**[Creación de las funciones de devolución de llamada del controlador de rutas](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes" \l "create_the_route-handler_callback_functions)**

Antes de definir nuestras rutas, primero crearemos todas las funciones de devolución de llamada ficticias/esqueleto que invocarán. Las devoluciones de llamada se almacenarán en módulos "controladores" separados para , , , y (puede usar cualquier estructura de archivo/módulo, pero esta parece una granularidad apropiada para este proyecto).BookBookInstanceGenreAuthor

Comience creando una carpeta para nuestros controladores en la raíz del proyecto (**/controllers**) y luego cree archivos/módulos de controlador separados para manejar cada uno de los modelos:

/express-locallibrary-tutorial //the project root

/controllers

authorController.js

bookController.js

bookinstanceController.js

genreController.js

Los controladores usarán el módulo, por lo tanto, antes de continuar, instálelo en la biblioteca usando:express-async-handlernpm

juergaCopiar en el portapapeles

npm install express-async-handler

[**Controlador de autor**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#author_controller)

Abra el archivo **/controllers/authorController.js** y escriba el siguiente código:

JsCopiar en el portapapeles

const Author = require("../models/author");

const asyncHandler = require("express-async-handler");

// Display list of all Authors.

exports.author\_list = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Author list");

});

// Display detail page for a specific Author.

exports.author\_detail = asyncHandler(async (req, res, next) => {

res.send(`NOT IMPLEMENTED: Author detail: ${req.params.id}`);

});

// Display Author create form on GET.

exports.author\_create\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Author create GET");

});

// Handle Author create on POST.

exports.author\_create\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Author create POST");

});

// Display Author delete form on GET.

exports.author\_delete\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Author delete GET");

});

// Handle Author delete on POST.

exports.author\_delete\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Author delete POST");

});

// Display Author update form on GET.

exports.author\_update\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Author update GET");

});

// Handle Author update on POST.

exports.author\_update\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Author update POST");

});

El módulo primero requiere el modelo que usaremos más adelante para acceder y actualizar nuestros datos, y el contenedor que usaremos para detectar cualquier excepción lanzada en nuestras funciones de controlador de rutas. A continuación, exporta funciones para cada una de las URLs que deseamos manejar. Tenga en cuenta que las operaciones de creación, actualización y eliminación utilizan formularios y, por lo tanto, también tienen métodos adicionales para manejar las solicitudes de publicación de formularios: discutiremos esos métodos en el "artículo de formularios" más adelante. AuthorasyncHandler

Todas las funciones utilizan la función contenedora descrita anteriormente en [Control de excepciones en funciones de ruta](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#handling_exceptions_in_route_functions), con argumentos para la solicitud, la respuesta y el siguiente. Las funciones responden con una cadena que indica que la página asociada aún no se ha creado. Si se espera que una función de controlador reciba parámetros de ruta, estos se emiten en la cadena de mensaje (ver arriba). req.params.id

Tenga en cuenta que, una vez implementadas, es posible que algunas funciones de ruta no contengan ningún código que pueda generar excepciones. Podemos volver a cambiarlas a las funciones "normales" del controlador de rutas cuando lleguemos a ellas.

**Controlador BookInstance**

Abra el archivo **/controllers/bookinstanceController.js** y copie el siguiente código (esto sigue un patrón idéntico al módulo del controlador):Author

JsCopiar en el portapapeles

const BookInstance = require("../models/bookinstance");

const asyncHandler = require("express-async-handler");

// Display list of all BookInstances.

exports.bookinstance\_list = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: BookInstance list");

});

// Display detail page for a specific BookInstance.

exports.bookinstance\_detail = asyncHandler(async (req, res, next) => {

res.send(`NOT IMPLEMENTED: BookInstance detail: ${req.params.id}`);

});

// Display BookInstance create form on GET.

exports.bookinstance\_create\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: BookInstance create GET");

});

// Handle BookInstance create on POST.

exports.bookinstance\_create\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: BookInstance create POST");

