VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING



**BUILDING A DECENTRALIZED EXCHANGES (DEX) PLATFORM WITH AN AUTOMATED MARKET MAKER (AMM) MODEL**

By

NGUYEN MINH QUAN

A thesis submitted to the School of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of   
Bachelor of Computer Science

Ho Chi Minh City, Vietnam  
2022

**BUILDING A DECENTRALIZED EXCHANGES (DEX) PLATFORM WITH AN AUTOMATED MARKET MAKER (AMM) MODEL**

APPROVED BY: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ,   
Tran Thanh Tung, Ph.D

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

THESIS COMMITTEE

# ACKNOWLEGMENTS

Words cannot express my gratitude to my professor, Dr. Tran Thanh Tung for his invaluable patience and feedback. I also could not have undertaken this journey without my defense committee, who generously provided knowledge and expertise. Additionally, I am also grateful to my friends for their editing help, late-night feedback sessions, and moral support.

# TABLE OF CONTENTS

[ACKNOWLEGMENTS 3](#_Toc121721777)

[TABLE OF CONTENTS 4](#_Toc121721778)

[LIST OF TABLES 6](#_Toc121721779)

[LIST OF FIGURES 7](#_Toc121721780)

[LIST OF ABBREVIATIONS 9](#_Toc121721781)

[ABSTRACT 10](#_Toc121721782)

[CHAPTER 1 11](#_Toc121721783)

[INTRODUCTION 11](#_Toc121721784)

[1.1. Background 11](#_Toc121721785)

[1.2. Problem Statement 11](#_Toc121721786)

[1.3. Objectives 12](#_Toc121721787)

[1.4. Assumption 12](#_Toc121721788)

[CHAPTER 2 13](#_Toc121721789)

[BACKGROUND/APPLIED TECHNOLOGIES 13](#_Toc121721790)

[2.1. Overview of Web 3.0 13](#_Toc121721791)

[2.2. ReactJS with Tailwind CSS (Front-End) 14](#_Toc121721792)

[2.2.1. ReactJS 15](#_Toc121721793)

[2.2.2. Tailwind CSS 15](#_Toc121721794)

[2.3. NodeJS (Running Environment) and NPM 17](#_Toc121721795)

[2.3.1. NodeJS 17](#_Toc121721796)

[2.3.2. NPM 18](#_Toc121721797)

[2.4. Overview of Ethereum Blockchain technologies 19](#_Toc121721798)

[2.4.1. Blockchains Explained 19](#_Toc121721799)

[2.4.2. The Ethereum Blockchain 20](#_Toc121721800)

[2.4.3. Ethereum Platform 21](#_Toc121721801)

[2.4.4. Decentralized Finance 28](#_Toc121721802)

[2.5. CRANQ (Back-End) 34](#_Toc121721803)

[CHAPTER 3 37](#_Toc121721804)

[METHODOLOGY 37](#_Toc121721805)

[3.1. System Description 37](#_Toc121721806)

[3.2. System Use Case 38](#_Toc121721807)

[3.3. System Functionality 39](#_Toc121721808)

[CHAPTER 4 41](#_Toc121721809)

[IMPLEMENTATION 41](#_Toc121721810)

[4.1. Configuration 41](#_Toc121721811)

[4.1.1. NodeJS and NPM 41](#_Toc121721812)

[4.1.2. CRANQ 41](#_Toc121721813)

[4.1.3. MetaMask 43](#_Toc121721814)

[4.1.4. Alchemy 44](#_Toc121721815)

[4.2. Implementation 46](#_Toc121721816)

[4.2.1. Back-End 46](#_Toc121721817)

[4.2.2. Front-End 46](#_Toc121721818)

[CHAPTER 5 47](#_Toc121721819)

[RESULT AND EVALUATION 47](#_Toc121721820)

[5.1. Discussion 47](#_Toc121721821)

[5.2. Comparison 47](#_Toc121721822)

[5.3. Evaluation 47](#_Toc121721823)

[CHAPTER 6 48](#_Toc121721824)

[CONCLUSION AND FUTURE WORK 48](#_Toc121721825)

[6.1. Conclusion 48](#_Toc121721826)

[6.2. Future work 48](#_Toc121721827)

[REFERENCES 49](#_Toc121721828)

[APPENDIX 51](#_Toc121721829)

# LIST OF TABLES

[Table 1. The difference between Fungible Tokens and Non-fungible Tokens 30](#_Toc121721830)

[Table 2. System’s Functional Description 39](#_Toc121721831)

# LIST OF FIGURES

[Figure 2.1.1-1. The Evolution of the Web 14](#_Toc121721832)

[Figure 2.2-2. ReactJS with Tailwind CSS 14](#_Toc121721833)

[Figure 2.2.2-3. ReactJS Abilities 15](#_Toc121721834)

[Figure 2.2.2-4. Tailwind CSS Intellisense for VS Code 16](#_Toc121721835)

[Figure 2.2.2-5. “Subcribe” button with Tailwind CSS 17](#_Toc121721836)

[Figure 2.2.2-6. The result of “Subcribe” button 17](#_Toc121721837)

[Figure 2.3.1-7. NodeJS Architecture 18](#_Toc121721838)

[Figure 2.3.2-8. NodeJS local dependencies managed by NPM in package.json 19](#_Toc121721839)

[Figure 2.4.1-9. Visualization of a Blockchain 20](#_Toc121721840)

[Figure 2.4.2-10. Peer-to-peer Network model with EVM 21](#_Toc121721841)

[Figure 2.4.3.1-11. The different between EOAs and Contract Account 22](#_Toc121721842)

[Figure 2.4.3.3-12. Blocks in Ethereum 24](#_Toc121721843)

[Figure 2.4.3.5-13. Smart Contract ABI 26](#_Toc121721844)

[Figure 2.4.3.6-14. The algorithm x \* y = k 27](#_Toc121721845)

[Figure 2.4.3.6-15. How Uniswap works 28](#_Toc121721846)

[Figure 2.4.4.1-16. Application of dApps 29](#_Toc121721847)

[Figure 2.4.4.7-17. MetaMask Wallet 33](#_Toc121721848)

[Figure 2.4.4.7-18. MetaMask approves transaction 34](#_Toc121721849)

[Figure 2.5-19. Creating a new factory contract using CRANQ 35](#_Toc121721850)

[Figure 2.5-20. Every node with details in CRANQ 36](#_Toc121721851)

[Figure 3.2-1. System Use Cases 38](#_Toc121721852)

[Table 2. System’s Functional Description 39](#_Toc121721853)

[Figure 4.1.1-1. Node and NPM version 41](#_Toc121721854)

[Figure 4.1.2-2. CRANQ first screen 42](#_Toc121721855)

[Figure 4.1.2-3. Library of CRANQ 42](#_Toc121721856)

[Figure 4.1.3-4. MetaMask icon in browser 43](#_Toc121721857)

[Figure 4.1.3-5. MetaMask Homescreen 44](#_Toc121721858)

[Figure 4.4.4-6. API key and provider URL 45](#_Toc121721859)

# LIST OF ABBREVIATIONS

DEXs

AMM

UI

NPM

DLT

MVC

EOAs

ABI

ETH

TTF

EIPs

ERCs

LPs

SPA

IDE

Decentralized Exchanges

Automated Market Maker

User Interface

Node Package Management

Distributed Ledger Technology

Model-View-Controller

Externally Owned Accounts

Application Binary Interface

Ether

Token Taxonomy Framework

Ethereum Improvement Proposals

Ethereum Request for Comments

Liquidity Providers

Single Page Application

Integrated Development Environment

# ABSTRACT

Innovations in cryptocurrency exchanges in recent times have made it easier for people to trade cryptocurrency assets. The increasing number of exchanges has  
made it clear that this domain will keep evolving in the coming years. Decentralized exchanges (DEXs) revolutionized the way we trade cryptocurrencies by eliminating  
third parties and allowing transparent and open access to all the activities occurring on the platform. As the value of cryptocurrencies has increased, several exchanged platforms have appeared like Uniswap, PancakeSwap, Binance,… allow traders to directly swap from one token to another tokens.

As an integral part of the decentralized finance (DeFi) ecosystem, DEX with automated market maker (AMM) protocols have gained massive traction with the recently revived interest in blockchain and distributed ledger technology (DLT) in general. Instead of matching the buy and sell sides, AMMs employ a peer-to-pool method and determine asset price algorithmically through a so-called conservation function.

This thesis is carried out exploring how DEXs apply with AMM using protocol from Uniswap V2. Traders do not have to trade directly with each other. Instead, they trade with a token pool that has both tokens reserved. Everything is done with automatically.

# CHAPTER 1

# INTRODUCTION

## Background

DeFi is a software infrastructure, based on blockchains and smart contracts, which allows users to create and trade crypto-tokens without the intermediation of central authorities, unlike traditional finance. AMM is one of the main DeFi archetypes. Moreover, AMM is decentralized markets of crypto-tokens, providing users with three core operations:

* Depositing crypto-tokens to obtain shares in an AMM;
* The dual operation of redeeming shares in the AMM for the underlying tokens;
* Swapping tokens of a given type for tokens of another type.

The amount of tokens received by a user upon a swap is algorithmically determined by the AMM: roughly, this is the amount of tokens sent from the user to the AMM, times the swap rate, which is computed by the AMM based on its internal state and the input amount.

## Problem Statement

Before having AMM, people use “order books” to trade tokens. In an order book system, a buying order is called a bid, and a selling order is an ask. When buyers upload their orders in DEXs, the order book is in charge of organizing suitable sellers that have related-price orders. Thus, DEX traders can do trading activities successfully when their orders are matched with other traders’ orders.

There is an existing problem of the on-chain order book system in DEXs. Unlike an off-chain order book, an on-chain order book lacks censorship resistance by having relayers to store the orders on centralized servers. In this system, if users want to put limit orders or cancel existing orders, they need to create and submit a transaction. This step includes a dangerous threat for users when the transactions in blockchain storage can be revealed to miners, although they are not included in the block.

## Objectives

The main goal of this thesis paper is to provide the fundamental knowledge of Ethereum technology and show blockchain capabilities compared to traditional Web Application Development.

This paper contains system requirement, which shows some system criterion that must be fulfilled, then showing system architecture and system design. Last but not least, shows all implementation how to construct a demonstration for this thesis.

In this thesis, I will create a DEX with AMM for swapping tokens that students earn from club’s activities to behavior score’s token just like Uniswap V2. Instead of storing on computers, you can use Blockchain and easily check the history of tokens exchanges. Besides, students can control and see their beahvior score’s token.

## Assumption

The context of this thesis proposes that all the required documents of the

property has been legally digitalized by the government, International University and the usage of the system are approved.

# CHAPTER 2

# BACKGROUND/APPLIED TECHNOLOGIES

## Overview of Web 3.0

The premise of “Web 3.0” was coined by Ethereum co-founder Gavin Wood shortly after Ethereum launched in 2014 [1]. In Web 3.0, developers do not usually build and deploy applications that run on a single server or that store their data in a single database. Instead, Web 3.0 applications either run on blockchains, decentralized networks of many peer-to-peer nodes (servers), or a combination of the two that forms a [cryptoeconomic protocol](https://thegraph.com/blog/modeling-cryptoeconomic-protocols-as-complex-systems-part-1). These apps are often referred to as decentralized apps (dApps).

* Verifiable
* Trustless
* Self-governing
* Permissionless
* Distributed and robust
* Stateful
* Native built-in payments

