Documentation for Solana Blockchain in Avy

# Wallet (Keypair)

A **wallet** on Solana represents an account controlled by a keypair (public and private keys).

* The private key is used to sign transactions and prove ownership.
* In this context, wallets are used to:
  + **Sign transactions** when creating collections or NFTs
  + **Act as authorities** (e.g., collection authority, mint authority)
* Wallets are typically loaded from .env secrets during server-side NFT creation.

# Metaplex

**Metaplex** is an open-source protocol and suite of tools that powers NFTs on the Solana blockchain.

* It defines the **Token Metadata Standard**, used by almost all Solana NFTs.
* Metaplex ensures consistency across wallets, marketplaces, and creators by standardizing how NFTs are represented.

# Umi (Universal Metaplex Interface)

**Umi** is a lightweight and modular framework developed by **Metaplex** for building on Solana.

* It simplifies common tasks like uploading files, creating NFTs, managing metadata, and sending transactions.
* Built with modern JavaScript/TypeScript patterns, it offers a plugin-based approach (e.g., mplTokenMetadata, irysUploader).
* Think of Umi as a developer-friendly wrapper around Solana programs that handles serialization, signing, and networking efficiently.

# Umi Initialization and Wallet Setup:

* **createUmi(...)** -Initializes the Umi SDK with a specified Solana RPC endpoint. It provides the core interface for interacting with the Solana blockchain.
* **.use(mplTokenMetadata())** - Loads the Metaplex Token Metadata plugin, enabling operations like creating NFTs, collections, and metadata updates according to the Metaplex standard.
* **use(irysUploader({...}))** - Adds the Irys (formerly Bundlr) uploader plugin, which allows assets (e.g., images and metadata) to be uploaded to Arweave in a decentralized and permanent way.

# Wallet and Identity Setup:

* **WALLET\_SECRET\_KEY** - The admin wallet's secret key, loaded from an environment variable. Used for signing and sending transactions like minting NFTs or managing collections.
* **createKeypairFromSecretKey(...)** - Converts the raw secret key into a keypair compatible with Umi/EdDSA, enabling signing capabilities.
* **createSignerFromKeypair(...)** - Wraps the keypair as a Signer object, which Umi uses as a transaction authority.
* **umi.use(signerIdentity(...))** - Sets the current wallet identity for the Umi client. All transactions and interactions will be signed by this wallet.

# Collection Authority Setup:

* **collectionAuthoritySecretKey** - Parses the secret key from the WALLET\_SECRET\_KEY environment variable. This key represents the wallet that will act as the collection authority — the entity allowed to create and manage NFT collections.

# Collections:

* In the Solana blockchain ecosystem, **collections** refer to a grouping mechanism primarily used with **NFTs (Non-Fungible Tokens)**. A **collection** is a way to associate multiple NFTs under a single unified identifier, making it easier to organize, discover, and verify related assets.
* Each collection is represented by a **"collection NFT"**, which has its own metadata, image, and attributes — just like a regular NFT. Other NFTs can then reference this collection NFT to signify they belong to it.

# /api/create-collection:

1. **Request Body Fields**

* **Name -** The display name of the NFT. For example, "My NFT Collection"
* **Symbo**l -A short ticker-like identifier, like "MYCOL" or "ART"
* **Description -**A human-readable description of what this NFT represents (shown in wallets & marketplaces)
* **Image -** A URL (usually on Arweave or IPFS) that points to the main image of the NFT
* **External url -** A link to an external site – like your project page, collection website, or documentation
* **Attributes -** Describes traits or properties of the NFT

1. **Functions used**
   * **createGenericFile(fileContent, 'my-image.jpg', { ... }) -** Wraps the raw image buffer into a Umi-compatible file object.
   * **umi.uploader.upload([file]) -** Uploads the image file to a decentralized storage provider like Irys. Returns a image Uri
   * **umi.uploader.uploadJson(metadata)** - Uploads the NFT metadata as a JSON file to decentralized storage. Returns a uri that points to the uploaded JSON. A metadata object defines the collection’s name, symbol, description, image, external URL, attributes, and more.
   * **findMetadataPda(umi, { mint })** - Computes the Program Derived Address (PDA) for the metadata account. PDAs are used in Solana to deterministically find metadata accounts associated with a given mint address.
   * **createNft(umi, { ... }).sendAndConfirm(...)** - Sends the actual instruction to create the NFT on-chain using the provided mint and metadata URI. This CreateNft function can be used to create collection by setting **isCollection** flag to be **true.**

const tx = await createNft(umi, {

        mint: collection,

        name,

        uri: metadataUri,

        sellerFeeBasisPoints: percentAmount(0),

        isCollection: true,

      }).sendAndConfirm(umi, { send: { commitment: "finalized" } });