});

// Display BookInstance delete form on GET.

exports.bookinstance\_delete\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: BookInstance delete GET");

});

// Handle BookInstance delete on POST.

exports.bookinstance\_delete\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: BookInstance delete POST");

});

// Display BookInstance update form on GET.

exports.bookinstance\_update\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: BookInstance update GET");

});

// Handle bookinstance update on POST.

exports.bookinstance\_update\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: BookInstance update POST");

});

**Controlador de género**

Abra el archivo **/controllers/genreController.js** y copie el siguiente texto (esto sigue un patrón idéntico al de los archivos y):AuthorBookInstance

JsCopiar en el portapapeles

const Genre = require("../models/genre");

const asyncHandler = require("express-async-handler");

// Display list of all Genre.

exports.genre\_list = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Genre list");

});

// Display detail page for a specific Genre.

exports.genre\_detail = asyncHandler(async (req, res, next) => {

res.send(`NOT IMPLEMENTED: Genre detail: ${req.params.id}`);

});

// Display Genre create form on GET.

exports.genre\_create\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Genre create GET");

});

// Handle Genre create on POST.

exports.genre\_create\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Genre create POST");

});

// Display Genre delete form on GET.

exports.genre\_delete\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Genre delete GET");

});

// Handle Genre delete on POST.

exports.genre\_delete\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Genre delete POST");

});

// Display Genre update form on GET.

exports.genre\_update\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Genre update GET");

});

// Handle Genre update on POST.

exports.genre\_update\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Genre update POST");

});

**Controlador de libros**

Abra el archivo **/controllers/bookController.js** y copie el siguiente código. Esto sigue el mismo patrón que los otros módulos del controlador, pero además tiene una función para mostrar la página de bienvenida del sitio: index()

JsCopiar en el portapapeles

const Book = require("../models/book");

const asyncHandler = require("express-async-handler");

exports.index = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Site Home Page");

});

// Display list of all books.

exports.book\_list = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Book list");

});

// Display detail page for a specific book.

exports.book\_detail = asyncHandler(async (req, res, next) => {

res.send(`NOT IMPLEMENTED: Book detail: ${req.params.id}`);

});

// Display book create form on GET.

exports.book\_create\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Book create GET");

});

// Handle book create on POST.

exports.book\_create\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Book create POST");

});

// Display book delete form on GET.

exports.book\_delete\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Book delete GET");

});

// Handle book delete on POST.

exports.book\_delete\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Book delete POST");

});

// Display book update form on GET.

exports.book\_update\_get = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Book update GET");

});

// Handle book update on POST.

exports.book\_update\_post = asyncHandler(async (req, res, next) => {

res.send("NOT IMPLEMENTED: Book update POST");

});

[**Creación del módulo de ruta de catálogo**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#create_the_catalog_route_module)

A continuación, creamos *rutas* para todas las URL [necesarias para el sitio web de LocalLibrary](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#routes_needed_for_the_locallibrary), que llamará a las funciones del controlador que definimos en las secciones anteriores.

El esqueleto ya tiene una carpeta **./routes** que contiene rutas para el *índice* y *los usuarios*. Cree otro archivo de ruta (**catalog.js**) dentro de esta carpeta, como se muestra.

/express-locallibrary-tutorial //the project root

/routes

index.js

users.js

catalog.js

Abra **/routes/catalog.js** y copie el siguiente código:

JsCopiar en el portapapeles

const express = require("express");

const router = express.Router();

// Require controller modules.

const book\_controller = require("../controllers/bookController");

const author\_controller = require("../controllers/authorController");

const genre\_controller = require("../controllers/genreController");

const book\_instance\_controller = require("../controllers/bookinstanceController");

/// BOOK ROUTES ///

// GET catalog home page.

router.get("/", book\_controller.index);

// GET request for creating a Book. NOTE This must come before routes that display Book (uses id).

router.get("/book/create", book\_controller.book\_create\_get);