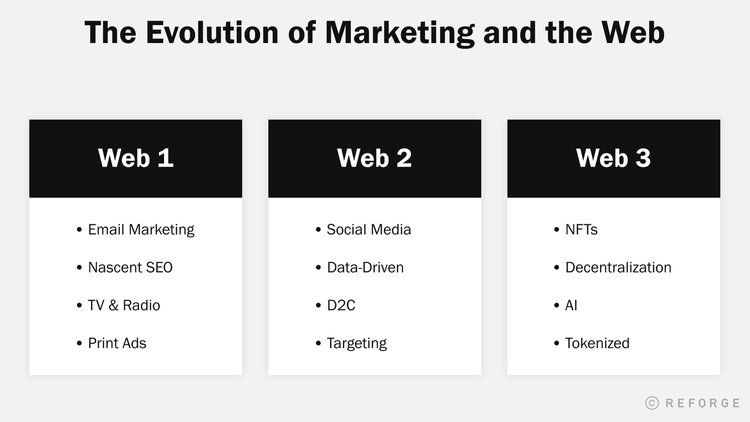


Figure 2.1.1-1. The Evolution of the Web

## ReactJS with Tailwind CSS (Front-End)

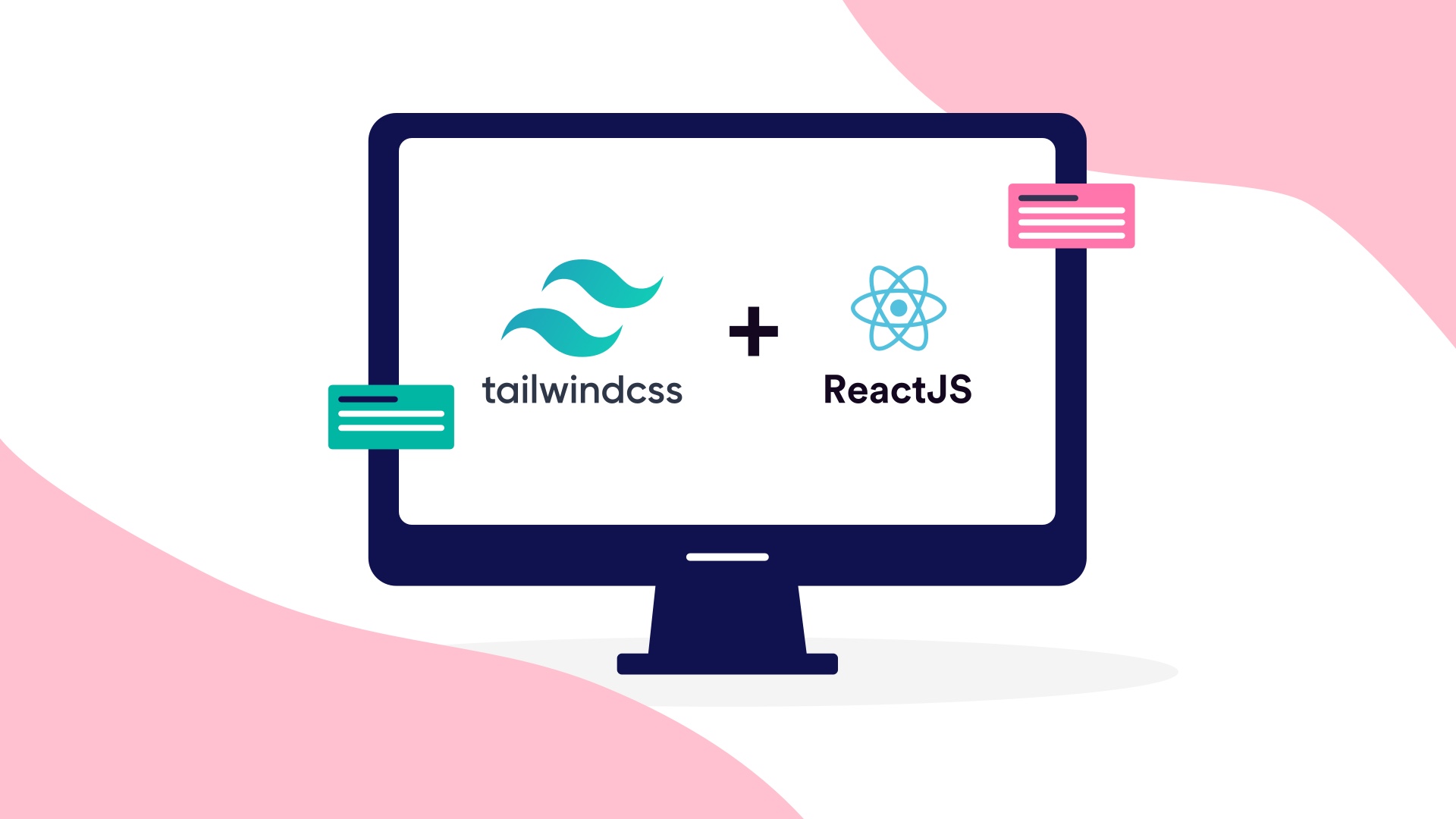


Figure 2.2-2. ReactJS with Tailwind CSS

### ReactJS

The React.js framework is an open-source JavaScript framework and library developed by Facebook [2]. It is used for building interactive user interfaces and web applications quickly and efficiently with significantly less code than you would with vanilla JavaScript.

React’s primary role in an application is to handle the view layer of that application just like the V in a model-view-controller (MVC) pattern by providing the best and most efficient rendering execution. Rather than dealing with the whole user interface as a single unit, React.js encourages developers to separate these complex UIs into individual reusable components that form the building blocks of the whole UI.

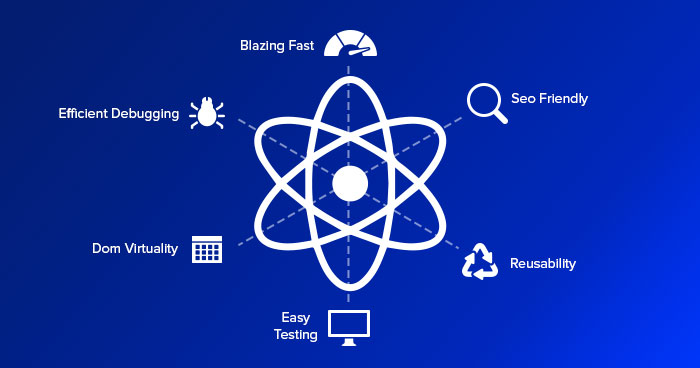


Figure 2.2.2-3. ReactJS Abilities

### Tailwind CSS

Tailwind CSS is a utility-first CSS framework designed to enable users to create applications faster and easier. You can use utility classes to control the layout, color, spacing, typography, shadows, and more to create a completely custom component design — without leaving your HTML or writing a single line of custom CSS.

There are many benefits of using a CSS framework like Tailwind. Below are the major ones:

* **Write less custom CSS**: With Tailwind, you style elements by applying pre-existing classes directly in your [HTML](https://blog.hubspot.com/website/html). By using utility classes in this way, you can build custom designs without writing CSS.
* **Keep CSS files small:** Without a framework like Tailwind, you have to keep writing CSS as you add new features and components. ASa result, your CSS files continue to grow and get heavier. By using utilities like Tailwind’s flexbox and padding utilities, most styles are reusable, so you rarely need to write new CSS.
* **Do not have to invent class name**: When Tailwind, you’re choosing classes from a pre-defined [design system](https://blog.hubspot.com/website/design-system). That means you don’t need to agonize over picking the “perfect” class names for certain styles and components or remember complicated ones like sidebar-inner-wrapper.
* **Can make safe changes:** With the traditional approach, if you make changes to CSS, you may break something across your site. Unlike CSS, utility classes in your HTML are local. That means you can change them without worrying about breaking something else on your site.

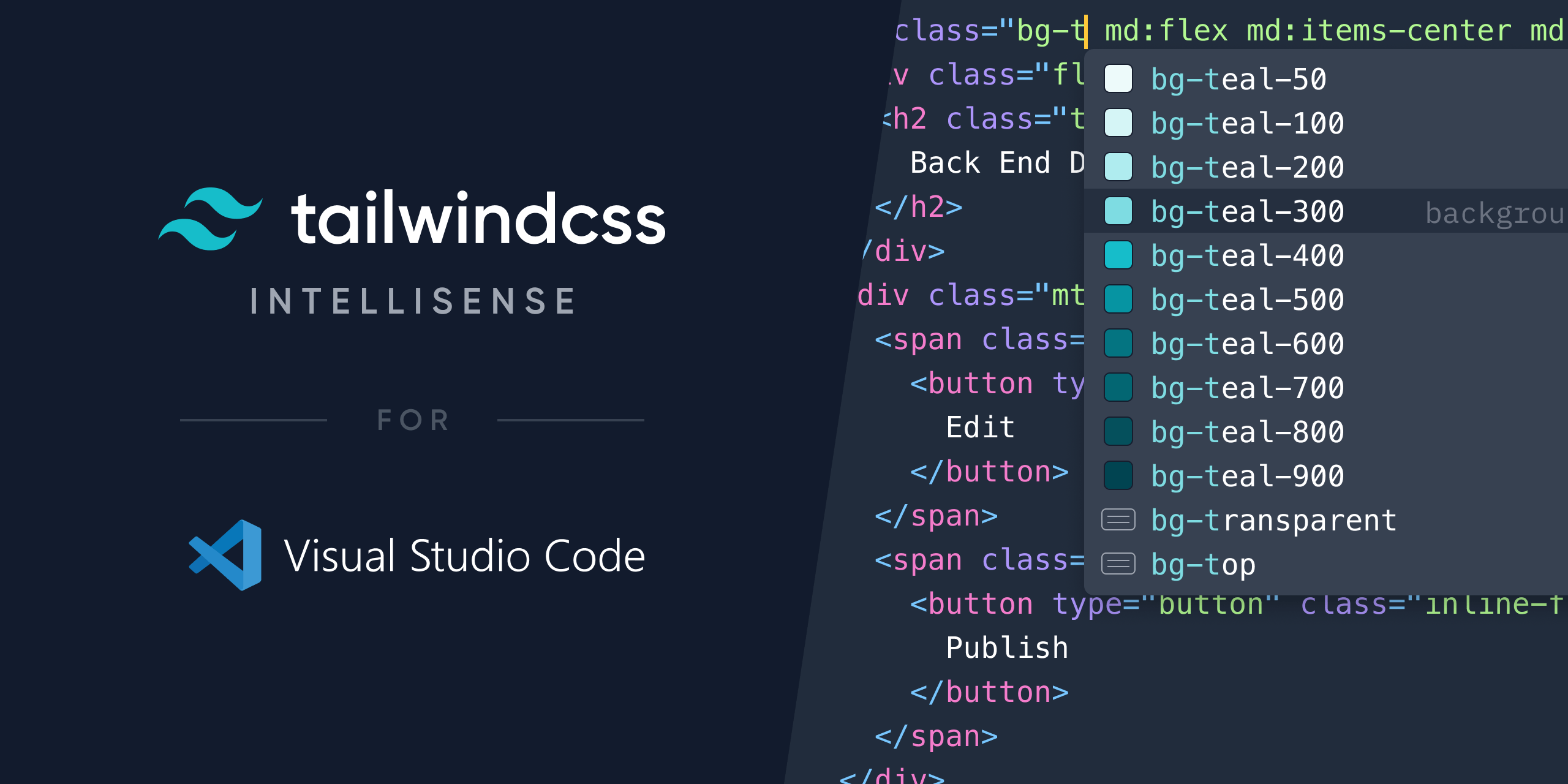


Figure 2.2.2-4. Tailwind CSS Intellisense for VS Code

For example, let’s say I want to create a simple yellow rounded “Subcribe” button, here is the code:

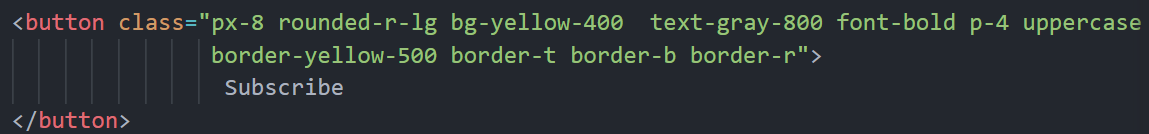


Figure 2.2.2-5. “Subcribe” button with Tailwind CSS

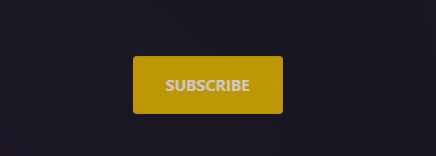


Figure 2.2.2-6. The result of “Subcribe” button

## NodeJS (Running Environment) and NPM

### NodeJS

Node.js is an open-source, cross-platform JavaScript runtime environment and library for running web applications outside the client's browser. Developers use Node.js to create server-side web applications, and it is perfect for data-intensive applications since it uses an asynchronous, event-driven model.

* NodeJs is built on Google Chrome’s V8 engine, and for this reason its execution time is very fast, and it runs very quickly.
* As NodeJs do not need to wait for an API to return data , so for building real time and data intensive web applications, it is very useful. It is totally asynchronous in nature that means it is totally non-blocking.
* As NodeJs is open-source and it is nothing but a JavaScript framework , so for the developers who are already used to JavaScript, for them starting developing their projects with NodeJs is very easy.

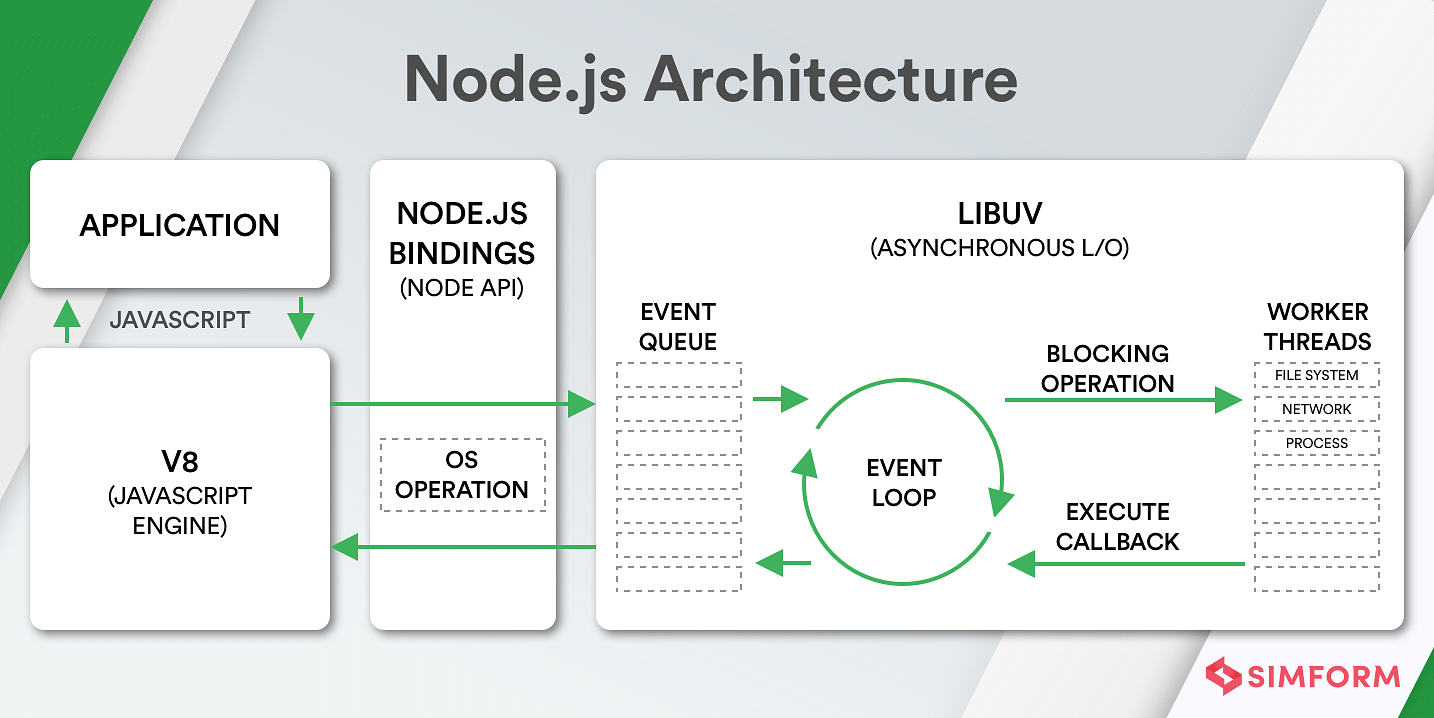


Figure 2.3.1-7. NodeJS Architecture

### NPM

**N**ode **p**ackage **m**anager (NPM) is an open-source repository of tools engineers use to develop applications and websites. NPM is two things:

* A repository for publishing open-source projects. Simplified version: a digital storage and retrieval facility.
* A command-line interface (CLI) for interacting with the repository. Simplified version: a tool to communicate with the storage facility.

