NetApp:
Swagger UI Alternatives

Group Members: Sophie Sfeir, Anthony McGaw, Jarod Miller, Madison Haugh

Sophie Sfeir

About me:

- University of Pittsburgh Student, BS in Computer Science and Spanish, Graduating in Fall 2021

Why I chose this project:

- I am interested in learning new ideas/concepts and wanted to learn more about APIs and how they work. I also wanted to gain more professional experience in software development.

Anthony McGaw

About me:

- University of Pittsburgh Student, BS in Computer Science, Graduating in May 2021

Why I chose this project:

- I have recently been interested in UX and this felt like a interesting project to get experience working on a project which is important to end-users. Also valuable experience with Agile, React, and API's.

Jarod Miller

About me:

- University of Pittsburgh Student, BS in Computer Science, Graduating in May 2021

Why I chose this project:

- I’m interested in gaining experience in full-stack website development and learning new technologies in the tech industry.

Madison Haugh

About me:

- University of Pittsburgh Student, BS in Computer Science, Minor in Applied Statistics, Graduating in May 2021

Why I chose this project:

- I've had previous experience with full-stack web development and was interested in developing and expanding my skill set

Outline

- Project Development Process
- Project Experience
- Project Goals
- Choosing a Visualizer
- Challenges
- Features/Functionalities
- Demo
- Testing
- Future Goals

Development Process

Student Team:

- ❏ Daily collaboration through our team Discord server
- ❏ Weekly meetings on Tuesday nights and Friday afternoons via Zoom
  - ❏ (which evolved into us adding Thursday nights as well)
- ❏ Split up certain feature development into sub-teams within our group

\+ Netapp:

- ❏ Second collaborative Discord including our NetApp sponsors to ask questions and discuss challenges
- ❏ Weekly meetings Friday morning at 9am ☀️

Project Experience

- Meet with team members frequently and work well together
- Research and project development process has been a great learning experience
- Communication with NetApp is positive and helpful
- Great industry learning experience

Project Goals

- Investigate current and new visualizers to determine the best option based on functionality and user experience
- Research OpenAPI Specification and understand how and what the visualizer is displaying
- Implement or inherit a deep-search ability for the documentation
- Implement version tracking
- Stretch Goal: Enhance styling and determine other potential features

Choosing an API Visualizer

Swagger
- **Pros:** open source, currently implemented
- **Cons:** one-panel design which leads to long scrolling

Redoc
- **Pros:** built-in search function, three-panel design, easy navigation
- **Cons:** search is not deep, large codebase

Carte
- **Pros:** Based off of Swagger UI
- **Cons:** Built in Ruby, limited documentation

Choosing an API Visualizer

Slate
- **Pros:** built in search, three panel design
- **Cons:** yaml must be converted to markdown with third-party tool, written in ruby (unfamiliar)

DapperDox
- **Pros:** side panel, easy customization,
- **Cons:** yaml must be converted to json, no search functionality

LucyBot
- **Pros:** Search functionality, three panel design
- **Cons:** Simplistic, paid version