// POST request for creating Book.

router.post("/book/create", book\_controller.book\_create\_post);

// GET request to delete Book.

router.get("/book/:id/delete", book\_controller.book\_delete\_get);

// POST request to delete Book.

router.post("/book/:id/delete", book\_controller.book\_delete\_post);

// GET request to update Book.

router.get("/book/:id/update", book\_controller.book\_update\_get);

// POST request to update Book.

router.post("/book/:id/update", book\_controller.book\_update\_post);

// GET request for one Book.

router.get("/book/:id", book\_controller.book\_detail);

// GET request for list of all Book items.

router.get("/books", book\_controller.book\_list);

/// AUTHOR ROUTES ///

// GET request for creating Author. NOTE This must come before route for id (i.e. display author).

router.get("/author/create", author\_controller.author\_create\_get);

// POST request for creating Author.

router.post("/author/create", author\_controller.author\_create\_post);

// GET request to delete Author.

router.get("/author/:id/delete", author\_controller.author\_delete\_get);

// POST request to delete Author.

router.post("/author/:id/delete", author\_controller.author\_delete\_post);

// GET request to update Author.

router.get("/author/:id/update", author\_controller.author\_update\_get);

// POST request to update Author.

router.post("/author/:id/update", author\_controller.author\_update\_post);

// GET request for one Author.

router.get("/author/:id", author\_controller.author\_detail);

// GET request for list of all Authors.

router.get("/authors", author\_controller.author\_list);

/// GENRE ROUTES ///

// GET request for creating a Genre. NOTE This must come before route that displays Genre (uses id).

router.get("/genre/create", genre\_controller.genre\_create\_get);

//POST request for creating Genre.

router.post("/genre/create", genre\_controller.genre\_create\_post);

// GET request to delete Genre.

router.get("/genre/:id/delete", genre\_controller.genre\_delete\_get);

// POST request to delete Genre.

router.post("/genre/:id/delete", genre\_controller.genre\_delete\_post);

// GET request to update Genre.

router.get("/genre/:id/update", genre\_controller.genre\_update\_get);

// POST request to update Genre.

router.post("/genre/:id/update", genre\_controller.genre\_update\_post);

// GET request for one Genre.

router.get("/genre/:id", genre\_controller.genre\_detail);

// GET request for list of all Genre.

router.get("/genres", genre\_controller.genre\_list);

/// BOOKINSTANCE ROUTES ///

// GET request for creating a BookInstance. NOTE This must come before route that displays BookInstance (uses id).

router.get(

"/bookinstance/create",

book\_instance\_controller.bookinstance\_create\_get,

);

// POST request for creating BookInstance.

router.post(

"/bookinstance/create",

book\_instance\_controller.bookinstance\_create\_post,

);

// GET request to delete BookInstance.

router.get(

"/bookinstance/:id/delete",

book\_instance\_controller.bookinstance\_delete\_get,

);

// POST request to delete BookInstance.

router.post(

"/bookinstance/:id/delete",

book\_instance\_controller.bookinstance\_delete\_post,

);

// GET request to update BookInstance.

router.get(

"/bookinstance/:id/update",

book\_instance\_controller.bookinstance\_update\_get,

);

// POST request to update BookInstance.

router.post(

"/bookinstance/:id/update",

book\_instance\_controller.bookinstance\_update\_post,

);

// GET request for one BookInstance.

router.get("/bookinstance/:id", book\_instance\_controller.bookinstance\_detail);

// GET request for list of all BookInstance.

router.get("/bookinstances", book\_instance\_controller.bookinstance\_list);

module.exports = router;

El módulo requiere Express y luego lo usa para crear un objeto. Todas las rutas se configuran en el router, que luego se exporta.Router

Las rutas se definen mediante métodos or en el objeto router. Todas las rutas se definen mediante cadenas (no usamos patrones de cadena ni expresiones regulares). Las rutas que actúan sobre algún recurso específico (por ejemplo, un libro) utilizan parámetros de ruta para obtener el ID del objeto de la URL. .get().post()

Todas las funciones del controlador se importan de los módulos de controlador que creamos en la sección anterior.

