Decentralised Index Funds

Luke Kirwan
University of Bristol
jz21126@bristol.ac.uk

Prajwal Kulkarni
University of Bristol
ez21956@bristol.ac.uk

Cameron McEwan
University of Bristol
cm17521@bristol.ac.uk

Yingtong Qi
University of Bristol
kk21812@bristol.ac.uk

Alexander Straw
University of Bristol
as17163@bristol.ac.uk

I. Introduction

Crypto is booming. In April 2017, the size of the market was $40 billion. Fast forward five years and the current market stands at over $2 trillion [1]. This impressive growth, coupled with the potential use cases of crypto, has brought many new retail investors to the market. However, they have a problem. Investing in crypto is not simple and is very risky, due to the high volatility of the market. The natural response from investors is to diversify, by obtaining a variety of different crypto assets, but this is not easy. Firstly, an investor will incur separate transaction fees for every asset they want to purchase. Secondly, tracking ownership of these assets becomes increasingly difficult the more assets an investor owns. Thirdly, it is a time-consuming and a highly skilled task to curate a profitable portfolio. In traditional finance, the proven method for dealing with these three problems is to invest in an index fund, which allows an investor to gain exposure to multiple assets in a single transaction. However, for investors in crypto, few such products exist. This is where the Folio platform comes in.

Folio is the blockchain equivalent of an exchange traded index fund (ETF), intending to give investors easy access to low risk and low cost investment options in the crypto space. With Folio, an investor can create their own portfolio of cryptoassets, or buy into an existing portfolio created by other users of the platform. There is only one transaction, one fee, and one asset to track (the portfolio itself). What's more, an investor can redeem their holdings in a portfolio and receive the underlying assets. Also, this product serves the unbanked, who may not have the credentials to access traditional index funds. Finally, it is possible to see the ownership breakdown of a portfolio thanks to the transparency of the blockchain. These features highlight some benefits of decentralisation and the edge of Folio over traditional index funds.

There are similar products already established in the market. Firstly, Crypto20 tracks the top twenty largest cryptocurrencies by market capitalisation [2]. This product has performed exceptionally well, however it breaks the principle of decentralisation by relying on an off-chain (centralised) rebalancing mechanism. Secondly, there are the products offered by Set Protocol, such as the DeFi-Pulse index and the Metaverse index [3]. These are properly decentralised and have gained significant traction since their inception. For example, the DeFi-Pulse index, introduced as recently as September 2020, has a total market capitalisation of over $88 billion. In fact, the DeFi industry as a whole has grown from $72 billion to over $100 billion in the past year [4]. The explosion of DeFi and the growth of Set's products shows that investors are flocking to this new market.

II. Design Documentation

A. Front end

Every element of the website design should be simple and intuitive. The purpose of the website will be to support the user in the use cases identified in Figure 1.

The dapp's main functionalities should be displayed across two interactive pages: Explore and Create. The flow chart in Figure 2 shows how our users should interact with the front end. The user will start at the Landing Page, where they will be able to navigate to both the Explore and Create pages with emphasised buttons. The user should also be able to navigate between the Landing, Explore and Create pages at any point via a sticky navigation bar. When the user is on either the Explore or Create page, it must be simple to follow the steps required to complete the relevant activities. The tasks on the Create page should be broken down into small steps, simplifying the process of creating a portfolio.

The purpose of the Explore page will be to browse existing portfolios. When a portfolio is clicked, additional information

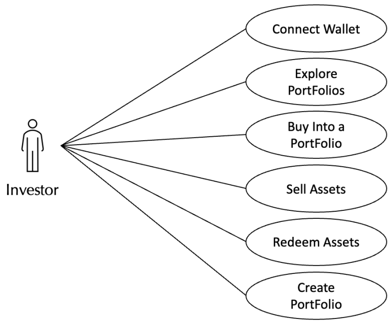


Fig. 1. Use case diagram showing how our user can interact with the dapp.

TABLE I
DEFAULT STATE AND METHODS (FROM ERC20) FOR PORTFOLIO.SOL SMART CONTRACT.

State	**Description**
name	Name of the Folio ERC20 token.
symbol	Symbol or 'ticker' for the Folio ERC20 token.
totalSupply	Total supply of Folio ERC20 token. Each portfolio has a unique supply defined by the amount initialised.
Method	**Description**
mint	Creates new Folio ERC20 tokens and allocates them to a specific address.
burn	'Deletes/Destroys' some Folio ERC20 tokens by removing them from the supply.
transfer	Transfers Folio ERC20 tokens to another address.

TABLE II
CUSTOM STATE AND METHODS FOR PORTFOLIO.SOL SMART CONTRACT.

State	**Description**
tokenAddresses	Contract addresses of the assets (ERC20 tokens) held in the portfolio.
percentageHoldings	Percentage holding for each asset, corresponding to the address in tokenAddresses.
assetQuantities	Mapping of asset address to the quantity of that asset held in the portfolio.
owner	Address of the owner of the portfolio (the address that deployed it).
ownerFee	The amount (in basis points, so a value of 1 = 0.01%) to transfer to the owner every time someone buys into the portfolio.
Method	**Description**
initialisePortfolio	Called by the owner once the contract has been deployed to initialise the portfolio. Payable function which takes Eth and initialises the portfolio with the correct amount of the specified tokens.
buy	Payable function which enables the user to buy into the portfolio with Ether.
sell	Returns the amount of Eth obtained from the sale of a specified number of Folio tokens.
redeemAssets	Returns the underlying assets corresponding to a specified number of Folio tokens.

should be displayed along with buttons to buy, sell and redeem tokens of the portfolio, which must call the respective functions in the back end.

The user will need to be connected to their MetaMask wallet to interact with any portfolios, therefore the Connect To MetaMask button should be placed in the navbar and styled with a red fill and glow effect to make it clear that this is a call-to-action button. All components throughout the website should be styled consistently, including hover effects for the main buttons.

B. Back end

Excess complexity when designing blockchain systems results in expensive gas fees and unnecessary security risks. Therefore, efficiency and simplicity should be at the core of all development decisions made when writing these smart contracts. From a high level there should be careful consideration regarding: overall system architecture, user interaction with the contracts; and how aspects of the contract could be manipulated by bad actors. In addition, incentive mechanisms should be added to encourage the creation of high-quality portfolios, such as a fee paid to the owner whenever an individual buys into their portfolio.

A flow-chart illustrating the life cycle of a portfolio is shown in Figure 3. The dapp should have two distinct activities that a user can engage in: the creation of portfolios and interaction with already created portfolios.

When creating a portfolio, the user should be able to select: a name, symbol, the assets the portfolio will hold, the distribution of those assets, and a fee for the owner (see Table I, and Table II for further details). Once created, the owner will then initialise the portfolio with Ether, and in return they will receive an arbitrary quantity of the portfolio's native ERC20 token. The reason this is arbitrary is because the token has no value prior to being initialised.

To invest in a portfolio, a user must call the *buy* function, and transfers some Ether to the portfolio. The portfolio will then use that Ether to buy the underlying assets on an exchange, and subsequently issue native portfolio tokens to the user, which will represent their share of assets in the portfolio. The best exchange to use for purchasing these assets is Uniswap, due it being the most liquid decentralised exchange (DEX) on Ethereum. To calculate the number of tokens to mint for the user, first the prior-value-locked (PVL) should be found using Equation 1, making use of the returned asset quantities from Uniswap (Chainlink oracles should be used here in future). Next, the contract should estimate the fair price of the native portfolio token using the total supply and Equation 2. The reverse process: selling assets, or redeeming assets, should use the known quantities of each asset held by the portfolio, and the quantity of tokens that the user wishes to sell, to calculate the quantity of assets that they are entitled to (see Equation 3). The user should have two choices when selling, either to redeem the assets directly, or to automatically swap these on Uniswap to receive their equivalent value in Ether.

$$PVL = \sum_{i=0}^{n} \frac{wethAmount * assetQuantities[asset_i]}{NumTokensOut_i} \tag{1}$$

$$N = \frac{msg.value * Portfolio.totalSupply}{PVL} \tag{2}$$

$$quantityAssetToSell_i = \frac{assetQuantities[asset_i] * numTokensSold}{Portfolio.totalSupply} \tag{3}$$

