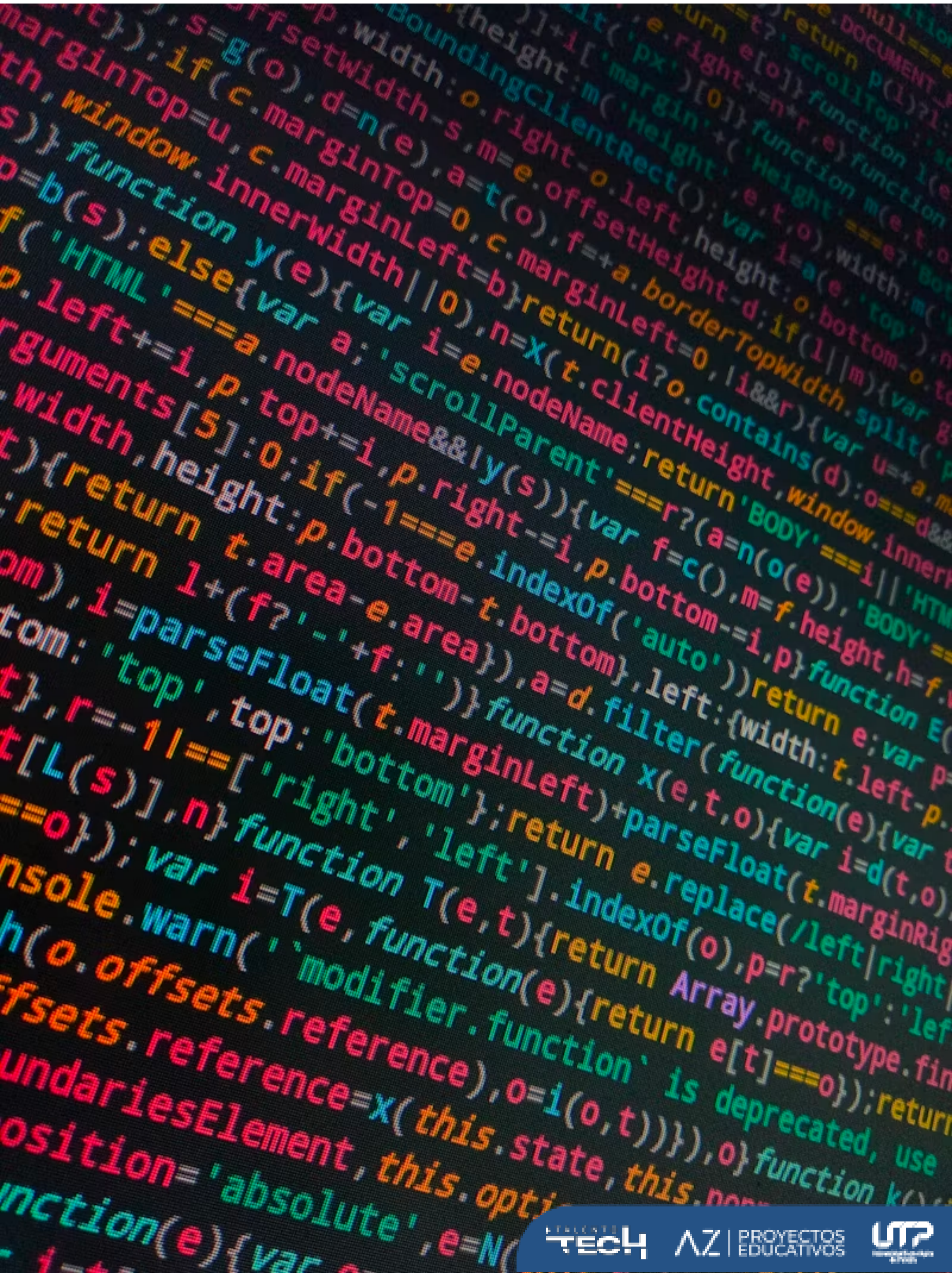
COLOMBIA POTENCIA DE LA VIDA

TIC

Lección 1:

Carga de datos desde archivos .json



TECH | AZ PROYECTOS EDUCATIVOS | UTP

Tiempo de ejecución: 4 horas

Planteamiento de la sesión

Materiales

- JSON Lint
- JSON Data AI
- https://github.com/daduque/innovador_project.git
- https://nodejs.org/docs/v20.11.1/api/path.html#path
- https://nodejs.org/api/fs.html#fsreadfilesyncpath-options

A partir de este punto, los módulos que se van desarrollando estarán enfocados en un proyecto particular que se describe a continuación:

Se creará una tienda en línea enfocada en libros, de cada libro se guardará información como el título, imagen de portada, editorial, categoría, precio, disponibilidad, autor, sinopsis y año de publicación. Toda la información se guardará en un archivo JSON desde donde se leerán y escribirán los datos.

El aplicativo cuenta con otro archivo de usuarios con campos como nombre, correo, contraseña, estado y roles como administradores, clientes y editores.

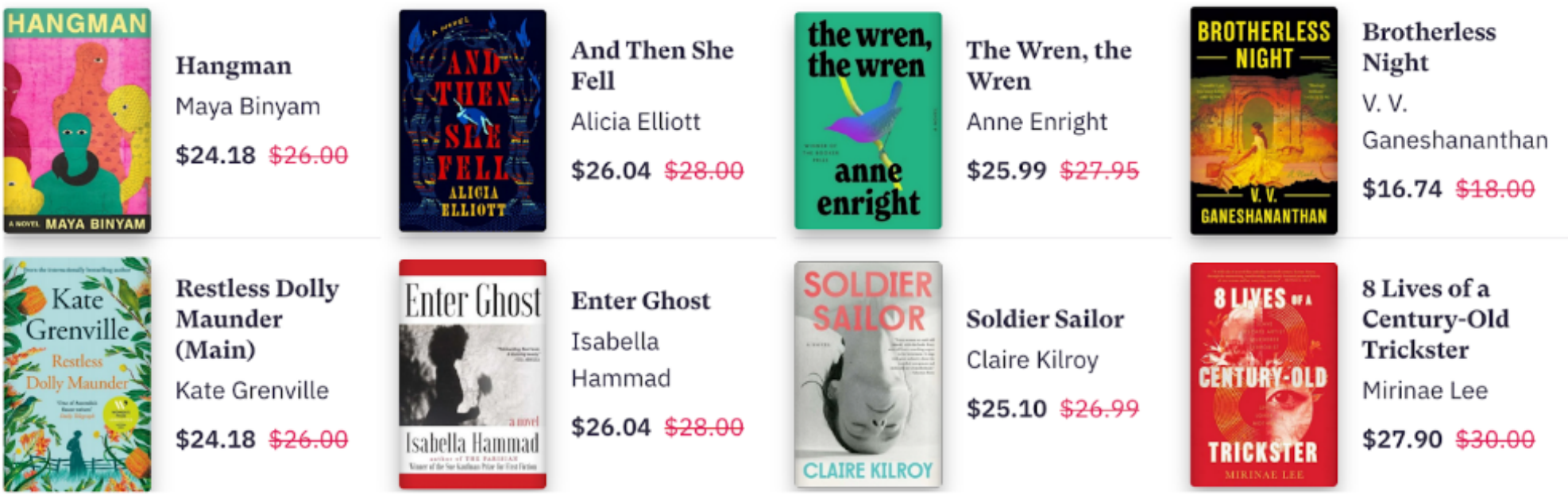
Por ahora no se implementará un rol de cliente ni la categoría o autor como un archivo diferente pero puede tener esta información en mente a la hora de la creación de las bases de datos.

El sistema permite la autenticación de los usuarios con su correo y contraseña, inicialmente no encriptada, proceso que se cubre más adelante, y un middleware que verifica el acceso a determinadas rutas dependiendo del rol asignado.

Algunas funciones contarán con registro en un archivo de logs.

Finalmente, por esta etapa, cada libro contará con una imagen de su portada que se almacenará en el servidor utilizando Multer.

