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

My appreciation to my professor, Dr. Tran Thanh Tung, for his tremendous patience and feedback is beyond words. In addition, I could not have made this voyage without the rich knowledge and experience my defense committee supplied. I am also appreciative of my friends' assistance with editing, late-night critique sessions, and moral support.

# TABLE OF CONTENTS

[ACKNOWLEGMENTS 3](#_Toc121771927)

[TABLE OF CONTENTS 4](#_Toc121771928)

[LIST OF TABLES 6](#_Toc121771929)

[LIST OF FIGURES 7](#_Toc121771930)

[LIST OF ABBREVIATIONS 9](#_Toc121771931)

[ABSTRACT 10](#_Toc121771932)

[CHAPTER 1 11](#_Toc121771933)

[INTRODUCTION 11](#_Toc121771934)

[1.1. Background 11](#_Toc121771935)

[1.2. Problem Statement 11](#_Toc121771936)

[1.3. Objectives 12](#_Toc121771937)

[1.4. Assumption 12](#_Toc121771938)

[CHAPTER 2 13](#_Toc121771939)

[BACKGROUND/APPLIED TECHNOLOGIES 13](#_Toc121771940)

[2.1. Overview of Web 3.0 13](#_Toc121771941)

[2.2. ReactJS with Tailwind CSS (Front-End) 14](#_Toc121771942)

[2.2.1. ReactJS 14](#_Toc121771943)

[2.2.2. Tailwind CSS 15](#_Toc121771944)

[2.3. NodeJS (Running Environment) and NPM 17](#_Toc121771945)

[2.3.1. NodeJS 17](#_Toc121771946)

[2.3.2. NPM 18](#_Toc121771947)

[2.4. Overview of Ethereum Blockchain technologies 19](#_Toc121771948)

[2.4.1. Blockchains Explained 19](#_Toc121771949)

[2.4.2. The Ethereum Blockchain 20](#_Toc121771950)

[2.4.3. Ethereum Platform 21](#_Toc121771951)

[2.4.4. Decentralized Finance 28](#_Toc121771952)

[2.5. CRANQ (Back-End) 34](#_Toc121771953)

[CHAPTER 3 37](#_Toc121771954)

[METHODOLOGY 37](#_Toc121771955)

[3.1. System Description 37](#_Toc121771956)

[3.2. System Use Case 38](#_Toc121771957)

[3.3. System Functionality 39](#_Toc121771958)

[3.4. System Workflow 40](#_Toc121771959)

[CHAPTER 4 41](#_Toc121771960)

[IMPLEMENTATION 41](#_Toc121771961)

[4.1. Configuration 41](#_Toc121771962)

[4.1.1. NodeJS and NPM 41](#_Toc121771963)

[4.1.2. CRANQ 41](#_Toc121771964)

[4.1.3. MetaMask 44](#_Toc121771965)

[4.1.4. Alchemy 45](#_Toc121771966)

[4.2. Implementation 47](#_Toc121771967)

[4.2.1. Back-End 47](#_Toc121771968)

[4.2.2. CRANQ Smart Contracts 53](#_Toc121771969)

[4.2.3. Front-End 57](#_Toc121771970)

[CHAPTER 5 63](#_Toc121771971)

[RESULT AND EVALUATION 63](#_Toc121771972)

[5.1. Discussion 63](#_Toc121771973)

[5.2. Comparison 63](#_Toc121771974)

[5.3. Evaluation 63](#_Toc121771975)

[CHAPTER 6 64](#_Toc121771976)

[CONCLUSION AND FUTURE WORK 64](#_Toc121771977)

[6.1. Conclusion 64](#_Toc121771978)

[6.2. Future work 64](#_Toc121771979)

[REFERENCES 65](#_Toc121771980)

[APPENDIX 67](#_Toc121771981)

# 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 13](#_Toc121771982)

[Figure 2.2-2. ReactJS with Tailwind CSS 14](#_Toc121771983)

[Figure 2.2.2-3. ReactJS Abilities 15](#_Toc121771984)

[Figure 2.2.2-4. Tailwind CSS Intellisense for VS Code 16](#_Toc121771985)

[Figure 2.2.2-5. “Subcribe” button with Tailwind CSS 16](#_Toc121771986)

[Figure 2.2.2-6. The result of “Subcribe” button 17](#_Toc121771987)

[Figure 2.3.1-7. NodeJS Architecture 18](#_Toc121771988)

[Figure 2.3.2-8. NodeJS local dependencies managed by NPM in package.json 19](#_Toc121771989)

[Figure 2.4.1-9. Visualization of a Blockchain 20](#_Toc121771990)

[Figure 2.4.2-10. Peer-to-peer Network model with EVM 21](#_Toc121771991)

[Figure 2.4.3.1-11. The different between EOAs and Contract Account 22](#_Toc121771992)

[Figure 2.4.3.3-12. Blocks in Ethereum 24](#_Toc121771993)

[Figure 2.4.3.5-13. Smart Contract ABI 26](#_Toc121771994)

[Figure 2.4.3.6-14. The algorithm x \* y = k 27](#_Toc121771995)

[Figure 2.4.3.6-15. How Uniswap works 28](#_Toc121771996)

[Figure 2.4.4.1-16. Application of dApps 29](#_Toc121771997)

[Figure 2.4.4.7-17. MetaMask Wallet 33](#_Toc121771998)

[Figure 2.4.4.7-18. MetaMask approves transaction 34](#_Toc121771999)

[Figure 2.5-19. Creating a new factory contract using CRANQ 35](#_Toc121772000)

[Figure 2.5-20. Every node with details in CRANQ 36](#_Toc121772001)

[Figure 3.2-1. System Use Cases 38](#_Toc121772002)

[Table 2. System’s Functional Description 39](#_Toc121772003)

[Figure 4.1.1-1. Node and NPM version 41](#_Toc121772004)

[Figure 4.1.2-2. CRANQ first screen 42](#_Toc121772005)

[Figure 4.1.2-3. Library of CRANQ 43](#_Toc121772006)

[Figure 4.1.3-4. MetaMask icon in browser 44](#_Toc121772007)

[Figure 4.1.3-5. MetaMask Homescreen 45](#_Toc121772008)

[Figure 4.1.4-6. API key and provider URL 46](#_Toc121772009)

[Figure 4.1.4-7. Alchemy provides free 0.5 Goerli ETH 47](#_Toc121772010)

[Figure 4.2.1.1-8. Using Remix to compile contract creating a mintable ITYU token 48](#_Toc121772011)

[Figure 4.2.1.1-9. Remix will ask for small fee to create a new token 49](#_Toc121772012)

[Figure 4.2.1.1-10. New ITYU token 50](#_Toc121772013)

[Figure 4.2.1.1-11. ITYU token 51](#_Toc121772014)

[Figure 4.2.1.2-12. Create new IUC Token 52](#_Toc121772015)

[Figure 4.2.1.2-13. IUC Token 53](#_Toc121772016)

[Figure 4.2.2.1-14. Factory Contract in CRANQ 54](#_Toc121772017)

[Figure 4.2.2.2-15. Router Contract in CRANQ 55](#_Toc121772018)

[Figure 4.2.2.3-16. Liquidity Contract in CRANQ 56](#_Toc121772019)

[Figure 4.2.2.3-17. Smart Contracts created in CRANQ 57](#_Toc121772020)

[Figure 4.2.3.1-18. Project airdrop structure 58](#_Toc121772021)

[Figure 4.2.3.1-19. ITYU’s tokenAddress and tokenABI 59](#_Toc121772022)

[Figure 4.2.3.1-20. Main function in airdrop 60](#_Toc121772023)

[Figure 4.2.3.2-21. Project DEX structure 61](#_Toc121772024)

# 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

Due to developments in cryptocurrency exchanges, people now have a simpler time buying and selling bitcoin assets. The ever-increasing stream of information suggests that advancements in this area are inevitable. Decentralized exchanges (DEXs) eliminated intermediaries and opened up trading of cryptocurrencies by making all platform activity visible to users. As the price of cryptocurrencies has increased, numerous trading sites like Uniswap, PancakeSwap, Binance, and others have cropped up. Traders on these platforms can instantaneously change the currency they're working in.

In light of the renewed focus on blockchain and DLT, decentralized exchanges (DEXs) that use automated market maker (AMM) protocols are now an essential part of the DeFi ecosystem. Instead of matching the buy and sell sides, AMMs utilize a peer-to-pool approach to decide asset pricing using an analytical mechanism known as a conservation function.

Using the Uniswap V2 protocol, this thesis looks into how DEXs interact with AMM. In today's modern world, face-to-face meetings between buyers and sellers are unnecessary. As an alternative, they interact with a token pool where some of each token is held in reserve. Everything is done in an automated fashion.

# CHAPTER 1

# INTRODUCTION

## Background

DeFi is a blockchain- and smart-contract-based software architecture that, unlike traditional financial systems, removes the need for a third party to act as an intermediary when users issue and trade digital tokens. One of the primary DeFi types is the AMM. Furthermore, AMMs are decentralized marketplaces of crypto-tokens, allowing users to:

* Deposit crypto-tokens to receive shares in an AMM;
* Perform a dual operation of redeeming shares in an AMM for the underlying tokens;
* Swap tokens of a certain type for tokens of a different type.

In the case of a swap, the AMM uses an algorithm to compute the swap rate, which is the number of tokens received by the user multiplied by the amount submitted to the AMM by the user.

## Problem Statement

The "order books" were used to transact tokens before AMM was implemented. Bids and Asks are the terms used to describe orders to purchase and sell, respectively, in an order book system. The order book in a DEX is responsible for matching buying requests with sellers who are willing to meet such requests at a fair price. Therefore, trading on the DEX is possible because traders' orders are matched with those of other traders.

The on-chain order book system in DEXs currently has issues. The orders in an on-chain order book are stored on centralized servers, making them vulnerable to censorship, in contrast to off-chain order books. Limit orders and the cancellation of existing orders in this system are handled through the submission of separate transactions. Users are put in jeopardy at this stage because miners can learn about transactions that aren't included in the block because they are stored in the blockchain.

## Objectives

The primary objective of this thesis paper is to introduce readers to Ethereum technology and demonstrate the advantages of blockchain over conventional Web Application Development.

Requirements for the system are outlined in this article, along with examples of the system's architecture and design. At last, it demonstrates to all implementations how to build a demonstration for this thesis.

My thesis will be a decentralized exchange (DEX) with an automated market maker (AMM) wherein organization tokens can be traded for behavior score tokens. Blockchain allows you to readily view the transaction history of tokens without the need for centralized computer storage. In addition, students have access to and can control the token that represents their behavior grade.

## Assumption

This thesis assumes that the property's necessary legal documents have been digitalized and that the government, International University, and the proposed use of the system all approve of the digitalization.

# CHAPTER 2

# BACKGROUND/APPLIED TECHNOLOGIES

## Overview of Web 3.0

Shortly after the debut of Ethereum in 2014, Ethereum co-founder Gavin Wood popularized the term "Web 3.0" [1]. In Web 3.0, developers often do not create and deploy apps that run on a single server and store their data in a single database. Web 3.0 apps instead run on blockchains, decentralized networks of several peer-to-peer nodes (servers), or a combination of the two that constitutes a cryptoeconomic protocol. These apps are commonly known as decentralized apps (dApps).

* Verifiable
* Trustless
* Self-governing
* Permissionless
* Distributed and resilient
* Stateful
* Native integrated payments