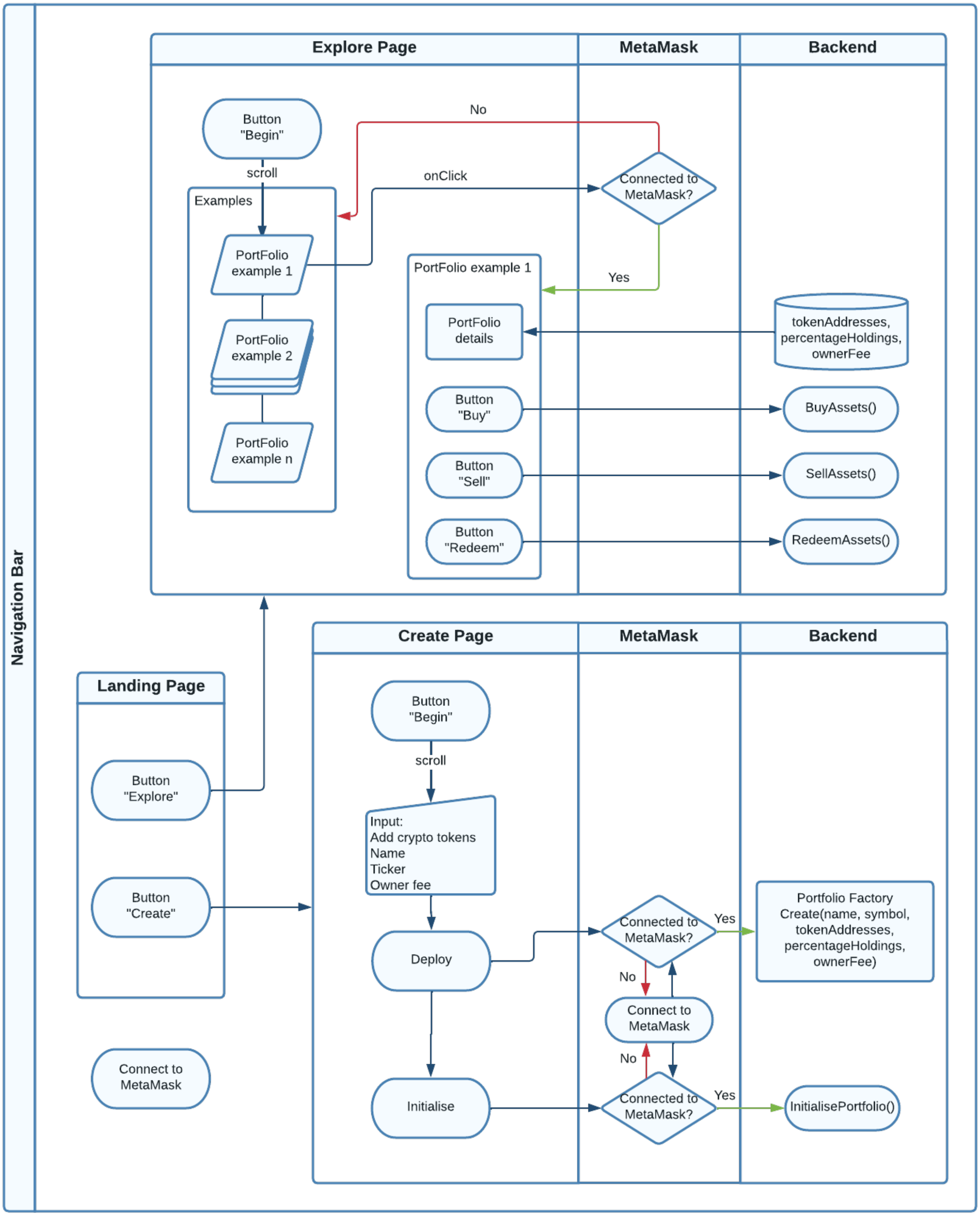
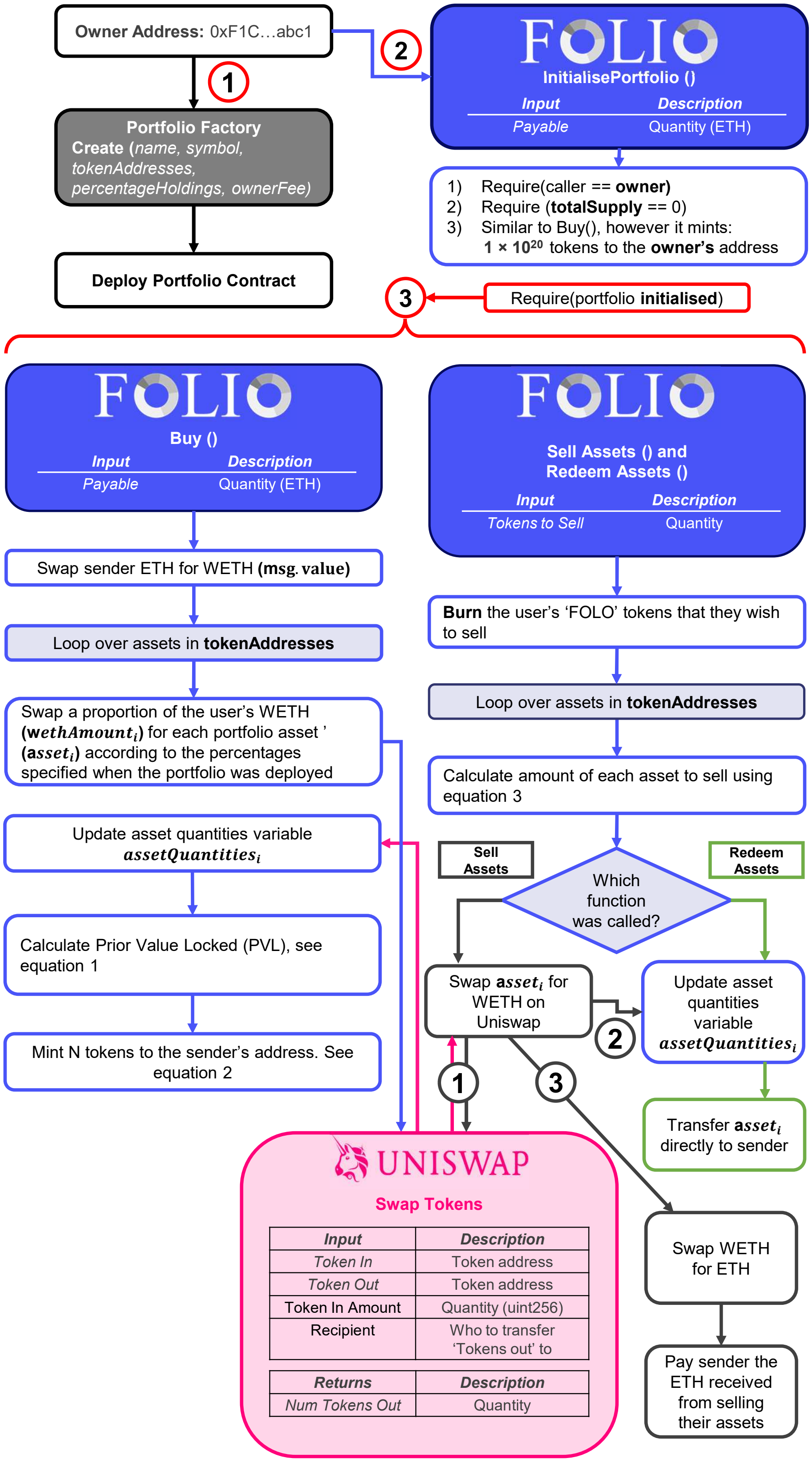


Fig. 2. Full user interface flow chart showing main interactive components on the front end and calls to the back end.


Owner Address: 0xF1C…abc1
1
Portfolio Factory
Create (name, symbol, tokenAddresses, percentageHoldings, ownerFee)
Deploy Portfolio Contract
2
FOLIO
InitialisePortfolio ()
Input | Description
Payable | Quantity (ETH)
1) Require(caller == owner)
2) Require (totalSupply == 0)
3) Similar to Buy(), however it mints: 1×10^{20} tokens to the owner's address
3
Require(portfolio initialised)
FOLIO
Buy ()
Input | Description
Payable | Quantity (ETH)
Swap sender ETH for WETH ($\mathbf{msg.value}$)
Loop over assets in tokenAddresses
Swap a proportion of the user's WETH ($\boldsymbol{wethAmount_i}$) for each portfolio asset ' ($\boldsymbol{asset_i}$) according to the percentages specified when the portfolio was deployed
Update asset quantities variable $\boldsymbol{assetQuantities_i}$
Calculate Prior Value Locked (PVL), see equation 1
Mint N tokens to the sender's address. See equation 2
FOLIO
Sell Assets () and Redeem Assets ()
Input | Description
Tokens to Sell | Quantity
Burn the user's 'FOLO' tokens that they wish to sell
Loop over assets in tokenAddresses
Calculate amount of each asset to sell using equation 3
Sell Assets
Redeem Assets
Which function was called?
Swap $\boldsymbol{asset_i}$ for WETH on Uniswap
2
Update asset quantities variable $\boldsymbol{assetQuantities_i}$
1
3
Transfer $\boldsymbol{asset_i}$ directly to sender
Swap WETH for ETH
Pay sender the ETH received from selling their assets
UNISWAP
Swap Tokens

Input	Description
Token In	Token address
Token Out	Token address
Token In Amount	Quantity (uint256)
Recipient	Who to transfer 'Tokens out' to

Returns	Description
Num Tokens Out	Quantity

Fig. 3. Full flow chart for *Portfolio.sol* life-cycle: deployment, initialisation, and user interaction.

