Lab 8

Submit Assignment

Due Monday by 11:59pm **Points** 100 **Submitting** a file upload

CS-546 Lab 8

Template Time

For this lab, you will be using HTML, CSS, and Handlebars to make your first simple templated web application! You will be building a form that allows you to search through the TV Maze API for a keyword.

You will not need to use a database for this lab.

You **must** use the `async/await` keywords (not Promises). You will also be using `axios` (https://github.com/axios/axios) , which is a HTTP client for Node.js; you can install it with `npm i axios`.

TV Maze API

You will be using two endpoints of the TV Maze API which is an API about TV shows for your Axios calls. The search show endpoint where you pass the search term as a query string parameter: `http://api.tvmaze.com/search/shows?Search_Term_Here` and then you'll get an individual show using the endpoint `http://api.tvmaze.com/shows/:id` where `:id` is the ID of the show you are looking up.

You will use these two endpoints to make your axios.get calls depending on which route is called.

You will be making three routes/pages in your application:

- `http://localhost:3000/` the main page of this application will provide a search form to start a search of shows for a keyword.
- `http://localhost:3000/search` this page will make the axios call to the search endpoint and return up to 20 matching results that contain the provided request form param, `searchTerm`

- `http://localhost:3000/shows/{id}` this page will show all the details of the show with the id matching the provided URL param, `id`

All other URLS should return a 404

```
## `GET http://localhost:3000/`
```

This page will respond with a valid HTML document. The title of the document should be "*Show Finder*". You should have the title set as the `<title>` element of the HTML document and as an `h1` in your document.

Your page should reference a CSS file, `/public/site.css`; this file should have *at least 5 rulesets* that apply to this page; these 5 rules can also apply to elements across all of your pages, or be unique to this page.

You should have a `main` element, and inside of the `main` element have a `p` element with a brief (2-3 sentence description) of what your website does.

Also inside the `main` element, you will have a `form`; this `form` will `POST` to `/search`. This `form` will have an `input` and a `label`; the `label` should properly reference the same `id` as the `input`. You should also have a `button` with a type of `submit` that submits the form. The `input` in your `form` should have a `name` of `searchTerm`.

`POST http://localhost:3000/search`

This route will read the `searchTerm` parameter and then make an axios call to the TV Maze endpoint searching for that keyword. For example, if the user typed `under` in the input field, you would make the axios call to: **http://api.tvmaze.com/search/shows?q=under (http://api.tvmaze.com/search/shows?q=girls)**

This route will respond with a valid HTML document with the results returned from the API. The title of the document should be "*Shows Found*". You should have the title set as the `<title>` element of the HTML document and as an `h1` in your document. In an `h2` element, you will print the supplied `searchTerm`.

Your page should reference a CSS file, `/public/site.css`; this file should have *at least 5 rulesets* that apply to this page; these 5 rules can also apply to elements on `/`, or be unique to this page.

You should have a `main` element, and inside of the `main` element have a `ul` tag that has a list of up to 20 shows matching the `searchTerm` found in the request body in the following format (after searching `under`). **DO NOT SHOW MORE THAN 20 SHOWS.**

```
<ul>
    <li>
        <a href="/shows/5828">Over Under</a>
    </li>
    <li>
        <a href="/shows/1">Under The Dome</a>
    </li>
    <li>
        <a href="/shows/48087">Scottish Vets Down Under</a>
    </li>
</ul>
```

You must also provide an `a` tag that links back to your `/` route with the text `Make another search`.

If no matches are found, you will print the following HTML paragraph:

```
<p class="not-found">We're sorry, but no results were found for {searchTerm}.</p>
```

If the user does not input text into their form or enters just spaces into the input field, make sure to give a response status code of 400 on the page, and render an HTML page with a paragraph class called `error`; this paragraph should describe the error.

`GET http://localhost:3000/shows/{id}`

This route will query the TV Maze API using the the `id` parameter in the URL (for example: **http://api.tvmaze.com/shows/1 (http://api.tvmaze.com/shows/:id)**) and will respond with a valid HTML document with some of the show details. The title of the document should be the `name` of the show. You should have the title set as the `<title>` element of the HTML document.

Your page should reference a CSS file, `/public/site.css`; this file should have *at least 5 rulesets* that apply to this page; these 5 rules can also apply to elements on `/`, or be unique to this page.

You should have a `main` element, and inside of the `main` element, you will have a `div` tag that has an `h1` with the show `name`, an `img` which the `src` is set to the value read from `image.medium` in the data which is a URL to an image for the show, and a `dl` (definition list) of

the following properties of the matching show: `language`, `genres` (the entire array in an `ul`), `rating.average`, `network.name`, and `summary` in the HTML structure noted below.

Note: Some show summaries contain HTML tags (not all, but some), as shown in the example below. You should use a regex expression to strip any HTML tags out of the Summary before displaying it on the page.

