

**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**  
**SINGAPORE**

CZ4153 Blockchain Technology

Semester 1 AY 2020/21

Project 10 – BlueJay Gasless Tokens

ReadMe File

Name	Matriculation Number
Fazili Numair	U1822056F
Gupta Jay	U1822549K
Kanodia Ritwik	U1822238H

School of Computer Science & Engineering  
Nanyang Technological University, Singapore

## Table of Contents

1. Introduction	3
2. Approach	3
3. Development	4
3.1 Smart Contracts	4
3.2 User Interface	4
4. Deployment	7
4.1 Contracts	7
4.2 Paymaster	7
5. Testing	8
5.1 Deployment Commands	8
5.2 Application Commands	8
5.3 Compatibility Matrix	9
6. Future Works	10
7. Conclusion	10

## 1. Introduction

Public blockchain networks are open access, therefore, anyone can join the network while remaining anonymous. Anyone who sends an Ethereum transaction must have Ether (ETH) to pay for its gas fees. This forces new users to purchase Ether (which can be a daunting task due to compliance and regulatory requirements such as KYC processes and other transaction/processing fees) before they can start using a decentralized application.

In this project, we have designed a decentralized Ethereum-based blockchain application where the user does not need to pay for any gas fees while performing a transaction. The gas fees are managed and paid by another entity '*paymaster*'.

A simple use case can be of a vendor switching to Ethereum-based smart contract network to accept transactions using its own token currency. In this scenario, user onboarding and payments by the customers will be a hurdle if they need to firstly buy ether, and then, pay a gas fee for every purchase. Therefore, our prototype can be utilized to allow the vendor to cover any transaction charges on behalf of its customers.

## 2. Approach

Our approach builds on top of the sample implementation provided by the OpenGSN team. In the original version, a whitelist paymaster is used which essentially accepts all transactions from the sender. In our application, we introduce two token paymaster contracts which take an ERC20 token from the sender and transfer it to a recipient contract. One of the paymasters charges a token fee to the user as a substitute for the transaction fee and the other subsidizes the transaction fee by making a zero-fee transaction, where the latter case presents a case for customer onboarding.

The corresponding application flow chart is given in Figure 2.1

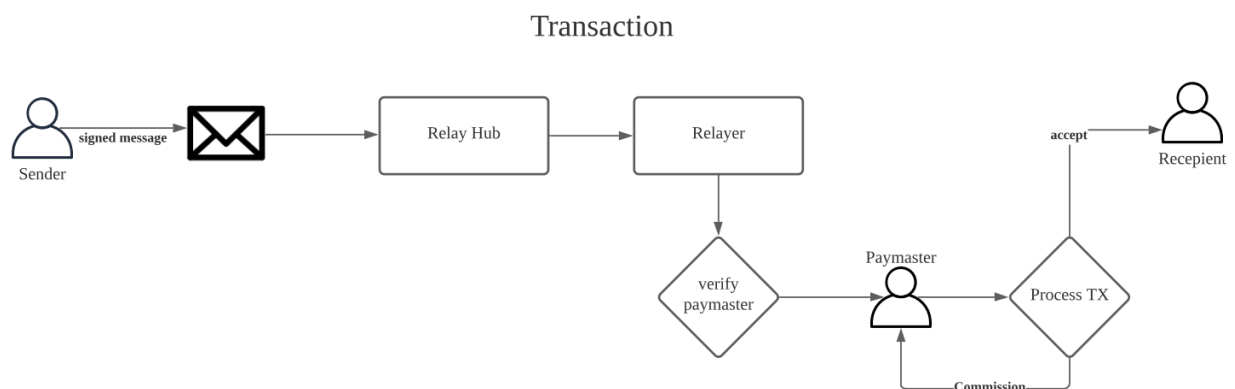


Figure 2.1: Transaction Flow

## 3. Development

### 3.1 Smart Contracts

The following smart contracts are implemented which can be found in the Contracts library.

#### 3.1.1 Token.sol

Custom ERC20 token contract used for demonstration.

#### 3.1.2 TokenBank.sol

Token bank is a vendor contract to buy or sell ERC20 tokens. Following the contract creation, the token bank mints 1000 ERC20 tokens with value pegged to 1:1 with ETH.

#### 3.1.3 TargetContract.sol

This is the recipient contract (vendor) that inherits from Base Relay Recipient to support receiving relayed transactions. The contract does not offer any business logic and contains two functions pertaining to no commission or a token commission transaction. These functions are invoked by the sender and used to demonstrate the functioning of the two modes of transaction.

#### 3.1.4 NoFeePaymaster.sol

Paymaster contract transfers a single ERC20 token from the sender's account to the recipient's account. The logic is stored inside the preRelayedCall function where the contract checks if the sender has authorized the paymaster to withdraw tokens required for the current transaction.