# /api/edit-collection:

* **new PublicKey(collectionAddress**) - Converts the string into a valid Solana PublicKey instance used by Umi.
* **fetchMetadataFromSeeds(umi, { mint: collectionPublicKey })** - Fetches current metadata (including the **update authority**) using Metaplex's PDA-based fetching system.

 const tx = await updateV1(umi, {

      mint: collectionPublicKey,

      authority: initialMetadata.updateAuthority.toString(),

      data: { ...initialMetadata, name: 'Updated Asset' },

      isMutable: true,

      // ...

    }).sendAndConfirm(umi)

* **updateV1()** - is used to modify the NFT’s metadata.
* **Mint** - the address of the collection NFT.
* **Authority** - must be the current update authority of the NFT.
* **Data** - new metadata (you can spread in existing values and override only what’s changed).
* **isMutable : true -**  ensures the NFT ( In this case NFT flagged as collection)remains editable.

# /api/create-asset:

* The process is almost similar to our collection creation api, as we are using createNft function to create a NFT and mark it as a collection.
* **Request Body Fields:**
  + **holderEmail** - The email of the person receiving the NFT. Used as a trait in the metadata to identify ownership.
  + **Level** - Indicates the course level (e.g., Beginner, Intermediate, Advanced). Added to metadata as a trait.
  + **courseCode** - A unique identifier for the course (e.g., "CS101"). Useful for referencing. Stored in metadata.
  + **nftImage** - image that will be uploaded and used as the NFT's visual representation (e.g., a certificate or badge).
  + **Holder** - Name of the individual who completed the course. Appears in metadata as a trait.
  + **collectionMintAddress** - The mint address of the existing collection NFT that this new NFT will belong to. Enables grouping and verification.
  + **Category** - Describes category of the course (IT, Marketing, etc). Appears in metadata as a trait.
  + **titleCourse** - The title of the course (e.g., "Intro to Blockchain"). This becomes the NFT name and is shown in explorers.
  + **Description** - A short description about the course or what this NFT represents. Goes into the metadata.
* Most of the NFT creation process is similar to Collection creation.

const tx = await createNft(umi, {

        mint: asset,

        name: metadata.name,

        symbol: "ByAni",

        uri: metadataUri,

        sellerFeeBasisPoints: percentAmount(0), // for 0% royalties

        updateAuthority: umi.identity.publicKey,

        isMutable:false,

        collection: {

            key: publicKey(collectionMintAddress),

            verified: false

        },

    }).sendAndConfirm(umi, { send: { commitment: "finalized" } });

* There is no **IsCollection** flag, as it is a NFT.
* **isMutable: false -** Once minted, the NFT is **frozen**, and fields like name, image, attributes, etc., **cannot be changed**. This is ideal for **certificates, diplomas, or badges**, where the content should never be modified after issuance.
* **collection: { key, verified }** - This block links the NFT to a **parent collection NFT**.
* **verifyCollectionF(umi, asset.publicKey, collectionMintAddress)** - This sends a transaction to verify the collection association of the NFT. Until this happens, the NFT only has an unverified link to the collection.
* **Once verified:** Marketplaces and wallets (like Phantom, Magic Eden) will show the NFT as part of that collection.

# /api/transfer-nft:

* Transfers ownership of a specific NFT (assetAddress) to another wallet (destinationWallet).
* **Request Body Fields:**
  + **assetAddress** - The mint address (public key) of the NFT you want to transfer. This uniquely identifies the digital asset on Solana.
  + **destinationWallet** - The public key of the wallet to which you want to transfer the NFT. This is the recipient’s address.
* **fetchDigitalAsset(umi, publicKey(assetAddress))** - Grabs full metadata for the NFT (also called a mint) from the blockchain using its public key.

const tokenAccount = findAssociatedTokenPda(umi, {

      mint: publicKey(assetAddress),

      owner: umi.identity.publicKey, // Assuming the current wallet is the owner

    });

* **findAssociatedTokenPda** - Finds the **Associated Token Account (ATA)** that currently holds this NFT — basically a Solana SPL token account tied to your wallet and the NFT mint
* **getAccount(umi.rpc.getConnection(), tokenAccount)** - Pulls info about the token account — like who owns it, how many tokens are there (should be 1 for NFTs), etc.

