UI Development

Part 2

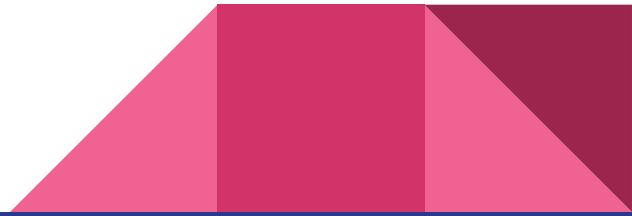
Town Hall Presentation - 12/2020

What We Know So Far

An Overview

DataHub UI is a...

- Mono-repository using Yarn workspaces (https://classic.yarnpkg.com/en/docs/workspaces/)
- Which consists of an Ember application (https://guides.emberjs.com/release/)
- And various Ember addons (https://cli.emberjs.com/release/writing-addons/) and npm packages
- Written in TypeScript (https://www.typescriptlang.org/) for Ember (https://github.com/typed-ember/ember-cli-typescript)

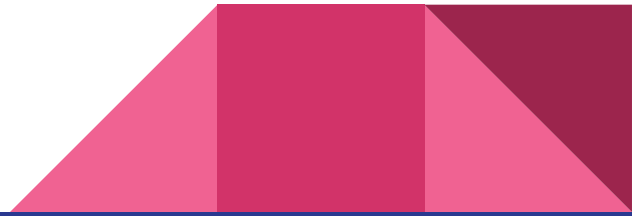


What We Know So Far

An Overview

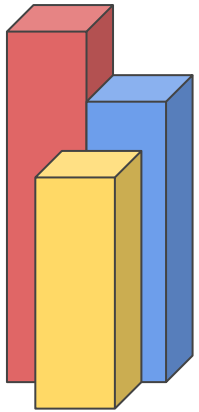
DataHub UI code consists of

- packages/data-portal => a Ember application
- @datahub/** => addons that are consumed by the Ember application representing parts of the application
- @nacho-ui/** => addons that contain general UI components not specific to DataHub features