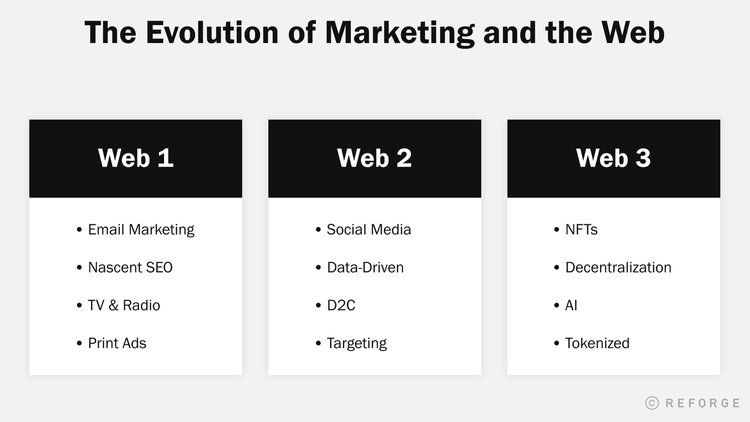


Figure 2.1.1-1. The Evolution of the Web

## ReactJS with Tailwind CSS (Front-End)

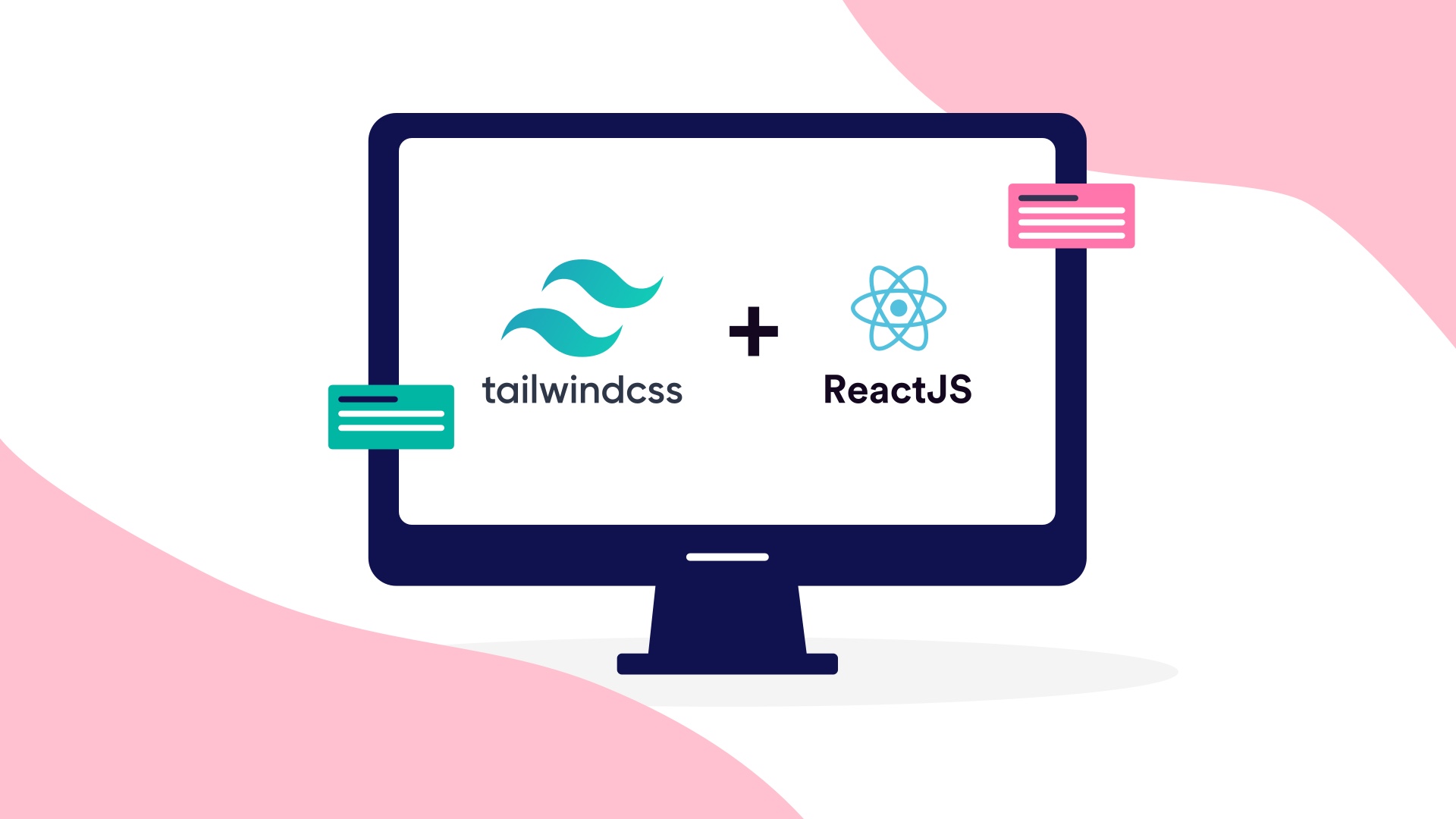


Figure 2.2-2. ReactJS with Tailwind CSS

### ReactJS

Facebook developed the React.js framework as an open-source JavaScript framework and library [2]. It is used to construct interactive user interfaces and web applications more rapidly and efficiently than with standard JavaScript, and with substantially less code.

React's major function in an application is to manage the view layer, similar to the V in the model-view-controller (MVC) paradigm, by delivering the most optimal and efficient rendering execution. Instead of treating the entire user interface as a single unit, React.js encourages developers to divide these complicated UIs into distinct, reusable components that serve as the interface's building blocks.

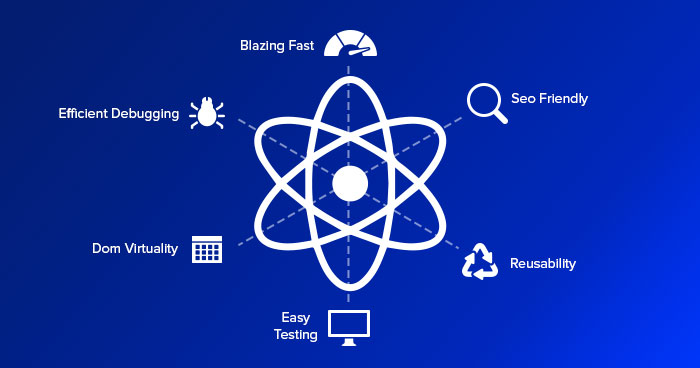


Figure 2.2.2-3. ReactJS Abilities

### Tailwind CSS

Tailwind CSS is a utility-first CSS framework meant to facilitate the creation of applications by users. You can utilize utility classes to modify the component's layout, color, spacing, typography, and shadows, among other things, without leaving HTML or creating a single line of custom CSS.

There are numerous advantages of utilizing a CSS framework like Tailwind. Here are the primary ones:

* **Write less personalized CSS:** With Tailwind, you style items by directly applying pre-existing HTML classes. By utilizing utility classes in this manner, it is possible to create custom designs without coding CSS.
* **Minimize CSS file sizes:** Without a framework such as Tailwind, you must continually write CSS while adding new features and components. As a result, the size and weight of your CSS files continue to increase. By utilizing Tailwind's flexbox and padding utilities, the majority of styles are reusable, reducing the need to write new CSS.
* **No need to create a class name:** When using Tailwind, classes are selected from a predefined design framework. That means you don't have to fret over selecting the "right" class names for specific styles and components, nor do you have to remember difficult ones like sidebar-inner-wrapper.
* **Changes can be made safely:** With the conventional technique, if you make changes to CSS, you risk breaking something on your site. In contrast to CSS, HTML utility classes are local. Therefore, you can modify them without worrying about damaging other elements of your website.

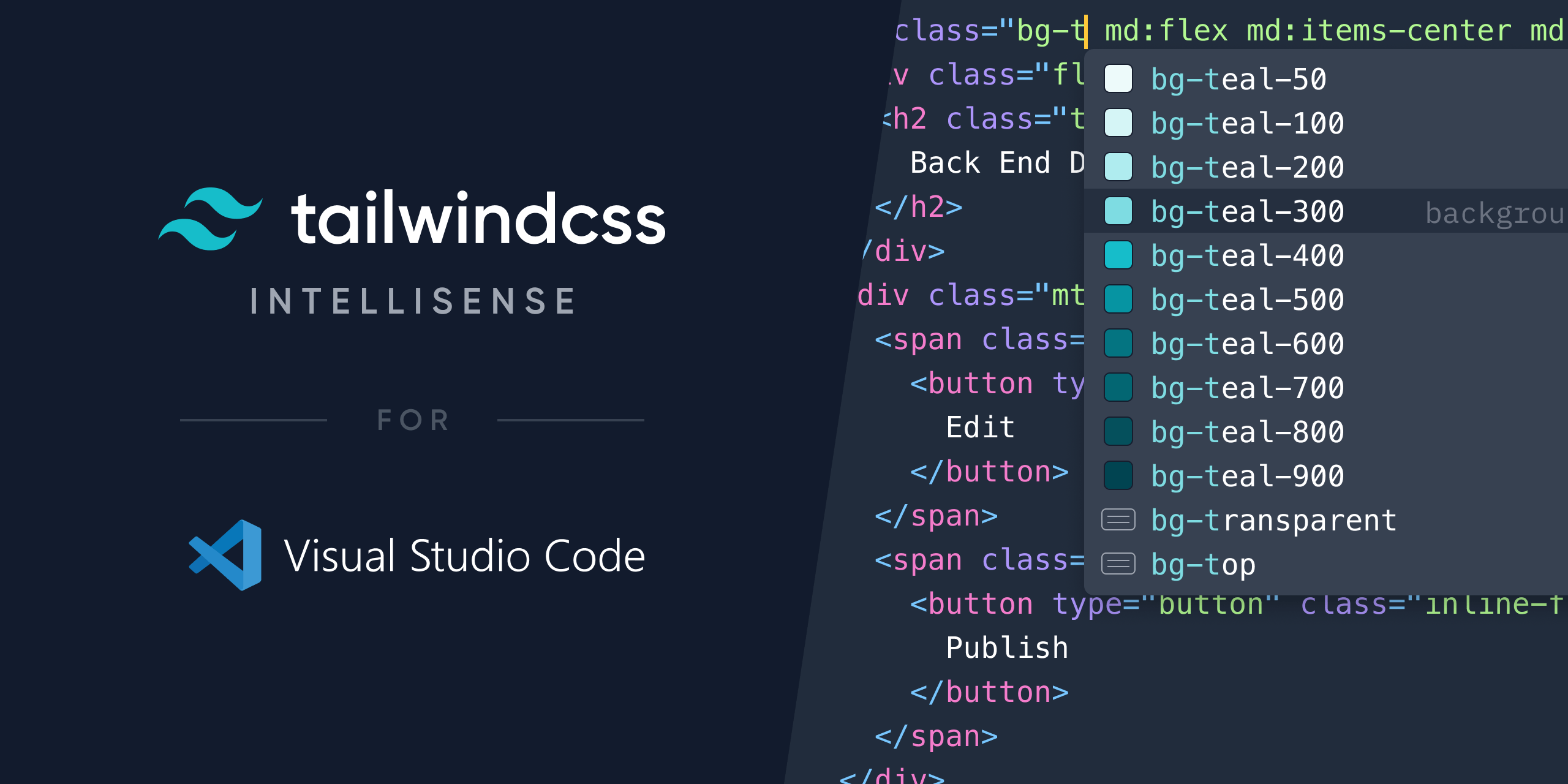


Figure 2.2.2-4. Tailwind CSS Intellisense for VS Code

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

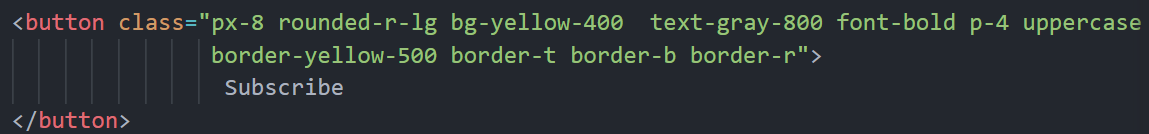


Figure 2.2.2-5. “Subcribe” button with Tailwind CSS

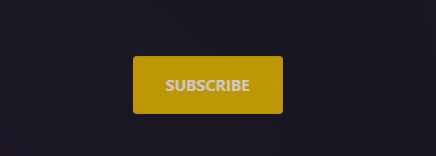


Figure 2.2.2-6. The result of “Subcribe” button

## NodeJS (Running Environment) and NPM

### NodeJS

Node.js is a cross-platform, open-source JavaScript runtime environment and library for executing web applications outside of the client's browser. Node.js is used by developers to create server-side web applications, and its asynchronous, event-driven approach makes it ideal for data-intensive applications.

* NodeJs is based on the V8 engine of Google Chrome, and as a result, its execution time is quite swift, and it operates very quickly.
* As NodeJs does not need to wait for an API to return data, it is useful for developing real-time, data-intensive web applications. It is completely asynchronous, which means it is completely non-blocking.
* As NodeJs is open-source and is nothing more than a JavaScript framework, it is simple for developers who are already familiar with JavaScript to begin developing applications with NodeJs.

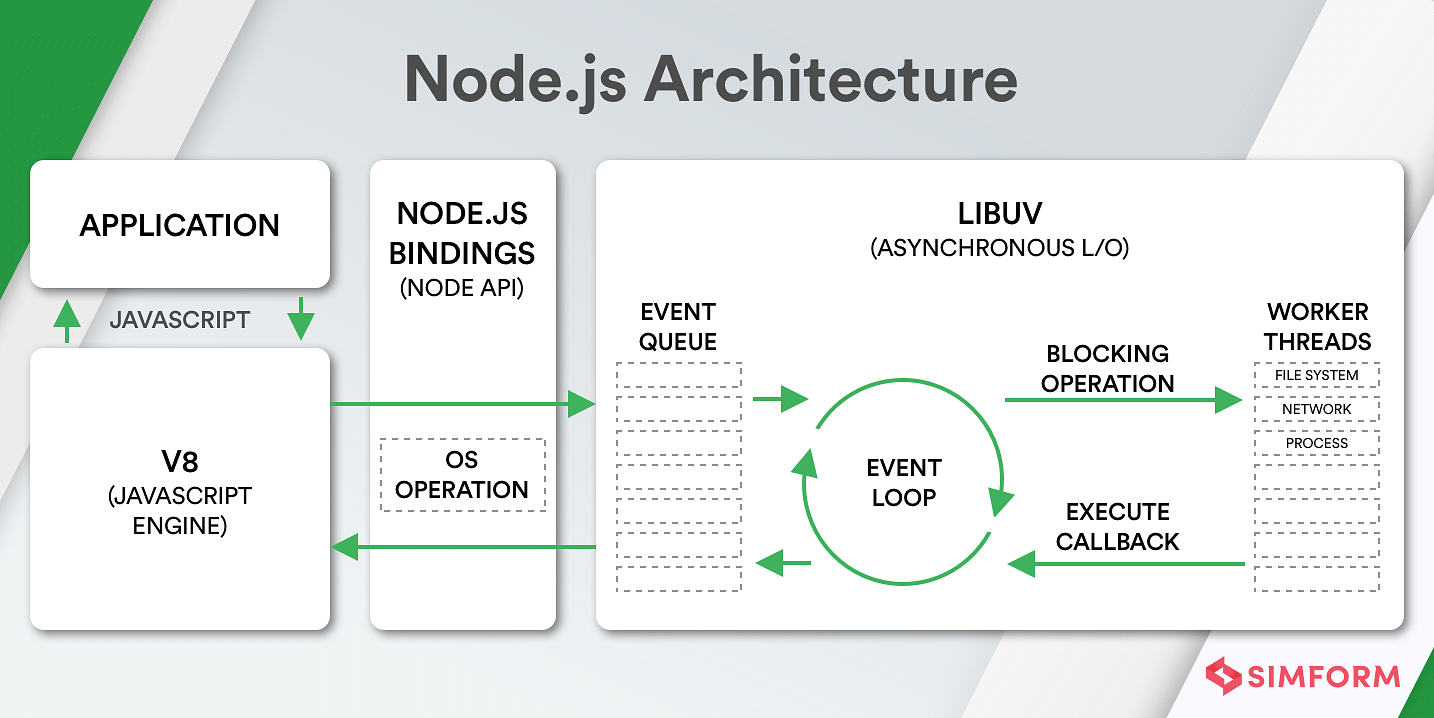


Figure 2.3.1-7. NodeJS Architecture

### NPM

Node package manager (NPM) is an open-source repository of application and website development tools. NPM consists of two elements:

* A repository for open-source project publication. A facility for the storage and retrieval of digital data.
* A command-line interface (CLI) for the repository. An instrument for communicating with the storage facility.

