**Инструкция по взаимодействию с PancakeSwap**

Сайт PancakeSwap:

Мейннет - <https://pancakeswap.finance/>

Тестнет - https://pancake.kiemtienonline360.com/#/swap

Адреса, которые понадобятся

Мейннет BSC:

1. Router - 0x10ED43C718714eb63d5aA57B78B54704E256024E
2. Factory - 0xcA143Ce32Fe78f1f7019d7d551a6402fC5350c73
3. WBNB - 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c

Тестнет BSC:

1. Router - 0x9Ac64Cc6e4415144C455BD8E4837Fea55603e5c3
2. Factory - 0xB7926C0430Afb07AA7DEfDE6DA862aE0Bde767bc
3. WBNB - 0xae13d989daC2f0dEbFf460aC112a837C89BAa7cd

ABI, которые понадобятся:

1. PancakeRouter\_ABI.json
2. IPancakeFactory\_ABI.json
3. IPancakePair\_ABI.json
4. IERC20\_ABI.json

Для взаимодействия с биржей в основном будут использоваться методы смарт-контракта Router, так как он предоставляет удобный интерфейс для операций обмена и взаимодействия с ликвидностью. Смарт Factory понадобится для получения адреса пары, если известны адреса 2 токенов (может и не понадобится, так как адреса пар уже могут быть вшиты). Смарт PancakePair понадобится для вычисления количества получаемых токенов при обмене или количества необходимых токенов для обмена. Также с помощью него вычисляется параметр Price impact и проверка достаточности ликвидности.

Работа с нативным токеном платформы. Если пара состоит из токена и нативной криптовалюты (в нашем случае - BNB). То в качестве адреса для нативного токена используем адрес - WBNB.

Параметры, которые может настроить сам пользователь - slippage, deadline

slippage - процент отклонения желаемого количества токенов от фактического. Так как транзакция совершается не моментально, то цена токена может измениться за время подтверждения, соответственно меняется и количество получаемых токенов. Чтобы избежать потерь, мы ставим slippage. На сколько процентов может отклониться желаемое количество токенов

deadline - количество минут с момента нажатия на кнопку, в течение которых будет валидна транзакция. Если подтверждение займет слишком много времени, то транзакция не пройдет, так как истечет deadline.

Работа exchange состоит из двух разных частей: работа с ликвидностью и работа со свапами

Обозначим tokenA - токен в верхнем input-е, tokenB - токен в нижнем input-е. Получается меняем tokenA на tokenB.

**Рассмотрим работу с ликвидностью**

1. Сначала пользователь выбирает пару токенов, которой хочет предоставить ликвидность (адреса токенов должны быть разные). После выбора пары токенов, необходимо узнать адрес пары (если он не вшит заранее). Если пара содержит нативный токен BNB, то используем адрес WBNB. Для этого у смарта Factory вызываем метод getPair. Если выдаст ошибку или не будет такой пары, то переходим к шагу 3:

address pair = await contractFactory.methods.getPair(address tokenA, address tokenB).call()

tokenA - адрес первого токена

tokenB - адрес второго токена

pair - адрес смарт-контракта пары PancakePair

В данном случае порядок токенов не имеет значение

1. После того, как получили адрес пары, нужно получить ликвидность(количество токенов A и B) пары. Так как в смарте пары токены сортируются (сортировка идет по значению адреса), то чтобы получить корректные значения нужно написать примерно такую функцию:

// Примерный код на JavaScript

// tokenA, tokenB - адреса токенов

function sortTokens(tokenA, tokenB) {

[token0, token1] = tokenA < tokenB ? [tokenA, tokenB] : [tokenB, tokenA]

return [token0, token1]

}

// pair - адрес пары

function getReserves(pair, tokenA, tokenB) {

[token0, \_ ] = sortTokens(tokenA, tokenB)

[reserve0, reserve1] = await contractPair.methods.getReserves().call()

[reserveA, reserveB] = (token0 == tokenA) ? [reserve0, reserve1] : [reserve1, reserve0]

return [reserveA, reserveB]

}

1. У нас есть несколько вариантов, либо пары не существует, либо резервы нулевые, либо резервы ненулевые. Первые два случая идентичные. По сути пользователь сам предоставляет первоначальную ликвидность и может сам ставить количество токенов. Для этого вызываем метод у Router addLiquidity или addLiquidityETH, если один из токенов пары нативный - BNB.