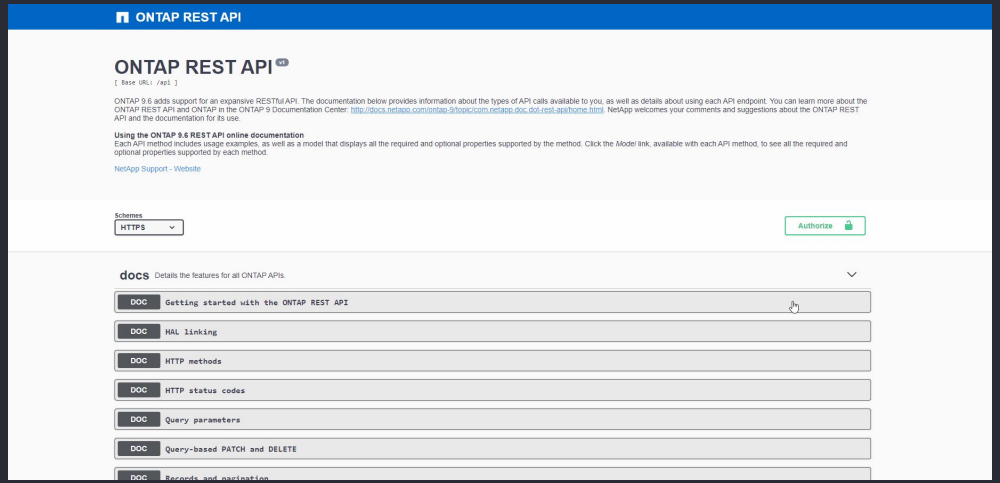
Swagger UI

Benefits:

- ❏ NetApp's current implementation
  - Has custom components
- ❏ Try-It-Out feature
- ❏ Built-in Model rendering

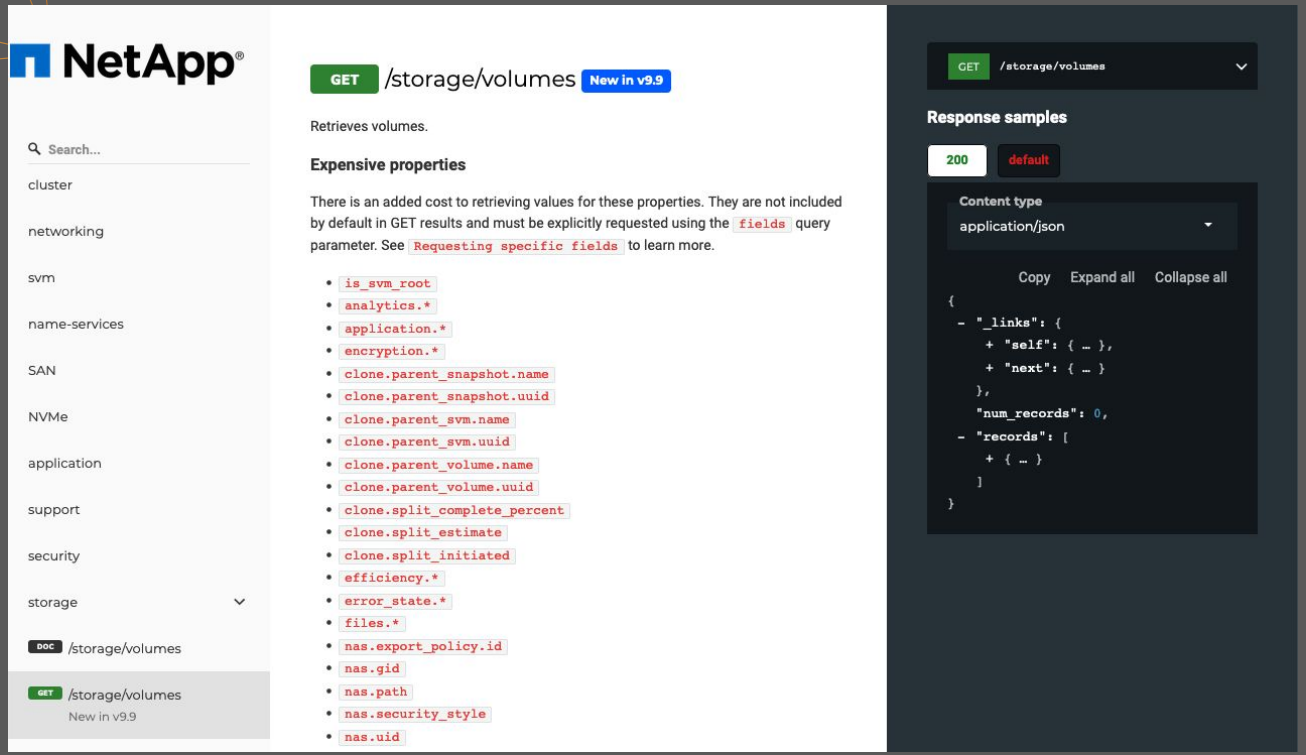
Shortcomings:

- ❏ No deep search functionality
- ❏ Single panel design



scrolling....scrolling...scrolling

Decision: Redoc



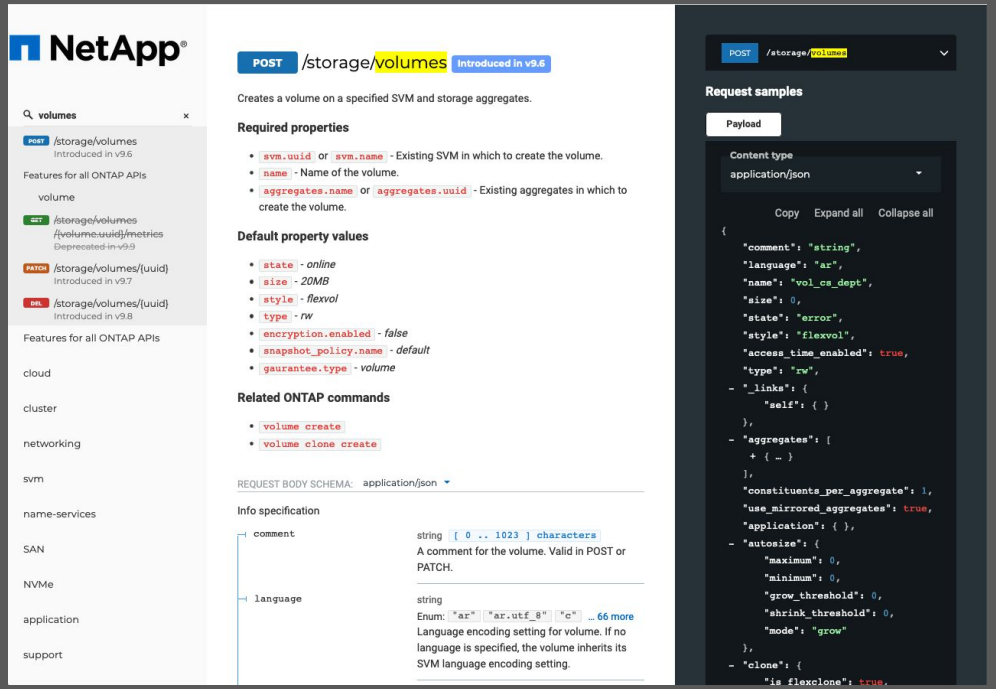
Why Redoc?

Reasoning

- ❏ Built-in search
  - Main feature
- ❏ Three-panel display
  - Much more user friendly
- ❏ Language
  - React, Typescript
  - Best suited for our skillset

Shortcomings

- ❏ Extra work converting from Swagger UI



Challenges

1. Reading through the OpenAPI specification and understanding the items within the documentation

2. Installing the necessary dependencies and getting our local development servers running

3. Learning and putting into practice the elements of an Agile Development Team

Challenges cont...

4 Understanding and analyzing the large codebase from which we would be developing

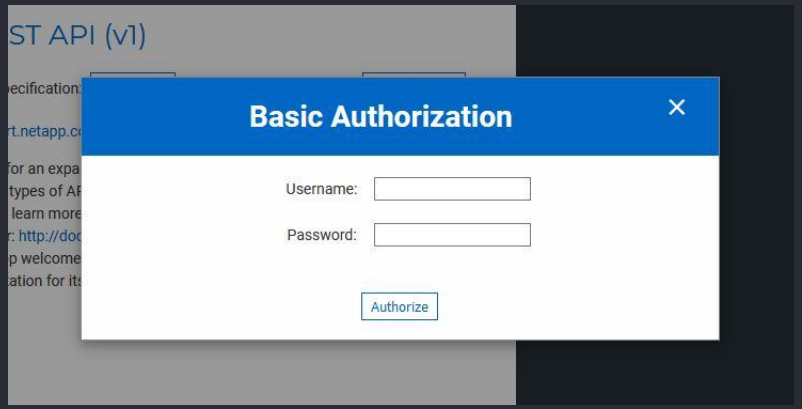
5 Navigating our busy schedules and meeting as a team

