Manual Técnico.

Generalidades del uso del web service (Back-end)

Autenticación del servidor JSON :

Empezando:

Instale el servidor JSON y la autenticación del servidor JSON:

```
# MNP

npm install -D json-servidor json-servidor-auth



# Hilado

hilo agregar -D json-server json-server-auth
```

Crear un db. json archivo con una users colección:

```
{

   " usuarios " : []

}
```

Inicie el servidor JSON (con la autenticación del servidor JSON como middleware):

```
json-server db.json -m ./node_modules/json-server-auth

 # con json-server instalado globalmente y json-server-auth
instalado localmente
```

Para su comodidad, json-server-authCLI expone json-serverjunto con su middleware:

```
json-server-auth db.json

 # con json-server-auth instalado globalmente
```

Expone y funciona de la misma manera para todas las banderas del servidor JSON .

Flujo de autenticación

JSON Server Auth agrega un flujo de autenticación simple basado en JWT .

Registro 👪

Cualquiera de las siguientes rutas registra un nuevo usuario:

- POST /register
- POST /signup
- POST /users

Email y password se requieren en el cuerpo de la solicitud:

```
POST /registrar

{

  " correo electrónico " : " olivier@mail.com " ,

   " contraseña " : " bestPassw0rd "

}
```

La contraseña está encriptada por bcryptjs . La respuesta contiene el token de acceso JWT (tiempo de caducidad de 1 hora):

```
201 Creado

{

  " token de acceso " : " xxx.xxx.xxx "

}
```

Otras propiedades

Se puede agregar cualquier otra propiedad al cuerpo de la solicitud sin ser validada:

```
POST /registrar

{
```

```
  " email " : " olivier@mail.com " ,
   " contraseña " : " bestPassw0rd " ,
   " firstname " : " Olivier " ,
   " lastname " : " Monge " ,
   " age " : 32
}
```

Actualizar:

Cualquier actualización de un usuario existente (vía PATCH o PUT métodos) pasará por el mismo proceso para email y password.

Acceso 💪

Cualquiera de las siguientes rutas registra a un usuario existente:

- POST /login
- POST /signin

Email y password se requieren, por supuesto:

```
POST / inicio de sesión
{
  " correo electrónico " : " olivier@mail.com " ,
   " contraseña " : " bestPassw0rd "
}
```

La respuesta contiene el token de acceso JWT (tiempo de caducidad de 1 hora):

```
200 bien
{
  " token de acceso " : " xxx.xxx.xxx "
```

```
}
```

Carga útil JWT

El token de acceso tiene las siguientes reclamaciones:

- **sub:** el usuario id (según las especificaciones de JWT)
- **email:** el usuario email.

Flujo de autorización.

JSON Server Auth proporciona protecciones genéricas como middleware de ruta.

Para manejar casos de uso comunes, JSON Server Auth se inspira en los permisos del sistema de archivos de Unix, especialmente en la notación numérica.

- Agregamos 4para permiso de lectura.
- Agregamos 2para permiso de escritura.

Por supuesto, CRUD no es un sistema de archivos, por lo que no agregamos 1 para el permiso de ejecución.

De manera similar a Unix, tenemos tres dígitos para hacer coincidir cada tipo de usuario:

- El primer dígito son los permisos para el propietario del recurso.
- El segundo dígito son los permisos para los usuarios registrados.
- El tercer dígito son los permisos para los usuarios públicos.

Por ejemplo, 640significa que solo el propietario puede escribir el recurso, los usuarios registrados pueden leer el recurso y los usuarios públicos no pueden acceder al recurso en absoluto.

El dueño del recurso

Un usuario es el propietario de un recurso si ese recurso tiene una userId propiedad que coincide con su id propiedad. Ejemplo:

```
// El dueño de

{  id : 8 ,  text : 'blabla' ,  userId : 1  }

// es
```

```
{  id : 1 ,  email : 'olivier@mail.com'  }
```

Las rutas protegidas privadas utilizarán el subreclamo JWT (que es igual al usuario id) para verificar si el usuario realmente posee el recurso solicitado, comparándolo subcon la userIdpropiedad.

Excepto por la userscolección real, donde el subreclamo JWT debe coincidir con la idpropiedad.