La vista pública del sitio contará con un listado de los libros presentados en forma de rejilla como las tiendas online, de 3 o 4 columnas, donde se presentará la portada, el título, autor y precio.



Fuente: Bookshop

Teniendo en mente el proyecto a realizar uno de los primeros elementos que se deben recordar es el formato JSON, ya que inicialmente los datos los tendremos, como se indica en la descripción, en archivos con este formato.

Para esta labor, vamos a utilizar 2 herramientas, una que nos ayuda con el formateo de los datos para que sean de fácil lectura, adicional a que nos valida que el json que estemos creando tenga un formato válido, se llama JSON Lint . La segunda herramienta de es una IA generadora de data según temática y campos que les especifiquemos, llamada JSON Data AI

JSON

Cómo se estudió en las lecciones de JS, podemos encontrar información sobre JSON en sitios como Formato JSON - Javascript en español - Lenguaje JS, ya que es una notación o formato de texto para el intercambio de información muy popular en el mundo de las tecnologías hoy en día. Los objetos json se definen entre llaves “{}” y se basa en un par clave o llave y valor.

```
{
  "name": "Manz",
  "life": 3,
  "totalLife": 6
  "power": 10,
 "injuries": null,
  "dead": false,
  "props": ["invisibility", "coding", "happymood"],
  "senses": {
 "vision": 50,
 "audition": 75,
 "taste": 40,
 "touch": 80
  }
}
```

Teniendo en mente el proyecto a realizar uno de los primeros elementos que se deben recordar es el formato JSON, ya que inicialmente los datos los tendremos, como se indica en la descripción, en archivos con este formato.

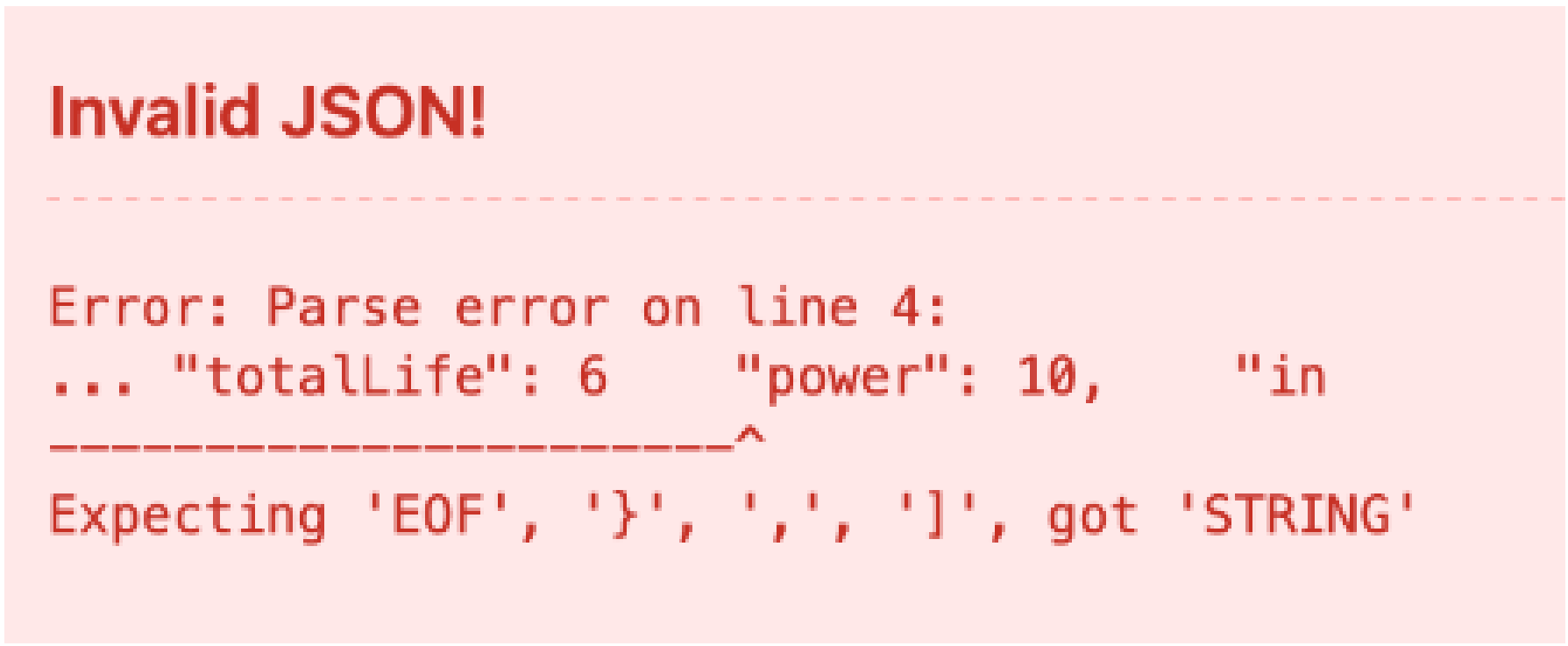
Para esta labor, vamos a utilizar 2 herramientas, una que nos ayuda con el formateo de los datos para que sean de fácil lectura, adicional a que nos valida que el json que estemos creando tenga un formato válido, se llama JSON Lint . La segunda herramienta de es una IA generadora de data según temática y campos que les especifiquemos, llamada JSON Data AI

JSON

Cómo se estudió en las lecciones de JS, podemos encontrar información sobre JSON en sitios como Formato JSON - Javascript en español - Lenguaje JS, ya que es una notación o formato de texto para el intercambio de información muy popular en el mundo de las tecnologías hoy en día. Los objetos json se definen entre llaves “{}” y se basa en un par clave o llave y valor.

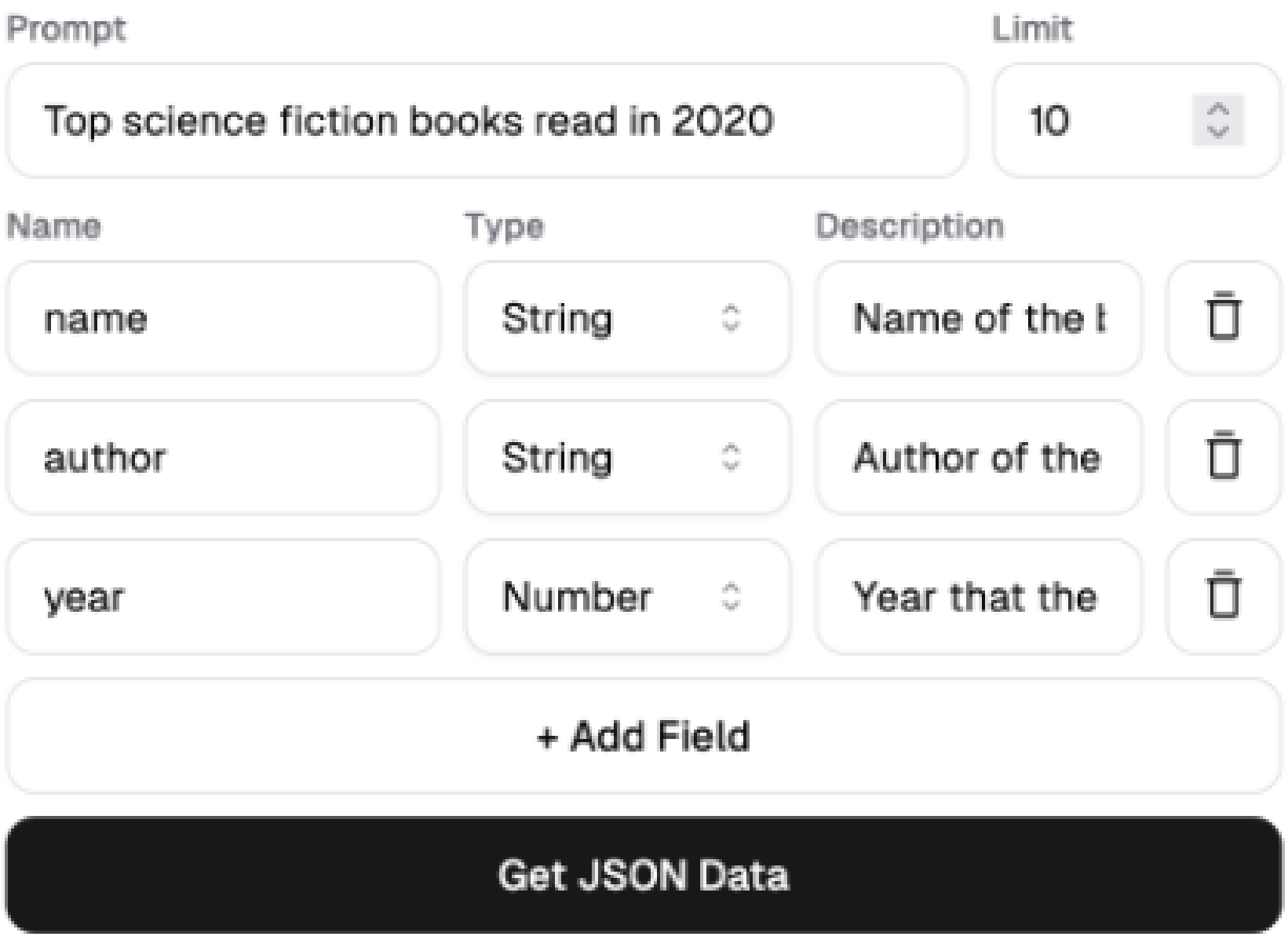
```
{
  "name": "Manz",
  "life": 3,
  "totalLife": 6
  "power": 10,
 "injuries": null,
  "dead": false,
  "props": ["invisibility", "coding", "happymood"],
  "senses": {
 "vision": 50,
  "audition": 75,
 "taste": 40,
 "touch": 80
  }
}
```

