# Project Report

# Web3 Music Player Application

**Work done by: Sinda Ben Marzouk & Bochra Feki & Rihab Cheberli & Anas Hamrouni**

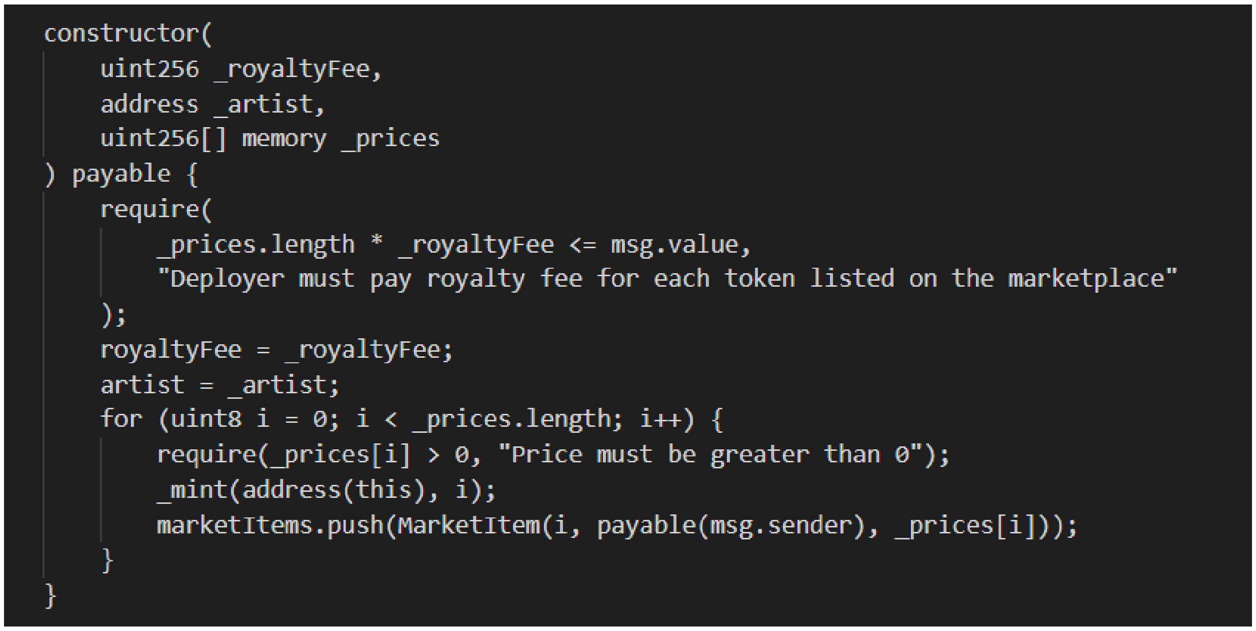
## Description:

This is a music player application like spotify. On the home page, we have a music player that will allow us to listen to any of the music nfts listed for sale. But what makes this player different from a regular music player is that the users can purchase the selected music token with ether, and once you have purchased one you can view it in my tokens page, where you can also play it, see the price that you purchased at and resell it using a price that you choose. We can view it later on my resales page with the price that we have listed at, and we can also view all the assets that have been sold.

Our application is build using Javascript for the react application and testing, solidity for writing Smart Contract, Ethers for the blockchain interaction, Hardhat as a development framework, IPFS for the metadata storage and React routers as navigational components.

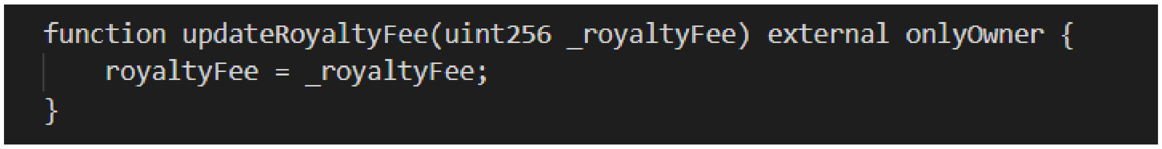
## Code of the Smart Contract:

In constructor we initialize royalty fee, artist address and prices of music nfts:



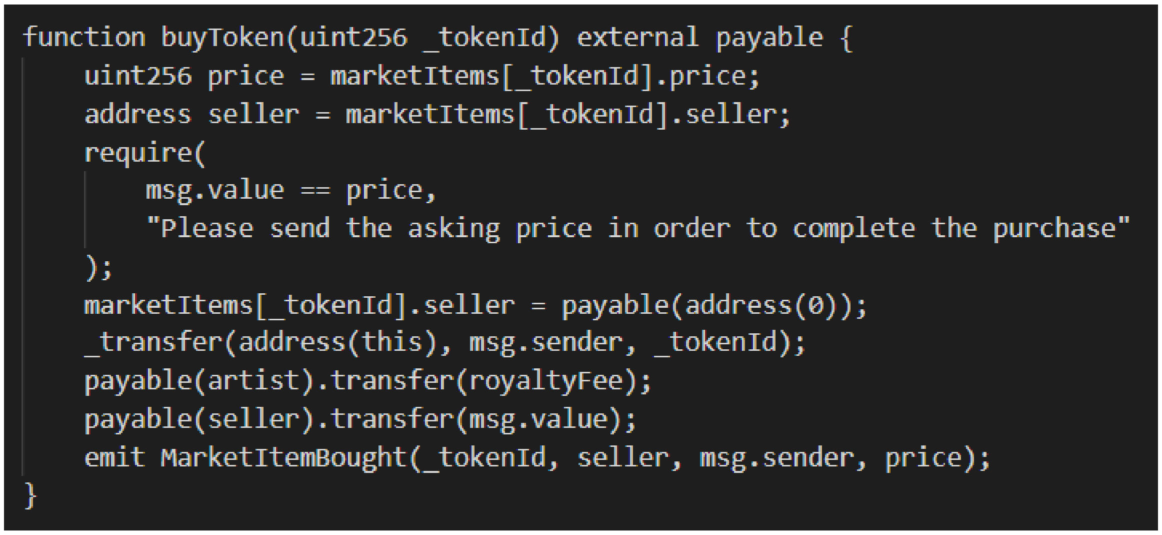
```
constructor(
    uint256 _royaltyFee,
    address _artist,
    uint256[] memory _prices
) payable {
    require(
        _prices.length * _royaltyFee <= msg.value,
        "Deployer must pay royalty fee for each token listed on the marketplace"
    );
    royaltyFee = _royaltyFee;
    artist = _artist;
    for (uint8 i = 0; i < _prices.length; i++) {
        require(_prices[i] > 0, "Price must be greater than 0");
        _mint(address(this), i);
        marketItems.push(MarketItem(i, payable(msg.sender), _prices[i]));
    }
}
```

This function allow us to update the royalty fee of the contract:



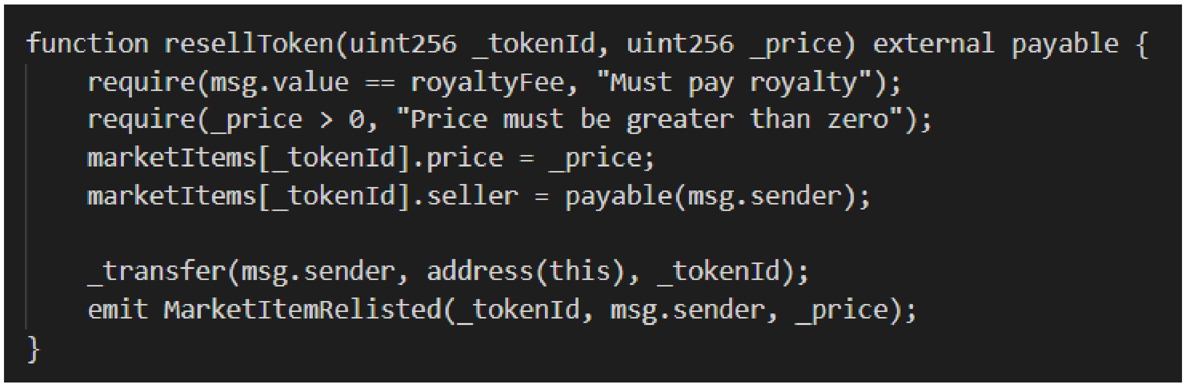
```
function updateRoyaltyFee(uint256 _royaltyFee) external onlyOwner {
    royaltyFee = _royaltyFee;
}
```

We can then create the sale of a music nft listed on the marketplace, and transfer ownership of the nft, as well as funds between parties:



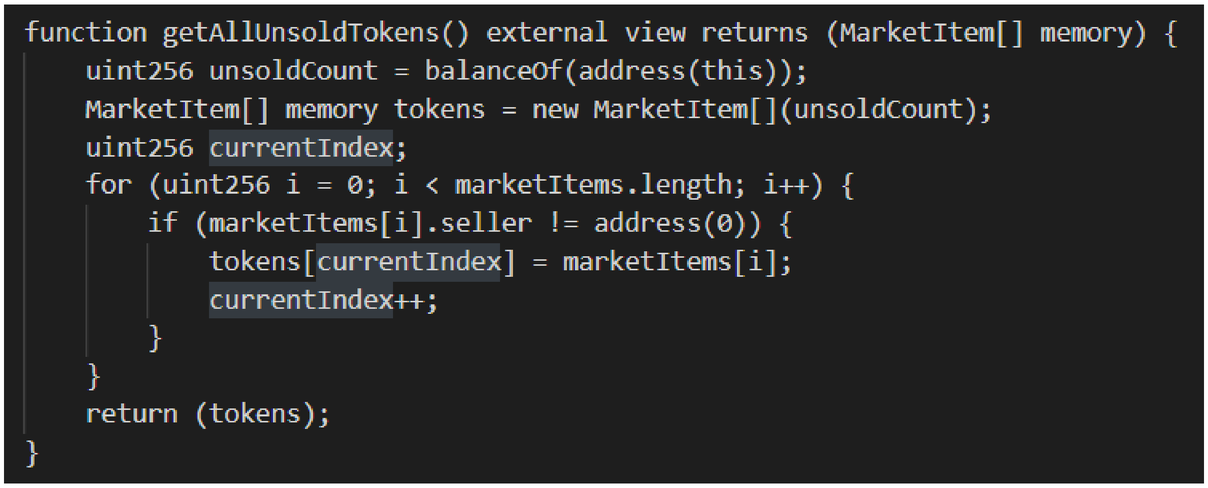
```
function buyToken(uint256 _tokenId) external payable {
    uint256 price = marketItems[_tokenId].price;
    address seller = marketItems[_tokenId].seller;
    require(
        msg.value == price,
        "Please send the asking price in order to complete the purchase"
    );
    marketItems[_tokenId].seller = payable(address(0));
    _transfer(address(this), msg.sender, _tokenId);
    payable(artist).transfer(royaltyFee);
    payable(seller).transfer(msg.value);
    emit MarketItemBought(_tokenId, seller, msg.sender, price);
}
```

