Express with APIs - OMDB

We'll be creating an app that connects to OMDB, a public movie API. You will need a key. Go to the site to register for a free one. Keep API keys out of public repos!

Pre-reqs

- axios - in we do
- node/express/ejs/express-ejs-layouts
- forms in full-stack GET action/query params?

Getting Started

- Fork and clone this repository, which has a starter app provided for you.
- Run `npm install` to install dependencies
- Read the API documentation.

Part 1: Search

User Stories

- [] As a user, I want to go to a home page to search for movies.
- [] As a user, I want to see movie results based on my search query.
- [] As a user, I want to pick a movie result and see detailed information about the movie.

Steps to Achieve

First, review the project folder:

You have been provided with four `.ejs` files in your folder.

- `index.ejs` to show a search form
- `results.ejs` to show the OMDB search data
- `detail.ejs` to show the details about one movie
- `layout.ejs` to for your page layouts

`server.js` already has the ejs template engine and layouts setup.

Part 1: Plan your app

Consider the user stories and plan out the routes you will need. How will the form submit data? How will you get the details or one specific movie?

Remember, you can submit an html form with an HTTP action of `GET` and the input's `name` attributes will be used as *request query parameters*

this form:

```
<form action="/resource" method="GET">
  <input type="text" name="search_term" />

  <input type="submit" />
</form>
```

will result in this URL:

`/resource?search_term=my+search`

where `/resource` is the route, `search_term` is the query parameter, and `my+search` is what was typed into the input when the form was submitted.

you could log this on your backend route with `console.log(req.query.search_term)`

▶ Help! How do I turn user stories into features?

one by one, make an actionable feature/features for the user stories:

- [] As a user, I want to go to a home page to search for movies.
  - [] make a new GET `/` route
  - [] GET `/` should render a form to the user that searches omdb
- [] As a user, I want to see movie results based on my search query.
  - [] make a new GET `/results` route
  - [] GET `/results` should accept a request query parameter of a movie to search (`req.query` on your backend)
  - [] GET `/results` should search the omdb API with the request query parameter
  - [] GET `/results`should render the search results from omdb to the use
- [] As a user, I want to pick a movie result and see detailed information about the movie.
  - [] make new GET `/detail:movie_idId` route
  - [] GET `/detail/:movie_id` should accept an movie_id as URL path varaible
  - [] GET `/detail/:movie_id` should search the omdb API with the movie_id from the url path
  - [] GET `/detail/:movie_id` should render a detail view of the movie the omdb API responds with

Plan out each route you will need along with the HTTP request verb and what the response will be.

▶ Wait! What routes do I need?

You will need three routes for mvp, factoring them into a controller is a desgin implementation that is left up to you.

Method	Path	Purpose
GET	`/`	renders a search form to user
GET	`/results`	accepts a movie title as a query parameter, searches the omdb API and shows the search results to the user

Method	Path	Purpose
GET	`/detail/:movie_id`	accepts an movie_id as a URL parameter, searches the omdb API for details with the id from the URL and shows the detail response to the user

Part 2: Stub those routes

write a simple route stub for each of your routes to test the functionality. At first, they only need to respond with a simple message to verify that everything is hooked up right.

Part 3: Build your route stubs

Build out the functionality of each route one by one, and test with postman to verify that everything is working correctly.

Part 4: Render your views

Build the functionality of each ejs view one by one and reconfigure the the corresponding route to render it. Test that your view works correctly before moving on to the next one!

Tips

- Remember the axios syntax for a GET request:
-
- ```
  axios.get('some url goes here')
    .then(function (response) {
      console.log(response);
    })
  ```
- The movie api returns an array of movies inside the `Search` Key.
- Example Search URL: http://www.omdbapi.com/?s=matrix
- Example Movie Detail URL: http://www.omdbapi.com/?i=tt0133093
- You will need to modify the above URLs to include an entry in the query string for your API Key. The exact thing to add is `apikey=XXXXXXXXX` replacing the Xs with your key. How do we separate multiple key-values pairs in a query string?
- Keep your API key as an environment variable inside a .env file and make sure you DO NOT EVER check this file into git. How do we ensure it stays out?
- In the results from the API, we notice that in every movie entry there is an omdbID. In the rendered HTML for `/results`, have each movie link to a route like `/detail/tt234323` (where `tt234323` is the id for that movie).
- Make sure you call `res.render` inside the callback function of the API call.
  ```

Bonuses

- Add stars images to reflect the imdb ratings
- Add an error page that renders when a route has a problem
- add a 404 page that renders when a route isn't found
- Figure out what parameters are need to access the Rotten Tomato information, and display that information to the page

Saving Faves Super Duper Bonus

User Stories

- [] As a user, I want to save movies from my search results to a list of my faves.
- [] As a user, I want to perform this action from the movie detail page.

Steps to Achieve

1. Install the new node modules needed for database access.
2. Initialize sequelize for this project.
3. Update the config file and create a database named `omdb`.
4. Create a `fave` model with two fields- `title:string` and `movie_idid:string`
5. Run migrations.
6. Require your model into the location of your routes.
7. Modify your `detail.ejs` to include a form for adding this movie as a fave:

- This form should have a `POST` method, with an action of `/faves`
- It should contain two *hidden* fields containing the title and movie_id ID of this movie. These fields should be named the same as your model attribute names.

8. Write your POST route for `/faves`:

- Use `req.body` to access body data from the form.
- Use the fave model to save this data to your database. YOU WILL NEED TO REQUIRE THE MODEL TO USE IT.
- In the callback of your `create`, use `res.redirect` to redirect to the GET route for your faves.

9. Write your GET route for `/faves`:

- Use the fave model to get all faves from your database.
- In the callback, use `res.render` to render all your faves to a page named `faves.ejs` (not provided).

Licensing

1. All content is licensed under a CC-BY-NC-SA 4.0 license.
2. All software code is licensed under GNU GPLv3. For commercial use or alternative licensing, please contact legal@ga.co.