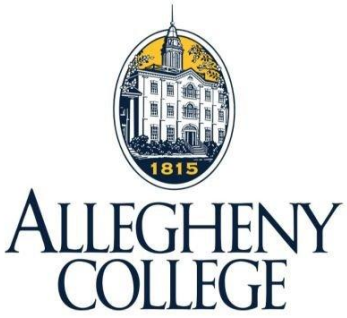
Enhancing User Experience in Rest API Visualization

Midterm Presentation

Noor Buchi, Teona Bagashvili,
Christian Lussier, Kobe Coleman



Team Introduction

Teona Bagashvili

- About Me:
  - Allegheny College Junior, Computer Science Major & Dance and Movements Minor

- Why I chose this Project:
  - Get industry experience in software engineering
  - Improve my web development skills

Christian Lussier

- About Me:
  - Allegheny College Senior, Computer Science Major & Economics Minor

- Why I chose this Project:
  - I wanted to gain industry experience and apply what I learned in the classroom
  - Wanted to become more skilled in developing with REACT and JavaScript

Kobe Coleman

- About Me:
  - Allegheny College Sophomore, Major in Computer Science and Minor in Music Theory

- Why I chose this Project:
  - Wanted to go deeper into React applications and how the different components in a visualizer interacted with each other
  - Have hands on experience with open source projects

Noor Buchi

- About Me:
  - Allegheny College Junior, Computer Science Major, Political Science Minor

- Why I chose this Project:
  - Get hands on experience in software engineering using industry standards
  - Collaborate with a team and develop new skills

Outline

- Motivation behind this project
- Team organization
  - Communication
  - Experience
- Preliminary work and research
- Approach and implementation
- Demo
- Challenges
- Future Work
- Questions

Project Background

Main Goal: To enhance the user experience in NetApp's REST API visualization.

- *For the Midterm:* Add deep search functionality
- *For the Final:* Finalize deep search and implement API version control tracking

Existing issues with current tool:

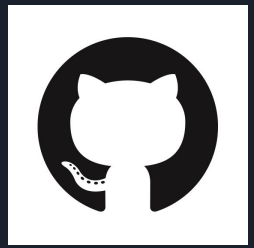
- Swagger UI is not user friendly, users must scroll through a long list of APIs and manually expand them to find specific information
- No way to filter or search for a specific endpoint/model/parameter
- No automated version tracking -- could lead to dependency issues

Possible Solutions:

- Use a tool other than Swagger UI to generate a more user friendly visualization
- Add features to Swagger UI that would solve these difficulties

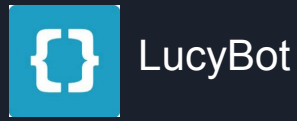
Team Organization

- Slack, Zoom, and Google Meets for constant communication
- Git workflow using Github and it's features (project board, pull requests)
- AGILE development practices -- one weekly sprint meeting with the team from NetApp (Sami and Anuradha)
  - Discussed our progress through each week
  - Created project board stories
  - Identified completed and remaining tasks
  - Solved technical difficulties on setting up a Javascript environment and working with tools such as React



Preliminary Research

- Familiarize ourselves with Swagger UI
- Identified and assessed alternative API visualizers :
  - **ReDoc.ly:** Deep search, version tracking, free, open source, visually appealing
  - **ReadMe.io:** Shallow search, version tracking, paid, limited modifiability, very visually appealing
  - *Postman*
  - *LucyBot*
  - *DapperDox*
- Looked into YAML configuration file
- Investigated various pull requests and issues that looked to implement the features we were looking for
- Documented findings from research in a GitHub repository



Swagger UI

- An open source API visualizer built on HTML, JS, CSS, and REACT that automatically visualizes documentation from Open API specifications
- Currently being used by NetApp to visualize RESTful API's
- A free & open source project with a large, active community of developers
    - Easily customizable!
- Dependency free while offering a try-it-out feature and expandable methods
- Missing the needed deep search and automated version tracking features

