Universitetet
i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Datateknologi	Vårsemesteret 2022 [✓ Åpen] eller Konfidensiell

Forfatter(e): Aleksander Vae Haaland, Marcus Meihack Thompson

Fagansvarlig: Antorweep Chakravorty

Veileder(e): Antorweep Chakravorty

Tittel på bacheloroppgaven: NFT Marketplace

Engelsk tittel: NFT Marketplace

Studiepoeng: 20

Emneord: NFT, Marketplace, Blockchain Ethereum, Solidity	Sidetall: 24 + vedlegg/annet: 37 Stavanger 15. mai 2022

Contents

Acknowledgements

We would like to thank our internal supervisor and course coordinator Ass. Prof Antorweep Chakravorty (University of Stavanger) for his guidance and support through this thesis.

Abstract

With the rapidly growing blockchain global markets, developing decentralised applications residing on blockchains such as Ethereum, Fantom and Avalanche are of growing interest [Publishing, 2021]. Therefore, the main objective of this thesis is to investigate non-fungible tokens and develop a trading platform for users with Ethereum wallets to mint, buy and sell tokens between themselves. To do this the decentralised application will utilise smart contracts deployed on an Ethereum TestNet and Next.js in combination with Tailwind CSS to create a user-friendly experience in form of a single page application.

Link to the Vercel hosted application

Link to GitHub repository for application source code

Chapter 1

Introduction

1.1 Background & Motivation

NFT is an abbreviation for non-fungible token. Fungible indicates that one has an asset that is exchanged for another. It is a commodity, similar to a dollar or a gram of gold. If someone was to exchange a dollar bill for another, it makes no difference which dollar bill one has seeing as they have the same monetary value. When something is non-fungible, it is one-of-a-kind.

Ethereum is related to a cryptocurrency called ether. One of the reasons it is such a unique system is that it allows for the creation of new currencies and tokens on top of it, but the NFT was an innovation apart from that. This token cannot be swapped for any other token. It is a one-of-a-kind token, and one's ownership of this particular token is noteworthy because it signifies something unique.

While NFTs have recently been connected with artwork, the reality is that they can represent anything, which is how its proponents perceive it. As a symbol of what decentralised finance could potentially be. The concept of NFTs are seen as far back as 2012, but they did not gain much traction until 2014, when people began creating and exchanging these tokens. When NFTs truly gained traction was when "CryptoKitties" launched. These were cartoon cats that one could purchase, collect, breed, and sell. They were all unique, and some of those ended up selling for hundreds of thousands of dollars, and that is when this concept took off. Around the same time, there was an early NFT project called "crypto punks." These pixelated images of faces were generated, and people began buying and selling them. Today these are selling for exorbitant prices, but the true sonic boom occurred when Michael Joseph Winkelmann, known as "Beeple," sold his 69 million dollar sketchbook called "Everydays" through Christie's auction house. This was something that everyone noticed, and it is still the most popular NFT artwork to date. [Marcobello, 2022] [Christie's, 2021]

Many people view NFTs as a get-rich-quick scheme. One could earn millions of dollars, or one can earn nothing and lose their money due to transaction costs and the cost of purchasing ether to mint the NFT in the first place. By definition, a given NFT is only worth what someone else is prepared to pay for it. These things will occur in fads and interest waves. Prior to digital scarcity, the issue with digital content was that one could not fully own it. For example, after finishing reading a book bought on a kindle there is not much one can do. It is not possible to sell the book second-hand. With non-fungible tokens people can truly own it in a way that they could not have done previously.

Artists have developed a following of passionate admirers who follow their work on platforms like Instagram, but as one knows, artists have often not been reimbursed for that labour until now. Now, for the first time, NFTs are providing a means for these digital artists to profit from their labour.

The public will begin to understand what it means to own something digitally and possibly appreciate the potential significance of those items.
So the billion-dollar question is whether NFTs are a fad or are here to stay.

Many crypto investors hope that NFTs will become a natural part of this decentralised financial world. That NFTs will become a method to express ones fandom and demonstrate ones affiliation with a particular group or community. There are so many different things that can be NFTs, and while some of them will be temporary, it's almost certain some will stick around.

The global blockchain market is estimated to reach $56.7 billion by 2026 from $6 billion in 2021. The forecasted compound annual growth rate during this period is 56.9%. Learning and developing an application that is connected to such a rapidly growing industry, could be of interest. [Publishing, 2021]

1.2 Objectives

The Objective of this project is to develop a web application that allows users with an Ethereum wallet to mint, buy and sell their NFTs.

Objectives for our project

- User authentication and login with Ethereum wallet
- Restricted file upload
- Mint token to blockchain with smart contracts
- List token for sale with user regulated price
- Cancel listing, returning token to owner
- Buying token that is listed
- Transferring token to new owner
- Rich description with markdown editor

Chapter 2

Technology Choices

2.1 React

For this NFT marketplace the goal was to make a responsive and modern web app that would function with mobile as well. React is a JavaScript library used to build user interfaces specifically for SPAs. This makes it easier to create an responsive and modern looking app. The library allows the developer to reuse UI components which eases the Front-end development. One can develop React apps using either Typescript or JavaScript. This project however utilises JavaScript.

