

[< Back to Blockchain Developer](#)

Decentralized Star Notary

REVIEW

CODE REVIEW 7

HISTORY

Meets Specifications

Congratulations!

Your project has met all specifications! If you have also passed the previous projects, then congratulations too in having completed Term 1! This last part is challenging because of the many new tools and concepts introduced but you managed to make it all work to develop and deploy this star notary smart contract. I hope you also had fun making it and is inspired to learn more. This activity showed you a glimpse of the workflow blockchain developers go through and you can certainly build on the foundational skills you've learned in this course. Feel free to continue interacting with the community as you level up in this field. Without further ado, congratulations again and all the best in your future endeavors!

Write a smart contract with functions

Smart contract implements the ERC-721 or ERC721Token interface

Implements ERC721 with OpenZeppelin! Good idea to use this library of well-tested code vetted by the community!

The star token should have these pieces of metadata added:

- Star coordinators
 - Dec
 - Mag
 - Cent
- Star story

Added required metadata in Star struct!

```
struct Star {  
    string name;  
    string ra;  
    string dec;  
    string mag;  
    string story;  
}
```

Smart contract prevents stars with the same coordinates from being added

First project I've reviewed that used a modifier function! Works well to configure uniqueness! Good job!

Concatenates strings and puts it into a mapping to store all previously created stars:

```
// starsMap for uniqueness checking  
mapping(string => uint256) internal starsMap;
```

```
// concatenation of strings for low gas consumption:  
// ...
```

```

// https://ethereum.stackexchange.com/questions//29/how-to-concaten
ate-strings-in-solidity
// best to do this work off-chain in front-end client
function append3(string _a, string _b, string _c) internal pure ret
urns (string) {
    return string(abi.encodePacked(_a, _b, _c));
}

// this method of uniqueness checking will fail if rules to control
rounding aren't implemented
// eg: if magnitude is 200 and a user inputs 199.99999
function createStarStats(string _ra, string _dec, string _mag) inte
rnal pure returns (string){
    return append3(_ra, _dec, _mag);
}

modifier uniqueStar(string _ra, string _dec, string _mag) {
    string memory starStats = createStarStats(_ra, _dec, _mag);
    require(starsMap[starStats] == 0, "Star already registered");
    _;
}

function createStar(string _name, string _ra, string _dec, string _
mag, string _story, uint256 _tokenId) public uniqueStar(_ra, _dec, _ma
g) {

    Star memory newStar = Star(_name, _ra, _dec, _mag, _story);
    string memory starStats = createStarStats(_ra, _dec, _mag);

    tokenIdToStarInfo[_tokenId] = newStar;
    starsMap[starStats] = _tokenId;

    _mint(msg.sender, _tokenId);
}

function checkIfStarExist(string _ra, string _dec, string _mag) pub
lic view returns (bool) {
    string memory starStats = createStarStats(_ra, _dec, _mag);

    return starsMap[starStats] > 0;
}

```

```
}

```

You can try comparing actual gas consumption with this approach and using a hashing function (for the star coordinates) such as keccak256 to determine uniqueness.

Smart contract implements all these functions - createStar(), putStarUpForSale(), buyStar(), checkIfStarExist(), mint(), approve(), safeTransferFrom(), SetApprovalForAll(), getApproved(), isApprovedForAll(), ownerOf(), starsForSale(), tokenIdToStarInfo()

Expected response for tokenIdToStarInfo():

```
[ "Star power 103!", "I love my wonderful star", "ra_032.155", "dec_121.874", "mag_245.978" ]
```

Implements all functions not found in OpenZeppelin!

Test smart contract code coverage

Project contains tests for the following functions and all tests are approved without error:

createStar(), putStarUpForSale(), buyStar(), checkIfStarExist(), mint(), approve(), safeTransferFrom(), SetApprovalForAll(), getApproved(), isApprovedForAll(), ownerOf(), starsForSale(), tokenIdToStarInfo()

Tests required functions not found in OpenZeppelin!

Contract: StarNotary

can create a star

✓ can create a star and get its name (263ms)

buying and selling stars

✓ user1 can put up their star for sale (78ms)

user2 can buy a star that was put up for sale

✓ user2 is the owner of the star after they buy it (71ms)

✓ user2 ether balance changed correctly (267ms)

star duplication not possible


```
✓ impossible to own same star based on stats (89ms)
verify checkIfStarExist()
✓ star exists return for existing (96ms)
```

6 passing (2s)

Deploy smart contract on a public test network (Rinkeby)

- Smart contract is deployed on on the Ethereum *RINKEBY* test network
- Execute createStar() function
- Place your star for sale using putStarUpForSale() function

Deployed contract on test network and executed required functions!



RINKEBY (CLIQUE) TESTNET

Search by Address / Txhash / Block / Token / Ens

GO

Language

HOMEBLOCKCHAINTOKENMISC

Contract

0xe757c17bdc5a0decC59eb31F3e1Fa29C266387D3

Home / Accounts / Address

Contract Overview

Misc:

More Options

Balance:

0 Ether

Contract Creator:

0x0782140de231a91...at txn 0x1560eb3824e0cfc...

