DKAN Starter Documentation

Release 0.0.1

DKAN Starter

Aug 16, 2018

Contents

1 Table of Contents 3

Note: These docs are a work in progress.

CHAPTER 1

Table of Contents

1.1 Introduction

For the pros and const of this library see the ReadMe.

The following is an overview of concepts and use.

1.1.1 Metadata

The purpose of this project is to make it as easy as possible to catalog metadata. The primary use-case is "open data" but it can be adapted to any schema that takes a similar shape. Files can also be stored and published along side the metadata. Services such as Carto can be used to provide an API for interacting with files or what open data catalog's refer to as a DataStore.

A document store offers a good solution for storing metadata because it limits the service area for reciving (JSON Objects), storing end editing (JSON Objects with references), and publishing (JSON Objects) metadata.

By limiting the scope and functionality of a metadata catalog this project is designed to make it easier to interact with outside services. For example the integration with ElasticSearch consists of only two methods.

1.1.2 Collections and Docs

Content in this catalog is divided into `collections` and `documents` similar to MongoDB or other Document Stores. `Collections` are types of content such as a "dataset" or "organization" though they can be anything defined by a schema. `Docs` are the individual content items.

The content model contains a `FileStorage` and `MongoDB` sub-classes which are options for storage. The `FileStorage` class treats the local file system as a document databse storing and retrieving results from disk. Using files to store metadata is a primary advantage of this project however a Mongo option is offered since it is necessary for Interra Catalog Admin and the methods for interacting with the data (ie InsertOne) are identical. Note the Mongo methods are not fully supported yet.

1.1.3 Structure

This project consists of the following:

```
config.yml
models/
schemas/
build/
sites/
app/
cli.js
plopfile.js
```

The rest of the files and folders are artifacts of react boilerplate which drives the `app/`.

config.yml

Contains variables for the location of the `sites/`, `build/`, `schemas/` directories and the storage mechanism. Storage options are the default FileStorage. Mongo is not fully supported yet.

models/

Contains classes for creating, storing and building catalogs.

schemas/

Contains the base schemas for the catalog. For adding new schemas it is recommended to change the schema directory in the base `config.yml` file.

build/

Contains the fully-built sites separated by site name. Each site consists of an export of the collection data of the site as well as the built version of the app. The site folder is the production version of the site.

sites/

Contains the data for each site separated by site name. Each site contains a `config.yml` file, media files, collection data, and harvest sources.

app/

Contains the react app that builds renders the catalog.

cli.js

A cli for performing site tasks built using Caporal. Run `node cli.js` to see a list of available commands:

```
validate-site-contents site
validate-site site
build-collection-data site
build-collection-data-item site collection interraId
build-config site
build-datajson site
build-routes site
build-schema site
build-search site
build-swagger site
build-apis site
build-site site
run-dev site
run-dev-dll
run-dev-tunnel site
harvest-cache site
harvest-run site
load-doc site collection interraId
```

1.1.4 React App Front-End

The front end app is built using react boilerplate. This will likely be separated from the models at some point. Data for each site is exported to the `build/` folder. Content data is exported to `build/SITE-NAME/api/v1`. Each site exports a swagger-based documentation of the APIs at `/api` which is exported to `build/SITE-NAME/api/v1/swagger.json`.

Development

Thanks to react boilerplate this project includes a hot-reloading dev-server. To run first build the DLLs `node cli.js run-dev-dlls SITE-NAME` and then run the dev server `node cli.js run-dev SITE-NAME`.

1.2 Create a New Site

Sites are stored in the `sitesDir` which is stored in the root directory's `config.yml` file. The default sites directory is `./sites`.

1.2.1 Use Plop Generator

A new site can be created using plop: `node_modules/.bin/plop`.

1.2.2 Create a Site Manually

New sites can be created manually by copying the "test-site" in `./internals/models/tests/sites/test-site`.

1.2.3 Use Catalog Admin

Interra Catalog Admin includes a user interface for creating and editing sites.

1.2.4 Validating Sites

To validate your site configuration type: `node cli.js validate-site SITE`.

1.2.5 Site Configration

Below is a description of the site configuration file: `config.yml`.

