

React + Node.js Express: ejemplo de autenticación de usuario con JWT

Fuente: <https://www.bezkoder.com/>

Última modificación: 23 de septiembre de 2022 bezkoder Pila completa , Node.js , React , Seguridad

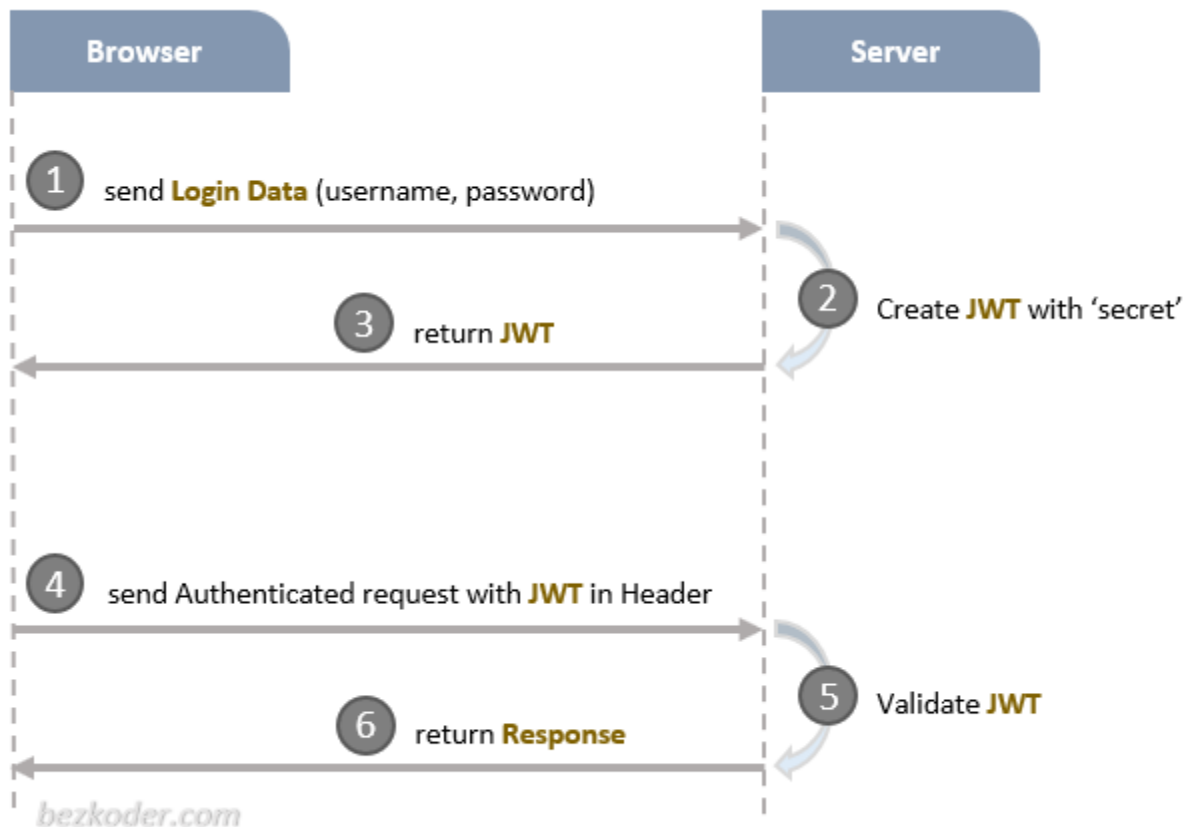
En este tutorial, aprenderemos cómo crear un ejemplo completo de autenticación y autorización de React.js + Express. El servidor back-end usa Node.js Express con jsonwebtoken para la autenticación JWT y Sequelize para interactuar con la base de datos MySQL. El front-end se creará con React, React Router, Axios. También usaremos Bootstrap y realizaremos la validación de formularios.

Contenidos [ocultar]

- JWT (token web JSON)
- Vídeo de demostración
- Back-end con Node.js Express y Sequelize
 - Descripción general
 - Tecnología
 - Estructura del proyecto
 - Implementación
- Interfaz con React, React Router
 - Descripción general
 - Tecnología
 - Estructura del proyecto
 - Implementación
- Código fuente
- Conclusión
- Otras lecturas

JWT (token web JSON)

En comparación con la autenticación basada en sesión que necesita almacenar la sesión en una cookie, la gran ventaja de la autenticación basada en token es que almacenamos el token web JSON (JWT) en el lado del cliente: almacenamiento local para el navegador, llavero para iOS y preferencias compartidas para Android... Por lo tanto, no necesitamos crear otro proyecto de back-end que admita aplicaciones nativas o un módulo de autenticación adicional para usuarios de aplicaciones nativas.



Hay tres partes importantes de un JWT: encabezado, carga útil y firma. Juntos se combinan en una estructura estándar: `header.payload.signature`.

El cliente normalmente adjunta JWT en el encabezado **x-access-token** :

```
x-access-token: [header].[payload].[signature]
```

Para obtener más detalles, puede visitar:

[Introducción detallada al token web JWT-JSON](#)

Ejemplo de autenticación de React Express

Será una pila completa, con Node.js Express para back-end y React.js para front-end. El acceso se verifica mediante la autenticación JWT.

- El usuario puede registrarse en una nueva cuenta, iniciar sesión con nombre de usuario y contraseña.
- Autorización por el rol del Usuario (administrador, moderador, usuario)

Back-end con Node.js Express y Sequelize

Descripción general

Construiremos una aplicación Node.js Express en eso:

- El usuario puede registrarse en una nueva cuenta o iniciar sesión con nombre de usuario y contraseña.
- Por el rol del Usuario (administrador, moderador, usuario), autorizamos al Usuario a acceder a los recursos

Esta es nuestra demostración de la aplicación Node.js que se ejecuta con la base de datos MySQL y prueba Rest Apis con Postman.