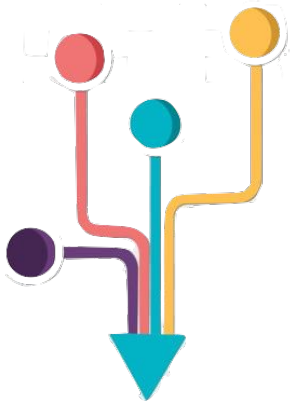


Topics for This Discussion

Pillars of DataHub UI



Data Flow in DataHub UI

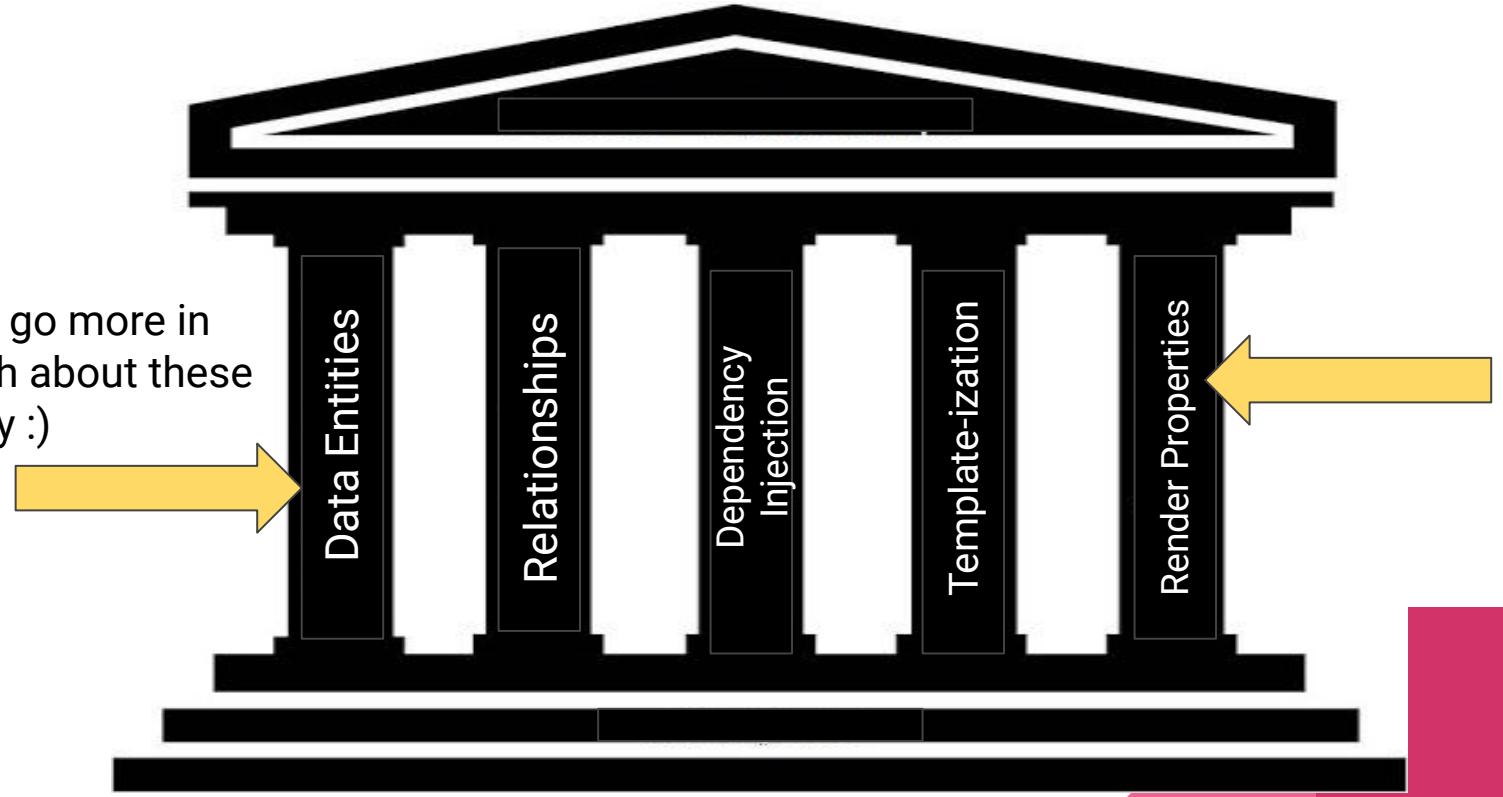


Configuration Based UI

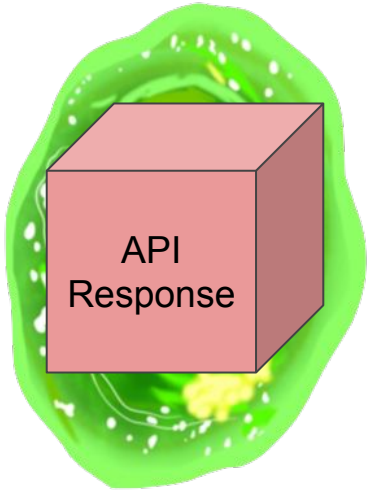
Roadmap & Next Steps

Pillars of DataHub UI

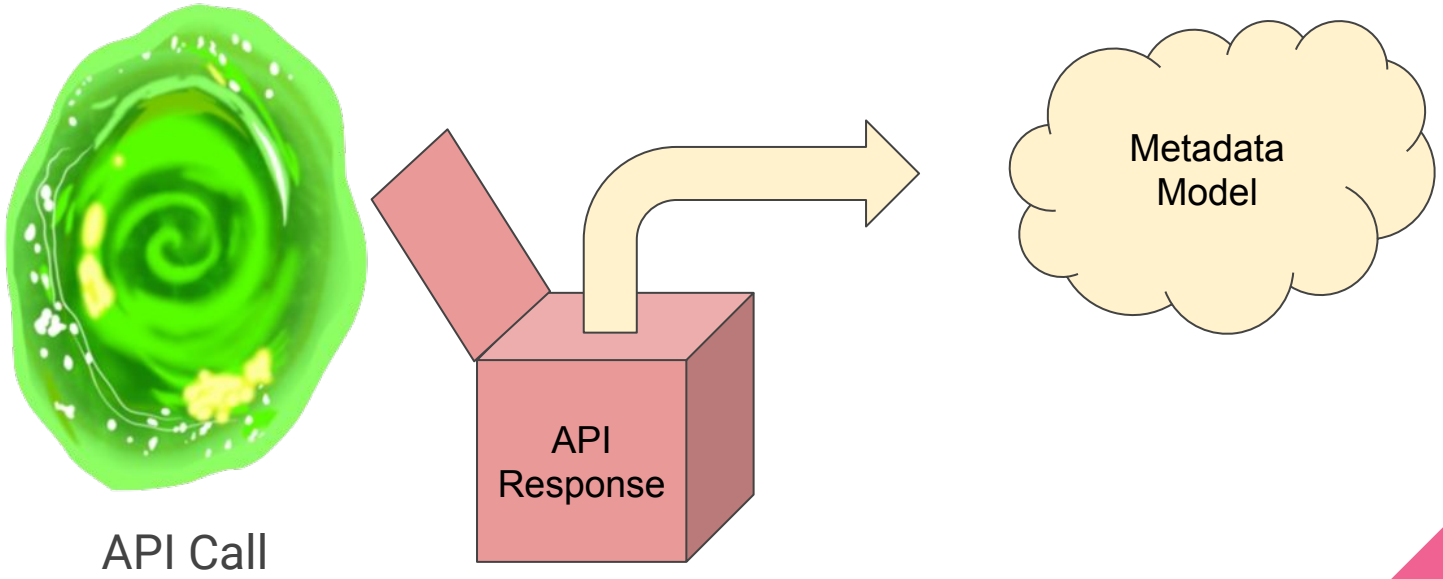
We'll go more in depth about these today :)



Data Flow in DataHub UI

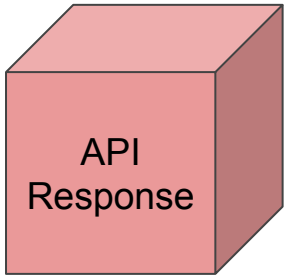


Data Flow in DataHub UI

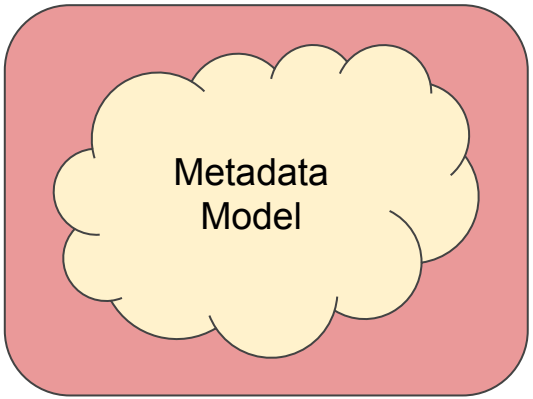


Data Flow in DataHub UI

API Call



UI Data Entity



Entities

If it’s an entity modeled in our metadata, it should be an entity on the UI.

```
export class DatasetEntity {
  static displayName: 'datasets' = 'datasets';
  entity?: Com.Linkedin.Dataset.Dataset;
  /**
   * For open source support only, creates a reference to the customProperties, which can be found as an aspect of the
   * dataset entity. This helps to display various properties that may be specific to certain datasets and allows for
   * flexible display
   */
  @oneWay('entity.customProperties')
  customProperties?: Com.Linkedin.Dataset.DatasetProperties['customProperties'];

  /**
   * Computes the custom dataset properties as a list of label and values rather than object mapping
   */
  Complexity is 3 Everything is cool!
  @computed('customProperties')
  get customDatasetProperties(): Array<{ label: string; value: string }> | void {
    const { customProperties } = this;
    if (customProperties) {
      return Object.keys(customProperties).map((key): { label: string; value: string } => ({
        label: key,
        value: customProperties[key]
      }));
    }
  }
}
```

