Clase 18: Manejo de sesiones: Sistema de Autenticación

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Nilo Crespi
Curso:	Curso de Backend Developer - Turno Noche	Día:	Friday, 23 de January de 2026, 10:34
Libro:	Clase 18: Manejo de sesiones: Sistema de Autenticación		

Descripción

Tabla de contenidos

1. Introducción

En esta clase hemos implementado cubre tanto el **back-end** (servidor) como el **front-end** (cliente) de una aplicación web. A continuación, se explican los conceptos y el flujo paso a paso para comprender cómo funciona cada parte del sistema de autenticación, desde la creación y validación del usuario en el servidor hasta el manejo del estado de autenticación en el cliente.

2. Back-end: Servidor de autenticación con Node.js y Express

Back-end: Servidor de autenticación con Node.js y Express

Conceptos clave en el Back-end

1 . **Mongoose y MongoDB** :

- **Mongoose** se utiliza para interactuar con la base de datos MongoDB. Creamos un esquema de usuario (UserSchema) que define los datos que cada usuario tendrá, como username y password.

2. **Bcrypt para el hashing de contraseñas** :

- **Bcrypt** se emplea para encriptar las contraseñas de los usuarios antes de almacenarlas en la base de datos. Esto agrega una capa de seguridad, ya que las contraseñas no se guardan en texto plano.

- **Session secret:**

El "session secret" es una cadena aleatoria utilizada por express-session para firmar el ID de la sesión. Esto ayuda a prevenir ataques de falsificación de solicitudes entre sitios (CSRF) y asegura que la sesión no pueda ser falsificada por un atacante. Es crucial mantener este secreto seguro y cambiarlo si se sospecha que ha sido comprometido.

3. **Sesiones de Express** :

- Usamos **express-session** para gestionar la sesión del usuario en el servidor. Almacena un identificador de sesión único que permite reconocer al usuario en solicitudes futuras después de iniciar sesión.

4. **Rutas Protegidas** :

- Implementamos rutas protegidas en el servidor, las cuales requieren que el usuario esté autenticado para acceder. Si no está autenticado, se le responde con un mensaje de "No autorizado".

5 . **Generación del secreto:**

Para implementar el sistema de autenticación, es necesario generar un secreto que se utilizará para firmar los tokens JWT y asegurar las sesiones. Podés usar el siguiente comando en Node.js para generar un secreto:

```
node -e
"console.log(require('crypto').randomBytes(32).toString('hex'));"
```

Modelo de usuario (User.js):

Este modelo define cómo se almacena y compara la contraseña de los usuarios.

1 . **Definición de Esquema** :

- username y password son obligatorios y el username debe ser único.

2. **Método comparePassword** :

- Este método usa bcrypt para comparar una contraseña ingresada con la contraseña encriptada almacenada en la base de datos. Es útil para la autenticación, pues permite verificar si la contraseña ingresada es correcta.

Rutas de Autenticación (authRoutes.js)

Creamos rutas para registrar usuarios, iniciar sesión y acceder a contenido protegido.

1. **Registro de usuarios** :

- Busca un usuario existente en la base de datos.
- Si no existe, se encripta la contraseña con bcrypt.hash() y se guarda el nuevo usuario.

Inicio de sesión y rutas protegidas

1 . **Inicio de sesión** :

- Busca el usuario por username y compara la contraseña ingresada.
- Si coincide, se almacena el username en la sesión.

2 . **Ruta protegida** :

- Verifica si el usuario tiene una sesión activa.
- Si no está autenticado, responde con un mensaje de "No autorizado".

3. Front-end: Manejo de Autenticación en React

Front-end: Manejo de Autenticación en React

Conceptos clave en el Front-end

1 . **Context API de React:**

- Utilizamos el Context API para compartir el estado de autenticación entre componentes sin tener que pasar props manualmente.

2. **AuthProvider y AuthContext:**

- Creamos un contexto AuthContext para proporcionar el estado de autenticación (si el usuario está o no autenticado) y funciones para manejar el inicio y cierre de sesión.

3. **Manejo de estado en Componentes:**

- Componentes como Login utilizan el contexto para iniciar sesión y actualizar el estado de autenticación de la aplicación.

