Documentation for Staking Contract

I. Information about Pools: -

1. Each pool is identified by a three key pair: stakeToken + rewardToken + poolIndex.
2. stakeToken and rewardToken are self-explanatory.
3. poolIndex is a number GREATER than 0, and it is a consecutive number for pools with SAME stakeToken and rewardToken.
4. poolIndex for a pool is generated automatically when the pool is created.

II. How does pools work: -

1. First, a pool is created by specifying stakeToken, rewardToken, rewardPerBlock, and a few other parameters.
2. Next, the pool needs to be funded with rewardToken. While doing so, a startBlock needs to be specified.
    a. startBlock is the block number from which the rewards start to get distributed to the stakers.
    b. This step needs to be performed within the staleBlockDuration from the point of creation of the pool.
3. Now, based on the amount funded, the contract will calculate the endBlock for the pool.
    a. endBlock is the block number till which the rewards are distributed to the stakers, after which the pool gets ended.
    b. $\text{endBlock} = \text{startBlock} + \frac{\text{amount funded}}{\text{rewardPerBlock}}$
4. Now, people can come and stake with the pool once the current block number on the blockchain becomes >= startBlock.
5. For each block, the reward distributed will be rewardPerBlock. This will be distributed among staked in proportion to their staking amount and the duration for staking.
6. Once the current block number on the blockchain becomes >= endBlock, people will no longer be able to stake with the pool.
7. Stakers can unstake from the pool anytime, once their staking duration has exceeded the min. no. of blocks for which the stakeToken is locked in the pool.
8. After the pool has ended, the poolIndex counter will be increment by 1, and then the new incremented value will be the poolIndex of the next pool if it is created with the same stakeToken and rewardToken.
9. If the pool has not ended, its duration can be increased by funding the pool with more rewardToken.

III. Fees: -

1. The contract has a global public variable called gasAmount.
    a. When pools are created, the pool-specific gasAmount is by default set to this global gasAmount.
    b. This gasAmount has a default value of 0.005 BNB / ETH / MATIC (unit depends upon the chain on which the contract is deployed).
    c. When users stake or unstake, the value of the transaction must be >= the pool-specific gasAmount.
2. Each pool has a deposit fee and a withdrawal fee associated with it.
    a. By default, deposit fee = 0‰ and withdrawal fee = 20‰ for every pool.
    b. The fee is either calculated over the deposit amount of stakeToken or the withdrawal amount of stakeToken during staking or un-staking respectively.
3. All fees generated as part of gasAmount or via deposit/withdrawal fee are stored in the staking contract and can be withdrawn by the owner at any time.