III. Code Analysis

A. Front end

This section discusses the front-end code located on Github [5]. We decided to use the React framework for building the front-end. There are a few reasons for this. Firstly, it was the framework we were most familiar with, having been taught it on the course. Secondly, React's component-centric nature makes the interface very responsive to user interaction. When a user interacts with the interface, only the relevant component is re-rendered, rather than the entire app. Also, React apps run entirely on the client's browser, so there is no need for a central web server. This is especially important for dapps, which use the Ethereum blockchain as their back-end.

CSS, SCSS and MaterialUI components were utilized to ensure consistent styling throughout the user interface. The echarts library was used to create the interactive pie chart component, *Pie.js*, which has been reused throughout the dapp. React-router-dom enabled us to host multiple pages in our app and route to them via React components. The React *useRef()* hook was used to scroll between React section components on the Explore and Create pages, which is activated when Begin, Back and Continue buttons are clicked.

The search list in step 1 on the Create page also uses React hooks. Firstly, we retrieve the list of approved tokens from an external file and read them into a React state variable called *searchList*. Then, we have a *useEffect* hook that watches for changes to the *tokenSearchText* variable. When this variable changes, the token list is filtered, and the variable *searchList* is updated with the filtered list. The component then re-renders because a state variable has updated, and so the user sees the new filtered list presented to them.

We used Ethers for interacting with our smart contracts. In *src/components/PortfolioDetail.js* we implement three functions for interacting with the smart contracts. The *buy* method uses the React state variable *portfolioContract* (which has already been initialised with an Ethers instance of our smart contract *Portfolio.sol*) to call the *buy* function with a custom *msg.value*. We use an *await* to wait for the transaction to complete, and then update the users balance which is presented to them on screen.

B. Back-end Code Analysis

This section explores the back-end code, located in Github [6]. Recall from Section I that the purpose of the dapp is to provide investors with a straight-forward mechanism for holding a diversified set of crypto assets. That is, an investor should be able to: create their own portfolio, invest into an existing portfolio, and sell/redeem their holdings in an existing portfolio. The use case is simple, and the code reflects this with only two smart contracts: *Portfolio.sol* and *PortfolioFactory.sol*.

The first, *Portfolio.sol*, represents a portfolio of crypto assets and is itself an ERC20 token. Therefore, any instances of *Portfolio.sol* inherit the methods and attributes (state) of the ERC20 standard, such as minting, burning, and transferring tokens. We build on this basic functionality by adding some extra methods and attributes which enable the contract to operate as a fully functioning portfolio.

Firstly, the array attributes *tokenAddresses* and *percentageHoldings* describe what cryptocurrencies (assets) are held by the portfolio, and the proportion of every purchase that will be allocated to each asset respectively. These are set through the constructor. Each address in *tokenAddresses* is the contract address of an ERC20 token. The attribute *assetQuantities* maps the address of each crypto asset to the quantity of that asset held by the portfolio. This information is crucial in determining how many tokens to issue when a user buys in, as well as the amount of each asset to sell/transfer when a user sells/redeems their tokens. The last attributes, *owner* and *ownerFee*, determine the owner of the portfolio and the fee (number of basis points).

On top of the attributes, the extra public methods are: *initialisePortfolio*, *buy*, *sellAssets*, and *redeemAssets*. The *initialisePortfolio* function enables the owner of a deployed portfolio to supply some Ether, which is swapped for the underlying assets and an initial supply is determined. The *buy* function is payable and enables the user to spend Ether on the assets specified by the portfolio, all in a single transaction. Behind the scenes, the portfolio swaps the spent Ether for the underlying assets using UniSwap, and then issues an appropriate number of portfolio tokens to the buyer. Crucially, it is these tokens that the buyer owns, not the underlying assets. The underlying assets are owned by the contract, so only the contract can sell/transfer them. Thus, if a user wants to sell their holdings, they must interact with the contract through the *sellAssets* or *redeemAssets* functions. Therefore, the user cannot sell the underlying assets separately, which is a crucial feature of the contract and keeps the portfolio tokens linked to the underlying assets. An extra benefit of this abstraction is that users can transfer their holdings in the portfolio to another address, just like any other token. Once the portfolio tokens are transferred, the new owner has the right to the underlying assets which are owned by the contract. The remaining methods, *sellAssets* and *redeemAssets*, enable a user to sell or redeem their holdings in the portfolio provided they own some portfolio tokens. When a user chooses the *sellAssets* function, the contract swaps their portion of the underlying assets for Ether and transfers the Ether back to the user. The difference with *redeemAssets* is that the contract releases ownership of the assets and transfers them to the user, instead of selling them on UniSwap.

There are many benefits to having all this functionality inside a single contract. Firstly, it makes deployments cheaper, as there are fewer contracts to deploy. Secondly, it reduces the security risk by reducing the number of possible attack vectors. Originally, we had an extra contract called *Vault.sol* which was responsible for the ownership of the underlying assets. However, this meant we were transferring ownership of the underlying assets much more frequently, increasing the risk of something going wrong. So, we decided that having the Portfolio own the underlying assets was the best solution.

The second contract in our repository, *PortfolioFactory.sol*, is a simple and secure mechanism for creating portfolios which adheres to the Factory pattern from object-oriented-programming [7]. A modifier on the constructor of the Portfolio contract ensures that portfolio instances can only be created by the PortfolioFactory, allowing us to control the creation and tracking of portfolios.

Overall, many lessons in security were taken from the textbook *Mastering Ethereum* [8], and in protocol design from the Set protocol whitepaper [9] and Github [10].

IV. Testing Methodology

A. Front end

For the front-end, we leveraged the *Jest* JavaScript testing framework. The *expect* function, together with the matcher function *it*, were used to test values. When the testing library ran, it kept notes of all failed matches so that error messages could be printed. The first component we tested was the MetaMask button, because the application could not function without connecting a wallet. Tests were carried out to see if the button was functional, and that the address and balance were retrieved successfully.

B. Back end

Hardhat was used as the development environment for the smart contracts. This framework facilitated compilation, deployment and testing of the contracts, whilst still allowing the use of familiar libraries such as Mocha and Chai (see Figure 7). Hardhat was used instead of Truffle because it made testing contracts deployed on Kovan straightforward, and because it is used widely in industry. Our test suite primarily focused on integration tests, which gave us confidence that the key user interactions were functioning correctly. The tests also ensured that newly added features weren't breaking other parts of the system.

The portfolio factory contract was tested using the *integration-test-portfolio-factory.js* script, which automatically deployed two portfolios. This suite of tests verified that the factory contract was correctly storing portfolio addresses, and that each portfolio was interactive once deployed.

A second suite of tests, *integration-test-portfolio.js*, was created to test the portfolio contract. These tests introduced a 'mock' life cycle of user interactions: deployment, initialisation, buying, selling of assets, and redeeming of assets. Test were in place to verify that: events were emitted correctly, the total token supply was updated correctly, and assets were being purchased and sold through Uniswap in the correct quantities. All functionality that involved swapping assets on Uniswap relied on real liquidity offered by external individuals on the test-nets, which meant that our testing had to be conducted on a test net. This was an unconventional approach, as testing is ideally performed on local blockchains due to greater speed and reproducibility. This meant it was possible a test could fail due to external factors outside of our control.

Further work should be done to verify our smart contracts against a range of malicious inputs, and more advanced methods could also be applied such as fuzzing; a novel method that tests code against a range of valid inputs to reveal defects [11].

V. User Manual

A. Landing page

The Folio website enables the activities shown in Figure 1. Starting on the Landing page (Appendix, Figure 9), you will see two buttons that link to the Explore and Create pages (Appendix, Figure 10 and Figure 12), which can also be reached through the navigation panel at the top of the website. Clicking the Folio logo will also return you to the Landing page. The last component in the navigation panel is the MetaMask button, which will connect your MetaMask account when clicked. This is necessary to access the main functionalities of the dapp.

B. Explore page

Clicking the 3D 'Explore Portfolios' button (Appendix, Figure 10 will take you to a section presenting existing portfolios (Appendix, Figure 11: there are currently three examples available to view). Try clicking on a portfolio to view the details - if you have not connected your MetaMask wallet the block will expand and tell you to connect. Clicking on a portfolio whilst connected to MetaMask will expand the block to show details about the PortFolio, such as circulating supply and the amount of tokens you currently hold, as well as three functional buttons. Type an amount into any of the Buy, Sell or Redeem input boxes and then click the button to the right - this will call a function on our smart contract. These buttons require a valid MetaMask connection to proceed. Once clicked, the MetaMask extension will pop-up and require approval for the transaction. 'Your Tokens' will update once the transaction has been mined.