Reference [Platforms, 2022]

2.1.1 Next.js

To make life even easier, this project uses Next.js in addition to React. Next.js is used by some of the biggest websites like Twitch.TV, GitHub, Netflix and Ticketmaster. Next.js is a React framework for constructing JavaScript single-page applications, with numerous advantages accrue from this architecture, both for users and for developers. The more one connects digitally as consumers, the more irritated one can become when websites and apps fail to load within milliseconds. The primary motivation for choosing Next.js is mainly linked to speed and performance.

All React components that comprise a website's user interface are firstly rendered on the server side. This implies that once the HTML is provided to the client, nothing further is required for the user to be able to see the page's content. This gives the user the impression that the website is loading considerably faster. By rendering the same components on the server and client sides, means it may minimise development time by building the React components once and letting Next.js take care of re-rendering them in the user's browser.

Next.js is intelligent, meaning it only needs to load the JavaScript and CSS required for each given page. This results in much quicker page loading speeds since the user's browser is not required to retrieve the JavaScript and CSS that is not relevant to the current page.

Hot Module Replacement enables developers to immediately view changes they make during development live in the application. Unlike typical live reload solutions, it simply reloads the modules that have changed, keeping the application's previous state and decreasing the time necessary to see changes in action.

[Neutkens et al., 2021]

2.1.2 Tailwind CSS

Tailwind CSS comprises a set of utility classes that perform a specific function. This could refer to the colour of the text, the background, the spacing between the lines, or the borders. This strategy enables the developer to create a page using atomic CSS. In the atomic approach to CSS programming, the developer has classes that control a subset of the look.

Reference [Labs Inc., 2022]

2.2 Solidity

Solidity is a programming language for developing smart contracts that run on blockchains such as Ethereum. It is an object-oriented programming language that borrows heavily from JavaScript, Python, and C++. It is written in the C++ programming language and is optimised for use with the Ethereum Virtual Machine (EVM).

Solidity is used to write the contracts that run on the EVM. It is a high-level programming language compatible with the way humans express instructions through numbers and letters rather than binary code. While early Turing machines relied on binary inputs, Solidity eliminates this complexity. It humanises the input process by using more approachable code that is akin to JavaScript in many respects.

Smart contracts in Solidity are composed of instructions that are subsequently compiled to the EVM's bytecode. The nodes in the Ethereum network run EVM instances that enable them to cooperate in implementing the same set of instructions.

Reference [Moralis, 2021]

2.2.1 OpenZeppelin Contracts

Writing Solidity contracts from scratch can introduce a number of serious issues, the most notable of which being long execution times and security risks. This is where OpenZepplin contracts come into use. OpenZepplin is a set of Solidity smart contracts that implements typical smart contract features. For instance, the implementations of the ERC20 and ERC721 and other major token specifications. Additionally, there are a few variants of ICOs and other solidity patterns.

Reference [ope, 2022]

2.2.2 Remix IDE

Remix is an open-source Ethereum integrated development environment (IDE) for writing, compiling, and debugging Solidity code. As such, Remix can be a critical tool in the development of Web3 and DApps.

What is then Remix IDE? The in-browser coding is a unique and rather practical feature of Remix IDE. This is an open-source application that enables the creation of Solidity contracts directly from the browser using JavaScript.

Remix IDE is structured similarly to the most popular programming languages. Three of the most often used modules include ones for testing, debugging, and deploying smart contracts. Additionally, Remix includes various libraries that help accelerate development.

Reference [Aniket et al., 2020]

2.3 HardHat

Hardhat is a development environment for testing, compiling, deploying, and debugging decentralized applications based on the Ethereum network. It enables the developer to manage a large number of the duties associated with developing DApps and smart contracts. Additionally, they provide the developer with the tools necessary to manage this process. Additionally, Hardhat assists in automating several of these tasks.

Hardhat comes with a pre-built local Ethereum network that is optimized for development. This network is dedicated to debugging Solidity and includes stack traces, messages when DApp transactions fail, and more. This is something that aids the developer in determining where or why an application fails during development.

The environment is defined via plugins, from which a large portion of the functionality derives. This means the developer may pick and choose which plugins to include in their development process.

Reference [Foundation, 2021]

2.4 Moralis

Moralis provides backend management for this blockchain project. The Moralis SDK lets the developer construct a decentralized application with user authentication using a cryptocurrency wallet such as MetaMask and blockchain data such as user token balances, NFTs, transactions, and events.