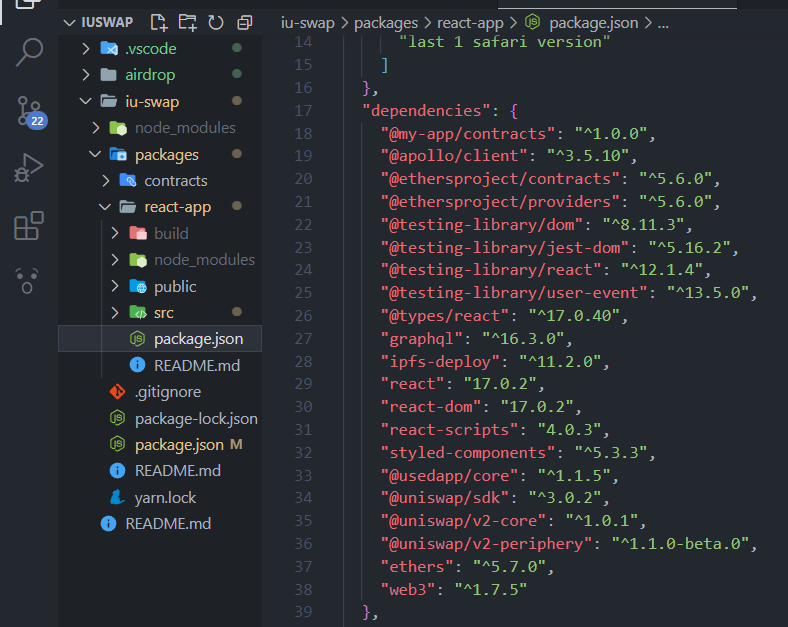


Figure 2.3.2-8. NodeJS local dependencies managed by NPM in package.json

## Overview of Ethereum Blockchain technologies

### Blockchains Explained

Blockchains are the most essential components of this endeavor. A blockchain is a public ledger, or more simply, a database, that stores all transactions on a decentralized network of nodes. As shown in Figure 2.4.1-9, this database is known as a blockchain because its transactions are kept in linearly successive blocks that each contain a reference to the previous block. Nodes collect a set of transactions not included in any previous block and apply a cryptographic hashing algorithm to solve a difficult cryptographic challenge in order to generate new blocks.

This process is referred to as mining, and when a new block is formed, the node that mined and distributed the new block into the network first is rewarded with new coins. Every transaction must pay a fee, which is then passed to the miner of the block when the transaction is included in a new block. This charge and the benefits received from mining a new block provide further incentives for cryptocurrency miners to continue their work.

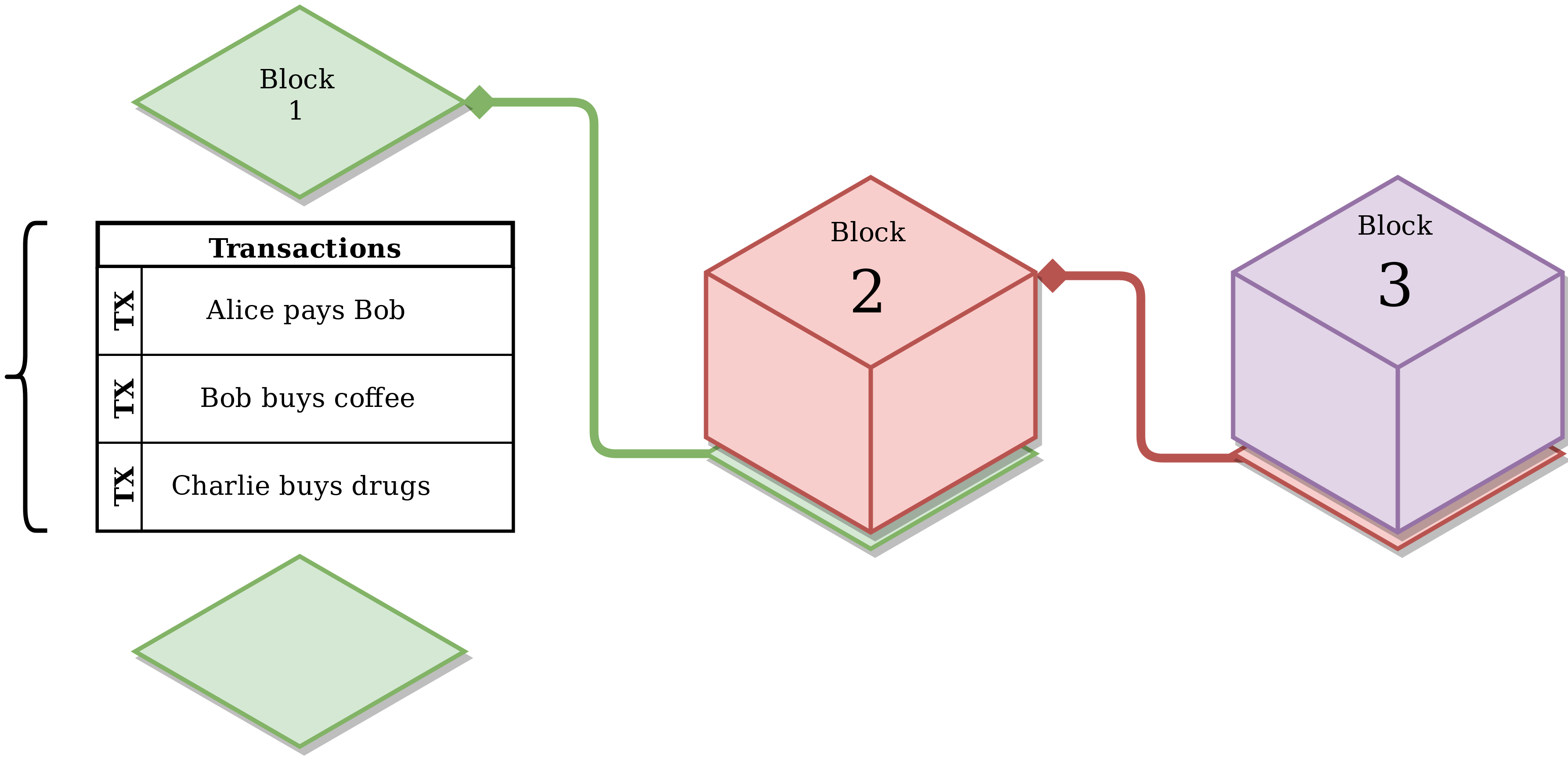


Figure 2.4.1-9. Visualization of a Blockchain

### The Ethereum Blockchain

Ethereum is an open-source, blockchain-based, distributed software platform. It has its own native cryptocurrency, Ether (ETH), as well as a programming language known as Solidity. The distributed ledger technology blockchain maintains an immutable, tamper-proof list of records. Ethereum is the primary challenger to Bitcoin.

The Ethereum platform provides the fully functional Ethereum Virtual Machine (EVM). EVM executes scripts globally over its distributed network of public nodes. These nodes provide the processing power for decentralized applications that run on the network and are created by developers. Developers can purchase ETH to pay for network access, or they can mine for tokens themselves to join the network. Gas, an internal mechanism, determines the cost of network transactions.

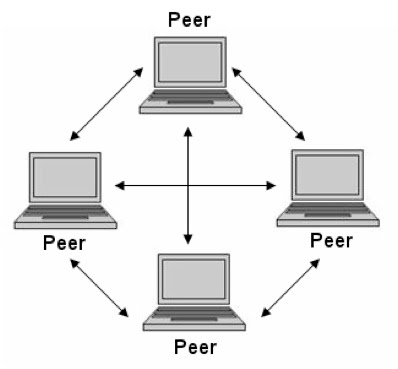


Figure 2.4.2-10. Peer-to-peer Network model with EVM

### Ethereum Platform

The Bitcoin blockchain proven to be too restrictive for more generic applications. This resulted in the creation of Ethereum, the first and most renowned platform for smart contracts. Ethereum differs from Bitcoin in that users can program and design their own operations, whereas Bitcoin only offers established alternatives. This architecture's central component is the EVM, which provides a sandbox environment where code of arbitrary complexity can be executed.

#### Account

The global state of Ethereum consists of numerous entities communicating via a framework for passing messages. These are referred to as accounts. Each account is distinguished by a 20-byte address and maintains an internal state. In Ethereum, there are two account types: externally owned accounts (EOAs) and contract accounts.

An EOA can send transactions, is managed by a private key, and has no related code. When a contract account receives a transaction from an EOA, an associated code is executed. A contract account cannot initiate its own transactions. Always, transactions must start from an EOA.

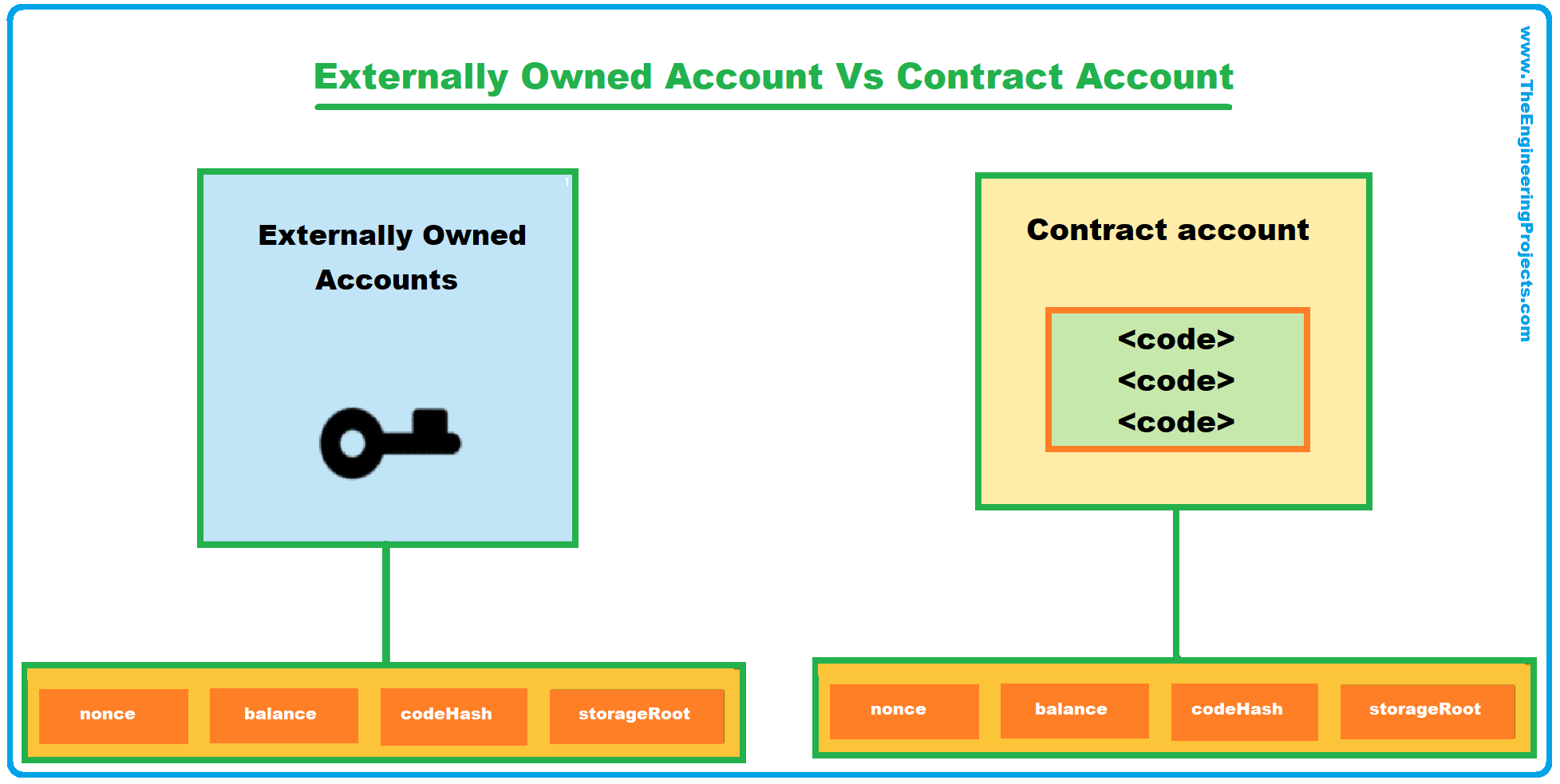


Figure 2.4.3.1-11. The different between EOAs and Contract Account

#### Transactions

Every interaction that modifies the state of the blockchain begins with a transaction. I noted before that Ethereum may be considered as a global state machine; this state can only be altered through transactions. Consider a transaction to be a single instruction that is generated, cryptographically signed, serialized, and added to the blockchain by an EOA. Message calls and contract creations are the two transaction kinds. Closer inspection of the components of a transaction reveals that the following fields are present in both types:

* **nonce:** counter reflecting the number of transactions started by the sender, intended to mitigate attack responses.
* **gasPrice:** the amount of Wei (1 ETH = 1018 Wei is the smallest Ethereum subdenomination) the sender agrees to pay per gas unit for the transaction's execution.
* **gasLimit:** the maximum quantity of gas that can be used during transaction execution.
* **to:** the 20-byte recipient's address. When a contract is created, it is empty (zero).
* **value:** the amount of Wei deducted from the sender's balance and sent to the recipient's address. In the event of contract generation, an initial balance will be established for the new smart contract.
* **v, r, and s:** these values correspond to the transaction's signature and are used to identify the sender.
* **init:** is an array of bytes containing the code used to initialize the new contract account. This piece of code is executed just once during contract generation, after which it is discarded. Another code fragment named body is returned from its execution. Permanently linked to the contract account will be the body.
* **data:** a byte array containing the message call's parameters. A function from a smart contract may, for example, expect as a parameter an integer indicating an id.