Matching Show Data Returned from API (We will not be using all the fields, just the ones noted above):


```
{
  "id": 1,
  "url": "http://www.tvmaze.com/shows/1/under-the-dome  (http://www.tvmaze.com/shows/1/under-the-dome) ",
  "name": "Under the Dome",
  "type": "Scripted",
  "language": "English",
  "genres": [
    "Drama",
    "Science-Fiction",
    "Thriller"
  ],
  "status": "Ended",
  "runtime": 60,
  "premiered": "2013-06-24",
  "officialSite": "http://www.cbs.com/shows/under-the-dome/  (http://www.cbs.com/shows/under-the-dome/) ",
  "schedule": {
    "time": "22:00",
    "days": [
      "Thursday"
    ]
  },
  "rating": {
    "average": 6.5
  },
  "weight": 97,
  "network": {
    "id": 2,
    "name": "CBS",
    "country": {
      "name": "United States",
      "code": "US",
```

```
      "timezone": "America/New_York"
    }
  },
  "webChannel": null,
  "externals": {
    "tvrage": 25988,
    "thetvdb": 264492,
    "imdb": "tt1553656"
  },
  "image": {
    "medium": "http://static.tvmaze.com/uploads/images/medium_portrait/81/202627.jpg (http://static.tvmaze.com/uploads/images/medium_portrait/81/202627.jpg)",
    "original": "http://static.tvmaze.com/uploads/images/original_untouched/81/202627.jpg (http://static.tvmaze.com/uploads/images/original_untouched/81/202627.jpg)"
  },
  "summary": "<p><b>Under the Dome</b> is the story of a small town that is suddenly and inexplicably sealed off from the rest of the world by an enormous transparent dome. The town's inhabitants must deal with surviving the post-apocalyptic conditions while searching for answers about the dome, where it came from and if and when it will go away.</p>",
  "updated": 1573667713,
  "_links": {
    "self": {
      "href": "http://api.tvmaze.com/shows/1 (http://api.tvmaze.com/shows/1)"
    },
    "previousepisode": {
      "href": "http://api.tvmaze.com/episodes/185054 (http://api.tvmaze.com/episodes/185054)"
    }
  }
}
```


HTML Format Printed for the show. This will go into your `main` element:

```
<div>
  <h1>Under The Dome</h1>
  <img alt= "Under TheDome"src="http://static.tvmaze.com/uploads/images/medium_portrait/81/202627.jpg"/>
  <dl>
    <dt>Language</dt>
    <dd>English</dd>
    <dt>Genres</dt>
    <dd>
      <ul>
```

```
        <li>Drama</li>
        <li>Science-Fiction</li>
        <li>Thriller</li>
      </ul>
    </dd>
    <dt>Average Rating</dt>
    <dd>6.5</dd>
    <dt>Network</dt>
    <dd>CBS</dd>
    <dt>Summary</dt>
    <dd>Under the Dome is the story of a small town that is suddenly and inexplicably sealed off from the rest of the world by an enormous transp
arent dome. The town's inhabitants must deal with surviving the post-apocalyptic conditions while searching for answers about the dome, where it
 came from and if and when it will go away.</dd>
  </dl>
</div>
```

If there is no show found for the given ID, make sure to give a response status code of 404 on the page, and render an HTML page with a paragraph class called `error`; this paragraph should describe the error.

`http://localhost:3000/public/site.css`

This file should have 5 rulesets that apply to the `/` route, and 5 rulesets that apply to all of your pages. Rulesets may be shared across both pages; for example, if you styled a `p` tag, it would count as 1 of the 5 for both pages.

You may include more than 5 rulesets if you so desire.

References and Packages

Basic CSS info can easily be referenced in the **MDN CSS tutorial (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started)** .

Hints

You can use variables in your handlebars layout, that you pass to res.render. For example, in your layout you could have:

```
<meta name="keywords" content="{{keywords}}" />
```

And in your route:

```
res.render("someView", {keywords: "dogs coffee keto"});
```

Which will render as:

```
<meta name="keywords" content="dogs coffee keto" />
```

Or, perhaps, the title tag.

Requirements

1. You **must not submit** your node_modules folder
2. You **must remember** to save your dependencies to your package.json folder
3. You must do basic error checking in each function
4. Check for arguments existing and of proper type.
5. Throw if anything is out of bounds (ie, trying to perform an incalculable math operation or accessing data that does not exist)
6. You **MUST** use `async/await` for all asynchronous operations.
7. You **must remember** to update your package.json file to set `app.js` as your starting script!
8. **Your HTML must be valid (https://validator.w3.org/#validate_by_input)** or you will lose points on the assignment.
9. Your HTML must make semantical sense; usage of tags for the purpose of simply changing the style of elements (such as `i`, `b`, `font`, `center`, etc) will result in points being deducted; think in terms of content first, then style with your CSS.
10. **You can be as creative as you'd like to fulfill front-end requirements**; if an implementation is not explicitly stated, however you go about it is fine (provided the HTML is valid and semantical). Design is not a factor in this course.
11. All inputs must be properly labeled!
12. All previous requirements about the `package.json` author, start task, dependenices, etc. still apply