When a user logs into the DApp utilising crypto wallet authentication, their wallet address will be automatically added to the Moralis database and any other data required for the marketplace, such as token balances, past transactions, or events. [Moralis, 2022b]

2.4.1 Web3 API

The Moralis Web3 API allows the developer to retrieve information from the EVM blockchain, including block information, transaction information, NFT metadata, token values, user balances, and the owner list of a particular NFT. This project makes use of the Moralis SDK to access and use the Web3 API and its functions. [Moralis, 2022a]

2.4.2 IPFS

It can be highly costly to store decentralized files and data within ones contracts. This is where one can use IPFS instead. IPFS is a protocol for the decentralized storage of data such as files, applications, and webpages. IPFS's primary advantage is that it eliminates some of the problems associated with centralization, including such as censorship and single point of failure. Moralis has native support for IPFS making the integration even smoother. [Developers, 2021b]

2.5 Visual Studio Code

Visual Studio Code is a desktop-based, lightweight, yet comprehensive source code editor. It includes support for JavaScript and Node.js and has a robust ecosystem of extensions for other languages to make the development more accessible and efficient. [Developers, 2021a]

2.6 GitHub

GitHub is a company that provides a cloud-based hosting solution for Git repositories. Essentially, it makes using Git for version control and collaboration considerably simpler. Version control enables the developer to track and manage modifications to a project's source code. [Docs, 2018]

Chapter 3

Implementation

This chapter will describe:

- Construction and solutions for creating a decentralised application
- Construction and solutions for creating an NFT Marketplace
- Difficulties around smart contracts

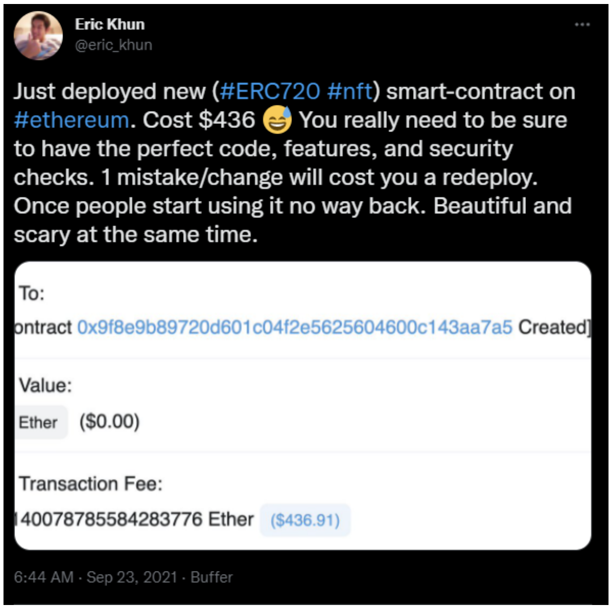
3.1 Base Application

The base of the decentralised application, hereby referred to as DApp, is React and Next.js. When creating a simple React application, one usually uses the `npx create-react-app app-name` command. This creates the directories and files necessary to have a basic react app, and is a good place to start when building the app. Using `npx create-next-app app-name`, to create the base of the application. From there the developer starts installing all the different npm packages needed. The Moralis SDK and ethers.js are two of them.

3.2 HardHat - Local Development environment

Developing smart contracts and testing them in the DApp is a costly procedure, especially if one was to deploy a contract every time changes were made. Doing this to any blockchain costs ERC20 tokens, also known as crypto. Deploying to the Ethereum MainNet blockchain would cost a fair amount of money, largely because the blockchain is so busy that the "gas money" is quite substantial. A software developer called Eric Khun with experience in Web3 development, posted this on twitter September of 2021. Refer to figure 3.1 He paid a total of 436 USD for the deployment of his smart contract, which is not a small amount to pay for ones creation to become accessible to everyone. The explanation of the cost can be separated into four parts.

1. The amount of bytecode in the smart contract increases the cost quickly. The larger the contract is, the more expensive it becomes to deploy. It needs more space on the blockchain to be stored in the first place. Each byte costs 200 gas. The total of bytecode includes the inherited parent contracts one might use to complete ones contract.
2. A flat fee of 32 000 gas for the creation operation, which is called when one creates the contract. Which in turn is on top of 21 000 gas for a normal transaction.
3. All the TX data has a cost. 4 gas for zero bytes and 68 for all non-zero bytes.
4. If ones contract has an extensive constructor, the additional computing cost would be substantial.



Eric Khun
@eric_khun

Just deployed new (#ERC720 #nft) smart-contract on #ethereum. Cost $436 😅 You really need to be sure to have the perfect code, features, and security checks. 1 mistake/change will cost you a redeploy. Once people start using it no way back. Beautiful and scary at the same time.

To:
ontract 0x9f8e9b89720d601c04f2e5625604600c143aa7a5 Created]

Value:
Ether ($0.00)

Transaction Fee:
140078785584283776 Ether ($436.91)

6:44 AM · Sep 23, 2021 · Buffer

Figure 3.1: Eric Khun tweeting about the beauty and horror of deploying a smart contract onto Mainnet. Source: https://twitter.com/eric_khun/status/1440899940453060608?s=20&t=6f1xYRCf1su8CaDULvNmzQ, May 2022

If one googles, GAS to ETH, they would likely find that 1 ETH is equal to around 630 GAS. But GAS and the gas that is referred to here are not the same. Gas used to describe transaction fees within ETH is actually called gwei, and 1 ETH is equal to one billion gwei. Which means that even though the gas numbers mentioned earlier could sound rather expensive, it is at least more sensible than what one would fear.