6 Preserving features already implemented into SwaggerUI as we transition to Redoc

Features and Functionalities

Front-End Features

Authorization Login



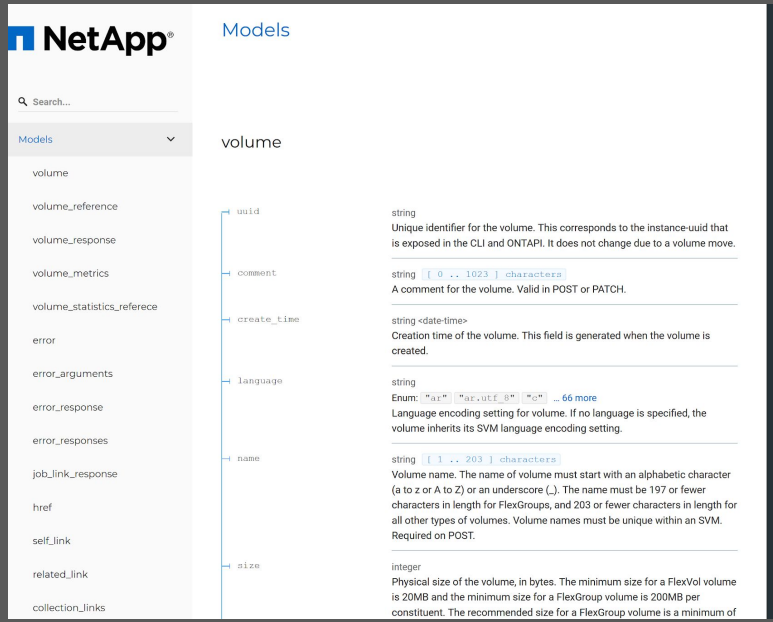
Search Tooltip

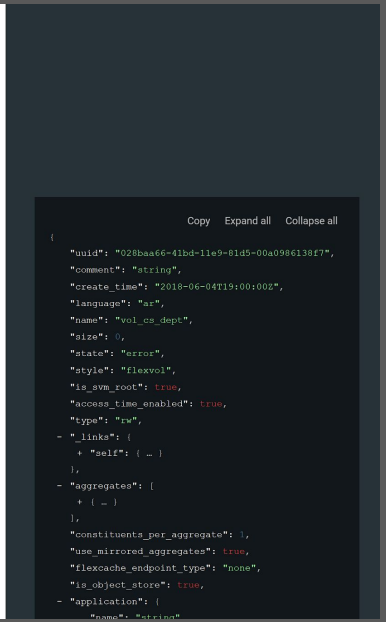
Editing Global Theme

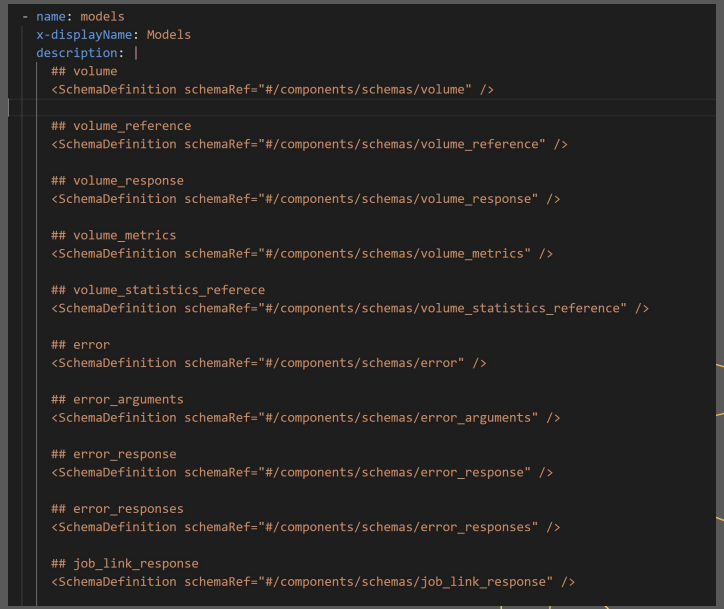
```
main: '#0067C5',  // NetApp Blue
```