Represented as a JS/TS class in the UI

Abstraction over an object that is the actual entity returned from the API

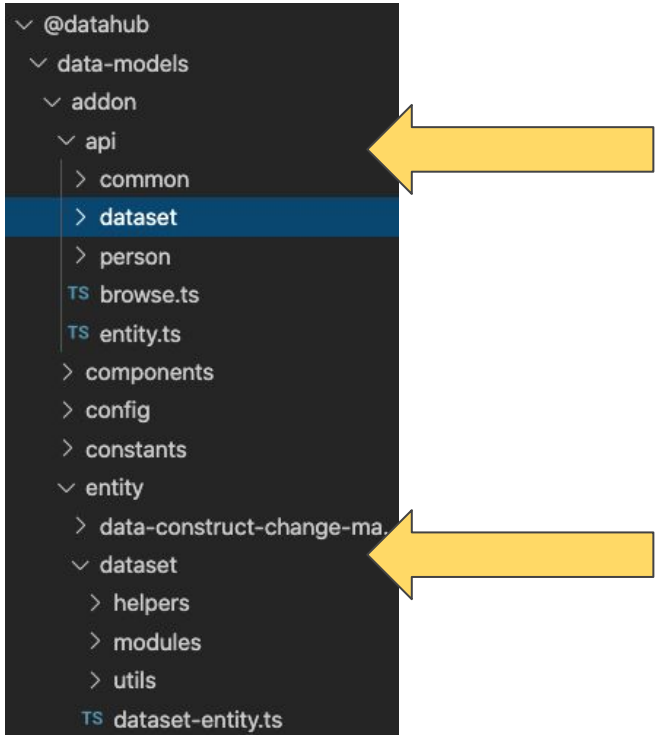
Contains information about entity behavior in UI rendering/data fetching

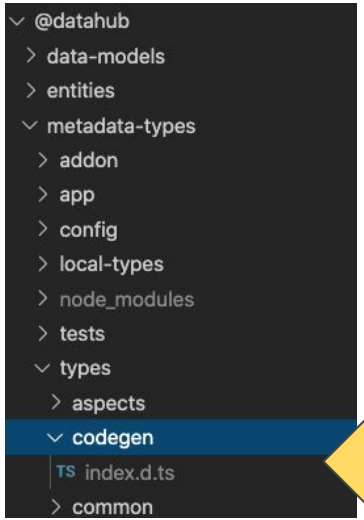
Entities

Why do we have this?

- We need to understand more about an entity than just what is returned from the API, including behavior of the entity on the UI and how it should interact with other entities
- Centralizes definitions of the above as well as being an instance of a store for data management
- Abstracts concerns about API response object and allows the view based components to focus more on how to interact with the entity itself

