Arranger Documentation

Release 0.1.0

overture.bio

Apr 04, 2020

First Steps

CHAPTER 1

Introduction

1.1 What is Arranger?

Arranger is a collection of reusable components for creating centric search portals with Elasticsearch. Arranger consists of the f

- *Arranger Search API* provides a layer that sits above your Elasticsearch cluster to expose a data-model aware GraphQL API, generated from your own Elasticsearch index mapping.
- *Arranger Components* provides a rich set of UI components that are configured to speak to the search API.
- *Arranger Admin* provides the API and UI for configuring the search API and content management for the search portal.

Arranger is one of many products provided by Overture and is completely open-source and free for everyone to use.

See also:

For additional information on other products in the Overture stack, please visit https://overture.bio

1.2 Features

- GraphQL API for query flexibility.
- SQON query filter notation balances between human-interpretability and machine-readability to simply search.
- Admin UI for API configuration and content management.
- Configuration import and export for easy migration.

1.3 License

CHAPTER 2

Getting Started

The easiest way to understand Arranger, is to simply use it!

Below is a description of how to get Arranger quickly up and running, as well as a description of how Arranger works and some important terms.

2.1 Quick Start

The goal of this quick start is to get a working application quickly up and running.

Using Docker:

1. Download the latest version of Arranger.
2. From the Arranger root directory, run docker-compose:

```
$ docker-compose up -d
```

Arranger should now be deployed locally.

Alternatively, see the Installation instructions.

2.2 How Arranger Works

1. Starting with some Elasticsearch (ES) indices with mappings.

- Arranger makes no assumption about your data model.
- Model your index mappings and index them.
- For demo convenience, you can follow a *tutorial bellow* to index some test data from our Kids First project.

See also:

The Overture software suite also provides Maestro for indexing genomic data to ES

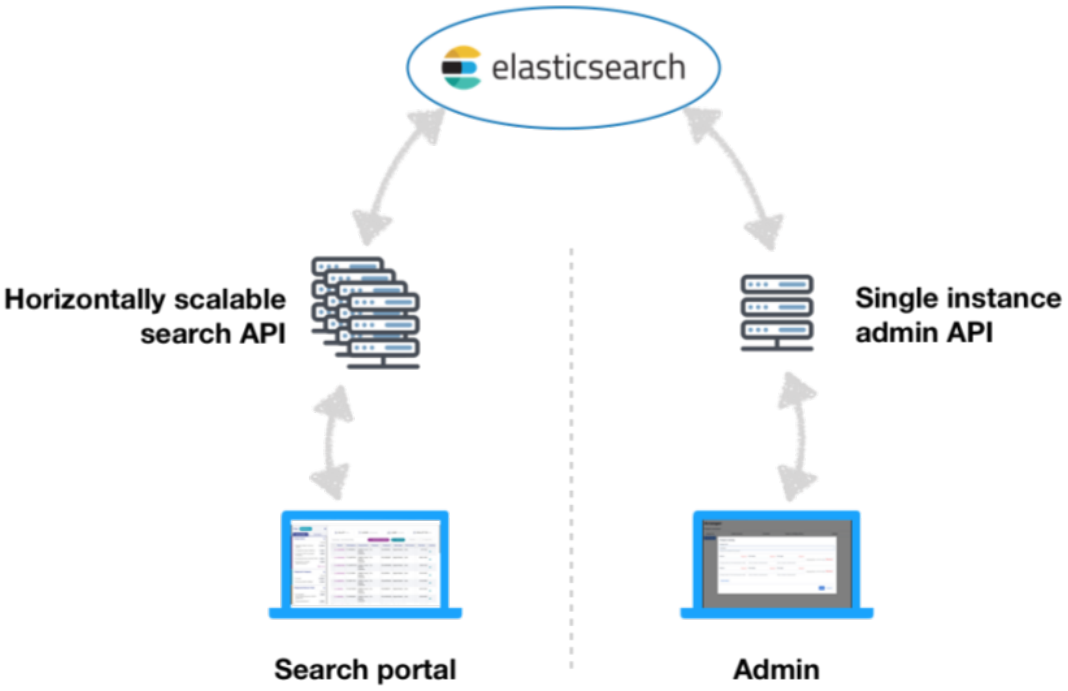
2. Create an API version for your project from Arranger Admin.

