

# Skeleton and JavaScript Architecture

In this article, you'll learn about the basic architecture behind the NFT contract that you'll develop while following this "Zero to Hero" series. You'll discover the contract's layout and you'll see how the JavaScript files are structured in order to build a feature-complete smart contract. **caution** The JS-SDK is currently in [Alpha](#).

## Introduction

This tutorial presents the code skeleton for the NFT smart contract and its file structure. You'll find how all the functions are laid out as well as the missing JS code that needs to be filled in. Once every file and function has been covered, you'll go through the process of building the mock-up contract to confirm that everything is working correctly.

## File structure

Following a regular [JavaScript](#) project, the file structure for this smart contract has:

- package.json
- file to define the packages and scripts used in the project.
- src
- folder where all the JavaScript source files are stored
- build
- folder where the compiled wasm
- will output to.

## Source files

File Description [approval.ts](#) Has the internal functions that controls the access and transfers of non-fungible tokens. [enumeration.ts](#) Contains the internal methods to query for NFT tokens and their owners [index.ts](#) Holds the exposed smart contract functions. [metadata.ts](#) Defines the token and metadata structures. [mint.ts](#) Contains the internal token minting logic. [nft\\_core.ts](#) Has the internal core logic that allows you to transfer NFTs between users [royalty.ts](#) Contains the internal payout-related functions. nft-tutorial-js └── src market-contract nft-contract ├── approval.ts ├── enumeration.ts ├── index.ts ├── metadata.ts ├── mint.ts ├── nft\_core.ts └── royalty.ts tip Explore the code in our [GitHub repository](#).

## approval.ts

This allows people to approve other accounts to transfer NFTs on their behalf. This file contains the internal logic that complies with the standard's [approvals management](#) extension. Here is a breakdown of the methods and their functions:

Method Description internalNftApprove Approves an account ID to transfer a token on your behalf. Called during nft\_approve . internalNftIsApproved Checks if the input account has access to approve the token ID. Called during nft\_is\_approved . internalNftRevoke Revokes a specific account from transferring the token on your behalf. Called during nft\_revoke . internalNftRevokeAll Revokes all accounts from transferring the token on your behalf. Called during nft\_revoke\_all . src/nft-contract/approval.ts loading ... [See full example on GitHub](#) You'll learn more about these functions in the [approvals section](#) of the Zero to Hero series.

## enumeration.ts

This file provides the internal functions needed to view information about NFTs, and follows the standard's [enumeration](#) extension. Method Description internalNftTotalSupply Returns the total amount of NFTs stored on the contract. Called during nft\_total\_supply . internalNftTokens Returns a paginated list of NFTs stored on the contract regardless of their owner. Called during nft\_tokens . internalNftSupplyForOwner Allows you view the total number of NFTs owned by any given user. Called during nft\_supply\_for\_owner . internalNftTokensForOwner Returns a paginated list of NFTs owned by any given user. Called during nft\_tokens\_for\_owner . src/nft-contract/enumeration.ts loading ... [See full example on GitHub](#) You'll learn more about these functions in the [enumeration section](#) of the tutorial series.

## metadata.ts

This file is used to keep track of the information to be stored for tokens, and metadata. In addition, you can define a function

to view the contract's metadata which is part of the standard's [metadata](#) extension. Name Description TokenMetadata This structure defines the metadata that can be stored for each token. (title, description, media, etc. Token This structure outlines what information will be stored on the contract for each token. JsonToken When querying information about NFTs through view calls, the return information is stored in this JSON token. internalNftMetadata This function allows users to query for the contract's internal metadata. Called during `nft_metadata` . `src/nft-contract/metadata.ts` loading ... [See full example on GitHub](#) You'll learn more about these functions in the [minting section](#) of the tutorial series.