Entities

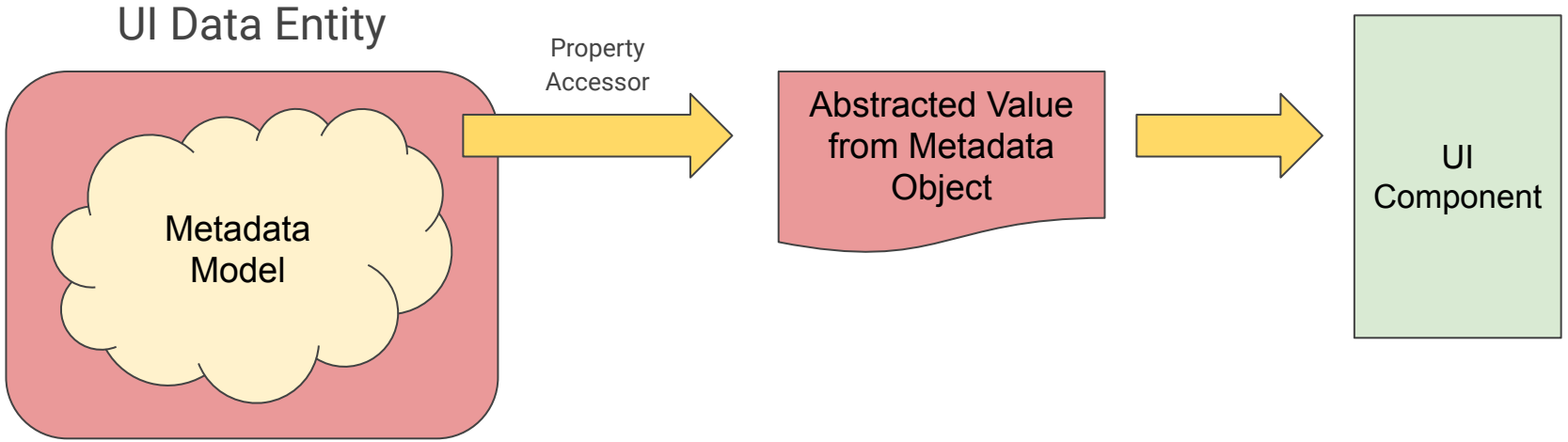




API calls and data models are both located in @datahub/data-models under the api and entity folders.

API response “metadata-types” are generated from PDL models and found in @datahub/metadata-types under types/codegen

Data Flow in DataHub UI



Data Flow in DataHub UI

Entity-specific components that are specialized for one particular kind of entity

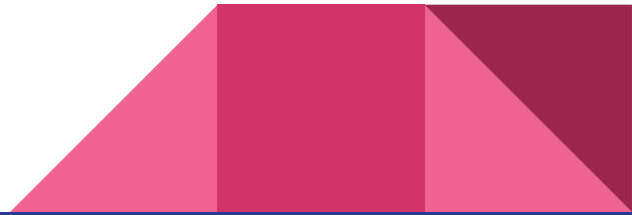
Components that generically relate to a feature or idea that are common among two or more entities

UI Component

Configuration Based UI

What is it?

- Generalized UI templates for commonly used components, such as tables, search results, and even entire entity pages
- Which components to render and how they behave is based on a JS/TS object resembling a JSON object
- This object is attached to an entity class so that it can be used where the entity is used and read when relevant



Configuration Based UI

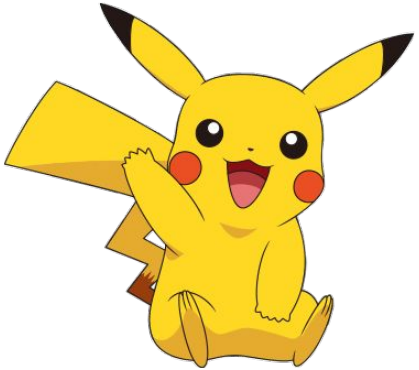
Why do we have it?

- Allows a familiar UI developer to easily spin up basic components for a new entity or new feature/aspect on an existing entity without having to rebuild similar UI over and over
- Allows a developer who is not familiar with UI (such as a backend dev) to be able to make changes and edits to a set of UI behavior without needing in depth knowledge on how the UI runs (if you can edit JSON, you can edit UI)
- Current methodology is not perfect, it’s a work in progress

Configuration Based UI

How it works, an easy example

```
tableProps = {
    headers: [‘Name’, ‘Type’],
    picture: ‘pikachu’,
    propertyNames: [‘displayName’, ‘type’]
};

objects = [
    {
        displayName: ‘Pikachu’,
        id: ‘pikachu’,
        type: ‘electric’
    },
    {
        displayName: ‘Charmander’,
        id: ‘charmander’,
        type: ‘fire’
    }
];
```