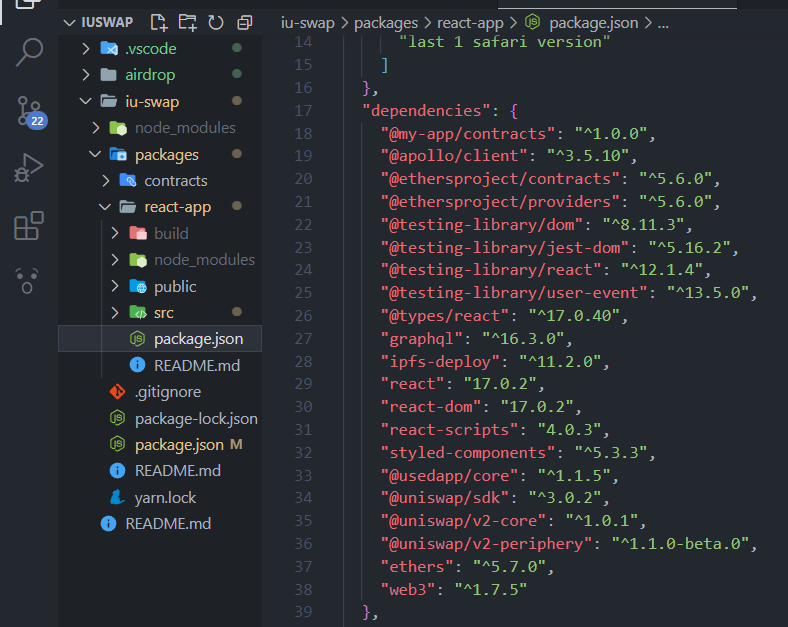


Figure 2.3.2-8. NodeJS local dependencies managed by NPM in package.json

## Overview of Ethereum Blockchain technologies

### Blockchains Explained

Blockchains are the most fundamental parts of this project. A blockchain is a public ledger, or more simply a database, of all transactions stored on a decentralized network of nodes. This database is called a blockchain because the transactions included in the database are stored in linearly consecutive blocks where all blocks have a reference to the earlier block as can be seen in figure 2.1. To create new blocks, nodes gather up a set of transactions not included in any previous block and apply a cryptographic hashing function to solve a challenging cryptographic puzzle.

This process is called mining and when a new block has been created, the node that was able to first mine and distribute the new block into the network is rewarded with new coins. A fee has to be paid for every transaction and this is transferred to the miner of the block when the transaction is included in a new block. This fee and the rewards gained through mining a new block give extra incentives for miners of cryptocurrencies to continue with their activity.

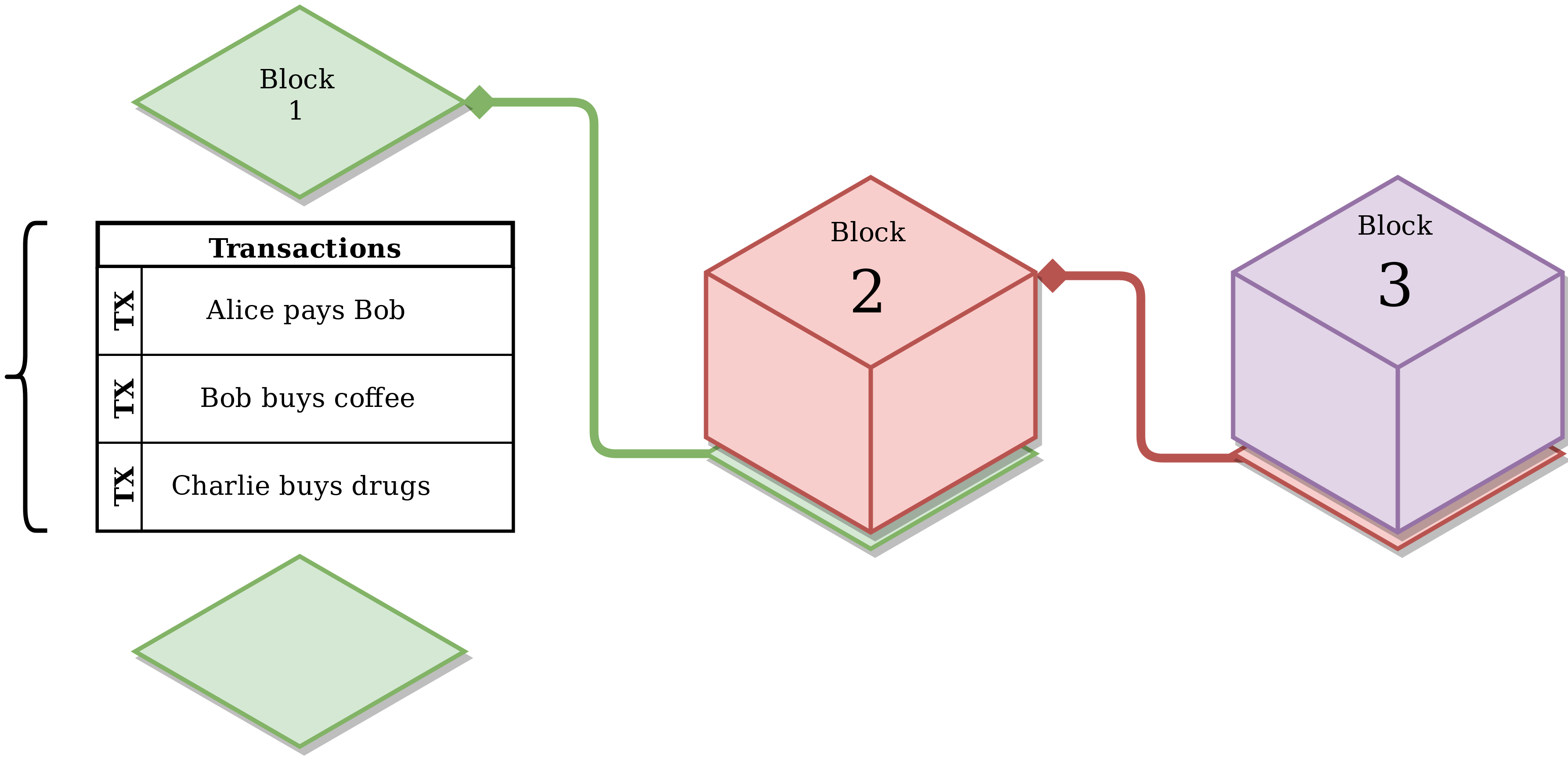


Figure 2.4.1-9. Visualization of a Blockchain

### The Ethereum Blockchain

Ethereum is an open source, distributed software platform based on blockchain technology. It has its own native crypto-currency called Ether (ETH) and a programming language called Solidity. Blockchain is a distributed ledger technology that keeps a permanent, tamper-proof list of records. Ethereum is Bitcoin's main competitor.

The Ethereum platform offers the computationally complete Ethereum Virtual Machine (EVM). EVM executes scripts worldwide across its network of distributed public nodes. These nodes provide the processing power for decentralized applications developers create to run on the network. Developers may buy ETH to pay for the use of the network, or they can mine for the tokens themselves, becoming a part of the network. An internal mechanism called “Gas” sets the pricing of transactions on the network.

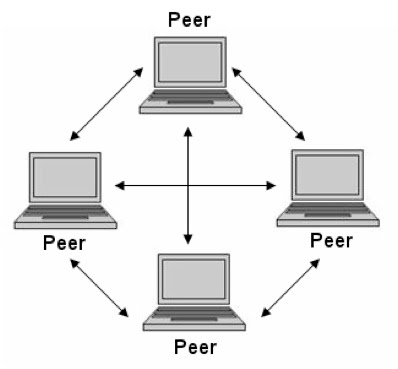


Figure 2.4.2-10. Peer-to-peer Network model with EVM

### Ethereum Platform

The blockchain used by Bitcoin proved to be too restrictive for more general use cases. This led to the birth of Ethereum, the first and most prominent smart contract platform. Ethereum is different from Bitcoin as it allows users to program and create their own operation instead of offering just a predefined set of options. The core component of this architecture is the EVM, which offers a sandbox environment where code of random complexity can be executed.

#### Account

The Ethereum global state consists of many entities that are communicating through a message passing framework. These entities are called accounts. Each account is identified by a unique 20-byte address and holds an internal state. There are two types of accounts in Ethereum: externally owned accounts (EOAs) and contract accounts.

An EOA is controlled by a private key, has no associated code, and can send transactions. A contract account has an associated code that executes when it receives a transaction from an EOA. A contract account cannot initiate transactions on its own. Transactions must always originate from an EOA.

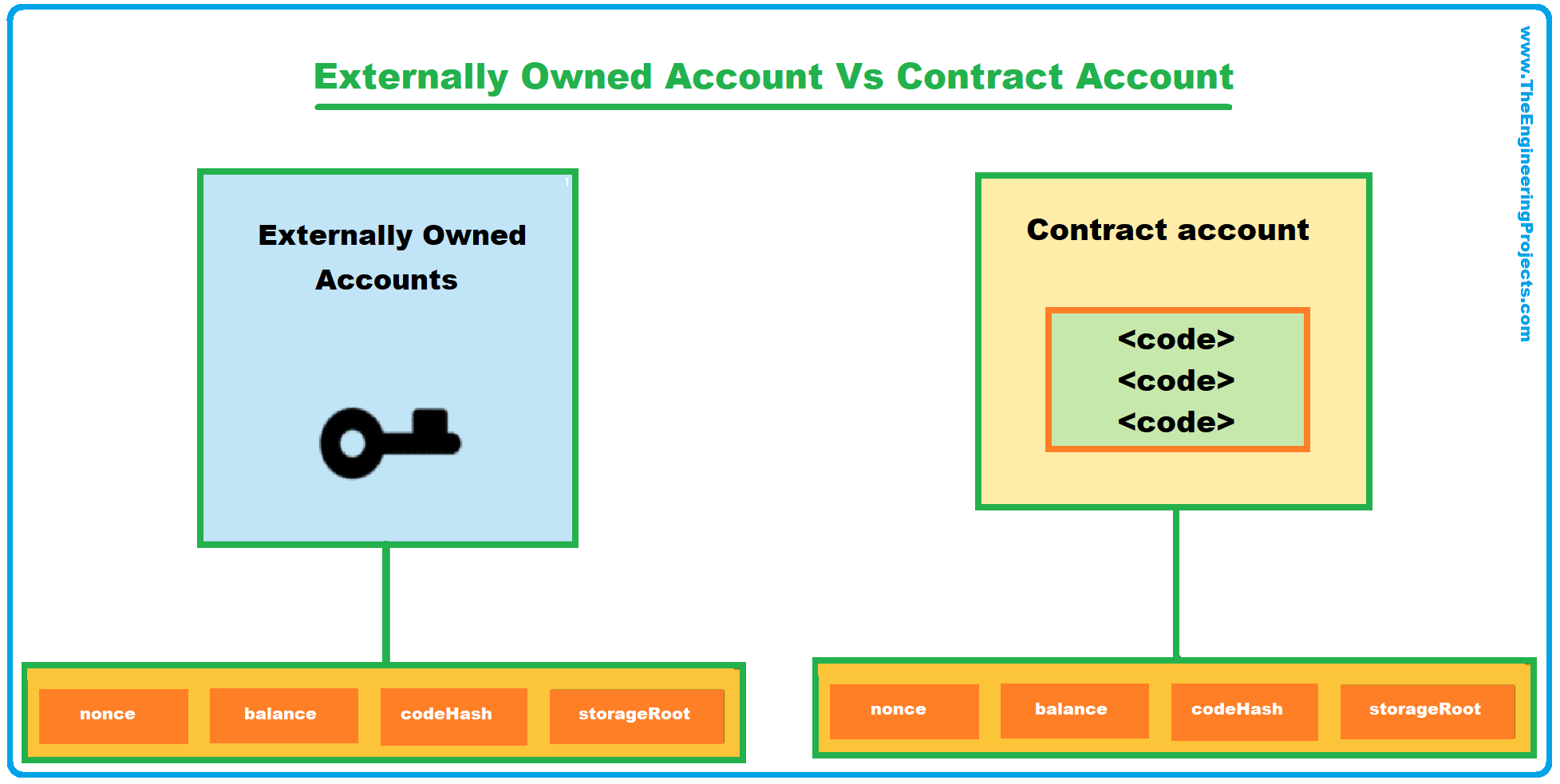


Figure 2.4.3.1-11. The different between EOAs and Contract Account

#### Transactions

Transactions are the starting point of any interaction which alters the blockchain state. I mentioned earlier that Ethereum can be viewed as a global state machine, only using transactions can this state change. A transaction can be thought of as being a single instruction which is created, cryptographically signed, serialized, and submitted to the blockchain by an EOA. There are two types of transactions: message calls and contract creations. When looking closer to the components of a transaction, you see the following fields can be found in both types:

* **nonce:** counter representing the number of transactions initiated by the sender, used for mitigating reply to attacks.
* **gasPrice:** the amount of Wei (1 ETH = 1018 Wei is the smallest Ethereum subsenomination) per gas unit the sender agrees to pay for the transaction execution.
* **gasLimit:** the maximum amount of gas that can be used for executing the transaction.
* **to:** the 20-byte address of the recipient. In case of contract creation, it is empty (zero).
* **value:** the number of Wei that will be deducted from the sender’s balance and transferred to the recipient address. In case of contract creation, an initial balance for the new smart contract will be set.
* **v, r, s:** these values correspond to the signature of the transaction and are used for identifying the sender.
* **init:** represents a byte array storing the code used to initialize the new contract account. This piece of code is run only once at contract creation and then is discarded. From its execution, another code fragment called body is returned. The body will be permanently linked to the contract account.
* **data:** a byte array containing the parameters of the message call. For example, a function from a smart contract might expect as parameter an integer representing an id.

#### Blocks

Relevant pieces of information form what we call a ‘block’ in the network. In Ethereum, a block contains a header, information about the transactions it includes and a set of other blocks’ headers (these blocks, called ommers, have the same parent as the current block’s parent’s parent).

Considering that, in Ethereum, blocks are added to the network much faster ( 15 seconds) than other blockchains (Bitcoin 10 minutes), more competing blocks are mined. Because only one of them can be added, the other blocks remain “orphaned”. A solution to also include these blocks in the main chain is for miners to add their header in their current block. Some of the more relevant components of a block’s header are:

* **parentHash:** the hash of the parent block’s header, this link makes the set of blocks a chain.
* **ommerHash:** the hash of the list of ommers added to the block.
* **beneficiary:** the account address of the block’s miner.
* **difficulty:** the difficulty level of the block.
* **number:** a counter for all the previous blocks starting from the genesis block which has the number 0.
* **gasLimit:** the current limit of gas per block.
* **gasUsed:** the total gas used by the transactions included in the block.
* **timestamp:** the Unix time at the block’s creation.
* **mixHash:** a 256-hash which, together with the nonce, shows that enough computation has been put into mining this block.
* **nonce:** a 64-bit value which is combined with mixHash.