Admite diferentes tipos de datos como Arrays, Objetos, Números, Booleanos, Cadenas e incluso Nulos como se presenta en el ejemplo anterior, ahora, para corroborar que el json tenga un formato válido, para ello dentro de la web de JSON Lint vamos a copiar y pegar el código, al dar clic sobre validar debe saltar un mensaje de error indicando un problema de final de archivo o línea en el renglón 4, esto es debido a que falta una coma.



Al corregir y agregar la coma, el mensaje al validar el JSON debe ser

Ahora imaginemos que quiero generar datos de una temática para comenzar a trabajar en mi sitio web, por ejemplo datos de 10 libros de ciencia ficción indicando el título, autor y año, tal como aparecen los valores por defecto del ejemplo de la herramienta json data ai.



El prompt, la cantidad de datos, los campos y los tipos de datos que se requieran para cada proyecto, pueden ser modificados según necesidad. El ejemplo por defecto, nos da como salida la siguiente lista:


```
[
 {
"name": "Dune",
"author": "Frank Herbert",
"year": "1965"
 },
 {
"name": "Ender's Game",
"author": "Orson Scott Card",
"year": "1985"
 },
 {
"name": "The Hitchhiker's Guide to the Galaxy",
"author": "Douglas Adams",
"year": "1979"
 },
 {
"name": "1984",
"author": "George Orwell",
"year": "1949"
 },
 {
"name": "Brave New World",
"author": "Aldous Huxley",
"year": "1932"
 },
 {
"name": "Foundation",
"author": "Isaac Asimov",
"year": "1951"
 },
 {
"name": "Neuromancer",
"author": "William Gibson",
"year": "1984"
 },
```

```
{
 "name": "The War of the Worlds",
 "author": "H.G. Wells",
 "year": "1898"
  },
  {
 "name": "Snow Crash",
 "author": "Neal Stephenson",
 "year": "1992"
  },
  {
 "name": "The Martian",
  "author": "Andy Weir",
  "year": "2011"
   }
]
```


Para el proyecto de la librería vamos a ir ajustando la data hasta que tengamos un resultado como el planteado al inicio, agregando los campos faltantes y buscando las portadas de los libros.

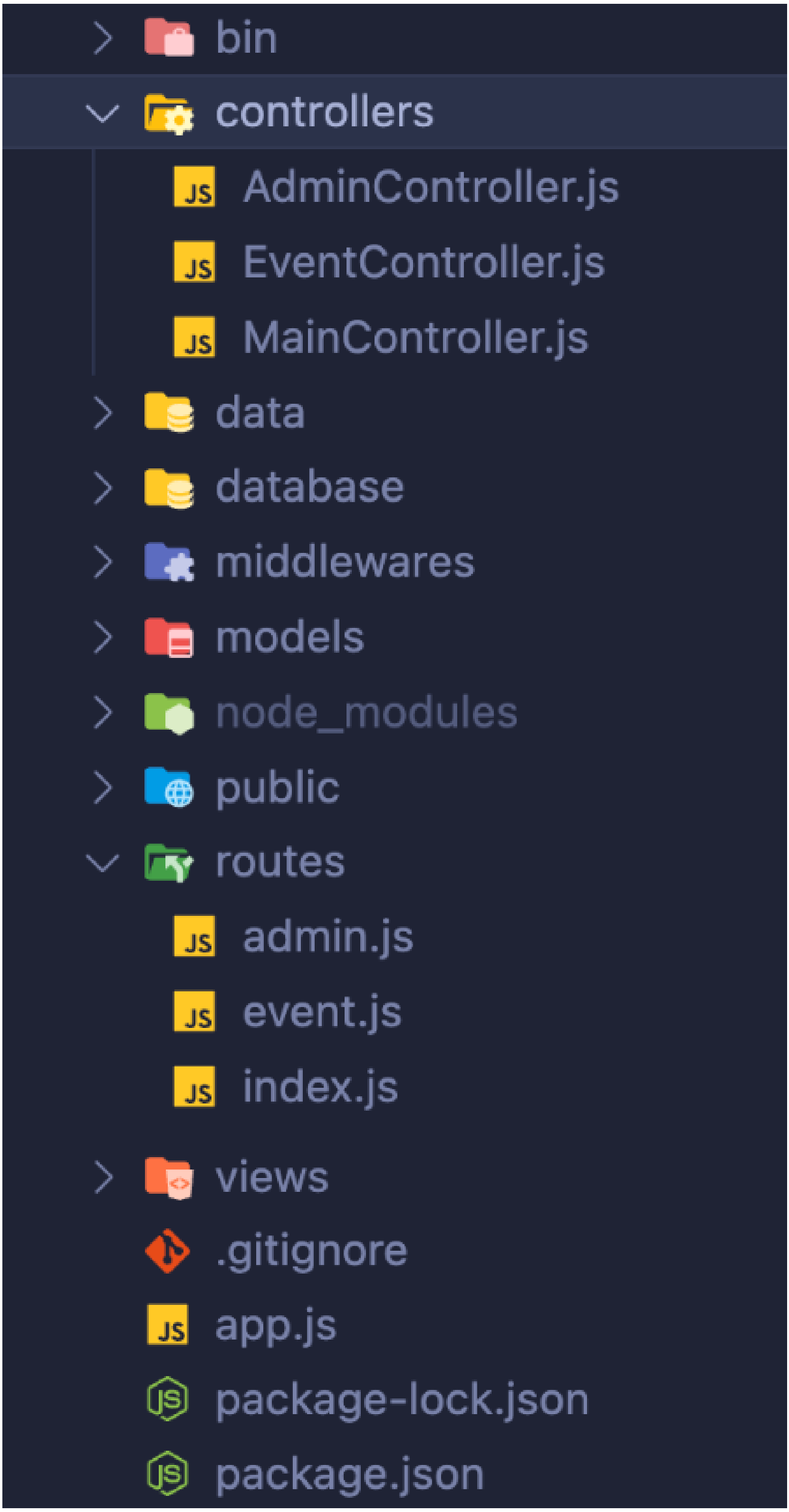
Codificando

Por otro lado, se creará el proyecto en Express con un par de componentes adicionales, uno será que adicional al enrutador principal, se creará un enrutador por cada bloque de funcionalidades, es decir que lo que corresponda a libros tendrá uno, que será diferente a lo que corresponda con usuarios y así sucesivamente.