This function allows someone to resell their music nft:



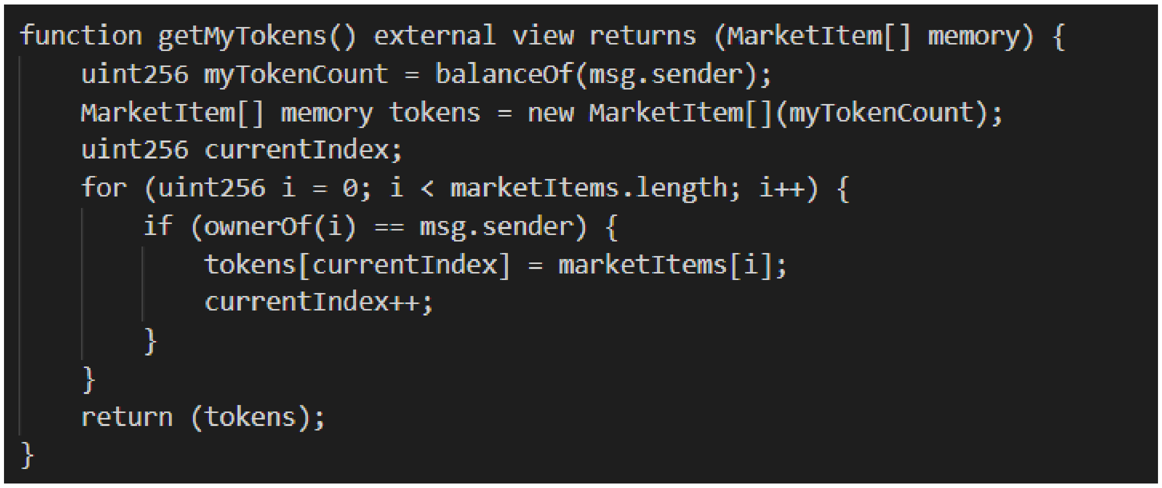
```
function resellToken(uint256 _tokenId, uint256 _price) external payable {
    require(msg.value == royaltyFee, "Must pay royalty");
    require(_price > 0, "Price must be greater than zero");
    marketItems[_tokenId].price = _price;
    marketItems[_tokenId].seller = payable(msg.sender);

    _transfer(msg.sender, address(this), _tokenId);
    emit MarketItemRelisted(_tokenId, msg.sender, _price);
}
```

We can also fetch all the tokens currently listed for sale:



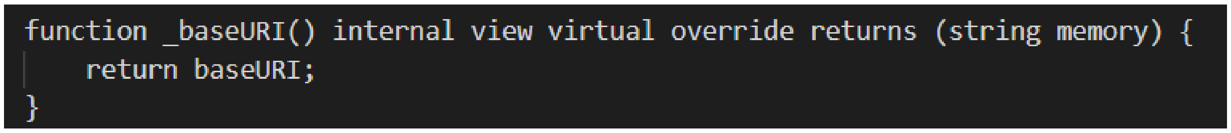
```
function getAllUnsoldTokens() external view returns (MarketItem[] memory) {
    uint256 unsoldCount = balanceOf(address(this));
    MarketItem[] memory tokens = new MarketItem[](unsoldCount);
    uint256 currentIndex;
    for (uint256 i = 0; i < marketItems.length; i++) {
        if (marketItems[i].seller != address(0)) {
            tokens[currentIndex] = marketItems[i];
            currentIndex++;
        }
    }
    return (tokens);
}
```

Or fetch all the tokens owned by the user:



```
function getMyTokens() external view returns (MarketItem[] memory) {
    uint256 myTokenCount = balanceOf(msg.sender);
    MarketItem[] memory tokens = new MarketItem[](myTokenCount);
    uint256 currentIndex;
    for (uint256 i = 0; i < marketItems.length; i++) {
        if (ownerOf(i) == msg.sender) {
            tokens[currentIndex] = marketItems[i];
            currentIndex++;
        }
    }
    return (tokens);
}
```

Finally, this internal function gets the baseURI initialized in the constructor:



```
function _baseURI() internal view virtual override returns (string memory) {
    return baseURI;
}
```