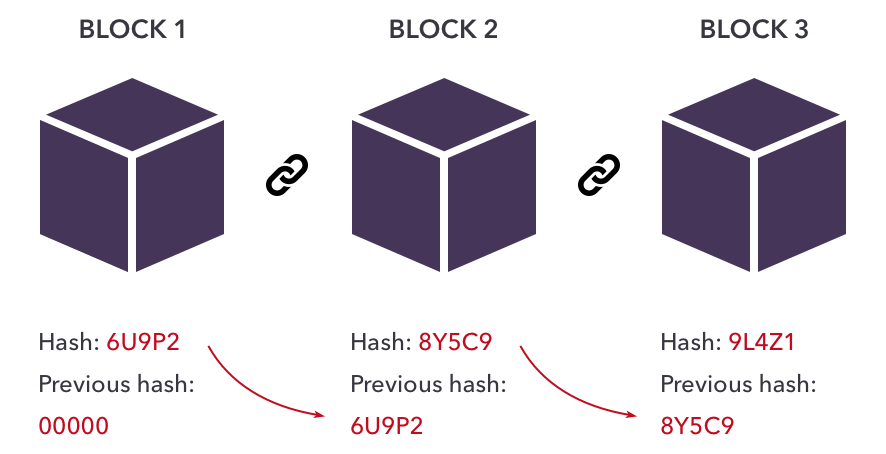


Figure 2.4.3.3-12. Blocks in Ethereum

#### Gas

Since transactions have to be run by all network nodes to be validated, a Turing-complete language can easily enable software bugs that could result in the transaction running indefinitely, commonly known as the halting problem. Whether accidentally or intentionally, an infinite loop in a smart contract would essentially result in a denial of service for the platform. To combat this issue, Ethereum introduced the concept of gas fees. Each instruction executed in the EVM has an associated cost measured in gas units.

When a transaction is created, two of the fields which need to be set are the **gasPrice** and the **gasLimit**. While the transaction is executed, the gas units for all instructions are summed up and multiplied with the specified gasPrice, resulting in the total gas fee. The **gasLimit** represents the maximum amount of computational steps the transaction can go through before it runs out of gas and stops. This mechanism ensures no transaction will run indefinitely as it becomes prohibitively expensive. The **gasPrice**, given in Wei, represents the price a user is willing to pay per unit of gas. It has a major impact on how quickly the respective transaction will be included in a block since miners give priority to the transactions having the highest **gasPrice**. In addition to being a metering mechanism, gas fees are an incentive for the miners as well since they are the ones collecting the fees.

#### Smart Contracts

The term “smart contract” was first defined by Nick Szabo in 1994 as “a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries.

Smart contracts are collections of code and data (or methods and state) which are deployed on the blockchain using the “contract creation” type of transaction. Since transactions have to be executed by each network node, all participants need to end up in the same state after the execution, meaning the smart contract code has to be deterministic.

To achieve this, smart contracts can only work with the data given as input. Data from outside the blockchain can be fed by oracles which are discussed in a later section. Before being deployed, smart contracts have to be compiled. From the compilation process, the most important artifacts are the bytecode and the interface. The smart contracts are compiled from the high-level language used by developers to machine code so that they can be run by every node in the EVM. As bytecode is not human-readable, developers need something in-between to allow them to interact with the deployed smart contracts.

This is achieved using the ABI (application binary interface) which defines a standard scheme (JSON format) for representing the smart contract code. Calls to deployed smart contracts are done using the ABI. Once deployed on the blockchain, the smart contract code cannot be altered and remains there as long as the network exists. Only the bytecode is stored on the blockchain, not also, the ABI.

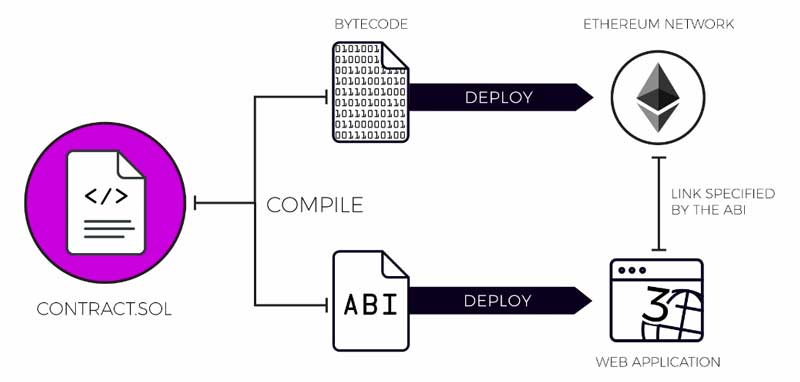


Figure 2.4.3.5-13. Smart Contract ABI

#### AMM Decentralized Exchanges

Automated market makers (AMM) are decentralized exchanges that pool liquidity from users and price the assets within the pool using algorithms. The exact mechanics vary from exchange to exchange, but generally, AMMs offer deep liquidity, low transaction fees, and 100% uptime for as many users as possible.

Traditional exchanges require buyers and sellers to meet at an overlapping price point on a centralized order book. In contrast, AMMs do quite a few things differently:

* Incentivize users in a process called yield farming to deposit crypto assets in liquidity pools.
* Use an algorithm, usually x \* y = k, to provide everyone trading with the pool a constant price.
* Automatically swap assets between traders and liquidity pools using smart contracts.

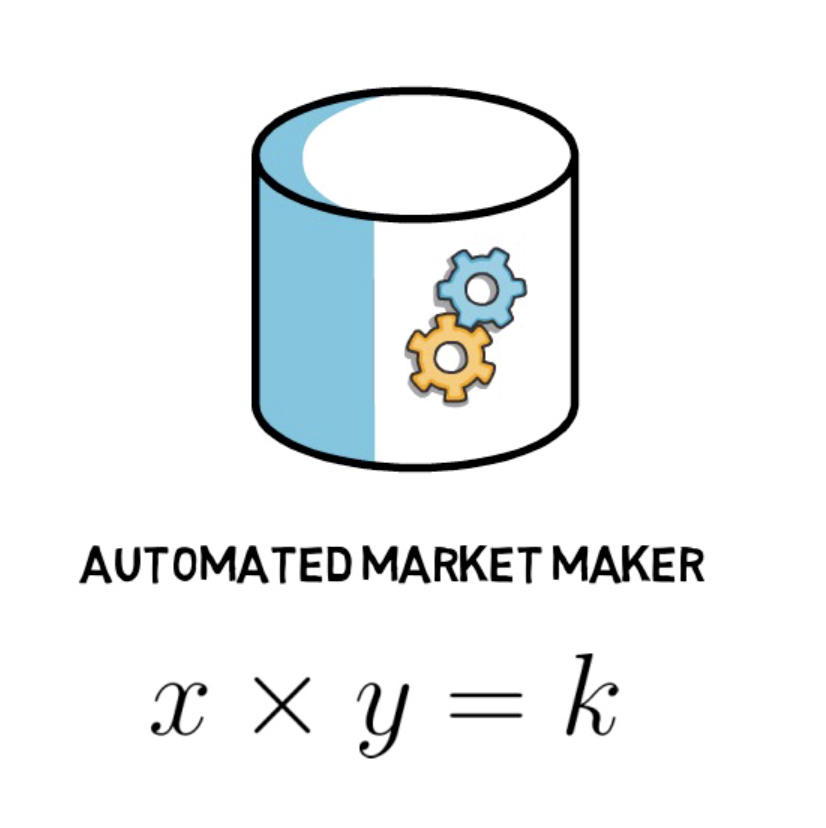


Figure 2.4.3.6-14. The algorithm x \* y = k

AMMs use a constant product formula to price assets, which states: x \* y = k. x and y are equal amounts of a liquidity pool’s assets while k is the total or constant amount of pool liquidity.

Uniswap is a prime example of how AMMs work. Uniswap is an AMM protocol that acts like a robot waiter serving up trades between you and a liquidity pool bootstrapped by liquidity providers (LPs). Under the AMM model, you can play several roles: trader, liquidity provider, and protocol governor. The protocol itself achieves two things: prices assets and executes trades via smart contracts.

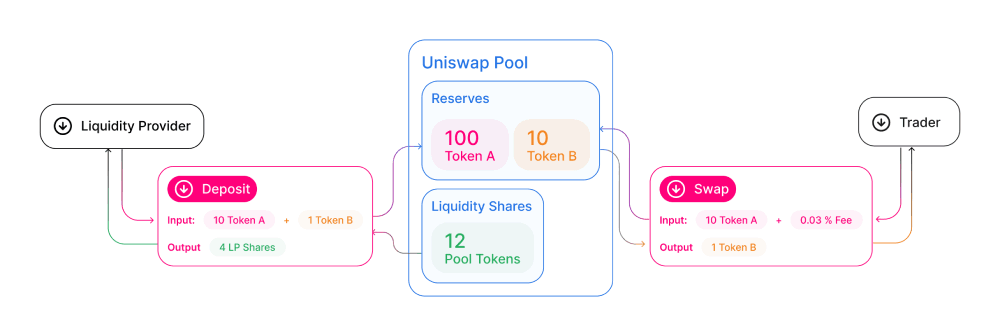


Figure 2.4.3.6-15. How Uniswap works

### Decentralized Finance

#### Decentralized Applications (dApps)

Decentralized applications (dApps) are a crucial element of the DeFi space. Unlike regular applications, dApps live on a smart contract platform like Ethereum. The main advantages of dApps over traditional software applications stem from the underlying blockchain infrastructure: permissionless nature and censorship-resistance. Anyone having an Ethereum wallet can interact with a dApp, as long as the smart contract conditions are met, once it has been deployed.

The problem dApps are trying to address is that being control over one’s data. When users use the centralized application such as Google services, their actions are likely being recorded on that platform. Hence, the centralized service providers can store that data and sell it to advertisers so that marketers could tailor ads to suit users’ preferences. For many people, it is an uncomfortable feeling to have their data treated in this manner. Ideally, data should remain oneself. dApps give users back control over their data, by ensuring that no single entity is in charge of any user’s data.



Figure 2.4.4.1-16. Application of dApps

#### Tokens

The Token Taxonomy Framework (TTF) was launched in 2019 by the Interwork Alliance [3]. The main purpose of the TTF is to establish a knowledge base for the token economy and, therefore, is an import step towards the Alliance’s mission to “empower organizations to adopt and use token-powered distributed services in their day-to-day business operations. The TTF is platform-agnostic, and it does not take any stand regarding the tokens implementation, as only the specification is considered.

Even though tokens can be created for different applications and purposes, there is a set of common features that all tokens share: valuable, representative, digital, discrete, and authentic. We say tokens are valuable because usually they can be evaluated in terms of a widely accepted standard, mainly the US dollar. By representative we refer to how tokens show the ownership or claim of someone to an asset, be it digital or physical. Because tokens live in the digital realm, usually recorded on blockchain, we say they are digital in nature. . The authenticity of tokens stems from the blockchain layer, being both public and permissionless, together with the consensus protocol, it enables us to verify the authenticity of each token in the same way we do for paper money.

#### Fungibility

Broadly speaking, there are two types of tokens: fungible and non-fungible. Fungible tokens are modelled after the fiat currencies we use in our everyday life. They can be divided depending on the declared number of decimals, with individual units being interchangeable and identical to each other, exactly like two newly minted bills of $1.

Meanwhile, non-fungible tokens were introduced to represent the ownership over a unique asset. One could draw a comparison to a piece of artwork, even if created by the same artist, no two pictures can be totally identical.

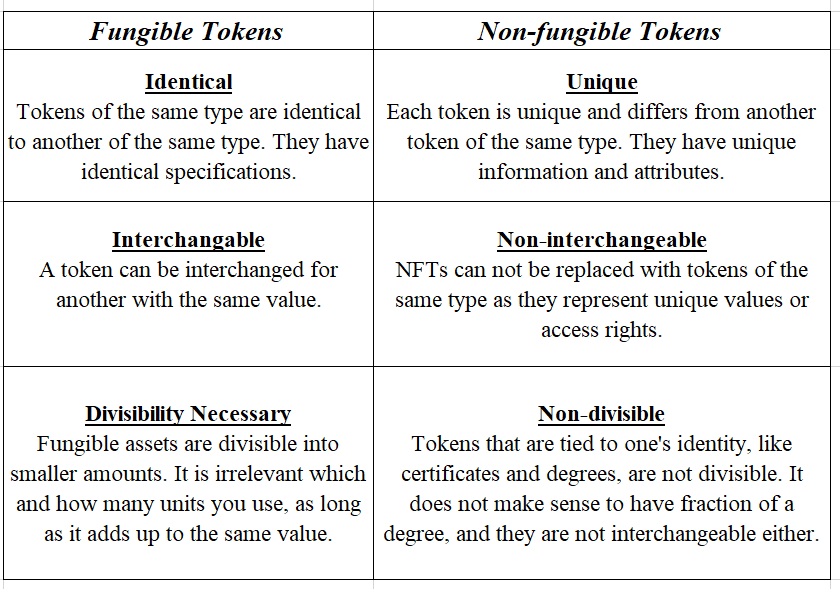


Table 1. The difference between Fungible Tokens and Non-fungible Tokens

#### ERC-20

As applications started to be built on Ethereum, the need for interoperability grew. The Ethereum community introduced Ethereum Improvement Proposals (EIPs) which are design documents describing standards for the platform. There are different types of EIPs, the most relevant type for this thesis is ERC (Ethereum Request for Comments) which defines “application-level standards and conventions, including contract standards.

These are interfaces providing a core set of functionalities which should be implemented by every token smart contract. The first such interface was ERC-20 for fungible tokens. Because from the interfaces the most common one in the DeFi space is the ERC-20, we will explore its core functionalities in more detail. The methods which all ERC-20 compliant token contract has to implement are the following:

* **totalSupply():** returns the total supply of the token.
* **balanceOf(owner):** returns the balance of the owner.
* **transfer(to, value):** transfers the value amount of tokens from the caller’s balance to the to address.
* **transferFrom(from, to, value):** allows contracts to transfer tokens on a user’s behalf since the amount is not deducted from the caller’s account balance, but from the from address. To not trigger an error, the user with the from address must have previously approved the caller to transfer a certain amount of tokens on his behalf.
* **approve(spender, value):** allows the spender to transfer from the caller’s balance up to the value amount.
* **allowance(owner, spender):** returns the amount the spender can withdraw from the owner’s account.