Choosing a Suitable Documentation Visualizer

- We were looking for the best tradeoff:
  - Wanted to keep features in Swagger already implemented by NetApp
  - Provide NetApp with an easy transition to a new or updated visualizer
  - Be able to create custom layouts and features, or be limited in modification abilities?
  - Wanted a maintained, free, and open source tool
    - Active open source community
    - Extensive documentation

Swagger UI

ONTAP REST API v1

[Base URL: /api]

ONTAP 9.6 adds support for an expansive RESTful API. The documentation below provides information about the types of API calls available to you, as well as details about using each API endpoint. You can learn more about the ONTAP REST API and ONTAP in the ONTAP 9 Documentation Center: http://docs.netapp.com/ontap-9/topic/com.netapp.doc.dot-rest-api/home.html. NetApp welcomes your comments and suggestions about the ONTAP REST API and the documentation for its use.

Using the ONTAP 9.6 REST API online documentation
Each API method includes usage examples, as well as a model that displays all the required and optional properties supported by the method. Click the *Model* link, available with each API method, to see all the required and optional properties supported by each method.

NetApp Support - Website

Schemes
HTTPS

Authorize

docs Details the features for all ONTAP APIs.

DOC **Getting started with the ONTAP REST API**

Overview

Let's review some key things about RESTful APIs and how they're implemented in ONTAP:

- REST API URLs identify the resources that you'll be working with, including clusters, SVMs, and storage.
- REST APIs use HTTP methods GET, POST, PATCH, DELETE, and OPTIONS to indicate their actions.
- REST APIs return common HTTP status codes to indicate the results of each call. Additional error details can be included in the results body.
- REST APIs request and response bodies are encoded using JSON.
- REST APIs support hyperlinking among resources using the Content-Type "application/hal+json".
- GET calls on collections usually return only name and UUID by default. If you want to retrieve additional properties, you need to specify them using the "fields" query parameter.
- ONTAP supports query-based DELETE or PATCH for all collection endpoints.

If you're already familiar with the ONTAPI API (also known as ZAPI), there are some similarities between ONTAP REST APIs and ONTAPI. For example:

- Both support the same transport and security mechanisms.
- Both paginate results based on either number of seconds or number of records.
- Both support filtering the returned records based on property values.
- Both support limiting the returned properties.

cloud Manages cloud (object storage) targets

DOC `/cloud/targets`

GET `/cloud/targets`

POST `/cloud/targets`

Creates a cloud target.

Required properties

- `name` - Name for the cloud target.
- `owner` - Owner of the target: *fabricpool*, *snapmirror*.
- `provider_type` - Type of cloud provider: *AWS_S3*, *Azure_Cloud*, *SGWS*, *IBM_COS*, *AliCloud*, *GoogleCloud*.
- `server` - Fully qualified domain name of the object store server. Required when `provider_type` is one of the following: *SGWS*, *IBM_COS*, *AliCloud*.
- `container` - Data bucket/container name.
- `access_key` - Access key ID if `provider_type` is not *Azure_Cloud* and `authentication_type` is *key*.
- `secret_password` - Secret access key if `provider_type` is not *Azure_Cloud* and `authentication_type` is *key*.
- `azure_account` - Azure account if `provider_type` is *Azure_Cloud*.
- `azure_private_key` - Azure access key if `provider_type` is *Azure_Cloud*.
- `cap_url` - Full URL of the request to a CAP server for retrieving temporary credentials if `authentication_type` is *cap*.
- `svm.name` or `svm.uuid` - Name or UUID of SVM if `owner` is *snapmirror*.
- `snapmirror_use` - Use of the cloud target if `owner` is *snapmirror*: data, metadata.

Recommended optional properties

- `authentication_type` - Authentication used to access the target: *key*, *cap*, *ec2_iam*.
- `ssl_enabled` - SSL/HTTPS enabled or disabled.
- `port` - Port number of the object store that ONTAP uses when establishing a connection.
- `ipspace` - IPspace to use in order to reach the cloud target.