C. Create page

On the Create page, you will see the steps required for you to create your own portfolio. In total, there are five easy-to-follow steps, with 'Back' and 'Continue' buttons at every step. Step 1 (Figure 13) enables you to choose tokens to add to your portfolio. You can scroll through the list of available tokens on the right hand side, or search if desired. Click on a token to add it to your portfolio, which is displayed on the left hand side. Once you've added tokens, you can set their weights using the slider. These are visualised interactively in the pie chart. You can remove tokens from your portfolio by clicking the bin icon. In step 2 (Figure 14), choose a name and symbol for your portfolio. In step 3 (Figure 15), you can set an optional owner fee. Step 4 (Figure **??**) enables you to deploy your portfolio (to the Kovan network), but gives you a chance to review your portfolio (click 'Edit' to return to the first step) before doing so. If you have connected your MetaMask wallet, you will be prompted to do so. When you click 'Deploy', a MetaMask window will pop up, asking you to sign a transaction and accept the gas fee. This transaction is necessary to deploy your portfolio to the blockchain. Once your portfolio is deployed, the final step (step 5, Figure **??**) will become available to you. At this point, your portfolio does not contain any cryptoassets. So, step 5 enables you to initialise your portfolio with some Ether, which is used to buy the cryptoassets. Enter an amount (in Ether or Wei) and press 'Initialise'. Again, a MetaMask window will pop up asking you to sign a transaction. This is necessary for you to send the Ether amount to your deployed portfolio, which will then buy the cryptoassets on your behalf and issue you some tokens. You will see a loading bar until the transaction is mined. Once complete, you'll be able to view your portfolio and balance.

VI. Critical Evaluation

1) Gas cost analysis: Gas is used to pay miners for executing contracts on Ethereum; gas prices on Ethereum are notoriously high, with an article from mid-2021 indicating that deployment of only moderately complex contracts was costing upwards of $10,000 in fees [12].

Gas cost analysis was performed on three different portfolios and shown in Figure 4. Despite present gas fees being relatively low, the cost to call *create* was extremely high, averaging well over £350. The *create* call was far more expensive than the other functions because this deployed a new smart contract. The portfolio functions themselves had a range of fees, which increased with respect to the number of assets held. This was as expected as each additional currency added meant each function tested required additional Ethereum Virtual Machine (EVM) usage. Whilst these fees do not increase with size of the transaction, it means that the relative gas cost with respect to small purchases is very high, this is problematic for our target audience. Gas costs were minimised with each development iteration of our smart contracts, by removing additional deployments (such as *Vault.sol*), and minimising the amount of data stored.

An active strategy for scaling our dapp could be to perform transactions on a layer-2 scaling solution that uses zero-knowledge rollups, which bundle many transactions together off-chain to reduce fees [13]. Alternative chains are also an option for reducing fees, however lower fees tend to be associated with a reduction in decentralisation, which could go against the ethos of our product. Ethereum sharding may help scale all main-net transactions in the future [14].

2) Runtime analysis: The average Ethereum transaction takes 16 seconds, however it may never be processed, depending on the amount of network congestion and the transaction fee paid [15]. Larger fees are associated with faster processing times (more likely to be included in the current block). The user decides how much they wish to pay and this corresponds to a certain processing time. For functions that query the state of a contract, these do not need to be mined into a block and therefore are relatively fast, depending on the speed of the RPC provider. Folio used Alchemy (Default Provider) instead of Infura because it did not require API keys.

3) Security analysis: Security of decentralised applications is critical because smart contract code is public and can be executed by anyone. This is especially true for DeFi applications, where large sums of money are at stake. Similar products to our portfolios, such as the DeFi Pulse Index, hold millions of dollars of crypto assets [16]. We anticipate our most popular portfolios to grow to a similar size, which could make them an attractive target to hackers.

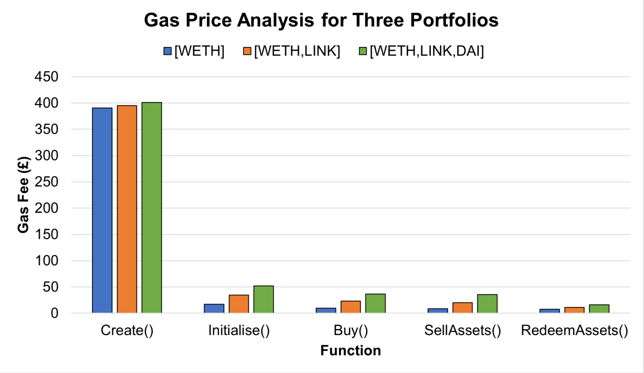


Fig. 4. Gas prices converted to British Pounds (£), using the current gas price: 54.69 Giga-Wei (GWEI) and the current Ethereum price: £2,241.36 (as of 30/04/2022).

Whilst writing the smart contract code, we stuck closely to the principles of 'defensive programming'. This paradigm is laid out in the textbook 'Mastering Ethereum', and recommends the following best practices: minimalism, code reuse, code quality, readability, and test coverage [8]. Firstly, we achieved code minimalism and simplicity by reducing the code wherever possible. For example, we cut our code down to two smart contracts after realising that the third (*Vault.sol*) was not necessary. Secondly, we reused established code libraries rather than writing code from scratch. An example of this is the way we extended OpenZeppelin's ERC20 interface [17]. We also ensured high code quality and readability by performing rigorous code reviews, in which we not only ensured the code was performant but also ensured the code was clean; following the recommendations in Robert Martin's book 'Clean Code' [18]. Finally, we had a high level of test coverage.

Aside from defensive programming we also screened for common security flaws. One such flaw is 'reentrancy', which caused the infamous DAO hack in 2016. A reentrancy bug allows an attacker to 'reenter' the function before an internal variable is updated to prevent them from doing so. An analogy for this could be withdrawing money from an ATM repeatedly before the bank updates your balance, effectively allowing you to withdraw more money than you own. Figure 8 shows our *redeemAssets* function. We burn the caller's tokens before sending them the assets, to stop them being able to 'reenter' the function and redeem more assets that they are not entitled to.

4) Summary: In this section, we have analysed our smart contracts from the perspective of gas, runtime, and security. We've taken steps to reduce the gas consumed and increase the level of security. Despite this, we're still not happy with current gas costs on Ethereum which are too high for our target audience. Runtime is acceptable, but again the high gas

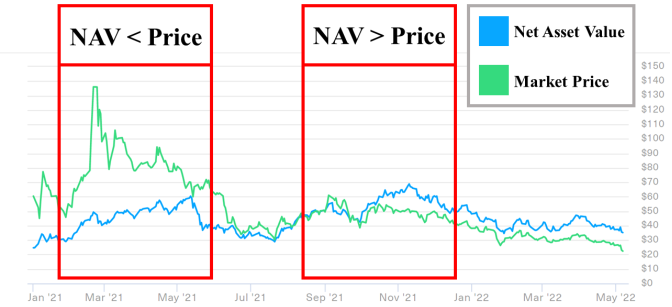


Fig. 5. Bitwise 10 ETF market price and net asset value (NAV). The market price for the index and the net asset value frequently diverged.

costs will stop our users from being able to speed up their transactions if desired. Using Layer-2 solutions and alternative networks could mitigate these gas issues. Finally, we have achieved a high standard of security on our contracts. However, before deploying to Mainnet, we would pay for an external audit to review the contract for security vulnerabilities.

VII. Critical Comparison

As mentioned in Section I, there are several similar on-chain products to Folio. There also exist traditional, or off-chain, indices that track the crypto market. This section covers these in detail and outlines the key advantages of on-chain index funds.

Bitwise are an American company offering a selection of traditional index funds that track cryptoassets. The majority of their offerings are private funds, which are only available to accredited investors with a minimum of $25,000, and are therefore inaccessible to our target users. They also have a public fund, the Bitwise 10 (BITW) ETF, which tracks the top 10 largest cryptocurrencies by market cap (screened for certain risks), and has $800m in assets under management. At first glance, it may appear a popular investment tool for gaining exposure to crypto. However, there are several problems with the index, which extend generally to all off-chain cryptoasset-tracking indices.

Firstly, the index is dominated by Bitcoin and Ethereum, which make up over 90% of the index, leaving investors little option for diversification. Secondly, the underlying cryptoassets are held by a third party custodian (Coinbase), which exposes investors in the index to counterparty risk. Also, the custodian charges a fee, which is passed on to the investor. The total expense ratio for the index is 2.5%, which includes the custody fee just mentioned, as well as admin, audit, and management fees. With an on-chain equivalent, these fees can be dramatically reduced: admin tasks can be automated in the smart contract, there is no need for a custodian since the contract itself stores the assets, and there are no audit fees.

