

# Blockchain Challenge Task

---

## Challenge Task HS 2024

This semester's challenge task (CT) is the design and implementation of a stablecoin with rebalancing on decentralized exchange (DEX). Ideally, the application is a decentralized application (DApp).

- Your app must include a simple frontend (e.g., HTML, Vue, React, Svelte, ...)
  - Issue and redeem stablecoin
- Your app must include a public blockchain (e.g., Ethereum, Solana, Sui, ...)
  - Trading can be done with existing DEXes

## Requirements

All requirements below must be met in order to pass this lecture.

1. A working prototype with your stablecoin.
  2. Use latest stable releases of chosen libraries and frameworks.
  3. Interaction with a public blockchain (can be testnet).
  4. Must be tradable on a decentralized exchange (DEX)
  5. Stablecoin needs to rebalance on DEX to remain stable
  6. Status and process need to be shown in the frontend.
- The solution may use existing libraries and code, but those must open software software.

## Overview

The source code can be found in the [github repo](#)

### CCR Token

For this project, the CCR Token was created. CCR stands for Corsin, Cedric and Roman. The CCR token is ERC-20 based and heavily (if not completely) based on the OpenZeppelin template and runs on the Sepolia Testnet

### Pool CCR - WETH (Wrapped ETH)

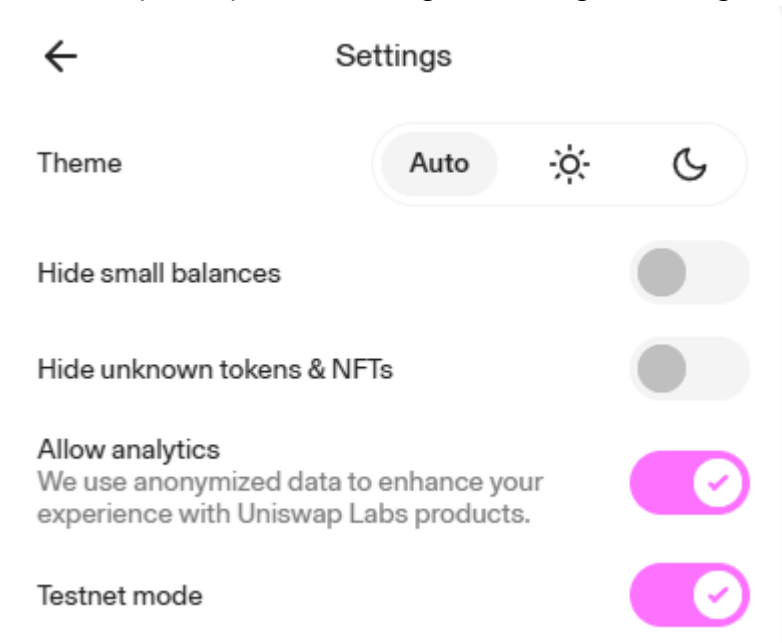
We used Sepolia ETH for token swapping due to its ease of access, zero-cost transactions, fast confirmations, compatibility with Ethereum standards, and safe environment for development and testing.

## Technical implementation

### Connection to uniswap

We use uniswap to store our fictive currency.

On uniswap its important to change the settings according to this picture:



## Pool rebalancing with python-bot

The code for the python-bot is the identically named folder. It contains a `environment.yaml` file to import python dependencies (in a conda environment).

### Reasons for a python bot

While working on this project, our team struggled with the rebalancing with the pool. The lack of documentation left us with our heads scratching. The python-bot seemed like a "simple" option, as in terms of what the bot should do, it was easy to understand. Implementing it had its challenges.

### Target ration & ratio threshold

The target ratio is perfect ratio of what the bot wants to achive. In our case the **target ratio is 1000**, meaning that the 1 WETH token is worth 1000 CCR tokens.

Due to transactions fees, achieving the exact target ratio is pratically impossible. Therefore the ratio threshold (in our case 10), gives some wiggle room around the target ratio which is accepted and no rebalancing is done. If this threshold didn't exist, the bot would continue to rebalance the pool in an attempt to hit the target ratio.

