Build a Polkadot NFT shop with Kodadot

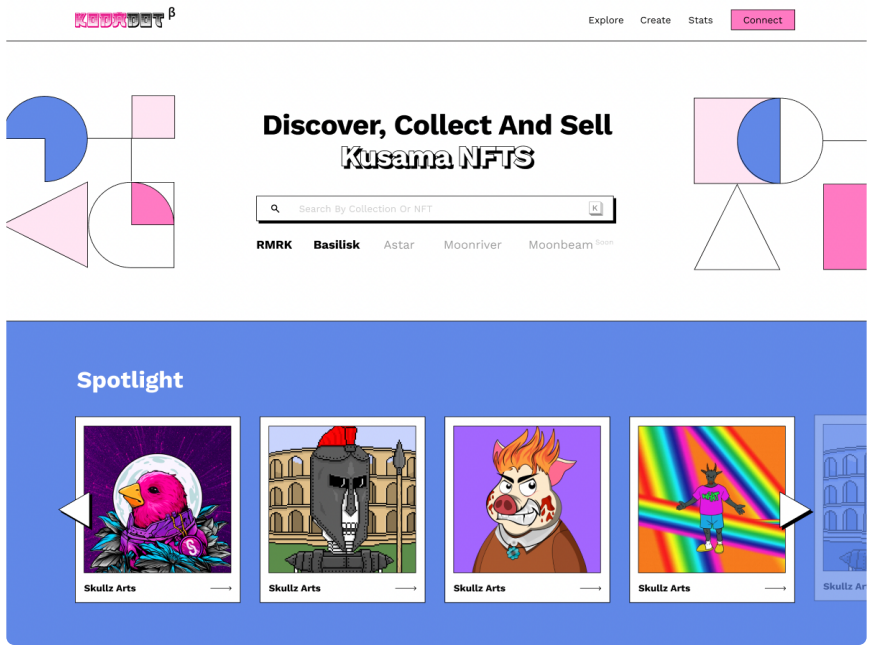
using Uniquery, Deno and Fresh

Hey, I am Viki 👋

- co-founder of KodaDot
- technical wizard
- bleeding edge implementations in KodaDot
- Github / Twitter: @vikiival
- Discord: vikiival|KodaDot#0001
- Web: vik.ink

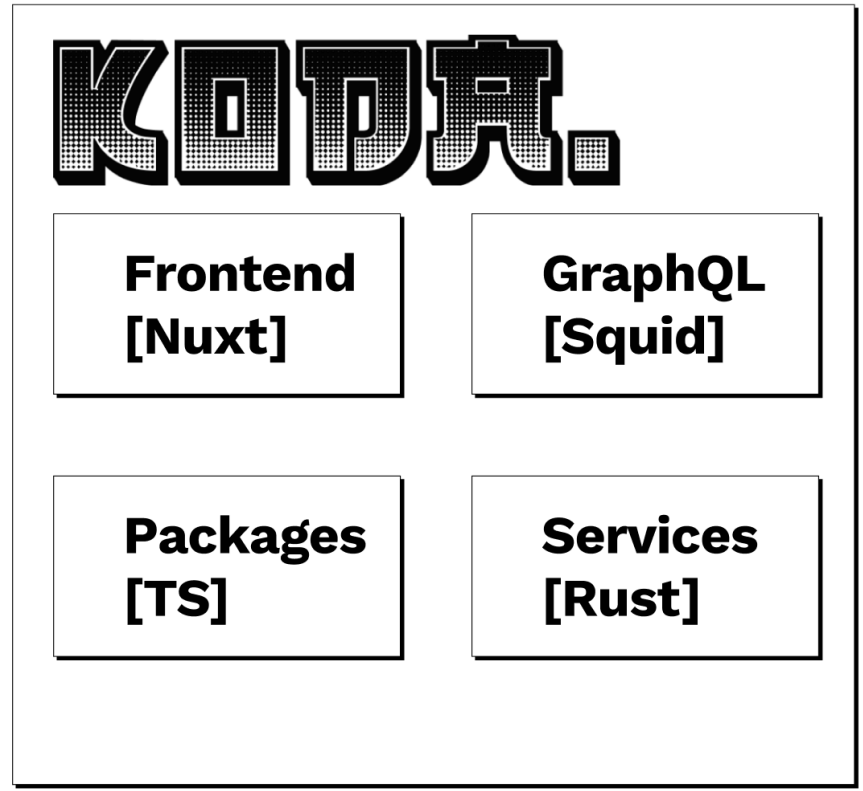
What is KodaDot?

