Introduction to

Hands-On Workshop

Part 2 - Stitch

Overview

In Part 1 of this workshop you've set the foundation by creating a MongoDB cluster and loading some data. Now it's time to put that data to action. In part 2 of this workshop we'll create microservices to expose the data via REST APIs and create a basic front-end application that leverages those APIs.

Specifically, we'll create APIs to query movies. Then we'll host all of this on MongoDB Stitch!

Hands-on Lab

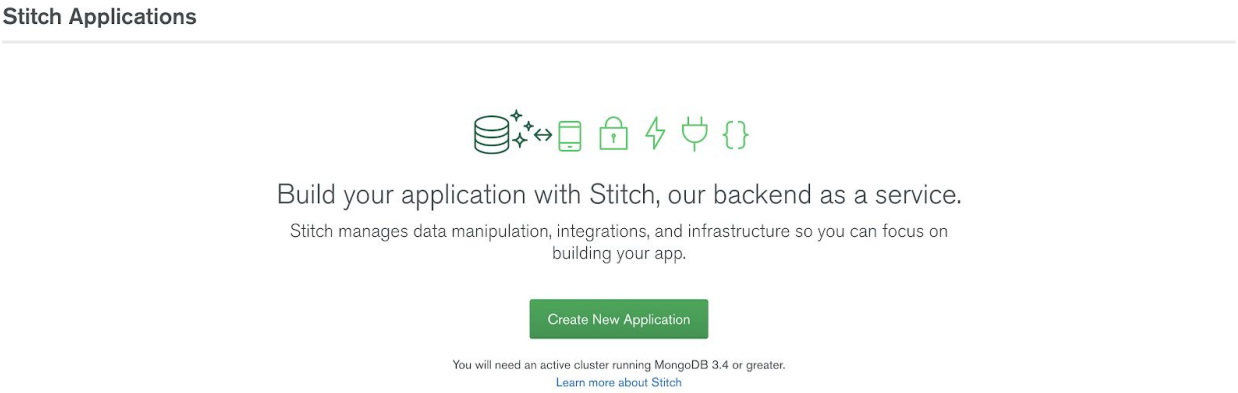
Lab 7 - Create a Microservice

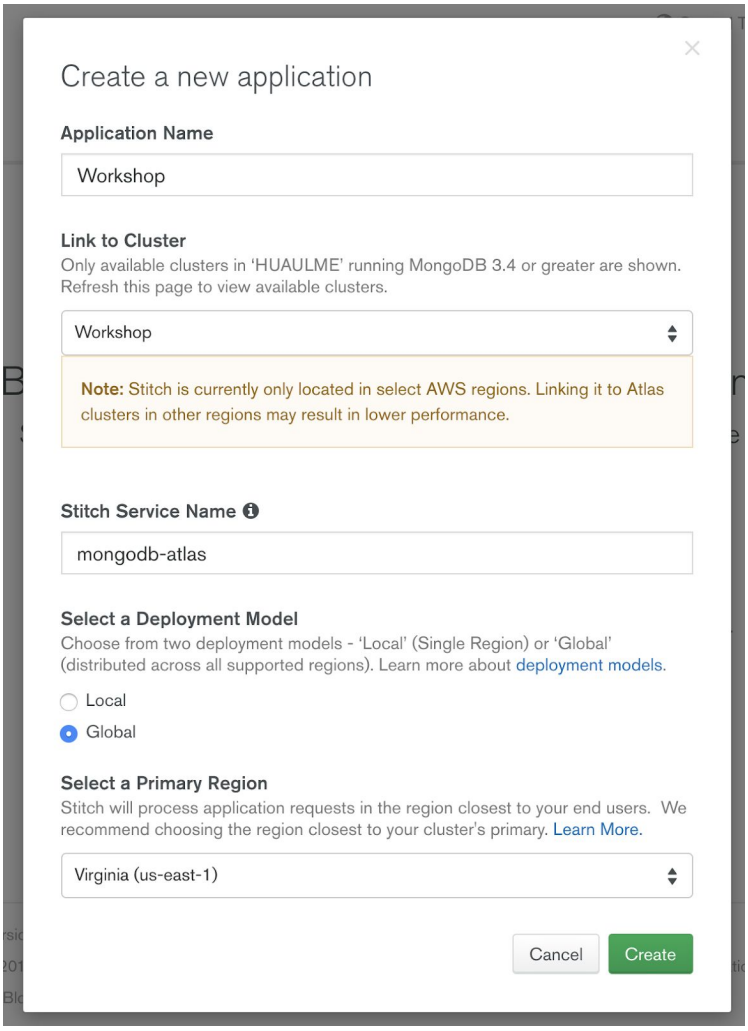
Next we'll create a microservice that we'll expose to our application teams as a REST API. We'll accomplish this via a MongoDB Stitch Function and HTTP Service. Our microservice will allow us to query for movies by name.