#### Blocks

Relevant pieces of data compose what we refer to as a "block" in the network. A block in Ethereum has a header, information about the transactions it contains, and a set of headers from other blocks (these blocks, called ommers, have the same parent as the current block's parent's parent).

Given that blocks are introduced to the Ethereum network significantly faster (15 seconds) than other blockchains (Bitcoin, 10 minutes), more competing blocks are mined. Because only one can be inserted, the remaining blocks are "orphaned." To include these blocks in the main chain, miners might include their header in their current block. The following are some of the more significant components of a block's header:

* **parentHash:** the hash of the header of the parent block; this connection makes the set of blocks a chain.
* **ommerHash:** Hash of the list of ommers added to the block.
* **beneficiary:** the address of the miner's account.
* **difficulty:** the level of difficulty of the block.
* **number:** a counter for all prior blocks beginning with block genesis, which has the value 0.
* **gasLimit:** the current gas per block limit.
* **gasUsed:** the total amount of gas consumed by the transactions in the block.
* **timestamp:** the Unix time at the creation of the block.
* **mixHash:** a 256-hash that, along with the nonce, demonstrates that sufficient computation has been applied to mine this block.
* **nonce**: consists of a 64-bit value that is concatenated with mixHash.

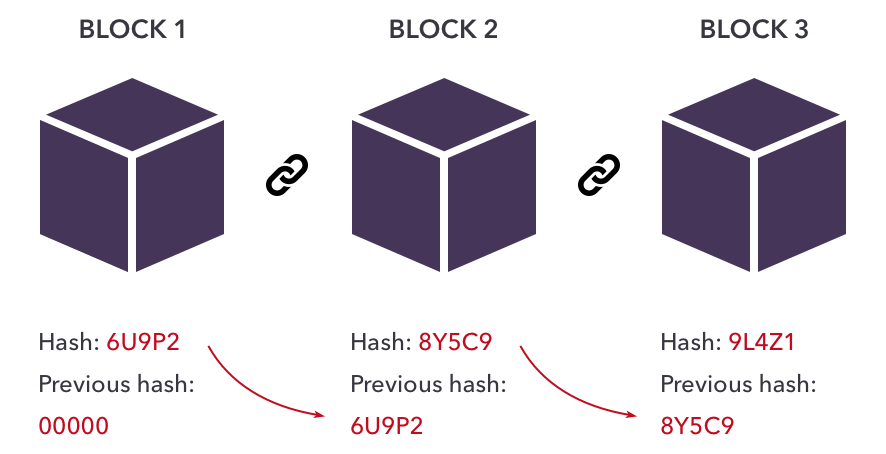


Figure 2.4.3.3-12. Blocks in Ethereum

#### Gas

Since transactions must be executed by all network nodes in order to be validated, a Turing-complete language can easily enable software faults that could cause a transaction to run endlessly. This is usually referred to as the halting problem. Whether unintentionally or deliberately, an infinite loop within a smart contract would result in a denial of service for the platform. Ethereum implemented gas fees as a solution to this problem. Each instruction executed by the EVM incurs a cost quantified in gas units.

Two of the fields that must be set when a transaction is established are the gasPrice and the gasLimit. During the execution of the transaction, the gas units for all instructions are added together and multiplied by the provided gasPrice to get the total gas fee. The gasLimit specifies the maximum number of computing steps a transaction can execute before running out of gas and stopping. This technique guarantees that no transaction would run forever, as doing so would be prohibitively expensive. The gasPrice, provided in Wei, is the price per unit of gas that a user is willing to pay. Since miners give preference to transactions with the highest gasPrice, the gasPrice has a significant impact on how quickly a transaction will be included in a block. In addition to serving as a metering method, gas fees also serve as an incentive for miners, who are responsible for collecting the payments.

#### Smart Contracts

Nick Szabo first defined the phrase "smart contract" in 1994 as "a computerized transaction protocol that performs the provisions of a contract. The fundamental goals of smart contract design are to satisfy typical contractual constraints (such as payment terms, liens, secrecy, and even enforcement), limit malicious and inadvertent exceptions, and reduce the need for trusted intermediaries.

Smart contracts are collections of code and data (or procedures and state) that are placed on the blockchain via "contract generation" transactions. Since transactions must be processed by each network node, the smart contract implementation must be deterministic such that all participants end up in the same state after execution.

To accomplish this, smart contracts can only operate with the input data provided. Oracles, which will be covered in a later section, can feed data from outside the blockchain. Prior to deployment, smart contracts must be compiled. The most essential outputs of the compilation process are the bytecode and the interface. The smart contracts are compiled from the developer's high-level language into machine code so that they may be executed by every node in the EVM. As bytecode is not readable by humans, developers require an intermediary to interface with deployed smart contracts.

ABI (application binary interface) establishes a standard scheme (JSON format) for encoding smart contract code. Using the ABI, calls to deployed smart contracts are made. Once put on the blockchain, the code of a smart contract cannot be modified and remains there for the duration of the network's existence. On the blockchain, only the bytecode is preserved, not the ABI.

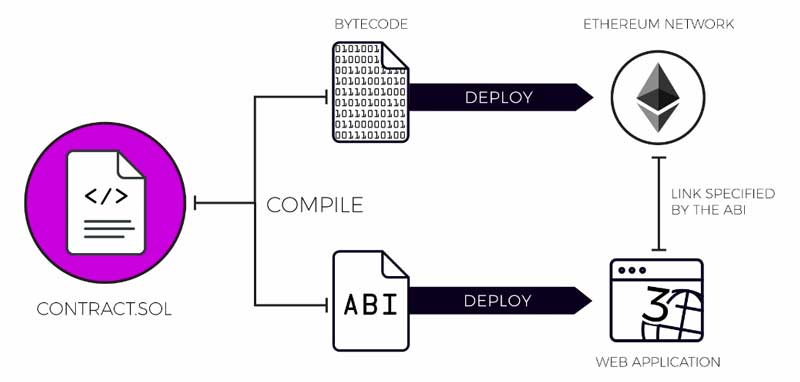


Figure 2.4.3.5-13. Smart Contract ABI

#### AMM Decentralized Exchanges

Automated market makers, often known as AMMs, are decentralized exchanges that pool liquidity provided by users and price the assets that are contained within the pool via algorithms. The precise mechanisms differ from exchange to exchange, but in general, AMMs provide as many users as possible with deep liquidity, cheap transaction fees, and 100% uptime.

Traditional exchanges necessitate that buyers and sellers meet at a price point of overlap on a centralized order book. In comparison, AMMs conduct themselves significantly differently:

* Incentivize users to deposit crypto assets in liquidity pools through a technique known as yield farming.
* Use an algorithm, often x \* y = k, to provide a constant price to everyone dealing with the pool.
* Utilize smart contracts to automatically swap assets between traders and liquidity pools.

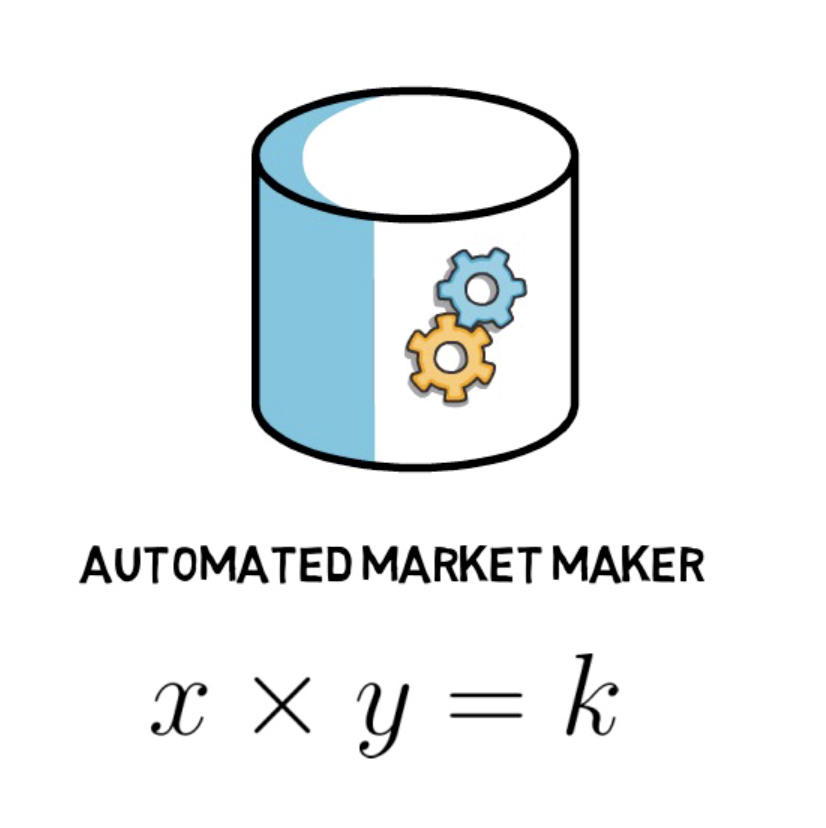


Figure 2.4.3.6-14. The algorithm x \* y = k

When determining the value of an asset, AMMs apply a method known as the constant product formula, which states: x \* y = k. x and y represent equal amounts of a liquidity pool's assets, whereas k represents the total or continuous liquidity of the pool.

Uniswap is a prime illustration of how AMMs function. Uniswap is an AMM protocol that serves trades between you and a liquidity pool funded by liquidity providers (LPs). You can play multiple roles under the AMM paradigm, including trader, liquidity provider, and protocol governor. The protocol itself prices assets and executes transactions using smart contracts.

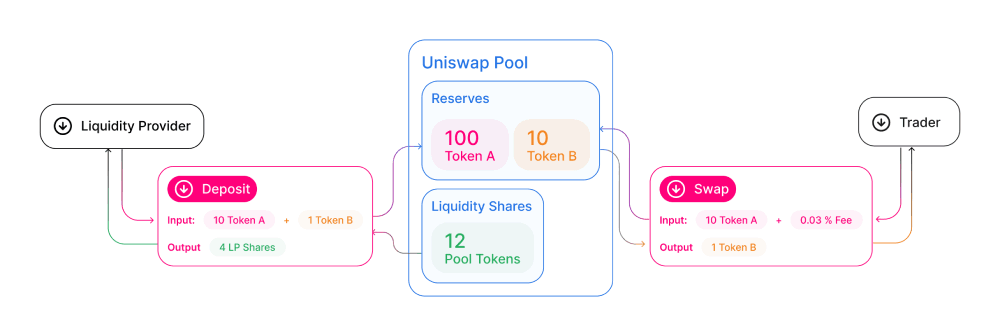


Figure 2.4.3.6-15. How Uniswap works

### Decentralized Finance

#### Decentralized Applications (dApps)

Decentralized applications (dApps) are essential to the DeFi ecosystem. dApps, unlike traditional programs, exist on smart contract platforms such as Ethereum. The primary advantages of decentralized applications over conventional software applications arise from the blockchain's permissionless and censorship-resistant characteristics. Upon deployment, anyone with an Ethereum wallet can interact with a decentralized application so long as the smart contract conditions are met.

The issue dApps aim to solve is the lack of control over one's data. When users utilize centralized applications such as Google services, it is likely that their actions are logged on that platform. Thus, centralized service providers can keep this information and sell it to advertising so that marketers can tailor advertisements to the tastes of customers. It is unsettling for many individuals to have their data handled in this manner. Ideally, information should remain private. dApps restore users' sovereignty over their data by ensuring that no single entity controls the data of any user.



Figure 2.4.4.1-16. Application of dApps

#### Tokens

The Token Taxonomy Framework (TTF) was introduced by the Interwork Alliance [3] in 2019. The TTF's primary objective is to develop a knowledge base for the token economy; as such, it is an important step towards the Alliance's mission to "enable enterprises to adopt and utilize token-powered distributed services in their daily business operations." The TTF is platform-agnostic and takes no position on the implementation of tokens, as only the specification is examined.

Even while tokens can be designed for a variety of applications and purposes, they all possess the following characteristics: they are valuable, representative, digital, distinct, and legitimate. Tokens are deemed valuable because they can typically be measured against a widely acknowledged currency, primarily the U.S. dollar. By representative, we refer to how tokens demonstrate ownership or claim to a digital or real object. Because tokens exist in the digital environment and are typically recorded on blockchain, we refer to them as having a digital nature. The blockchain layer, which is both public and permissionless, and the consensus protocol enable us to check the legitimacy of each token in the same manner we verify the authenticity of physical currency.

#### Fungibility

There are two broad categories of tokens: fungible and non-fungible. The design of fungible tokens is based on the fiat currencies we use in daily life. They are divisible based on the declared number of decimals, and individual units are interchangeable and identical, just like two freshly minted $1 bills.

