1. TokensSwap contract

It's a custom router contract just like a router contract of Sushiswap. It uses the factory contract address & WETH contract address of Polygon testnet of Sushiswap while deploying this contract.

Constructor parameters :

```
0xc35DADB65012eC5796536bD9864eD8773aBc74C4
0x0d500b1d8e8ef31e21c99d1db9a6444d3adf1270
0xf9e1761385db44b23da15c4c2750306ef90050f9
0xf9e1761385db44b23da15c4c2750306ef90050f9
15
10
```

Contract functions :

1. constructor( ) : It initialises the contract address provided in parameters for specific contracts and also sets msg.sender as owner of the contract and sets discount rates.

2. swapExactTokensForTokens( ) : This function is used to swap ERC20 tokens to other ERC20 tokens provided that the user enters the input amount he is ready to give. It's the same as the function used in Sushiswap router , just a small modification is that the user gives some amount of MATIC tokens to be sent as a transaction fee that is to be transferred to AppFees contract as platform fees. If a user sends some extra MATIC tokens other than platform fees, it's refunded back to the user. It's a payable function.

It validates that feePath has an input token and MATIC tokens in the array.
It also validates that the transaction fees (platform fees) sent by the user is sufficient.
It also validates that the minimum amount to get to the user is there in liquidity.

Function parameters :

```
        uint amountIn,    // input amount give by user
        uint amountOutMin, // output amount to be expected to receive
        address[] calldata path, // path of selected ERC20 tokens
        address[] calldata feePath, // path of input token to MATIC
        address to, // address where tokens has to be send after swap
        uint deadline // deadline for transaction to be reverted
```

3. swapTokensForExactTokens( ) : This function is used to swap ERC20 tokens to other ERC20 tokens provided that the user enters the output amount he expects after swapping. It's the same as the function used in Sushiswap router , just a small modification is that the user gives some amount of MATIC tokens to be sent as a transaction fee that is to be transferred to AppFees contract as platform fees. If a user sends some extra MATIC tokens other than platform fees, it's refunded back to the user. It's a payable function.

It validates that feePath has an input token and MATIC tokens in the array.
It also validates that the transaction fees (platform fees) sent by the user is sufficient.
It also validates that the minimum amount to get to the user is there in liquidity.

Function parameters :
```
        uint amountOut, // output amount to be expected to receive
        uint amountInMax,  // maximum input amount given by the user
        address[] calldata path, // path of selected ERC20 tokens
        address[] calldata feePath, // path of input token to MATIC
        address to, // address where tokens has to be send after swap
        uint deadline // deadline for transaction to be reverted
```

4. swapTokensForExactETH( ) : This function is used to swap ERC20 tokens to MATIC tokens provided that the user enters the output amount he expects after swapping. It's the same as the function used in Sushiswap router , just a small modification is that the user gives some amount of MATIC tokens to be sent as a transaction fee that is to be transferred to AppFees contract as platform fees. If a user sends some extra MATIC tokens other than platform fees, it's refunded back to the user. It's a payable function.

It validates that feePath has an input token and MATIC tokens in the array.
It also validates that the transaction fees (platform fees) sent by the user is sufficient.
It also validates that the minimum amount to get to the user is there in liquidity.

Function parameters :
```
        uint amountOut, // output amount to be expected to receive
        uint amountInMax,  // maximum input amount given by the user
        address[] calldata path, // path of selected ERC20 tokens
        address to, // address where tokens has to be send after swap
        uint deadline // deadline for transaction to be reverted
```

5. swapExactTokensForETH( ) : This function is used to swap ERC20 tokens to MATIC tokens provided that the user enters the input amount he spends for swap. It's the same as the function used in Sushiswap router , just a small modification is that the user gives some amount of MATIC tokens to be sent as a transaction fee that is to be transferred to AppFees contract as platform fees. If a user sends some extra MATIC tokens other than platform fees, it's refunded back to the user. It's a payable function.

It validates that feePath has an input token and MATIC tokens in the array.
It also validates that the transaction fees (platform fees) sent by the user is sufficient.

It also validates that the minimum amount to get to the user is there in liquidity.