Another fundamental issue with BITW is that its market price and net asset value (NAV) frequently diverge, as shown in Figure 5. In April and May 2021, the price fell despite the NAV increasing. These events are disastrous for investors, who rightly assume that the value of their investment should reflect the value of the underlying assets they have invested in. To combat this, decentralised solutions (such as Folio and Set) allow users to withdraw the underlying assets directly. This means that if the market price for the portfolio was lower than the NAV, a trader could purchase tokens in the portfolio on the market and redeem the underlying assets for a profit (arbitrage). Conversely, if the market price was higher than the NAV, a trader could purchase the underlying assets, use them to mint tokens in the portfolio, and sell the portfolio tokens on the market for a profit. An efficient market should take advantage of these arbitrage opportunities, keeping the NAV in line with the market price. These problems are systemic to traditional (off-chain) crypto tracking indices, caused by the illiquidity of the assets held by the custodian. Decentralised index funds bring more accurate index prices, greater functionality (through redemption of assets), and reduced management costs to investors.

As well as traditional financial products there are several on-chain alternatives to Folio, including Crypto20 and Set Protocol, which were briefly mentioned in Section I. The biggest concern with Crypto20 is the centralised rebalancing mechanism. The calculations are handled off-chain, and the smart contract allows a user with special access to rebalance the underlying assets. As the index size grows there are significant incentives for that user to act dishonestly. It is not uncommon for privileged users such as this to extract money from the contracts, and with pseudonymous wallets it is often impossible to recover funds.

The other alternative, Set Protocol, style themselves as providing “asset management for a DeFi world”. Compared to Crypto20, they offer a much wider range of indices to choose from, and have additional governance which allows for decentralised rebalancing. This is a considerable improvement over Crypto20, and has led to significant growth of their platform. However, developing a profitable business model in a decentralised economy is difficult, and Set Protocol’s primary issue is the high gas fees associated with using the EVM. They do support scaling solutions such as Optimism and Polygon, but the liquidity of exchanges on these side-chains is not enough to support their platform. Whilst they have built a range of advanced features, without reducing user fees there is no market fit. To break into this market, the winner will be the solution that can provide decentralised indices cheaply for retail investors. Therefore, Folio’s primary aim is to reduce gas fees in order to bring decentralised asset management to the retail investor. This provides justification for the simplicity and lack of governance in Folio.

VIII. Conclusion and Future Work

This paper has introduced Folio, a dapp for interacting with and creating crypto index tokens. Folio isn’t the first index-as-a-token provider, that accolade belongs to Crypto20. However, Folio brings a number of advantages over Crypto20, primarily the fact that it is fully decentralised. The first phase of Folio's future work focuses on a number of features which will improve the prices Folio provides for its users, as well as the security of their assets. This will help us compete with our biggest competitor, Set Protocol. Phase 2 of the roadmap takes a different path to Set Protocol, aiming to pivot towards a community-centric, social platform for crypto investing. The full roadmap is shown in the appendix in Figure 6.

REFERENCES

[1] statista, "Overall cryptocurrency market capitalization," 2022. [Online]. Available: https://www.statista.com/statistics/730876/cryptocurrency-maket-value/

[2] crypto20, "The world's first tokenized cryptocurrency index fund," 2022. [Online]. Available: https://www.crypto20.com/en/

[3] tokensets, "Asset management for a defi world," 2022. [Online]. Available: https://www.tokensets.com/

[4] finance.yahoo, "Defi market on the rise," 2022. [Online]. Available: https://finance.yahoo.com/news/defi-market-rise-47-growth-114400683.html

[5] group 4, "group-4-frontend." [Online]. Available: https://github.com/cameronmcewan/group-4-frontend

[6] ——, "group-4-backend." [Online]. Available: https://github.com/cameronmcewan/group-4-backend

[7] tutorials˙point, "Design pattern - factory pattern." [Online]. Available: https://www.tutorialspoint.com/design˙pattern/factory˙pattern.htm

[8] A. M. Antonopoulos and G. Wood, *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

[9] B. W. F. Feng, "Set: A protocol for baskets of tokenized assets," 2019. [Online]. Available: https://www.setprotocol.com/pdf/set˙protocol˙whitepaper.pdf

[10] S. Labs, "set-protocol-v2." [Online]. Available: https://github.com/SetProtocol/set-protocol-v2

[11] B. Jiang, Y. Liu, and W. Chan, "Contractfuzzer: Fuzzing smart contracts for vulnerability detection," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 259–269.

[12] E. [sic] Lastname, "How much does it cost to deploy a smart contract on ethereum?" Dec 2021. [Online]. Available: https://medium.com/the-capital/how-much-does-it-cost-to-deploy-a-smart-contract-on-ethereum-11bcd64da1

[13] A. Garoffolo, D. Kaidalov, and R. Oliynykov, "Zendoo: a zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 1257–1262.

[14] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer *et al.*, "On scaling decentralized blockchains," in *International conference on financial cryptography and data security*. Springer, 2016, pp. 106–125.

[15] V. P. Ranganthan, R. Dantu, A. Paul, P. Mears, and K. Morozov, "A decentralized marketplace application on the ethereum blockchain," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2018, pp. 90–97.

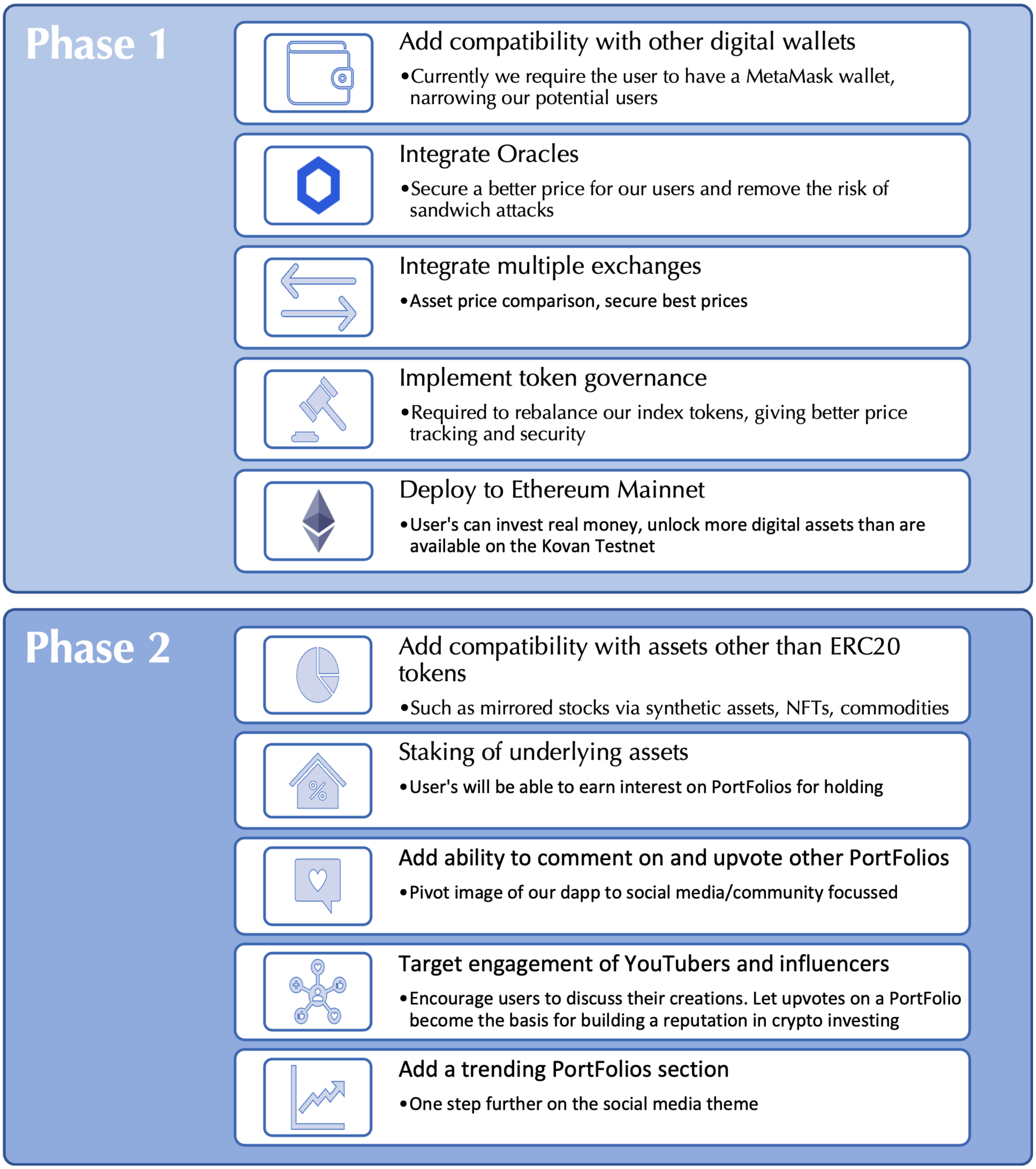
[16] 2022. [Online]. Available: https://coinmarketcap.com/currencies/defi-pulse-index/

