Class 20: REST API Filtering & Text Search

- Filtering Resources with Query Parameters
- Text Search with MongoDB
- Sub-Resources in REST APIs

Review

Review: REST API

Representtation **S**tate **T**ransfer API is a type of API that adheres to the constraints of REST architecture to support communication between client and server to exchange resources, like data and media.

A **RESTful API** supports the **creation**, **retrieval**, **updating**, and **deletion** of resources (CRUD operations) using HTTP methods.

Review: REST Design Principles

- ALL **CRUD** operations are defined by **HTTP methods**:
  `POST`, `GET`, `PUT`, `PATCH`, and `DELETE`.
- All **resources** are accessed using **URI endpoints** that represent a **collection of resources** or a **specific resource**.
  `GET /api/documents` and `GET /api/documents/:id`
- Every HTTP request to the API is **independent** of all other requests (i.e. stateless).
  (The request contains all the information needed to process the request.)

RESTful API Considerations

Why Use a RESTful API?

- Limit access to your database.
- Provide a consistent and familiar interface for clients to interact with your data.
- Allow for separation of concerns between client and server.
- Enable scalability and flexibility in your application architecture.

Caution! Security Concerns

Our current RESTful API design is **insecure** because it allows any client to access and manipulate our database without any restrictions.

Implement authentication and authorization mechanisms to control who can access your API and what actions they can perform. (More about this next class.)

REST API Filtering

Activity: URL Filtering Exploration

Working with your peer (2-4), explore how the URL changes when searching (i.e. filtering) DuckDuckGo for different queries.

Search:

- cats
- dogs
- cats and dogs

How does the URL change when you search for different queries?

Filtering Resources

When retrieving a **collection** of resources, clients may want to filter the results based on certain criteria. This can be achieved using **query parameters** in the URI endpoint **and** a **MongoDB query**.

Review: MongoDB Query

```
db.collection('documents').find({
  field: value,
  field: { $operator: value },
  field: { $operator: [value1, value2, ...] },
  ...
});
```

Activity: MongoDB Query Practice

Working with your peer (2-4), write a MongoDB query to find all **vegetables** from the **Mexico** in the **produce** collection on your **handout**.

```
db.collection('documents').find({
  field: value,
  field: { $operator: value },
  field: { $operator: [value1, value2, ...] },
  ...
});
```

Then, update the `GET /api/produce` endpoint to filter all produce by **vegetables** from the **Mexico** in your Codespace.

HTTP Query Parameters

A query parameter is a key-value pair that is appended to the end of a URI endpoint to provide additional information about the request.

The query parameters are separated from the URI endpoint by a `?` and multiple query parameters are separated by `&`.

Example: (2 parameters: `q` and `sort`)

```
GET /api/documents?q=cats&sort=asc
```

Activity: API Query Parameters

Working with your peers (2-4), design query parameters to filter the **produce** collection by **country** and by **category**. (Complete items 2 and 3 on your handout.)

```
GET /api/documents?q=cats&sort=asc
```

Express.js: Query Parameters

Use the `req.query` object to access query parameters in an Express.js route handler.

```
app.get('/api/documents', (req, res) => {
  const qParam = req.query.q;
  const sortParam = req.query.sort;
})
```

Activity: Implementing API Query Parameters

Working with your peers (2-4), implement query parameters to filter the **produce** collection by **country** and by **category** in your Codespace.

```
app.get('/api/documents', (req, res) => {
  const qParam = req.query.q;
  const sortParam = req.query.sort;
})
```

Gotcha: Query Parameters are *Optional*

If no query parameters are provided, then the response should include all resources in the collection.

Not all query parameters should be required. For example, if a client wants to filter by **country** but not by **category**, then the API should still return the correct results.

Activity: Optional Query Parameters

Working with your peers (2-4), **test** the produce collection to ensure it returns all produce if no query parameters are provided, and that it returns the correct results if only one query parameter is provided.

Update the produce collection endpoint if necessary to ensure that query parameters are optional.

Query Parameter Design

- Parameters should be short and concise, but also descriptive.
- As is the case with URLs, query parameters should also be all lowercase.
- If case sensitivity is not meaningful for the parameter's value, then the API should treat the parameter as case-insensitive (e.g. `?category=fruits` and `?category=Fruits`).
- Single word parameters are preferred. If using multi-word parameters, use hyphens to separate words (e.g. `sort-order`).
- Typically, the order of the parameters in the URI should not matter. (If it does, your API's documentation should clearly specify the expected order of parameters.)