In the meanwhile, non-fungible tokens representing ownership of a unique item were introduced. Even though made by the same artist, two works of art cannot be identical.

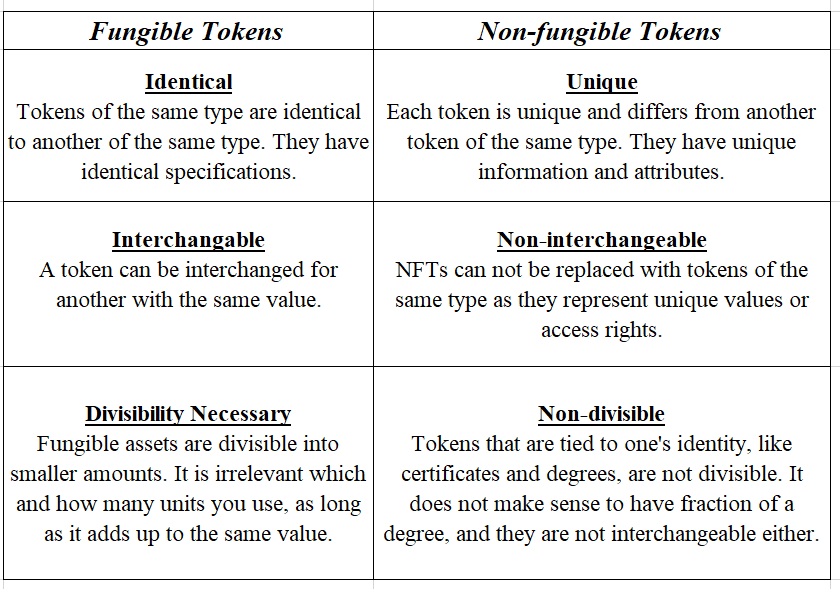


Table 1. The difference between Fungible Tokens and Non-fungible Tokens

#### ERC-20

As Ethereum applications began to be developed, the necessity for interoperability increased. Ethereum Improvement Proposals (EIPs) are design documents that describe the platform's standards and were introduced by the Ethereum community. There are various forms of EIPs, with ERC (Ethereum Request for Comments) being the most pertinent for this thesis. ERC establishes "application-level standards and conventions, including contract standards."

These are interfaces providing a fundamental set of functionalities that every token smart contract must implement. ERC-20 was the first such interface for fungible tokens. Because the ERC-20 is the most prevalent interface in the DeFi domain, we shall examine its fundamental capabilities in greater depth. All ERC-20 compliant token contracts are required to implement the following methods:

* **totalSupply():** yields the total number of tokens in circulation.
* **balanceOf(owner):** returns the owner's balance.
* **transfer(to, value):** transfers the tokens with a value equal to value from the caller's balance to the to address.
* **transferFrom(from, to, value):** permits contracts to transfer tokens on behalf of a user, as the money is not debited from the caller's account balance but rather from the from address. The user with the from address must have previously authorized the caller to transmit a particular number of tokens on his behalf in order to avoid an error.
* **approve(spender, value):** permits the spender to transfer up to the value amount from the caller's balance.
* **allowance(owner, spender):** returns the amount the spender is permitted to withdraw from the account of the owner.

#### Liquidity Pool

A liquidity pool is a pool of funds that are secured by a smart contract. Liquidity pools are utilized to support decentralized trading, lending, and a variety of other activities that we shall examine in the future.

Many DEXs, including Uniswap, rely heavily on liquidity pools. Users known as liquidity providers (LP) contribute two tokens of equal value to a pool to form a market. In exchange for providing their funds, they receive trading fees proportional to their percentage of the overall liquidity for trades that occur in their pool. AMMs have made market making more accessible since anyone can be a liquidity provider.

#### Impermanent Loss

When you give liquidity to a liquidity pool and the price of your deposited assets fluctuates relative to when they were deposited, you incur temporary loss. The greater this change, the greater your vulnerability to transitory loss. In this instance, the loss corresponds to a lower dollar amount at the time of withdrawal compared to the time of deposit. Trading fees can still be used to offset temporary loss. Due to the trading fees, even pools on Uniswap that are highly susceptible to temporary loss can be profitable.

Uniswap assesses a 0.3% fee on all trades that go directly to liquidity providers. Even if a pool is highly sensitive to temporary loss, it may be advantageous to supply liquidity if there is a high volume of trading activity. This is contingent upon the protocol, the particular pool, the deposited assets, and even broader market conditions.

#### Metamask Extension

MetaMask is one of the top cryptocurrency wallets that relies on browser integration and strong design to serve as one of the primary entry points to the world of Web3, DeFi, and NFTs.

MetaMask is a browser extension that functions as an Ethereum wallet and is installed in the same manner as other browser extensions. Once installed, it permits users to store Ether and other ERC-20 tokens, enabling transactions with any Ethereum address.

By connecting MetaMask to Ethereum-based decentralized applications (dapps), users can spend their currencies in games, stake tokens in gambling applications, and trade them on decentralized exchanges (DEXs). It also serves as a gateway to the growing realm of decentralized finance, or DeFi.

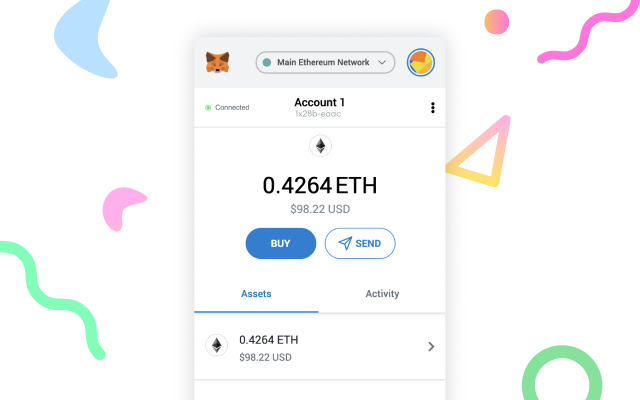


Figure 2.4.4.7-17. MetaMask Wallet

In the JavaScript context, both the web3 object and the convenience library provided by Web3.js are injected by MetaMask. It facilitates user authorization using their private key. Whenever a user performs a transaction that requires a private key to sign the transaction, MetaMask will alert the user for permission before sending the signed request to the blockchain.

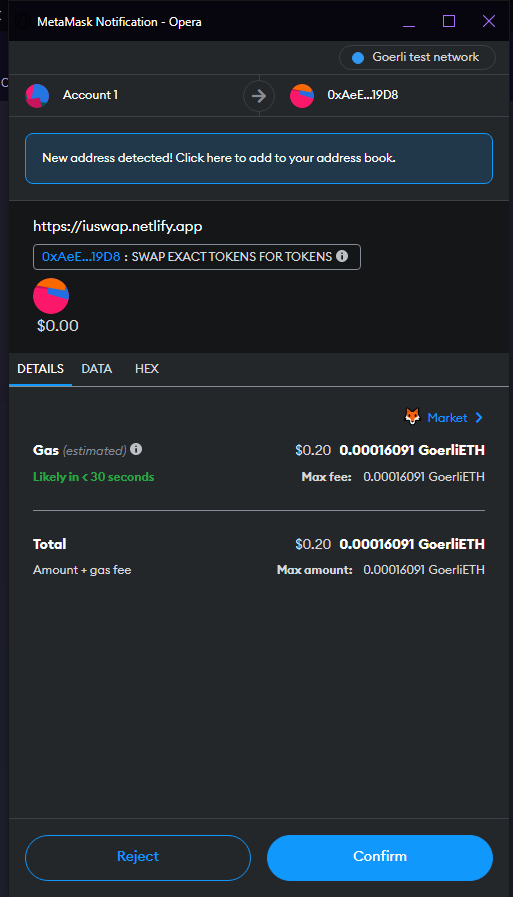


Figure 2.4.4.7-18. MetaMask approves transaction

## CRANQ (Back-End)

CRANQ is a graphical and user-friendly integrated development environment (IDE) that allows users to compile and deploy smart contracts. It offers a graphical user interface that allows users to track the execution of code by observing its flow. Due to its emphasis on defined datatypes and ports, intent can be easily verified.

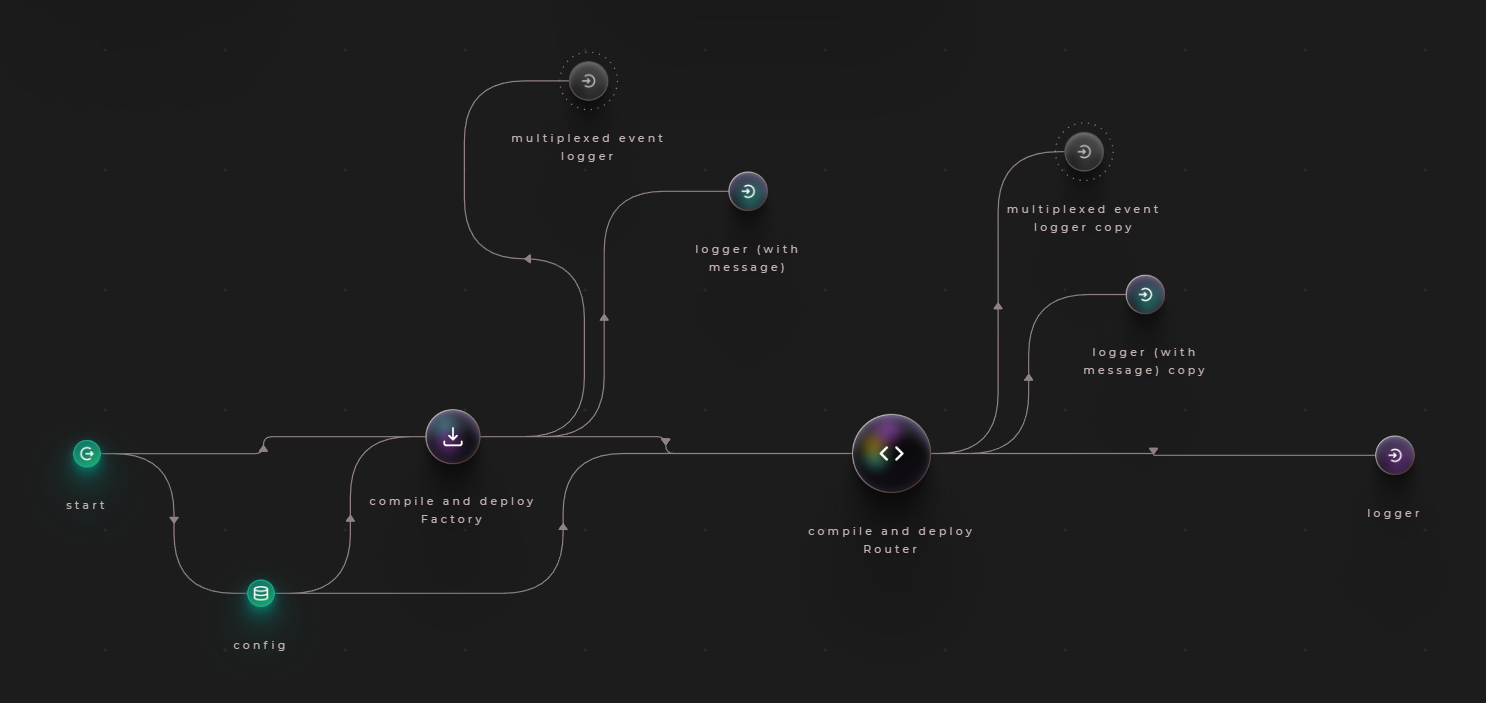


Figure 2.5-19. Creating a new factory contract using CRANQ

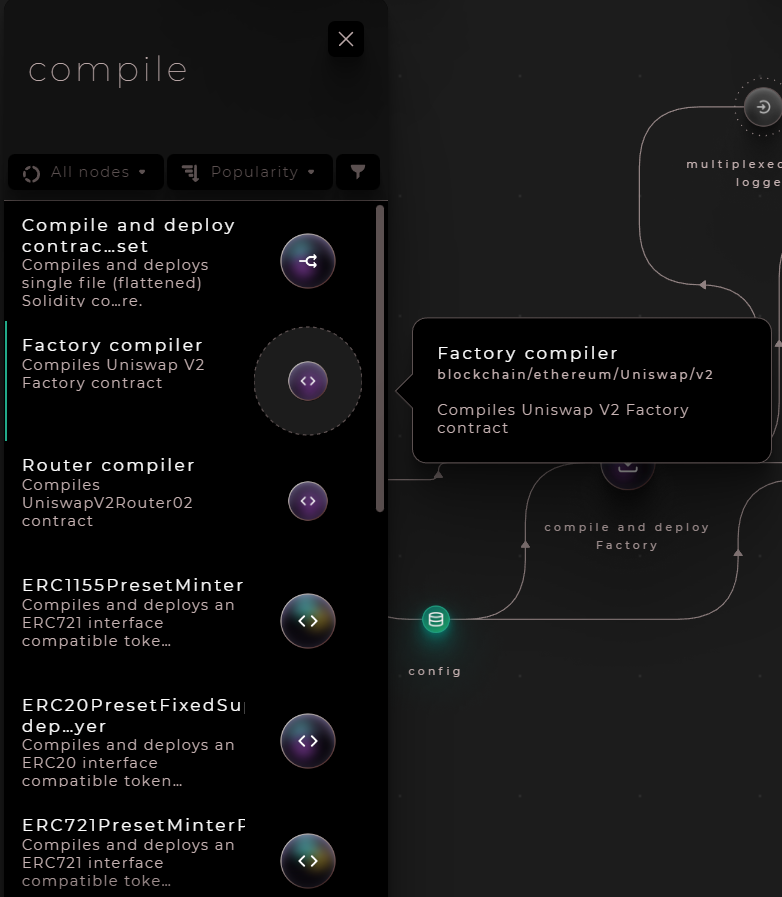


Figure 2.5-20. Every node with details in CRANQ

# CHAPTER 3

# METHODOLOGY

## System Description

ERC-20 will be the standard token format for the system. Before the system is delivered to Mainnet in this thesis paper, it will be released on Goerli Network, an Ethereum test network used for development and testing.