- From your browser, navigate to http://localhost:8080
- Click **"Add Project"**
- Input your project id in **snake_case**
- Click **"Add Index"** for each index you want to expose from ES, with the following fields:
  - **"Name"**: any name for your index, in **camelCase**
  - **"ES Index"**: the index that you want to expose
  - **"ES Type"**: the type that you want to expose
- Click **"Add"** once finalized.
- Navigate into your newly registered project's configuration and ensure that **"Has Mapping"** is **"yes"** for all indices registered.
- Configure your project from the API and click **"Save"** to save as a new project.

3. View your data in a portal.

- From a UI:
  - Go to http://localhost:8081/?selectedKind=Portal.
  - Select your project and index from the dropdown.
  - Note: a production-ready white-label portal using UI components provided by Arranger is in our roadmap for Arranger.
- From the GraphQL API:
  - Each Arranger project created through the Admin system in step 2 creates a new Graphql endpoint.
  - Start a GraphQL IDE (such as GraphiQL or GraphQL Playground
  - Point your IDE to `http://localhost:5050/<project_id>/graphql` to explore the API schema (where `<project_id>` is the project id you have input in step 2).
  - For documentation regarding this API, check out the Arranger for Application Developers guide

2.3 Architecture



2.4 Indexing Demo Data

- From your browser, visit the locally running Kibana at http://localhost:5601 and go to Dev Tools
- Creating a `file_centric` index:
  - Run these commands to create a `file_centric` index and add a mapping then these commands to index some demo documents into the index
  - Run these commands to create a `participant_centric` index and add a mapping then these commands to index some demo documents into the index
- You can run `GET file_centric/_mapping` and `GET participant_centric/_mapping` to confirm that the mapping has been created successfully

CHAPTER 3

Arranger for Administrators

3.1 Tutorial

To administer Arranger, the admin must:

1. Install Arranger.

View the installation instructions.

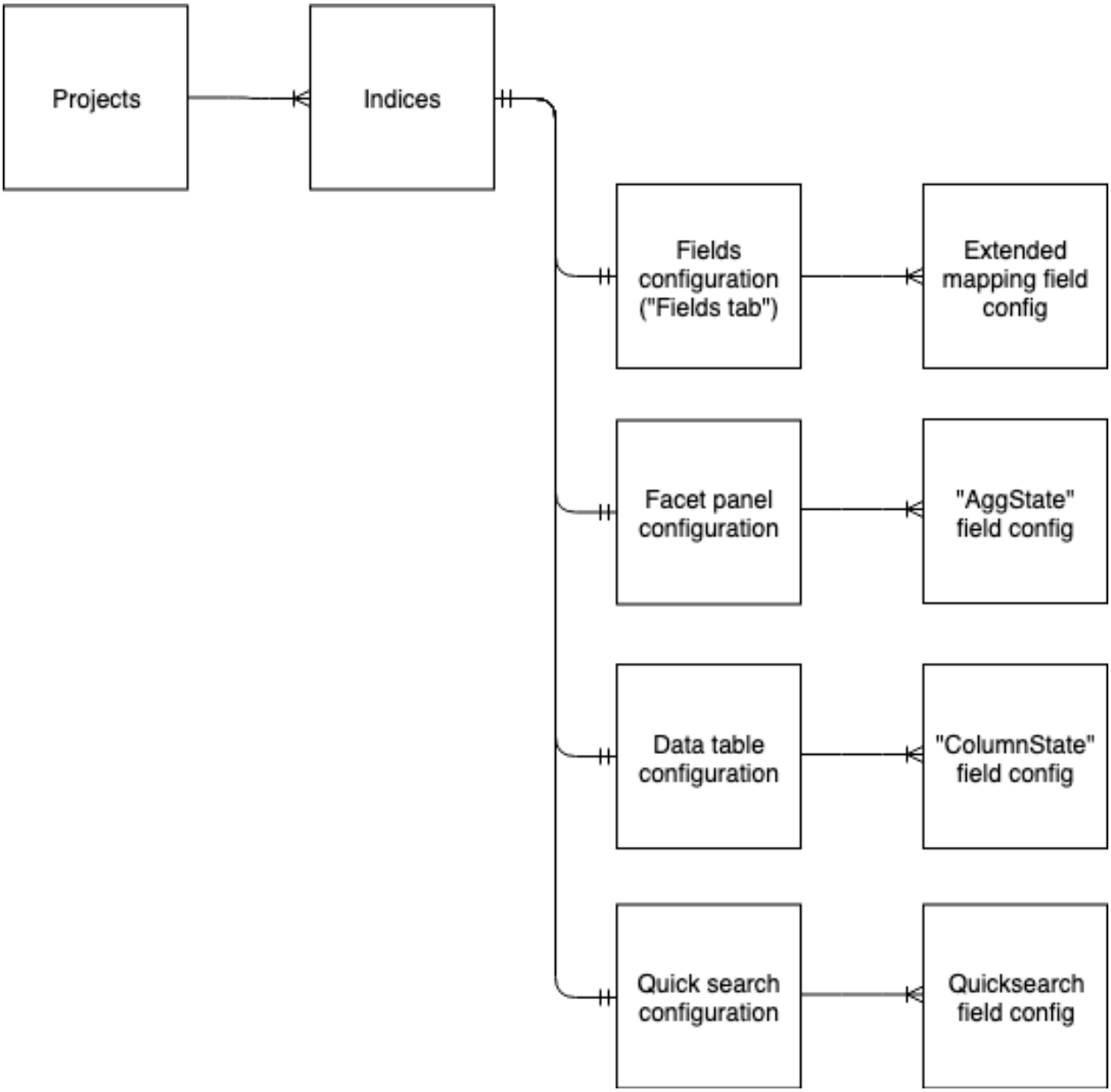
2. Have an Elasticsearch mapping and data indexed to search.