Default property values

- `authentication_type`
  - *ec2_iam* - if running in Cloud Volumes ONTAP in AWS
  - *key* - in all other cases.
- `server`
  - *s3.amazonaws.com* - if `provider_type` is *AWS_S3*
  - *blob.core.windows.net* - if `provider_type` is *Azure_Cloud*
  - *storage.googleapis.com* - if `provider_type` is *GoogleCloud*
- `ssl_enabled` - *true*
- `port`
  - *443* if `ssl_enabled` is *true* and `provider_type` is not *SGWS*
  - *8082* if `ssl_enabled` is *true* and `provider_type` is *SGWS*
  - *80* if `ssl_enabled` is *false* and `provider_type` is not *SGWS*
  - *8084* if `ssl_enabled` is *false* and `provider_type` is *SGWS*
- `ipspace` - *Default*
- `certificate_validation_enabled` - *true*
- `ignore_warnings` - *false*
- `check_only` - *false*

Related ONTAP commands

- `storage aggregate object-store config create`

Work Completed

- Implemented a preliminary version of a deep-search feature in the current NetApp Swagger user interface
  - Builds upon the existing filter feature to detect matches throughout each endpoint's description and operations, not just the tag
    - Adds a properly formatted and visually improved search bar
  - Algorithm ranks search results based on the number and location of matches, finding the most relevant content
- Took inspiration from existing pull requests and issues on official Swagger UI Github repository

Front-End Improvements

_layout.scss

```
.filter
{
    .filter-icon {
       float: left;
       font-size: 275%;
       padding: 10px 30px;
    }
    .operation-filter-input
    {
        width: 90%;
        margin: 2px 0;
        padding: 21px 10px;
        border: 2px solid $operational-filter-input-border-color;
    }

    padding-bottom: 20px;
}
```

filter.jsx

```
    return (
      <div>
        {filter === null || filter === false ? null :
          <div className="filter-container">
            <Col className="filter wrapper" mobile={12}>
            <div className="filter-icon">🔍</div>
              <input className="operation-filter-input"
              placeholder="Enter your search query here..."
              type="text"
                     onChange={this.onFilterChange} value={filter
                     === true || filter === "true" ? "" : filter}
                     disabled={isLoading} style={inputStyle}/>
            </Col>
          </div>
        }
      </div>
    )
  }
}
```

src/style/layouts/layout.scss

Add styling that formats the search bar on the web page along with its accompanying icon.

src/core/containers/filter.jsx

Adds a search bar icon to the original filter feature's barebones search bar.

Enter your search query here...

cloud Manages cloud (object storage) targets

cluster Manages clusters, nodes, jobs, and cluster software

DOC `cluster`

networking Manages physical and logical networking

DOC `networking`

svm Manages storage virtual machines (SVMs)

DOC `svm`

name-services Manages SVM name services such as LDAP, NIS, and DNS

DOC `name-services`

SAN Manages resources for storage area networks (SAN)

DOC `SAN`

NVMe Manages resources for Non-Volatile Memory Express (NVMe)

Deep Search Algorithm

- Enhanced the *opsFilter* function to perform a deep search by searching for matches of a query in the:
  - Path
  - Description
  - Operations
- Weight-based ranking of operations
  - Tags take precedence over paths and descriptions
- Displays the results with matching words from the search query, while hiding all irrelevant APIs until the search bar is emptied

The taggedOps Object

*Shallow Search ----> **Deep Search using taggedOps***


```
"storage": {
    "tagDetails": {
        "name": "storage",
        "description": "Manages physical and logical storage",
        "x-ntap-long-description": "## Overview\n\nThe ONTAP storage APIs can
        be used to manage physical and logical storage. This includes
        management of aggregates, volumes, LUNs, qtrees, snapshots, quotas,
        and storage effeciency.\n<br/>\n"
    },
    "operations": [{
        "path": "/storage/volumes",
        "method": "x-ntap-long-description",
        "operation": {
            "tags": ["storage"],
            "description": "## Overview\nFlexVol volumes are logical
            containers used by ONTAP to serve data to clients.  They contain
            file systems in a NAS environment and LUNs in a SAN
            environment.<br/>\nA FlexGroup volume is a scale-out NAS
            container that provides high performance along with automatic
            load distribution and scalability. A FlexGroup volume contains
            several constituents that automatically and transparently share
            the traffic.</br>\nFlexClone volumes are writable, point-in-time
```