Now back to the local development environment. Using HardHat one can start a local blockchain that they can test in. HardHat provides the developer with ten different crypto wallets filled with Ethereum, that they can use in their testing. The developer can add these to their wallet to start tesing different aspects of their DApp. Deploying to HardHat is also easily done with a deployment script as shown in the code below.

```
const hre = require("hardhat");

async function main() {
  const Market = await ...
      hre.ethers.getContractFactory("Market");
  const market = await Market.deploy();
  await market.deployed();
  console.log("nftMarket deployed to:", market.address);

  const NFT = await hre.ethers.getContractFactory("NFT");
  const nft = await NFT.deploy(market.address);
  await nft.deployed();
  console.log("NFT deployed to:", nft.address)
}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

Code 3.1: deploy.js code snippet for deploying the smart contracts using HardHat

This is a fairly standard script used by many to deploy their NFT and Market contracts to their local blockchain. One defines the contract factory, and awaits for it to deploy with the variables the particular contract needs to be created. For instance, the NFT contract used needs the Marketplace contracts address on the blockchain before it can be constructed. After having been deployed, one gets their addresses logged in the console, ready for use in the development of this DApp.

3.2.1 Final Deployment of smart contracts

When developing a DApp there are lots of different blockchains one could deploy the smart contracts to. Some "blockchains" such as Polygon are layer two solutions that work atop of the Ethereum platform, which gives them benefits such as being less expensive and faster but binds them to the platform which makes them vulnerable to the same downtime and faults the original platform may experience. There are also other blockchains for instance, Avalanche and Fantom, which are powered by its respective native tokens. However, since the marketplace was developed with Ethereum in mind, the Ethereum testnets made most sense to consider. Currently there are two official Ethereum testnets, Sepolia which is a proof-of-work testnet and Görli which is a proof-of-authority testnet. Unfortunatly they are both pretty fresh of the press, which makes it hard getting ETH tokens to deploy and interact with it but also there is a lack of resources and guides surrounding them. For this thesis then, out of the three deprecated testnets, Rinkeby was selected as the TestNet of choice. It is solid, works well and has an established community around it making it easier to develop on.

3.3 Minting an NFT

It was decided that users should have the ability to mint NFTs on the site. This feature makes it relatively easy to incorporate their NFTs into the marketplace infrastructure and functionality.

3.3.1 Front-End

Beginning with the front-end aspect of the minting process. Technically, one could upload any file and mint an non-fungible token. But this marketplace, like many others, only allows images to be minted. Further setting up a simple but clean form using Tailwind CSS, with a file upload that showcases the image, photo and/or art one wants to make into a token. The file upload restricts one from uploading anything that is not an image. JPEG, PNG, WEBP and GIFs are allowed on the site.

3.3.2 NFT - Smart Contract

The smart contract for the NFT is relatively simple yet effective. See attachment A.

The contract uses Counters from “Counters.Counter”. Counters are a very effective way of handling ids in a contract. It has built-in functionality for increments, decrements, showing the current value and resetting altogether, which helps keep an accurate count on how many tokens this contract has and what the subsequent tokens id must be.

The contract demands an address to deploy and create the contract on the blockchain. This address is the marketplace smart contracts address, which are set up as the operator. Next up is the `mintNFT();` function. This function retrieves the id for this token and mints the token using the message sender and the new id. The message sender is whoever called the mint function on the site. The tokens URI is set with its id so that all the metadata is connected to the corresponding token. The marketplace then has to be approved, which means that the marketplace contract will be able to execute transactions on the owners behalf.

When a user buys another users token, the marketplace will control the transaction on the users behalf. Since the site focuses on NFTs that has been minted on the site, it will not handle tokens that have been minted by the user somewhere else. It would be possible to let tokens from other marketplaces and non-fungible token smart contracts be exchanged on the DApp, as long as they reside on the same blockchain as this sites smart contracts are deployed. It was decided to not go down that route since that would require a rewrite of how the contracts and website operates on base level. See chapter 4.5

All NFTs minted on the DApp becomes part of a collection. The collection has the name of "Bachelor T. Tokens", and will be shown as part of that collection on sites like OpenSea. If the user would want to have their token as part of another collection, they would have to transfer the token between the smart contracts or mint it on that contract. Essentially, a smart contract which mints non-fungible tokens is the collection that token is a part of.

3.3.3 Moralis decentralised database and IPFS

Since decentralisation is the whole concept, the images and data must be saved in a decentralised matter. A web server with a database is the standard in the web development industry, but for a DApp the project needs IPFS and a decentralised database. This is where Moralis comes into the picture, reference to chapter 2.4. Moralis hosts servers connected to a blockchain of choice to save metadata and data related to ones DApp, aswell as hosting an IPFS service to save files onto.

When the user presses the "Mint NFT" button on the webpage, the image is created into a Moralis File object with the `.saveIPFS();` function which reads the base64 data of the image and uploads it to Moralis IPFS service. Using the `.ipfs();` function on the same object afterwards will return the hash for the file. This is then used in the metadata object, since this is where the DApp will need to fetch the image from to show it on the site. Saving the metadata is done with the exact same method. A metadata object is created, made into a Moralis File object and sent off to Moralis IPFS. The URI is saved on the blockchain as the tokens URI. This makes the reference point consistent, and one can not edit the metadata easily without direct access to the IPFS service and its files.