Function parameters :

```
        uint amountIn,  // input amount given by the user
        uint amountOutMin, // output amount to be expected to receive
        address[] calldata path, // path of selected ERC20 tokens
        address to, // address where tokens has to be send after swap
        uint deadline // deadline for transaction to be reverted
```

6. swapETHForExactTokens( ) : This function is used to swap MATIC tokens to ERC20 tokens provided that the user enters the output amount he expects after swapping. It's the same as the function used in Sushiswap router , just a small modification is that the user gives some amount of MATIC tokens to be sent as a transaction fee that is to be transferred to AppFees contract as platform fees. If a user sends some extra MATIC tokens other than platform fees, it's refunded back to the user. It's a payable function.

It validates that feePath has an input token and MATIC tokens in the array.
It also validates that the transaction fees (platform fees) sent by the user is sufficient.
It also validates that the minimum amount to get to the user is there in liquidity.

Function parameters :

```
        uint amountIn,  // input amount given by the user
        uint amountOut, // output amount to be expected to receive
        address[] calldata path, // path of selected ERC20 tokens
        address to, // address where tokens has to be send after swap
        uint deadline // deadline for transaction to be reverted
```

7. swapExactETHForTokens( ) : This function is used to swap MATIC tokens to ERC20 tokens provided that the user enters the input amount to send for swapping. It's the same as the function used in Sushiswap router , just a small modification is that the user gives some amount of MATIC tokens to be sent as a transaction fee that is to be transferred to AppFees contract as platform fees. If a user sends some extra MATIC tokens other than platform fees, it's refunded back to the user. It's a payable function.

It validates that feePath has an input token and MATIC tokens in the array.
It also validates that the transaction fees (platform fees) sent by the user is sufficient.
It also validates that the minimum amount to get to the user is there in liquidity.

Function parameters :

```
        uint amountIn,  // input amount given by the user
        uint amountOutMin, // output amount to be expected to receive
        address[] calldata path, // path of selected ERC20 tokens
        address to, // address where tokens has to be send after swap
        uint deadline // deadline for transaction to be reverted
```

8. transferOwnership( ) : This function is used to set the new contract owner. It can be called by only the current contract owner. By default contract deployer is the owner.

Function parameters:

```
address newOwner // new contract owner address
```

9. setGieAppFees( ) : This function is used to set transaction fee percentage (platform fees). It can only be set by the contract owner. It takes 2 arguments that are transaction fees with decimal places as well. Fees and decimals can't be 0 and total percentage value cannot be greater than 100.

Function parameters:

```
        uint fees,   // fees value like 15
        uint decimals // decimal value like 10
```

So overall gieAppFees becomes 15/10 = 1.5 %

10. setGieAppContract( ) : This function is used to set the new GieApp contract address. It can be called by only the current contract owner. This is the address where platform fees are stored..

Function parameters:

```
        address _gieAppContract // new gie app contract address
```

11. setGieTokenContract( ) : This function is used to set the new Gie Token contract address. It can be called by only the current contract owner.

Function parameters:

```
        address _gieTokenContract // new gie token contract address
```

12. updateDiscount( ) : This function is used to update the discount rate for a specific tier with the provided amount. It can be called by only the current contract owner.

Function parameters:

```
        uint _tierNo,   // tier no. to modify
        uint _amount,   // amount to set for discount in this tier
        uint8 _discountPercent // discount percent of this tier
```

13. _updateDiscount( ) : This is an internal function which is used to update the discount rate for a specific tier with the provided amount. It updates the state of the contract.

Function parameters:

```
        uint _tierNo,   // tier no. to modify
        uint _amount,  // amount to set for discount in this tier
        uint8 _discountPercent // discount percent of this tier
```

14. _preUpdateDiscountCheck( ) : This is an internal function which is used to do pre checking of the values before updating the discount values.

Function parameters:

```
        uint _tierNo,  // tier no. to modify
        uint _amount,  // amount to set for discount in this tier
        uint8 _discountPercent // discount percent of this tier
        uint8 _index // number to check where tier lies
```

15. _swapPreCheck( ) : This is an internal function which is used to perform pre checking of the values before swapping of the tokens.

Function parameters:

```
        address account, // address of user who called swap function
        address[] memory feePath, // array of fee path
        uint amountIn, // input amount to swap
        bool isInputEth, // boolean value to check input is ETH or not
        uint ethAsFee // fees sent by user as transaction fees
```

16. calculateDiscountPercent( ) : This is a view function that calculates the discount to be given to the user holding GIE tokens.