taggedOps Map

The object storing the API content from the YAML file. *How do we parse this?*

Implemented Deep Search Algorithm

opsFilter.js

opsFilter.js

```
14    let filteredOps = value.get("operations"); // get the operations from the filteredOps
15
16    if (filteredOps.size !== 0) {
17      for (let i = 0; i < filteredOps.size; i++) {
18        let op = filteredOps.get(i); // get the current operation from filteredOps
19        // set the base value of variable that will hold the ranking score of a operation
20        let opWeight = 0;
21
22        let pathMatches = op.get("path").match(re); // list of matches in path key (eight )
23        // list of matches in ["operation", "description"] key
24        let descMatches = op.getIn(["operation", "description"]).match(re);
25
26        // update opWeight per the matches found:
27        if (pathMatches) {
28          opWeight += pathMatches.length * 100; // opWeight of path match = 100
29        }
30        if (descMatches) {
31          opWeight += descMatches.length; // opWeight of description match = 1
32        }
33
34        // see if the current operation contained any matches:
35        if (opWeight === 0) {
36          filteredOps = filteredOps.delete(i); // remove the operation with zero matches
37        } else {
38          // add the opWeight key to the operation info
39          filteredOps = filteredOps.set(i, op.set("opWeight", opWeight));
40          tagWeight += opWeight; // update the overall tag weight to reflect the operation within it's weight
41        }
42      }
```

Find matches in the description & paths of operations

opsFilter.js

```
44        // sort the tag's operations by weight:
45        filteredOps = filteredOps.sort(function(value1, value2) {
46          if (value1.get("opWeight") > value2.get("opWeight")) {
47            return -1;
48          }
49          if (value1.get("opWeight") < value2.get("opWeight")) {
50            return 1;
51          }
52          if (value1.get("opWeight") === value2.get("opWeight")) {
53            return 0;
54          }
55        });
56      }
57
58      // Set filtered operations to respective tag
59      taggedOps = taggedOps.setIn([key.toString(), "operations"], filteredOps);
```

Rank the operations containing matches.

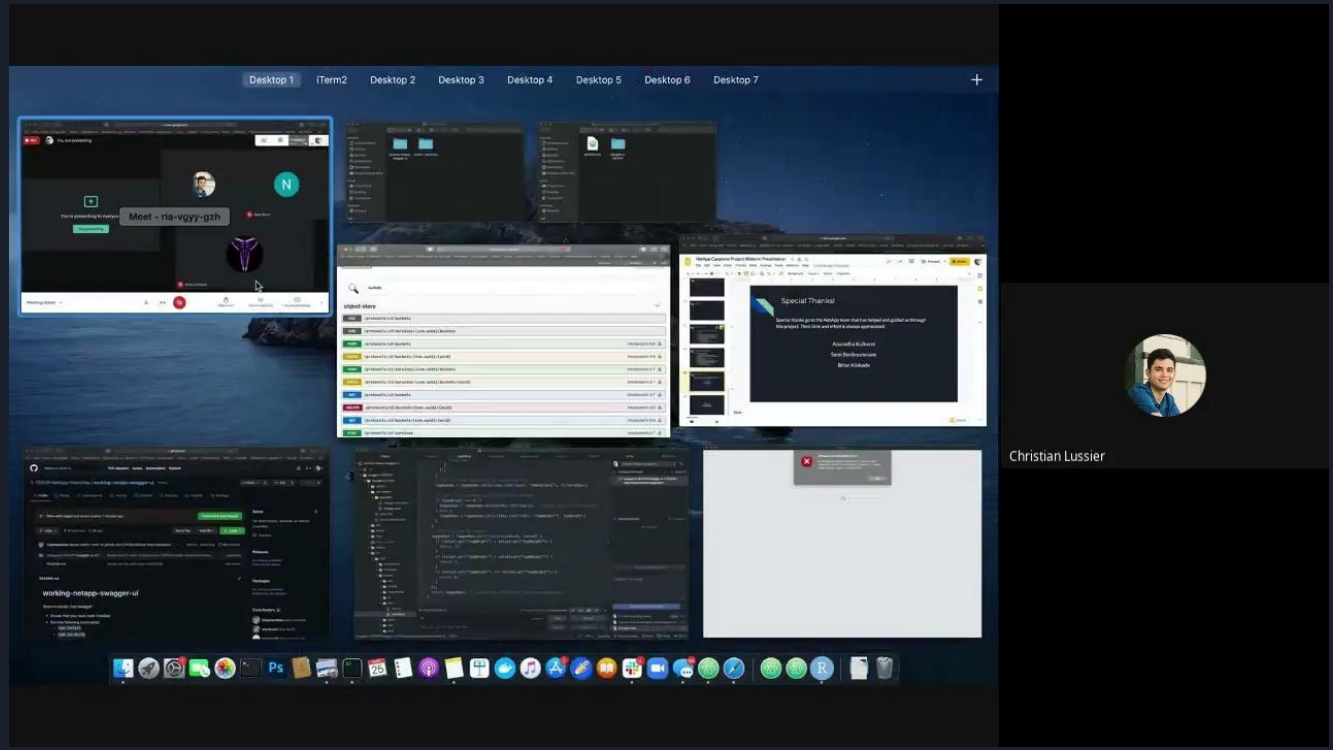
Implemented Deep Search Algorithm

```
export default function(taggedOps, phrase) {
  // create a reg expression using the phrase (case insensitive & matches all occurrences)
  let re = new RegExp(phrase, "ig");
  // iterate through key and value pairs within the taggedOps object:
  for (let [key, value] of taggedOps) {
    let tagWeight = 0; // used to track weight of the tag (big category)
    let keyMatches = key.toString().match(re); // find number of matches in the tag

    // if the tag matches the search phrase give it a higher weight:
    if (keyMatches) {
      tagWeight += 1000;
    }

    let filteredOps = value.get("operations"); // get the operations from the filteredOps
```