const tokenAccount = findAssociatedTokenPda(umi, {

      mint: publicKey(assetAddress),

      owner: umi.identity.publicKey, // Assuming the current wallet is the owner

    });

* **transferV1** – **transfer instruction for NFT**
  + **source**: token account holding the NFT.
  + **destination**: recipient's associated token account for this NFT.
  + **amount**: 1 (because NFTs are non-divisible).
  + **authority**: current wallet that owns the NFT.
* **transferIx.sendAndConfirm - broadcasts the transaction** and waits until it’s finalized on the blockchain

# /api/all-collections:

* This api is used to fetch all collections that are available inside our wallet.
* **if (fullMetadata.isCollection === true) -**
  + Since we created our collection using CreateNft and flagging it as a collection )we will need this condition to find collections
* **createUmi("https://devnet.helius-rpc.com/?api-key=...")** –
  + A **new Umi instance** is created specifically for this request, using **Helius RPC** (which supports DAS - Digital Asset Standard).
  + Helius RPC is a Solana RPC provider that supercharges your app with fast queries, enriched metadata, and developer-friendly tools.
* **dasApi()** –
  + Adds support for **Digital Asset Standard (DAS)** from Helius.
  + Exposes methods like getAssetsByOwner()
* **umi.rpc.getAssetsByOwner({ owner })** –
  + Queries all **NFTs or tokens owned** by a wallet address.
  + Returns metadata such as asset ID, content URI, and creator info.

# /api/collection-data:

* Used to fetch all the assets ( e.g. NFTs ) present in a particular collection.
* **fetchDigitalAsset(umi, collectionMintAddress) –**
  + Uses Umi’s fetchDigitalAsset method to get **metadata about the collection** NFT.
* **createUmi("https://devnet.helius-rpc.com/?api-key=...") –** 
  + A **new Umi instance** is created specifically for this request, using **Helius RPC** (which supports DAS - Digital Asset Standard).
  + Helius RPC is a Solana RPC provider that supercharges your app with fast queries, enriched metadata, and developer-friendly tools.
* **dasApi()** –
  + Adds support for **Digital Asset Standard (DAS)** from Helius.
  + This is required because **getAssetsByGroup** is part of the **DAS API** which needs the plugin added.
* **umi2.rpc.getAssetsByGroup({groupKey: "collection", groupcollectionMintAddress}) –** 
  + Uses **Helius DAS API** to fetch **all NFTs** that belong to the collection mint address.
  + **groupKey: "collection**" tells Helius you're filtering NFTs by collection grouping.
  + groupValue is the **mint address of the collection**

# Explaining basic Solana Blockchain Terms:

# ****1. SOL (Solana Token)****

* **Native currency** of the Solana blockchain, used for transactions and fees.
* Used to **pay for smart contract execution** and other network activities.
* Can be **staked** to support network security and earn rewards.

# ****2. Wallet****

* A **digital tool** to store, send, and receive SOL and Solana-based tokens/NFTs.
* Examples include **Phantom, Solflare, and Backpack** wallets.
* Wallets can be **browser extensions, mobile apps, or hardware devices**.

# ****3. Collection****

* A group of NFTs **linked by a shared theme, creator, or project**.
* Collections help organize NFTs, making them easier to browse or trade.
* Often used in **profile picture (PFP) projects, gaming assets, or art series**.

# ****4. NFT (Non-Fungible Token)****

* A **unique digital asset** on the blockchain (can represent art, music, game items, etc.).
* Unlike cryptocurrencies, each NFT is **distinct and not interchangeable**.
* Stored on-chain with **metadata** like images, traits, and creator details.

# Avy Apps Blockchain integration Overview:

* **Blockchain-Powered Certificates**
  + Every course completion certificate is issued as a **unique NFT** on the blockchain.
  + This makes certificates **tamper-proof, verifiable, and impossible to forge**.
  + Anyone can verify a certificate’s authenticity through its public NFT record.
* **Course-to-Collection Mapping**
  + When a **new course is created**, a corresponding **NFT collection** is automatically created on the blockchain.
  + This collection serves as a **grouping of all certificates** (NFTs) issued for that course.
  + It helps in organizing and validating certificates issued for the same curriculum.
* **Certificate Issuance as NFTs**
  + Once a user **completes a course**, an **NFT certificate** is minted in their name.
  + This NFT is **part of the course's collection**, linking it to the specific course.
  + The NFT includes **metadata like user info, course name, completion date, etc.**