Properties

	Type	Description	Required
name	`string`	The name of the site	Yes
schema	`string`	The schema of the site	Yes
identifier	`string`	Unique ID of the site	Yes
description	`string`	Description of the site	No
search	`string`	Search settings of the site	Yes
private	`object`	Private settings of the site	No
front-page-ic on-collection	`string`	The collection to be used for front page icons	No
front-page-ic ons	`array[]`	The icons to be used for front page icons	No
fontConfig	`object`	Configuration object for fonts	No

Additional properties are allowed.

name

The name of the site

- **Type**: `string`
- **Required**: No

schema

The schema of the site

- **Type**: `string`
- **Required**: No
- **Allowed values**:
  - `"pod-full"`
  - `"pod"`
  - `"test-schema"`

identifier

Unique ID of the site

- **Type**: `string`
- **Required**: No

search

Search settings for the site.

type:

The search engine to use for the site. Options include elasticLunr, elasticSearch, and simpleSearch

- type: `string`
- required: Yes

fields:

The fields to include in the search.

- type: `array`
- required: Yes

endpoint:

Endpoint for Elastic Search or other remote search backend

- type: `string`
- required: no
- format: uri

private

Private settings for the site that are not exported to the production instance. Includes settings for AWS and other services

description

Description of the site

- **Type**: `string`
- **Required**: No

front-page-icon-collection

The collection to be used for front page icons

- **Type**: `string`
- **Required**: No

front-page-icons

The icons to be used for front page icons

- **Type**: `array[]`
- **Required**: No

fontConfig

Configuration object for fonts

- **Type**: `object`
- **Required**: No

1.3 Schemas

Schemas are collections or JSON-Schema files as well as a few other settings. Each schema should have the following files and folders:

```
collections/
hooks/
config.yml
map.yml
UISchema.yml
PageSchema.yml
```

1.3.1 collections/

A JSON-Schema representation of each collection for the catalog. `$ref:` references are supported.

1.3.2 hooks/

Hooks for overriding docs. This is currently required.

1.3.3 config.yml

The schema's config.yml file has the following properties:

- **name** Human readable name of the schema
- **api** Currently "1" supported
- **collections** A list of collections the schema describes
- **facets** A list of facets that a catalog's search page would use for this schema. Will be moved to individual sites.
- **references** An object listing each collection that contains references to other collections and on what properties they are connected.
- **routeCollections** Collections that should have routes in the catalog.

1.3.4 map.yml

Each doc has several required fields that the catalog needs:

- title
- identifier
- created

- modified

The `map.yml` file allows schemas to map one of the required fields to an existing field in the schema. This keeps schema's from having to keep redundent data and schema's to remain as untouched as possible. For example Project Open Data uses `name` for the `Organization` instead of `title`. The `map.yml` file allows the `name` to be mapped to `title` for organizations for use in the catalog.

1.3.5 UISchema.yml

Used to map fields in each schema to a widget for the document creation and edit forms in the **'Interra Catalog Admin <>'_** project. This uses the React JSON-Schema Form library which provides that API. See that projects documentation for more details.

1.3.6 PageSchema.yml

Similar to the `UISchema.yml` file but for rendering collection pages in the react app. Still under development.

1.4 Harvesting

Harvests are configured and stored in the `sites/SITE-NAME/harvest` folder.

1.4.1 Harvest Sources

The `sources.json` file in the `harvest` folder provides a list of harvest sources. The list uses the following format:

- **id** The human readable identifier for the harvest source
- **source** The source of the harvest. Can be a remote `http://` or `https://` source or a local source `file://`.
- **type** The type of source. Currently DataJSON is the only option.
- **filters** Allows the filtering of sources by a key and value that will need to appear in each source document that is included in the harvest.
- **exclude** The opposite of filter.
- **overrides** Override a value in each doc.
- **defaults** Provides a default value only if that value is missing from each source doc.

1.4.2 Running Harvests

Caching

Harvests sources are first cached to local files before processing. This makes dealing with remote source timeout issues easier. Cached sources are stored in the `harvest/SOURCE-NAME/SOURCE-TYPE` folder. To run the cache type:

```
node cli.js harvest-cache SITE-NAME
```

Running

Once files are cached type the following to run a harvest:

```
node cli.js harvest-run SITE-NAME
```

Harvest sources are now stored in the site's `collections` folder as site documents. The harvest source is added to the `interra` object in each doc.

1.5 Publish Site

Sites are published as single page apps in the `build/` folder.

The published react app uses data from two sources:

1. Exported collection data and apis in the `api/v1` folder.
2. Contents of the site's `config.yml` that are exported by webpack as the `interraConfig` global variable for the app

Running the `node cli.js` command shows the available build commands.

1.5.1 Building Collection Data

Collection data is stored in `sites/SITE-NAME/collections`. The data stored there is "referenced" meaning it contains references to documents stored in other collections. When data is exported the documents are "derefenced" so they contain the full document with all of their referenced objects. Documents use the `interra.id` to reference each other.