View the Indexing Demo Data for a demo setup.

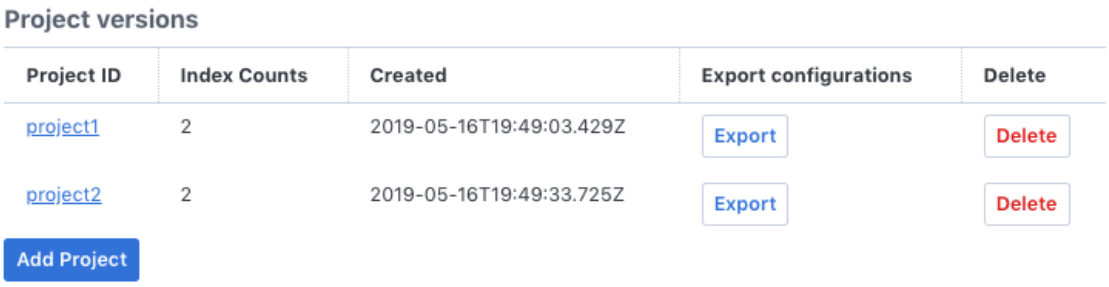
3. Admin registers the indices with arranger through the admin UI and apply configurations.

3.2 Using the Admin UI

The arranger UI reflects the following pseudo entity relationship:



1) Projects:



This page lists the available projects and provides an interface for registering new projects

Available functionalities:

- Adding a new project
- Removing existing project
- Export configuration data (exported data can then be imported into new projects to migrate data).