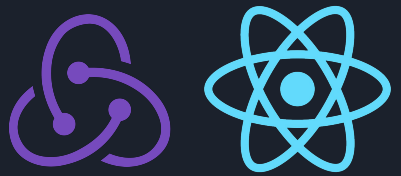
```
    // sort the tags by weight:
    taggedOps = taggedOps.sort(function(value1, value2) {
      if (value1.get("tagWeight") > value2.get("tagWeight")) {
        return -1;
      }
      if (value1.get("tagWeight") < value2.get("tagWeight")) {
        return 1;
      }
      if (value1.get("tagWeight") === value2.get("tagWeight")) {
        return 0;
      }
    });
    return taggedOps; // return the sorted tags and their operations
```

Find & rank the tags best matching the search phrase.

Demo


Desktop 1
iTerm2
Desktop 2
Desktop 3
Desktop 4
Desktop 5
Desktop 6
Desktop 7
Meet - ria-vgyy-gzh
Special Thanks!
working-netapp-swagger-ui
Christian Lussier

Challenges Faced



- Working with new tools/languages/frameworks with little previous experience
  - Javascript
  - React
  - Redux
- Understanding the existing structure in the NetApp Swagger repository
  - Learning about REACT & other code components -- code tracing
  - How/where could we add the deep search functionality without breaking other features?
- Implementing a deep search algorithm
  - Parsing deeper in the YAML file
  - Determining how to best rank search results

Our Goals Moving Forward

- Enhance the Deep Search feature:
  - Improve the ranking of search query results
  - Improve the efficiency of the deep search feature
  - Searching for multiple words/phrases at once
  - Give the user the ability to narrow down the search based on key search terms
- Add other required features:
  - Automated API version tracking and a changelog
  - Auto-expanding endpoints when searching or loading the page
- If time allots: other additions to improve user experience
  - New layout that makes use of the current empty space
    - Multiple columns/panes
    - Similar to other visualization tools

Special Thanks!

Special thanks go to the NetApp team that has helped and guided us through this project. Their time and effort is always appreciated.

Anuradha Kulkarni

Sami Benbourenane

Brian Kinkade

Thank You!
Any Questions?