PROJECT SPECIFICATION

# Decentralized Star Notary

## Write a smart contract with functions

| CRITERIA | MEETS SPECIFICATIONS |
|---|---|
| Define and implement interface | Smart contract implements the ERC-721 or ERC721Token interface |
| Add metadata to the star token | The star token should have these pieces of metadata added:<br><br>• Star coordinators<br><ul><li>Dec</li><li>Mag</li><li>Cent</li></ul><br>• Star story |
| Configure uniqueness with the stars | Smart contract prevents stars with the same coordinates from being added |
| Smart contract contains required functions | Smart contract implements all these functions - createStar(), putStarUpForSale(), buyStar(), checkIfStarExist(), mint(), approve(), safeTransferFrom(), SetApprovalForAll(), getApproved(), isApprovedForAll(), ownerOf(), starsForSale(), tokenIdToStarInfo()<br><br>Expected response for tokenIdToStarInfo():<br><br>`["Star power 103!", "I love my wonderful star", "ra_032.155", "dec_121.874", "mag_245.978"]` |

## Test smart contract code coverage

| CRITERIA | MEETS SPECIFICATIONS |
|---|---|
| Properly test all required functions | Project contains tests for the following functions and all tests are approved without error: |

| CRITERIA | MEETS SPECIFICATIONS |
|---|---|
| | createStar(), putStarUpForSale(), buyStar(), checkIfStarExist(), mint(), approve(), safeTransferFrom(), SetApprovalForAll(), getApproved(), isApprovedForAll(), ownerOf(), starsForSale(), tokenIdToStarInfo() |

## Deploy smart contract on a public test network (Rinkeby)

| CRITERIA | MEETS SPECIFICATIONS |
|---|---|
| Deploy smart contract on a public test network | • Smart contract is deployed on on the Ethereum *RINKEBY* test network<br>• Execute createStar() function<br>• Place your star for sale using putStarUpForSale() function |
| Project submission includes transaction ID and contract address | Project submission includes a document (.md, .txt) that includes:<br><br>• Transaction ID<br>• Contract address<br><br>**Hint:** You can view Transaction ID and Contract ID from a blockchain explorer (e.g. Etherscan). Example Contract ID:<br>https://rinkeby.etherscan.io/address/0xfb0720c0715e68f80c0c0437c9c491abfed9e7ab#code |

## Modify client code to interact with a smart contract

| CRITERIA | MEETS SPECIFICATIONS |
|---|---|
| | |
| Client code interacts with smart contract | Front-end is configured to: |

| CRITERIA | MEETS SPECIFICATIONS |
|---|---|
| | • Claim a new star. Each new star support these pieces of metadata:<br><br>    ○ Star coordinators<br><br>        ■ Dec<br>        ■ Mag<br>        ■ Cent<br><br>    ○ Star story<br><br>• Lookup a star by ID using tokenIdToStarInfo() |

**Suggestions to Make Your Project Stand Out!**

## Optional: Configure RESTful API endpoint to return a registered star

**NOTE:** This section is not required for your project to pass. These steps are purely optional and a way for you to expand on your project.

- Utilizing your previous coursework expereince in course three, create a new web API using Node.js and a RESTful framework. Include a library that will connect with your smart contract using a web3 instance.
- Configure an endpoint function that will use the web3 instance to communicate with your smart contract and return a star by the star token ID. The endpoint must utilize a GET method with URL path for star token ID.

**Route path example:**

```
method: 'GET',
path: '/star/{starTokenId}'
```

- Endpoint returns a json object

**JSON response example:**

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