- Open-source NFT marketplace in the Polkadot ecosystem
- Multichain powered - Kusama, Basilisk, Moonbeam, Moonriver, Astar
- Technology agnostic - RMRK strings, runtime pallets, EVM, ink!
- Bounty based - bringing dev talent to the ecosystem
- **430+ stars, ~250 forks, 80+ contributors**



Architecture of KodaDot

- Frontend - Nuxt3, Vue composition API & Brick
- Services - Rust-based workers - pinning, images
- Backend - FireSquid Indexers by SubSquid, GraphQL API
- Packages
  - Minting
  - API
  - SubSquid utils
  - Conditional rendering



Why do we need indexers?

- **Do not sacrifice UX** - Reading data from the blockchain takes a lot of time and resources.
- **Analytics** - making data views is a great way to get insights into the data.
- **You can’t process RMRK** - RMRK is basically string in the blockchain, so without indexers you can’t process it.
- **Simple GraphQL interface** - a great way to get data from the blockchain.

Indexers 101

What does this thing do?

- **Two types of data in each block** - Events and extrinsics
  - **Extrinsics** - The call that each user makes to the chain (`system.remark`).
  - **Events** - Interactions that are emmited by the chain (`balances.Transfer`).
- **Rule of thumb** - It is more effective to process *events*.

How does the processing works?

1. For each block, extracts all requested extrinsics/events.
2. For each event/extrinsic, call the pre-defined handler.
3. Handler should transform the data to match the schema we defined.
4. Save into the Postgres database.

How does the schema look like?

```
type CollectionEntity @entity {
  version: String
  name: String
  max: Int!
  issuer: String
  symbol: String
  id: ID!
  metadata: String
  currentOwner: String
  nfts: [NFTEntity!] @derivedFrom(field: "collection")
  events: [CollectionEvent!]
  blockNumber: BigInt
  meta: MetadataEntity
  createdAt: DateTime!
}
```

```
type NFTEntity @entity {
  name: String
  instance: String
  transferable: Int
  collection: CollectionEntity!
  issuer: String
  sn: String
  id: ID!
  hash: String! @index
  metadata: String
  currentOwner: String
  price: BigInt!
  burned: Boolean!
  blockNumber: BigInt
  events: [Event!] @derivedFrom(field: "nft")
  emotes: [Emote!] @derivedFrom(field: "nft")
  meta: MetadataEntity
  createdAt: DateTime!
  updatedAt: DateTime!
}
```

Let’s query some data

SubSquid Basilisk

Task (try it yourself)

Using **SubSquid** fetch the first first `2` collections that has some not-burned and fetch `2` non-burned NFTs

- For collection I want: `id`, `name`
- For NFTs get: `name`

Let’s query some data

Task (how I would write it)

Using **SubSquid** fetch the first first `2` collections that has some not-burned and fetch `2` non-burned NFTs

- For collection I want: `id`, `name`
- For NFTs get: `name`

```
query collectionWithSomeNFTs  {
  collectionEntities(limit:2, where: {nfts_some: {burned_eq: false}}) {
    id
    name
    nfts(limit:2, where: { burned_eq: false }) {
      name
    }
  }
}
```

Finally, we have an API 🎉

- Let's meet Uniquery
- Query builder on top of SubSquid
- two implementations
  - Builder
  - REST
- No middleware server is needed
- Easily extendable
- Easy to use
- **20+ calls supported**

KoDragons#1

creator:
bXmx9pNXZw6dYkwaoafJyTrAZ…

Why is Uniquery cool?

before

```
query nftListByCollectionId {
  nft: nftEntities(where: {
    collection: {id_eq: "2305670031"}}
) {
    id
    metadata
    currentOwner
    issuer
  }
}
```

after

```
import { getClient, ask } from '@kodadot1/uniquery'

// using builder
const client = getClient()
const id = '2305670031'
const query = client.itemListByCollectionId(id)

// using REST
const path = `/nftListByCollectionId/${id}`
const result = await ask(path)
```