[17] [Online]. Available: https://docs.openzeppelin.com/contracts/2.x/api/token/erc20

[18] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

IX. Appendices

A. Roadmap and Future Work



Phase 1

Add compatibility with other digital wallets

- Currently we require the user to have a MetaMask wallet, narrowing our potential users

Integrate Oracles

- Secure a better price for our users and remove the risk of sandwich attacks

Integrate multiple exchanges

- Asset price comparison, secure best prices

Implement token governance

- Required to rebalance our index tokens, giving better price tracking and security

Deploy to Ethereum Mainnet

- User's can invest real money, unlock more digital assets than are available on the Kovan Testnet

Phase 2

Add compatibility with assets other than ERC20 tokens

- Such as mirrored stocks via synthetic assets, NFTs, commodities

Staking of underlying assets

- User's will be able to earn interest on PortFolios for holding

Add ability to comment on and upvote other PortFolios

- Pivot image of our dapp to social media/community focussed

Target engagement of YouTubers and influencers

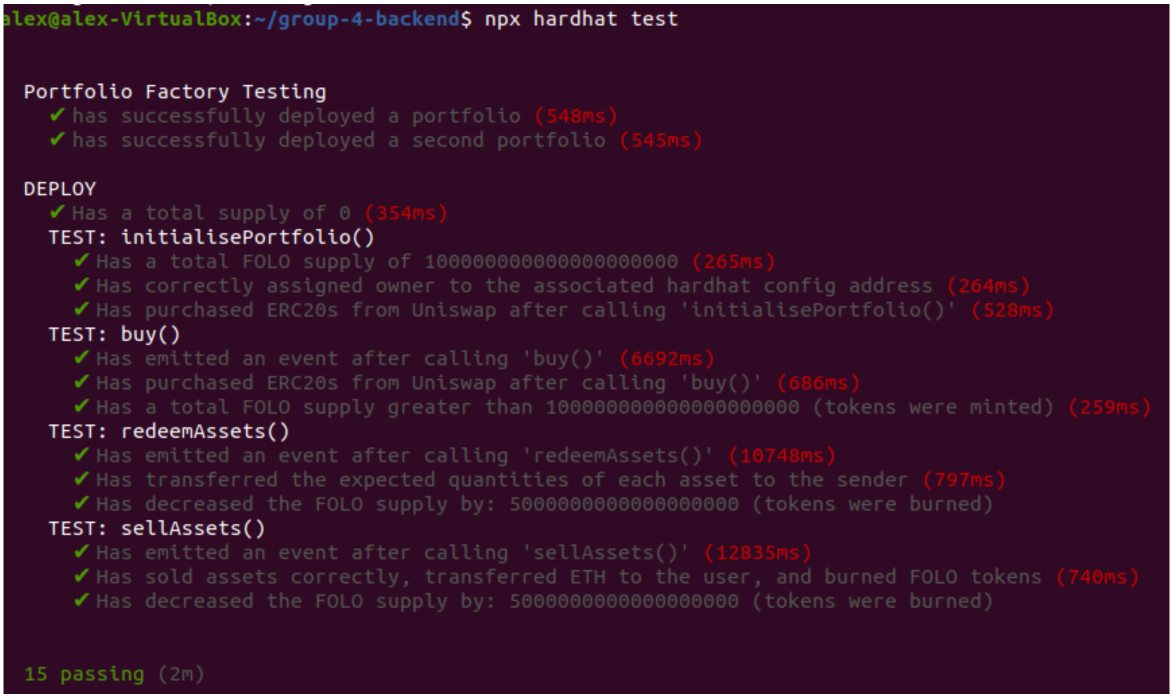
- Encourage users to discuss their creations. Let upvotes on a PortFolio become the basis for building a reputation in crypto investing

Add a trending PortFolios section

- One step further on the social media theme

Fig. 6. Folio roadmap: future steps.

B. Testing and Code Snippets



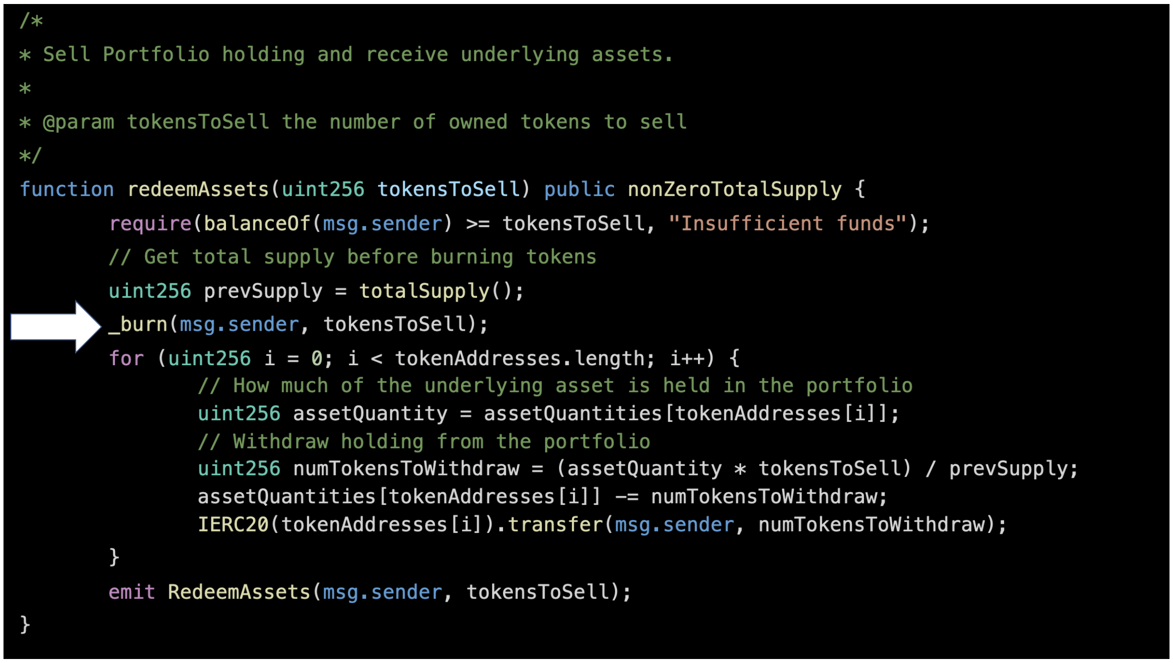
```
alex@alex-VirtualBox:~/group-4-backend$ npx hardhat test


  Portfolio Factory Testing
    ✓ has successfully deployed a portfolio (548ms)
    ✓ has successfully deployed a second portfolio (545ms)

  DEPLOY
    ✓ Has a total supply of 0 (354ms)
    TEST: initialisePortfolio()
      ✓ Has a total FOLO supply of 100000000000000000000 (265ms)
      ✓ Has correctly assigned owner to the associated hardhat config address (264ms)
      ✓ Has purchased ERC20s from Uniswap after calling 'initialisePortfolio()' (528ms)
    TEST: buy()
      ✓ Has emitted an event after calling 'buy()' (6692ms)
      ✓ Has purchased ERC20s from Uniswap after calling 'buy()' (686ms)
      ✓ Has a total FOLO supply greater than 100000000000000000000 (tokens were minted) (259ms)
    TEST: redeemAssets()
      ✓ Has emitted an event after calling 'redeemAssets()' (10748ms)
      ✓ Has transferred the expected quantities of each asset to the sender (797ms)
      ✓ Has decreased the FOLO supply by: 5000000000000000000 (tokens were burned)
    TEST: sellAssets()
      ✓ Has emitted an event after calling 'sellAssets()' (12835ms)
      ✓ Has sold assets correctly, transferred ETH to the user, and burned FOLO tokens (740ms)
      ✓ Has decreased the FOLO supply by: 5000000000000000000 (tokens were burned)


  15 passing (2m)
```

