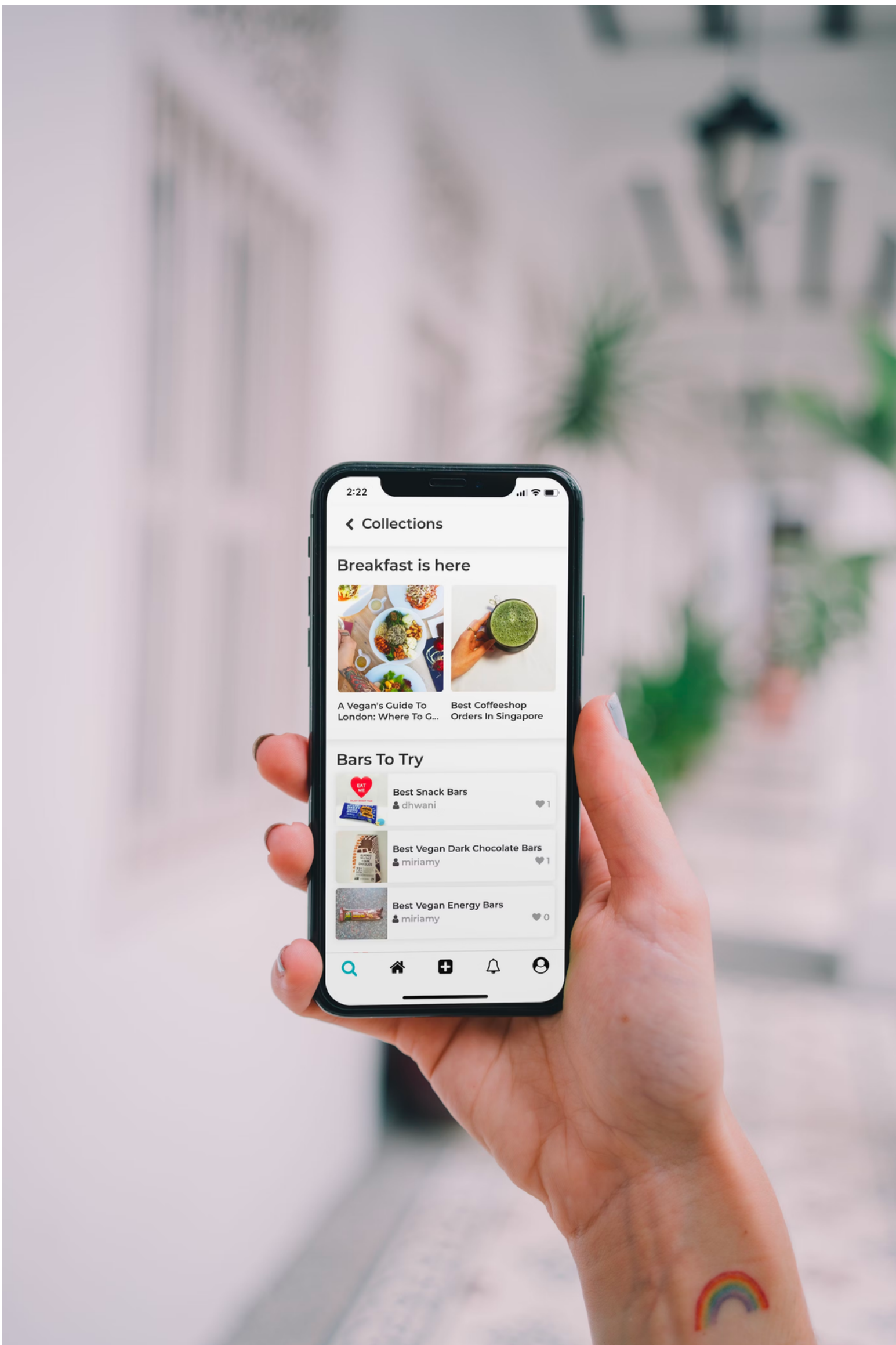


# Lección 4:

# Logueando un

# usuario



## Tiempo de ejecución: 4 horas

### Planteamiento de la sesión



El reto es sencillo, debemos configurar una serie de elementos que permita autenticar a un usuario en nuestro aplicativo.

### Definiendo necesidades

Debido a que por el momento no tenemos muchas rutas, es necesario implementar una que nos permita realizar la tarea de autenticarnos, el nombre generalmente es /login y es el que vamos a utilizar.