When this is happening, a record of the NFT is also saved in a database table called "MintedNFTs" on the DApps Moralis database. This makes querying for NFTs relatively simple to do. The homepage only queries the tables for the information needed since it contains everything that it needs.

3.4 Marketplace core structure

The marketplace has its own smart contract which handles, listing, buying and selling non-fungible tokens on the DApp. To see the source code for the marketplace smart contract, see attachment B

3.4.1 Listing the NFT

When the user mints the NFT, it is also added to the marketplace smart contract. That is why there are two transactions to be approved by the user when one mints a token. This allows the DApp to have control over the token and know the status of it. This is because the contract can contain custom variables holding information, like if a token is for sale.

If the user then wants to list the non-fungible token for sale on the marketplace, they would have to view the token and press `Sell`. This opens an input box where the user sets the price they want to sell the token for. They then confirm that they want to sell the token, and it is set as for sale on the contract. The token is transferred to the marketplace contract, the seller variable is set to the current owner, owner is set to a null address, the price is set to the desired value and it is set as `for sale`. The owner could then view the same token and cancel the listing by pressing the `Cancel Sale` button on the page. This sets the price back to 0, transfers the token back to the owner and puts it as not for sale. The seller and owner variables are also reset back to what they were before it was put up for sale.

3.4.2 Buying an NFT

Another user can view this token whenever they want with the name, description and image shown to them. When a user that is neither the owner nor the seller views the token while it is up for sale, they will have the option to buy the token for the price that the owner has specified. If this occurs, the market smart contract executes the `createMarketSale();` function. The function transfers the specified price from the buyer to the seller, transfers the token from the smart contract to the buyer and sets all relevant variables to their respective new value. The counter for items currently for sale, is also reduced. The user can now find the token in their inventory. This can be verified at sites like Etherscan and testnets.OpenSea.io.

3.4.3 Querying the non-fungible tokens

The querying of tokens is done by two different methods, depending on where the user travels on the DApp.

Moralis Cloud Functions

Moralis Cloud Functions is used to query the two tables; "ListedNFTs" and "MintedNFTs", which then get joined to create a more complete picture of the tokens. This is slightly quicker than querying the smart contract for information on every page load. The Moralis tables are also updated when certain actions like buying and listing occurs on the DApp. By having the Moralis cloud server run the code, the end-users devices won't need to use computing power for those functions. It is also handy as it cleans up the source code, especially when the function is used multiple times across several pages.

Pages that use the Moralis cloud method to gather data concerning the tokens are the index page, dashboard and feed.

Source code can be referenced in attachment C.

Smart Contract

The dynamic pages which presents a specific token, query the marketplace smart contract. This is because it is where all the interaction with the contracts happen. It is therefore extra important that all information regarding the token is up to date and correct. Moralis might have had a bad sync which results in outdated information regarding the token. Outdated token information is not as big of a deal on the front page since users would end up seeing the current data, if they were to view the token.

The access control checks that is done on the current viewing user needs to be precise to show the correct functions. A random user should not be able to press the `Sell` button, regardless of the internal access control in the smart contract.

3.4.4 Rich description

One of the key features of the DApp is the ability to write a rich description. Using the npm package called `react-md-editor`, the users are able to write exactly the description they want. With everything from titles, font styles like bold and italic, lists and images. The description is saved as a string with everything the preview component needed to display the description in the correct manner. Other marketplaces with markdown enabled will have the ability to show the tokens description as the user wrote it.

Chapter 4

Discussion and future improvements

With all the developments around Ethereum 2.0 and new standards for fungible and non-fungible tokens alike, there are certainly future improvements to be done on this marketplace.

4.1 Royalties

One of the talking points about how NFTs can positively change the art industry is how it handles royalties. An artist can mint their art, display it digitally and sell it to practically anyone; like Beeple who sold his NFT for 69 million dollars, referenced in chapter 1.1. If he has set a royalties rate on this NFT he could earn around 5 - 10% of what the NFT gets sold for next time. This has to be implemented of course, and sites like OpenSea does this within its own marketplace. Essentially the artist can choose their royalty rate when they put the token up for sale. There has been developed a standard for this called EIP-2981.

"A standardized way to retrieve royalty payment information for non-fungible tokens (NFTs) to enable universal support for royalty payments across all NFT marketplaces and ecosystem participants." [Burks et al., 2020]

This helps artists get royalties no matter which marketplace their non-fungible tokens gets sold on next. Even though this standard was published back in September of 2020, it has not been enforced enough. Plenty of marketplaces use their own royalty payment method, which makes getting the artist what they deserve quite hard at times.

4.2 Rework the approval process

As mentioned earlier, it was decided to simplify the selling process using two contracts. The NFT contract handles minting and only minting of the contract, meanwhile the Marketplace contract has control over what is on the site and how transactions are carried out. When minting, the user grants the Marketplace contract the freedom to handle transaction on the behalf of the owner. This is a misstep. One could grant the approval per user basis, which would be a lot safer. The buyer would then have to "ask" the owner for permission to buy it, the owner would grant the permission and the buyer could then execute the transaction. By using this method, one could actually run the marketplace with only one contract. Everything could be handled by the NFT contract alone.