IV. Read Functions / Variables: -

Sr. No.	Name	Inputs	Outputs
1.	accessToken	-	Address of the ERC20 Token that must be held by the stakers before staking.
2.	activePools	a. *Index* (uint256) - Position in the array	Pool Identifier present at the input index in the active pools array.
3.	allPoolsBasicInfo	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool index of the pool.	Basic Pool Information of the pool with the input stakeToken, rewardToken, and the poolIndex.
4.	allPoolsDetailedInfo	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool index of the pool.	Detailed Pool Information of the pool with the input stakeToken, rewardToken, and the poolIndex.
5.	currentBlock	-	The latest block number of the blockchain.
6.	currentPoolToBeUpdated	-	Index of the pool in the active pools array which will be updated next during mass updating of active pools.
7.	defaultDepositFee	-	Permille default deposit fee used when pools are created.
8.	defaultWithdrawFee	-	Permille default withdraw fee used when pools are created.
9.	endedPools	*a. Index* (uint256) - Position in the array	Pool Identifier present at the input index in the ended pools array.
10.	gasAmount	-	Value of default global gasAmount.
11.	getActivePoolCount	-	Number of active pools
12.	getEndedPoolCount	-	Number of ended pools
13.	getNFTInfo	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool index of the pool. d. *TokenId* (uint256) - Id of the NFT.	Information associated with a NFT such as amount staked, initial deposit block, last deposit block, actual withdrawn rewards and effective withdrawn rewards.
14.	getNFTStakedAmount	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. *c. poolIndex* (uint256) - Pool index of the pool. *d. TokenId* (uint256) - Id of the NFT.	Amount of stakeToken associated with input NFT in the pool with the input stakeToken, rewardToken, and the poolIndex.
15.	getPendingRewardsOfNFT	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. *c. poolIndex* (uint256) - Pool index of the pool. *d. TokenId* (Uint256) - Id of the NFT.	Amount of reward that the NFT holder has earned from the pool with the input stakeToken, rewardToken, and the poolIndex.
16.	getPendingRewardsOfPool	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool index of the pool.	Amount of reward that is yet to be distributed to the stakers of the pool with the input stakeToken, rewardToken, and the poolIndex.
17.	hasSetNFTContract	-	A Boolean value signifying whether the owner has specified the NFT contract or not.
18.	indicesOfActivePools	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool index of the pool.	Index of the pool with the input stakeToken, rewardToken, and the poolIndex in the active pools array.
19.	indicesOfEndedPools	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool index of the pool.	Index of the pool with the input stakeToken, rewardToken, and the poolIndex in the ended pools array.
20.	latestPoolNumber	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token.	Returns the latest poolIndex for given pair of stakeToken and rewardToken. 0 => No pool created yet. Else, If a pool is active, then its poolIndex is returned. Otherwise, it returns the poolIndex that will be assigned to the next pool if created in the future.
21.	massUpdatePoolCount	-	The number of pools that are updated during mass pool update.
22.	minAccessTokenRequired	-	The minimum amount of access tokens that a staker must hold for them to be eligible for staking with any pool.
23.	nftContract	-	Address of the NFT Smart contract.

24.	owner	-	The address of the owner of the contract.
25.	requireAccessToken	-	A true/false value indicating that whether stakers must hold the minimum amount of access token to stake in any pool.
26.	staleBlockDuration	-	The number of stale blocks.
27.	treasury	-	The address of the treasury.
28.	withdrawableFee	*a. tokenAddress* (address) - Address of an ERC20 token.	The amount of input token that can be withdrawn from the contract by the owner.

V. Write Functions: -

Sr. No.	Name	Inputs	Description
1.	adjustGasGlobal	a. *amount* (uint256) - gasAmount with decimal 0s.	Can be called by the owner only. It sets the value of the global gasAmount to the amount passed as input.
2.	adjustPoolGas	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *amount* (uint256) - gasAmount with decimal 0s.	Can be called by the owner only. It sets the value of gasAmount to the amount passed as input for the pool with the input stakeToken and rewardToken at the LATEST index if it exists.
3.	changeDepositFee	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *amount* (uint256) - Permille fee to be charged.	Can be called by the owner only. It sets the value of the deposit fee to the amount passed as input for the pool with the input stakeToken and rewardToken at the LATEST index if it exists.
4.	changePoolMaxStakers	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *amount* (uint256) - Max. number of stakers.	Can be called by the owner only. It sets the value of max. stakers to the amount passed as input for the pool with the input stakeToken and rewardToken at the LATEST index if it exists.
5.	changeTreasury	a. *_address* (address) - Address of the new Treasury	Can be called by the owner only. It sets the address of the treasury to the address passed as input.
6.	changeWithdrawFee	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *amount* (uint256) - Permille fee to be charged.	Can be called by the owner only. It sets the value of the withdrawal fee to the amount passed as input for the pool with the input stakeToken and rewardToken at the LATEST index if it exists.
7.	createNewStakingPool	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *rewardPerBlock* (uint256) - Permille fee to be charged. d. *minStake* (uint256) - Min. tokens that can be staked. e. *maxStake* (uint256) - Max. tokens that can be staked. f. *maxStakers* (uint256) - Max. number of stakers allowed.	Can be called by the owner only. Creates a new pool with given input and returns the poolIndex of the created pool. If the input value of maxStake or maxStakers is 0, then they are set to infinite.
8.	emergencyWithdraw	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool index of the pool. d. *TokenId* (uint256) - Token Id of the NFT.	The function caller will get back their staked token for the input token Id from the specified pool WITHOUT any rewards. This withdrawal ignores whether the tokens have been staked for min. amount of lock blocks not. After withdrawal, the NFT is burnt.
9.	emergencyWithdraw	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool index of the pool.	Similar to above function, but token Id is not required to be specified. Instead, the contract will detect the list of all NFTs associated with the function caller for the pool with input details, and selects the first NFT of the list for withdrawal.
10.	increasePoolFunding	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *amount* (uint256) - Amount to fund the pool with.	This function allows anyone to fund the LATEST pool (if it exists) with more reward tokens thereby increasing the running time of the pool. It can only be