=====

Video demostración



<https://www.youtube.com/watch?v=fLcxkyMDW08>

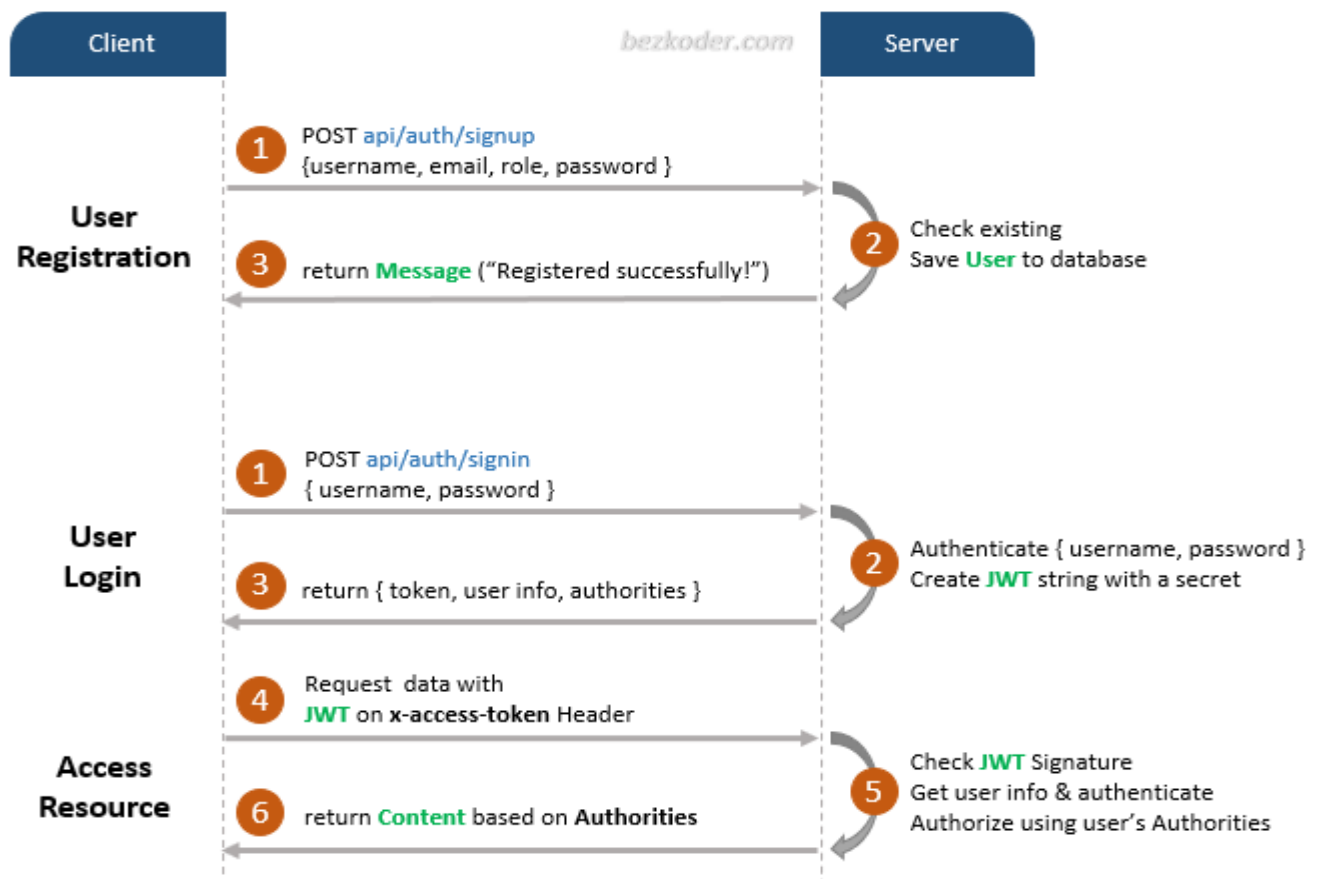
=====

Estas son las API que debemos proporcionar:

Métodos	URL	Comportamiento
CORREO	/api/autorización/registro	registrarse nueva cuenta
CORREO	/api/autorización/inicio de sesión	iniciar sesión en una cuenta
CONSEGUIR	/api/prueba/todos	recuperar contenido público
CONSEGUIR	/api/prueba/usuario	acceder al contenido del usuario
CONSEGUIR	/api/prueba/mod	acceder al contenido del moderador
CONSEGUIR	/api/prueba/admin	acceder al contenido del administrador