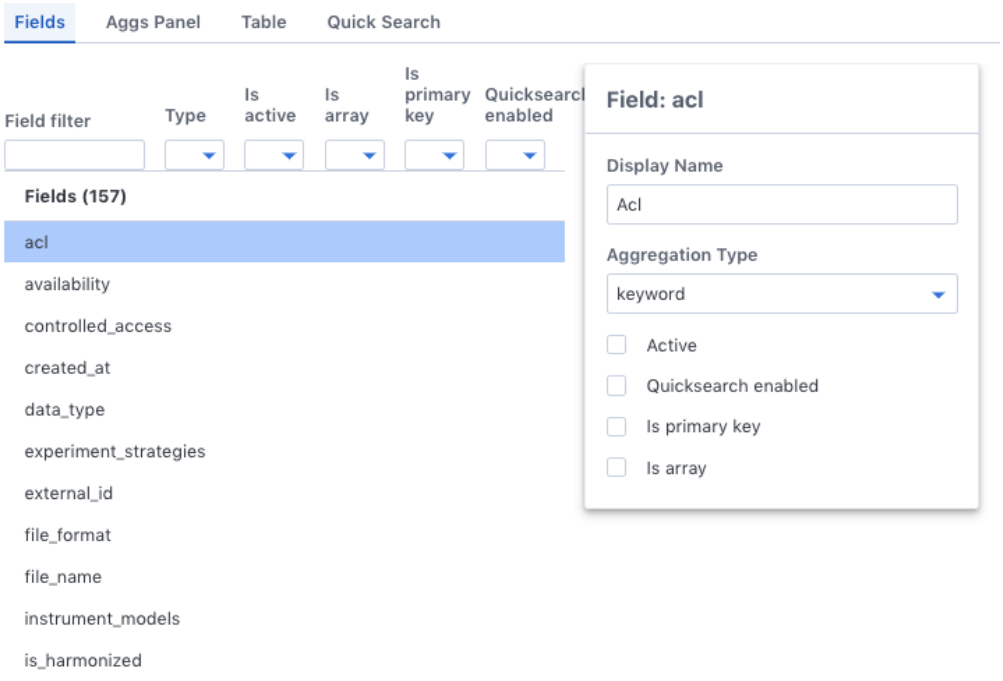
Clicking on a project id will navigate to that project's list of indices.

2) Indices:

This page lists the indices registered to Arranger under the selected project.

Clicking on an index name will navigate to the configuration page for the index. The following configurations are available:

a) Fields configurations



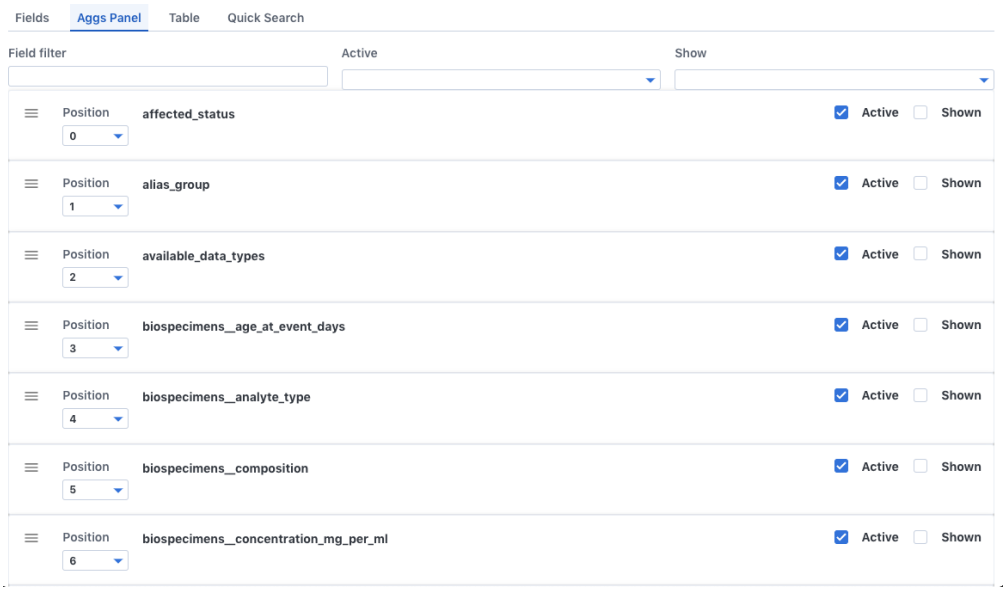
This lists all fields available in the index and allows configuration of Arranger metadata for these fields, including:

- **Display Name**: how the field should be displayed to user.
- **Aggregation Type**: lets the search portal know how to display aggregation filters for the field.
- **Active**: this field is **DEPRECATED**
- **Quicksearch enabled**: whether the field is enabled for quicksearch using the *@arranger/components*'s *QuickSearch* component.