#### Liquidity Pool

A liquidity pool is a collection of funds locked in a smart contract. Liquidity pools are used to facilitate decentralized trading, lending, and many more functions we will explore later.

Liquidity pools are the backbone of many DEX, such as Uniswap. Users called liquidity providers (LP) add an equal value of two tokens in a pool to create a market. In exchange for providing their funds, they earn trading fees from the trades that happen in their pool, proportional to their share of the total liquidity. As anyone can be a liquidity provider, AMMs have made market making more accessible.

#### Impermanent Loss

Impermanent loss happens when you provide liquidity to a liquidity pool, and the price of your deposited assets changes compared to when you deposited them. The bigger this change is, the more you are exposed to impermanent loss. In this case, the loss means less dollar value at the time of withdrawal than at the time of deposit. impermanent loss can still be counteracted by trading fees. In fact, even pools on Uniswap that are quite exposed to impermanent loss can be profitable thanks to the trading fees.

Uniswap charges 0.3% on every trade that directly goes to liquidity providers. If there is a lot of trading volume happening in a given pool, it can be profitable to provide liquidity even if the pool is heavily exposed to impermanent loss. This, however, depends on the protocol, the specific pool, the deposited assets, and even wider market conditions.

#### Metamask Extension

MetaMask is one of the leading crypto wallets and relies on browser integration and good design to serve as one of the main gateways to the world of Web3, DeFi, and NFTs.

MetaMask is a browser plugin that serves as an Ethereum wallet and is installed like any other browser plugin. Once it is installed, it allows users to store Ether and other ERC-20 tokens, enabling them to transact with any Ethereum address.

By connecting to MetaMask to Ethereum-based dapps, users can spend their coins in games, stake tokens in gambling applications, and trade them on decentralized exchanges (DEXs). It also provides users with an entry point into the emerging world of decentralized finance, or DeFi.

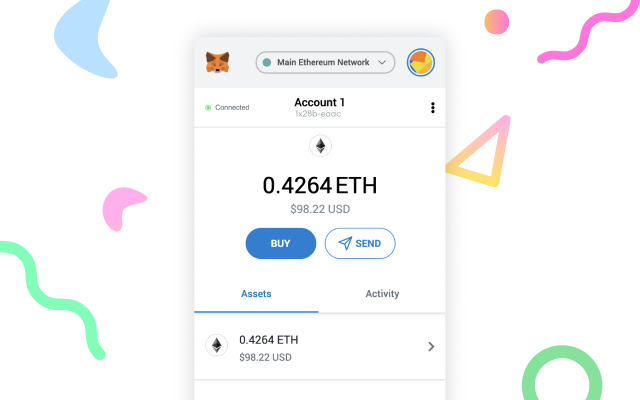


Figure 2.4.4.7-17. MetaMask Wallet

MetaMask injects web3 object and convenience Web3.js library into the JavaScript context. It helps user to handle authorization using their private key. Any time users make a transaction that requires private key to sign the transaction, MetaMask will automatically prompt the user for permission, and then forward the signed request to the blockchain.

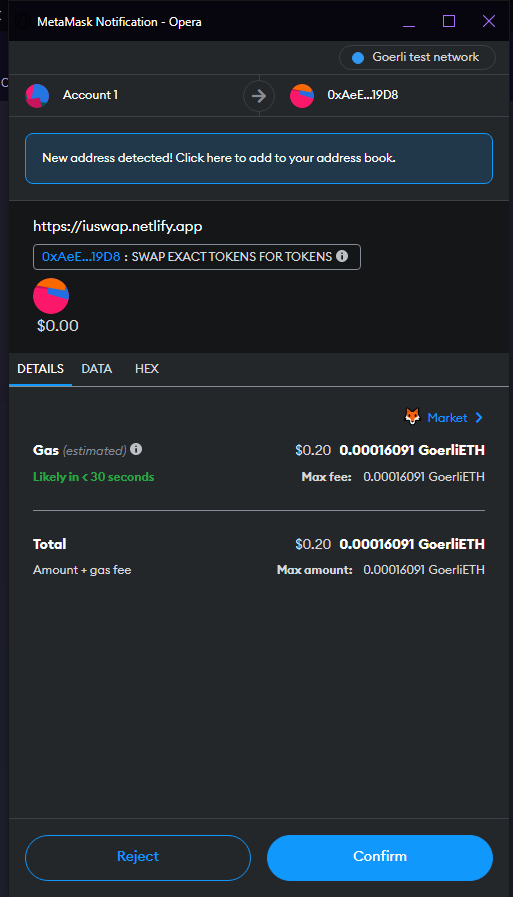


Figure 2.4.4.7-18. MetaMask approves transaction

## CRANQ (Back-End)

CRANQ is a graphical and intuitive IDE that allows users to compile and deploy their smart contracts. It has a graphical interface, where users can visually track how the code is being executed by looking at its flow. Its focus on standardized datatypes and ports means that intent can be easily checked.

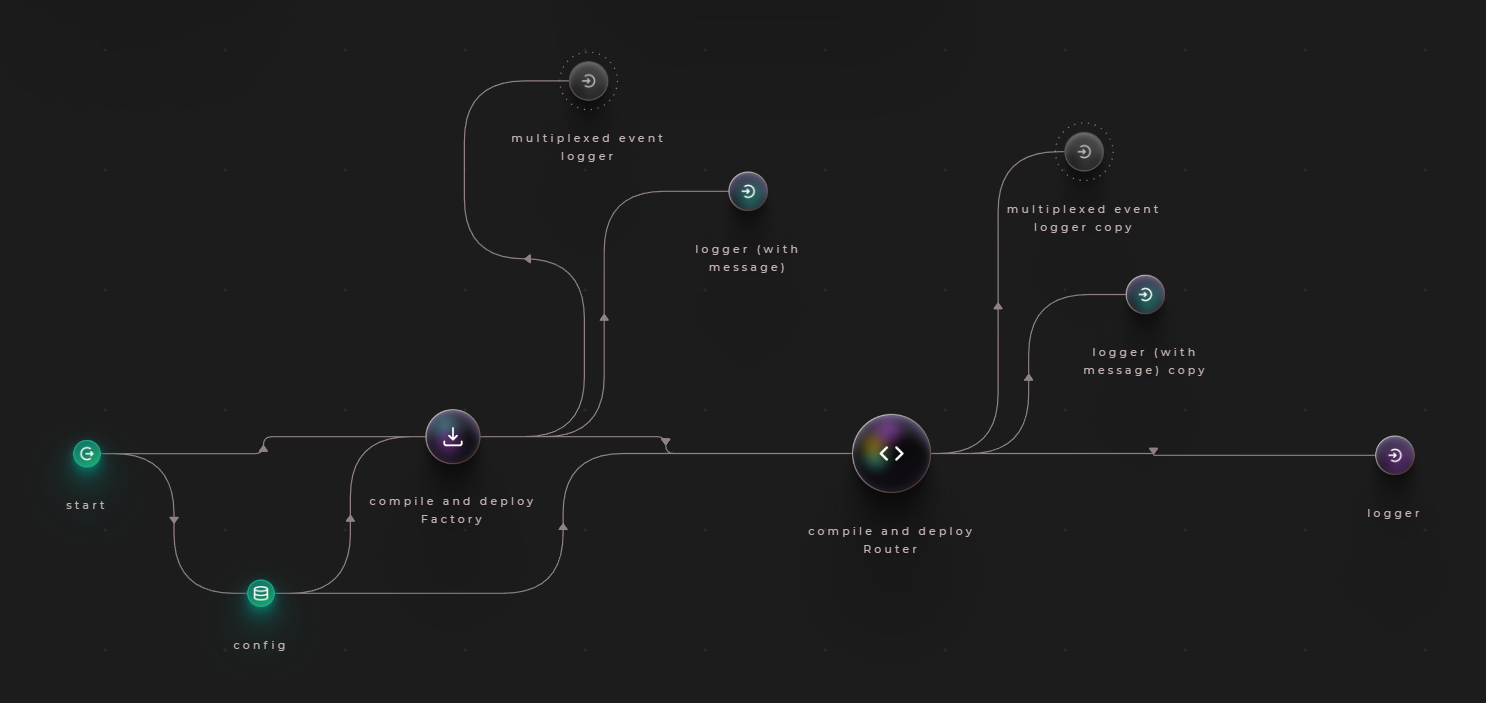


Figure 2.5-19. Creating a new factory contract using CRANQ

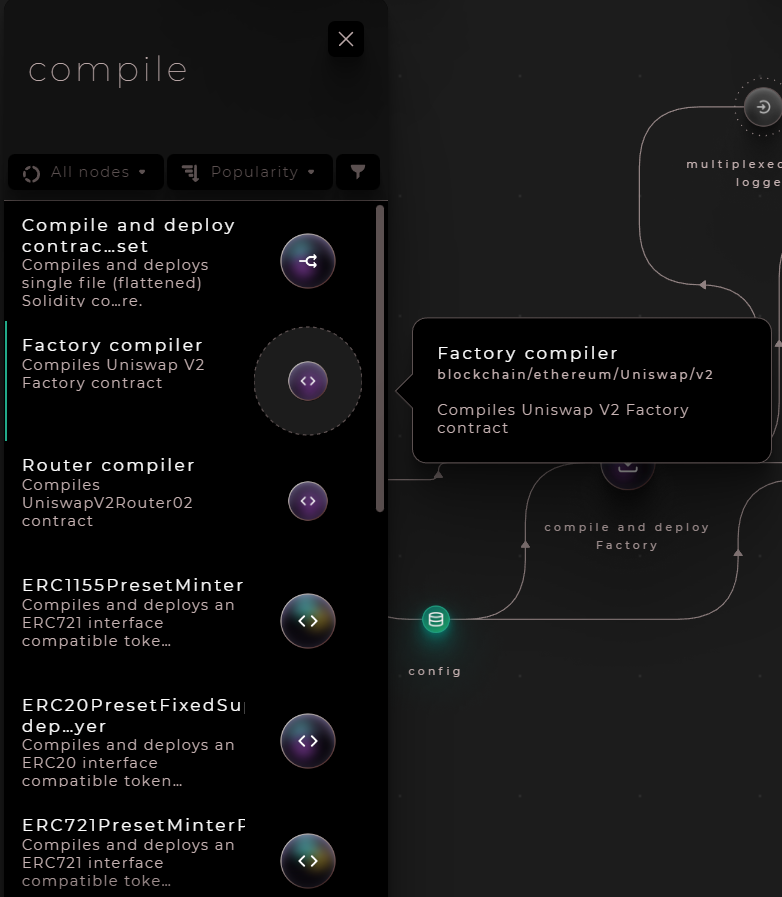


Figure 2.5-20. Every node with details in CRANQ

# CHAPTER 3

# METHODOLOGY

## System Description

ERC-20 will be the token standard for tokens in the system. In this thesis paper, system will be first launched on Goerli Network, which is an Ethereum test network for development and testing purpose before being deployed to Mainnet.

To exchange tokens, users can store their tokens in a crypto wallet such as MetaMask. All the transactions of tokens, smart contracts created can be track through Etherscan - a block explorer for the Ethereum blockchain.

User Interface (UI) must be clear, simple, intuitive, and responsive to help users finish the task with less effort and time, thus Single Page Application (SPA) is applied for Client Side in this thesis paper.

## System Use Case

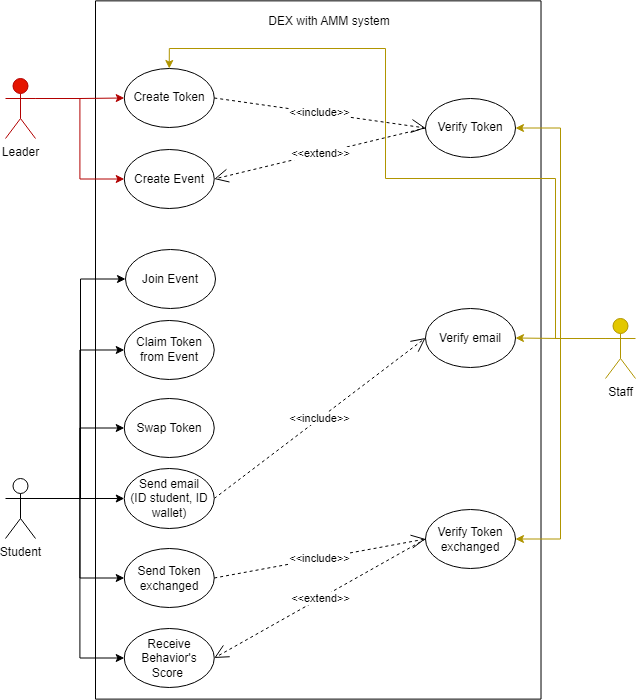


Figure 3.2-1. System Use Cases

## System Functionality

The functionality of the system is described as below

|  |  |  |
| --- | --- | --- |
|  | **Function** | **Description** |
| **Front-End** | Claim Token | Allow user claim token after joinning an event |
| Connect Wallet | Allow user to connnect their MetaMask wallet |
| Select Tokens to exchange | User can exchange many tokens from different organizations to school’s token |
| Approve the swap | User can check the amount of token exchanged before approving the swap |
| Swap Tokens | User start swapping two tokens and claim the amount of token exchanged |
| **Back-End** | Compile smart contracts | Compile all Ethereum contracts |
| Deploy smart contracts to Ethereum test net | Deploy the contract to Goerli network, print the address of contract |
| Create Tokens | Create two kind of tokens: one can be mintable and one has the initial amount |
| **Ethereum Blockchain** | Manage all Exchange’s contracts | Approve, swap two tokens are forward to Etherscan |
| Verify smart contracts | Check for user’s permission when performing any transactions. |

Table . System’s Functional Description

## System Workflow

# CHAPTER 4

# IMPLEMENTATION

## Configuration

### NodeJS and NPM

NodeJs is the most commonly used platform to develop applications. It supports asynchronous programming, single-threaded which can manage huge traffic and high scalability. Both NodeJS and NPM can be installed on local machine via <https://nodejs.org/en/download/>. After installing, open you command and run these following commands:

* <node -v>
* <npm -v>

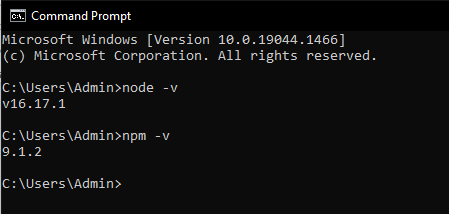


Figure 4.1.1-1. Node and NPM version

Now all packages in “package.json” file can be easily dowloaded and we can start running scripts.