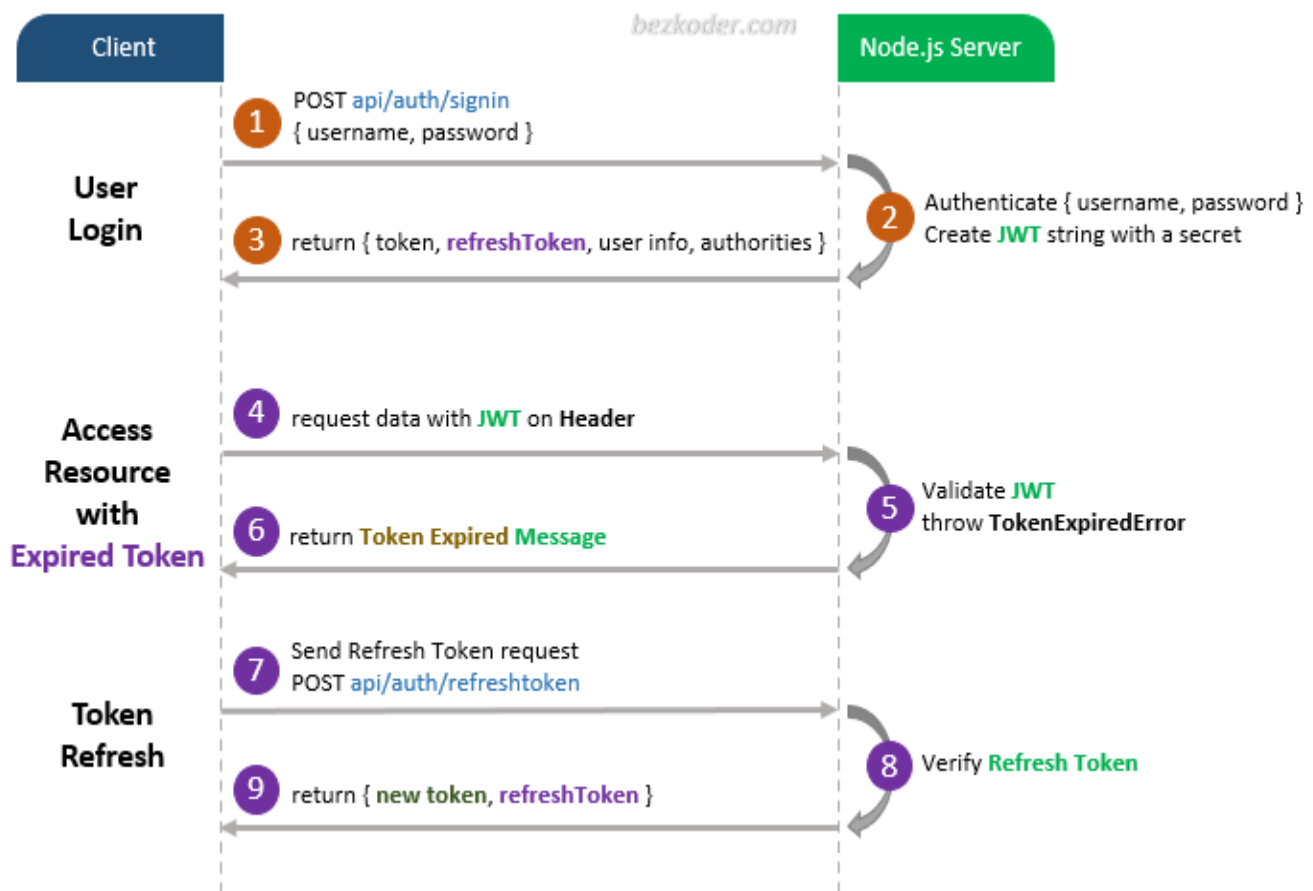
Flujo de registro e inicio de sesión con autenticación JWT

El diagrama muestra el flujo del proceso de registro de usuario, inicio de sesión de usuario y autorización.



Se debe agregar un JWT legal al encabezado HTTP **x-access-token** si el cliente accede a los recursos protegidos.

Deberá implementar Refresh Token:



Más detalles en: [Implementación del token de actualización de JWT en el ejemplo de Node.js](https://www.bezkoder.com/jwt-refresh-token-node-js/)

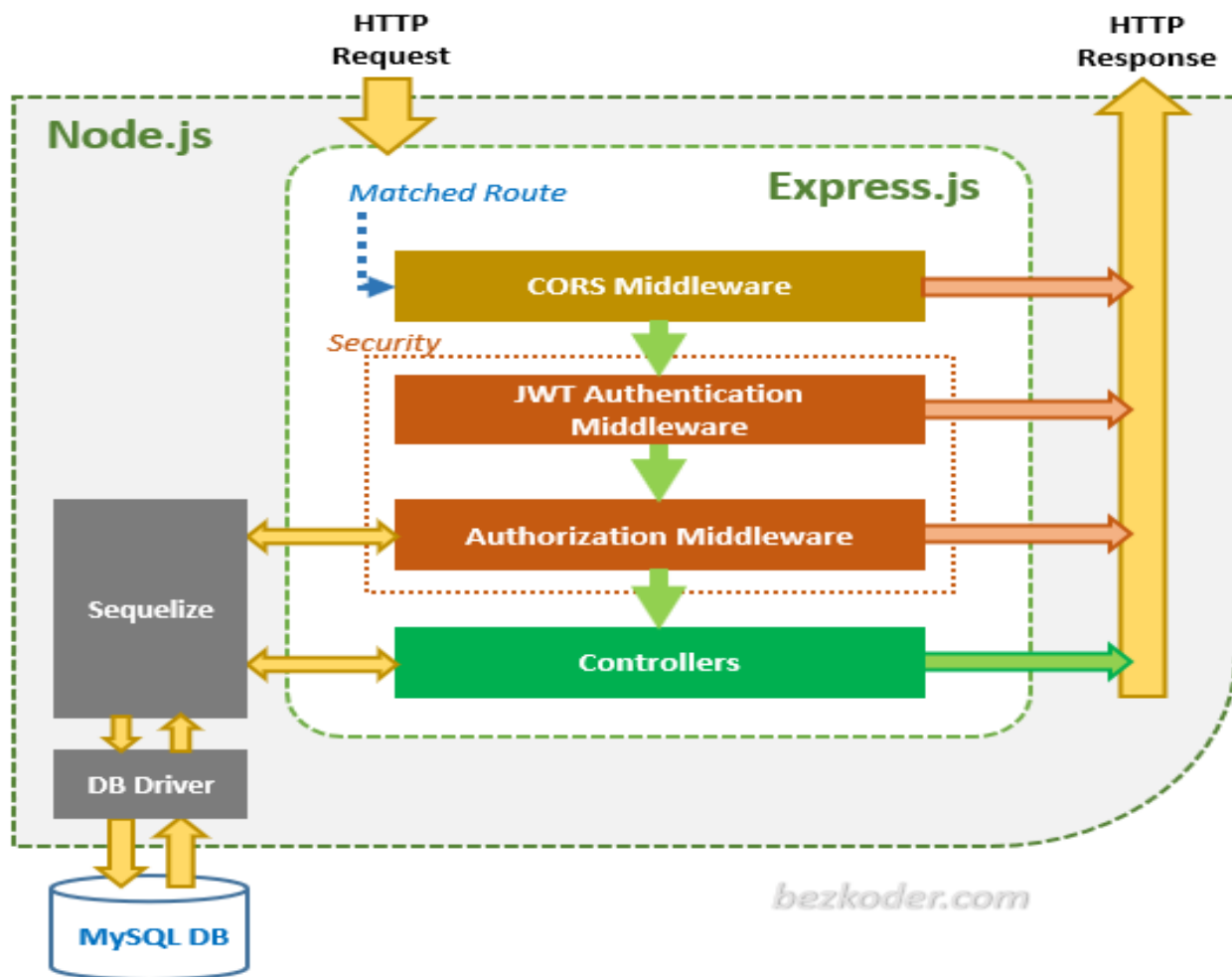
<https://www.bezkoder.com/jwt-refresh-token-node-js/>