4.3 ERC721 vs ERC1155

ERC721 is the golden standard for non-fungible tokens as it stands today, however ERC1155 is constantly gaining ground. ERC1155 is the big brother, the successor to ERC721, building on all the previous standards like ERC20, ERC721 and ERC777. This is a novel token standard with the ability to represent fungible and non-fungible tokens, with the added bonus of greater gas efficiency. Moving to this standard with EIP-2981 implemented would be a great choice for a future rework of the core structure of the marketplace.

One thing that is for certain, is the fact that blockchain technology is not standing still. It is constantly moving, developing and figuring out ways to become better, safer and more efficient.

4.4 Image conversion

A problem using IPFS services to store metadata and images, is the fact that no matter how fast ones computer and internet connection is, one is restricted by the IPFS. Larger images can be very slow to download and preview on the site, especially if the IPFS service is sort of busy. A way to sort this out may be the use of the new WebP format.

"WebP is a modern image format that provides superior lossless and lossy compression for images on the web."[Google, 2010]

"WebP lossless images are 26% smaller in size compared to PNGs. WebP lossy images are 25-34% smaller than comparable JPEG images at equivalent SSIM quality index." [Google, 2010]

Saving space, faster loading and fast upload could make the app feeling more responsive and better to use. It could be used for JPEG, PNG and GIFs. It however has a drawback when it comes to animation. WebP needs more processing power to show the animation than what a comparable GIF demands. Which is one of the reasons why WebP has not taken the web by storm yet.

4.5 From closed to open marketplace

The marketplace as it stands is a closed market. Only non-fungible tokens minted on the site has the opportunity to be exchanged and listed on the market. The move from closed to open would require rewrite of the smart contract and DApp logic. If the DApp was open, the users would see all of their non-fungible tokens on the Rinkeby blockchain and could select which ones they wanted to list on the marketplace. The logic would have to handle duplicate token ids, getting approval from the user to complete the transaction on the users behalf and talking to the correct token contract.

Chapter 5

Conclusion

A decentralised application was developed in this thesis. This app was created with React and Next.js. The purpose for this NFT marketplace was to create a responsive and modern online interface that would allow users to login, upload, mint, buy, and sell non-fungible tokens. Hardhat was used to design and test the solidity smart contracts before publishing them on the Ethereum Testnet.

The application is presently hosted on Vercel [Hos, 2022] in the Rinkeby network and is available for public testing and minting. Login to the site with ones MetaMask wallet, which is also linked to the Rinkeby network. Only non-fungible tokens created on the website can be exchanged and published on the market. However, the tokens minted on the site can also be viewed via OpenSea.

Bibliography

[Hos, 2022] (2022). Nft marketplace hosted on vercel. https://nft-marketplace-bachelor-thesis.vercel.app/.

[ope, 2022] (2022). Openzeppelin smart contract documentation. https://docs.openzeppelin.com/.

[Aniket et al., 2020] Aniket, Raj, P., and Rob (2020). Welcome to remix's documentation! https://remix-ide.readthedocs.io/.

[Burks et al., 2020] Burks, Z., Morgan, J., Malone, B., and Seibel, J. (2020). Eip-2981: Nft royalty standard. https://eips.ethereum.org/EIPS/eip-2981.

[Christie's, 2021] Christie's (2021). Beeple's masterwork: The first purely digital artwork offered at christie's: Christie's. https://www.christies.com/features/Monumental-collage-by-Beeple-is-first-purely-digital-artwork-NFT-to-come-to-auction aspx.

[Developers, 2021a] Developers, M. (2021a). Documentation for visual studio code. https://code.visualstudio.com/docs.

[Developers, 2021b] Developers, P. L. (2021b). Welcome to the ipfs docs. https://docs.ipfs.io/.

[Docs, 2018] Docs, G. (2018). Learning about github. https://docs.github.com/en/get-started/learning-about-github.

[Foundation, 2021] Foundation, N. (2021). Ethereum development environment for professionals by nomic foundation.

[Google, 2010] Google (2010). An image format for the web | webp | google developers. https://developers.google.com/speed/webp.

[Labs Inc., 2022] Labs Inc., T. (2022). Installation: Tailwind cli - tailwind css. https://tailwindcss.com/docs.

[Marcobello, 2022] Marcobello, M. (2022). Cryptopunks, cryptocats and cryptokitties: How they started and how they're going. https://www.coindesk.com/learn/cryptopunks-cryptocats-and-cryptokitties-how-they-started-and-how-theyre-going/.

[Moralis, 2021] Moralis (2021). Solidity explained - what is solidity? " moralis " the ultimate web3 development platform. https://moralis.io/solidity-explained-what-is-solidity.

[Moralis, 2022a] Moralis (2022a). Moralis web3 api. https://docs.moralis.io/moralis-dapp/web3-api.

[Moralis, 2022b] Moralis (2022b). What is moralis? https://docs.moralis.io/.

[Neutkens et al., 2021] Neutkens, T., Kanezawa, N., Rauch, G., Susiripala, A., Kovanen, T., and Zajdband, D. (2021). Next.js documentation. https://nextjs.org/docs/basic-features/pages.

[Platforms, 2022] Platforms, M. (2022). Getting started. https://reactjs.org/docs/getting-started.html.

[Publishing, 2021] Publishing, B. (2021). Blockchain: Global markets. https://www.bccresearch.com/market-research/information-technology/global-blockchain-market.html.

Attachment A

Solidity smart contact

A.1 Minter.sol - NFT smart contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import ...
    "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract Minter is ERC721URIStorage {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;
    address contractAddress;

    constructor(address marketplaceAddress) ...
        ERC721("Bachelor T. Tokens", "BTT") {
        contractAddress = marketplaceAddress;
    }

    function mintNFT (string memory tokenURI) public ...
        returns (uint) {
        _tokenIds.increment();
        uint256 newItemId = _tokenIds.current();

        _safeMint(msg.sender, newItemId);
        _setTokenURI(newItemId, tokenURI);
        setApprovalForAll(contractAddress, true);
        return newItemId;
    }
}
```

