Contract Overview DeMarket

State Variables

itemCount

```
uint256 public itemCount;
```

users

```
mapping(address user => User) public users;
```

items

```
mapping(uint256 itemId => Item) public items;
```

userBalance

```
mapping(address user => uint256 balance) public userBalance;
```

Functions

onlyRegisteredUser

Reverts with `UserNotRegistered` if the caller is not registered.

```
modifier onlyRegisteredUser();
```

registerUser

Reverts with `UserAlreadyRegistered` if the user already exists.

```
function registerUser(string memory _username) external;
```

Parameters

Name	Type	Description
_username	string	The username for the new user.

listItem

Lists a new item for sale.

```
function listItem(string memory _name, string memory _description,
uint256 _price) external onlyRegisteredUser;
```

Parameters

Name	Type	Description
_name	string	The name of the item.
_description	string	The description of the item.
_price	uint256	The price of the item in wei.

purchaseItem

Purchases an item.

```
function purchaseItem(uint256 _itemId) external payable
onlyRegisteredUser;
```

Parameters

Name	Type	Description
_itemId	uint256	The ID of the item to purchase.

withdrawFunds

Withdraws accumulated funds.

```
function withdrawFunds() external;
```

getItemInfo

Returnes information about specific item.

```
function getItemInfo(uint256 _itemId) external view returns (string
memory, string memory, uint256, bool, address);
```

Parameters

Name	Type	Description
_itemId	uint256	The ID of the item.

Returns

Name	Type	Description
<none>	string	name Name of the item.
<none>	string	description Description of the item.
<none>	uint256	price Price of the item in wei.
<none>	bool	available Whether the item is available for purchase.
<none>	address	owner The address of the item's owner.

getUserPurchaseHistory

Retrieves the purchase history of a user.

```
function getUserPurchaseHistory(address _userAddress) external view
onlyRegisteredUser returns (uint256[] memory);
```

Parameters

Name	Type	Description
_userAddress	address	The address registered user.

Returns

Name	Type	Description
<none>	uint256[]	An array of item IDs representing the user's purchase history.

Events

UserRegistered

```
event UserRegistered(address indexed userAddress, string username);
```

ItemListed

```
event ItemListed(uint256 indexed itemId, string name, uint256 price,
address indexed owner);
```

ItemPurchased

```
event ItemPurchased(uint256 indexed itemId, address indexed buyer,
address indexed seller);
```

FundsWithdrawn

```
event FundsWithdrawn(address indexed user, uint256 amount);
```

Errors

UserNotRegistered

```
error UserNotRegistered();
```

NoFundsToWithdraw

```
error NoFundsToWithdraw();
```

WithdrawFailed

```
error WithdrawFailed();
```

UserAlreadyRegistered

```
error UserAlreadyRegistered();
```

ItemNotAvailable

```

```

```
error ItemNotAvailable();
```

InsufficientFunds

```
error InsufficientFunds();
```

CannotPurchaseOwnItem

```
error CannotPurchaseOwnItem();
```

ItemAlreadyListed

```
error ItemAlreadyListed();
```

Structs

User

Represents a user with a username, existence status, and purchase history.

```
struct User {
    string username;
    bool exists;
    uint256[] purchaseHistory;
}
```

Properties

Name	Type	Description
`username`	`string`	The user's registered username.
`exists`	`bool`	A bool indicating if the user exists.
`purchaseHistory`	`uint256[]`	An array of user's purchase history.

Item

Represents an item with details like name, description, price, availability, and owner.

Struct to store item details.

```
struct Item {
    string name;
    string description;
    uint256 price;
    bool available;
    address owner;
}
```

Properties

Name	Type	Description
`name`	`string`	The name of the item.
`description`	`string`	The description of the item.
`price`	`uint256`	The price of the item in wei.
`available`	`bool`	A boolean indicating if the item is available for purchase.
`owner`	`address`	The address of the item's owner.

** Created with forge doc

Contract Overview OTCSwap

State Variables

swaps

```
mapping(bytes32 => Swap) public swaps;
```

Functions

createSwap

Initiates a swap with a specified counterparty, token pair, and amounts.

The expiration time must be in the future at the time of swap creation.

```
function createSwap(
    address _counterparty,
    address _tokenXAddress,
    address _tokenYAddress,
    uint256 _amountX,
    uint256 _amountY,
```

```
    uint256 _expirationTime
) external returns (bytes32);
```

Parameters

Name	Type	Description
_counterparty	address	The address of the counterparty who can accept the swap.
_tokenXAddress	address	The contract address of token X being offered.
_tokenYAddress	address	The contract address of token Y being requested.
_amountX	uint256	The amount of token X being offered by the initiator.
_amountY	uint256	The amount of token Y expected from the counterparty.
_expirationTime	uint256	The timestamp by which the swap must be executed.

Returns

Name	Type	Description
<none>	bytes32	swapId A unique identifier for the newly created swap.

executeSwap

Executes a swap, transferring the specified tokens.

The function can only be called by the counterparty within the expiration time.

```
function executeSwap(bytes32 _swapId) external;
```

Parameters

Name	Type	Description
_swapId	bytes32	The identifier of the swap to be executed.

cancelSwap

Cancels a swap agreement that has not been executed yet.

Only the initiator of the swap can cancel it.

```
function cancelSwap(bytes32 _swapId) external;
```

Parameters

Name	Type	Description
_swapId	bytes32	The unique identifier of the swap to be canceled.

Events

SwapCreated

```
event SwapCreated(bytes32 indexed swapId, address indexed initiator,
address indexed counterparty);
```

SwapExecuted

```
event SwapExecuted(bytes32 indexed swapId);
```

SwapCancelled

```
event SwapCancelled(bytes32 indexed swapId);
```

Errors

SwapAlreadyExecuted

```
error SwapAlreadyExecuted();
```

SwapExpired

```
error SwapExpired();
```

OnlyCounterpartyCanExecute

```
error OnlyCounterpartyCanExecute();
```

OnlyInitiatorCanCancel

```
error OnlyInitiatorCanCancel();
```

ExpirationMustBeFuture

```
error ExpirationMustBeFuture();
```

Structs

Swap

Represents a swap agreement between two parties.

```
struct Swap {
    address initiator;
    address counterparty;
    address tokenXAddress;
    address tokenYAddress;
    uint256 amountX;
    uint256 amountY;
    uint256 expirationTime;
    bool executed;
}
```

Properties

Name	Type	Description
`initiator`	`address`	The address of the user initiating the swap.
`counterparty`	`address`	The address of the counterparty who can execute the swap.
`tokenXAddress`	`address`	The address of the token that the initiator offers.
`tokenYAddress`	`address`	The address of the token that the counterparty offers.
`amountX`	`uint256`	The amount of token X to be swapped.
`amountY`	`uint256`	The amount of token Y to be swapped.
`expirationTime`	`uint256`	The time until which the swap is valid.
`executed`	`bool`	A boolean indicating whether the swap has been executed.

** Created with forge doc

For design explanations please review README.md.