Contexto de Autenticación (AuthContext.js)

El AuthProvider gestiona el estado de autenticación y proporciona funciones login y logout.

1. **Estado isAuthenticated** :

- Indica si el usuario está autenticado (true o false).

2. **Funciones login y logout** :

- Cambian el estado de autenticación al iniciar o cerrar sesión.

3 . **Hook useAuth** :

- Simplifica el acceso al contexto en cualquier componente.

Componente de inicio de sesión (Login.js)

Este componente maneja el proceso de inicio de sesión del usuario.

1 . **Manejo del Formulario** :

- Al enviar el formulario, se realiza una solicitud POST al servidor con username y password.
- Si la autenticación es exitosa, se llama a login() para actualizar el contexto.

Ruta protegida en el Front-end (ProtectedRoute.js)

Este componente protege las rutas que solo deben ser accesibles para usuarios autenticados.

- **Redireccionamiento condicional** :
  - Si isAuthenticated es true, muestra el contenido protegido.
  - Si no, redirige al usuario a la página de inicio de sesión (/login).

4. ACTIVIDAD PRÁCTICA

ACTIVIDAD PRÁCTICA: implementación de un Sistema de Autenticación

Parte 1: Configuración del servidor (Back-end)

1.1. Configuración inicial

- Crear o reutilizar proyecto de Node.js y configurar Express.

Dependencias necesarias: express, mongoose, express-session, bcrypt, connect-mongo.

1.2 . Creación del esquema de usuario

- Definir el esquema de usuario utilizando Mongoose.
- Crear un archivo User.js que defina UserSchema, o similar, con los campos username (único) y password (encriptado).
  - Implementar el método comparePassword para comparar contraseñas.

1.3 . Implementación de rutas de autenticación

- Crear rutas para el registro y el inicio de sesión.
- Crear un archivo authRoutes.js y definir las rutas para:
  - Registro (POST /register): Encriptar la contraseña y almacenar el usuario en la base de datos.
  - Inicio de sesión (POST /login): Validar el usuario y almacenar el username en la sesión.
  - Ruta protegida (GET /protected): Responder con un mensaje de "No autorizado" si el usuario no está autenticado.

1.4. Prueba del servidor

- Probar las rutas de autenticación utilizando Postman o un cliente similar.
- Asegurarse de que los usuarios se pueden registrar, iniciar sesión y acceder a rutas protegidas.

Parte 2: Desarrollo del Cliente (Front-end)

2.1. Configuración inicial

- Crear o adaptar un proyecto de React.

2.2. Implementación del contexto de autenticación

Crear un contexto para manejar la autenticación globalmente.

- Crear AuthContext.js que incluya AuthProvider, que maneje el estado isAuthenticated, y las funciones login y logout.

2.3. Componente de inicio de sesión

- Crear un componente Login.js.
- Implementar un formulario que capture el username y password.
- Al enviar el formulario, hacer una solicitud POST al servidor para autenticar al usuario. Si es exitoso, actualizar el contexto de autenticación.

2.4 . Ruta protegida

- Crear un componente ProtectedRoute.js.
- Implementar la lógica de redireccionamiento según el estado de isAuthenticated.

2.5. Prueba del cliente

- Probar la aplicación en el navegador.
- Asegurarse de que el inicio de sesión funcione correctamente y que las rutas protegidas se comporten según lo esperado.

Parte 3: Documentación y presentación

3.1. Documentación

- Crear un documento breve README.md que explique cómo funciona el sistema de autenticación.
- Incluir detalles sobre cómo se maneja la sesión en el servidor y cómo se gestiona la autenticación en el cliente.

3.2. Presentación

- Preparar una breve presentación para compartir con el resto de la clase.

Explicar la arquitectura de la aplicación, los desafíos encontrados y las soluciones implementadas.

Criterios a considerar

- Funcionalidad del sistema de autenticación (registro, inicio de sesión, acceso a rutas protegidas).
- Uso adecuado de Mongoose y bcrypt en el back-end.
- Implementación correcta del Context API en el front-end.
- Calidad de la documentación y presentación.