and replaced with JavaScript code that programmatically generates this markup. There is also a loading animation that will need to be displayed/hidden.

2. Examine **ch10-proj01.js** in the editor of your choice. In it, you will see the URL for the external API that will provide the color scheme data. Examine this URL in the browser in order to see the structure of the data.
3. Fetch this scheme data from the API and display it within the `<article>` element. As you can see from the sample supplied markup, this will require creating `<h3>`, `<section>`, `<div>`, and `<button>` elements.
4. Display the loading animation before the fetch and then hide it after the data is retrieved.
5. Set up a single click event handler for *all* the View buttons. This will require using event delegation. When the user clicks a view button, display the scheme details in the `<aside>` element. As you can see from the sample supplied markup, this will require creating `<div>` elements within the supplied `<fieldset>`. You will also have to change the `<h2>` content to the clicked scheme name. Hint: use the `find()` method to retrieve the correct scheme object from the `data-id` property of the clicked button. Also, remember to clear out the previous content of the `<fieldset>` by setting its `innerHTML` to `""`.

Guidance and Testing

1. Break this problem down into smaller steps. First verify the fetch works, perhaps with a simple `console.log` statement. Then write a function that generates the markup for a single color scheme in the `<article>` and test to make sure it works. This will require a loop, so try using `forEach()` instead of a `for` loop.
2. Then add in support for the loading animation.
3. Before generating the scheme details, add in the event handler using event delegation and verify (again using `console.log`) if you are able to retrieve the correct scheme object using `find()`.
4. Finally, write a function that generates the scheme details. This will require a loop, so try using `forEach()` instead of a `for` loop.

PROJECT 2: Text Viewer

DIFFICULTY LEVEL: Intermediate

Overview

This project focuses on the first two sections of the chapter (array functions and prototypes/classes/modules). It also uses fetch to retrieve data. Figure 10.28 indicates what the final result should look like in the browser.

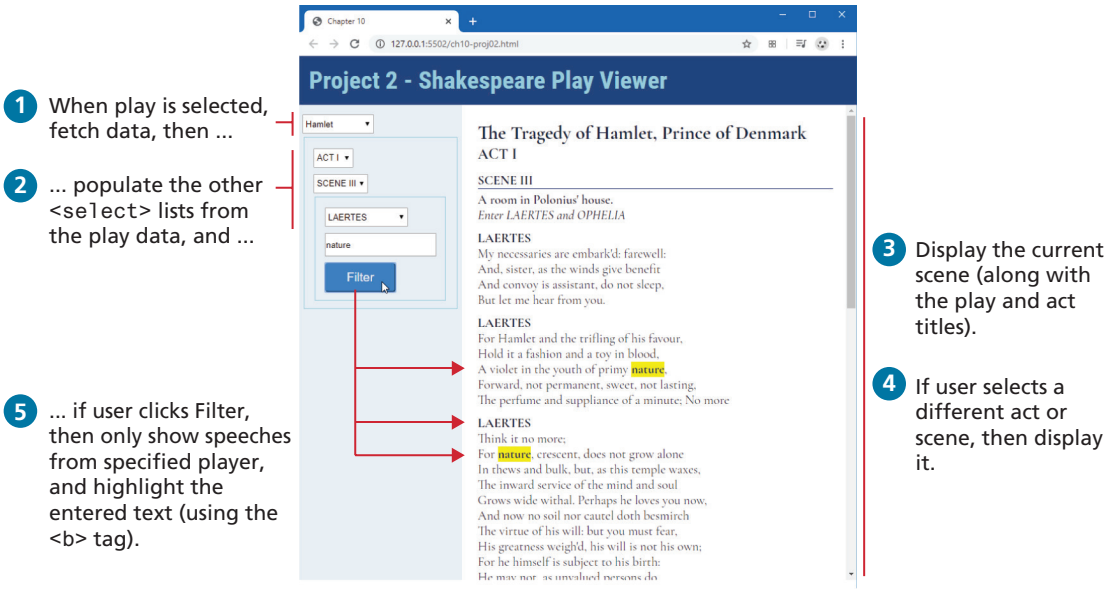

FIGURE 10.28 Completed Project 2

Instructions

1. You have been provided with the necessary styling and markup already. Examine **ch10-proj02.html** in the editor of your choice. Notice the containers for the fetched data in the `<aside>` and `<section>` elements. Notice the sample markup for the play data. This will be eventually commented out and replaced with JavaScript code that programmatically generates this markup.
2. Examine **ch10-proj02.js** in the editor of your choice. In it, you will see the URL for the external API that will provide the color scheme data. Examine this URL in the browser in order to see the structure of the data. A Shakespeare play contains multiple acts; each act contains multiple scenes. (To reduce the size of the downloaded files, not all acts and scenes have been included).
3. Add a change event handler to the first `<select>`, which contains a preset list of plays. When the user selects a play, fetch the play data by adding the `value` attribute of the `<option>` for the play as a query string, as shown in the comments in **ch10-proj02.js**. When the fetched play is retrieved, populate the three other `<select>` elements from this data. Also populate the `<section id="playHere">`, `<article id="actHere">`, and `<div id="sceneHere">` elements with the first scene from the first act of the selected play.

4. To make the code more manageable, create classes named `Play`, `Act`, and `Scene`, which will be responsible for outputting the relevant DOM elements. Using object-oriented techniques, the `Play` class will contain a list of `Act` objects, the `Act` class will contain a list of `Scene` objects, while the `Scene` class will contain a list of speeches. These classes will reside within a JavaScript module named **play-module.js**.
5. Add event handlers to the other `<select>` elements. They will change what part of the play is displayed.
6. The filter button will highlight all occurrences of the user-entered text in the play and only show the speeches from the specified player.

Guidance and Testing

1. Break this problem down into smaller steps. First verify the fetch works, perhaps with a simple `console.log` statement. Then populate the `<select>` lists based on the fetched data.
2. You may decide to move your code into classes within your module after you finished your code, or you may decide to work with classes and modules right from the start. This latter approach was that used by the author.

PROJECT 3: Stock Dashboard

DIFFICULTY LEVEL: Advanced

Overview

This project focuses on browser and external APIs. It also uses fetch, classes, and modules. Figure 10.29 indicates what the final result should look like in the browser.

Instructions

1. You have been provided with the necessary styling and markup already. Examine **ch10-proj03.html** in the editor of your choice. Examine **ch10-proj03.js**. In it, you will see the URL for the external API that will provide the color scheme data. Examine this URL in the **browser** in order to see the structure of the data.
2. Create a class named `CompanyCollection` within the module **companies.js**. This class will have the responsibility of fetching the data and displaying it in `<ul id="companiesList">`. Each `<li>` will need to contain the stock symbol value via the `data-id` attribute.
3. Add a single click event handler for this `<ul>`. This will require event delegation. When the user clicks a list item, then display the following information in the four different `<article>` boxes: the company information, the latitude and longitude of the company using Google Maps, and the financial information. Not every company has financial information, so your code has to handle that possibility.