Si desea utilizar cookies, visite:

[Node.js Express: ejemplo de inicio de sesión y registro con JWT](https://www.bezkoder.com/node-js-express-login-example/)

<https://www.bezkoder.com/node-js-express-login-example/>

Nuestra aplicación Node.js Express se puede resumir en el siguiente diagrama:



A través de las rutas *Express* , **CORS Middleware** verificará **la solicitud HTTP** que coincida con una ruta antes de llegar a la capa **de seguridad** .

La capa de seguridad incluye:

- Middleware de autenticación JWT: verificar el registro, verificar el token
- Middleware de autorización: verifique las funciones del usuario con registro en la base de datos

Si estos middlewares arrojan algún error, se enviará un mensaje como respuesta HTTP.

Los controladores interactúan con la base de datos MySQL a través de *Sequelize* y envían **una respuesta HTTP** (token, información de usuario, datos basados en roles...) al cliente.

Si desea utilizar HttpOnly Cookie para almacenar JWT, visite:

[Ejemplo de inicio de sesión y registro de React.js: JWT y HttpOnly Cookie](#)

<https://www.bezkoder.com/react-login-example-jwt-hooks/>

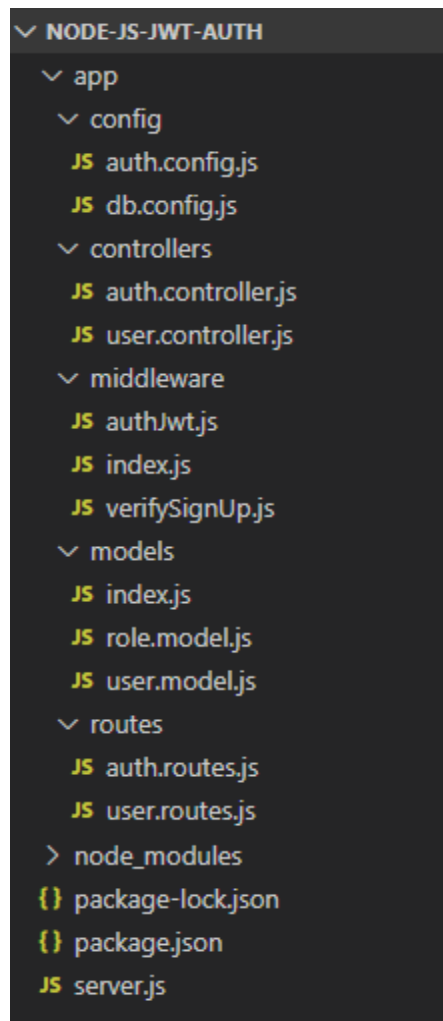
=====

Tecnología

- Express 4.17.1
- bcryptjs 2.4.3
- jsonwebtoken 8.5.1
- Sequelize 5.21.3
- MySQL

Estructura del proyecto

Esta es la estructura de directorios para nuestra aplicación Node.js Express:



– configuración

- configurar la base de datos MySQL y Sequelize
- configurar la clave de autenticación

– rutas

- *auth.routes.js* : POST registro e inicio de sesión
- *user.routes.js* : OBTENGA recursos públicos y protegidos

– programas intermedios

- *verificarSignUp.js* : verifique el nombre de usuario o el correo electrónico duplicados
- *authJwt.js* : verificar token, verificar roles de usuario en la base de datos

– controladores

- *auth.controller.js* : manejar las acciones de registro e inicio de sesión
- *user.controller.js* : devolver contenido público y protegido

– modelos para Sequelize Models

- *usuario.modelo.js*
- *role.modelo.js*

– *server.js* : importa e inicializa los módulos y rutas necesarios, escucha las conexiones.

Crear aplicación Node.js

Primero, creamos una carpeta para nuestro proyecto:

```
$ mkdir node-js-jwt-auth
$ cd node-js-jwt-auth
```

Luego inicializamos la aplicación Node.js con un archivo *package.json* :

```
npm init

name: (node-js-jwt-auth)
version: (1.0.0)
description: Node.js Demo for JWT Authentication
entry point: (index.js) server.js
test command:
git repository:
keywords: node.js, express, jwt, authentication, mysql
author: bezkoder
license: (ISC)

Is this ok? (yes) yes
```

Necesitamos instalar los módulos

necesarios : express, cors, sequelize, mysql2y jsonwebtoken. Ejecute el comando:bcryptjs

```
npm install express sequelize mysql2 cors jsonwebtoken bcryptjs --save
```