			used if the initial funding of the pool has been completed and the pool is still active.
11.	massUpdatePoolStatus	-	Mass updates the status of active pools. The number of pools updated in such a way depends on the value of massUpdatePoolCount.
12.	performInitialFunding	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *amount* (uint256) - Amount to fund the pool with. d. *startBlock* (uint256) - When reward starts distributing. e. *lockBlocks* (uint256) - Min. blocks to stake for.	This function allows anyone to fund the LATEST pool (if it exists) with the rewardToken. Consequently, the contract calculates the endBlock for the pool. startBlock <= (current block number + staleBlockDuration). If startBlock is less than the current block number, then the start block is set to the current block number. lockBlocks cannot be greater than (endBlock - startBlock).
13.	renounceOwnership	-	Can be called by the owner only. Renounces the ownership of the contract and sets zero address as the new owner of the contract. This step is irreversible.
14.	scrapeFees	a. *startIndex* (uint256) - Starting index. b. *endIndex* (uint256) - Ending index.	Iterates over the ended pools array from starting index till the ending index provided as input to collect any left-over reward tokens as fees and make them withdrawable by the owner. 0 < startIndex <= endIndex < (total number of endedPools)
15.	setAccessToken	a. *accessToken* (address) - Address of the reward token.	Can be called by the owner only. Sets the input ERC20 Token as the new access token for staking.
16.	setDefaultDepositFee	a. *amount* (uint256) - Permille fee to be charged.	Can be called by the owner only. It sets the value of the default deposit fee that is assigned to pool when they are created.
17.	setDefaultWithdrawFee	a. *amount* (uint256) - Permille fee to be charged.	Can be called by the owner only. It sets the value of the default withdraw fee that is assigned to pool when they are created.
18.	setMassUpdatePoolCount	a. *amount* (uint256) - Number of pools to be mass updated.	Can be called by the owner only. Sets the maximum number of pools that get auto-updated by setting the massUpdatePoolCount to the input amount.
19.	setMinAccessTokenRequired	a. *amount* (uint256) - Min. amount of accessToken required.	Can be called by the owner only. Sets the input amount as the new minimum amount of access token that stakers needs to hold for staking.
20.	setNFTContract	a. *NFTContract* (address) - Address of the NFT Contract.	Can be called by the owner only. Can be called only ONCE. It sets the address of the NFT contract that the staking contract should use for representing staking positions.
21.	setRequireAccessToken	a. *value* (bool) - Should require access token.	Can be called by the owner only. Sets the input true/false value that enables/disables the check whether the staker has the minimum amount of access token or not.
22.	setStaleBlockDuration	a. *amount* (uint256) - Number of blocks.	Can be called by the owner only. It sets the number of stale blocks to the amount passed as input.
23.	stakeWithPool	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *amount* (uint256) - Amount to stake with the pool.	Users need to call this function to stake with the pool. First, the user needs to approve the input amount of stakeToken to the staking contract and then call this function. The value for the transaction must be at least the gasAmount for the pool. When this function is called, the amount is added to the first NFT that the user holds for that pool. If the user

