6.4.3 Create a Link Between Pages

In this step, we'll convert the list of repos into a list of links. We need a process that will programmatically navigate to a new page once a user has clicked a repo name on the homepage, thus linking the homepage, `index.html`, to the `single-repo.html` page.

To accomplish this, let's open the `homepage.js` file to find the `displayRepos()` function and target the `for` loop that's dynamically creating HTML elements from the GitHub API response. Change the expression that creates a `<div>` to create an `<a>` element instead. We'll also need to add an expression to create a new `href` attribute.

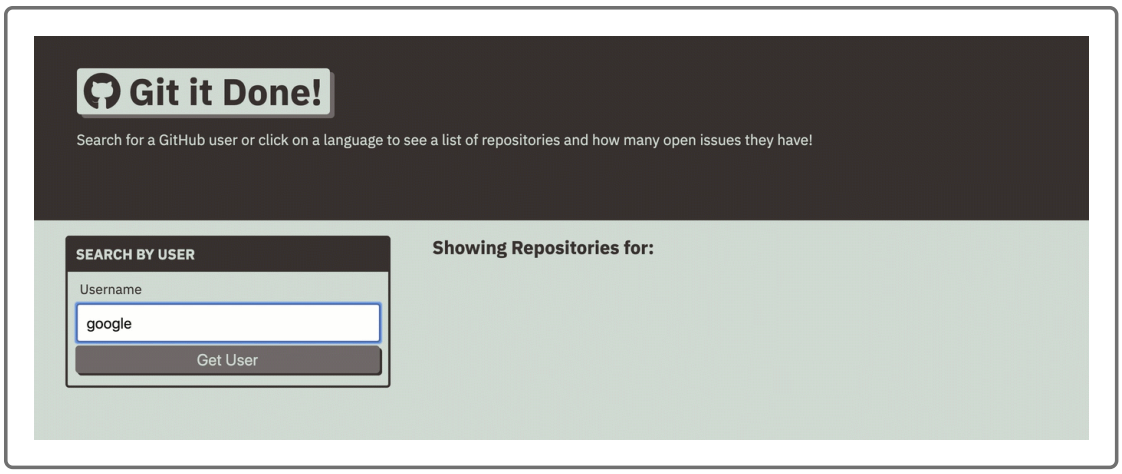
Make the following changes to the code:

```
// loop over repos
for (var i = 0; i < repos.length; i++) {
  // format repo name
  var repoName = repos[i].owner.login + "/" + repos[i].name;

  // create a container for each repo
  var repoEl = document.createElement("a");
  repoEl.classList = "list-item flex-row justify-space-between align-c
  repoEl.setAttribute("href", "./single-repo.html");
  // create a span element to hold repository name
```

```
  ...
}
```

Let's see if the changes are linking the `single-repo.html` page to the `index.html` page. Save the files and load `index.html` in the browser, then enter a username. Select and click on a repo name; doing so should take you to the issues page. Repeat the process a few times, clicking on different repo names to see if this effect is consistent. The following animation illustrates what you should be seeing:



Notice that although we were routed to the `single-repo.html` page, the list of issues remained the same. Why's that?

Although the routing has been corrected, the request to the GitHub API issues endpoint is still hardcoded with `facebook/react` (or whatever you used when testing). In order to change this to reflect the user's selected repo name, we'll need to pass the repo name from `index.html` to `single-repo.html`. Thankfully, we can do this with query parameters.

IMPORTANT

> Notice that the path to the `single-repo.html` page in the newly created `href` attributes is a relative path from the `index.html` page, not `homepage.js`, where the element is created. From the browser's

> perspective, although dynamically created, these HTML elements become part of the markup—as shown in the page source of the rendered page. So when you create links to HTML pages in JavaScript, make sure the paths are relative to the HTML pages, not the JavaScript file.

Query Parameters

We use **query parameters**—strings appended to the end of URLs—to define actions, pass information, or specify content to the webpage or API endpoint. The `?` symbol at the end of the URL identifies the parameters. Parameters are assigned values in a `key=value` format; the `=` assigns the value to the parameter/key. This information is passed to the website as part of the URL.

PAUSE

Where have we seen the `?key=value` at the end of a URL before?

We saw it previously when we requested issues in ascending order in the `single.js` file in the `getRepoIssues()` function. We appended `?direction=asc` to the URL request as part of the `fetch` call.

Hide Answer

Let's start with an example by typing the following into the browser's address bar: `https://www.google.com/search?q=javascript`.

This diagram shows the breakdown of the query parameter:

On the URL `www.google.com/search?q=javascript`, we appended a query parameter designated by the `?` character. The parameter was defined as `q`, and its value is the word `javascript`. A query string is the string that follows the `?`, which in this case would be `q=javascript`.

When the server at Google receives the request, it looks at the query parameters for further information. When it sees a value for the key `q`, it interprets this as a search for the corresponding value. In this case, the value is "javascript", so the server at Google responds with the search results that match "javascript".

DEEP DIVE

DEEP DIVE

To learn more, read the **Wikipedia article on query parameters (https://en.wikipedia.org/wiki/Query_string)**.

Now let's use query parameters to pass a repo name to the `single-repo.html` page. We must first open the `homepage.js` file, revisit the `for` loop containing the newly created `<a>`, and adjust the `href` attribute to contain the query parameter, the selected repo's name. We've already assigned the variable `repoName`, so we can use that value to construct a query string to append to the end of the URL, `./single-repo.html`.

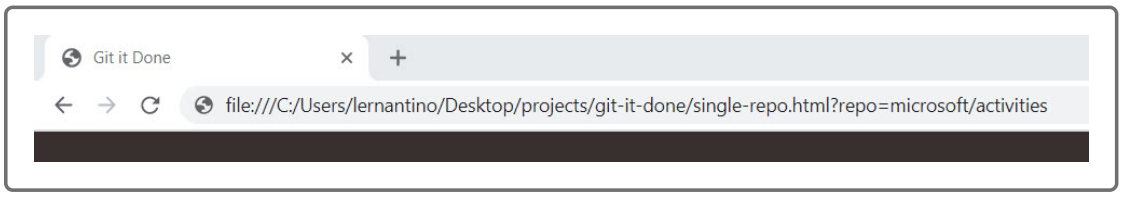
Try this task yourself by creating a new parameter called `repo` that's equal to the value saved in the `repoName` variable.

This is the query string we must append to the `href` value: `?repo=<repo>`.

Once updated, the `displayRepos()` function will look like this:

```
// create a link for each repo
var repoEl = document.createElement("a");
repoEl.classList = "list-item flex-row justify-space-between align-cen
repoEl.setAttribute("href", "./single-repo.html?repo=" + repoName);
```

To check if this change to the code works, let's save the files and open `index.html` in the browser. After entering a GitHub name and clicking on a repo name, we'll be directed to the `single-repo.html` page, as demonstrated previously. Nothing about the webpage will have changed yet. However, a look at the URL in the address bar of the browser will confirm that the query parameter is present. In the following example, the repo `microsoft/activities` was selected:



Excellent work! You've made a nice breakthrough, passing information from one page to another page. Notice how this URL looks so different than the previous example with Google, because we're loading the HTML file from a local directory, not from a server online.

So how does the second page retrieve this information so that it can pass to the API call? We'll explore that question in the next step.