LAB 13b

Intermediate Node

What You Will Learn

- How to use the EJS templating system

Note

This chapter's content has been split into two labs: Lab13a and Lab13b.

Approximate Time

The exercises in this lab should take approximately 15 minutes to complete.

Fundamentals of Web Development, 3rd Ed

Randy Connolly and Ricardo Hoar

Textbook by Pearson
http://www.funwebdev.com

Date Last Revised: March13, 2023

Preparing Directories

1 The starting `lab13b` folder has been provided for you (within the zip folder downloaded from Gumroad).

Intermediate Topics Overview

In this lab, you will be covering a variety of more advanced Node techniques. The first of these is how to integrate a templating system into Node so that it can be used in a way similar to PHP.

Exercise 13b.1 — Setting Up Node Project

1 Navigate to your working folder and enter the following command.

```
npm init
```

2 Install several packages at once via the following command.

```
npm install express
```

3 Install even more packages via the following command.

```
npm install ejs
```

This installs our view/template engine.

4 Examine `scripts/api-router.js`. Notice that it defines three routes.

5 Run `book-server.js` and then test in browser with the following requests.

```
http://localhost:8080/api/all
http://localhost:8080/api/isbn10/0321865820
http://localhost:8080/api/title/calc
http://localhost:8080/static/images/0132145375.jpg
```

Adding a View/Template Engine

In the first Node lab, you used Node as a file and API server. You may have wondered if Node can be used in a way similar to PHP. The answer is yes. It is possible to make use of a view engine to programmatically render HTML in Node. Perhaps the most popular of these are **EJS** (Embedded JavaScript templating) and **Pug** (formerly called Jade).

The way a view engine works is that you create your views using some specialized format that contains presentation information plus JavaScript coding (this file is the *template*). This template file is somewhat analogous to PHP files in that they are a blend of markup and programming).

With EJS, you specify your presentation in `.ejs` files, which uses regular HTML with JS embedded within `<% %>` tags. An EJS view has a similar feel to PHP in that you can mix markup and programming code (except the programming language with EJS is JavaScript).

Express has built-in support for view engines. You only need to install the appropriate package using npm, and then tell Express which folder contains the view files and which view engine to use to display those files.

Exercise 13b.2 — Introducing EJS

1. Examine `public/layout.html` and `public/single-book.html`. These files provide the basic markup you will be implementing in EJS.

2. Create a new folder in your application named `views`.

   This folder is going to contain your .ejs files.

3. Create a folder within `views` named `partials`.

4. Create a new file named `head.ejs` within the `partials` folder.

5. Copy and paste the contents of the `<head>` section from `layout.html` into this file.

6. Create a new file named `header.ejs` within the `partials` folder. Copy and paste the `<header>` element from `layout.html` into this file.

7. Create a new file named `sidebar.ejs` within the `partials` folder. Copy and paste the `<section>` element with the class of `sidebar` from `layout.html` into this file.

8. Create a new file named `footer.ejs` within the `partials` folder. Copy and paste the `<footer>` element from `layout.html` into this file.

9. Create a new file within `views` named `home.ejs`.

10 Add the following content to `home.ejs`.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <%- include('./partials/head'); %>
</head>
<body>
   <%- include('./partials/sidebar); %>
   <main class="book">
      <%- include('./partials/header'); %>
      <section class="pagecontent">
         Home content here
      </section>
   </main>
   <%- include('./partials/footer'); %>
</body>
</html>
```

11 Modify `book-server.js` as follows.

```
/* --- middle ware section --- */

// view engine setup
app.set('views', './views');
app.set('view engine', 'ejs');
```

12 Modify `book-server.js` as follows.

```
/*--- add in site page requests ----*/

app.get('/', (req, res) => {
    res.render('home.ejs');
});
```

13 Re-run `book-server.js` and test by requesting `http://localhost:8080/`

This should display the home page view. In the next exercise you will pass data to the page.

Exercise 13b.3 — Passing Data to an EJS

1 Modify `book-server.js` as follows.

```
app.get('/', (req, res) => {
    res.render('home.ejs',
        { data1: 'hello', data2: 'world'} );
});
```

2 Modify `home.ejs` as follows.

```
<section class="pagecontent">
   <%= data1 %> <%= data2 %>
</section>
```

3 Re-run `book-server.js` and test by requesting `http://localhost:8080/`

The page should now display the passed in data.

4 Add a new route by modifying `book-server.js` as follows.

```
app.get('/site/list', (req, res) => {
    res.render('list.ejs', { books: controller.getAll() } );
});
```

The getAll() method returns an array of book objects.

5 Make a copy of `home.ejs` named `list.ejs`.

6 Change the pagecontent `<section>` of `list.ejs` as follows:

```
<section class="pagecontent">
   <h1>List of Books</h1>
   <div class="booklist">
      <% books.forEach(book => { %>
         <div>
            <a href="/site/book/<%= book.isbn10 %>">
               <img src="/static/images/<%= book.isbn10 %>.jpg"
                    width="100" />
            </a>
         </div>
         <h3><%= book.title %></h3>
      <% }); %>
   </div>
</section>
```

Notice that this page loops through the passed book data and outputs the relevant markup.

7 Re-run `book-server.js` and test by requesting `http://localhost:8080/site/list`

The results should look similar to Figure 13b-1.

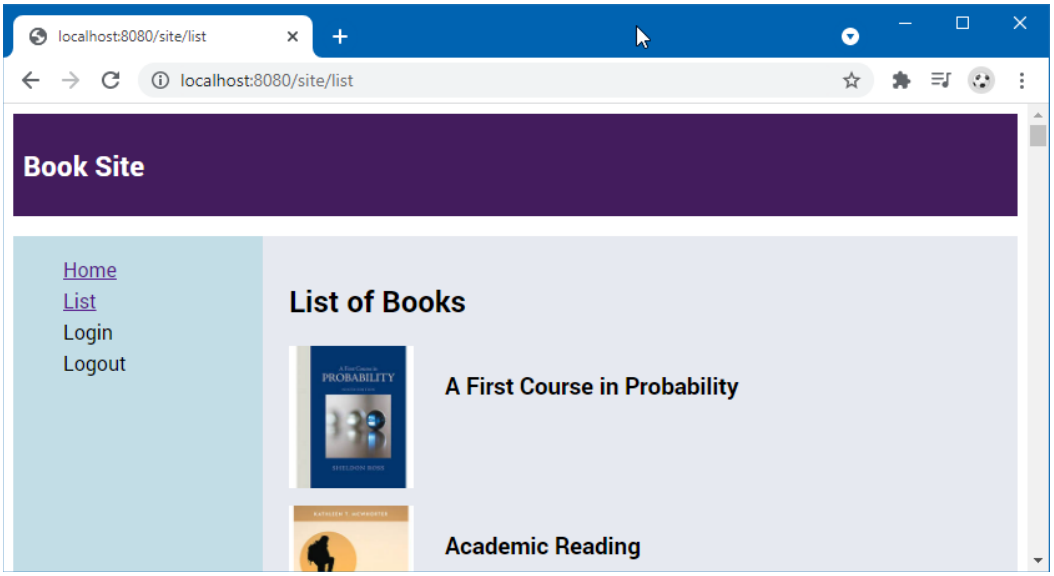

Figure 13.1 – The list page

5 Make a copy of `home.ejs` named `book.ejs`.

6 Change the pagecontent `<section>` of `book.ejs` as follows:

```
<section class="pagecontent">
   <article class="book">
      <div>
         <img src="/static/images/<%= book.isbn10 %>.jpg"  />
      </div>
      <div>
         <h1><%= book.title %></h1>
         <p>Year: <span><%= book.year %></span></p>
         <p>Publisher: <span><%= book.publisher %></span></p>
         <p>Pages: <span><%= book.production.pages %></span></p>
         <p>Categories: <span><%= book.category.main %>,
             <%= book.category.secondary %></span></p>
      </div>
   </article>
</section>
```

7 Add a new route by modifying `book-server.js` as follows.

```
app.get('/site/book/:isbn', (req, res) => {
    res.render('book.ejs', { book:
       controller.findByISBN10(req.params.isbn) } );
});
```

8 Re-run `book-server.js` and test by requesting `http://localhost:8080/site/list`. Click on any of the book cover images. The results should look similar to Figure 13b-2.

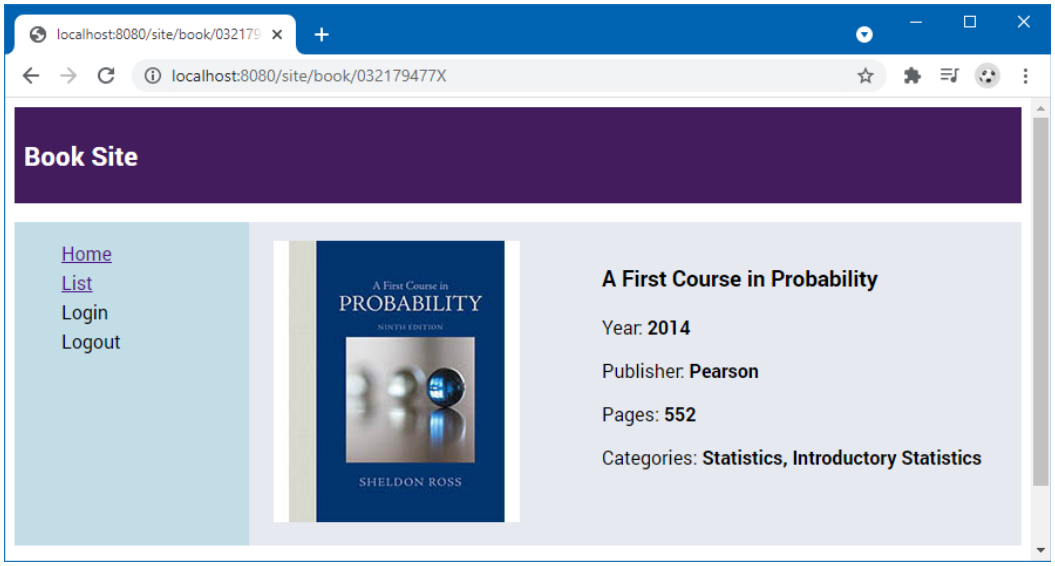

Figure 13.2 – The book page