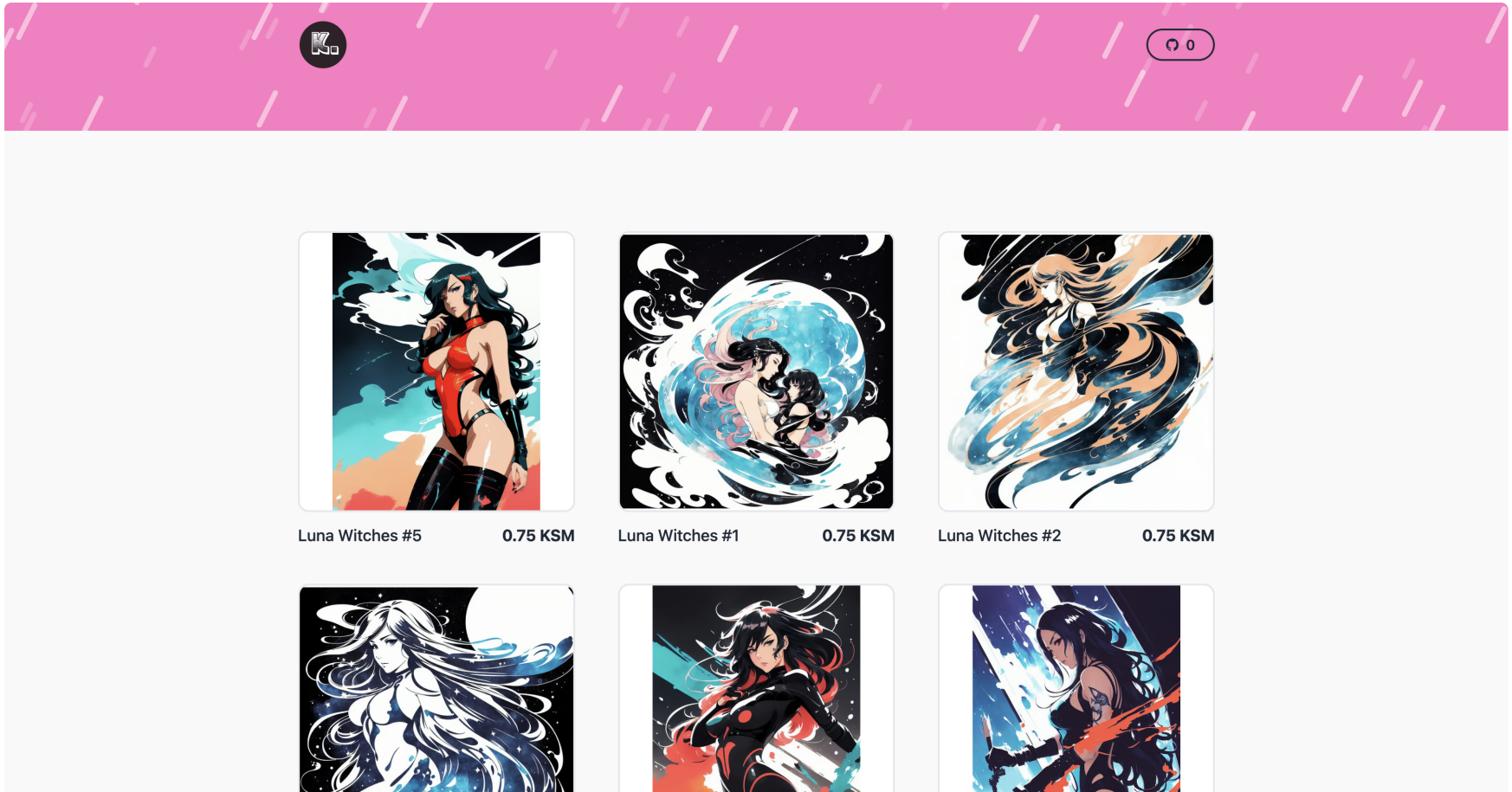
Why Deno & Fresh?

- JavaScript runtime (something like NodeJS)
- Written in Rust (We love Rust)
- Super fast and super safe
- Works with TypeScript out of the box
- Built-in tooling (no need for NPM)
- Works on serverless platforms (Deno Deploy)

What is Fresh?

- WebFramework for Deno (like NextJS or Nuxt)
- Uses Preact (smaller than React)
- Zero JS shipped until you need it

What we are going to build today



So leeet’s get started!

https://shop.kodadot.xyz

First 20, Use: SUBSTRATE60

Links and resources

- Demo shop (webapp): https://fandom.deno.dev/
- Demo shop (repo): https://github.com/vikiival/shop/
- Kodadot (GitHub org): https://github.com/kodadot
- Kodadot (twitter): https://twitter.com/kodadot
- Kodadot: https://kodadot.xyz/
- Uniquery API (repo): https://github.com/kodadot/uniquery

What are your questions?

and do not forget to give us a star