Collection data is exported to `builds/SITE-NAME/api/v1/collections`. To export all collection data run:

```
node cli.js build-collection-data SITE-NAME
```

To export an idividual document run:

```
node cli.js build-collection-data-item SITE-NAME COLLECTION-NAME DOCUMENT-NAME
```

1.5.2 Building APIs

APIs are built use the `node cli.js` command. Available build commands include:

- **build-datajson** builds Projet Open Data's `data.json` file.
- **build-routes** builds a list of available routes in the `routes.json` file. Used by the react app to render collection page
- **build-schema** builds a description of the site schema in the `schema.json` file.
- **build-search** builds a search index in the `search-index.json` if elasticLunr or simpleSearch are used
- **build-swagger** builds a swagger api file for the site at `swagger.json`.

All APIs for a site can be built at once using `node cli.js build-apis SITE-NAME`.

1.5.3 Building the React App

A version of the react app is exported to each site directory. The only difference between them are the site configuraiton contained in the `interraConfig` variable which is exported as part of the app by webpack.

To build the app run `node cli.js build-site SITE-NAME`.

1.6 Configuration

1.6.1 Search

This project supports the following backends:

- ElasticLunr
- ElasticSearch
- Javascript Search

The ElasticLunr and Javascript Search create a search index that is fully loaded by the client browser. This is ideal for small to midsized catalogs. Since there is no external search service or API to maintain costs and complexity are significantly lower. Since all the work is done in the browser search results are virtually instantaneous.

ElasticLunr has features consistent with search engines such as boosts, several parsing options, stopwords, and more. ElasticLunr file size is about one and a half times larger than the Javascript Search backend which just uses a filter on the datasets. The size of the index is a determining factor in which backend to use. The search index file size depends on how much data is included from each dataset and what fields are indexed. A catalog of 500 datasets might average 500kb gzipped using ElaticLunr and 200kb gzipped using Javascript Search.

ElasticSearch offers a full featured search backend. Currently facets need more work for ElasticSearch.

Configuration

To configure the search set the following in `config.yml`. The exact types are `ElasticLunr`, `ElasticSearch` and `SimpleSearch` (Javascript Search).

```
search:
  type: simpleSearch
  fields:
    - title
    - keyword
    - publisher
    - description
    - theme
    - modified
    - distribution
```

For ElasticSearch include the `endpoint:` which should be the URL to the endpoint. ElasticSearch also requires the following in the `private` key in order to index correctly when using `node cli.js build-search`:

```
private:
  aws:
    accessKeyId:
    secretAccessKey:
    region:
    es:
```

(continues on next page)

(continued from previous page)

```
      endpoint:
      index:
```

1.6.2 Front Page Icons

The front page icons are associated with a certain collection. To set the collection add the following in `config.yml`:

```
front-page-icon-collection:
  - [COLLECTION]
front-page-icons:
 - [COLLECTION ITEM IDS]
```

For example:

```
front-page-icon-collection: theme
front-page-icons:
 - city-planning
 - finance-and-budgeting
 - health-care
 - public-safety
 - transporation
```

Adding Icons to Collection Items

The actual icon types are added to the collection items with the `icon` key. For example:

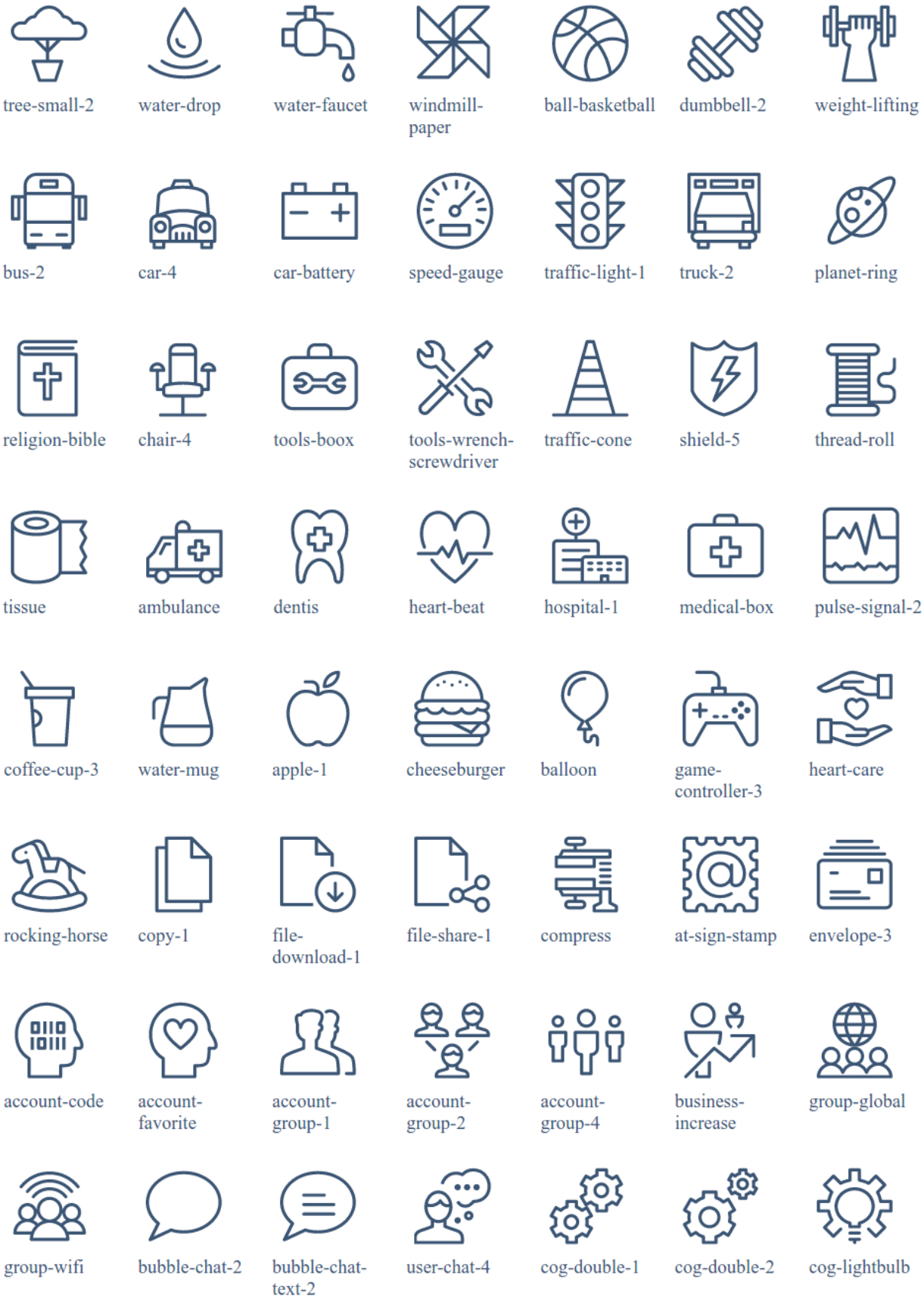
```
{
  "title": "City Planning",
  "identifier": "city-planning",
  "icon": "building-12"
}
```

Available Icons

Below is the default icon list:


computer-disk-drive-1
computer-imac-1
computer-notebook-2
sd-card-download
phone-2
phone-signal-full
mobile-phone-portrait
headphone
cursor-double-click-2
book-2
books-library
drawer-2
newspaper
crown-3
google-plus
trophy-3
lock-close-6
megaphone-1
download-3
cloud
graph
network
design-mug
picture-2
television
day-snow-thunder
umbrella-open


business-bag-cash
graph-bar-2
graph-bar-3d
graph-bar-increase
graph-line-2
graph-pie-2
bank-notes-3
coin-receive
piggy-bank
wallet-1
network-world
location-pin-8
location-pin-target-2
map-1
map-pin-2
bank-2
building-6
building-12
home-3
home-4
water-fountain
airplane-departure
cactus
eco-globe-1
eco-lightbulb
honeycomb
plant
sign-recycle


tree-small-2
water-drop
water-faucet
windmill-paper
ball-basketball
dumbbell-2
weight-lifting
bus-2
car-4
car-battery
speed-gauge
traffic-light-1
truck-2
planet-ring
religion-bible
chair-4
tools-boox
tools-wrench-screwdriver
traffic-cone
shield-5
thread-roll
tissue
ambulance
dentis
heart-beat
hospital-1
medical-box
pulse-signal-2
coffee-cup-3
water-mug
apple-1
cheeseburger
balloon
game-controller-3
heart-care
rocking-horse
copy-1
file-download-1
file-share-1
compress
at-sign-stamp
envelope-3
account-code
account-favorite
account-group-1
account-group-2
account-group-4
business-increase
group-global
group-wifi
bubble-chat-2
bubble-chat-text-2
user-chat-4
cog-double-1
cog-double-2
cog-lightbulb


cog
gauge
settings-1
settings-2
timer-full-2
watch-2
flash-1
typewriter-1
transport
safety
planning
healthcare
education
budget
data_dashboar
dataset
dkan_data_sto
feedback
group
resource
visualization
page