Users can swap tokens by storing them in a crypto wallet such as MetaMask. Etherscan, a block explorer for the Ethereum blockchain, allows for the monitoring of all token transactions and smart contract creations.

Single Page Application (SPA) is utilized for User Side in this thesis paper because the User Interface (UI) must be clean, simple, intuitive, and responsive to enable users complete the work with less effort and time.

## System Use Case

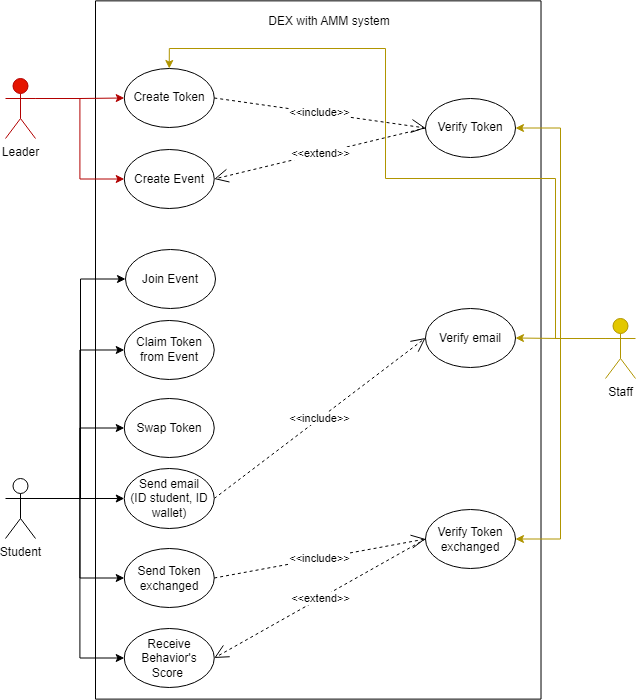


Figure 3.2-1. System Use Cases

## 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

The most popular platform for the development of software applications is called NodeJs. It manages massive amounts of traffic and has strong scalability thanks to its support for asynchronous programming and a single-threaded design. The installation of NodeJS and NPM on a local workstation can be performed by going to <https://nodejs.org/en/download/>. Once installation is complete, open your command prompt and enter the following commands:

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

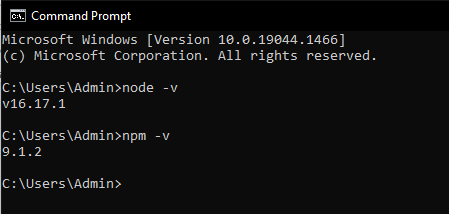


Figure 4.1.1-1. Node and NPM version

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

### CRANQ

As I was saying earlier, CRANQ would be utilized in the process of compiling as well as distributing smart contracts. Simply navigate to https://cranq.io on your web browser to download the software onto your device. After the installation is complete, launch CRANQ and open the first project you created; rename it "IUSwap."

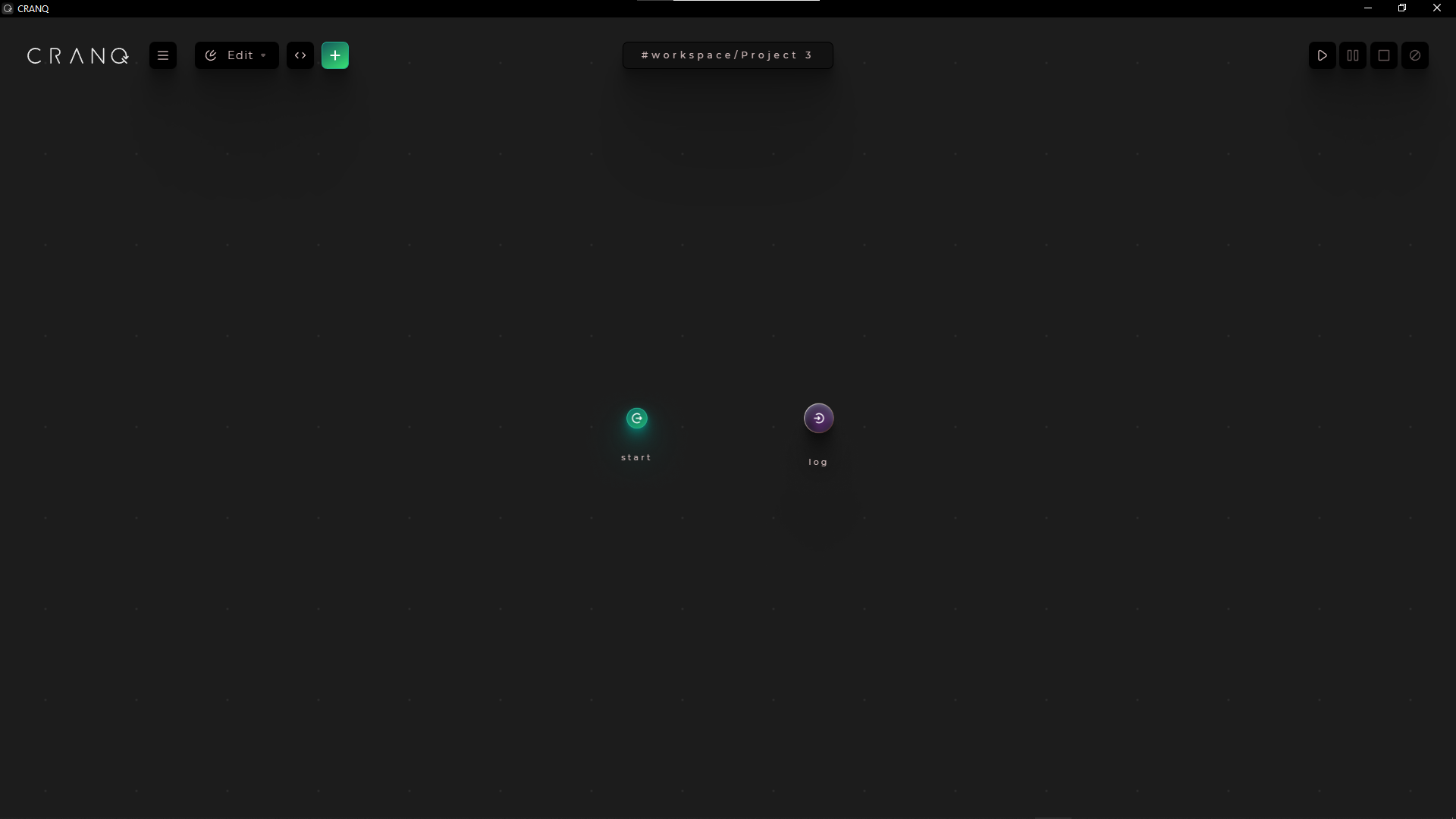


Figure 4.1.2-2. CRANQ first screen

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.

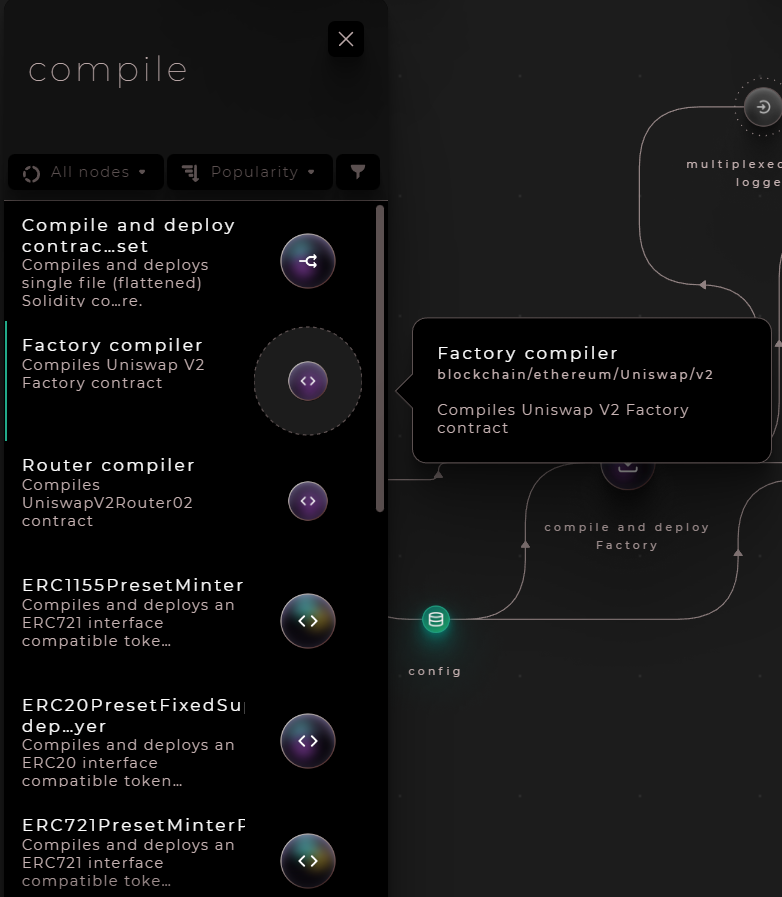


Figure 4.1.2-3. Library of CRANQ

### MetaMask

MetaMask is the most popular cryptocurrency wallet, with millions of users. Without local installation, users of metamask can communicate directly with the whole Ethereum ecosystem, the cornerstone of many decentralized applications. It is compatible with the majority of modern browsers, including Chrome, Edge, and Firefox. Additionally, it offers cross-platform storage 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:

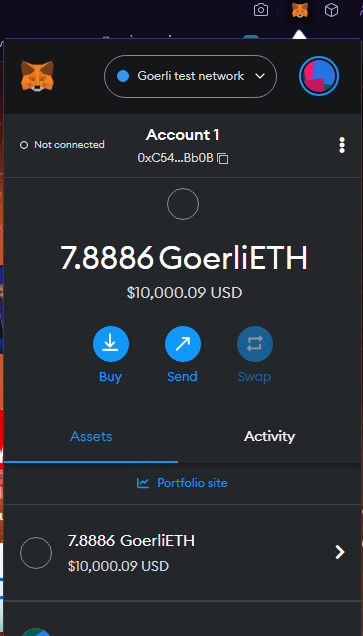


Figure 4.1.3-5. MetaMask Homescreen

### Alchemy

Our ability to actually deploy smart contracts on Blockchain is made possible by Alchemy, the Web3 development platform. Alchemy is a no-cost service that removes a potential source of discomfort by providing all the essential tools together with documentation that is simple to understand. If we go to <https://www.alchemy.com/> to set up a new account, we won't be charged anything for the privilege.

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.

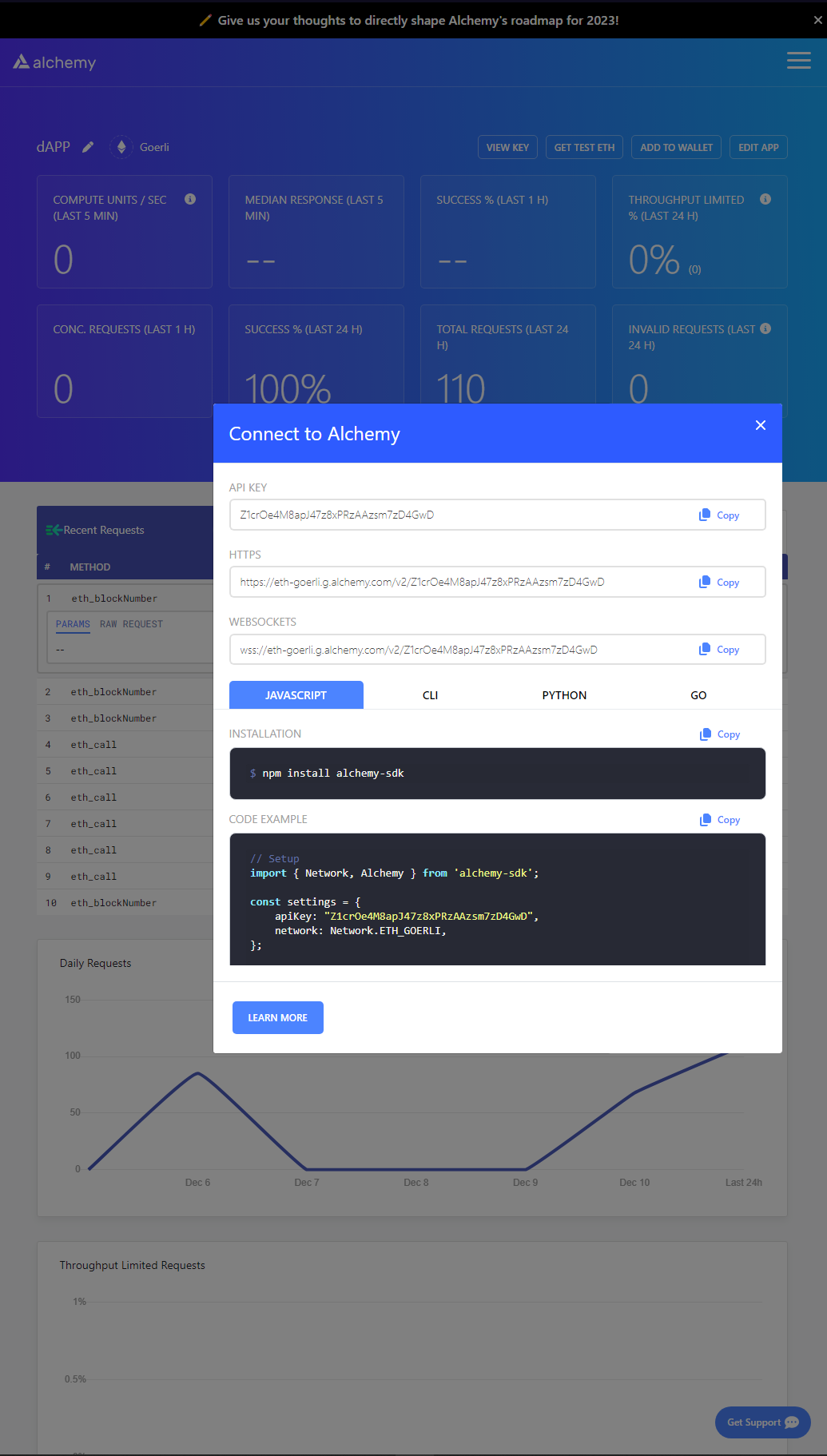


Figure 4.1.4-6. API key and provider URL

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.