### How does the bot work

The rebalancing only works if the python script is running. The script has access to a wallet and all the tokens within, the funds of said wallet are used to rebalance the pool. The bot performs the following steps:

0. Loads adresses of pool, tokens, router, ABIs and contracts.
1. Retrieves balances of CCR and WETH in pool and calculates ratio.
2. If ratio is +-10 from the target ratio, tokens need to be added to the pool.
3. Calculates CCR or WETH amount to add to the pool.
4. Transaction needs to be approved, so the Uniswap router can execute the transaction.
5. Executes `exactInputSingle` with the previously calculated value.

6. Waits for one minute, then starts at 1. again.

Because `exactInputSingle` exchanges coins in the pool, after adding CCR to the pool the amount of WETH is reduced. Means, when adding 20 CCR to the pool, 0.02 WETH are transferred out. In the next cycle, the bot counteracts that by adding 0.02 WETH to the pool, receiving 20 CCR again. This does not actually rebalance the pool, therefore the amount to put into the pool is taken  $\cdot 0.7$ . This helps getting closer to the preferred ratio, but also means that a couple of "rebalacings" are needed to hit the target ratio. Not sure if this is the smartest, cheapest or fastest way how to do it, but it works 😊.

## Console output

While the bot runs, it writes the token balance, ratios and approval and transaction hashes into the console. This screenshot shows the last rebalancing, it hashes and then the bot noticing the ratio is within the threshold.

[illegible]

## (Future?) Improvements

The first and most obvious one is not committing the private key to GitHub 🙄. Then the two `execute_trade` functions can probably be merged, but during the development the WETH token made problems. Now it works and I am not touching it again.

The calculation of the amount of tokens to place into the pool can most likely be optimized. Currently it over- and undershoots until it is within the threshold. Some clever maths needed. But for the sake of this challenge task, this is sufficient as the rebalancing works even though it isn't optimal.

## Frontend (Cedric Christen)

## Technologies

## Node.js

Our tech stack includes Node.js as the runtime environment and Express.js as the web framework, for building our backend services and APIs.

## Dependencies

The frontend uses 2 main dependencies.

1. Express with handlebars to structure the program and be able to dynamically show the results.
2. Web3 to be able to connect to the CCR Token Contract and the Pool Contract and get the needed informations.

## API calls

To get all the informations, the frontend needs to make 2 different api calls. First it needs to use the CCR token to get the past events of that token and list it. Here all the transactions are shown. The second call is for the price. Here it needs to connect to the pool and from there fetch the `sqrtprix96` price und calculate the human readable version of it.

## Calculations

The frontend fetches the status and processes from the backend to display it to a user. [Guide](#). The price received from the pool is in `sqrtn96` format which means, that it encapsulates the price in 96 bits precision `x96`. So this price has to be computed to a human readable float. [Calculation](#)

## UI

Our frontend UI leverages Handlebars.js to fetch data from the backend and dynamically inject it into HTML templates. For styling and design, we utilize CSS to create a responsive and visually appealing user interface.

Here the result can be seen:

CCR

Current Price:

0.0013196125512879154 ETH per CCR  
757.7981878271921 CCR per ETH

Refresh

Transaction History

Transaction 29:

From: 0x98646142B3661f3739EDAdfAb30947849e3117a6  
To: 0x4D9881E60cd26Cb5acc66c83BCd84cF27D3C3470  
Transferred: 4.547886867011493236 CCR

Transaction 28:

From: 0x4D9881E60cd26Cb5acc66c83BCd84cF27D3C3470  
To: 0x98646142B3661f3739EDAdfAb30947849e3117a6  
Transferred: 8.105907505989677056 CCR

Transaction 27:

From: 0x98646142B3661f3739EDAdfAb30947849e3117a6  
To: 0x4D9881E60cd26Cb5acc66c83BCd84cF27D3C3470  
Transferred: 19.232578960407721089 CCR

Transaction 26:

From: 0x4D9881E60cd26Cb5acc66c83BCd84cF27D3C3470  
To: 0x98646142B3661f3739EDAdfAb30947849e3117a6  
Transferred: 34.125801387071901696 CCR

Transaction 25:

From: 0x4D9881E60cd26Cb5acc66c83BCd84cF27D3C3470  
To: 0x98646142B3661f3739EDAdfAb30947849e3117a6  
Transferred: 4.473712218655421440 CCR