- **Is primary key**: check if the field is the unique identifier for the index's main entity.
- **Is array**: check if the field is an array. Elasticsearch's mapping does not specify this information.

For convenience, filtering on the fields can be done through the inputs above the header.

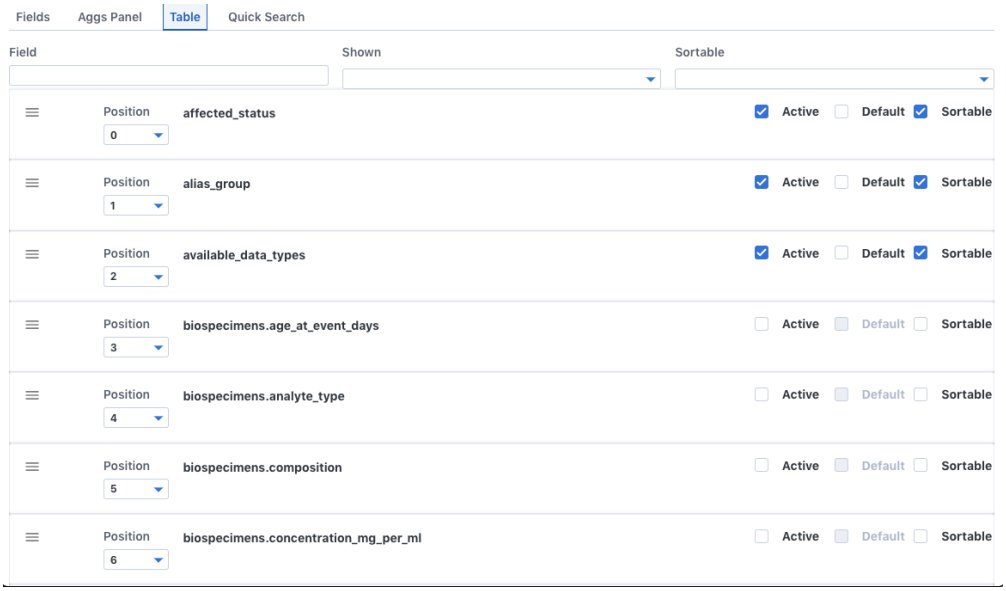
b) Facet panel configurations



This lists all available aggregations on the fields mentioned. On Arranger's default portal UI, this list is rendered as a facet panel. Each entry on Supported configutations:

- Ordering the facets: drag the facet on its "hamberger menu icon" to place the facet at the desired position. Alternatively, the position can also be set through the select menu beside the icon.
- **Shown**: displays the facet in the portal's facet panel.
- **Active**: enables this facet for search. An *Active* facet will appear in the *Advanced-FacetView* component in *@arranger/components*. Only facets that are both *Active* and *Shown* will be shown in the portal's facet panel.

