Lecture 7: API Development and Intermediate MongoDB

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Intermediate MongoDB

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Demonstration

In Lecture 7's repository, see *advanced_mongo.js* for examples. In this file, a module is exported detailing many of the functions listed.

I would recommend running node in the command line, requiring *advanced_mongo.js*, and experimenting with it accordingly. Or, you can write your own driver to experiment.

Note: the data for this collection will rebuild itself every time you require the file, and for simplicity's sake the id's are being stored as integers. At the end of every function, the changes will be logged. Feel free to change this!

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Advanced Finding

We can find documents many more ways than just matching on multiple fields:

- Query by subdocuments
- Query for matches inside an array
- Query for a field to be one of *many* values
- Matching fields that are less than (or equal to) a value
- Matching fields that are greater than (or equal to) a value
- Performing a logical query for all matching queries, or any matching queries
- **JavaScript based querying!**

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Advanced Updating

There are many ways we can update documents, rather than just replacing their entire content.

- We can change only specific fields
- Update subdocuments
- Increment fields
- Multiply fields value
- Remove fields
- Update to a minimum value
- Update to a maximum value
- Manipulate arrays

All of these are demonstrated in *advanced_mongo.js,* where you can experiment with them accordingly through the node command line or writing your own file.

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Array Querying Operation

Naturally, as JSON documents, we can store arrays in MongoDB.

- Entries can be primitives or objects!

We can query documents based on arrays and update arrays and their entries. When dealing with arrays containing subdocuments, we can query for matching fields on subdocuments.

We can query arrays to find documents that have arrays with matching entries

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Array Manipulation Operations

Arguably, the most difficult part of MongoDB is array manipulation due to the complex syntax of combining arrays and subdocuments.

There are many ways of updating arrays:

- Adding to the array if it does not already exist
- Adding to the array whether or not it exists
- Popping the first or last element
- Remove a single matching element
- Removing all matching elements

©2018 STEVENS INSTITUTE OF TECHNOLOGY

POST, PUT, DELETE (API)

©2018 STEVENS INSTITUTE OF TECHNOLOGY

POST, PUT, and DELETE

Last lecture, we focused on GET requests. GET requests are used to retrieve data, and do not have access to request bodies.

POST, PUT, and DELETE calls are used for other actions.

- POST requests call for the creation of an entity
- PUT requests call for an entity to be updated
- DELETE requests call for an entity to be deleted

Each of these request types can use the following types of data:

- Querystring parameters
- Request bodies
- URL Params
- Headers

©2018 STEVENS INSTITUTE OF TECHNOLOGY

A request body

POST, PUT, and DELETE requests can all provide data in a request body.

A request body is a series of bytes transmitted below the headers of an HTTP Request.

We will be submitting a request body in two ways:

- Text that is in a JSON format (modern format of submitting data)
- Text that is in a form data format (traditionally how browsers POST)

The request body will be interpreted by our server using the body-parser middleware

- https://github.com/expressjs/body-parser

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Using request body data

In order to access request body data, we must first apply the *body-parser* middleware.

We will be having our express app use the JSON body-parser middleware

- https://github.com/expressjs/body-parser#expressconnect-top-level-generic

This will allow us to add text that is formatted as JSON to a request body, and to have our server parse the JSON and place the object in the *request.body* property.

This will allow us to submit data with our POST, PUT, and DELETE calls and begin interacting with our server.

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Using Postman

As we use more methods, such as POST, PUT, and DELETE, it becomes increasingly difficult to test using just your browser, particularly because you cannot directly PUT and DELETE from the browser!

You can use a REST client such as Postman and PAW to test your API calls.

- https://www.getpostman.com/
- https://luckymarmot.com/paw

A REST client is a program that will allow you to easily configure and make HTTP Calls to your servers.