### CRANQ

As I mentioned before, CRANQ will be used for compiling and deploying smart contracts. Simply go to <https://cranq.io> and download the software from the website. After installing, open your first project in CRANQ and rename it into “IUSwap”.

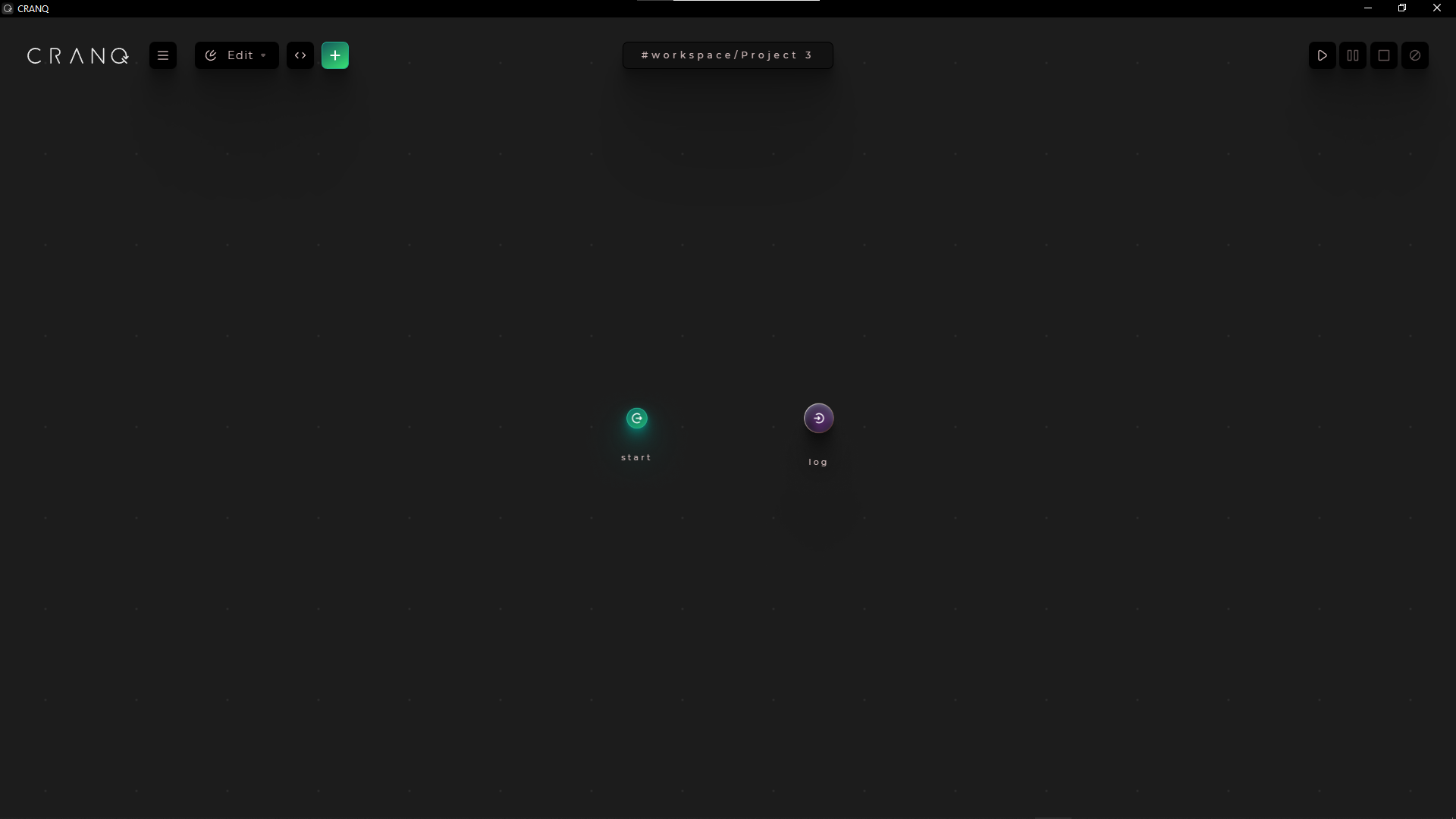


Figure 4.1.2-2. CRANQ first screen

The two circles you see on the screen right now are nodes. Every node has the Input and the Output. CRANQ also has a huge library with hundred of different nodes, you can simply access to them by clicking icon “+” on the top left.

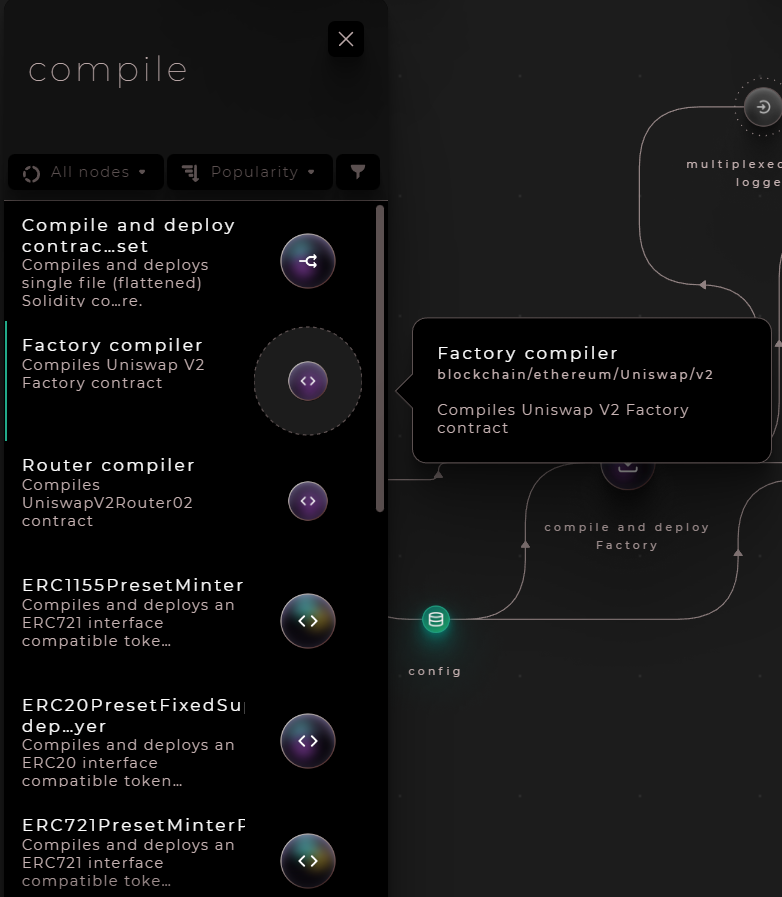


Figure 4.1.2-3. Library of CRANQ

### MetaMask

MetaMask is the most popular crypto wallet and used by million of people. With metamask, user can interact directly with the entire Ethereum ecosystem, which is the foundation of many decentralized applications, without local installation. It is compatible with mostly nowadays browsers such as Chrome, Edge, Firefox. It also supports cross-platform storage which is both on mobile and browser.

To setup MetaMask, let navigate to <https://metamask.io>. Then select “Download” button for browser > Add to Google Chrome > Add extension > Done.



Figure 4.1.3-4. MetaMask icon in browser

Click on Metamask icon and keep following user guide of Metamask to create an account. There will be 12 words private key, you need to save it for using later. Remember to switch network to “Goerli test network”. After reaching the main screen as below:

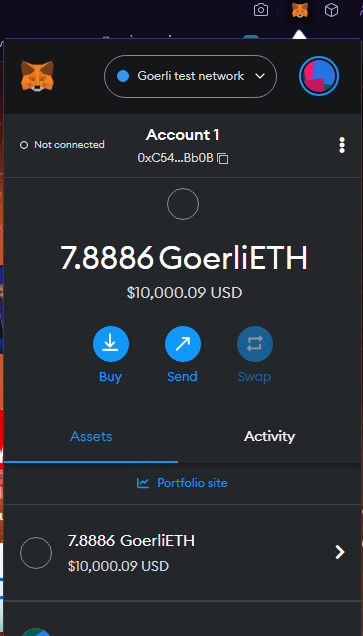


Figure 4.1.3-5. MetaMask Homescreen

### Alchemy

Alchemy is the Web3 development platform, allows us to actually deploy smart contract on Blockchain. Alchemy is a free service that has all the core products to eliminate that pain point with user-friendly documentation. We can sign up for free by following <https://www.alchemy.com> to create a new account.

After successfully logging in, you will have to create a new dApp in order to generate API key and provider URL for using later in CRANQ.

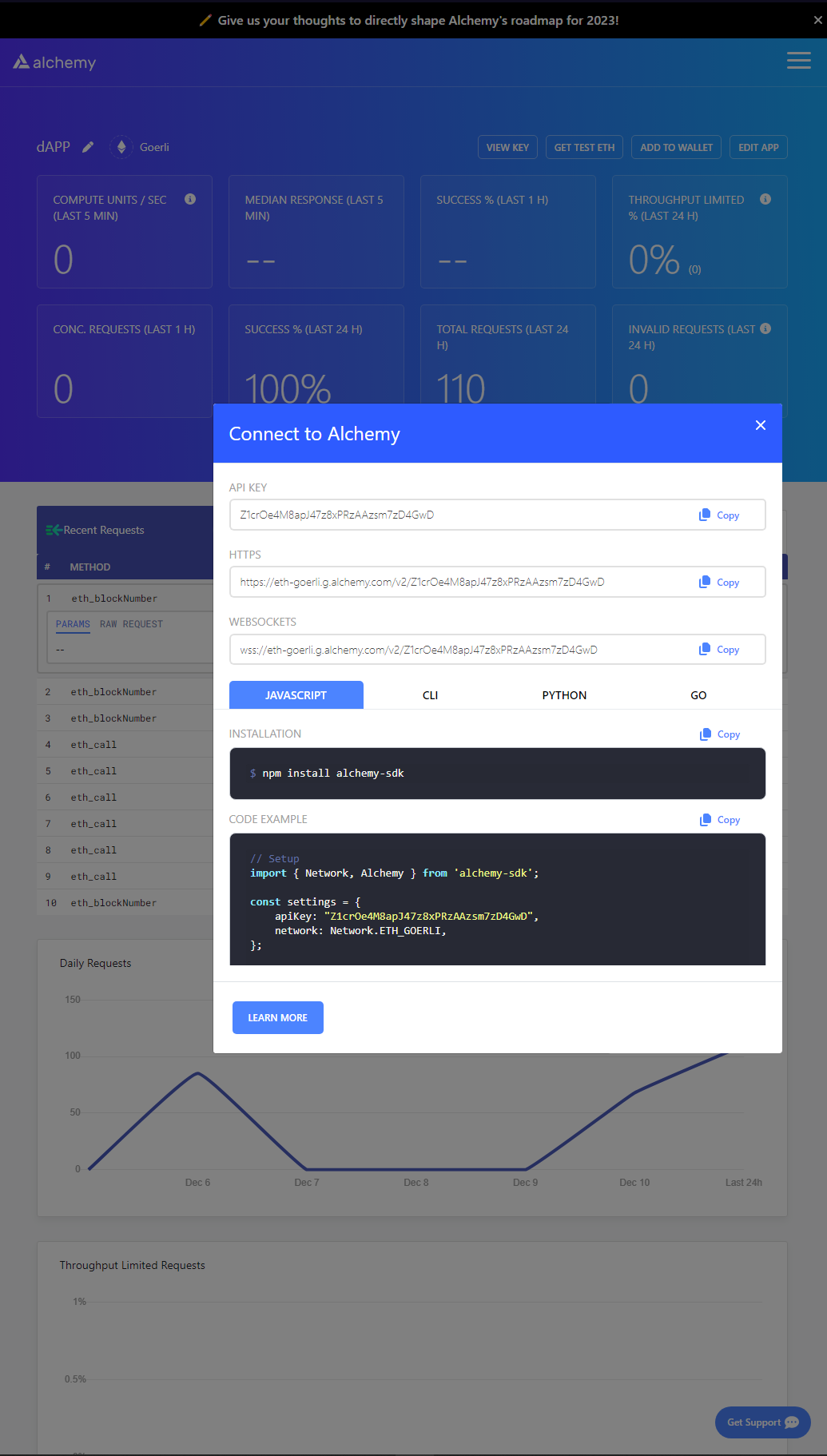


Figure 4.1.4-6. API key and provider URL

Alchemy also provides free 0.5 Goerli ETH for testing purpose. Go to <https://goerlifaucet.com>, copy and paste your ID wallet in the box, click “Send me ETH” to claim 0.5 Goerli ETH every 24 hours.

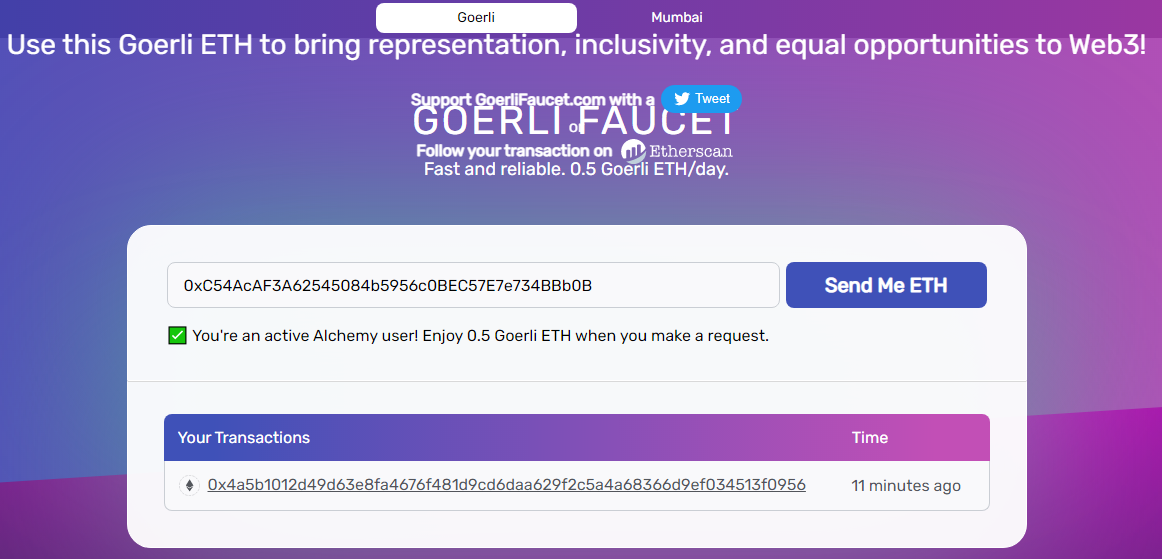


Figure 4.1.4-7. Alchemy provides free 0.5 Goerli ETH

## Implementation

In this part, I will neither show the details, nor else, fully explain the source code. But rather gives readers an abstract view of code structure and the functionality of each module or component specified in this thesis paper. Please visit my Github to access the fully documented paper and the source code at: <https://github.com/MiQannn/IUSwap>.