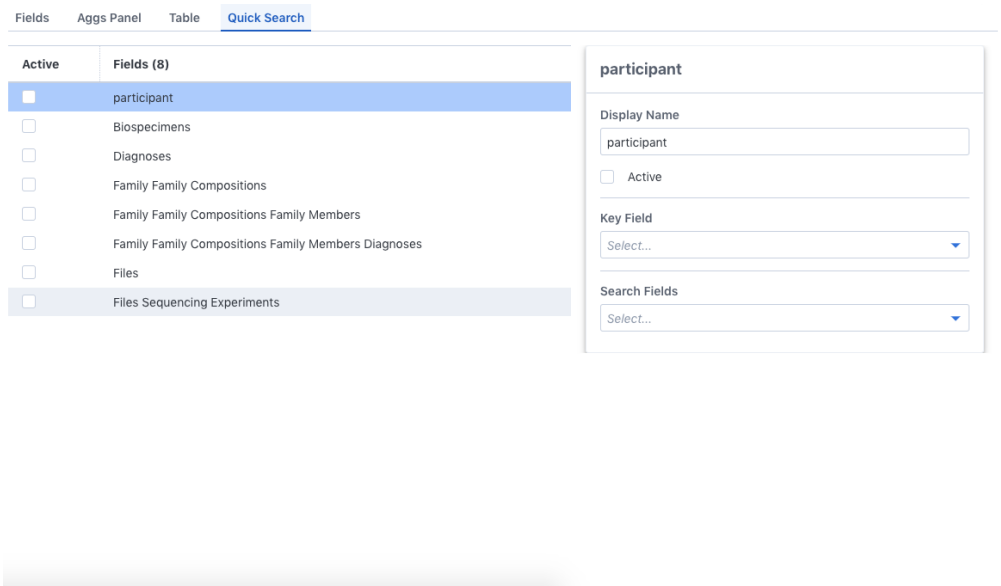
c) Data table configurations



This contains configuration for the data table in the default portal. Each entry in the list represents a column in the data table. Available configurations:

- Column order: positioning can be done by dragging or using the select, similar to the facet panel.
- **Active**: enables this column to be viewed in the table. Does not show by default.
- **Default**: shows this column by default. Can only be checked if *Active* is checked.
- **Sortable**: enables sorting of the table on this field.

d) Quick search configurations



This contains configuration for the portal's quick-search feature, which allows users to filter indexed entities by text. Currently, Arranger only supports exact match on quicksearch, but free-text search is in our roadmap to support. This feature can be exposed to end-users throught the *QuickSearch* UI component from *@arranger/components*.

Only entities (in other words, the root object and its "nested" fields in Elasticsearch) are available for quick search.

Available configurations:

- **Display Name**: the name to display this field as.
- **Active**: check to enable search for this entity.
- **Key Field**: the unique field that identifies each instance of this entity.
- **Search Field**: the properties of the entity to enable search on.

CHAPTER 4

Arranger for Application Developers

Arranger comes in individual pieces that can be flexibly composed together to meet your application's needs. These include:

- @arranger/server: the main server-side application
- @arranger/components: UI components used for building end-user facing applications
- @arranger/admin: the server-side admin Graphql API
- @arranger/admin-ui: the UI interface as described in the Arranger for Administrators guide.

Additionally, some packages that are used internally are also published. These include:

- @arranger/schema: contains the Graphql schema generated and served by *@arranger/server*.
- @arranger/mapping-utils: contains utility functions used for computing / interpreting elasticsearch mappings and Arranger metadata about the mappings.
- @arranger/middleware: responsible for translating SQON and aggregation parameters from the *@arranger/server* to elasticsearch queries and aggregations.

4.1 Server-side

On the server side, *@arranger/server* and *@arranger/admin* are the relevant packages.

Some prerequisit:

- Elasticsearch version 6.6.1 running.
- Kibana version 6.6.1 (optional)
- NodeJs version 10

There are multiple ways to get up and running with Arranger on the server-side:

1) Running a stand-alone all-in-one instance:
    - Using Docker:

1) The latest arranger server image is available on Dockerhub
2) Alternatively, you may build an image using the *Dockerfile.server* file from the Arranger source

- Running with Node:

1) Clone the Arranger repo: `git clone git@github.com:overture-stack/arranger.git`
2) Navigate to the directory: `cd arranger`
3) Install dependencies: `npm ci && npm run bootstrap`
4) Navigate to the *modules/server* directory: `cd modules/server`
5) Start the server: `npm start`

This will start an instance of `@arranger/server` on port `5050`.

By default, this bundle also comes with the admin API from `@arranger/admin` serverd at `/admin/api`. From your browser, navigate to http://localhost:5050/admin/graphql to explore this API

Limitation of this approach: the API from `@arranger/admin` is **not** meant to be exposed to end-users, hence also **not horizontally scalable**. For the second a production-ready setup, please use the next option:

2) Running with custom express apps:

- Example search app (horizontally scalable):

```
import express from 'express';
import Arranger from '@arranger/server';

const PORT = 9000

Arranger({
  esHost: "http://localhost:9200"
}).then(router => {
  const app = express();
  app.use(router);
  app.listen(PORT, () => {
    console.log(` search API listening on port ${PORT}  `)
  })
})
```

- Example admin app (single instance):

```
import express from "express";
import adminGraphql from "@arranger/admin/dist";

const PORT = 8000

adminGraphql({
  esHost: "http://localhost:9200"
}).then(adminApp => {
  const app = express();
  adminApp.applyMiddleware({
    app,
    path: "/admin"
  });
```

(continues on next page)

(continued from previous page)

```
  app.listen(PORT, () => {
    console.log(` Admin API listening on port ${PORT} `)
  })
})
```

Both applications should be interacting with the same Elasticsearch instance. Since they are two separate applications, they can be scaled separately, with separate authentication and authorization rules.