El otro cambio será agregar la carpeta de los controladores, en donde escribiremos lo que queremos hacer con la data y a qué vista la enviaremos.

En las lecciones siguientes también se agregarán carpetas adicionales como las de logs o middlewares, por ahora solo nos interesa fijarnos en routers, controllers y data (donde se guardan los archivos JSON).

Al final de esta unidad se tendrá una distribución de archivos como la que se presenta a continuación:


bin
controllers
AdminController.js
EventController.js
MainController.js
data
database
middlewares
models
node_modules
public
routes
admin.js
event.js
index.js
views
.gitignore
app.js
package-lock.json
package.json

Aprovechando los conocimientos en Git adquiridos hasta el momento, el proyecto estará en diferentes etapas en diferentes ramas, para poder visualizar el avance entre una etapa y la otra sin afectar el funcionamiento de cada versión.
Proyecto base

A continuación se comparte el proyecto desde el cual se inicia el proceso de desarrollo, el proyecto es generado por Express Generator añadiendo las carpetas mencionadas e inicialmente un enrutador, un controlador y la data en formato JSON para presentar los libros.

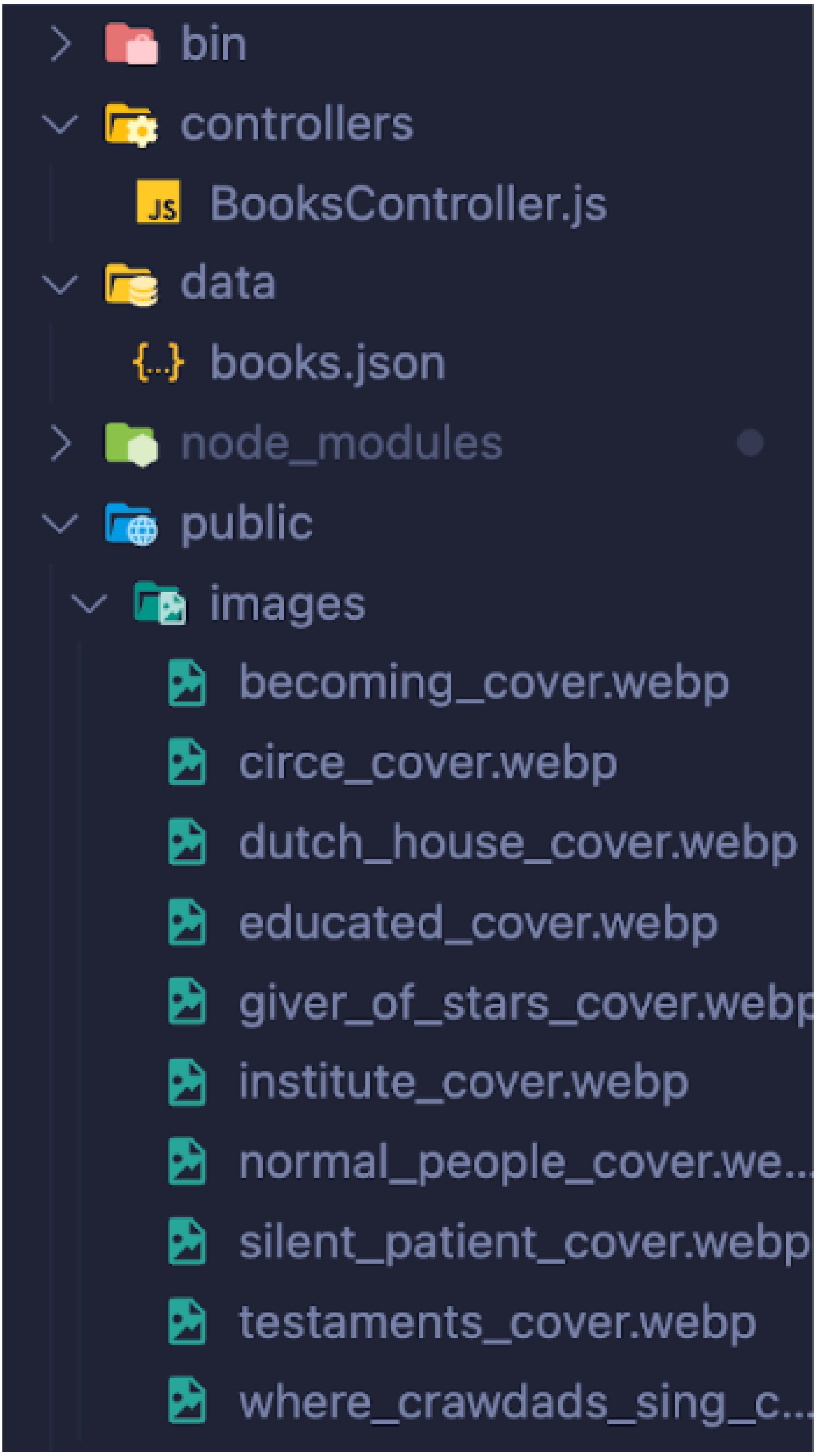
El proyecto tiene como adicional una instalación de un paquete llamado nodemon, que permite un reinicio automático del servidor cuando detecta un cambio, ahorrando así un poco de tiempo en estar apagando e iniciando el servidor para visualizar las modificaciones.

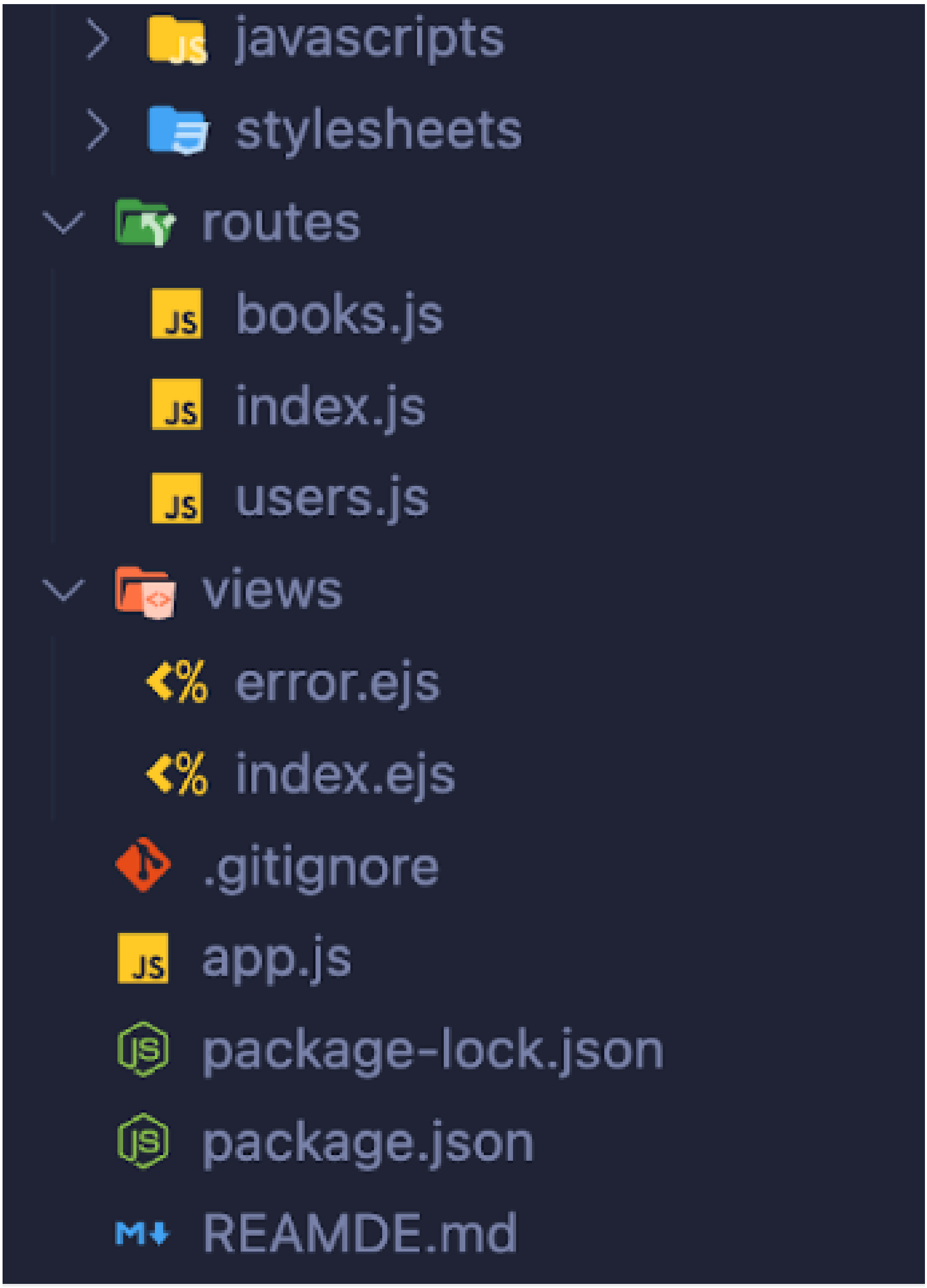
La instalación, como se indica en la documentación oficial, solo requiere del siguiente comando, nodemon - npm