©2018 STEVENS INSTITUTE OF TECHNOLOGY

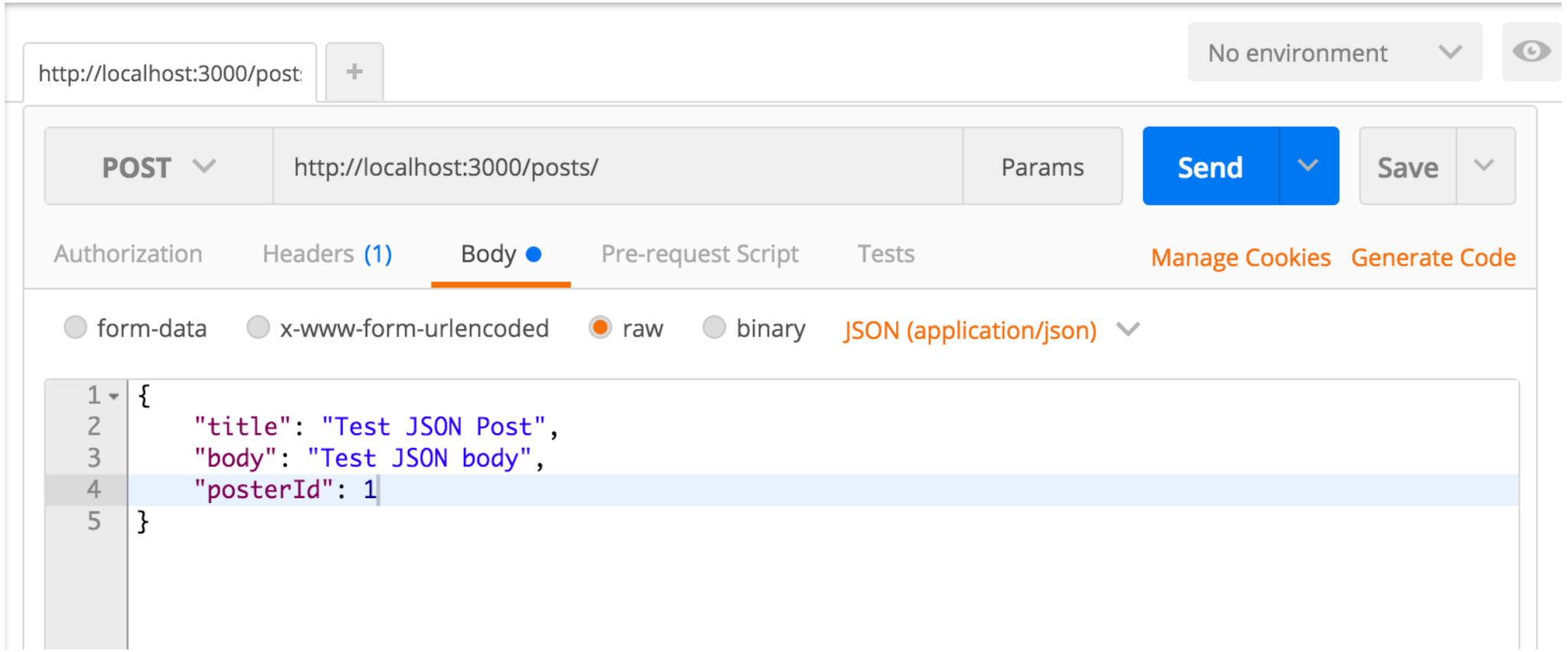
Using Postman to send JSON

In order to use Postman, you need:

- The URL you wish to submit data to
- The request method you wish to use
- Body data
  - You must set the body type to **raw**
  - You must also set the type to **JSON (application/json)**

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Adding a blog post with Postman



©2018 STEVENS INSTITUTE OF TECHNOLOGY

Using that post on the server

We can then use data posted on the server by accessing the *req.body* property.

```
24  router.post("/", (req, res) => {
25      let blogPostData = req.body;
26
27      postData.addPost(blogPostData.title, blogPostData.body, blogPostData.posterId)
28          .then(() => {
29              res.sendStatus(200);
30          }, () => {
31              res.sendStatus(500);
32          });
33  });
34
```

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Updating data

We use the PUT verb to update data. URLs to update object often include its identifier.

That means to update a blog post with an id of *3* you would PUT to http://localhost:3000/blog/3

Your request body would contain the new version of the blog post.

```
1  router.put("/:id", (req, res) => {
2      let updatedData = req.body;
3
4      postData.updatePost(req.params.id, updat
5          .then(() => {
6              res.sendStatus(200);
7          }, () => {
8              res.sendStatus(500);
9          });
0  });
```

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Deleting data

Informing your server that you want to delete an entity is extremely easy. Much like PUT, you would send a DELETE call to a URL that contains the identifier.

That means to delete a blog post with an id of *3* you would DELETE to http://localhost:3000/blog/3

```
52  router.delete("/:id", (req, res) => {
53      postData.removePost(req.params.id)
54          .then(() => {
55              res.sendStatus(200);
56          }, () => {
57              res.sendStatus(500);
58          });
59  });
60
```

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Server Side Error Checking

©2018 STEVENS INSTITUTE OF TECHNOLOGY

What is server side validation?

Users will submit errors; it's a fact of life that as a web developer, you will encounter situations where an error is submitted.

There are many types of errors that can occur:

- The user tries to request a resource that does not exist
- The user inputs data that does not make sense (bad arguments / parameters / querystring data)
- The user is not authenticated
- The input the user provides does not make sense
- The user is attempting to access resources they do not have access to

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Server side error checking

Whenever input comes from a user, you must check that this input is:

- Actually there!
- Actually the type you want!
  - For example, you may have to change from strings to numbers
- Actually valid!
  - When you write a calculator that you wouldn't let someone divide by 0

There are two places you will need to perform error checking:

- Inside of your routes; this will easily catch user submitted errors
- Inside of your data modules; this will allow you to ensure that you don't create bad data.

©2018 STEVENS INSTITUTE OF TECHNOLOGY

Error handling in an API

While we build out these APIs, error handling is extremely easy!

When you encounter an issue in your API routes, you will:

- Determine what type of error it is (ie, the user is requesting an object that does not exist) and respond with the proper status code.
- In addition to the failed status code, also send back a JSON object that describes what happened. It can be as simple as having a property called *errorMessage* with a string describing the error, or an array of all the errors!

©2018 STEVENS INSTITUTE OF TECHNOLOGY