4.2 Browser-side

On the browser side, *@arranger/admin-ui* and *@arranger/components* are the relevant packages. Both packages are both written in React, hence we recommend using React for your application for the most seamless integration.

- *@arranger/admin-ui*: This package provides the admin interface that is documented in the Arranger for administrator section.

  Integration with your React app:

  1) Install the package: `npm i @arranger/admin-ui`
  2) Integrate into your app:

```
import ArrangerAdmin from '@arranger/admin-ui/dist';
import { Route, Switch } from 'react-router-dom';

const ArrangerAdminPage = () => (
  <ArrangerAdmin basename="/admin" apiRoot="http://localhost:8000"␣
↪fetcher={fetch} />
)
```

  Configurations:

  - `basename`: tells `ArrangerAdmin` to treat `/admin` as the root path for client-side routing.
  - `apiRoot`: tells `ArrangerAdmin` to communicate with back-end API hosted at `http://localhost:8000`
  - `fetcher`: allows specifying custom data fetcher to use, this is usefull for integrating custom client-side loggins / authorization logics. `fetcher` must implment the Fetch API.

- *@arranger/components*: This package provides UI components that are pre-configured to work with the *@arranger/server* API. To explore the components this package provide, follow the steps bellow:

  1) Clone the Arranger repo: `git clone git@github.com:overture-stack/arranger.git`
  2) Navigate to the directory: `cd arranger`
  3) Install dependencies: `npm ci && npm run bootstrap`
  4) Navigate to the *modules/components* directory: `cd modules/components`
  5) Start the Storybook server: `npm run storybook`

  A basic repo UI can be found at: *arranger/modules/components/stories/Portal.js*

CHAPTER 5

SQON Filters

Arranger uses a custom JSON object format for filtering that is called SQON (pronounced like "Scone"). SQON provides a flexible system for combining many different filters.

A SQON object consists of nested objects of two types: **Operations** and **Values**.

Operation objects apply boolean logic to a list of operation objects. They are of the form:

Combination Operation (aka, Boolean Operation) which groups one or more filters

```
{
  "op":"", //Operation to apply to content ["and", "or", "not"]
  "content":[] //List of Operation objects that the boolean operation will␣
↪apply to
}
```

OR

Field Operation that applies to a filter to Value Object

```
{
  "op":"", //Operation to apply to content ["in", "<=", ">="]
  "content":{} //Value object specifying the field and list of values that␣
↪the field must be "in" or "not-in"
}
```

Value objects specify a list the field name and values for it that the wrapping . This filter can specify to include or exclude fields with any of the listed values. It will have the following format:

```
{
  "field":"", //name of the field this operation applies to
  "value":[] //List of values for the field if using the "in" operation, or a␣
↪scalar value for ">=" and "<=" operations
}
```

The top level of a SQON must always be a Combination Operation, even if only a single filter is being applied.

5.1 Sample

```
{
  op: "and",
  content: [
    {
      op: "or",
      content: [
        {
          op: "in",
          content: {
            field: "id",
            value: ["id123"]
          }
        }
      ]
    },
    {
      op: "in",
      content: {
        field: "id",
        value: ["id123"]
      }
    }
  ]
}
```

CHAPTER 6

Installation

6.1 Coming Soon

CHAPTER 7

Architecture

Coming Soon

CHAPTER 8

Technology Stack

8.1 Coming Soon

CHAPTER 9

Contributing to the Arranger Project

Coming Soon

CHAPTER 10

Contribute

If you'd like to contribute to this project, it's hosted on github.

See https://github.com/overture-stack/arranger

10.1 Indices and tables

- genindex
- search