7.A. Create the Stitch Application

Stitch is a serverless platform, where functions written in JavaScript automatically scale to meet current demand. Return to the Atlas UI and click **Stitch Apps** on the menu on the left and then click the **Create New Application** button.

Name the application **Workshop**. The other defaults are fine:

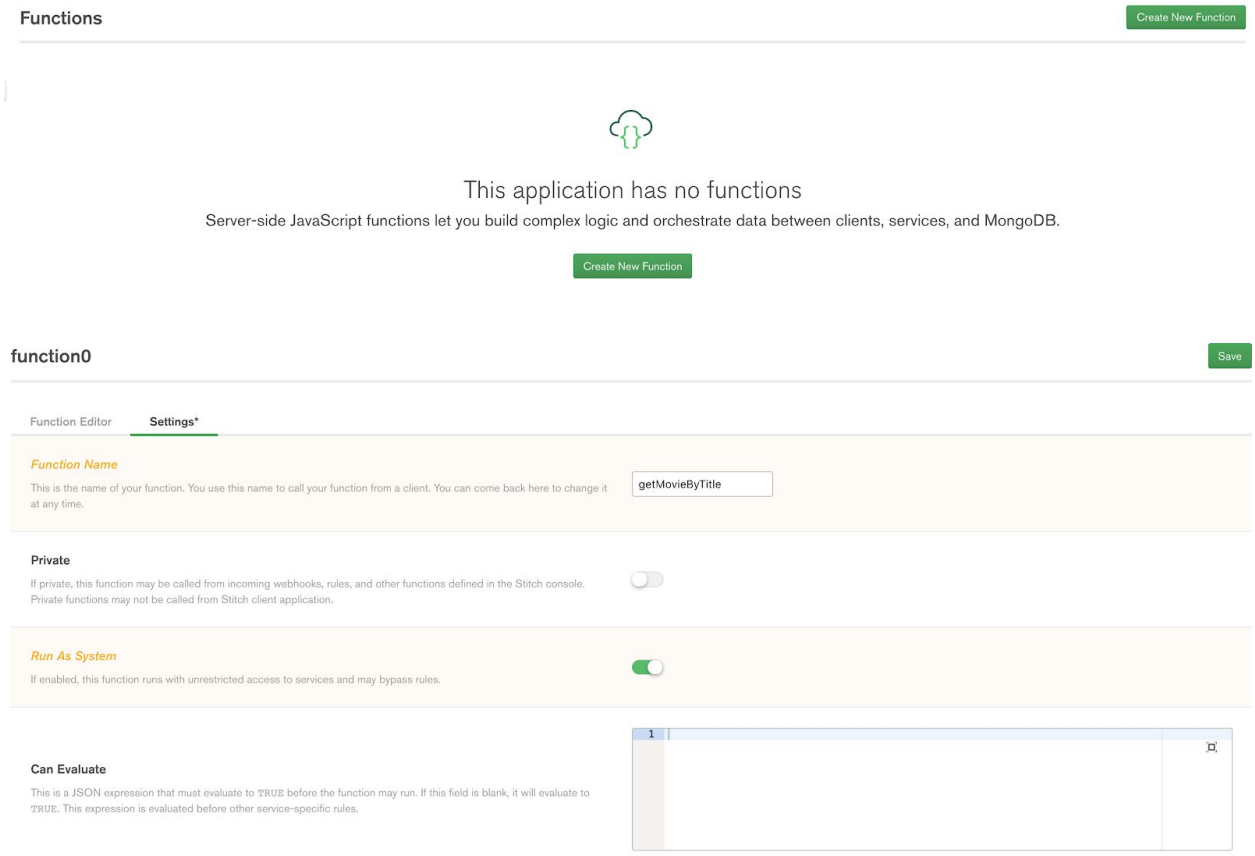




Click **Create**, which will take you to the **Welcome to Stitch!** page.

7.B. Create the Function getMovieByTitle

Now we'll create the function that queries movies by name. Click **Functions** on the left and then **Create New Function**. Name the function **getMovieByTitle** and choose **Run as System**:



Click **Save**, which will open the Function Editor.