Attachment B

Solidity smart contact

B.1 MarketplaceHandler.sol - Marketplace smart contract

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import ...
    "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract MarketplaceHandler is ReentrancyGuard {
    using Counters for Counters.Counter;
    Counters.Counter private _itemIds;
    Counters.Counter private _itemsSold;
    Counters.Counter private _itemsForSale;

    address payable owner;
    uint256 listingPrice = 0.005 ether;

    constructor() {
        owner = payable(msg.sender);
    }

    struct MarketItem {
        uint itemId;
        address nftContract;
        uint256 tokenId;
        address payable creator;
        address payable seller;
        address payable owner;
        uint256 price;
        bool forSale;
    }

    mapping(uint256 => MarketItem) private idToMarketItem;

    event MarketItemCreated (
        uint indexed itemId,
        address indexed nftContract,
        uint256 indexed tokenId,
        address seller,
        address owner,
        uint256 price,
        bool forSale
    );

    event NFTListed (
        uint indexed itemId,
        address indexed nftContract,
        uint256 indexed tokenId,
        address creator,
        address seller,
        address owner,
        bool forSale
    );

    event Transfer (
        uint indexed itemId,
        address indexed nftContract,
        uint256 indexed tokenId,
        address sender,
        address owner,
        bool isApproved
    );

    function getListingPrice() public view returns ...
       (uint256) {
        return listingPrice;
    }

    function listNFT(
        address nftContract,
        uint256 tokenId
        ) public {
```

```
        _itemIds.increment();
        uint256 itemId = _itemIds.current();

        idToMarketItem[itemId] = MarketItem(
            itemId,
            nftContract,
            tokenId,
            payable(msg.sender),
            payable(address(0)),
            payable(msg.sender),
            0,
            false
        );
    }

    function createMarketItem(
        address nftContract,
        uint256 tokenId,
        uint256 price,
        uint256 itemId
    ) public payable nonReentrant {
        require(price > 0, "Price must be at least 1 wei");
        require(msg.value == listingPrice, "Price must ...
           me equal to listing price");

        _itemsForSale.increment();
        idToMarketItem[itemId].forSale = true;
        idToMarketItem[itemId].seller = ...
           payable(msg.sender);
        idToMarketItem[itemId].owner = payable(address(0));
        idToMarketItem[itemId].price = price;

        IERC721(nftContract).transferFrom(msg.sender, ...
           address(this), tokenId);

        emit MarketItemCreated(
            itemId,
            nftContract,
            tokenId,
            msg.sender,
            address(0),
            price,
            true
        );
    }

    function createMarketSale(
        address nftContract,
        uint256 itemId
```

```
      ) public payable nonReentrant {
      uint price = idToMarketItem[itemId].price;
      uint tokenId = idToMarketItem[itemId].tokenId;
      bool approved = ...
        IERC721(nftContract).isApprovedForAll(msg.sender, ...
        address(this));
      require(msg.value == price, "Please submit the ...
        asking price in order to complete the ...
        purchase");

      emit Transfer(
        itemId,
        nftContract,
        tokenId,
        msg.sender,
        idToMarketItem[itemId].owner,
        approved
      );

      idToMarketItem[itemId].seller.transfer(msg.value);
      IERC721(nftContract).approve(msg.sender, tokenId);
      IERC721(nftContract).transferFrom(address(this), ...
        msg.sender, tokenId);
      idToMarketItem[itemId].owner = payable(msg.sender);
      idToMarketItem[itemId].seller = ...
        payable(address(0));
      idToMarketItem[itemId].forSale = false;
      _itemsForSale.decrement();
      payable(owner).transfer(listingPrice);



    }

    function cancelMarketSale(
      address nftContract,
      uint256 tokenId,
      uint256 itemId
      ) public payable nonReentrant {

        _itemsForSale.decrement();
        idToMarketItem[itemId].forSale = false;
        idToMarketItem[itemId].seller = ...
          payable(address(0));
        idToMarketItem[itemId].price = 0;
        idToMarketItem[itemId].owner = ...
          payable(msg.sender);

```

```
            IERC721(nftContract).transferFrom(address(this), ...
               msg.sender, tokenId);
        }

    function fetchMarketItems() public view returns ...
       (MarketItem[] memory) {
        uint totalItemCount = _itemIds.current();
        uint unsoldItemCount = _itemsForSale.current();
        uint currentIndex = 0;