El archivo package.json ahora se ve así:

```
{
  "name": "node-js-jwt-auth",
  "version": "1.0.0",
  "description": "Node.js Demo for JWT Authentication",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "node.js",
    "jwt",
    "authentication",
    "express",
    "mysql"
  ],
}
```



```

    "author": "bezkodeer",
    "license": "ISC",
    "dependencies": {
      "bcryptjs": "^2.4.3",
      "cors": "^2.8.5",
      "express": "^4.17.1",
      "jsonwebtoken": "^8.5.1",
      "mysql2": "^2.1.0",
      "sequelize": "^5.21.3"
    }
  }
}

```

Configuración del servidor web Express

En la carpeta raíz, creemos un nuevo archivo *server.js* :

```

const express = require("express");
const cors = require("cors");

const app = express();

var corsOptions = {
  origin: "http://localhost:8081"
};

app.use(cors(corsOptions));

// parse requests of content-type - application/json
app.use(express.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bezkodeer application." });
});

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});

```

Permítanme explicar lo que acabamos de hacer:

– importación expressy cors módulos:

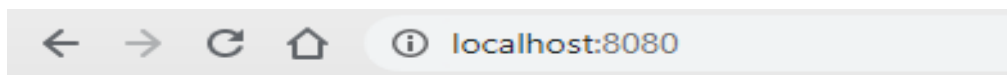
- Express es para construir las API Rest
- cors proporciona middleware Express para habilitar CORS

– cree una aplicación Express, luego agregue el analizador de cuerpo de solicitud y corslos middlewares usando app.use()el método. Observe que establecemos el origen: http://localhost:8081.

– definir una ruta GET que sea simple para la prueba.
– escuche en el puerto 8080 las solicitudes entrantes.

Ahora ejecutemos la aplicación con el comando: node server.js.

Abra su navegador con url http://localhost:8080/ , verá:



```
{"message": "Welcome to bezkoder application."}
```

Configurar la base de datos MySQL y Sequelize

En la carpeta **de la aplicación** , cree la carpeta **de configuración** para la configuración con el archivo `db.config.js` como este:

```
module.exports = {  
  HOST: "localhost",  
  USER: "root",  
  PASSWORD: "123456",  
  DB: "testdb",  
  dialect: "mysql",  
  pool: {  
    max: 5,  
    min: 0,  
    acquire: 30000,  
    idle: 10000  
  }  
};
```

Los primeros cinco parámetros son para la conexión MySQL.

pooles opcional, se usará para la configuración del grupo de conexiones de Sequelize:

- max: número máximo de conexiones en el grupo
- min: número mínimo de conexión en el grupo
- idle: tiempo máximo, en milisegundos, que una conexión puede estar inactiva antes de ser liberada
- acquire: tiempo máximo, en milisegundos, ese grupo intentará conectarse antes de arrojar un error

Para obtener más detalles, visite [Referencia de API para el constructor de Sequelize](#) .

Definir el modelo de sequelize

En la carpeta *de modelos* , cree Userun Rolemodelo de datos como el siguiente código:

modelos / usuario.modelo.js

```
module.exports = (sequelize, Sequelize) => {  
  const User = sequelize.define("users", {  
    username: {  
      type: Sequelize.STRING  
    },  
    email: {  
      type: Sequelize.STRING  
    },  
    password: {  
      type: Sequelize.STRING  
    }  
  });  
};
```

```

    return User;
  };
}
modelos / role.model.js
module.exports = (sequelize, Sequelize) => {
  const Role = sequelize.define("roles", {
    id: {
      type: Sequelize.INTEGER,
      primaryKey: true
    },
    name: {
      type: Sequelize.STRING
    }
  });

  return Role;
};

```

Estos Sequelize Models representan la tabla **de usuarios y roles** en la base de datos MySQL.

Después de inicializar Sequelize, no necesitamos escribir funciones CRUD, Sequelize las admite todas:

- crear un nuevo usuario: `create(object)`
- encontrar un usuario por id: `findByPk(id)`
- encontrar un usuario por correo electrónico: `findOne({ where: { email: ... } })`
- obtener todos los usuarios: `findAll()`
- encontrar todos los usuarios por *nombre de usuario* : `findAll({ where: { username: ... } })`

Estas funciones se utilizarán en nuestros Controladores y Middlewares.

Inicializar Sequelize

Ahora crea **app / models / index.js** con contenido como este:

```

const config = require("../config/db.config.js");

const Sequelize = require("sequelize");
const sequelize = new Sequelize(
  config.DB,
  config.USER,
  config.PASSWORD,
  {
    host: config.HOST,
    dialect: config.dialect,
    operatorsAliases: false,

    pool: {
      max: config.pool.max,
      min: config.pool.min,
      acquire: config.pool.acquire,
      idle: config.pool.idle
    }
  }
);

const db = {};

db.Sequelize = Sequelize;
db.sequelize = sequelize;

db.user = require("../models/user.model.js")(sequelize, Sequelize);
db.role = require("../models/role.model.js")(sequelize, Sequelize);

```

```

db.role.belongsToMany(db.user, {
  through: "user_roles",
  foreignKey: "roleId",
  otherKey: "userId"
});
db.user.belongsToMany(db.role, {
  through: "user_roles",
  foreignKey: "userId",
  otherKey: "roleId"
});

db.ROLES = ["user", "admin", "moderator"];

module.exports = db;

```

La asociación entre *Usuarios* y *Roles* es una relación Muchos a Muchos:

- Un Usuario puede tener varios Roles.
- Un Rol puede ser asumido por muchos Usuarios.

Usamos `User.belongsToMany(Role)` para indicar que el modelo *de usuario* puede pertenecer a muchos *Role*s y viceversa.

Con `through`, `foreignKey`, `otherKey`, tendremos una nueva tabla **user_roles** como conexión entre **los usuarios** y la tabla **de roles** a través de su clave principal como claves externas.