Transition Process

Importing the models







Doc Tags - Backend

```ts
export interface OpenAPITag {
  name: string;
  description?: string;
  longDescription?: string;
  externalDocs?: OpenAPIExternalDocumentation;
  'x-displayName'?: string;
}
```

openapi.ts

```ts
export type MenuItemGroupType = 'group' | 'tag' | 'section';

export type MenuItemType = MenuItemGroupType | 'operation';
/** Generic interface for MenuItems */
export interface IMenuItem {
  id: string;
  absoluteIdx?: number;
  name: string;
  description?: string;
  longDescription?: string;
```

MenuStore.ts

```ts
export class GroupModel implements IMenuItem {
  //#region IMenuItem fields
  id: string;
  absoluteIdx?: number;
  name: string;
  description?: string;
  longDescription: string;
  type: MenuItemGroupType;

  items: ContentItemModel[] = [];
```

Group.model.ts

```ts
this.description = tagOrGroup.description || '';
this.longDescription = tagOrGroup['x-ntap-long-description'] || '';
```

Doc Tags - Frontend

```
// longDescription added to this list of constant variables being destructured
const { name, description, longDescription, externalDocs, level } = this.props.item as GroupModel;
```

```
return (
  <>
    <Row>
      <MiddlePanel compact={false}>
        <Header>
          <ShareLink to={this.props.item.id} />
          {name}
        </Header>
      </MiddlePanel>
    </Row>
    <AdvancedMarkdown source={description || ''} htmlWrap={middlePanelWrap} />
    <AdvancedMarkdown source={longDescription || ''} htmlWrap={middlePanelWrap} />
    {externalDocs && (
      <Row>
        <MiddlePanel>
          <ExternalDocumentation externalDocs={externalDocs} />
        </MiddlePanel>
      </Row>
    )}
  </>
);
```

ContentItems.tsx

Search Functionality

- Built with lunr.js
  - Small, full-text search library

Example

A very simple search index can be created using the following:

```
var idx = lunr(function () {
  this.field('title')
  this.field('body')

  this.add({
    "title": "Twelfth-Night",
    "body": "If music be the food of love, play on: Give me excess of it…",
    "author": "William Shakespeare",
    "id": "1"
  })
})
```

Searchable index is built with passed in fields and items

SearchWorker.worker.ts

```
function initEmpty() {
  builder = new lunr.Builder();
  builder.field('title');
  builder.field('description');
  builder.ref('ref');

  builder.pipeline.add(lunr.trimmer, lunr.stopWordFilter, lunr.stemmer);

  index = new Promise(resolve => {
    resolveIndex = resolve;
  });
}

initEmpty();

const expandTerm = term => '*' + lunr.stemmer(new lunr.Token(term, {})) + '*';

export function add<T>(title: string, description: string, meta?: T) {
  const ref = store.push(meta) - 1;
  const item = { title: title.toLowerCase(), description: description.toLowerCase(), ref };
  builder.add(item);
}
```