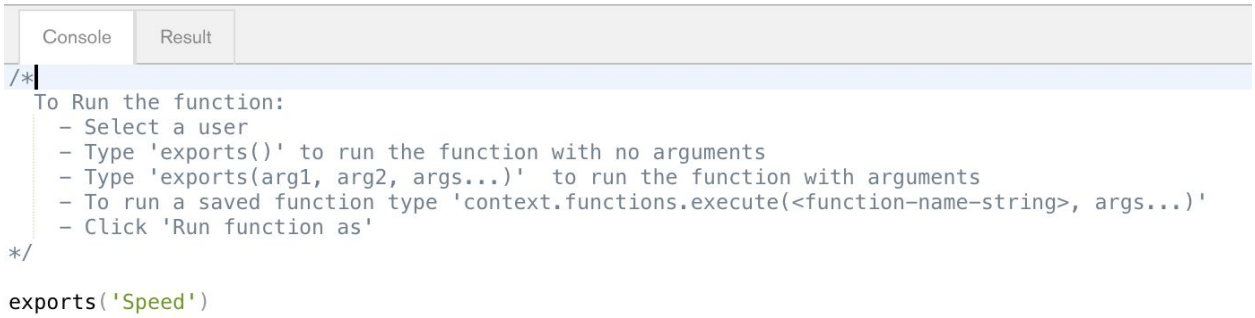
Replace the example code in the editor with the following:

```
exports = async function(arg){

  var collection = context.services
    .get("mongodb-atlas").db("sample_mflix").collection("movies");

  console.log ("IN GETMOVIEBYNAME FUNCTION");

  //Return a single document to matching the arg/movie name.
  var doc = collection.findOne({title: arg});
  if (typeof doc === "undefined") {
    return `No movies named ${arg} were found.`;
  }

  console.log(`FOUND A MATCHING MOVIE: ${arg}.`);

  return doc;
}
```

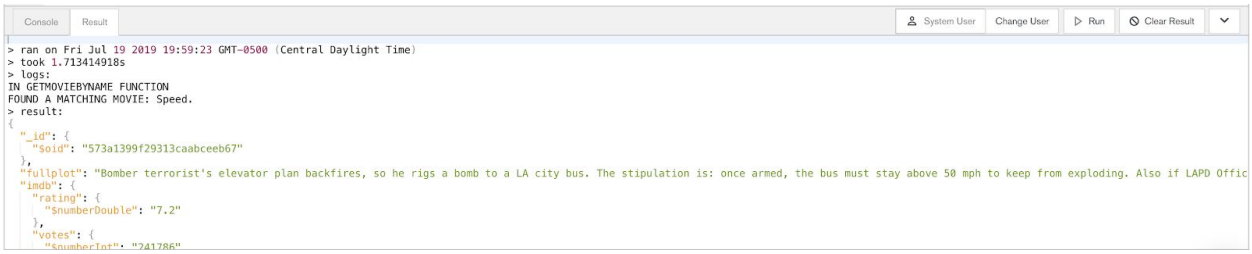
You can ignore any "Missing semicolon." warnings shown in the editor.

Let's review the code together. MongoDB has idiomatic drivers for most languages you would want to use. In this example we're using the findOne method to return a single document.

Click the **Console** tab below the editor to expand it. In the Console, change the argument from 'Hello world' to **'Speed'**:



Then click **Run** to test the function. You should get something similar to below with the result being the full document of the **Speed** movie.



Click **Save** and **Review & Deploy Changes** to save the function.

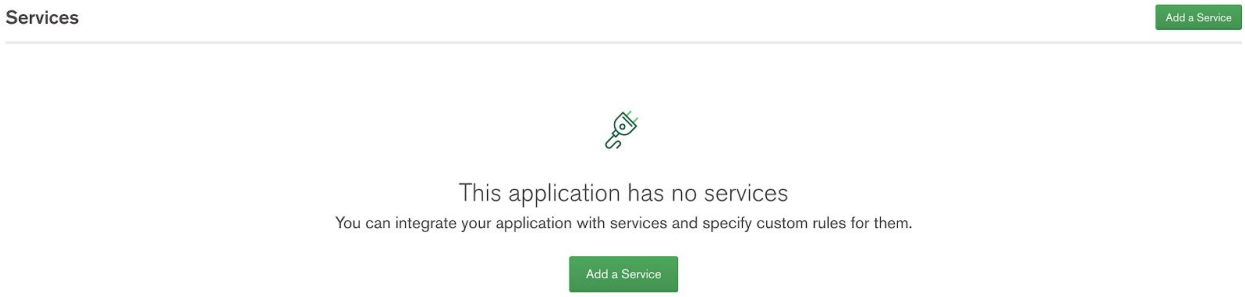
Optional: You can easily modify the function to look for movies by actor by making the changes below :