## mint.ts

Contains the internal token minting logic. Method Description internalNftMint This function mints a non-fungible token. Called during `nft_mint` . `src/nft-contract/mint.ts` loading ... [See full example on GitHub](#)

## nft\_core.ts

Core logic that allows you to transfer NFTs between users. Method Description internalNftTransfer Transfers an NFT to a receiver ID. Called during `nft_transfer` . internalNftTransferCall Transfers an NFT to a receiver and calls a function on the receiver ID's contract. The function returns true if the token was transferred from the sender's account. Called during `nft_transfer_call` . internalNftToken Allows users to query for the information about a specific NFT. Called during `nft_token` . internalNftResolveTransfer When you start then `nft_transfer_call` and transfer an NFT, the standard dictates that you should also call a method on the receiver's contract. If the receiver needs you to return the NFT to the sender (as per the return value of then `nft_on_transfer` method), this function allows you to execute that logic. Called during `nft_resolve_transfer` . `src/nft-contract/nft_core.ts` loading ... [See full example on GitHub](#) You'll learn more about these functions in the [minting section](#) of the tutorial series.

## royalty.ts

Contains the internal payout-related functions. Method Description internalNftPayout This internal method calculates the payout for a given token. Called during `nft_payout` . internalNftTransferPayout Internal method to transfer the token to the receiver ID and return the payout object that should be paid for a given balance. Called during `nft_transfer_payout` . `src/nft-contract/royalty.ts` loading ... [See full example on GitHub](#) You'll learn more about these functions in the [royalty section](#) of the tutorial series.

## index.ts

This file outlines the smart contract class and what information it stores and keeps track of. In addition, it exposes all public facing methods that are callable by the user. Method Description init Constructor function used to initialize the contract with some metadata and default state. nft\_mint Calls the internal mint function to mint an NFT. nft\_token Calls the internal function to query for info on a specific NFT nft\_transfer Calls the internal function to transfer an NFT nft\_transfer\_call Calls the internal function to transfer an NFT and call `nft_on_transfer` on the receiver's contract nft\_resolve\_transfer Calls the internal function to resolve the transfer call promise. nft\_is\_approved Calls the internal function to check whether someone is approved for an NFT nft\_approve Calls the internal function to approve someone to transfer your NFT nft\_payout Calls the internal function to query for the payout object for an NFT nft\_transfer\_payout Calls the internal function to transfer an NFT and return the payout object. nft\_revoke Calls the internal function to revoke someone access to transfer your NFT nft\_revoke\_all Calls the internal function to revoke everyone's access to transfer your NFT nft\_total\_supply Calls the internal function to query the total supply of NFTs on the contract. nft\_tokens Calls the internal function to paginate through NFTs on the contract nft\_tokens\_for\_owner Calls the internal function to paginate through NFTs for a given owner nft\_supply\_for\_owner Calls the internal function to query for the total number of NFTs owned by someone. nft\_metadata Calls the internal function to query for the contract's metadata `src/nft-contract/index.ts` loading ... [See full example on GitHub](#) You'll learn more about these functions in the [minting section](#) of the tutorial series.

## Building the skeleton

- If you haven't cloned the main repository yet, open a terminal and run:

```
git clone https://github.com/near-examples/nft-tutorial-js/ * Next, switch to the 1.skeleton * branch. * Install the dependencies (including the JS SDK): yarn * Build the contract with yarn build *
```

```
git clone https://github.com/near-examples/nft-tutorial-js/ cd nft-tutorial-js git checkout 1.skeleton yarn && yarn build Once this finishes, thenft-tutorial-js/build directory should contain thenft.wasm smart contract!
```

Building the skeleton is useful to validate that everything works properly and that you'll be able to compile improved versions of this NFT contract in the upcoming tutorials.

## Conclusion

You've seen the layout of this NFT smart contract, and how all the functions are laid out across the different source files. Using yarn, you've been able to compile the contract, and you'll start fleshing out this skeleton in the next [Minting tutorial](#).

Versioning for this article At the time of this writing, this example works with the following versions:

- near-sdk-js:0.4.0-5
- NFT standard: [NEP171](#)
- , version 1.0.0 [Edit this page](#) Last updated on Jan 19, 2024 by [Damián Parrino](#) Was this page helpful? Yes No

[Previous Pre-deployed Contract](#) [Next Minting](#)