Es necesario crear una nueva vista que se renderice al ingresar a la ruta login y que cuente con un formulario con los campos usuario y contraseña, estos datos por seguridad debemos enviarlos por el método POST, lo que nos obliga a generar una ruta que renderice por GET y otra que reciba los datos por POST.

Debemos tener una base de datos de usuarios, es decir un archivo JSON con datos de inicio de sesión de algunos usuarios y para llevarlos a una página u otra podemos utilizar el rol asignado, una vista que diga hola admin y el nombre del usuario y otra vista que salude a un usuario cliente.

Las vistas deberían estar protegidas, es decir que no sea posible ingresar a la ruta por ejemplo /admin si no se ha autenticado como usuario administrador, por ahora se implementará solo con express session, pero comúnmente esas validaciones también se realizan utilizando JSON Web Token o JWT, sobre todo si ofrecemos APIs.

En este punto nos ayudaremos de middlewares de rutas, que validen si el usuario registrado dentro de la sesión primero existe y luego qué rol tiene para permitirle un acceso u otro.

Es necesario también crear un nuevo controlador que cuente con la responsabilidad de autenticar a los usuarios, repartiendo las responsabilidades y cargas entre diferentes archivos según el desarrollo avance. La solución a este reto se encuentra en la rama step5 del proyecto.

## Configurando el ambiente

Primero vamos a crear la data de usuarios, solo necesitamos 2 o 3 usuarios diferentes para probar, recuerde que ya habíamos dicho los datos bases que necesitamos para un usuario en el sistema, nombre, correo, contraseña, estado y roles como administradores, clientes y editores.

Con la data ya generada el segundo paso es crear una ruta que me sirva la vista del formulario de login, luego crear esa vista con los campos usuario y contraseña y finalmente implementar una ruta que por POST me reciba los datos para procesarlos.

```
/* GET users login view. */  
router.get('/login', UserController.login);  
  
/* POST users login. */  
router.post('/login', UserController.processLogin);
```

Estas rutas van a direccionar la petición al controlador UserController que tiene los métodos login que devuelve la vista de login y el processLogin que valida la información enviada en el formulario.

Cuando el usuario ingrese los datos correctos, vamos a guardar la información del usuario en la sesión, para ello es necesario instalar express session con el comando

```
npm install express-session --save
```

Y luego instanciarlo como un middleware de aplicación a nivel del entry point

```
let session = require('express-session');  
app.use(session({secret : 'Shh, es un secreto!'}));
```

el secret puede ser cualquier frase secreta.

Finalmente redireccionamos al usuario a una vista determinada dependiendo de si es cliente o admin o editor.