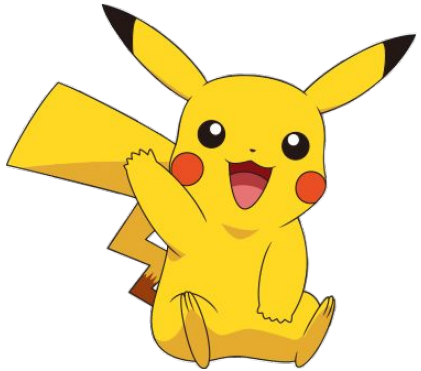


Name	Type
Pikachu	electric
Charmander	fire

Configuration Based UI

How it works, an easy example

```
tableProps = {
    headers: [‘Pokemon Name’, ‘Type’],
    picture: ‘pikachu’,
    propertyNames: [‘id’, ‘type’]
};

objects = [
    {
        displayName: ‘Pikachu’,
        id: ‘pikachu’,
        type: ‘electric’
    },
    {
        displayName: ‘Charmander’,
        id: ‘charmander’,
        type: ‘fire’
    }
];
```



Pokemon Name	Type
pikachu	electric
charmander	fire

Configuration Based UI

How it works, an easy example

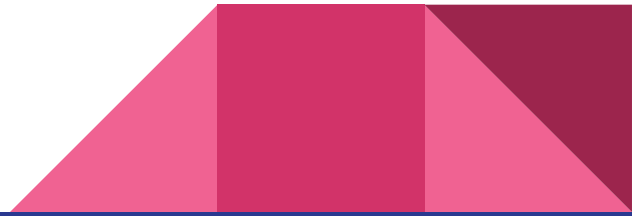
```
tableProps = {
    headers: [‘Name’, ‘Type’],
    picture: ‘eevee’,
    propertyNames: [‘displayName’, ‘type’]
};

objects = [
    {
        displayName: ‘Pikachu’,
        id: ‘pikachu’,
        type: ‘electric’
    },
    {
        displayName: ‘Charmander’,
        id: ‘charmander’,
        type: ‘fire’
    }
];
```

Name	Type
Pikachu	electric
Charmander	fire

Configuration Based UI

How it works, a real example

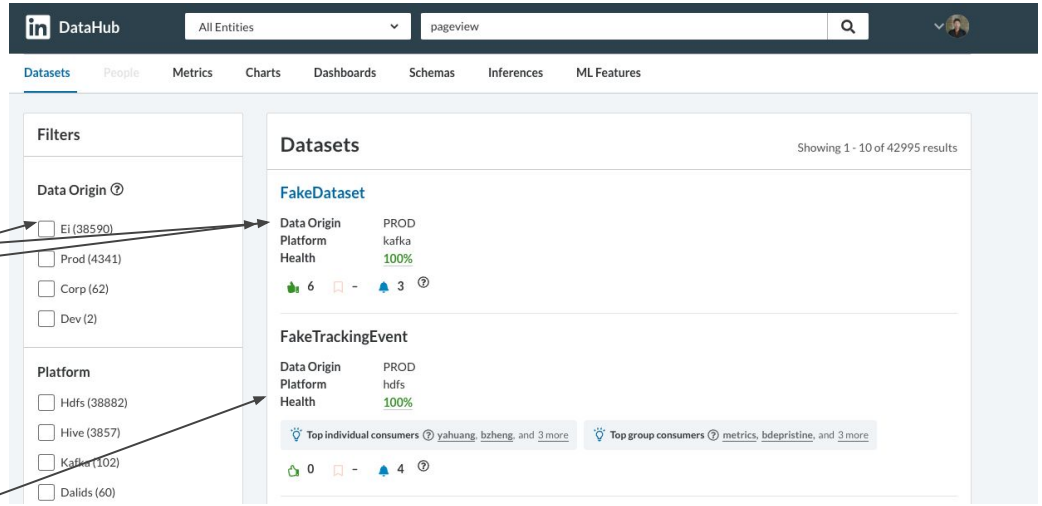
```
{
    apiEntityName,
    search: {
      placeholder: 'Search for datasets...',
      attributes: fields,
      secondaryActionComponents: [],
      customFooterComponents: [{ name: 'social/containers/social-metadata' }],
      isEnabled: true
    },
    userEntityOwnership: { … },
    browse: { showHierarchySearch: false },
    entityPage: { … }
  };
```