### Back-End

#### Remix

Remix, more commonly known as Remix IDE (Integrated Development Environment), is an open-source Ethereum IDE can use to write, compile and debug Solidity code. As such, Remix can be a hugely important tool in Web3 and dApps development.

In this thesis, I will use Remix to generate a token mintable for an organization called Information Technology Youth Union (ITYU). I use Solidity programming language to call out the contract to create new ERC-20 token name “ITYU Token” with “ITYU” symbol, and apply function mint() to set the token is mintable.

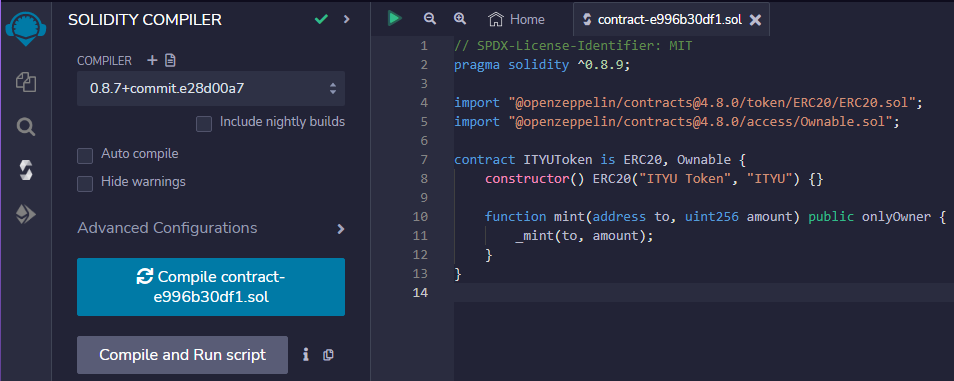


Figure 4.2.1.1-8. Using Remix to compile contract creating a mintable ITYU token

After compiling contract successfully, we have to deploy the contract to Goerli Test Network following these steps:

* You need to change the “Environment” option to “Injected Provider – MetaMask”, it will ask you to connect to your MetaMask wallet.
* Then click the button “Deploy”, MetaMask will pop up and ask if you accecpt the fee to create a new token. Press “Confirm” and wait for the contract deploys.

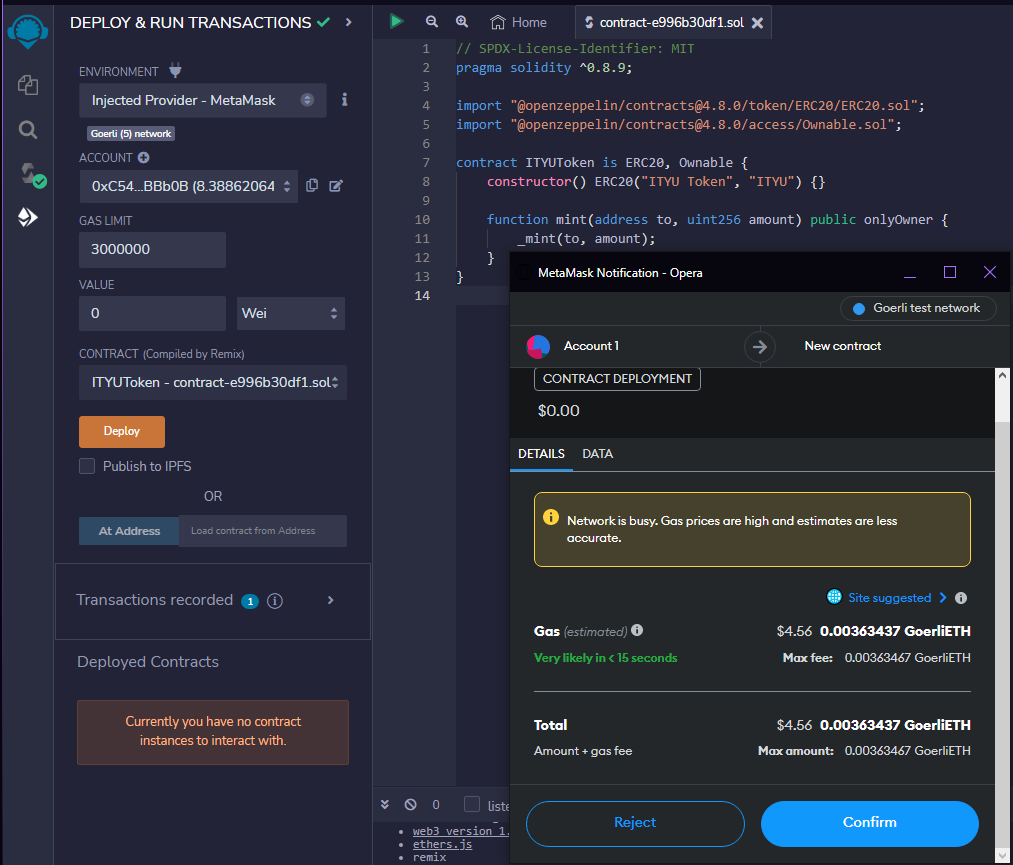


Figure 4.2.1.1-9. Remix will ask for small fee to create a new token

When you finish deploying contract, look at the “Deployed Contracts”, there will be an address for you to check on Etherscan: <https://goerli.etherscan.io>. Here is the result of new contract of ITYU token:

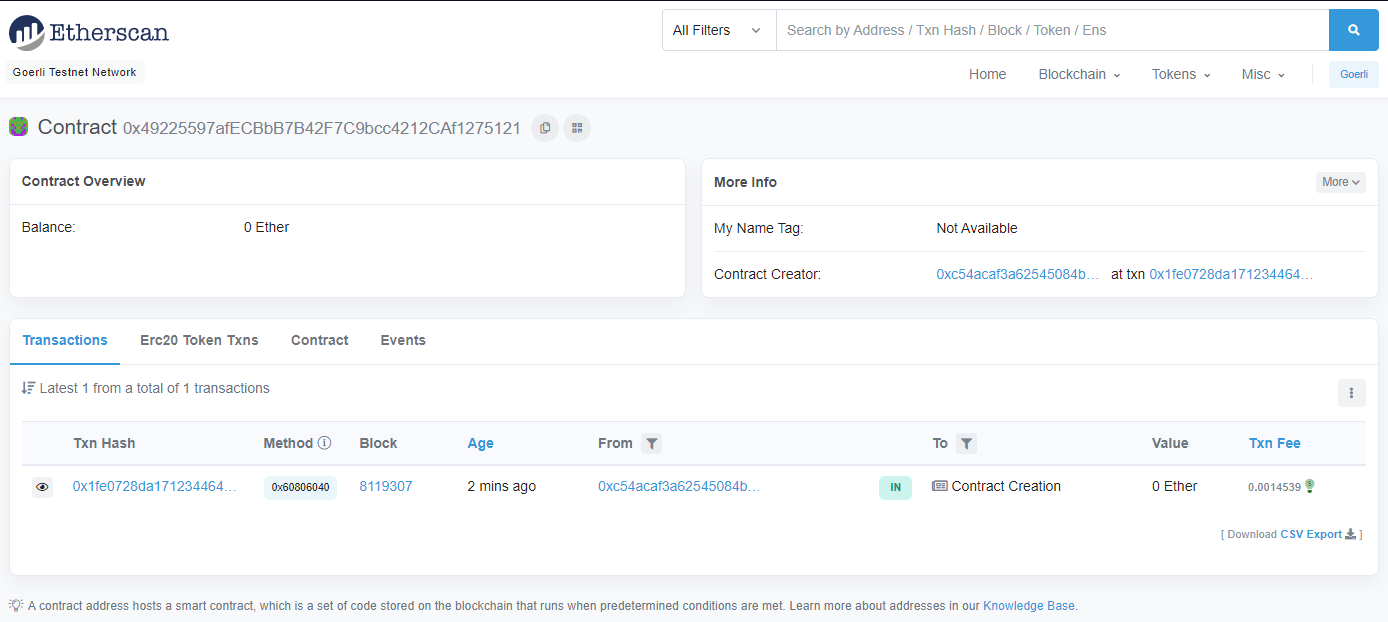


Figure 4.2.1.1-10. New ITYU token

Finally, go to your MetaMask wallet > Choose “Import tokens” > Copy and paste new address of ITYU token > Choose “Add custom token” > Done.

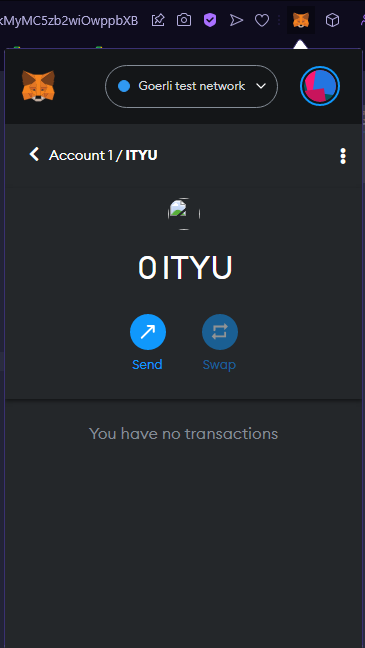


Figure 4.2.1.1-11. ITYU token

#### SmartContracts Tools

Next, I will create a new ERC-20 token for IU behavior’s score. This time I will use SmartContracts Tools (<https://www.smartcontracts.tools/token-generator/create/ethereum/>). SmartContracts Tools is a website easily deploy Smart Contract for a Standard ERC20 Token, no login, no setup, and no coding required. All you need to do is fill in token details and choose a network you want to deploy. In here, I will choose:

* Token Type: Simple ERC20
* Token Name: IU Token
* Token Symbol: IUC
* Initial Supply: 100
* Network: Goerli – Testnet

MetaMask will pop up and ask you to connect to this website, just choose “Next” > “Connect” and wait for creating new ERC-20 token.

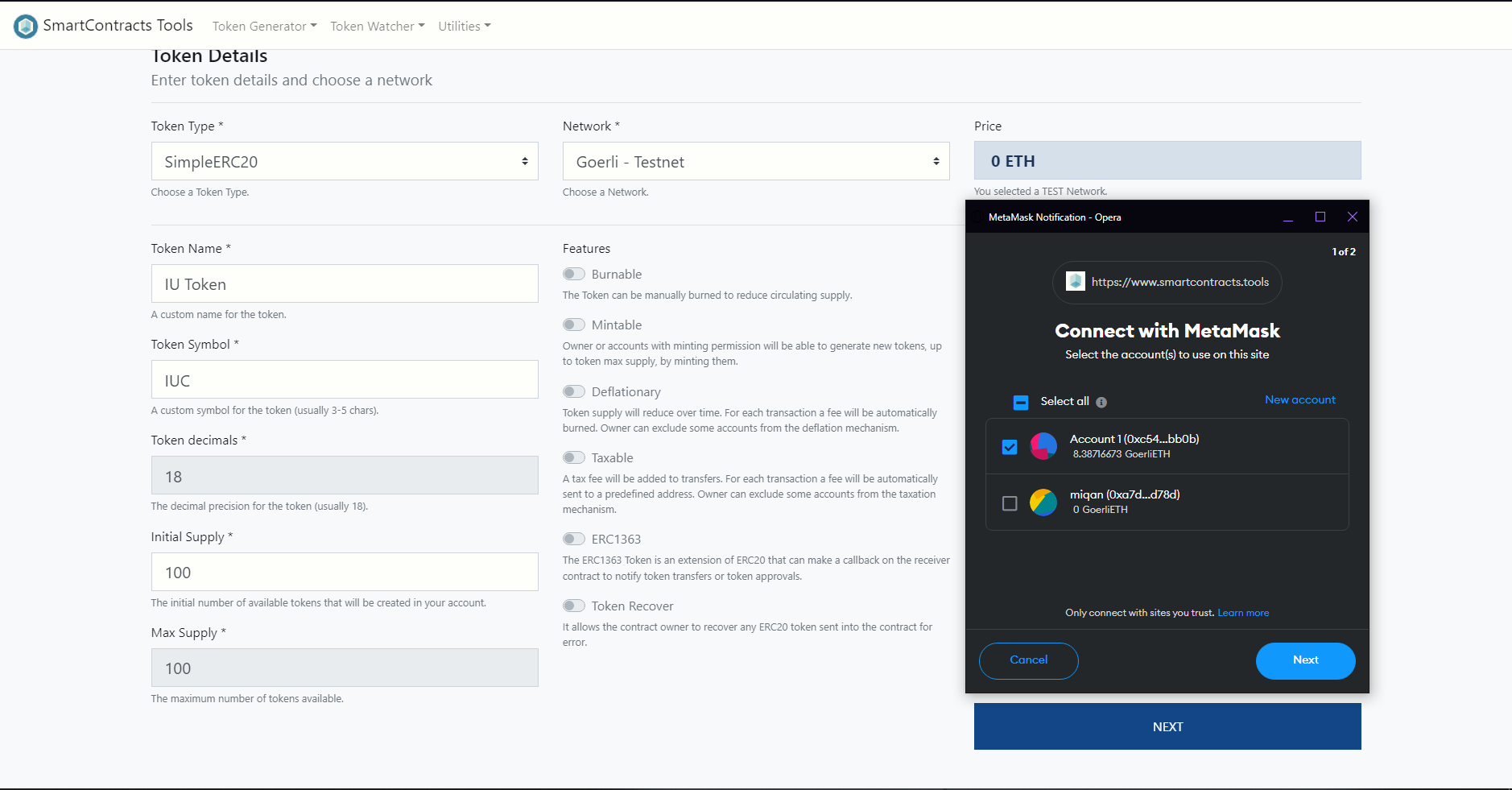


Figure 4.2.1.2-12. Create new IUC Token

When finishing all the steps, choose “Add to MetaMask” and now we have 100 IUC. Due to students need to exchange between tokens they earned from ITYU to IU behavior’s score, we need 100 tokens available for exchange.