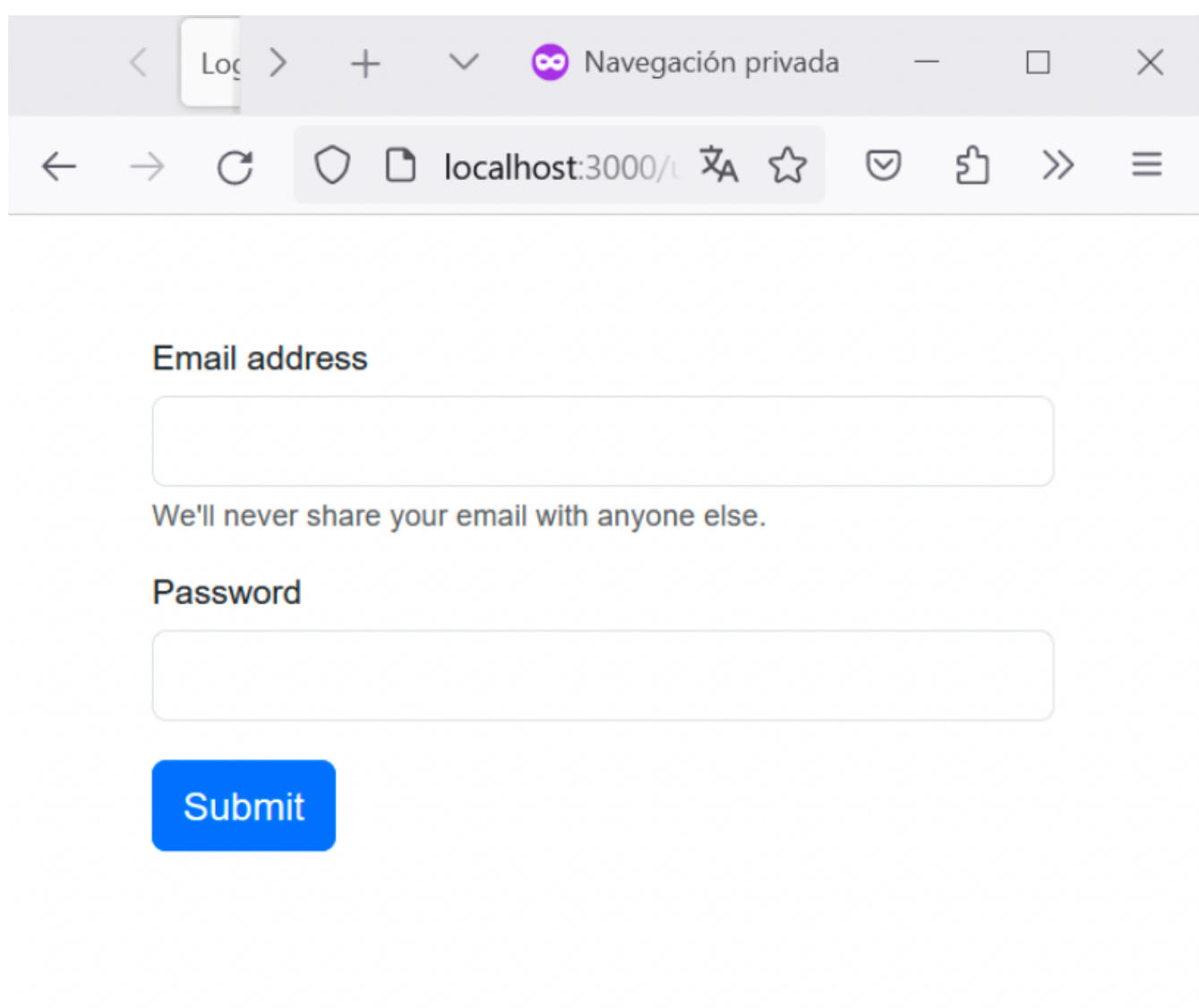
Definamos un poco estos perfiles, vamos a decir que el admin tiene permiso a todo lo del sistema, que el editor a todo menos a la gestión de usuario y que el cliente tiene permisos solo de ver sus datos y los libros.

Así entonces el admin y el editor los vamos a redireccionar a una vista admin y al cliente a la página principal con la diferencia de que aparecerá su nombre en una esquina.

el método que renderiza el login dentro del UserController quedaría así:

```
login: (req, res) => {
  res.render('login', { title: 'Login' });
},
```

el formulario de login, que se accede en la ruta /user/login, creado en la vista login se ve así:



Log > + v Navegación privada

← → ↻ localhost:3000/ A ☆

Email address

We'll never share your email with anyone else.

Password

Submit

Al ingresar los datos de admin, por ejemplo "jane@example.com" y "asdfghjk", el método que recibe los datos es el processLogin del mismo controlador, el método se ve así:



```
processLogin: (req, res) => {
  //read the file users.json from data directory and get the data
  let data = fs.readFileSync(path.join('data', 'users.json'), 'utf8');
  //parse the data to convert it into an array of objects
  let users = JSON.parse(data);
  //get the email and password from the request body
  let email = req.body.email;
  let password = req.body.password;
  //find the user with the email and password
  let user = users.find(user => user.email === email && user.password ===
password);
  //if the user is found, save the user in the session and redirect to the
home page
  if (user) {
    req.session.user = user;
    ['admin', 'editor'].includes(user.rol)
? res.redirect('/admin') : res.redirect('/');
  } else {
    //if the user is not found, render the login view with an error message
    res.render('login', { title: 'Login', error: 'Invalid email or password' });
  }
}
```