- Change field name to **“cast”**
- Change **findOne()** to **find()**
- Return results as an array: .**toArray()**

```
exports = async function(actor){

  var collection = context.services
    .get("mongodb-atlas").db("sample_mflix").collection("movies");

  console.log ("IN GETMOVIEBYACTOR FUNCTION");

  //Return a single document to matching the arg/cast name.
  var docs = await collection.find({cast: actor}).toArray();
  if (docs.length === 0) {
    return `No movies with ${actor} were found.`;
  }

  console.log(`FOUND A MATCHING MOVIE WITH: ${actor}.`);

  return docs;
}
```

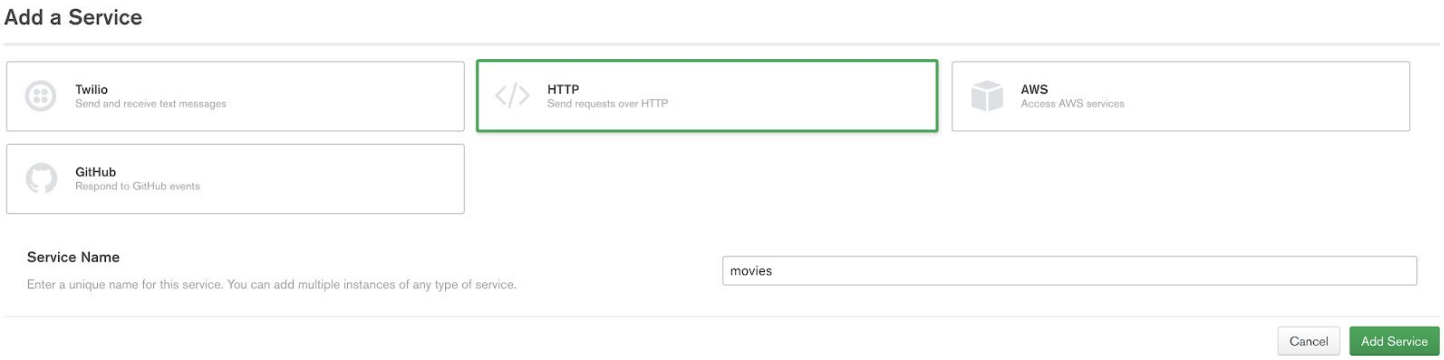
7.C. Expose the Function as a REST service

Click the **Services** menu on the left and then **Add a Service**.

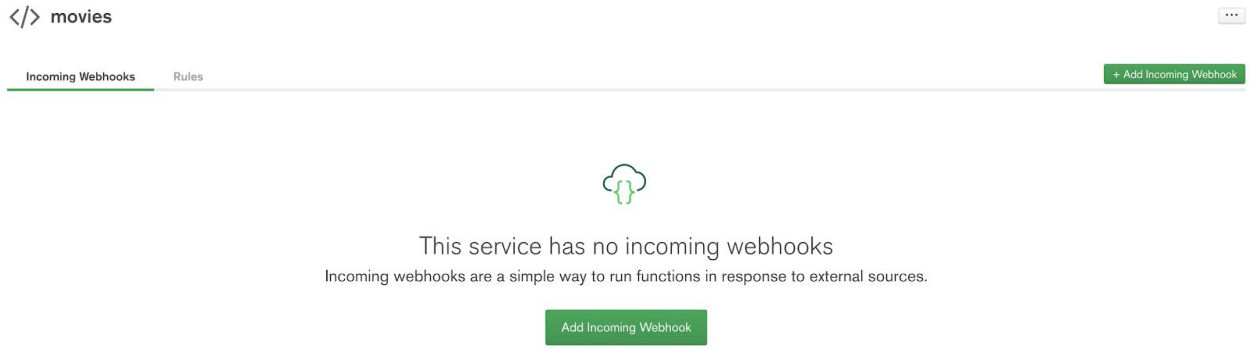


You'll notice Stitch supports service integrations with Twilio, AWS and GitHub, making it very easy for you to leverage these providers' unique capabilities. More generically, Stitch also provides an HTTP Service, which we will use to expose our function as a REST API.