// addLiquidity

const transactionParameters =

{

to: contractRouterAddress,

from: window.ethereum.selectedAddress,

‘data’: contractRouter.methods.addLiquidity(address tokenA, address tokenB, uint amountADesired, uint amountBDesired, uint amountAMin, uint amountBMin, address to, uint deadline).encodeABI()

}

tokenA - адрес токена А

tokenB - адрес токена B

amountADesired - количество токенов A, которое пользователь хочет отправить(умноженное на 10\*\*18)

amountBDesired - количество токенов B, которое пользователь хочет отправить(умноженное на 10\*\*18)

amountAMin - минимальное количество токенов A, которое пользователь согласен предоставить. Считается с помощью slippage

amountBMin - минимальное количество токенов B, которое пользователь согласен предоставить. Считается с помощью slippage

address to - адрес пользователя

uint deadline - время до которого транзакция валидна. Считается с помощью deadline

Расчет параметров:

Пусть slipage = 1 %

То amountAMin = amountADesired(1 - slippage/100)

Аналогично и для amountBMin

Пусть deadline = 20 min

То deadline(в параметрах транзакции) = now() + 20 \* 60,

где now() - текущее время в Unix формате. 20\*60 - количество секунд в 20 минутах

// addLiquidityETH

const transactionParameters =

{

to: contractRouterAddress,

from: window.ethereum.selectedAddress,

value: количество нативного токена, которое хочет отправить пользователь

‘data’: contractRouter.methods.addLiquidityETH(address token, uint amountTokenDesired, uint amountTokenMin, uint amountETHMin, address to, uint deadline).encodeABI()

}

Параметры аналогичные:

address token - адрес токена, которые не нативный

amountTokenMin, amountETHMin - также вычисляются с помощью slippage

1. Если резервы ненулевые. То при заполнении верхнего поля, поля с токеном A, мы можем сразу вычислить нужное количество токенов B. Для этого вызываем у Router метод quote:

amountB = await contractRouter.methods.quote(amountA, reserveA, reserveB)

amountA - количество токена A, которое ввел пользователь

reserveA, reserveB - резервы, которые из пункта 2

Если пользователь заполнил нижнее поле, поле с токеном B. То действия аналогичные, но меняются параметры

amountA = await contractRouter.methods.quote(amountB, reserveB, reserveA)

1. После того, как определились с amountA, amountB вызываем аналогичные методы из пункта 3
2. Перед тем, как отправить транзакции, необходимо проверить allowance, как делали с Bridge. allowance для нативного токена не нужен. Адрес spender - адрес Router.

**Возвращение ликивдности**

1. Если пользователь хочет вернуть свои деньги, то необходимо вызвать метод removeLiquidity или если один из токенов нативный - removeLiquidityETH.
2. Перед этим нужно проверить allowance токена пары для адреса Router. Если нет allowance, то делаем approve
3. Рассмотрим метод removeLiquidity:

const transactionParameters = {

to: contractRouterAddress,

from: window.ethereum.selectedAddress,

‘data’: contractRouter.methods.removeLiquidity(address tokenA, address tokenB, uint liquidity, uint amountAMin, uint amountBMin, address to, uint deadline).encodeABI()

}

address tokenA - адрес токена A

address tokenB - адрес токена B

uint liquidity - количество токенов пары, которые пользователь хочет отдать взамен на токены A и B, которые предоставлял до этого

uint amountAMin - минимальное количество токенов A, которое должен получить пользователь

uint amountBMin - минимальное количество токенов B, которое должен получить пользователь

address to - адрес пользователя

uint deadline - аналогичный предыдущим методам

Расчет amounAMin и amountBMin:

Для расчета нам потребуются reserveA и reserveB и totalSupply.

Получить totalSupply пары:

totalSupply = await contractPair.methods.totalSupply().call()

amountAMin = liquidity \* reserveA / totalSupply \* (1 - slippage/100)

amountBMin = liquidity \* reserveB / totalSupply \* (1 - slippage/100)

1. Рассмотрим метод removeLiquidityETH:

consr transactionParameters = {

to: contractRouterAddress,

from: window.ethereum.selectedAddress,

‘data’: contractRouter.methods.removeLiquidityETH(address token, uint liquidity, uint amountTokenMin, uint amountMinETH, address to, uint deadline).encodeABI()

}

address token - адрес токена ненативного

uint liquidity - количество токенов пары, которые пользователь хочет отдать взамен на токены A и B, которые предоставлял до этого

uint amountTokenMin - минимальное количество токенов ненативного, которое должен получить пользователь

uint amountETHMin - минимальное количество токенов нативного, которое должен получить пользователь

address to - адрес пользователя

uint deadline - аналогичный предыдущим методам

**Рассмотрим работу с свопами**

1. Сначала получаем резервы, как это делали в разделе с ликвидностью
2. Рассмотрим, когда оба токена не нативные. Если пользователь заполнил верхнее поле, поле токена А, то вызываем метод, который поможет получить количество токена B:

amountB = await contractRouter.methods.getAmountOut(uint amountA, uint reserveA, uint reserveB).call()

1. После этого вызываем метод Router-а - swapExactTokensForTokens:

const transactionParams = {

to: contractRouterAddress,

from: window.ethereum.currentAddress,

‘data’: contractRouter.methods.swapExactTokensForTokens(uint amountIn, uint amountOutMin, address[] path, address to, uint deadline).encodeABI()