        MarketItem[] memory items = new ...
           MarketItem[](unsoldItemCount);
        for (uint i = 0; i < totalItemCount; i++) {
            if (idToMarketItem[i + 1]. owner == ...
               address(0) && idToMarketItem[i + ...
               1].forSale == true) {
                uint currentId = idToMarketItem[i + ...
                   1].itemId;
                MarketItem storage currentItem = ...
                   idToMarketItem[currentId];
                items[currentIndex] = currentItem;
                currentIndex += 1;
            }
        }
        return items;
    }

    function fetchItemIds() public view returns (uint) {
        return _itemIds.current();
    }

    function fetchMyNFTs() public view returns ...
       (MarketItem[] memory) {
        uint totalItemCount = _itemIds.current();
        uint itemCount = 0;
        uint currentIndex = 0;

        for (uint i = 0; i < totalItemCount; i++) {
            if (idToMarketItem[i + 1]. owner == ...
               msg.sender) {
                itemCount += 1;
            }
        }

        MarketItem[] memory items = new ...
           MarketItem[](itemCount);
        for (uint i = 0; i < totalItemCount; i++) {
            if (idToMarketItem[i + 1]. owner == ...
               msg.sender) {
                uint currentId = idToMarketItem[i + ...
                    1].itemId;
                MarketItem storage currentItem = ...
                    idToMarketItem[currentId];
                items[currentIndex] = currentItem;
                currentIndex += 1;
            }
        }
        return items;
    }

    function fetchItemsCreated() public view returns ...
        (MarketItem[] memory) {
        uint totalItemCount = _itemIds.current();
        uint itemCount = 0;
        uint currentIndex = 0;

        for (uint i = 0; i < totalItemCount; i++) {
            if (idToMarketItem[i + 1].creator == ...
                msg.sender) {
                itemCount += 1;
            }
        }

        MarketItem[] memory items = new ...
            MarketItem[](itemCount);
        for (uint i = 0; i < totalItemCount; i++) {
            if (idToMarketItem[i + 1].creator == ...
                msg.sender) {
                uint currentId = idToMarketItem[i + ...
                    1].itemId;
                MarketItem storage currentItem = ...
                    idToMarketItem[currentId];
                items[currentIndex] = currentItem;
                items[currentIndex].itemId = currentId;
                currentIndex += 1;
            }
        }
        return items;
    }

    function fetchNFT(
        uint256 id
    ) public view returns (MarketItem[] memory) {
        uint totalItemCount = _itemIds.current();

        MarketItem[] memory item = new MarketItem[](1);
        for (uint i = 0; i < totalItemCount; i++) {
            if (idToMarketItem[i + 1].itemId == id) {
                uint currentId = idToMarketItem[i + ...
                    1].itemId;
                MarketItem storage currentItem = ...
                    idToMarketItem[currentId];
                item[0] = currentItem;
                item[0].itemId = currentId;
            }
        }
        return item;
    }
}
```

Attachment C

Moralis Cloud Functions

C.1 Moralis Cloud Functions for querying and joining table records

```
Moralis.Cloud.define("listedAndSoldNFTs", async ...
    (request) => {
  const query = new Moralis.Query("ListedNFTs");
  query.descending("createdAt");
  query.limit(10);
  const list = await query.find();

  const query2 = new Moralis.Query("MintedNFTs");
  query2.descending("createdAt");
  let mints = await query2.find();
  let listed = [];
  let sold = [];

  for (let i = 0; i < list.length; i++) {
    if (list[i].attributes.Sold === 0) {
      for (let j = 0; j < mints.length; j++) {
        if (list[i].attributes.TokenID === ...
            mints[j].attributes.tokenId) {
          const nft = {
            tokenId: list[i].attributes.TokenID,
            price: list[i].attributes.Price.toString(),
            createdAt: list[i].attributes.createdAt,
            image: mints[j].attributes.image,
            name: mints[j].attributes.name
          }
          listed.push(nft)
        }
      }
    }
  }

  for (let i = 0; i < list.length; i++) {
    if (list[i].attributes.Sold === 1) {
      for (let j = 0; j < mints.length; j++) {
        if (list[i].attributes.TokenID === ...
           mints[j].attributes.tokenId) {
          const nft = {
            tokenId: list[i].attributes.TokenID,
            price: list[i].attributes.Price.toString(),
            createdAt: list[i].attributes.createdAt,
            image: mints[j].attributes.image,
            name: mints[j].attributes.name
          }
          sold.push(nft)
        }
      }
    }
  }

  return [listed, sold]
});

Moralis.Cloud.define("createdNFTs", async (request) => {
  const query = new Moralis.Query("ListedNFTs");
  query.descending("createdAt");
  const list = await query.find();

  const query2 = new Moralis.Query("MintedNFTs");
  query2.descending("createdAt");
  let mints = await query2.find();
  let created = [];

  for (let i = 0; i < mints.length; i++) {
    if (mints[i].attributes.creator == ...
       request.params.acc) {
      for (let j = 0; j < list.length; j++) {
        if (mints[i].attributes.TokenID === ...
           list[j].attributes.tokenId) {
          const nft = {
            tokenId: list[j].attributes.TokenID,
            price: list[j].attributes.Price.toString(),
            createdAt: mints[i].attributes.createdAt,
            image: mints[i].attributes.image,
            name: mints[i].attributes.name,
          }
          created.push(nft)
        }
      }
    }
  }

  return created
});
```