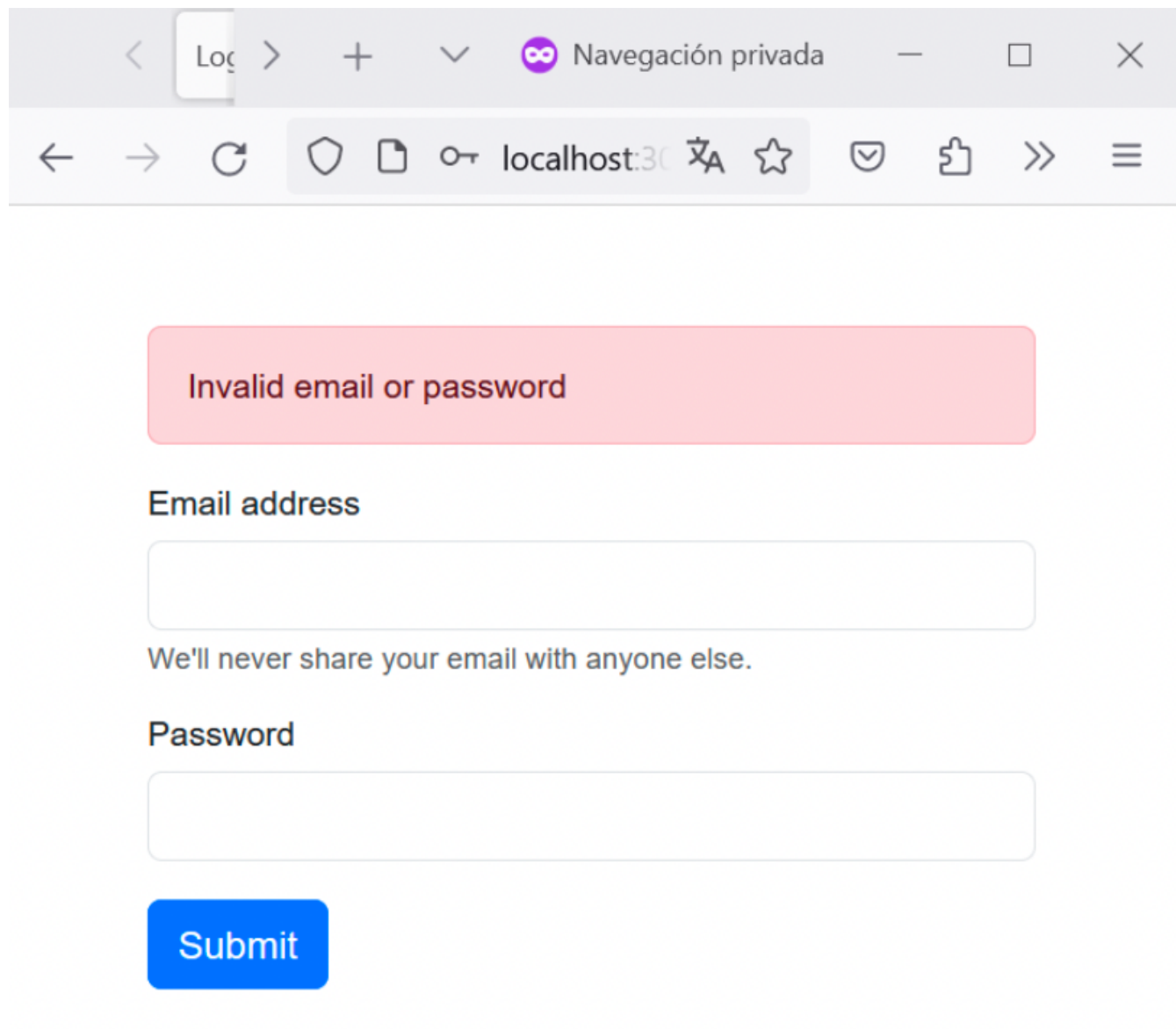
y su funcionamiento es muy sencillo, primero obtiene los datos de todos los usuarios de la base de datos, luego crea dos variables en donde guarda la data que viene del formulario de login, los inputs deben estar marcados con el name email y password para poderlos leer como se muestra en el código.

Ahora busca con el método find, un usuario que tenga el correo y la contraseñas indicadas, el método find retorna la primera coincidencia del criterio de búsqueda o un undefined si no encuentra el dato.

Cabe aclarar que más adelante las contraseñas no se guardan de manera plana sino encriptadas con un paquete llamado bcrypt.

Luego, si el usuario es encontrado entonces lo guardamos en la variable de sesión, ahora consultamos que tipo de usuario tenemos, como definimos anteriormente, los usuarios admin y editor tiene un comportamiento hasta aquí similar, entonces los enviamos a una vista /admin, caso contrario, que serían los usuarios de tipo cliente, los enviamos al index, pero ahora vamos a presentar su nombre y un botón llamado logout con el que cerraremos sesión un poco más adelante.