npm install --save-dev nodemon

```json
{
  "name": "m2",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www",
    "dev": "nodemon ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "ejs": "^3.1.9",
    "express": "^4.18.3",
    "http-errors": "~1.6.3",
    "morgan": "~1.9.1"
  },
  "devDependencies": {
    "nodemon": "^3.1.0"
  }
}
```

Por lo demás, se han creado las carpetas indicadas en la descripción las carpetas del proyecto se deben ver similar a la siguiente imagen:





https://github.com/daduque/innovador_project.git

Path y FileSystem

Los módulos path y file system son propios de node y permiten, por un lado path lo utilizamos habitualmente en 3 casos, el primero es path.join(), que permite concatenar cadenas para formar una ruta de un directorio o un archivo final.

```
const path = require('path');

const archivoAdmins = path.join('registros', 'usuarios', 'administradores.json');
// de este modo se guarda la ruta completa dentro de una variable.
```

Otra de las formas es para obtener la extensión de un tipo de archivo, por ejemplo .json del ejemplo anterior,, el método es path.extname(archivoAdmins)

```
const extension = path.extname(archivoAdmins);

// Returns: '.json'

path.extname('index.html');
// Returns: '.html'

path.extname('index.coffee.md');
// Returns: '.md'

path.extname('index.');
// Returns: '.'

path.extname('index');
// Returns: ''

path.extname('.index');
// Returns: ''

path.extname('.index.md');
// Returns: '.md'
```

El otro método es el path.dirname(), que retorna la ruta de directorios.

```
const directorio = path.dirname(archivoAdmins);
// Returns: '/registros/usuarios'

path.dirname('/foo/bar/baz/asdf/quux');
// Returns: '/foo/bar/baz/asdf'
```

Y por el otro el File System, nos permite interactuar con el sistema de archivos de la máquina o el proyecto en particular, cuentan con funciones que son asíncronas y otras que son síncronas, es decir que tiene la opción de trabajar con promesas o sin promesas. Vamos a revisar los métodos Sync, los cuales no necesitan promesas a la hora de leer o escribir en los archivos.

La primera función es fs.writeFileSync(), que recibe el archivo en donde queremos escribir (cuya ruta puede ser definida con path) y la información a escribir, este método siempre sobreescribe el contenido del archivo que se abre, por lo que sólo se conservaría el último registro escrito.

```
let persona ={
 nombre : 'Roberta',
 email: 'roberta@gmail.com',
 contraseña : 'minjuegos'
};

let personaJSON = JSON.stringify(persona);
// stringify nos permite convertir el objeto literal, en formato
// .json

fs.writeFileSync('persona.json', personaJSON);
// combinandolo podemos guardar mucha info.
```

Si se quiere conservar la información anterior del archivo, entonces el método que se debe utilizar es fs.appendFileSync(), que funciona igual pero agregando la nueva data al final del archivo.

```
fs.appendFileSync('persona.json', personaJSON);
```

Las funciones tienen su versión utilizando promesas o try/catch, la función asíncrona de writeFileSync() es writeFile() y la de appendFileSync() es appendFile().

La diferencia principal es que las funciones síncronas bloquean la ejecución del código siguiente, es decir, que hasta que no finalice la lectura o escritura del archivo el programa no continúa, mientras que la versión asíncrona ejecuta el código siguiente mientras se resuelve de manera exitosa o no la petición y cuando se obtiene la respuesta, notifica y ejecuta lo pendiente.

El tercer método utilizado a la hora de trabajar con el FS es el fs.readFileSync(), que recibe la ruta y nombre del archivo a leer, nuevamente la ruta puede ser definida utilizando path.

```
const fs = require('fs');

let texto = fs.readFileSync('bienvenida.txt', {encoding : 'utf-8'});
```

Estos métodos son las bases para trabajar en la siguiente lección.