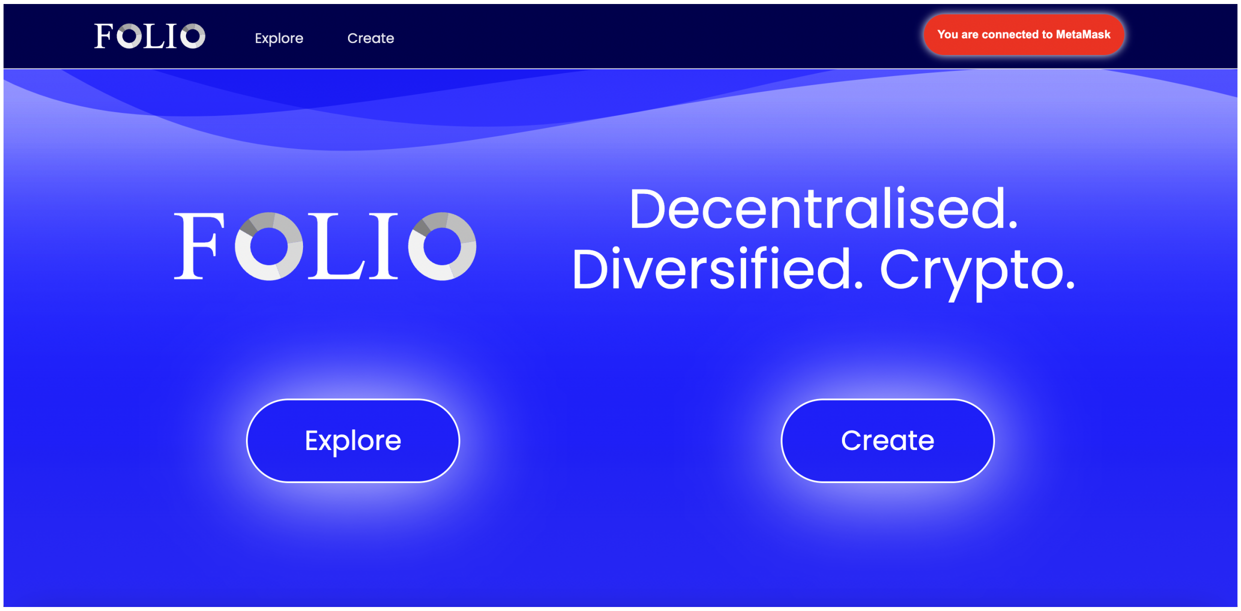
Fig. 7. HardHat testing suite for *Portfolio.sol*, and *PortfolioFactory.sol* smart contracts. These were tested on Kovan as they relied on live Uniswap contracts.



```
/*
* Sell Portfolio holding and receive underlying assets.
*
* @param tokensToSell the number of owned tokens to sell
*/
function redeemAssets(uint256 tokensToSell) public nonZeroTotalSupply {
        require(balanceOf(msg.sender) >= tokensToSell, "Insufficient funds");
        // Get total supply before burning tokens
        uint256 prevSupply = totalSupply();
        _burn(msg.sender, tokensToSell);
        for (uint256 i = 0; i < tokenAddresses.length; i++) {
                // How much of the underlying asset is held in the portfolio
                uint256 assetQuantity = assetQuantities[tokenAddresses[i]];
                // Withdraw holding from the portfolio
                uint256 numTokensToWithdraw = (assetQuantity * tokensToSell) / prevSupply;
                assetQuantities[tokenAddresses[i]] -= numTokensToWithdraw;
                IERC20(tokenAddresses[i]).transfer(msg.sender, numTokensToWithdraw);
        }
        emit RedeemAssets(msg.sender, tokensToSell);
}
```

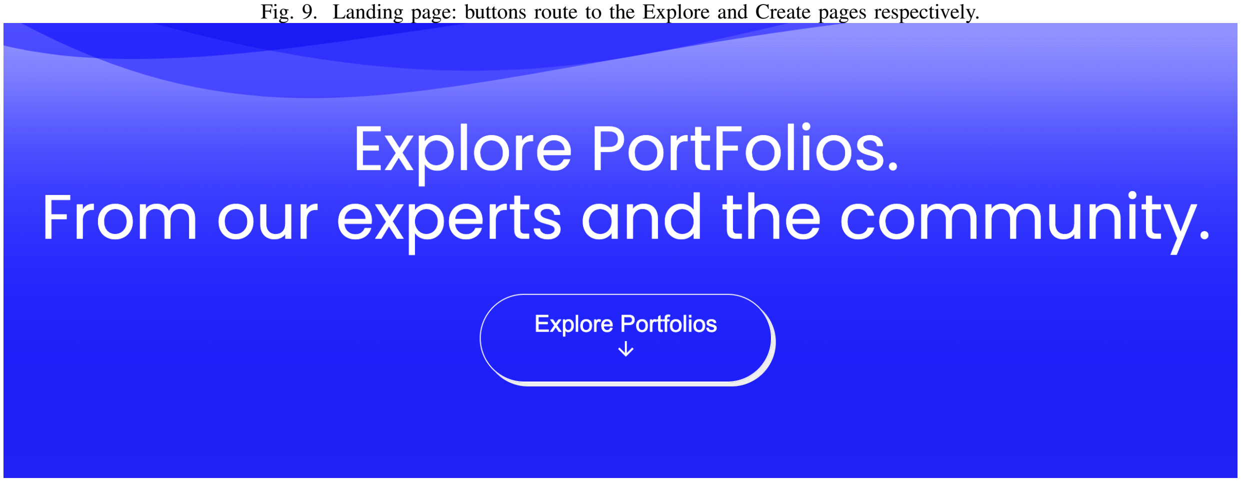
Fig. 8. Avoiding reentrancy attack when redeeming assets.

C. Front end



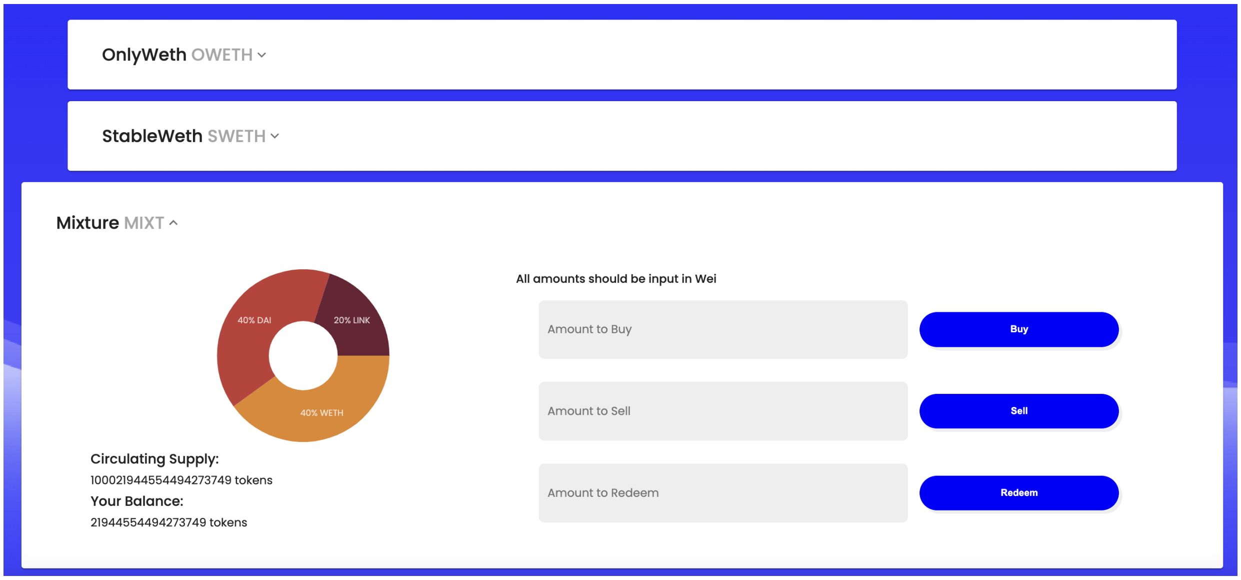
Decentralised.
Diversified. Crypto.

Fig. 9. Landing page: buttons route to the Explore and Create pages respectively.



Explore PortFolios.
From our experts and the community.

Fig. 10. Explore page: heading.



Circulating Supply:
100021944554494273749 tokens

Your Balance:
21944554494273749 tokens

All amounts should be input in Wei

Amount to Buy

Amount to Sell

Amount to Redeem

Fig. 11. Explore page: explore existing PortFolios.