			does not hold such NFT, then a new NFT is created and sent to the function caller.
24.	transfer	-	Can be called by the owner only. When this function is called, all the BNB / ETH / MATIC present with the staking contract is transferred to the treasury wallet.
25.	transferOwnership	a. *owner* (address) - Address of new owner.	Can be called by the owner only. Transfers the ownership of the contract to the new address provided as input during the function call.
26.	unstakeFromPool	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool Index of the pool. d. *amount* (uint256) - Amount to stake with the pool. e. *TokenId* (uint256) - Token Id of the NFT.	The input amount of stakeToken associated with the input NFT token Id will be returned to the function (provided they hold the NFT in their wallet) **AND MEANWHILE** also transfers the rewards accumulated by the staker. If the stake amount associated with the NFT becomes 0 after withdrawal, then the NFT is burnt. If the input amount is 0, then only the pending rewards are transferred to the function caller.
27.	unstakeFromPool	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. c. *poolIndex* (uint256) - Pool Index of the pool. d. *amount* (uint256) - Amount to stake with the pool.	Similar to above function, but token Id is not required to be specified. Instead, the contract will detect the list of all NFTs associated with the function caller for the pool with input details, and selects the first NFT of the list for unstaking.
28.	updatePoolStatus	a. *stakeToken* (address) - Address of the staking token. b. *rewardToken* (address) - Address of the reward token. *c. poolIndex* (uint256) - Pool Index of the pool.	Updates the pool by updating the value of the reward that needs to be distributed to stakers for every stakeToken staked with the pool based on the number of blocks elapsed from the last update. It also updates whether the pool has ended or not.
29.	withdrawFees	a. *token* (address) - Address of token to withdraw. b. *receiver* (address) - Address of the receiver wallet. c. *amount* (uint256) - Amount of fees to withdraw.	Can be called by the owner only. If the input amount is less than or equal to the fees generated till now for the input token, then the input amount of the input token is transferred to the specified receiver address.

VI. Notes: -

Terminology	Meaning
‰	Parts per thousand. (Read as - Permille). (**SIMILAR** to %, which means parts per hundred).
Pool Identifier	A structure that contains three values: stakeToken, rewardToken, poolIndex. It helps in uniquely identifying a pool.
Basic Pool Information	A structure containing basic information of a pool. It contains: - a. *doesExists* - Indicates whether a pool exists or not. b. *hasEnded* - Indicates whether a pool has ended or not. c. *stakeToken* - Token to be staked. d. *rewardToken* - Token in which reward is distributed. e. *createBlock* - Block Number when the pool was created. f. *startBlock* - Block Number when the reward starts getting distributed. g. *rewardPerBlock* - Amount of rewardToken distributed to holders per block. h. *gasAmount* - Pool specific gasAmount. i. *minStake* - Min. tokens that users need to stake to participate. j. *maxStake* - Max. tokens that users can stake with the pool. k. *stakeTokenDepositFee* - Permille fee calculated over deposit amount. l. *stakeTokenWithdrawFee* - Permille fee calculated over withdraw amount. m. *lockPeriod* - Number of blocks for which the user cannot withdraw their tokens.
Detailed Pool Information	A structure containing detailed information of a pool. It contains: - a. *tokensStaked* - Total number of tokens stake with the pool. b. *accRewardPerTokenStaked* - Accumulative reward per token staked with the pool. c. *paidOut* - Total reward transferred to the stakers. d. *lastRewardBlock* - Last block number when accRewardPerTokenStaked was updated. e. *endBlock* - Block number when the pool ends. f. *maxStakers* - Maximum number of stakers allowed to stake with the pool. g. *totalStakers* - Total number of stakers who have staked tokens in the pool presently.
Stale Blocks	It is just number of blocks. Its default value is 1000. If a pool is created and not funded within 1000 blocks, then the pool is automatically ended. When pools are funded initially, then startBlock can at most be 1000 block more than current block.
Treasury	When contract is deployed, treasury is set to the owner wallet.

VII. Deployment Details: -

a. Language : Solidity
b. Compiler Version : 0.8.11
c. Enable Optimization : YES
d. Optimization Runs : 1000

VIII. Deployment Process: -

1. Deploy the staking contract using the above-mentioned deployment details. There are no input parameters that needs to be specified for constructor during deployment of the staking contract.
2. Deploy the NFT contract using the above-mentioned deployment details. There are 4 parameters that need to be specified: -
   a. *Name* (string) - Name of the NFT.
   b. *Symbol* (string) - Symbol of the NFT.
   c. *Minter* (address) - Address of the minter. This HAS to be the address of the staking contract deployed in above step.
   d. *ImageURL* (string) - DIRECT LINK of the image that should be shown for the NFT.
3. Copy the address of the NFT Contract that was deployed in the above step and call the setNFTContract function of the staking contract using the copied NFT contract address as the input parameter, from the owner's wallet.