SearchWorker.worker.ts (modified)

```
function initEmpty() {
  builder = new lunr.Builder();
  builder.field('title');
  builder.field('description');
  builder.field('longDescription');
  builder.ref('ref');

  builder.pipeline.add(lunr.trimmer, lunr.stopWordFilter, lunr.stemmer);

  index = new Promise(resolve => {
    resolveIndex = resolve;
  });
}

initEmpty();

const expandTerm = term => '*' + lunr.stemmer(new lunr.Token(term, {})) + '*';

export function add<T>(title: string, description: string, longDescription: string, meta?: T) {
  const ref = store.push(meta) - 1;
  const item = { title: title.toLowerCase(), description: description.toLowerCase(), longDescription: longDescription.toLowerCase(), ref };
  builder.add(item);
}
```

SearchStore.ts

```
import Worker from './SearchWorker.worker';

function getWorker() {
  let worker: new () => Worker;
  if (IS_BROWSER) {
    try {
      // tslint:disable-next-line
      worker = require('workerize-loader?inline&fallback=false!./SearchWorker.worker');
    } catch (e) {
      worker = require('./SearchWorker.worker').default;
    }
  } else {
    worker = require('./SearchWorker.worker').default;
  }
  return new worker();
}

export class SearchStore<T> {
  searchWorker = getWorker();

  indexItems(groups: Array<IMenuItem | OperationModel>) {
    const recurse = items => {
      items.forEach(group => {
        if (group.type !== 'group') {
          this.add(group.name, group.description, group.longDescription || '', group.id);
        }
        recurse(group.items);
      });
    };

    recurse(groups);
    this.searchWorker.done();
  }

  add(title: string, body: string, longDescription: string, meta?: T) {
    this.searchWorker.add(title, body, longDescription, meta);
  }
```

Initial code changes to add x-ntap-longDescription to search index

Base search in Redoc only goes through section titles and descriptions.

Deep Search Index Builder

```
const recurse = items => {
  items.forEach(group => {
    if(group.type !== 'group') {
      // only operation types have the parameters/responses/etc.
      if(group.type === 'operation') {
        const params: string[][] = this.addParams(group.parameters);
        const req: string[][][] = this.addRequestBody(group.requestBody);
        const resp: string[][][] = this.addResponses(group.responses);
        const objects = req[0].concat(resp[0]);
        const properties = req[1].concat(resp[1]);
        this.add(group.name, group.description || '', group.longDescription || '',
          params[0], params[1], objects, properties, group.httpVerb, group.name, group.id);
      }
      else {
        this.add(group.name, group.description || '', group.longDescription || '',
          [''], [''], [['']], [['']], '', '', group.id);
      }
    }
    recurse(group.items);
  });
};
recurse(groups);
```

```
// function that returns the paths and queries of an operation as an object
addParams(parameters: FieldModel[]): string[][] {
  const paths: string[] = [];
  const queries: string[] = [];
  parameters.forEach(parameter => {
    if(parameter.in === "path") {
      paths.push(parameter.name);
    }
    if(parameter.in === "query") {
      queries.push(parameter.name);
    }
  });
  return [paths, queries];
}
```

SearchStore.ts

```
getDeepFields(field: FieldModel, temp: ReturnObj): ReturnObj | any {
  // if a field has a schema with other fields
  if(field.schema.fields !== undefined) {
    field.schema.fields.forEach(f => {
      temp = this.getDeepFields(f, temp);
    })
    // adds the field's name to the array of objects
    if(field.name !== undefined) {
      temp.objects.push(field.name);
    }
    return temp;
  }
  // if a field's schema doesn't have fields but the field's schema does have items in it
  if(field.schema.items !== undefined && field.schema.items.fields !== undefined) {
    field.schema.items.fields.forEach(f => {
      temp = this.getDeepFields(f, temp);
    });
    // adds the field's name to the array of objects
    if(field.name !== undefined) {
      temp.objects.push(field.name);
    }
    return temp;
  }
  // this is where we would like to add properties
  if(field.name !== undefined) {
    temp.properties.push(field.name);
    return temp;
  }
}
```

```
const regex = /(POST|GET|PATCH|DELETE)|([a-z\/\.\-\{\}]+)|(TITLE|PATH|QUERY|PROPERTY|OBJECT|ENDPOINT)\[(.+?)\]/g;
```