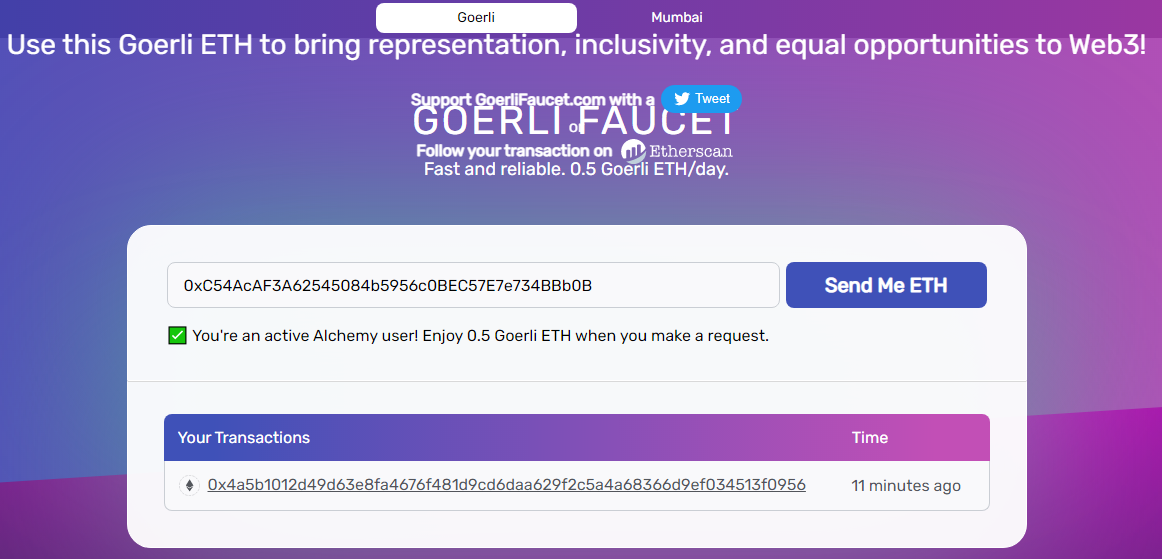


Figure 4.1.4-7. Alchemy provides free 0.5 Goerli ETH

## Implementation

In this section, I will neither present the specifics, nor will I provide a comprehensive explanation of the source code. Instead, it provides the readers with an abstract view of the structure of the code as well as the functionality of each module or component that is detailed in this thesis paper. Visit my Github at the following address to obtain the fully described article as well as the source code: <https://github.com/MiQannn/IUSwap>.

### Back-End

#### Remix

Remix, also known as Remix IDE (Integrated Development Environment), is an open-source Ethereum IDE (Integrated Development Environment) that can be used to write, compile, and debug Solidity code. As a result, Remix can be a very useful tool for Web3 and dApp 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.

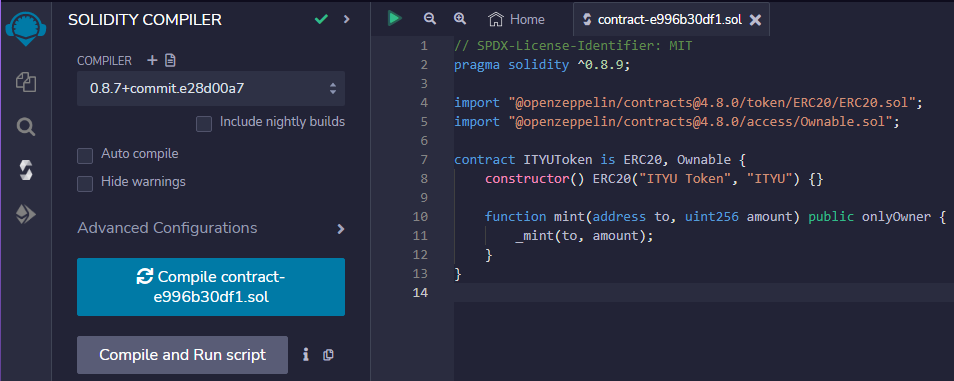


Figure 4.2.1.1-8. Using Remix to compile contract creating a mintable ITYU token

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.

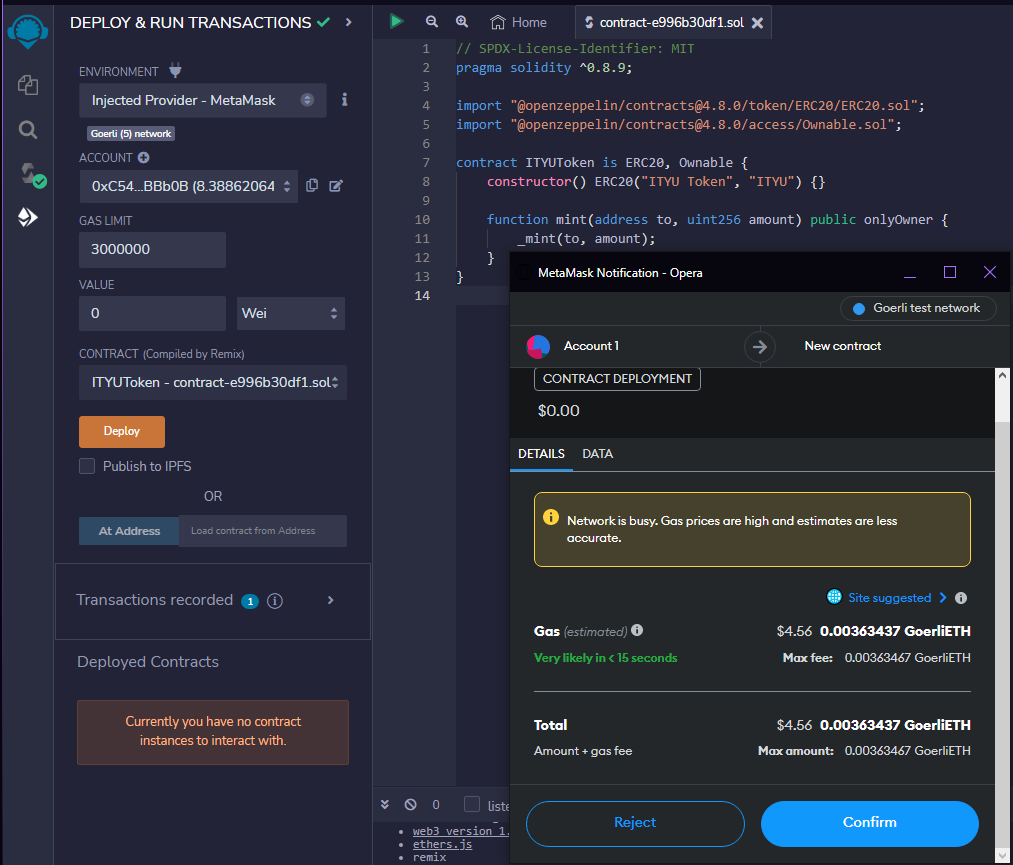


Figure 4.2.1.1-9. Remix will ask for small fee to create a new token

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:

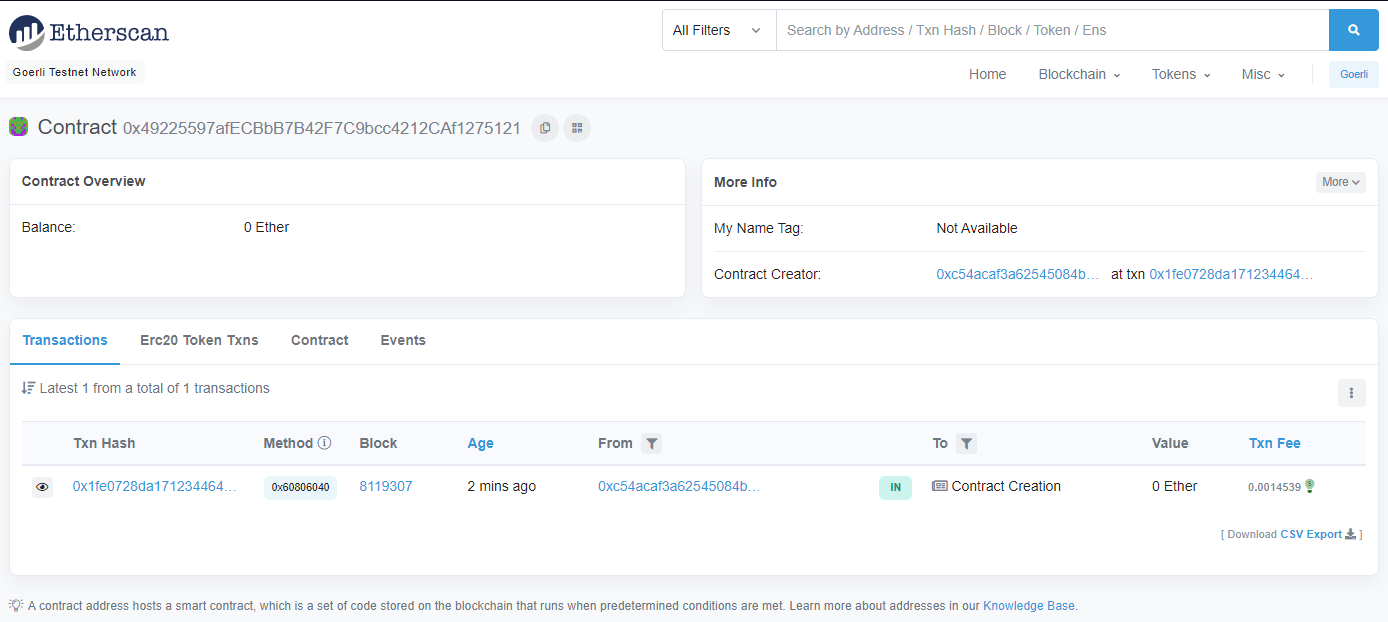


Figure 4.2.1.1-10. New ITYU token

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

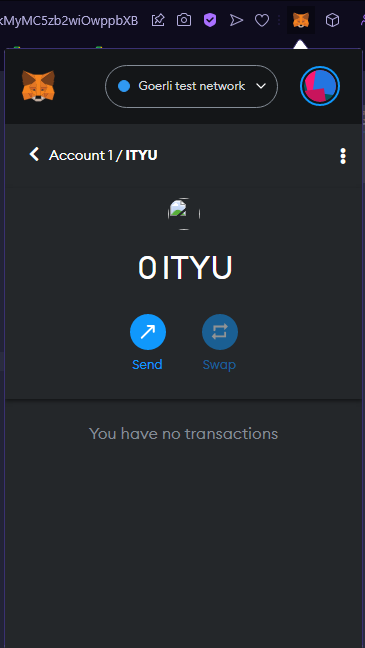


Figure 4.2.1.1-11. ITYU token

#### 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 that allows you to deploy a Smart Contract for a Standard ERC20 Token with no login, configuration, or coding. 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.

