they wish to borrow (no predetermined loan-to-value, it is simply decided by the market) and the expire date of the loan. It will then be up to the lenders to finance the borrowers' loan requests. In this the lenders are motivated since they can choose which loan to finance, plus through the mechanism of expire lotteries the lender is incentivized to finance the borrowers because the lottery will be concluded at expire dates and not when the tickets are all sold. This still spurs people to buy tickets, facilitate financing for borrowers, and increase earnings for lenders in case of cancelled lotteries.

In short, we have succinctly summarized the main features of the MeltyFi protocol and pointed out some win-win mechanisms for all users. In fact, with the proposed design, many issues present in platforms with the same intent as meltyfi are simply not there. In addition to not presenting many problems that plague other platforms, MeltyFi incentivizes lending and borrowing in a healthy way by minimizing risk and maximizing benefits for all users participating in the protocol.

MeltyFi **melts finance** in a really elegant way and manages to turn even really very illiquid and volatile assets like NFTs into liquid assets.

References

[1] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. https://ethereum.org/en/whitepaper.

[2] Coinmarketcap. https://coinmarketcap.com.

[3] Non-fungible tokens (nft). https://ethereum.org/en/nft.

[4] Kristen E. Busch. Non-fungible tokens (nfts). Government Report R47189, CRS Reports, July 2022. https://crsreports.congress.gov/product/pdf/R/R47189.

[5] Christie's beeple (b. 1981). https://onlineonly.christies.com/s/first-open-beeple/beeple-b-1981-1/112924.

[6] Valuables auction site 'just setting up my twttr'. https://v.cent.co/tweet/20.

[7] Lennart Ante. The non-fungible token (nft) market and its relationship with bitcoin and ethereum. *FinTech*, 1(3):216–224, 2022.

[8] Nonfungible nft market history. https://nonfungible.com/market-tracker.

[9] Chainlink. What is an nft floor price? https://blog.chain.link/what-is-an-nft-floor-price, October 2022.

[10] Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Eugenio Nerio Nemmi. Nft wash trading in the ethereum blockchain. *arXiv preprint arXiv:2212.01225*, 2022.

[11] What is decentralized finance (defi) and how does it work? https://www.investopedia.com/decentralized-finance-defi-5113835.

[12] Defi terms. https://medium.com/coinmonks/what-is-liquidation-in-defi-lending-and-borrowing-platforms-3326e0ba8d0.

[13] Collateral definition, types, & examples. https://www.investopedia.com/terms/c/collateral.asp.

[14] Loan-to-value (ltv). https://coinmarketcap.com/alexandria/glossary/loan-to-value-ltv.

[15] What is liquidation? https://www.investopedia.com/terms/l/liquidation.asp.

[16] What are liquidity pools? https://www.gemini.com/cryptopedia/what-is-a-liquidity-pool-crypto-market-liquidity.

[17] Aave. https://aave.com.

[18] Aave tvl. https://defillama.com/protocol/aave.

[19] Compound. https://compound.finance.

[20] Compound tvl. https://defillama.com/protocol/compound.

[21] Jon Frost Doerr, Anneke Kosse, Asad Khan, Ulf Lewrick, Benoît Mojon, Benedicte Nolens, and Tara Rice. Defi risks and the decentralisation illusion. *BIS Quarterly Review*, page 21, 2021.

[22] Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of defi liquidations: Incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 336–350, 2021.

[23] Monika Ghosh. Nft-backed loans gain traction, but what are the risks? https://forkast.news/nft-backed-loans-gain-traction-risks.

[24] Alex White-Gomez. Using nfts as collateral: A crash course with franklinisbored. https://www.one37pm.com/nft/using-nfts-as-collateral, October 2022.

[25] Danny Nelson. Bank run at nft lender benddao prompts attempt to avert another liquidity crisis. https://www.coindesk.com/business/2022/08/22/bank-run-at-nft-lender-benddao-prompts-attempt-to-avert-another-liquidity-crisi August 2022.

[26] David Pan. Rising ether prices spark liquidity crisis at nft lender benddao. https://www.bloomberg.com/news/articles/2022-08-23/nft-lender-seeks-to-ease-liquidity-crisis-with-protocol-changes, August 2022.

[27] Arcade: Decentralized nft loan marketplace. https://www.arcade.xyz.

[28] Benddao: Web3 data liquidity. https://www.benddao.xyz.

[29] Drops: Nft as collateral & instant nft loans. https://drops.co.

[30] Jpeg'd: bridging the gap between defi and nfts. https://jpegd.io.

[31] Liquidnfts: Borrow against your nfts or fund loans to earn. https://liquidnfts.com.

[32] Nftfi: Borrow & lend on the leading nft liquidity protocol. https://www.nftfi.com.

[33] Nftgo. https://nftgo.io.

[34] Pine: Instant nft loans. https://pine.loans.

[35] renft: Rental infrastructure for the metaverse. https://www.renft.io.

[36] X2y2 marketplace. https://x2y2.io.

[37] Charlie and the chocolate factory. https://www.warnerbros.com/movies/charlie-and-chocolate-factory, July 2005.

[38] VincenzoImp. Meltyfi github repository. https://github.com/VincenzoImp/MeltyFi, October 2022.

[39] Node.js. https://nodejs.org.

[40] Openzeppelin contracts. https://www.npmjs.com/package/@openzeppelin/contracts.

[41] Chainlink contracts. https://www.npmjs.com/package/@chainlink/contracts.

[42] Hardhat. https://www.npmjs.com/package/hardhat.

[43] Hardhat toolbox. https://www.npmjs.com/package/@nomicfoundation/hardhat-toolbox.

[44] Hardhat docgen. https://www.npmjs.com/package/hardhat-docgen.

[45] Dotenv. https://www.npmjs.com/package/dotenv.

[46] Alchemy api. https://docs.alchemy.com/reference/api-overview.

[47] Etherscan api. https://etherscan.io/apis.

[48] Remix: The native ide for web3 development. https://remix-project.org.

[49] Gerli testnet. https://goerli.net.

[50] Goerli faucet. https://goerlifaucet.com.

[51] Link faucet. https://faucets.chain.link.

[52] Chainlink cunstom logic automation. https://docs.chain.link/chainlink-automation/register-upkeep.

[53] Chainlink vrf direct funding. https://docs.chain.link/vrf/v2/direct-funding.

[54] React: A javascript library for building user interfaces. https://reactjs.org.

[55] React bootstrap: The most popular front-end framework. `https://react-bootstrap.github.io`.

[56] ethers.js: a simple, compact and complete library for all your ethereum needs. `https://ethers.org`.

[57] thirdweb: The complete web3 development framework. `https://thirdweb.com`.

[58] Metamask: The crypto wallet for defi, web3 dapps and nfts. `https://metamask.io`.

[59] Opensea testnets api. `https://docs.opensea.io/reference/testnets-api-overview`.

[60] Goerli etherscan. `https://goerli.etherscan.io`.