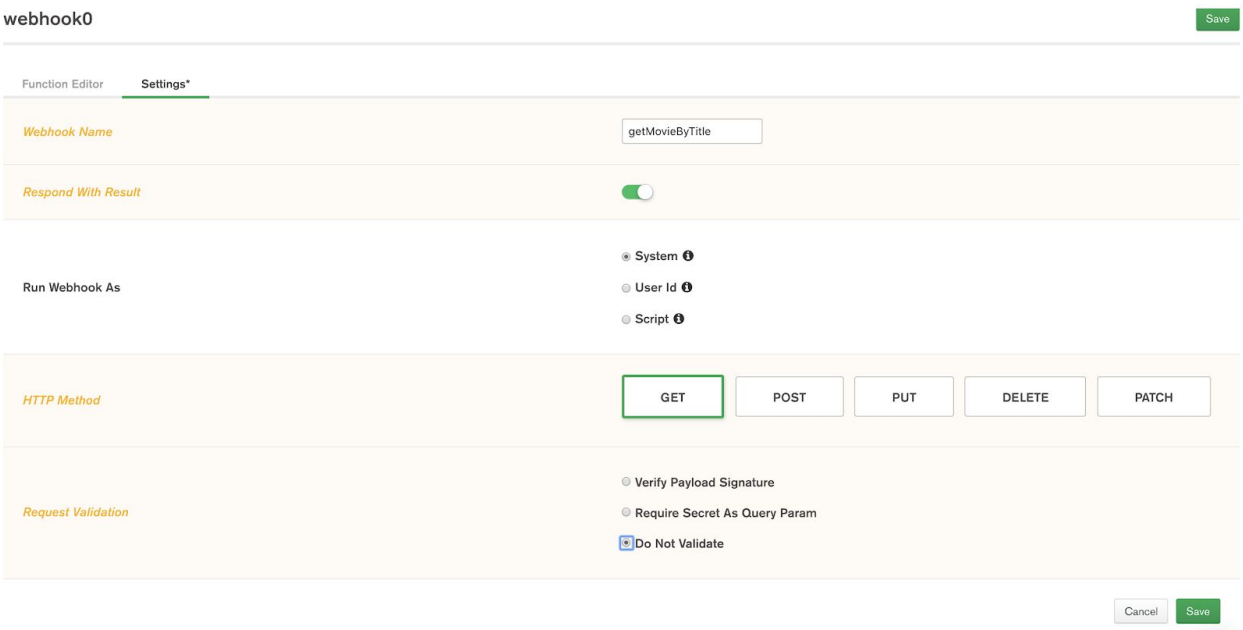
Select the HTTP service and name it **movies**:



Click **Add Service**. You'll then be directed to add an incoming webhook.



Click **Add Incoming Webhook** and configure the settings as shown below (the Webhook Name is **getMovieByTitle** and be sure to enable **Respond with Result**, set the HTTP Method to **GET** and Request Validation **Do Not Validate**):



To keep things simple for this introduction, we're running the webhook as the System user and we're skipping validation. Click **Save**, which will take us to the function editor for the service.

In the service function we will capture the query argument and forward that along to our newly created function. Note, I could have skipped creating the function and just coded the service functionality here, but the function allows for better re-use, such as calling it directly from a client application via the SDK. Replace the code with the following:

```
exports = function(payload) {

  var queryArg = payload.query.arg || '';
  return context.functions.execute("getMovieByTitle", queryArg);

};
```

Then set the **arg** in the Console to **'Speed'**:
*** note you must change from arg1 to arg ***

Function Editor | Settings

```
exports = function(payload) {

  var queryArg = payload.query.arg || '';
  return context.functions.execute("getMovieByTitle", queryArg);

};
```

Console | Result

```
/*
  To Run the function:
    - Select a user
    - Type 'exports()' to run the function with no arguments
    - Type 'exports(arg1, arg2, args...)'  to run the function with arguments
    - To run a saved function type 'context.functions.execute(<function-name-string>, args...)'
    - Click 'Run function as'
*/

exports({query: {arg: 'Speed'}, body: BSON.Binary.fromText('{"msg": "world"}')})
```

and click **Run** to verify the result:

Console | Result

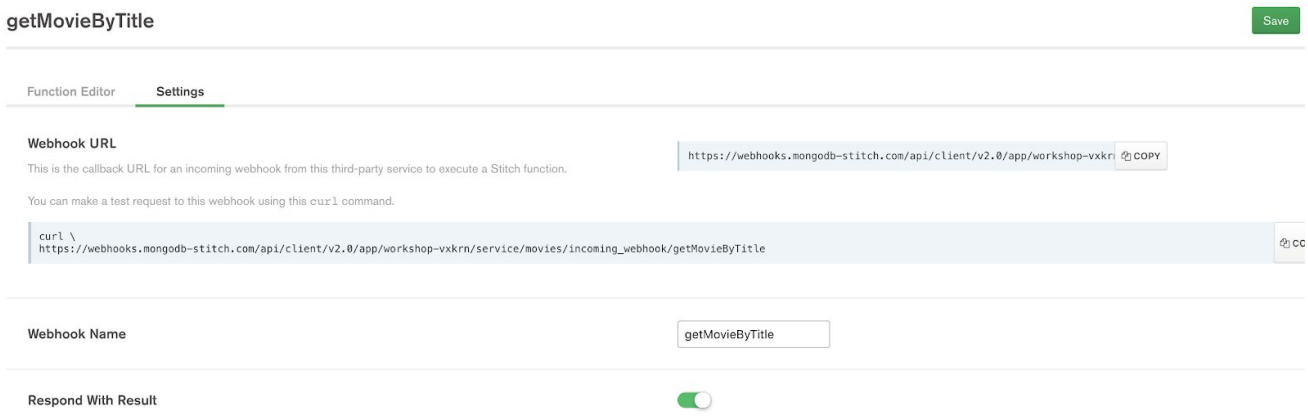
```
> ran on Fri Jul 19 2019 20:07:26 GMT-0500 (Central Daylight Time)
> took 790.844838ms
> logs:
IN GETMOVIEBYNAME FUNCTION
FOUND A MATCHING MOVIE: Speed.
> result:
{
  "_id": {
    "$oid": "573a1399f29313caabceeb67"
  },
  "fullplot": "Bomber terrorist's elevator plan backfires, so he rigs a bomb to a LA city bus. The stipulation
  "imdb": {
    "rating": {
      "$numberDouble": "7.2"
    },
    "votes": {
      "$numberInt": "241786"
    },
    "id": {
      "$numberInt": "111257"
    }
```

Click **Save** to the service. The **Review & Deploy Changes.**

7.D. Use the API

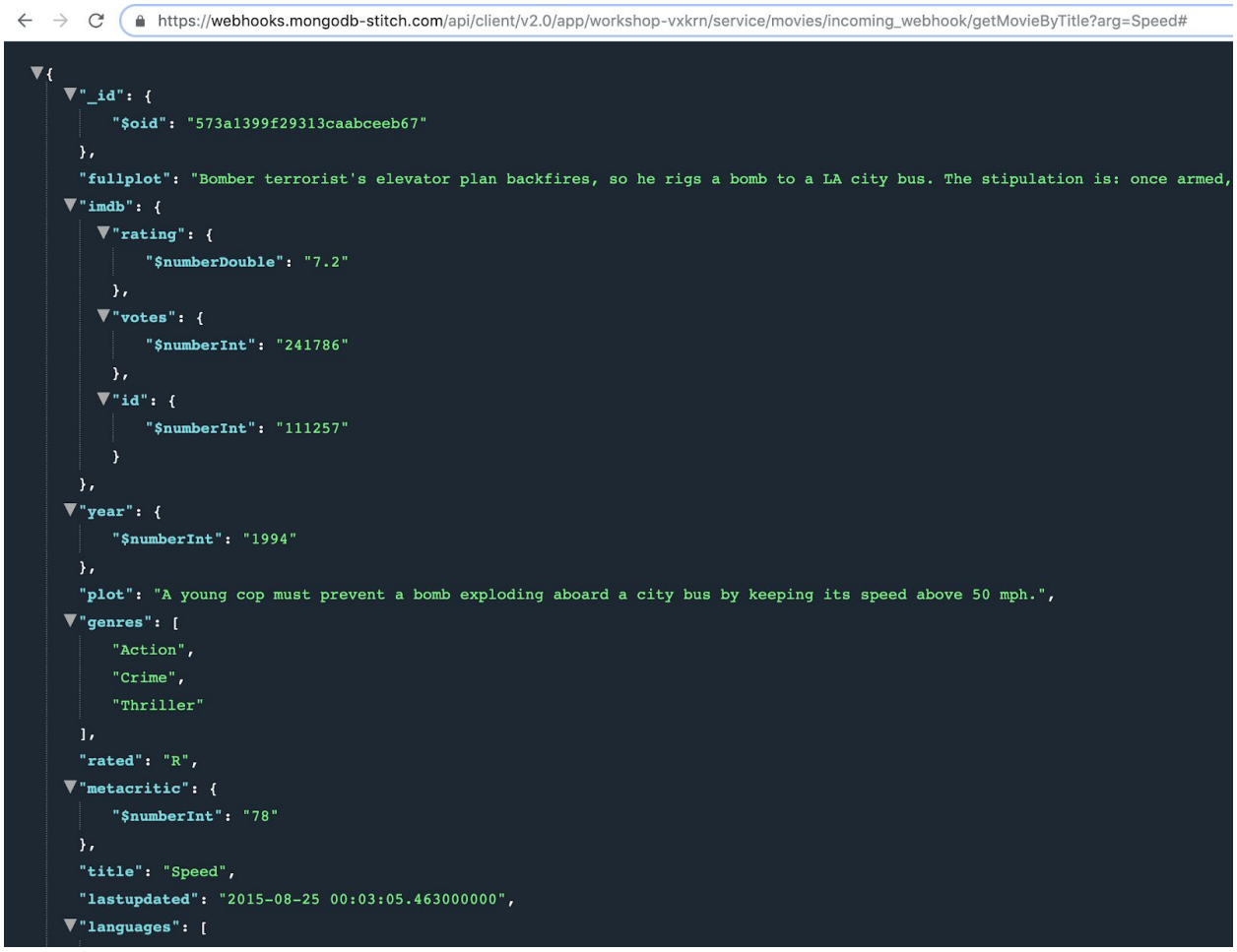
The beauty of a REST API is that it can be called from just about anywhere. For the purposes of this workshop, we're simply going to execute it in our browser. However, if you have tools like Postman installed, feel free to try that as well.