}

amountIn - amountA, которое ввел пользователь

amountOutMin - вычисляется с помощью amountB и slippage. amountOutMin = amountB(1 - slippage/100)

path - массив, такого вида [tokenA.address, tokenB.address]

to - адрес пользователя

deadline - deadline работает также, как и в разделе с ликвидностью

1. Если пользователь заполняет нижнее поле, поле с токеном B. Вызываем следующий метод для получения количества токенов A:

amountA = await contractRouter.methods.getAmountIn(uint amountB, uint reserveA, uint reserveB)

1. После этого вызываем метод Router-a - swapTokensForExactTokens:

const transactionParams = {

to: contractRouterAddress,

from: window.ethereum.currentAddress,

‘data’: contractRouter.methods.swapTokensForExactTokens(uint amountOut, uint amountInMax, address[] path, address to, uint deadline).encodeABI()

}

amountOut - amountB, которое ввел пользователь

amountInMax - вычисляется с помощью slippage и amountA

amountInMax = amountA(1 + slippage/100)

path = [tokenA.address, tokenB.address]

Остальное аналогично

1. Рассмотрим работу, когда один из токенов нативный. Принцип и методы вычисления amountA, amountB остаются такими же. Рассмотрим, какие методы нужно вызывать и в каких случаях
2. Когда tokenA - нативный токен и пользователь вводит его. Вызываем метод swapExactETHForTokens:

const transactionParams = {

to: contractRouterAddress,

from: window.ethereum.currentAddress,

value: amountA,

‘data’: contractRouter.methods.swapExactETHForTokens(uint amountOutMin, address[] path, address to, uint deadline).encodeABI()

}

amountOutMin = amountB(1 - slippage/100)

path = [WBWB.address, tokenB.address]

Остальное аналогично

1. Когда токен B - нативный токен и пользователь вводит его. Вызываем метод swapTokensForExactETH

const transactionParams = {

to: contractRouterAddress,

from: window.ethereum.currentAddress,

‘data’: contractRouter.methods.swapExactETHForTokens(uint amountOut, uint AmountInMax, address[] path, address to, uint deadline).encodeABI()

}

amountOut = amountB

amountInMax = amountA(1 + slippage/100)

path = [tokenA.address, WBNB.address]

Остальное аналогично

1. Когда токен B - нативный токен, но пользователь вводит токен А. Вызываем метод swapExactTokensForETH.

const transactionParams = {

to: contractRouterAddress,

from: window.ethereum.currentAddress,

‘data’: contractRouter.methods.swapExactETHForTokens(uint amountIn, uint AmountOutMin, address[] path, address to, uint deadline).encodeABI()

}

amountIn = amountA

amountOutMin = amountB(1 - slippage/100)

path = [tokenA.address, WBNB.address]

Остальное аналогично

1. Когда токен A - нативный токен, но пользователь вводит токен B. Вызываем метод swapETHForExactTokens.

const transactionParams = {

to: contractRouterAddress,

from: window.ethereum.currentAddress,

value: amountAMax,

‘data’: contractRouter.methods.swapExactETHForTokens(uint amountOut, address[] path, address to, uint deadline).encodeABI()

}

amountAMax = amountA(1 + slippage/100)

amountOut = amountB

path = [WBWB.address, tokenB.address]

Остальное аналогично

1. Также существуют токены, которые берут комиссии во время операций с ними. Работают они аналогично, только названия методов заканчиваются на SupportingFeeOnTransferTokens. Пока их не рассматриваемый, так как нужно заранее внести в базу токен - берет комиссии или нет.

1. Также перед транзакцией нужно убедиться, что amountB < reserveB. Иначе выдать ошибку о нехватке ликвидности
2. Перед тем, как отправить транзакции, необходимо проверить allowance, как делали с Bridge. allowance для нативного токена не нужен. Адрес spender - адрес Router.
3. Расчет price impact:

Price Impact - показывает насколько ожидаемая цена покупки, отличается от фактической

В случае, когда пользователь точно указывает количество токена А:

amountA

amount\_desired\_b = amountA \* (reserveB / reserveA)

price\_target\_for\_b = amountA / amount\_desired\_b

amountB - количество токенов B, которые мы получили с помощью метода getAmountOut

price\_actual\_for\_b = amountA / amountB

Price Impact = 100 \* price\_actual\_for\_b / price\_target\_for\_b - 100

В случае, когда пользователь точно указывает количество токена B:

amountB

amount\_target\_a = amountB \* (reserveA / reserveB)

price\_target\_for\_b = amount\_target\_a / amountB

amountA - количество токенов A, которые мы получили с помощью метода getAmountIn

price\_actual\_for\_b = amountA / amountB

Price Impact = 100 \* price\_actual\_for\_b / price\_target\_for\_b - 100

Полезные ссылки:

<https://dailydefi.org/articles/price-impact-and-how-to-calculate/> - расчет Price Impact