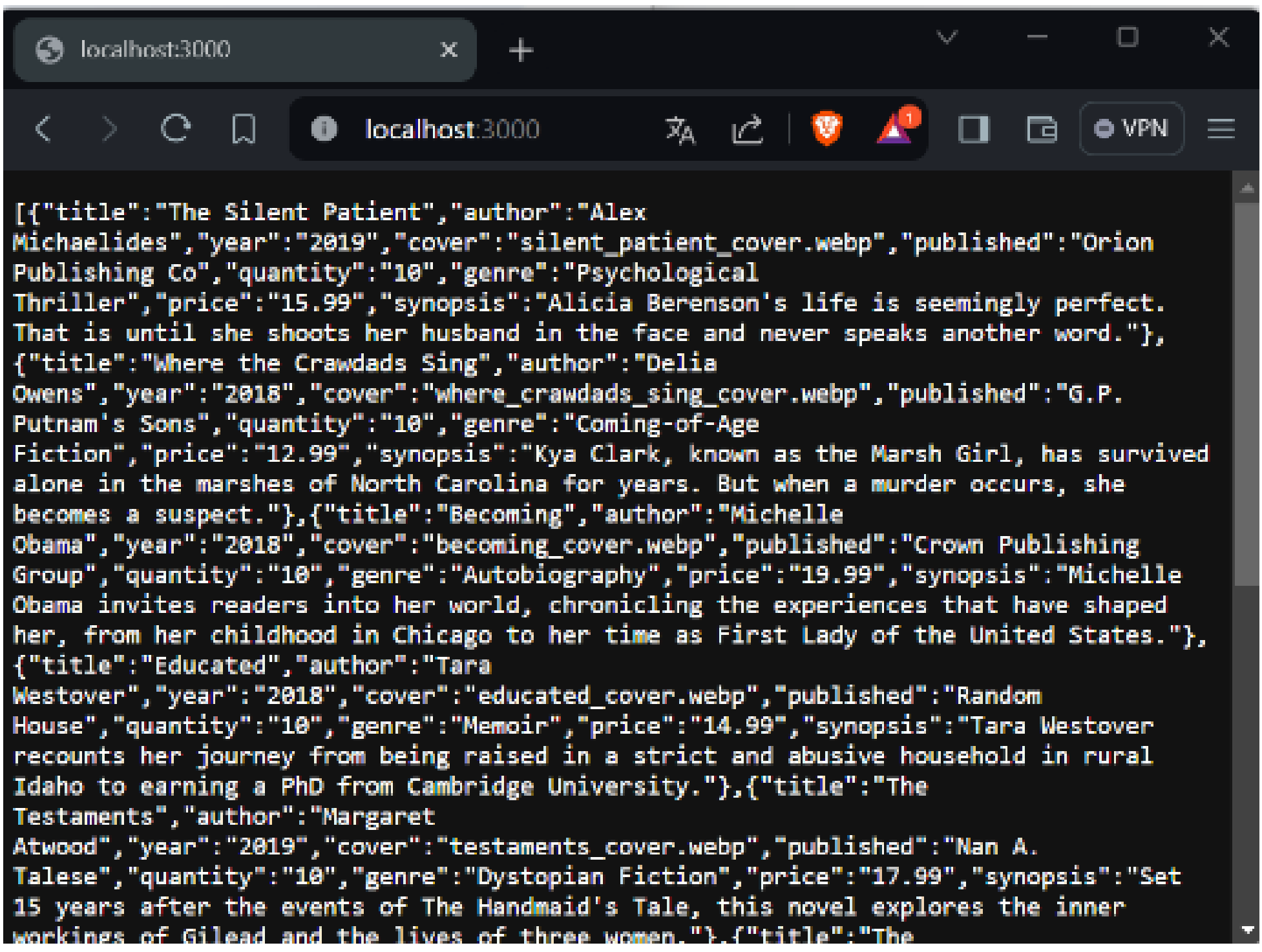
Ejemplo

```
// En .controllers/UsuariosController.js
const fs = require('fs');
//acá pedimos el paquete file system de express

const controller = {

    guardarUsuario : (req, res) =>{
     let usuario = {
      nombre: req.body.nombre,
      edad: req.body.edad,
      email: req.body.email,
     }

     //En el medio hay que GUARDARLA.
     // queremos que 'usuarios.json' guarde un ARRAY con la lista de usuarios
     // y cada elemento será un objeto literal con cada elemento de usuario

          // PRIMERO queremos leer qué cosas ya había: si ya tenía usuarios registrados, no
     // queremos pisar el archivo

     let archivoUsuario = fs.readFileSync('usuarios.json', {encoding : 'utf-8'});
 let usuarios;
     //definimos la variable antes del if así la toma globalmente
if(archivoUsuario == ''){
       usuarios = [];
     } else {
       usuarios = JSON.parse(archivoUsuario);
     // Para leer un archivo .json usamos parse para descomprimir la información.
```

```
// De este modo vamos a tener la variable usuarios como un array, con todos
    // los usuarios viejos. Si el archivo está vacío, de manera previa vamos a
    // escribir una condición.
    // O sea, si el archivo no tenía nada, va a ser un array vacío, y si ya tenía
        // contenido, ahí va a descomprimir el archivo para obtener el array de
usuarios.
    };

    usuarios.push(usuario);
    // Ahora le agregamos el usuario nuevo (el que se creo arriba con los dtos del
form)
    // ahora para poder escribirla hay que pasarla a JSON de nuevo, con stringify.

    let usuariosJSON = JSON.stringify(usuarios);

    fs.writeFileSync('usuarios.json', usuariosJSON);

    res.redirect('/users/list');
    //redireccionamiento
   },
};

module.exports = controller;
```

Ahora vamos a aplicar los elementos base en nuestro proyecto, vamos a realizar una petición a la raíz del proyecto es decir a la ruta principal, en este caso sería localhost:3000, lo que queremos, inicialmente es que el servidor conteste con el listado de los libros, inicialmente imprimir el json en pantalla y posteriormente recorrerlo en la vista y con bloques de EJS recorrerlo e imprimirlo.

Así entonces el primer reto se debe ver algo así:



La ruta de la petición es la siguiente:

el entry point (app.js), recibe la petición y consulta si tiene un método disponible para atenderla, en este caso la petición va a la raíz (/), la tarea es consultar si hay un método con use o get que atienda las peticiones a la raíz.

app.use('/', indexRouter);

En este caso, existe, lo que se indica es que hay un archivo, llamado en la variable indexRouter, que puede atender la petición, al buscar esta variable, vemos que requiere un nuevo archivo llamado index.js dentro de la carpeta routes.

const indexRouter = require('./routes/index');

Al ir al archivo en la ruta nos encontramos con que esté index.js funciona como una especie de índice que lista todas las rutas del aplicativo y que permite guiar a la petición al archivo que puede atender su petición, lo que allí se indica es que la petición debe viajar al archivo books.js dentro de este mismo directorio.

```
const booksRouter = require('./books');

//use the router object to define the routes for the application

//books routes
router.use('/', booksRouter);
```

Ahora el archivo books.js indica que el método index del controlador BooksController es el encargado de atender la petición, nótese que ahora se utiliza el método .get, indicando que en este punto la petición ya se debe resolver.

```
const BooksController = require('../controllers/BooksController');

/* GET home page. */
router.get('/', BooksController.index);
```

Finalmente llegamos al controlador, si revisamos el método index, vemos 3 elementos clave, primero leemos el archivo books.json, esta cadena de texto la guardamos en la variable data, segundo formateamos la data para que pasemos de una cadena de texto a una cadena en formato de objeto literal de JS, un objeto iterable con el que podemos trabajar en el código y finalmente se le envía la respuesta al usuario en el método res.send, con todo este proceso deberíamos obtener la respuesta esperada.

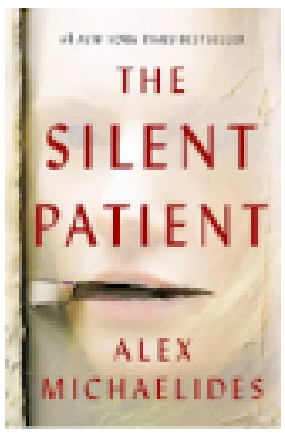
```
index: (req, res) => {
    //read the file books.json from data directory and get the data
    let data = fs.readFileSync(path.join('data', 'books.json'), 'utf8');
    //parse the data to convert it into an array of objects
    let books = JSON.parse(data);
    //send the response to the client
    res.send(books);
}
```

Ahora el segundo reto (la solución está en la rama step2, recuerde que para cambiar entre ramas utiliza el comando git checkout nombre_rama), es presentar de manera correcta el listado de libros, digamos que la imagen, el título y el autor inicialmente. algo que se presente como lo siguiente:

Bootcamp Book Store

Welcome to Bootcamp Book Store

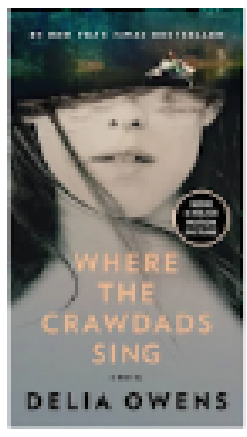
Book Catalog



The Silent Patient

by Alex Michaelides

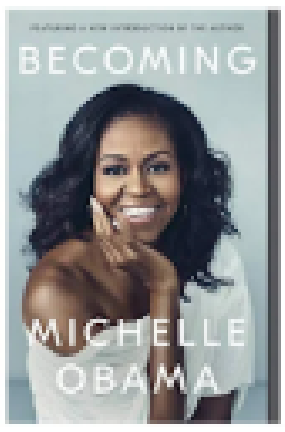
$15.99



Where the Crawdads Sing

by Delia Owens

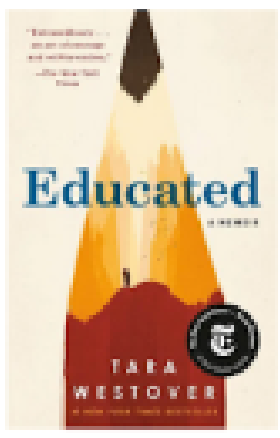
$12.99



Becoming

by Michelle Obama

$19.99



Educated

by Tara Westover

$14.99

The Testaments

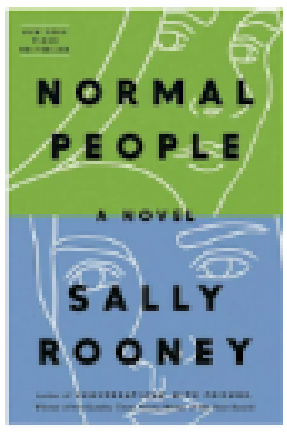
by Margaret Atwood

$17.99

The Institute

by Stephen King

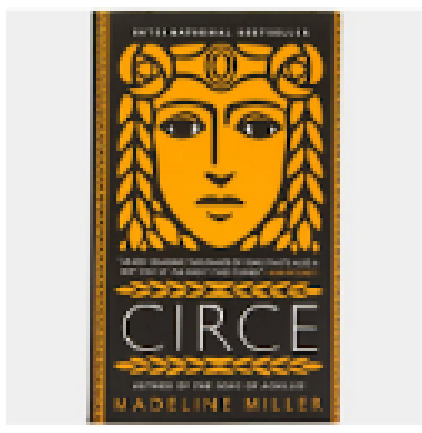
$16.99



Normal People

by Sally Rooney

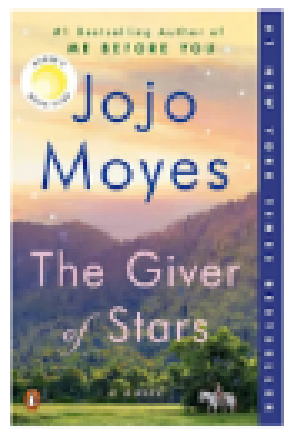
$13.99



Circe

by Madeline Miller

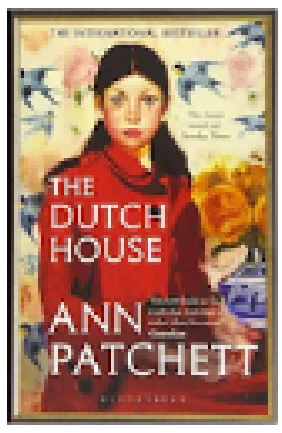
$11.99



The Giver of Stars

by Jojo Moyes

$15.99



The Dutch House

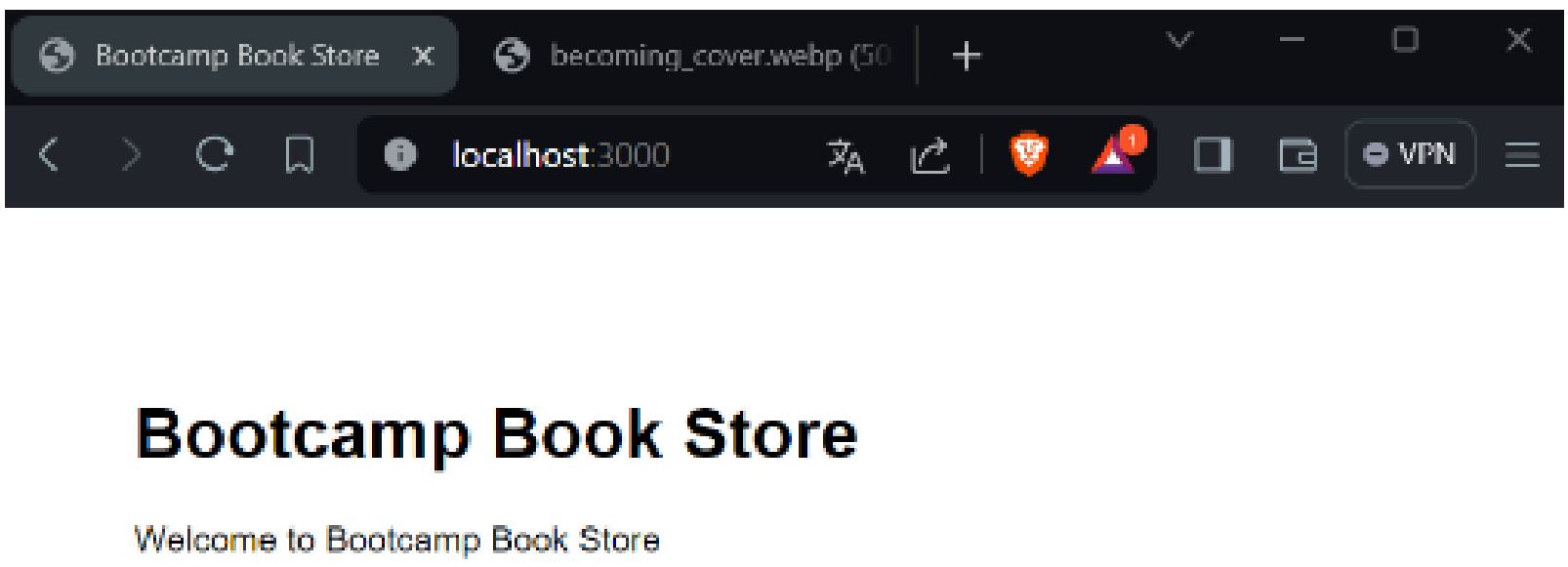
by Ann Patchett

$14.99

Para la primera parte del reto entonces vamos a cambiar el método send del controlador por un método render, para llamar nuevamente a una vista, en este caso la vista index, tal como funcionaba el proyecto al momento de generarlo con Express Generator, luego vamos a revisar si en ese index.ejs hay data que debamos enviar desde el render para que se muestre.

```
<header>
  <h1><%= title %></h1>
      <p>Welcome to <%= title %></p>
</header>
```

Como se aprecia en el código, la vista index requiere de una variable llamada title, la cual debemos o eliminar, borrándose de la vista o enviar desde el controlador, para nuestro caso, se la enviaremos desde el controlador para que se imprima algo como lo siguiente:



el método index del controlador se vería así:

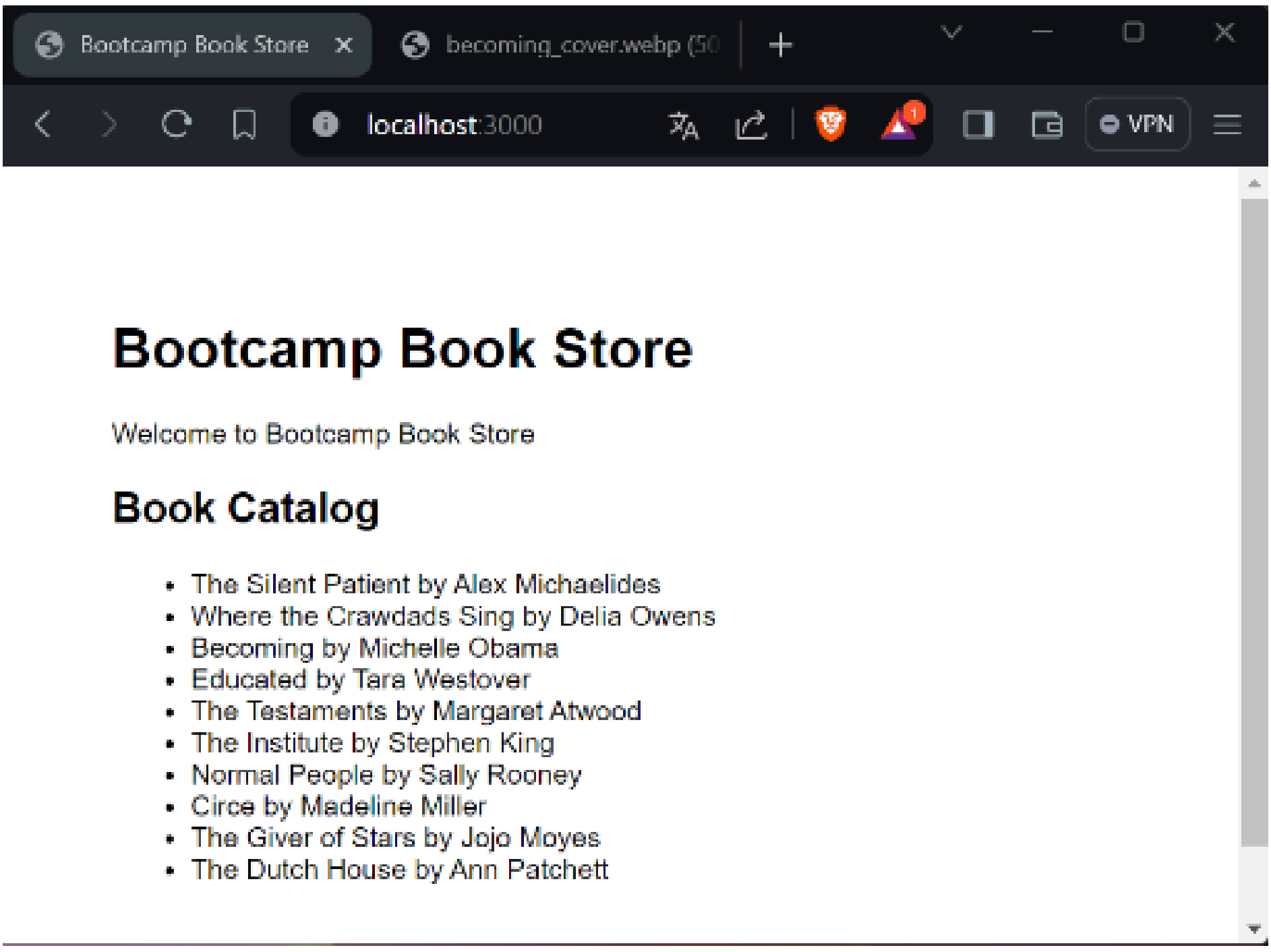
```
//render the index view and pass the title to the view
res.render('index', { title: 'Bootcamp Book Store' });
```