Switch back to the **Settings** tab of the getMovieByTitle service and you'll notice a Webhook URL has been generated.



Click the **COPY** button and paste the URL into your browser. Append the following to the end of your URL: **?arg=Speed**
REMEMBER THIS URL TO USE LATER IN YOUR CODE!

and submit (your output from your browser may look different):



Feel free to play around with the API argument to find other movies such as:

?arg=Constantine

or

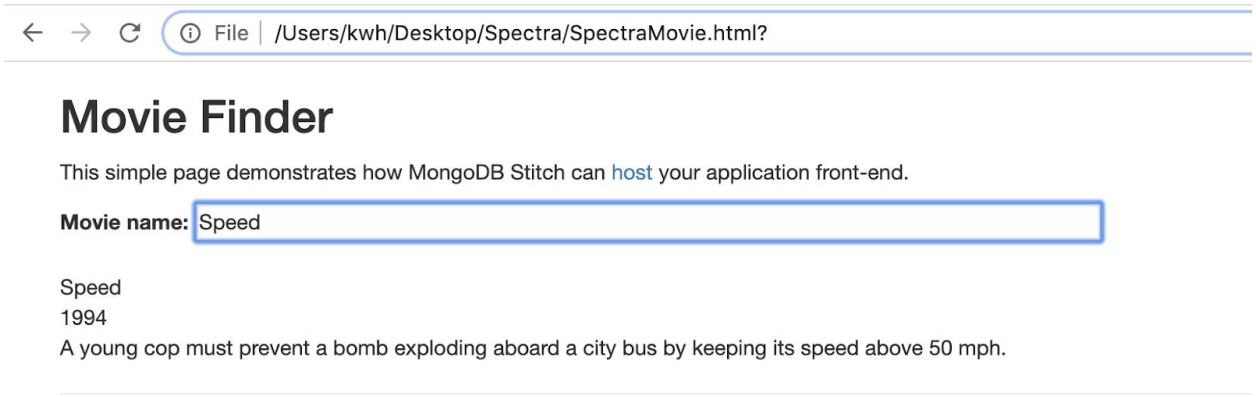
?arg=The%20Matrix

Lab 8 - Host your Application

Yes, Stitch can also host your application, therefore supporting the entire application stack. Let's see this in action using a very simple front-end that will use the REST API we just created and allow us to search for movies by title.

Download and Test the UI

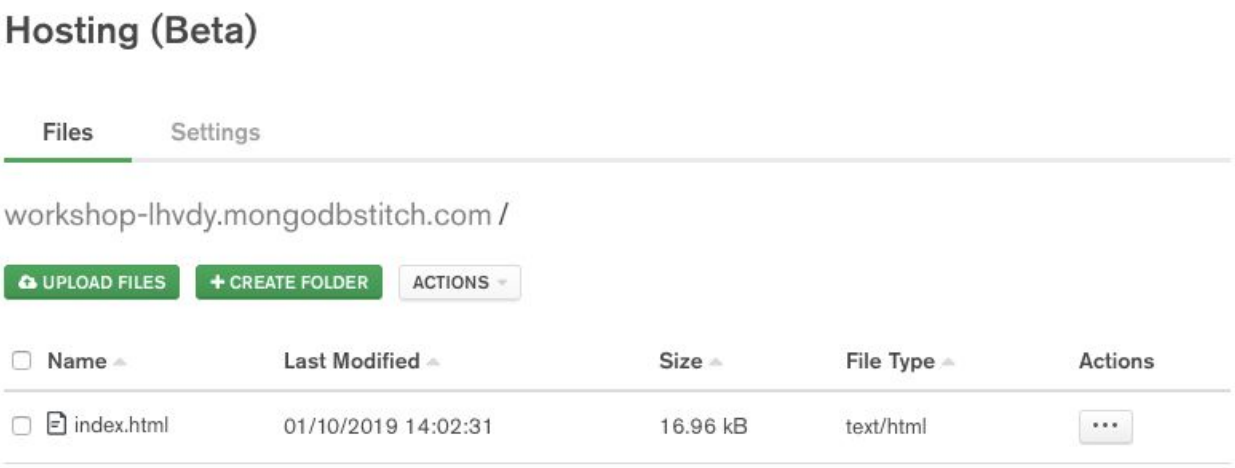
Download this index.html file and open it in your browser. It should work as is because it's currently pointing to a pre-existing REST API:



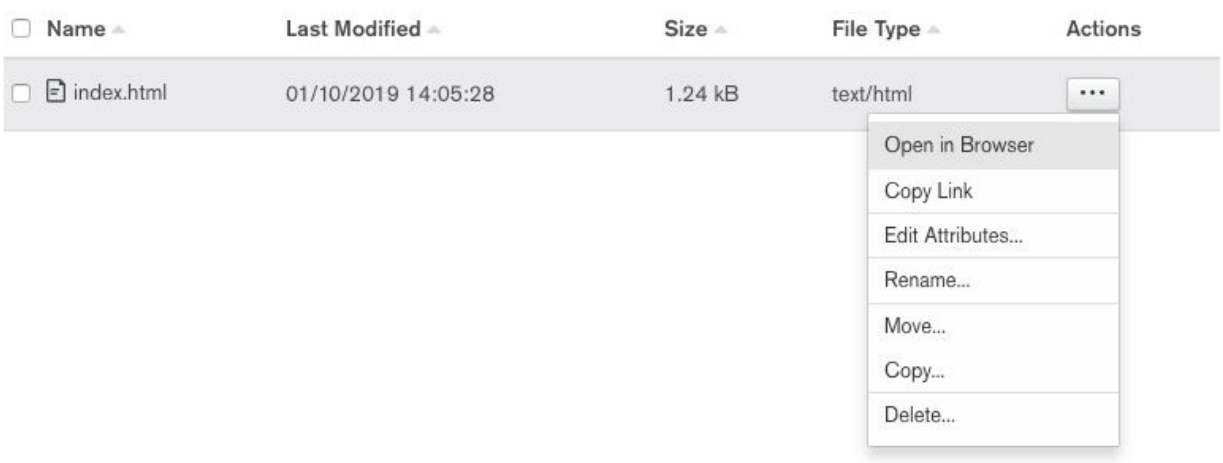
Open the index.html file in a code editor and familiarize yourself with the contents. Then replace the value of the **`webhook_url`** variable around **line 40** with the Webhook URL from the Stitch Service you created earlier. Save and test the UI.

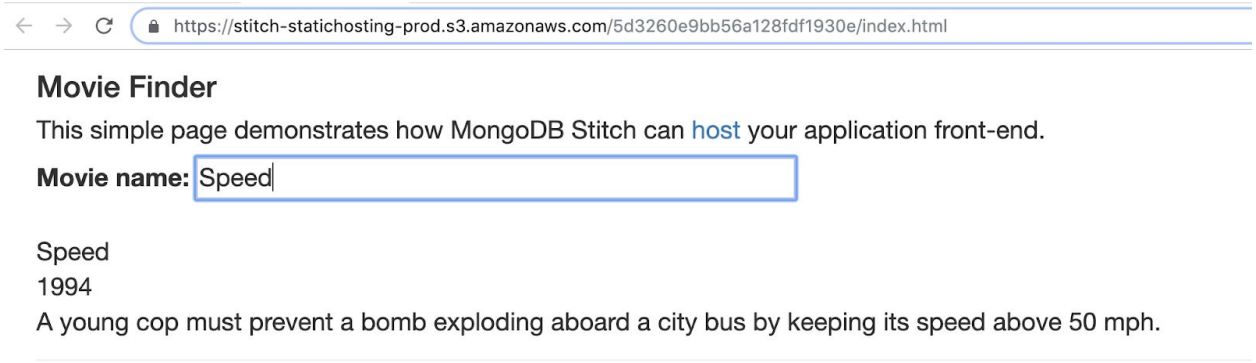
Host the UI on Stitch

In the Stitch UI, click the **Hosting** in the left navigation bar and then click **Enable Hosting**:



Upload your index.html file using the **UPLOAD FILES** button. When prompted if you want to overwrite the existing index.html file, click **Upload**. Then select the action to open your file in a browser:





Notice the URL in your browser. Your movie application is now live on the Internet! Test it and confirm that the app is still successfully using your movie microservice.

And that's a wrap!

Lab 9 Optional - View the Stitch Logs