Si desea conocer más detalles sobre cómo hacer una asociación de muchos a muchos con Sequelize y Node.js, visite:

[Ejemplo de asociación de muchos a muchos de Sequelize: Node.js y MySQL](#)

No olvide llamar al `sync()` método en *server.js* .

```

...
const app = express();
app.use(...);

const db = require("../app/models");
const Role = db.role;

db.sequelize.sync({force: true}).then(() => {
  console.log('Drop and Resync Db');
  initial();
});

...
function initial() {
  Role.create({
    id: 1,
    name: "user"
  });

  Role.create({
    id: 2,
    name: "moderator"
  });

  Role.create({
    id: 3,
    name: "admin"
  });
}

```

`initial()` La función nos ayuda a crear 3 filas en la base de datos.

En desarrollo, es posible que deba eliminar las tablas existentes y volver a sincronizar la base de datos. Así que puedes usarlo force: `true` como el código anterior.

Para la producción, simplemente inserte estas filas manualmente y utilícelas `sync()` sin parámetros para evitar perder datos:

```
...
const app = express();
app.use(...);

const db = require("../app/models");

db.sequelize.sync();
...
```

Aprenda a implementar Sequelize una relación de uno a muchos en:

[Sequelize Associations: One-to-Many example – Node.js, MySQL](#)

Configurar clave de autenticación

Las funciones `jsonwebtoken` como `verify()` o `sign()` usan un algoritmo que necesita una clave secreta (como String) para codificar y decodificar el token.

En la carpeta **app / config**, cree el archivo `auth.config.js` con el siguiente código:

```
module.exports = {
  secret: "bezkodeer-secret-key"
};
```

Puede crear su propio `secretString`.

Crear funciones de middleware

Para verificar una acción de registro, necesitamos 2 funciones:

- verificar si `username` o `email` está duplicado o no
- verificar si `roles` en la solicitud existe o no

middleware / `verificarSignUp.js`

```
const db = require("../models");
const ROLES = db.ROLES;
const User = db.user;

checkDuplicateUsernameOrEmail = (req, res, next) => {
  // Username
  User.findOne({
    where: {
      username: req.body.username
    }
  }).then(user => {
    if (user) {
      res.status(400).send({
        message: "Failed! Username is already in use!"
      });
      return;
    }

    // Email
    User.findOne({
      where: {
        email: req.body.email
      }
    }).then(email => {
      if (email) {
        res.status(400).send({
          message: "Failed! Email is already in use!"
        });
        return;
      }
    });
  });
  next();
}
```

```

    }
  }).then(user => {
    if (user) {
      res.status(400).send({
        message: "Failed! Email is already in use!"
      });
      return;
    }

    next();
  });
});
};

checkRolesExisted = (req, res, next) => {
  if (req.body.roles) {
    for (let i = 0; i < req.body.roles.length; i++) {
      if (!ROLES.includes(req.body.roles[i])) {
        res.status(400).send({
          message: "Failed! Role does not exist = " + req.body.roles[i]
        });
        return;
      }
    }
  }

  next();
};

const verifySignUp = {
  checkDuplicateUsernameOrEmail: checkDuplicateUsernameOrEmail,
  checkRolesExisted: checkRolesExisted
};

module.exports = verifySignUp;

```

Para procesar la Autenticación y Autorización, tenemos estas funciones:

- comprobar si token se proporciona, legal o no. Obtenemos el token de **x-access-token** de los encabezados HTTP, luego usamos la función **jsonwebtoken.verify()**.
- comprobar si roles el usuario contiene el rol requerido o no.

software intermedio / authJwt.js

```

const jwt = require("jsonwebtoken");
const config = require("../config/auth.config.js");
const db = require("../models");
const User = db.user;

verifyToken = (req, res, next) => {
  let token = req.headers["x-access-token"];

  if (!token) {
    return res.status(403).send({
      message: "No token provided!"
    });
  }

  jwt.verify(token, config.secret, (err, decoded) => {
    if (err) {
      return res.status(401).send({
        message: "Unauthorized!"
      });
    }
  });
}

```

```

    }
    req.userId = decoded.id;
    next();
  });
};

isAdmin = (req, res, next) => {
  User.findByPk(req.userId).then(user => {
    user.getRoles().then(roles => {
      for (let i = 0; i < roles.length; i++) {
        if (roles[i].name === "admin") {
          next();
          return;
        }
      }

      res.status(403).send({
        message: "Require Admin Role!"
      });
      return;
    });
  });
};

isModerator = (req, res, next) => {
  User.findByPk(req.userId).then(user => {
    user.getRoles().then(roles => {
      for (let i = 0; i < roles.length; i++) {
        if (roles[i].name === "moderator") {
          next();
          return;
        }
      }

      res.status(403).send({
        message: "Require Moderator Role!"
      });
    });
  });
};

isModeratorOrAdmin = (req, res, next) => {
  User.findByPk(req.userId).then(user => {
    user.getRoles().then(roles => {
      for (let i = 0; i < roles.length; i++) {
        if (roles[i].name === "moderator") {
          next();
          return;
        }

        if (roles[i].name === "admin") {
          next();
          return;
        }
      }

      res.status(403).send({
        message: "Require Moderator or Admin Role!"
      });
    });
  });
};

```

```

const authJwt = {
  verifyToken: verifyToken,
  isAdmin: isAdmin,
  isModerator: isModerator,
  isModeratorOrAdmin: isModeratorOrAdmin
};
module.exports = authJwt;
software intermedio / index.js
const authJwt = require("./authJwt");
const verifySignUp = require("./verifySignUp");

module.exports = {
  authJwt,
  verifySignUp
};

```

Crear controladores

Controlador de autenticación

Hay 2 funciones principales para la autenticación:

- signup: crear un nuevo usuario en la base de datos (el rol es **el usuario** si no se especifica el rol)
- signin:
 - encontrar username la solicitud en la base de datos, si existe
 - compare password con password en la base de datos usando **bcrypt**, si es correcto
 - generar un token usando **jsonwebtoken**
 - devolver información de usuario y token de acceso

```

controladores / auth.controller.js
const db = require("../models");
const config = require("../config/auth.config");
const User = db.user;
const Role = db.role;

const Op = db.Sequelize.Op;

var jwt = require("jsonwebtoken");
var bcrypt = require("bcryptjs");

exports.signup = (req, res) => {
  // Save User to Database
  User.create({
    username: req.body.username,
    email: req.body.email,
    password: bcrypt.hashSync(req.body.password, 8)
  })
  .then(user => {
    if (req.body.roles) {
      Role.findAll({
        where: {
          name: {
            [Op.or]: req.body.roles
          }
        }
      })
    }
  })
  .then(roles => {
    user.setRoles(roles).then(() => {
      res.send({ message: "User was registered successfully!" });
    });
  });
}

```



```

    });
  } else {
    // user role = 1
    user.setRoles([1]).then(() => {
      res.send({ message: "User was registered successfully!" });
    });
  }
})
.catch(err => {
  res.status(500).send({ message: err.message });
});
};

exports.signin = (req, res) => {
  User.findOne({
    where: {
      username: req.body.username
    }
  })
  .then(user => {
    if (!user) {
      return res.status(404).send({ message: "User Not found." });
    }

    var passwordIsValid = bcrypt.compareSync(
      req.body.password,
      user.password
    );

    if (!passwordIsValid) {
      return res.status(401).send({
        accessToken: null,
        message: "Invalid Password!"
      });
    }

    var token = jwt.sign({ id: user.id }, config.secret, {
      expiresIn: 86400 // 24 hours
    });

    var authorities = [];
    user.getRoles().then(roles => {
      for (let i = 0; i < roles.length; i++) {
        authorities.push("ROLE_" + roles[i].name.toUpperCase());
      }
      res.status(200).send({
        id: user.id,
        username: user.username,
        email: user.email,
        roles: authorities,
        accessToken: token
      });
    });
  })
  .catch(err => {
    res.status(500).send({ message: err.message });
  });
};

```

Controlador para pruebas Autorización

Hay 4 funciones:

- /api/test/all para acceso público
- /api/test/user para usuarios registrados (función: **usuario** / **moderador** / **administrador**)
- /api/test/mod para usuarios con función **de moderador**
- /api/test/admin para usuarios con función **de administrador**

controladores / *usuario.controlador.js*

```
exports.allAccess = (req, res) => {
  res.status(200).send("Public Content.");
};

exports.userBoard = (req, res) => {
  res.status(200).send("User Content.");
};

exports.adminBoard = (req, res) => {
  res.status(200).send("Admin Content.");
};

exports.moderatorBoard = (req, res) => {
  res.status(200).send("Moderator Content.");
};
```

Ahora, ¿tienes alguna pregunta? ¿Le gustaría saber cómo podemos combinar middlewares con funciones de controlador?

Hagámoslo en la siguiente sección.

Definir rutas

Cuando un cliente envía una solicitud para un punto final mediante una solicitud HTTP (GET, POST, PUT, DELETE), debemos determinar cómo responderá el servidor configurando las rutas.

Podemos separar nuestras rutas en 2 partes: para Autenticación y para Autorización (acceso a recursos protegidos).

Autenticación:

- CORREO/api/auth/signup
- CORREO/api/auth/signin

rutas / *auth.routes.js*

```
const { verifySignUp } = require("../middleware");
const controller = require("../controllers/auth.controller");

module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });

  app.post(
    "/api/auth/signup",
    [
      verifySignUp.checkDuplicateUsernameOrEmail,
```

```

        verifySignUp.checkRolesExisted
    ],
    controller.signup
);

app.post("/api/auth/signin", controller.signin);
};

```

Autorización:

- CONSEGUIR /api/test/all
- GET /api/test/user para usuarios registrados (usuario/moderador/administrador)
- OBTENER /api/test/mod para moderador
- OBTENER /api/test/admin para administrador

rutas / *usuario.routes.js*

```

const { authJwt } = require("../middleware");
const controller = require("../controllers/user.controller");

module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });

  app.get("/api/test/all", controller.allAccess);

  app.get(
    "/api/test/user",
    [authJwt.verifyToken],
    controller.userBoard
  );

  app.get(
    "/api/test/mod",
    [authJwt.verifyToken, authJwt.isModerator],
    controller.moderatorBoard
  );

  app.get(
    "/api/test/admin",
    [authJwt.verifyToken, authJwt.isAdmin],
    controller.adminBoard
  );
};

```

No olvide agregar estas rutas en *server.js* :

```

...
// routes
require('./app/routes/auth.routes')(app);
require('./app/routes/user.routes')(app);

// set port, listen for requests
...

```

Ejecutar y probar con resultados

Ejecute la aplicación Node.js con el comando: `node server.js`

Las tablas que definimos en el paquete *de modelos* se generarán automáticamente en la base de datos MySQL.

