

ERC20Token.sol:

```
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract ERC20Token is ERC20 {

    constructor(
        string memory _name,
        string memory _symbol
    ) ERC20(_name, _symbol) {}

    function mint(address _to, uint256 _amount) external {
        _mint(_to, _amount);
    }
}
```

ERC721Token.sol:

```
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract ERC721Token is ERC721 {

    constructor(
        string memory _name,
        string memory _symbol
    ) ERC721(_name, _symbol) {}

    function mint(address _to, uint256 _tokenId) external {
        _safeMint(_to, _tokenId);
    }
}
```

combined.sol:

```
pragma solidity ^0.8.9;
```

```
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
```

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

```
contract ERC721Token is ERC721 {
```

```
    mapping(uint256 => uint256) private _tokenIdToAmount;
```

```
    mapping(uint256 => address) private _tokenIdToERC20;
```

```
    constructor(string memory _name, string memory _symbol) ERC721(_name, _symbol) {}
```

```
    function mint(address _to, uint256 _tokenId, address _erc20Token) external {
```

```
        require(_erc20Token != address(0), "wrong ERC20 address");
```

```
        _safeMint(_to, _tokenId);
```

```
        _tokenIdToAmount[_tokenId] = 1000;
```

```
        _tokenIdToERC20[_tokenId] = _erc20Token;
```

```
    }
```

```
    function transferERC721(uint256 _tokenId, address _to) external {
```

```
        require(_exists(_tokenId), "Token does not exist");
```

```
        require(ownerOf(_tokenId) == msg.sender, "for the owner");
```

```
        _transfer(msg.sender, _to, _tokenId);
```

```

    }

    function transferERC20(uint256 _tokenId, address _to) external {
        require(_exists(_tokenId), "Token does not exist");
        require(ownerOf(_tokenId) == msg.sender, "for the owner");

        address erc20Token = _tokenIdToERC20[_tokenId];

        uint256 amount = _tokenIdToAmount[_tokenId];

        require(erc20Token != address(0), "wrong ERC20 address");

        IERC20(erc20Token).transferFrom(msg.sender, _to, amount);
    }
}

```

Deployment(js):

```

const ERC721Token = artifacts.require('ERC721Token');
const ERC20Token = artifacts.require('ERC20Token');

module.exports = function (deployer)
{
    deployer.deploy(ERC721Token, 'MyERC721', 'ERC721');
    deployer.deploy(ERC20Token, 'MyERC20', 'ERC20');
};

```

Deployment:

The screenshot shows the VS Code interface with the Truffle project open. The terminal window displays the output of the `truffle compile` command. The output shows that the contracts were compiled successfully, with warnings about the SPDX license identifier not being provided in the source files. The artifacts were written to `D:\integrate_asap\build\contracts`, and the compilation was successful using `solc 0.8.9+commit.60900d3e`. The output also shows the duplicate contract names found for `ERC721Token` and the final cost of the deployment.

```
PS D:\integrate_asap truffle compile

Compiling your contracts...
-----
> Compiling .\contracts\ERC20Token.sol
> Compiling .\contracts\ERC20Token.sol
> Compiling .\contracts\ERC721Token.sol
> Compiling .\contracts\ERC721Token.sol
> Compiling .\contracts\combined.sol
> Compilation warnings encountered:

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> project\contracts\ERC20Token.sol

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> project\contracts\ERC721Token.sol

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> project\contracts\combined.sol

> Artifacts written to D:\integrate_asap\build\contracts
> Compiled successfully using:
  - solc: 0.8.9+commit.60900d3e:linux.gcc.linux.linux
  - Duplicate contract names found for ERC721Token.
    - This can cause errors and unknown behavior. Please rename one of your contracts.
  - PS D:\integrate_asap truffle migrate --network development

Compiling your contracts...
-----
> Compiling @openzeppelin/contracts/token/ERC20/ERC20.sol
> Compiling @openzeppelin/contracts/token/ERC20/IERC20.sol
> Compiling @openzeppelin/contracts/token/ERC20/utils/ERC20Metadata.sol
> Compiling @openzeppelin/contracts/token/ERC721/ERC721.sol
> Compiling @openzeppelin/contracts/token/ERC721/IERC721.sol
> Compiling @openzeppelin/contracts/token/ERC721/utils/ERC721Receiver.sol
> Compiling @openzeppelin/contracts/token/ERC721/utils/ERC721Metadata.sol
> Compiling @openzeppelin/contracts/utils/Address.sol
> Compiling @openzeppelin/contracts/utils/Context.sol
> Compiling @openzeppelin/contracts/utils/Strings.sol
> Compiling @openzeppelin/contracts/utils/Introspection/ERC165.sol
> Compiling @openzeppelin/contracts/utils/Introspection/ERC165.sol
> Compiling @openzeppelin/contracts/utils/math/Math.sol
> Compiling @openzeppelin/contracts/utils/math/SignedMath.sol
> Compiling .\contracts\Contract.sol
> value sent: 0 ETH
> total cost: 0.0264838 ETH

> Saving artifacts
-----
> Total cost: 0.07532676 ETH

Summary
-----
> Total deployments: 2
> Final cost: 0.07532676 ETH
```