adding a single term to a query

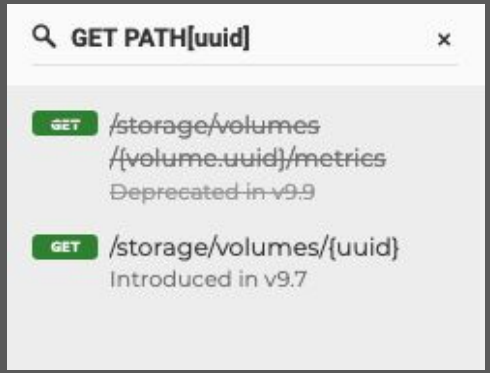
```
query.term("foo")
```

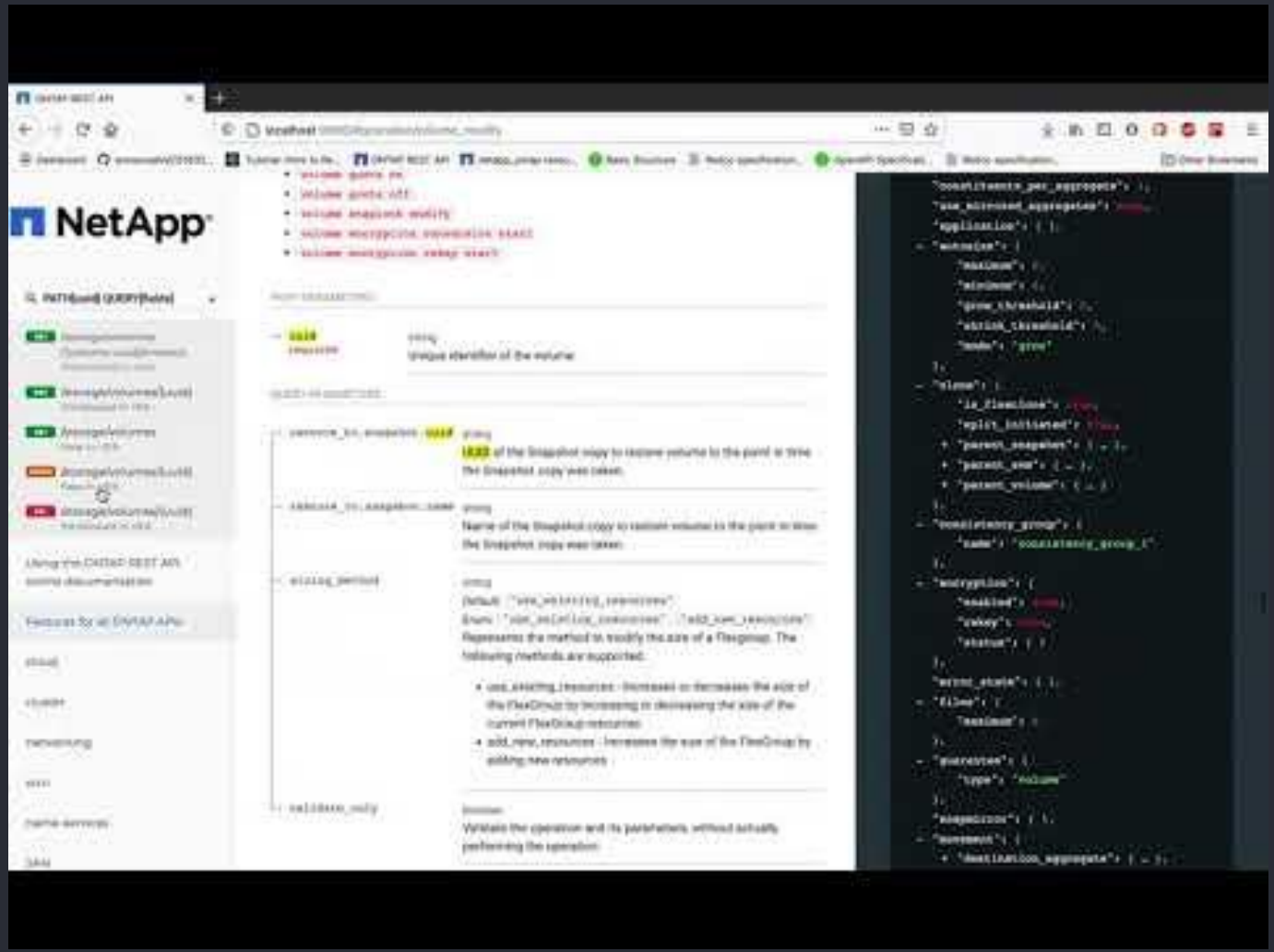
adding a single term to a query and specifying search fields, term boost and automatic trailing wildcard

```
query.term("foo", {
  fields: ["title"],
  boost: 10,
  wildcard: lunr.Query.wildcard.TRAILING
})
```

```
queryObject.term(searchItems[count], {
  fields: [field]
})
```