Rutas vigiladas

Las rutas protegidas existen en la raíz y pueden restringir el acceso a cualquier recurso que coloque después de ellas:

Ruta	Permisos de recursos
`/664/*`	El usuario debe iniciar sesión para *escribir* el recurso. Todos pueden *leer* el recurso.
`/660/*`	El usuario debe iniciar sesión para *escribir* o *leer* el recurso.
`/644/*`	El usuario debe poseer el recurso para *escribir* el recurso. Todos pueden *leer* el recurso.
`/640/*`	El usuario debe poseer el recurso para *escribir* el recurso. El usuario debe iniciar sesión para *leer* el recurso.
`/600/*`	El usuario debe poseer el recurso para *escribir* o *leer* el recurso.
`/444/*`	Nadie puede *escribir* el recurso. Todos pueden *leer* el recurso.
`/440/*`	Nadie puede *escribir* el recurso. El usuario debe iniciar sesión para *leer* el recurso.
`/400/*`	Nadie puede *escribir* el recurso. El usuario debe poseer el recurso para *leer* el recurso.

Ejemplos

- El usuario público (no registrado) realiza las siguientes solicitudes:

Pedido	***Respuesta***
`GET /664/posts`	`200 OK`
`POST /664/posts` `{text: 'blabla'}`	`401 UNAUTHORIZED`

- El usuario que ha iniciado sesión id: 1hace las siguientes solicitudes:

Pedido	***Respuesta***
`GET /600/users/1` `Authorization: Bearer xxx.xxx.xxx`	`200 OK`
`GET /600/users/23` `Authorization: Bearer xxx.xxx.xxx`	`403 FORBIDDEN`

Permisos de configuración 💡

Por supuesto, no desea utilizar rutas protegidas directamente en sus solicitudes. Podemos aprovechar la función de rutas personalizadas del servidor JSON para configurar los permisos de recursos por adelantado.

Crear un routes.json archivo:

```
{

   " /usuarios* " : " /600/usuarios$1 " ,

   " /mensajes* " : " /640/mensajes$1 "

}
```

Entonces:

```
json-server db.json -m ./node_modules/json-server-auth -r
route.json

 # con json-server instalado globalmente y json-server-auth
instalado localmente
```

📣, pero espera!

Para su comodidad, **json-server-auth** CLI le permite definir permisos de una manera más sucinta:

```
{

   " usuarios " : 600 ,

   " mensajes " : 640

}
```

Entonces:

```
json-server-auth db.json -r rutas.json

 # con json-server-auth instalado globalmente
```

Todavía puede agregar cualquier otra ruta personalizada normal :

```
{

   " usuarios " : 600 ,

   " mensajes " : 640 ,

   " /publicaciones/:categoría " : "
/publicaciones?categoría=:categoría "

}
```

Uso del módulo

Si opta por la programación y **usa JSON Server como un módulo**, hay un paso adicional para integrar correctamente JSON Server Auth:

⚠Debe vincular la propiedad del enrutador da la aplicación creada, como **lo hace la CLI del servidor JSON**, y debe aplicar los middlewares en un orden específico.

```
const  jsonServer  =  require ( 'json-server' )

const  auth  =  require ( 'json-server-auth' )



const  app  =  jsonServer . crear ( )

const  enrutador  =  jsonServer . enrutador ( 'db.json' )



// /!\ Vincula la base de datos del enrutador a la aplicación

app . db  =  enrutador . base de datos



// Debe aplicar el middleware de autenticación antes de la

aplicación del enrutador . usar ( autorización )

aplicación . use la aplicación ( enrutador )

. escuchar ( 3000 )
```

Re escritor de permisos

Se puede acceder al reescritor personalizado a través de una subpropiedad:

```
const  auth  =  require ( 'json-server-auth' )


 reglas  constantes =  auth . rewriter ( {
  // Reglas de permisos
  usuarios : 600 ,
  mensajes : 640 ,
  // Otras reglas
  '/posts/:category' : '/posts?category=:category' ,
} )


// Debe aplicar los middlewares en el siguiente orden
app . uso ( reglas )
aplicación . usar ( autorización )
aplicación . usar ( enrutador )
```

HACER 📜

- [ ] Usar servidor JSON id foreignKeySuffix parámetros
- [ ] Maneje los parámetros de consulta en las solicitudes de lista para proteger las rutas protegidas con mayor precisión

- [ ] Permitir configuración de :
    - [ ] Nombre de la colección de usuarios
    - [ ] Longitud mínima de la contraseña
    - [ ] Tiempo de caducidad de JWT
    - [ ] Nombre de propiedad JWT en respuesta
- [ ] Implementar el token de actualización de JWT
- [ ] ¿Posibilidad de deshabilitar el cifrado de contraseña?