The screenshot shows the Remix IDE interface. The left sidebar displays the "DEPLOY & RUN TRANSACTIONS" panel with a list of transactions. The main editor shows the Solidity code for the `ERC721Token` contract. The code includes imports for `pragma solidity`, `@openzeppelin/contracts/token/ERC721/ERC721.sol`, and `@openzeppelin/contracts/token/ERC20/IERC20.sol`. The contract `ERC721Token` is defined with a constructor, a `mint` function, and a `transferERC721` function. The bottom panel shows the transaction details for the `mint` function, including the transaction hash and the gas used.

```
1 pragma solidity ^0.8.9;
2
3 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
4 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
5
6 contract ERC721Token is ERC721 {
7
8     mapping(uint256 => uint256) private _tokenIdToAmount;
9     mapping(uint256 => address) private _tokenIdToERC20;
10
11     constructor(string memory _name, string memory _symbol) ERC721(_name, _symbol) {}
12
13     function mint(address _to, uint256 _tokenId, address _erc20Token) external {
14         require(!_erc20Token != address(0), "wrong ERC20 address");
15         _safeMint(_to, _tokenId);
16         _tokenIdToAmount[_tokenId] = 1000;
17         _tokenIdToERC20[_tokenId] = _erc20Token;
18     }
19
20     function transferERC721(uint256 _tokenId, address _to) external {
21         require(!_exists(_tokenId), "Token does not exist");
22         require(ownerOf(_tokenId) == msg.sender, "for the owner");
23         _transfer(msg.sender, _to, _tokenId);
24     }
25
26     function transferERC20(uint256 _tokenId, address _to) external {
27         require(!_exists(_tokenId), "Token does not exist");
28         require(ownerOf(_tokenId) == msg.sender, "for the owner");
29     }
30 }
31
32
33
34
35
```

Working:

firstly imported libraries in smart contracts and created deployment file then installed all the dependencies , compiled & deployed the same.

The resulting hash generated, block usage and total cost used is showed up.

Now, the functional components of the combined contract fulfilling the requirements:

ERC721Token.sol is an ERC721 token contract that inherits from OpenZeppelin's ERC721 implementation. The contract includes a constructor that takes in the token name and symbol as parameters, and I also implemented a mint function that allows the contract owner to mint new tokens and assign them to a specific address.

combined.sol is a contract that combines the functionality of ERC20Token.sol and ERC721Token.sol. The contract inherits from OpenZeppelin's ERC721 implementation and includes two mappings: `_tokenIdToAmount` and `_tokenIdToERC20`.

This contract also includes two functions for transferring tokens: `transferERC721` and `transferERC20`, which allow the owner of a token to transfer either the ERC721 token or the associated ERC20 tokens to another address.

Also, in future using I'll be using web3.js for interaction from frontend for good user experience on it.

If anything more required from my part, You can mention that as well.

Thankyou.