6.3.5 Convert Fetched Data into DOM Elements

Turning GitHub issue data into DOM elements will look similar to what you did with repositories in the `displayRepos()` function, so let's create a new function for the issues.

In `single.js`, add the following function that accepts a parameter called `issues`:

```
var displayIssues = function(issues) {

};
```

You can call this function after a successful HTTP request. Update the logic in `getRepoIssues()` to call `displayIssues(data)` instead of console logging the response:

```
fetch(apiUrl).then(function(response) {
  // request was successful
  if (response.ok) {
    response.json().then(function(data) {
      // pass response data to dom function
      displayIssues(data);
```

```
    });
  }
  else {
    alert("There was a problem with your request!");
  }
});
```

In the `displayIssues()` function, loop over the response data and create an `<a>` element for each issue, as shown here:

```
for (var i = 0; i < issues.length; i++) {
  // create a link element to take users to the issue on github
  var issueEl = document.createElement("a");
  issueEl.classList = "list-item flex-row justify-space-between align-
  issueEl.setAttribute("href", issues[i].html_url);
  issueEl.setAttribute("target", "_blank");
}
```

Again, this looks like what we did with repo data in `homepage.js`. The biggest difference is the data we're working with. Issue objects have an `html_url` property, which links to the full issue on GitHub. We also added a `target="_blank"` attribute to each `<a>` element, to open the link in a new tab instead of replacing the current webpage.

HIDE PRO TIP

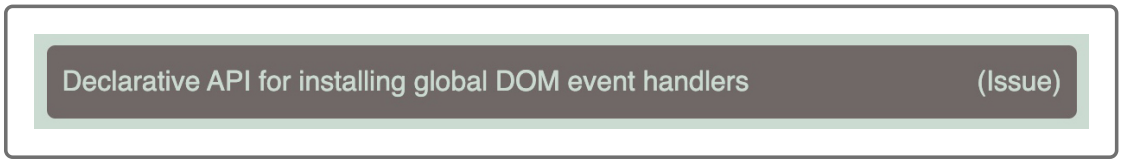
If you forget which properties each issue object has, you can check the Preview panel in the DevTools Network tab, load the requested URL in another browser tab, or `console.log()` the response data.

So far, we've only created an empty `<a>` element called `issueEl` for each issue. We still need to add content to these elements. We'll want to display each issue's name as well as its type—either an actual issue or a pull request.

Underneath the `issueEl` element logic, add the following block of code:

```
// create span to hold issue title
var titleEl = document.createElement("span");
titleEl.textContent = issues[i].title;

// append to container
issueEl.appendChild(titleEl);

// create a type element
var typeEl = document.createElement("span");

// check if issue is an actual issue or a pull request
if (issues[i].pull_request) {
  typeEl.textContent = "(Pull request)";
} else {
  typeEl.textContent = "(Issue)";
}

// append to container
issueEl.appendChild(typeEl);
```

This code will create an `<a>` element that will look like the following image:



However, you won't see anything on the page yet, because these `<a>` elements only exist in JavaScript. You still have to append them to the actual page somewhere.

Let's do that now! At the top of `single.js`, create a reference to the issues container:

```
var issueContainerEl = document.querySelector("#issues-container");
```

Then in `displayIssues()`, add the following line right before the `for` loop closes:

```
issueContainerEl.appendChild(issueEl);
```

Save and test the app in the browser. On page load, you should see a list of issues pertaining to the repo that you hardcoded in the function call (e.g., `getRepoIssues("facebook/react")`). Try hardcoding a couple of different repo names to ensure the logic works for other cases.

In your testing, you might come across a repository with no open issues. If the page displays nothing when there are no open issues, users might think your app is broken. Instead, you could check for no issues and display an appropriate message.

At the top of the `displayIssues()` function, add the following `if` statement:

```
if (issues.length === 0) {
  issueContainerEl.textContent = "This repo has no open issues!";
  return;
}
```

This code will display a message in the issues container, letting users know there are no open issues for the given repository.

Test the app a few more times; check that repositories both with and without issues display correctly. Once you like the results, `commit` and `push` your progress to the feature branch!

© 2020 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.