[**Actualizar el módulo de ruta de índice**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#update_the_index_route_module)

Hemos configurado todas nuestras nuevas rutas, pero todavía tenemos una ruta a la página original. En su lugar, redirijamos esto a la nueva página de índice que hemos creado en la ruta '/catalog'.

Abra **/routes/index.js** y reemplace la ruta existente con la función a continuación.

JsCopiar en el portapapeles

// GET home page.

router.get("/", function (req, res) {

res.redirect("/catalog");

});

**Nota:** Este es nuestro primer uso del método de respuesta [redirect().](https://expressjs.com/en/4x/api.html#res.redirect) Esto redirige a la página especificada, enviando de forma predeterminada el código de estado HTTP "302 Found". Puede cambiar el código de estado devuelto si es necesario y proporcionar rutas absolutas o relativas.

[**Actualización app.js**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#update_app.js)

El último paso es agregar las rutas a la cadena de middleware. Hacemos esto en . app.js

Abra **app.js** y solicite la ruta del catálogo debajo de las otras rutas (agregue la tercera línea que se muestra a continuación, debajo de las otras dos que ya deberían estar presentes en el archivo):

JsCopiar en el portapapeles

var indexRouter = require("./routes/index");

var usersRouter = require("./routes/users");

const catalogRouter = require("./routes/catalog"); //Import routes for "catalog" area of site

A continuación, agregue la ruta del catálogo a la pila de middleware debajo de las otras rutas (agregue la tercera línea que se muestra a continuación, debajo de las otras dos que ya deberían estar presentes en el archivo):

JsCopiar en el portapapeles

app.use("/", indexRouter);

app.use("/users", usersRouter);

app.use("/catalog", catalogRouter); // Add catalog routes to middleware chain.

**Nota:** Hemos agregado nuestro módulo de catálogo en una ruta . Se antepone a todas las rutas definidas en el módulo de catálogo. Así, por ejemplo, para acceder a una lista de libros, la URL será: .'/catalog'/catalog/books/

Eso es todo. Ahora deberíamos tener las funciones routes y skeleton habilitadas para todas las URL que eventualmente admitiremos en el sitio web de LocalLibrary.

[**Probando las rutas**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#testing_the_routes)

Para probar las rutas, primero inicie el sitio web utilizando su enfoque habitual

* El método predeterminado

juergaCopiar en el portapapeles

# Windows

SET DEBUG=express-locallibrary-tutorial:\* & npm start

# macOS or Linux

DEBUG=express-locallibrary-tutorial:\* npm start

* Si previamente configuraste [nodemon](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/skeleton_website#enable_server_restart_on_file_changes), puedes usar:

bashCopy to Clipboard

npm run serverstart

A continuación, vaya a una serie de direcciones URL de LocalLibrary y compruebe que no recibe una página de error (HTTP 404). A continuación se enumera un pequeño conjunto de URL para su comodidad:

* http://localhost:3000/
* http://localhost:3000/catalog
* http://localhost:3000/catalog/books
* http://localhost:3000/catalog/bookinstances/
* http://localhost:3000/catalog/authors/
* http://localhost:3000/catalog/genres/
* http://localhost:3000/catalog/book/5846437593935e2f8c2aa226
* http://localhost:3000/catalog/book/create

[**Resumen**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#summary)

Ahora hemos creado todas las rutas para nuestro sitio, junto con funciones de controlador ficticio que podemos completar con una implementación completa en artículos posteriores. A lo largo del camino, hemos aprendido mucha información fundamental sobre las rutas rápidas, el manejo de excepciones y algunos enfoques para estructurar nuestras rutas y controladores.

En nuestro próximo artículo crearemos una página de bienvenida adecuada para el sitio, utilizando vistas (plantillas) e información almacenada en nuestros modelos.