NFT Market place Solidity Smart Contract Break Down

1. Dependencies and Setup

solidity
Copy code

```
import "@openzeppelin/contracts/utils/Counters.sol";
import
"@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol"
;
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "hardhat/console.sol";
```

- **`@openzeppelin/contracts`**: Provides reusable smart contract components.
  - `Counters`: A utility for incrementing counters (e.g., token IDs).
  - `ERC721URIStorage`: Extension of the ERC721 standard for storing token metadata (e.g., URI for images or assets).
  - `ERC721`: Standard interface for non-fungible tokens (NFTs).
- **`console.sol`**: Enables debugging in Hardhat.

---

2. Contract State Variables

solidity
Copy code

```
using Counters for Counters.Counter;
Counters.Counter private _tokenIds;
Counters.Counter private _itemsSold;

uint256 listingPrice = 0.025 ether;
address payable owner;

mapping(uint256 => MarketItem) private idToMarketItem;
```

- **`_tokenIds`**: Tracks the total number of NFTs minted.
- **`_itemsSold`**: Tracks the number of NFTs sold.
- **`listingPrice`**: Fixed fee for listing an NFT on the marketplace.
- **`owner`**: The contract owner (set during deployment).
- **`idToMarketItem`**: Maps token IDs to `MarketItem` structs.

---

3. Structs and Events

Struct: `MarketItem`

solidity
Copy code

```
struct MarketItem {
  uint256 tokenId;
  address payable seller;
  address payable owner;
  uint256 price;
  bool sold;
}
```

- Represents a marketplace item with details such as seller, owner, price, and whether it's sold.

Event: `MarketItemCreated`

solidity
Copy code

```
event MarketItemCreated (
  uint256 indexed tokenId,
  address seller,
  address owner,
  uint256 price,
  bool sold
);
```

- Emits whenever a new marketplace item is created.

---

4. Constructor

solidity
Copy code
```
constructor() ERC721("Metaverse Tokens", "METT") {
  owner = payable(msg.sender);
}
```

- Sets the contract owner and assigns the NFT name (`Metaverse Tokens`) and symbol (`METT`).

---

5. Marketplace Functions

a) Update and Retrieve Listing Price

solidity
Copy code
```
function updateListingPrice(uint _listingPrice) public payable {
  require(owner == msg.sender, "Only marketplace owner can update
listing price.");
  listingPrice = _listingPrice;
}

function getListingPrice() public view returns (uint256) {
  return listingPrice;
}
```

- **`updateListingPrice`**: Allows only the contract owner to update the listing price.
- **`getListingPrice`**: Fetches the current listing price.

---

b) Mint and List an NFT

solidity
Copy code
```
function createToken(string memory tokenURI, uint256 price) public
payable returns (uint) {
  _tokenIds.increment();
  uint256 newTokenId = _tokenIds.current();

  _mint(msg.sender, newTokenId);
```

```
  _setTokenURI(newTokenId, tokenURI);
  createMarketItem(newTokenId, price);
  return newTokenId;
}
```

- **Mint**: Creates a new NFT with a unique token ID and metadata URI.
- **List**: Adds the NFT to the marketplace by calling `createMarketItem`.

---

c) Create a Market Item

solidity
Copy code

```
function createMarketItem(uint256 tokenId, uint256 price) private {
  require(price > 0, "Price must be at least 1 wei");
  require(msg.value == listingPrice, "Price must be equal to listing
price");

  idToMarketItem[tokenId] = MarketItem(
    tokenId,
    payable(msg.sender),
    payable(address(this)),
    price,
    false
  );

  _transfer(msg.sender, address(this), tokenId);
  emit MarketItemCreated(tokenId, msg.sender, address(this), price,
false);
}
```

- Validates the price and listing fee.
- Updates the `idToMarketItem` mapping and transfers ownership to the contract.

---

d) Resell a Token

solidity
Copy code

```
function resellToken(uint256 tokenId, uint256 price) public payable {
  require(idToMarketItem[tokenId].owner == msg.sender, "Only item
owner can perform this operation");
  require(msg.value == listingPrice, "Price must be equal to listing
price");

  idToMarketItem[tokenId].sold = false;
  idToMarketItem[tokenId].price = price;
  idToMarketItem[tokenId].seller = payable(msg.sender);
  idToMarketItem[tokenId].owner = payable(address(this));

  _itemsSold.decrement();
  _transfer(msg.sender, address(this), tokenId);
}
```

- Allows owners to relist their NFTs for resale.

---

e) Buy an NFT (Create a Market Sale)

solidity

Copy code

```
function createMarketSale(uint256 tokenId) public payable {
  uint price = idToMarketItem[tokenId].price;
  address seller = idToMarketItem[tokenId].seller;

  require(msg.value == price, "Please submit the asking price in order
to complete the purchase");

  idToMarketItem[tokenId].owner = payable(msg.sender);
  idToMarketItem[tokenId].sold = true;
  idToMarketItem[tokenId].seller = payable(address(0));

  _itemsSold.increment();
  _transfer(address(this), msg.sender, tokenId);

  payable(owner).transfer(listingPrice);
  payable(seller).transfer(msg.value);
}
```

- Handles the NFT sale by transferring funds and ownership.

---

f) Fetch Marketplace Data

Fetch Unsold Items

solidity

Copy code

```
function fetchMarketItems() public view returns (MarketItem[] memory)
{ ... }
```

1.
    - Returns all NFTs still owned by the contract.

Fetch User's Purchased NFTs

solidity

Copy code

```
function fetchMyNFTs() public view returns (MarketItem[] memory) { ...
}
```

2.
    - Lists NFTs owned by the caller.

Fetch User's Listed Items

solidity

Copy code

```
function fetchItemsListed() public view returns (MarketItem[] memory)
{ ... }
```

3.
    - Lists NFTs where the caller is the seller.

Key Features:

- **Minting**: Users can mint and list NFTs with metadata.
- **Market Listing**: Users can list and resell NFTs.
- **Ownership Transfers**: Ownership is updated upon sales.
- **Listing Fees**: Fees go to the marketplace owner.
- **Data Fetching**: Fetch details of unsold, owned, or listed NFTs.