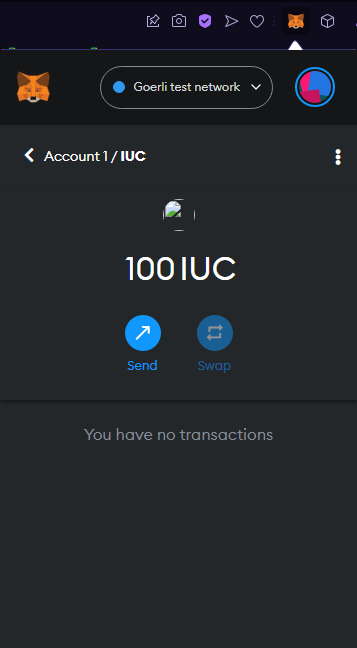


Figure 4.2.1.2-13. IUC Token

### CRANQ Smart Contracts

#### Factory Contract

A Factory Contract is a smart contract that produces other Smart Contracts. This is a useful pattern for many reasons. One reason is it allows you to create multiple instances of the same contract, much like a class in programming works. Define it once, and then you can create new instances of said class anywhere you would like. You are able to track all of the contracts that a factory has deployed if you so choose. It can even save you on gas, as you can deploy the factory, and then use it to deploy other smart contracts.

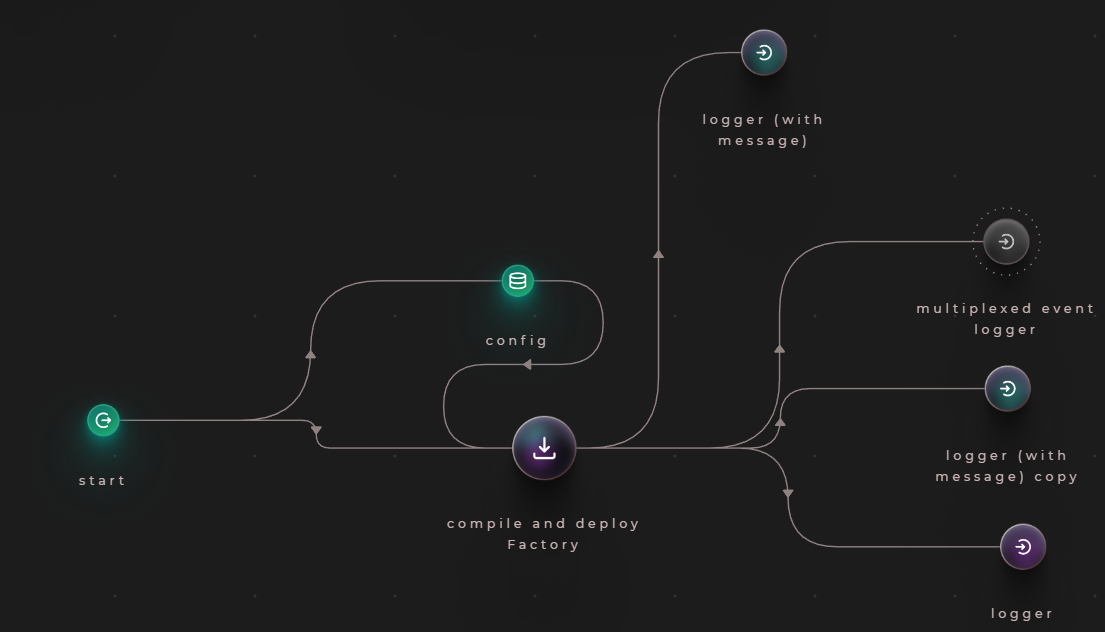


Figure 4.2.2.1-14. Factory Contract in CRANQ

From Figure 4.2.2.1-14., the Factory Contract contains many nodes:

* **start:** use to kick-start the entire program.
* **config:** store data and send it to other nodes, it includes:
  + **privateKey:** your private key from MetaMask
  + **accountAddress:** your ID wallet
  + **providerUrl:** HTTPS from dApp we created on Alchemy
  + **network:** network we want to work with
* **compile and deploy Factory:** compile and deploy Uniswap V2 Factory contract
* **logger (with message):** print out errors
* **mutiplexed event logger**
* **logger:** logs received data and tag to the output, formats data as JSON.

#### Router Contract

The Router Contract supports the basic functionalities performed for interacting with the Pair Contracts. For better modularity, a Factory Contract is added to hold the underlying logic that handles the Pair Contracts.

The Router Contract provides access to the Factory Contract through a series of endpoints and view functions. Notably here are the createPair and upgradePair endpoints. The first one will be used to create a new Pair Contract when there is no address associated with the two tokens intended to be used for the DEX activity. This endpoint is used only once per Pair creation. The latter endpoint will be used when a new version of a Pair Contract will be developed and be a need to upgrade a Pair Contract implementation. The Pair Contract will be use later in Liquidity Contract.

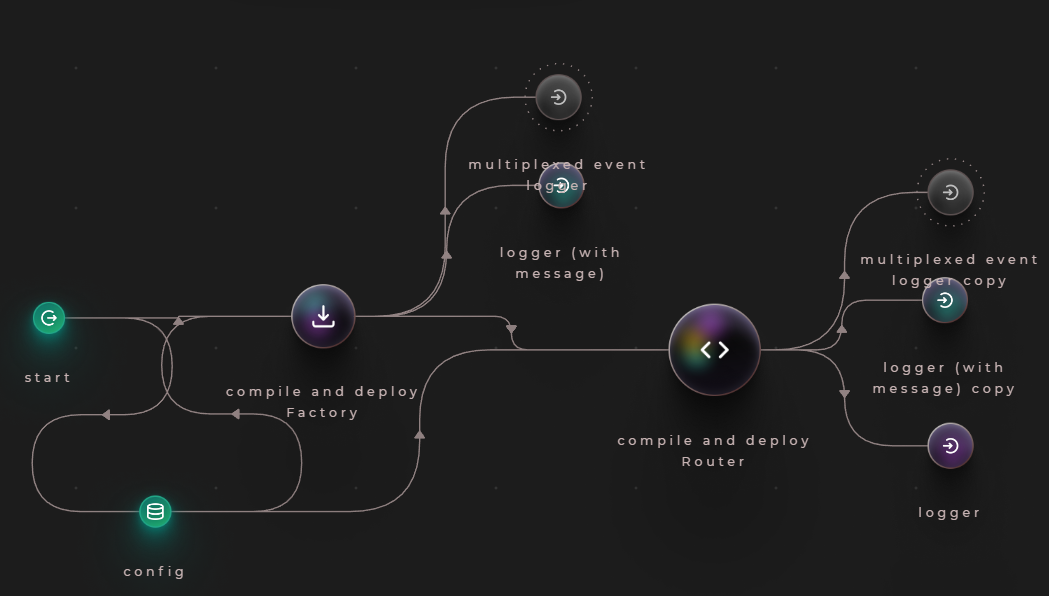


Figure 4.2.2.2-15. Router Contract in CRANQ

From Figure 4.2.2.1-15., the Router Contract contains one node:

* **compile and deploy Router:** compile and deploy UniswapV2Router02 contract.

#### Liquidity Contract

Without liquidity, no exchange can work. A **Liquidity Contract**is a smart contract where tokens are locked for the purpose of providing liquidity. When a user provides liquidity, a smart contract issues liquidity pool tokens (LPTs). These tokens represent the liquidity provider's share of assets in the pool.

Unlike traditional exchanges that use order books, the price in a DEX is typically set by an Automated Market Maker (AMM). When a trade is executed, the AMM uses a mathematical formula to calculate how much of each asset in the pool needs to be swapped in order to fulfill the trade.

I will set the rule: 10 ITYU Tokens are equal to 5 IUC Tokens.

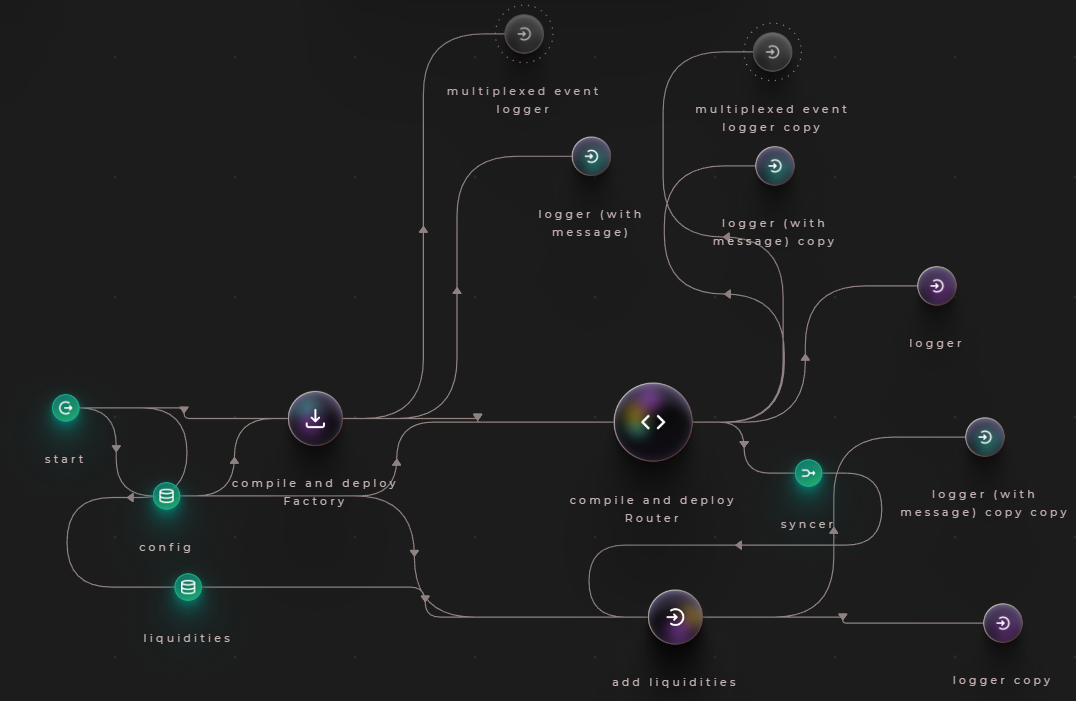


Figure 4.2.2.3-16. Liquidity Contract in CRANQ

From Figure 4.2.2.3-16., the Router Contract contains three nodes:

* **liquidities:** store liquiditiy pairs of token users want to exchange, it includes:
  + **address:** address of token.
  + **desireAmount:** initial amount of tokens when comparing to other tokens.
* **syncer:** combination of “router address” and “router ABI” from **compile and deploy Router** node.
* **add liquidities:** add liquidities to multiple pools.

After finishing set up CRANQ, press “Run” button on the top right and wait for the contract compile. There will be a contract address at the OUTPUT, copy and save it for later.

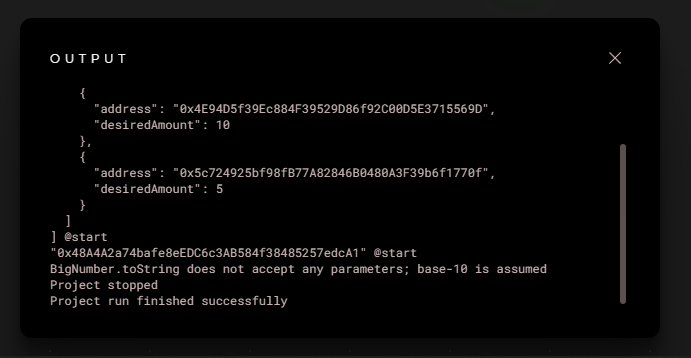


Figure 4.2.2.3-17. Smart Contracts created in CRANQ

### Front-End

#### Airdrop

This page is for rewarding students, after the event they joined. First of all, you need to choose Folder direction, then run <npx create-next-app airdrop>. File structure inside your project is illustrated as the Figure below

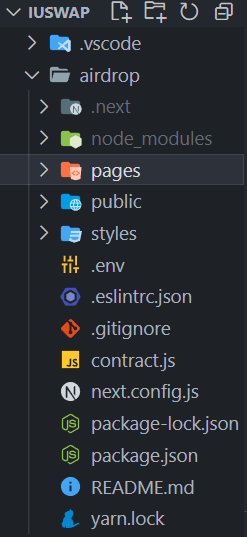


Figure 4.2.3.1-18. Project airdrop structure

contract.js is the place to put tokenAddress and tokenABI we have created from Remix, so we know which tokens to be given to students.



Figure 4.2.3.1-19. ITYU’s tokenAddress and tokenABI

In folder pages > api > airdrop.js, we will set up the environment to deploy the contract. In line (17), I want student can only claim 5 ITYU tokens so I set the value in ether.utils.parseEther() to “5”.



Figure 4.2.3.1-20. Main function in airdrop

#### IU Swap

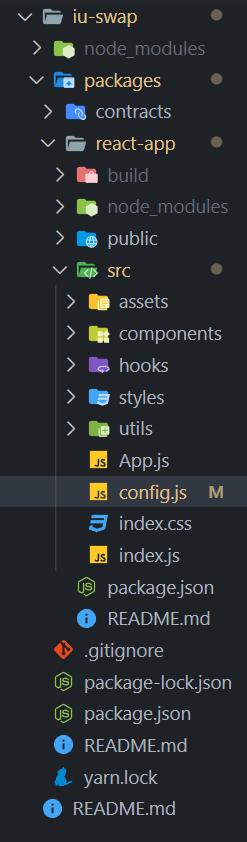


Figure 4.2.3.2-21. Project DEX structure

# CHAPTER 5

# RESULT AND EVALUATION

## Discussion

Abc…

## Comparison

Abc…

## Evaluation

Abc…

# CHAPTER 6

# CONCLUSION AND FUTURE WORK

## Conclusion

Abc...

## Future work

Abc…

# REFERENCES

|  |  |
| --- | --- |
| [1] | "What is Web3?," [Online]. Available: https://ethereum.org/en/web3/. |
| [2] | "What is React.js?," [Online]. Available: <https://blog.hubspot.com/website/react-js>. |

# APPENDIX