Transactions:

3 txns

Transactions


Code

Events

Latest 3 txns

TxHash	Block	Age	From	To	Value	[TxFee]
0x5bc1dc32e4cfb6...	3228791	13 hrs 12 mins ago	0x0782140de231a91...	IN	0xe757c17bdc5a0de...	0 Ether0.003933336
0x8ab5c81aa5f417d...	3228596	14 hrs 1 min ago	0x0782140de231a91...	IN	0xe757c17bdc5a0de...	0 Ether0.017201096
0x1560eb3824e0cfc...	3228562	14 hrs 9 mins ago	0x0782140de231a91...	IN	Contract Creation	0 Ether0.02586916

[Download CSV Export]



RINKEBY (CLIQUE) TESTNET

Search by Address / Txhash / Block / Token / Ens

GO

Language

HOMEBLOCKCHAINTOKENMISC



Etherscan
The Ethereum Block Explorer

Transaction 0x5bc1dcb32e4cfb65a7002bb440ca98c1e5733c0638b8b498cf55c77e4208f161

Home / Transactions / Tx Info

Overview

Transaction Information

Tools & Utilities

[This is a Rinkeby Testnet Transaction Only]

TxHash:

0x5bc1dcb32e4cfb65a7002bb440ca98c1e5733c0638b8b498cf55c77e4208f161

TxReceipt Status:

Success

Block Height:

3228791 (3172 Block Confirmations)

TimeStamp:

13 hrs 13 mins ago (Oct-26-2018 01:25:06 PM +UTC)

From:

0x0782140de231a91c2cef8c830107d985043cb2cc

To:

Contract 0xe757c17bdc5a0decc59eb31f3e1fa29c266387d3

Value:

0 Ether (\$0.00)

Gas Limit:

300000

Gas Used By Transaction:

44697

Gas Price:

0.000000088 Ether (88 Gwei)

Actual Tx Cost/Fee:

0.003933396 Ether (\$0.000000)

Nonce & (Position):

7 | {1}

Input Data:

Function: putStarUpForSale(uint256_tokenId, uint256_price) ***

MethodID: 0x316a4361

[0]: 0001

[1]: 0016

Project submission includes a document (.md, .txt) that includes:

- Transaction ID
- Contract address

Hint: You can view Transaction ID and Contract ID from a blockchain explorer (e.g. Etherscan). Example Contract ID:
<https://rinkeby.etherscan.io/address/0xf0720c0715e68f80c0c0437c9c491abfed9e7ab#code>

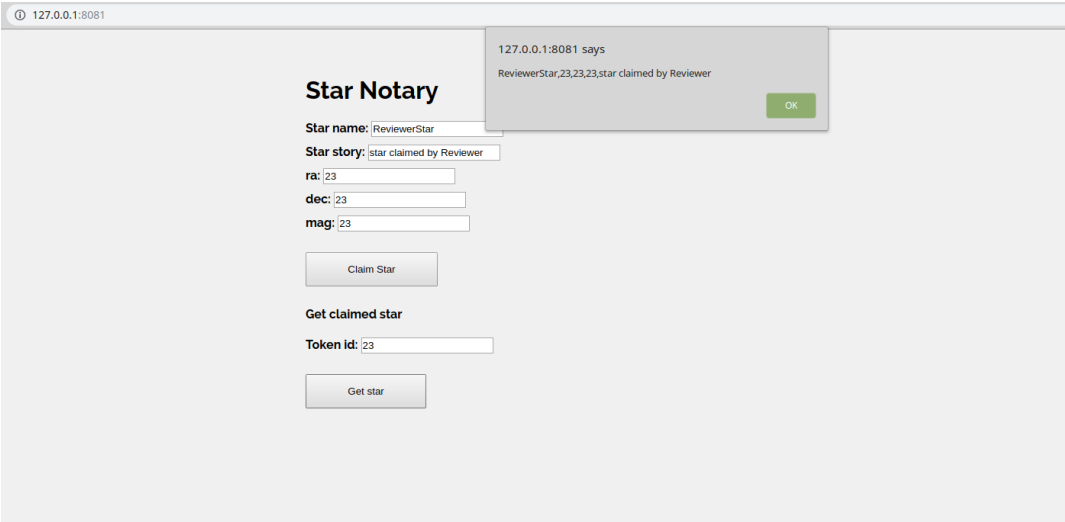
Modify client code to interact with a smart contract

Front-end is configured to:

- Claim a new star. Each new star support these pieces of metadata:
 - Star coordinators
 - Dec
 - Mag
 - Cont

- Star story
- Lookup a star by ID using tokenIdToStarInfo()

Now works! Able to create a star and look it up via the token ID!



 [DOWNLOAD PROJECT](#)

7 [CODE REVIEW COMMENTS](#)



RETURN TO PATH

Rate this review