Text Search

Text Search

DuckDuckGo doesn't "filter" by category or country, it performs a **text search** to find relevant documents.

A text search is a search that matches a query string against the text content of documents in a collection. Often, a text search employs *stemming* to match different forms of a word (e.g. "run", "running", "ran").

MongoDB: Text Search

Document databases, like MongoDB, typically support text search queries to allow clients to perform text search on string fields in the documents.

1. Create a text index on the fields you want to search.

   An index support efficient execution of queries. A text index allows you to perform text search queries on the indexed fields.

2. Query the collection using the text index.

MongoDB: Text Index

A MongoDB collection may have **exactly one** text index, which can include one or more fields.

```
db.COLLECTION.createIndex({
  FIELD1: "text",
  FIELD2: "text",
  ...
});
```

A string fields included in the index are searchable using the index.

Activity: Text Index

Working with your peers (2-4), create a text index for the `produce` collection.

```
db.COLLECTION.createIndex({
  FIELD1: "text",
  FIELD2: "text",
  ...
});
```

Only include the text fields that you want to be searchable using the text index.

MongoDB: Text Search Query

To perform a text search query, use the `$text` operator in the query filter.

```
db.collection('documents').find({
  $text: { $search: "SEARCH TERMS" }
});
```

Activity: Text Search Query

Working with your peers (2-4), write a MongoDB query to search the `produce` collection for all documents that are “sweet” on your **handout**.

```
db.collection('documents').find({
  $text: { $search: "SEARCH TERMS" }
});
```

Test your query in MongoDB Shell.

Search API Design

It's common design pattern to use the `q` query parameter for text search queries.

```
GET /api/documents?q=SEARCH+TERMS
```

This parameter can be combined with other query parameters to further filter the results.

```
GET /api/documents?q=SEARCH+TERMS&category=fruits
```

REST API Sub-Resources

Review: Hierarchical Resources

In some cases, resources may have a hierarchical relationship, such as a **document** resource that belongs to a **collection** resource. In this case, the URI for the resource endpoint should reflect this hierarchy.

Example:

- `GET /api/posts`
- `GET /api/posts/:collectionId/comments`
- `GET /api/posts/:collectionId/comments/:commentId`

REST API Design: Sub-Resources

A sub-resource is a resource that is nested within another resource. Typically, in a document database, a sub-resource is simply a field in a document.

Example: API Sub-Resource

`comments` as a sub-resource of `posts`:

```
{
  title: "Post Title",
  content: "Post Content",
  comments: [
    {
      author: "Comment Author",
      text: "Comment Text"
    },
    ...
  ]
}
```

Insert a comment:

```
POST /api/posts/:postId/comments
```

Delete a comment:

```
DELETE /api/posts/:postId/comments/:commentId
```

Sub-Resource Not As a Collection

Some sub-resources are not collections of resources, but rather a single resource that you want to access or manipulate.

```
{
  title: "Post Title",
  content: "Post Content",
  featured: false
}
```

To feature a post:

```
POST /api/posts/:postId/featured
```

To unfeature a post:

```
DELETE /api/posts/:postId/featured
```

Often we use `POST` (without request body) to affirmatively modify the sub-resource and `DELETE` to *undo* that change.

Discussion: Singular Sub-Resource

Why might we implement a sub-resource instead of simply let the client update the field using the `PATCH /api/posts/:postId` endpoint?

```
{
  title: "Post Title",
  content: "Post Content",
  featured: false
}
```

To feature a post:

```
POST /api/posts/:postId/featured
```

To unfeature a post:

```
DELETE /api/posts/:postId/featured
```

Summary

Summary

- REST APIs often support filtering of resources using query parameters.
- Text search is a common type of filtering that matches a query string against the text content of documents in a collection.
- Sub-resources are resources that are nested within another resource and can be accessed or manipulated using their own URI endpoints.

What's Next

Today:

- Homework 3 Due
- Project 2, Milestone 3 Assigned

Friday: Practice Problem Workshop

Next Week: Exam II