Finalmente si el usuario no es encontrado, cargará nuevamente la página del login con un mensaje de error indicando que hubo un problema a la hora de ejecutar el login.



The screenshot shows a web browser window with the address bar displaying 'localhost:3000'. The page contains a login form with the following elements:

- A red error message box at the top stating 'Invalid email or password'.
- A label 'Email address' above a text input field.
- A message 'We'll never share your email with anyone else.' below the email input field.
- A label 'Password' above a password input field.
- A blue 'Submit' button at the bottom.

Ahora, si nos autenticamos con un admin o editor, no hemos implementado ni las rutas ni el controlador ni las vistas para poder atender la solicitud, entonces ese será el siguiente paso.

## Funcionamiento de admin

Lo primero es crear la ruta /admin y el enrutador que atenderá todo lo que vaya en esta ruta, para ello en la index del router vamos a crear una nueva ruta:

```
const adminRouter = require('./admin');
//admin routes
router.use('/admin', adminRouter);
```

y ahora debemos crear ese adminRouter como archivo dentro de la carpeta, por ahora solo contará con un método GET que atiende las peticiones directamente que llegan a /admin:

## Protegiendo las rutas

Ahora, que pasa si un usuario ya autenticado intenta ingresar a la ruta /user/login o si un usuario no autenticado o tipo cliente intenta ingresar a una ruta /admin?

El aplicativo debe poder identificar estos comportamientos y autorizar o restringir el acceso, para esto utilizaremos dos middlewares muy sencillos que limitará el acceso a las rutas dependiendo de lo que definamos, por ejemplo, el rol.

El primer middleware se llamará authMiddleware, lo implementaremos en la ruta /user/login y solo validará si existe una variable de sesión para el navegador desde el que se hace la petición, si existe entonces redirecciona al usuario ya autenticado a la página principal, caso contrario dejará seguir al usuario a que se autentique. También es posible identificar el rol y reenviarlo hacía la ruta que nos parezca.

El middleware creado en la carpeta de middlewares se vería así:

```
function authMiddleware (req, res, next){  
  if (!req.session.user) next()  
  else res.redirect('/')  
};
```

```
module.exports = authMiddleware;
```

Y requerirlo en la ruta de usuarios se vería así:

```
const authMiddleware = require('../middlewares/authMiddleware');
```

```
/* GET users login view. */  
router.get('/login', authMiddleware, UserController.login);
```

Si nos probamos autenticar y luego ingresar a la misma ruta veremos que ya no es posible ingresar y nos envía a la página principal.

Para implementar el segundo middleware la dinámica es la misma, solo que ahora el middleware lo llamaremos adminMiddleware y en este vamos a validar que el rol sea admin o editor, si es así dejamos continuar, sino entonces redireccionamos al login o la página principal si el rol es otro como cliente, el archivo igual lo creamos en la carpeta de middlewares y se vería así:

```
function adminMiddleware (req, res, next){
  if (req.session.user) {
    if (['admin', 'editor'].includes(req.session.user.rol))

      {
        next()
      }else {
        res.redirect('/')
      }
    }else {
      res.redirect('/user/login')
    }
  };

  module.exports = adminMiddleware;
```

Ahora lo implementamos en la ruta que nos interesa, en /admin, el enrutador entonces se vería así:

```
var express = require('express');
var router = express.Router();

const AdminController = require('../controllers/AdminController');
const adminMiddleware = require('../middlewares/adminMiddleware');

/* GET users login view. */
router.get('/', adminMiddleware, AdminController.index);

module.exports = router;
```

Con esto, nuestro proyecto ahora cuenta con muchas nuevas funcionalidades y ya podemos autenticar usuarios y proteger rutas, en la siguiente lección vamos a utilizar multer para cargar archivos por ejemplo la portada a la hora de crear un nuevo libro, acción que solo puede desarrollar el admin o editor.



Para implementar el segundo middleware la dinámica es la misma, solo que ahora el middleware lo llamaremos adminMiddleware y en este vamos a vali

```
module.exports = router;
```

Con esto, nuestro proyecto ahora cuenta con muchas nuevas funcionalidades y ya podemos autenticar usuarios y proteger rutas, en la siguiente lección vamos a utilizar multer para cargar archivos por ejemplo la portada a la hora de crear un nuevo libro, acción que solo puede desarrollar el admin o editor.

Recuerden que el proyecto hasta este punto se encuentra en la rama step5.