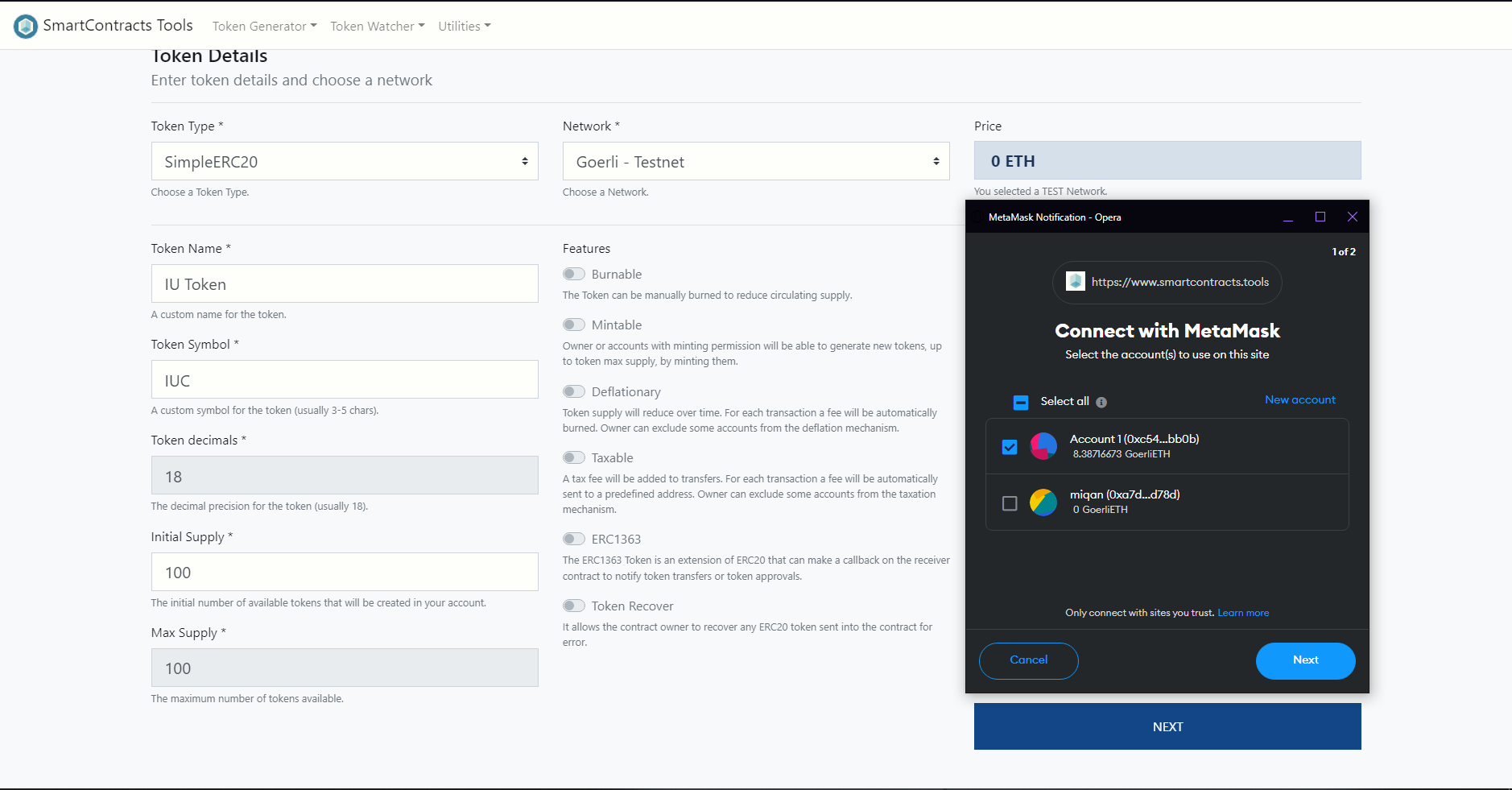


Figure 4.2.1.2-12. Create new IUC Token

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.

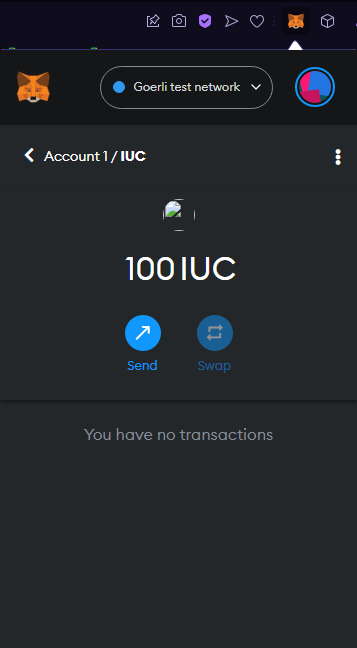


Figure 4.2.1.2-13. IUC Token

### CRANQ Smart Contracts

#### Factory Contract

A Factory Contract is an intelligent contract that produces other intelligent contracts. This design is beneficial for numerous reasons. One reason is because it enables the creation of numerous instances of the same contract, similar to how a programming class operates. Define it once, and then make new instances of that class wherever you like. If you so want, you can monitor every contract that a factory has deployed. It can even save you money on gas, as the factory may be deployed and then used to install other smart contracts.

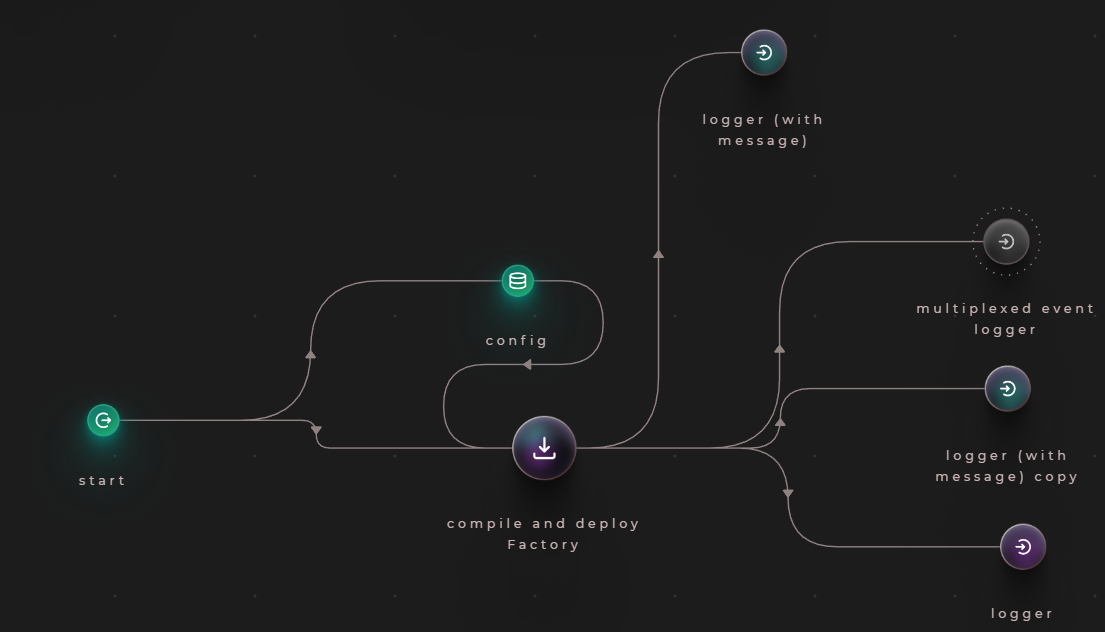


Figure 4.2.2.1-14. Factory Contract in CRANQ

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 provides support for the fundamental interactions with Pair Contracts. To improve modularity, a Factory Contract is created to store the logic that manages Pair Contracts.

Through a set of endpoints and view functions, the Router Contract grants access to the Factory Contract. Notable endpoints include createPair and upgradePair. When there is no address connected with the two tokens intended for use in the DEX activity, the first one will be utilized to generate a new Pair Contract. This URL is only utilized once during Pair formation. When a new version of a Pair Contract is developed and there is a requirement to upgrade an existing Pair Contract implementation, the latter endpoint will be utilized. The Pair Contract will be used in the Liquidity Contract in the future.

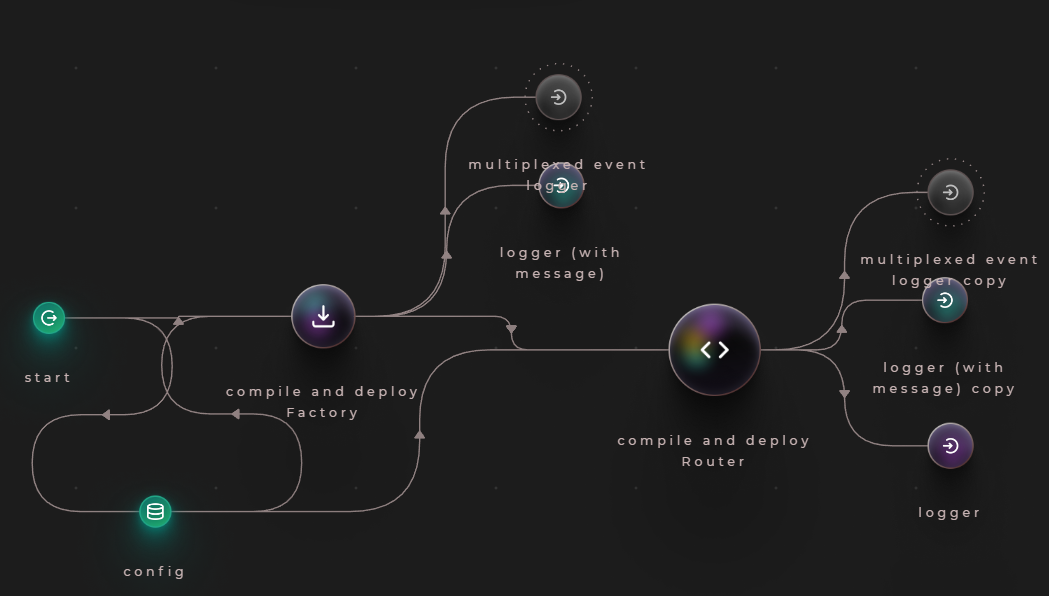


Figure 4.2.2.2-15. Router Contract in CRANQ

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 market can function. A Liquidity Contract is a smart contract in which coins are secured in order to provide liquidity. When a user supplies liquidity, a smart contract issues liquidity pool tokens (LPTs). These tokens represent the liquidity provider's portion of the pool's assets.

In contrast to traditional exchanges that utilize order books, a DEX's price is often determined by an Automated Market Maker (AMM). When a transaction is executed, the AMM employs a mathematical formula to determine how much of each item in the pool must be exchanged to complete the trade.

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

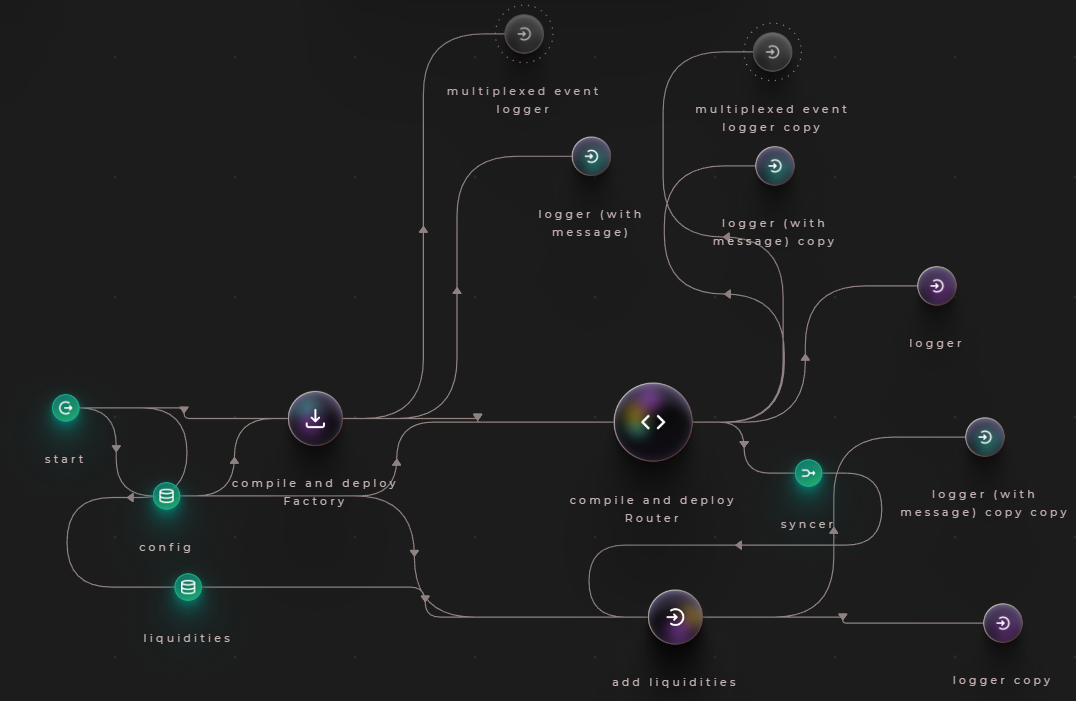


Figure 4.2.2.3-16. Liquidity Contract in CRANQ

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.

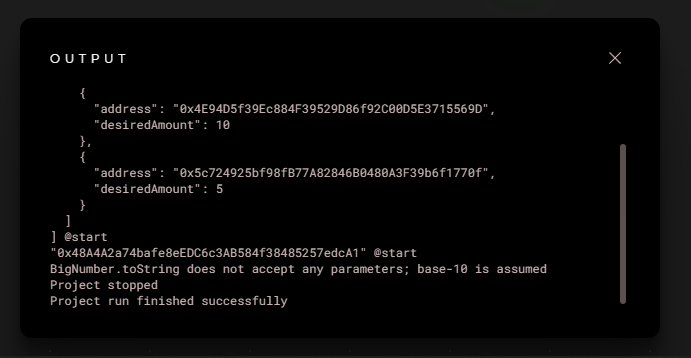


Figure 4.2.2.3-17. Smart Contracts created in CRANQ

### 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

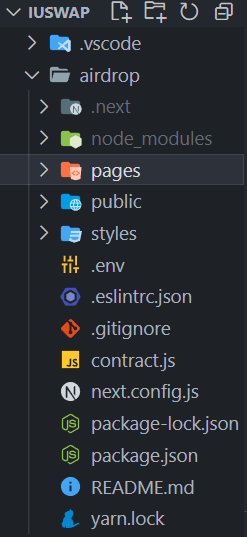


Figure 4.2.3.1-18. Project airdrop structure

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

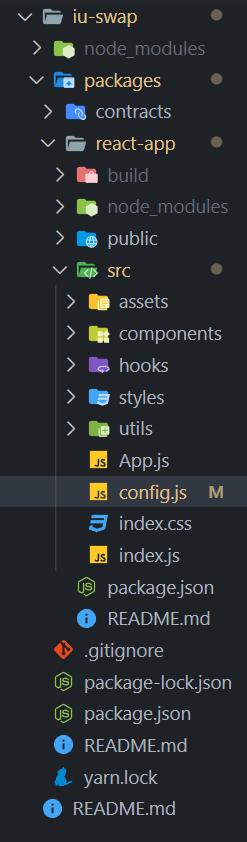


Figure 4.2.3.2-21. Project DEX structure

# 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