Ahora vamos a enviar el bloque de data, los libros, los cuales se almacenan en un array que contiene objetos literales que vamos a recorrer en la vista para pintarlos, inicialmente solo el título y el autor:

```
//render the index view and pass the title and the books data to the view
res.render('index', { books, title: 'Bootcamp Book Store' });
```

El atajo para generar el bloque y pintarlo en la vista, si se instaló la extensión de EJS en VS code, es EJSEACH, que genera un bloque foreach que nos permite recorrer el array con la data en su interior.

```
<main>
    <h2>Book Catalog</h2>
    <ul>
      <% books.forEach((book) => { %>
        <li><%= book.title %> by <%= book.author %></li>
      <% }); %>
    </ul>
  </main>
```



Ahora solo nos falta renderizar de imagen de cada libro y darle un poco de estilos para ajustarlo, primero pintemos la imagen antes de cada elemento de la lista solo para tener claridad sobre la forma de invocarlas.

Si recordamos la etiqueta img tiene algunos atributos como el src, alt, width y height, la ruta hacía la imagen se asigna en el src y es aquí donde debemos identificar la ruta que debemos generar.

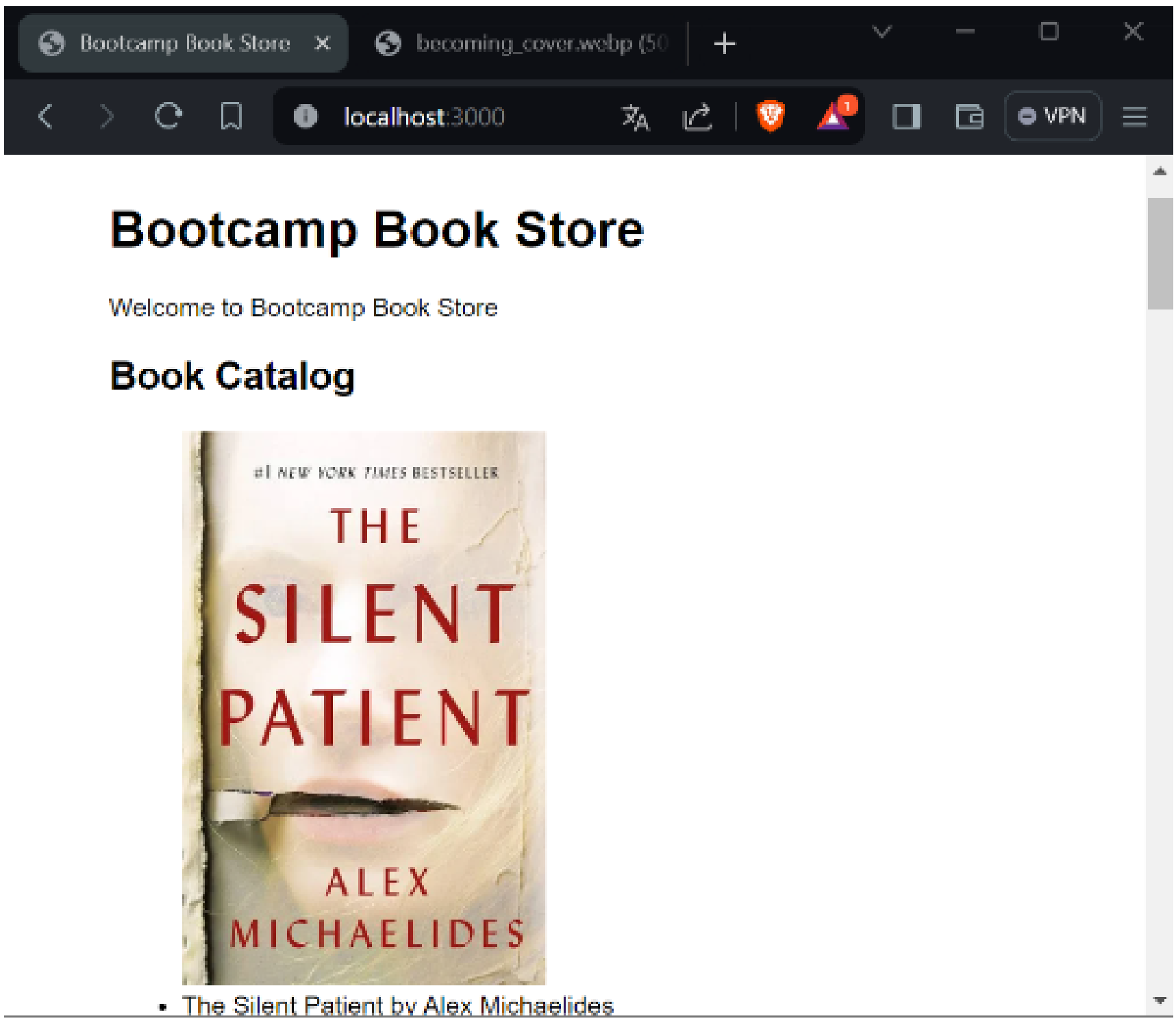
Debemos recordar que las imágenes fueron almacenadas en una carpeta llamada images dentro de la carpeta public, y si vamos al app.js podemos encontrar una línea que define el directorio de recursos estáticos.

```
app.use(express.static(path.join(__dirname, 'public')));
```

Esta definición permite acceder a los recursos almacenados como si se encontraran siempre en la raíz del proyecto, así entonces, para las rutas de las imágenes solo sería necesario indicar la carpeta images/nombre_cover, siendo el nombre_cover el dato que almacenamos dentro del objeto bajo la llave cover:

```
<main>
    <h2>Book Catalog</h2>
    <ul>
      <% books.forEach((book) => { %>
          <img src="images/<%= book.cover %>" alt="<%= book.title %> cover" width="200">
        <li><%= book.title %> by <%= book.author %></li>
      <% }); %>
    </ul>
</main>
```

Obteniendo como resultado, el mismo listado solo que ahora con la portada del libro antes de su título y autor.



Ahora solo falta darle un poco de estilo para que parezca al reto inicialmente planteado, para ello es necesario crear un archivo CSS que permitirá agregar estilos a las diferentes vistas. Dentro de la carpeta stylesheets en public, vamos a agregar más reglas de css al archivo llamado style.css.

Aunque algunas portadas de las descargadas no están en tan buena calidad o cuentan con fondos de color, a modo de ejemplo se ha cumplido el reto de presentar la información base de los libros, incluso agregamos el precio también, sientase libre de presentar la info en el formato que mejor le parezca y ajustar los tamaños a sus gustos.