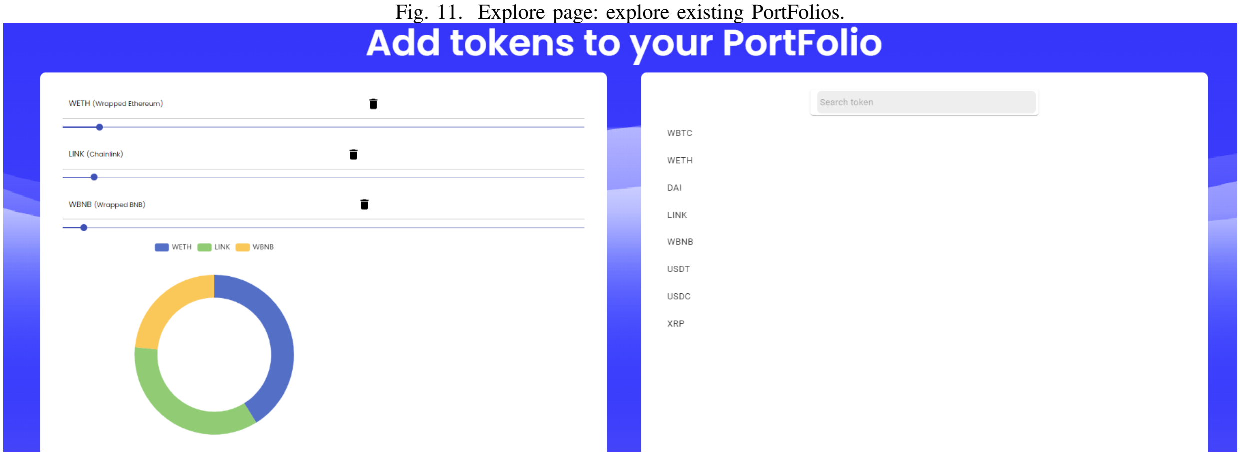
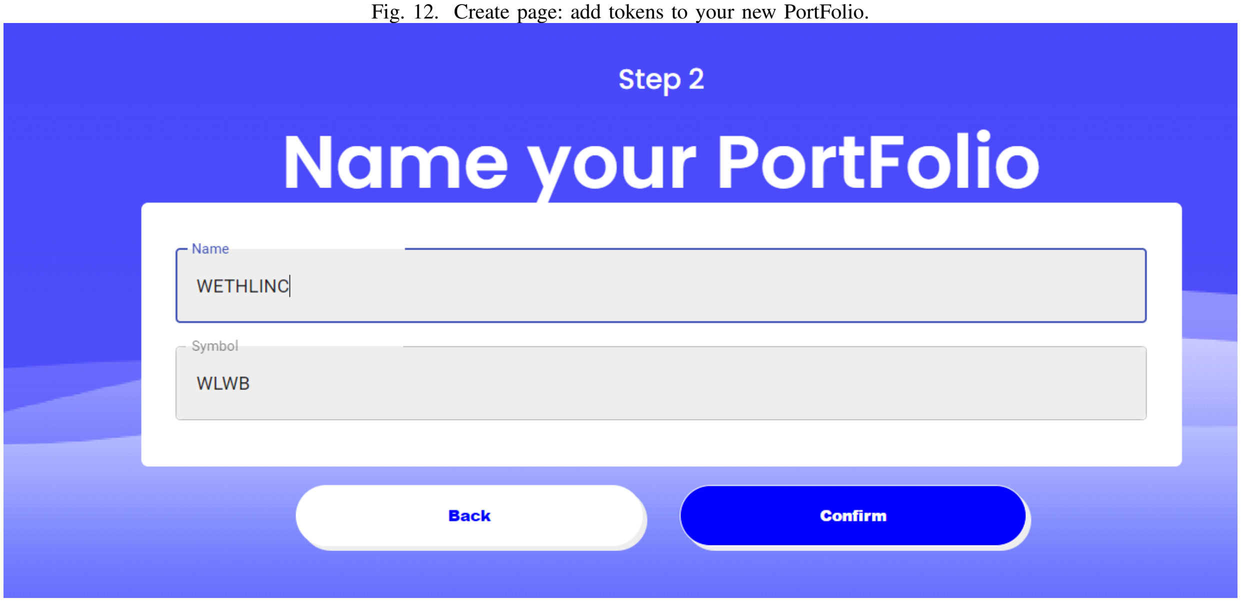

Add tokens to your PortFolio

Fig. 12. Create page: add tokens to your new PortFolio.

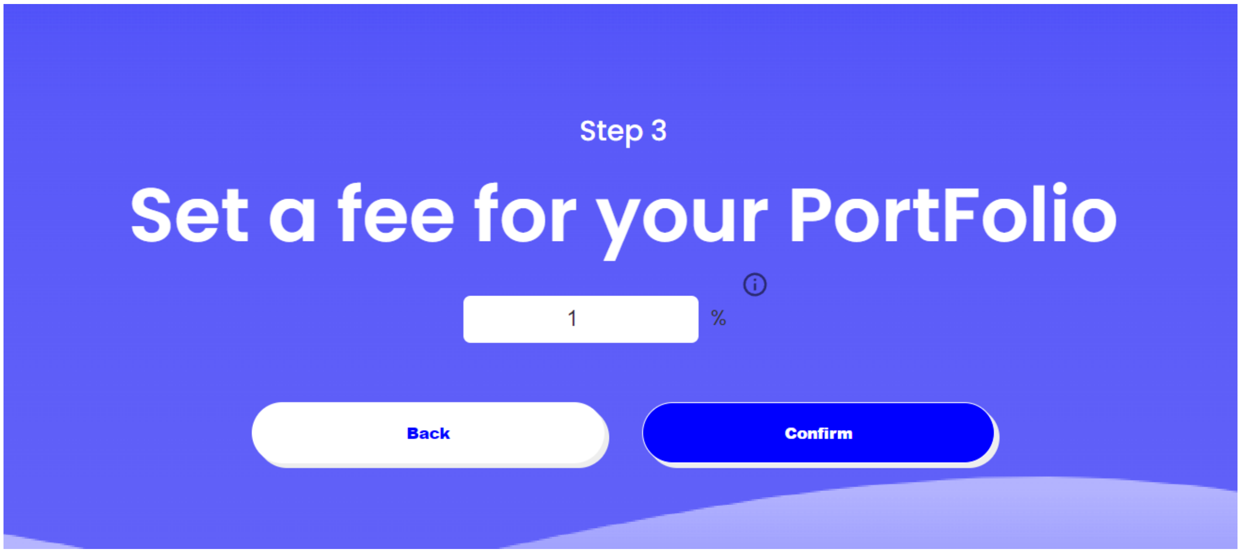

Step 2

Name your PortFolio

Name: WETHLINC

Symbol: WLWB

Fig. 13. Create page: give PortFolio a name and symbol.

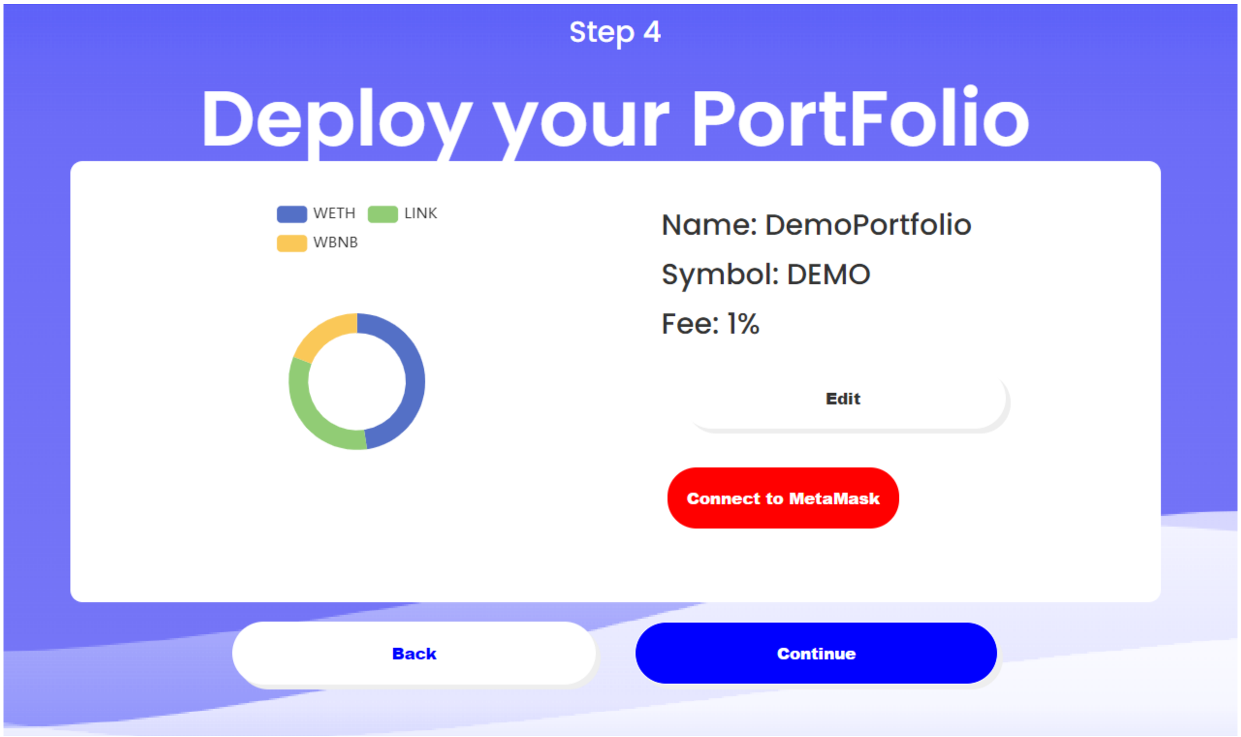


Step 3

Set a fee for your PortFolio

1 %

Fig. 14. Create page: set an optional fee for your new PortFolio.



Step 4

Deploy your PortFolio

Name: DemoPortfolio

Symbol: DEMO

Fee: 1%

Fig. 15. Create page: deploying your Portfolio.