Configuration Based UI

How it works, a real example

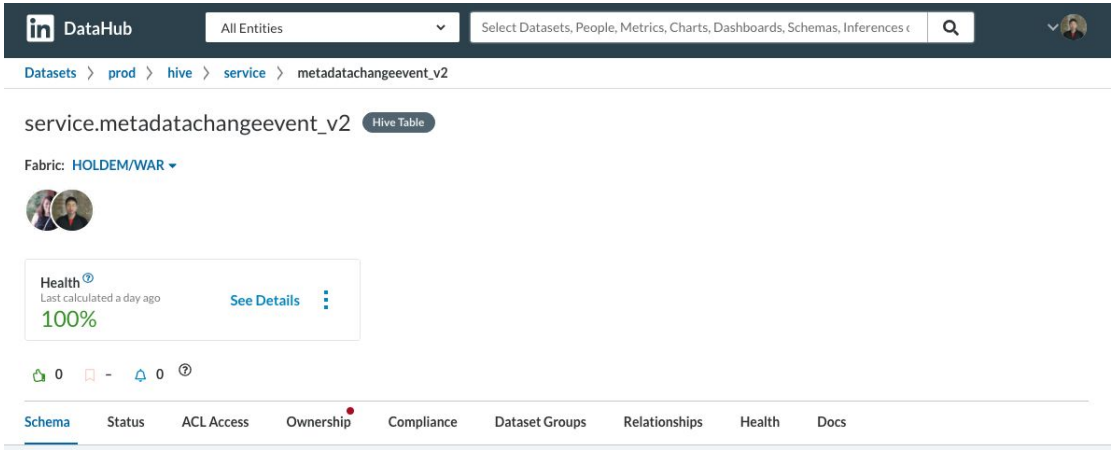
```
const fields: Array<ISearchEntityRenderProps> = [
  {
    showInAutoCompletion: true,
    fieldName: 'dataorigin',
    showInResultsPreview: true,
    displayName: 'Data Origin',
    showInFacets: true,
    desc: 'The data origin of the dataset',
    example: 'dataorigin:PROD',
    ...
  },
  {
    showInAutoCompletion: true,
    fieldName: 'health',
    showInResultsPreview: true,
    displayName: 'Health',
    showInFacets: false
    ...
  },
];
```



Configuration Based UI

How it works, a real example

```
{
    search: { … },
    entityPage: {
      route: 'datasets.dataset',
      tabProperties: [],
      defaultTab: DatasetTab.Schema,
      attributePlaceholder: '-',
      apiRouteName: 'datasets',
      pageComponent: {
        name: 'datasets/dataset-page'
      },
      customHeaderComponents: [
        {
          name: 'dynamic-components/entity/field',
          options: { className: 'dataset-header__description', fieldName: 'description' }
        },
        { name: 'datasets/containers/dataset-owner-list' }
      ]
    }
  };
```

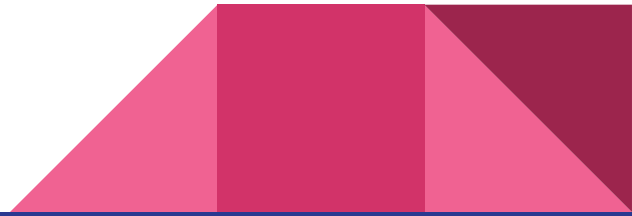


Roadmap & Goals

What we want to bring back to the future

How we want to build more engagement with our UI from developers:

- GraphQL
  - Part of data models sounds like a poor person's GraphQL, that's because it probably is
  - More standardized way to interact with our API and have similar abstractions
- Framework agnostic UI modeling and render props
  - Ember is too hard, let's make life easier. More plain old JS == easier to work with for non-Ember and/or non-UI experts
- React one day?
  - More popular, easier to pick up



Thanks for attending!

For more information, we are working on adding documentation and guides to provide additional clarity and insight into the UI work:

https://github.com/linkedin/datahub/tree/master/datahub-web