#### 3.1.5 TokenFeePaymaster.sol

This paymaster supports the same logic as NoFeePaymaster with the addition of a token commission required for the transaction. The commission is set to a flat fee of 1 ERC20 token and the other token is transferred to the recipient contract as done by the NoFeePaymaster.

### 3.2 User Interface

The interface of our application is built using React.js, Bootstrap, and deployed via a Node.js server. It consists of a single page containing the following functionalities.

3.2.1 Ether (ETH) and Token (ERC20) balances for the User, Free & Paid Paymaster, and the Recipient.

3.2.2 Token Bank balances in Ether (ETH) and Tokens (ERC20).

3.2.3 Free & Paid Paymaster allowances in Tokens (ERC20).

3.2.4 Payment system for the user to send a transaction of '1' token to the recipient whose gas fee can be chosen to be paid by either the Free or the Paid paymaster.

3.2.5 Purchase system to buy tokens for the user.

3.2.6 Paymaster Allowance management to increase the spending allowance limits for the Free & Paid paymasters.

3.2.7 A ‘refresh’ button which retrieves all the above-mentioned balances, tokens, and allowances for display in the interface.

‘web3.js’ is a collection of libraries that allow to interact with a local or remote Ethereum node using HTTP, IPC or WebSocket. It is used within the React framework to connect to the Ganache local Ethereum blockchain network.

A sample code snippet is given below which is used to retrieve User Ether (ETH) and token balances using *web3.js*.

```
// Web3 Connection
const web3 = new Web3(Web3.givenProvider || "http://localhost:8545");
const netId = await window.ethereum.request({ method: 'net_version' })

// ERC20 Token Contract
const tokenAddress = await tokenJson.networks[netId].address
const tokenContract = new web3.eth.Contract(tokenJson.abi, tokenAddress);

web3.eth.getAccounts().then(async (accounts) => {
  // Get User ETH Balance & Convert to Ether
  var balanceOfUser = await web3.eth.getBalance(accounts[0]);
  var etherOfUser = web3.utils.fromWei(balanceOfUser, "ether")
  this.setState({ etherOfUser: etherOfUser })

  // Get User Token Balance
  const tokenBalance = await tokenContract.methods.balanceOf(accounts[0]).call();
  this.setState({ tokenOfUser: tokenBalance })
});
```


There are some modifications done to the *webpack* configuration of the React application using the *react-app-rewired* package since we are accessing smart contract JSON configurations which lie outside of the source (*src*) folder of the interface. By default, React.js does not allow the JavaScript files to access any folder or file outside the source folder.

```
module.exports = function override(config, env) {
  config.resolve.plugins = config.resolve.plugins.filter(plugin => !(plugin instanceof ModuleScopePlugin));
  return config;
}
```

localhost

Option 10: BlueJay Gasless Tokens

Numair Fazili Jay Gupta Ritwik Kanodia




**User**

Entity that does not require Ether to execute a transaction on the Ethereum Platform.

ETH (Ether)0

ERC20 (Tokens)0

1 TokenChoose and Pay




**Free Paymaster**

Paymaster receives the transactions via the relay and pays the Gas Fee for the transactions.

ETH (Ether)0

ERC20 (Tokens)0




**Paid Paymaster**

Paymaster receives the transactions via the relay and pays the Gas Fee via the tokens supplied by the user.

ETH (Ether)0

ERC20 (Tokens)0



**Receipient**

Entity that receives the transactions that are sent from the user through the relay.

ETH (Ether)0

ERC20 (Tokens)0

Token Bank - Buy Tokens

20 TokensBuy

Token Bank Balances

Ether Balance0

Token Balance0

Allowance Manager

5 tokensIncrease allowance

Paymaster Allowances (in Tokens)

Free Paymaster (ERC20)0

Paid Paymaster (ERC20)0

Refresh all Balances

## 4. Deployment

The deployments are stored inside the migrations module and contract deployments are performed by the `deploy_contracts` script which performs the following actions:

### 4.1 Contracts

- 4.1.1 Deploy Token and Token Bank contract
- 4.1.2 Mint 1000 ERC20 tokens for the Token Bank address.
- 4.1.3 Assign Minter role to Token Bank contract
- 4.1.4 Deploy the recipient contract.

### 4.2 Paymaster

- 4.2.1 Fetch address of Forwarder and Relay Hub (These are static addresses for the test simulation).
- 4.1.2 Deploy paymaster contracts.
- 4.1.3 Set address of Relay Hub and Trusted forwarder to those derived in 4.2.1.
- 4.1.2 From Relay Hub, allocate each paymaster 1 ETH to conduct transactions.