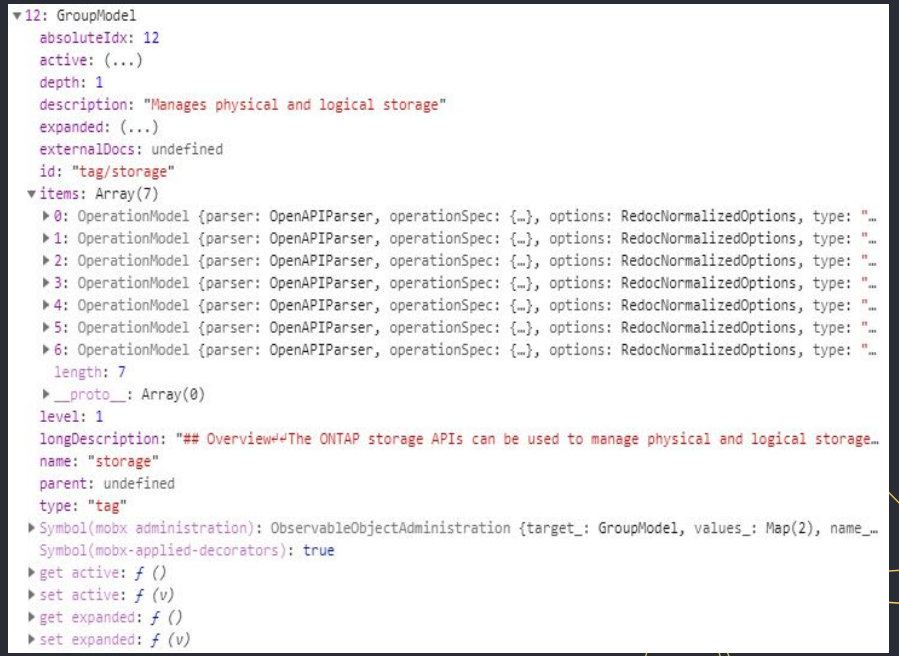
'get' was a word filtered out of lunr by default (caused us some trouble)




NetApp

Testing

- Debug code using Inspect element + console in web browser



Future Goals

- Ensure that search functionality indexes all required sections
  - Parameters, object definitions, responses
  - Maximize search performance

- Customize styling
  - Make it more user friendly
  - Fit it to the NetApp color scheme
  - Investigate built in Redoc theming options

- Automate Version Tracking

Thank you

Special thanks to Anuradha Kulkarni, Sami Benbourenane, and Brian Kinkade for the guidance they have given us so far, and we are excited to continue finishing this project with your help.

Questions?