Function parameters:

```
        address account,  // address of user whose discount is to be
                             calculated
```

17. _swap( ) : This function is copy pasted from the Sushiswap router contract without any modifications.

18. quote( ) : This function is copy pasted from Sushiswap router contract without any modifications.

19. getAmountOut( ) : This function is copy pasted from Sushiswap router contract without any modifications.

20. getAmountIn( ) : This function is copy pasted from Sushiswap router contract without any modifications.

21. getAmountsOut( ) : This function is copy pasted from Sushiswap router contract without any modifications.

22. getAmountsIn( ) : This function is copy pasted from Sushiswap router contract without any modifications.

23. getReserves( ) : This function is copy pasted from Sushiswap router contract without any modifications.

24. calculateFeesForTransaction( ) : This function is custom created by us which calculates the transaction fees(platform fees) for particular swapping.

Function parameters:

```
        address account,  // address of user whose transaction fees
                             is to be calculated
        uint amount // input amount sent by the user
```

25. receive( ) : This function is used to accept MATIC in the contract.

==

2. GIE Token contract

It's an ERC 20 token standard contract that is platform token.

Token Name : GIE Token
Token Symbol : GIE
Token decimals : 18
Total Supply : 0

Constructor parameters :

GIE Token
GIE

Contract functions :

1. name( ) : This function returns the name of the token.

2. symbol( ) : This function returns the symbol of the token.

3. totalSupply( ) : This function returns the total supply of the token. By default total supply is 1000000000 * 10 ** 18.

4. balanceOf( ) : This function returns the balance (amount of token) of a particular address.

5. allowance( ) : This function returns the amount of tokens that a particular user can spend on behalf of the actual owner of the token.

6. transfer( ) : This function is used to send a specific amount of tokens from one address to some other address.

7. transferFrom( ) : This function is used to send a specific amount of tokens by one address on behalf of the actual owner of the tokens (Owner permits the address to spend on his behalf).

8. approve( ) : This function is used to give approval to some address by the owner so that specific address can spend the tokens on behalf of the actual owner.

9. mint( ) : This function is used to mint new tokens and can only be called by the owner.

10. burn( ) : This function is used to burn existing tokens and can only be called by the owner.

11. decreaseAllowance( ) : This function is used to decrease the allowance by the owner for a particular address.

12. increaseAllowance( ) : This function is used to increase the allowance by the owner for a particular address.

13. transferOwnership( ) : This function is used to set the new contract owner. It can be called by only the current contract owner. By default contract deployer is the owner.

===

3. AppFees contract

This contract is responsible for collecting all the transaction fees (platform fees) given by the user while performing any of the swapping. This contract is completely handled by the admin of the project that is set in the contracts as contract owner. Owner is allowed to change the contract owner and transfer the funds to another address.

This contract’s constructor doesn’t take any parameters and sets msg.sender to contract owner.

Contract functions :

1. receive( ) : This function is used to accept MATIC in the contract and it updates total fees collected in the contract.

2. transferOwnership( ) : This function is used to set the new contract owner. It can be called by only the current contract owner. By default contract deployer is the owner.

Function parameters:

```
address newOwner // new contract owner address
```

3. transfer( ) : This function is used to transfer MATIC tokens to the provided address and the amount specified in the parameters. It can be called only by the contract owner.

Function parameters:

```
        address beneficiary,
        uint256 amount
```

4. getBalance( ) : This function is used to fetch the MATIC balance of the contract.

===

Important details of project

A) Blockchain used : Polygon Matic

B) Contract addresses of all contracts (deployed on Mumbai Matic testnet) :

1. **TokensSwap contract :** `0xB3355Ee1027D0ee025C822600E8DC3cDd50c5Ed4`

2. **AppFees contract :** `0xb566b9ff918dbAB0CE882E6c1bfa5DA52ec9fc1a`

3. **GieToken contract :** `0xC8E00f2c0554838530B5b851D7713378Ff249297`

C) Referenced / Forked source : Forked Router contract of Sushiswap & modified it slightly as per our use.

D) Dependencies : Dependent on Factory & WETH contract of Sushiswap.

======================= THE END ========================