## 5. Testing

### 5.1 Deployment Commands

**Warning:** Do not reinstall the node modules (except those in 5.6.1) as the it throws compatibility errors. All other node modules are packaged with the code.

- 5.1 Open command prompt / terminal and navigate to OpenGSN folder
- 5.2 Execute the command yarn ganache
- 5.3 In a separate terminal window execute npm gsn start (to initialize the gas station network)
- 5.4 In a separate terminal window execute rm -rf build/contracts (clear cache)
- 5.5 In the same terminal as 5.4 execute yarn start
- 5.6 After the deployed are completed in 5.5, in another terminal window navigate to OpenGSN/ui folder and execute command npm start
- 5.6.1 Install relevant node/npm dependencies if prompted to do so in 5.6
- 5.7 The application should be deployed at localhost:3000
- 5.8 Open Metamask and import any private key from ganache. In our experiments, we used the following key: 0x4f3edf983ac636a65a842ce7c78d9aa706d3b113bce9c46f30d7d21715b23b1d

### 5.2 Application Commands

Prior to the demonstration, the following actions must be performed

- 5.2.1 Purchase ERC20 tokens from the Token Bank
- 5.2.2 Allocate allowances to both paymasters

Actions in 5.2.1 and 5.2.2 are outlined using the flowchart given in Figure 2



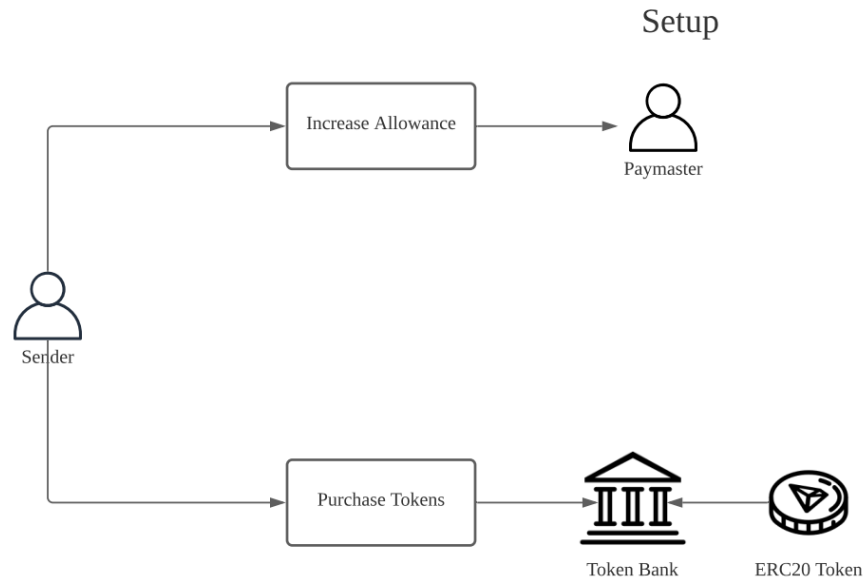


Figure 2: Initial Application Setup

5.2.3 Transfer tokens from sender to recipient using either a free paymaster or a token fee paymaster.

\*The transactions in 5.2.1 and 5.2.2 are only used to create a demo environment and paid using gas fees by the user.

\*Press the refresh button at the bottom of the screen to update the balances, this is implemented for the end user to track changes onclick.

### 5.3 Compatibility Matrix

Name	Version
Node	14.14.0
Npm	6.14.8
Solidity	0.7.6 (solc-js)
Yarn	1.22.15
Truffle	5.4.14
Web3	1.5.3

\*The application was tested on a Windows PC.

## 6. Future Works

The current implementation only supports primitive transactions as only a single ERC20 token is transacted in a single TX. In forthcoming implementations, the sender should be able to transfer a specific number of tokens to the recipient contract by adding the required parameter to the signed message. Moreover, the current implementation is constrained to a specific ERC20 token and in order to scale to any ERC20 token, subsequent implementations can leverage Uniswap or similar ERC20 token trading libraries.

## 7. Conclusion

In this project, our team developed an application to allow users to interact with the Ethereum network without paying GAS fees. This application has various use cases from removing KYC hassles to enhancing customer onboarding experiences. Besides, providing an introduction to solidity development, this project also enabled us to understand the idiosyncrasies associated with developing smart contracts and understanding the rapidly evolving literature.

Throughout the 15 weeks of the semester, we experienced a steep learning curve on Blockchain fundamentals, Ethereum Smart Contracts, and Security, Privacy, and Scalability aspect of blockchain systems. Coupled with this development project, consisting of a relatively new tech stack for us – Solidity, React, Ganache (Truffle Suite), it immensely helped us with new learning opportunities every day as well as understand the peculiarities of building blockchain-based systems in practice.