Si revisa la base de datos, puede ver cosas como esta:

```
mysql> describe users;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
username	varchar(255)	YES		NULL	
email	varchar(255)	YES		NULL	
password	varchar(255)	YES		NULL	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

```
mysql> describe roles;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
name	varchar(255)	YES		NULL	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

```
mysql> describe user_roles;
```

Field	Type	Null	Key	Default	Extra
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	
roleId	int(11)	NO	PRI	NULL	
userId	int(11)	NO	PRI	NULL	

Si no usa `initial()` la función en el método `Sequelize sync()`. Debe ejecutar el siguiente script SQL:

```
mysql> INSERT INTO roles VALUES (1, 'user', now(), now());
```

```
mysql> INSERT INTO roles VALUES (2, 'moderator', now(), now());
```

```
mysql> INSERT INTO roles VALUES (3, 'admin', now(), now());
```

Se crearán 3 registros en rolesla tabla:

```
mysql> select * from roles;
```

id	name	createdAt	updatedAt
1	user	2020-01-13 09:05:39	2020-01-13 09:05:39
2	moderator	2020-01-13 09:05:39	2020-01-13 09:05:39
3	admin	2020-01-13 09:05:39	2020-01-13 09:05:39

Registrar algunos usuarios con /signupAPI:

- **administrador** con adminrol
- **mod** con moderatory userroles
- **zkoder** con userrol

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/api/auth/signup`. The request body is a JSON object with the following fields: `username: "mod"`, `email: "mod@bezkode.com"`, `password: "12345678"`, and `roles: ["moderator", "user"]`. The response status is `200 OK` with a time of `645ms` and a size of `401 B`. The response body is a JSON object with the field `message: "User was registered successfully!"`.

```
POST http://localhost:8080/api/auth/signup
```

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL BETA

```
1 {
2   "username": "mod",
3   "email": "mod@bezkode.com",
4   "password": "12345678",
5   "roles": ["moderator", "user"]
6 }
```

Body Cookies Headers (9) Test Results Status: 200 OK Time: 645ms Size: 401 B

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "message": "User was registered successfully!"
3 }
```

Acceder al recurso público: GET/api/test/all

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/api/test/all`. The response status is `200 OK` with a time of `50ms` and a size of `361 B`. The response body is the text `Public Content.`.

```
GET http://localhost:8080/api/test/all
```

Body Cookies Headers (9) Test Results Status: 200 OK Time: 50ms Size: 361 B

Pretty Raw Preview Visualize BETA HTML

```
1 Public Content.
```

Acceso al recurso protegido: GET/api/test/user

GET http://localhost:8080/api/test/user **Send**

Params Authorization **Headers (7)** Body Pre-request Script Tests Settings

▼ Headers (0)

KEY	VALUE	DESCRIPTION
Key	Value	Description

► Temporary Headers (7) ⓘ

Body Cookies Headers (9) Test Results Status: 403 Forbidden Time: 42ms Size: 406 B

Pretty Raw Preview Visualize BETA JSON

```
1 {  
2   "message": "No token provided!"  
3 }
```

Iniciar sesión en una cuenta (con contraseña incorrecta): POST/api/auth/signin

POST http://localhost:8080/api/auth/signin **Send**

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL BETA

```
1 {  
2   "username": "mod",  
3   "password": "123456789"  
4 }
```

Body Cookies Headers (9) Test Results Status: 401 Unauthorized Time: 145ms Size: 427 B

Pretty Raw Preview Visualize BETA JSON

```
1 {  
2   "accessToken": null,  
3   "message": "Invalid Password!"  
4 }
```

Inicie sesión en una cuenta: POST/api/auth/signin

POST

http://localhost:8080/api/auth/signin

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL BETA

1 {

2 "username": "mod",

3 "password": "12345678"

4 }

Body

Cookies

Headers (9)

Test Results

Status: 200 OK

Time: 107ms

Size: 622 B

Pretty

Raw

Preview

Visualize BETA

JSON

1 {

2 "id": 2,

3 "username": "mod",

4 "email": "mod@bezkoder.com",

5 "roles": [

6 "ROLE_USER",

7 "ROLE_MODERATOR"

8],

9 "accessToken":

10 "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiYWV0IjoxNTc4OTUwMTU2LTY1NTZ9.houWz7W6WDJ3yoUpmsdVNzr2Vn1c9wAAP09tjYGdOLk"

Acceder a recursos protegidos: GET/api/test/user

GET

http://localhost:8080/api/test/user

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

▼ Headers (1)

	KEY	VALUE	DESCRIPTION	Bulk
<input checked="" type="checkbox"/>	x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC...		
	Key	Value	Description	

► Temporary Headers (7) ⓘ

Body

Cookies

Headers (9)

Test Results

Status: 200 OK

Time: 23ms

Size: 359 B

Pretty

Raw

Preview

Visualize BETA

HTML

1 User Content.

GET

http://localhost:8080/api/test/mod

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

▼ Headers (1)

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC...	
	Key	Value	Description

► Temporary Headers (7)

Body

Cookies

Headers (9)

Test Results

Status: 200 OK

Time: 68ms

Size: 365 B

Pretty

Raw

Preview

Visualize BETA

HTML

1

Moderator Content.

GET

http://localhost:8080/api/test/admin

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

▼ Headers (1)

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC...	
	Key	Value	Description

► Temporary Headers (7)

Body

Cookies

Headers (9)

Test Results

Status: 403 Forbidden

Time: 69ms

Size: 373 B

Pretty

Raw

Preview

Visualize BETA

HTML

1

{

2

"message": "Require Admin Role!"

3

}

Conclusión

¡Felicidades!

Hoy hemos aprendido muchas cosas interesantes sobre la autenticación basada en token de Node.js con JWT: JSONWebToken en solo un ejemplo de Node.js Express Rest Api.

A pesar de que escribimos mucho código, espero que comprenda la arquitectura general de la aplicación y la aplique en su proyecto con facilidad.

Debería seguir sabiendo cómo implementar Refresh Token:

Implementación

Puede encontrar el paso a paso para implementar esta aplicación Node.js en la publicación:

Node.js: ejemplo de autenticación y autorización JWT con JSONWebToken

<https://www.bezkoder.com/node-js-jwt-authentication-mysql/>



Para la base de datos MongoDB:

Node.js + MongoDB: autenticación y autorización de usuario con JWT

<https://www.bezkoder.com/node-js-mongodb-auth-jwt/>

O base de datos PostgreSQL:

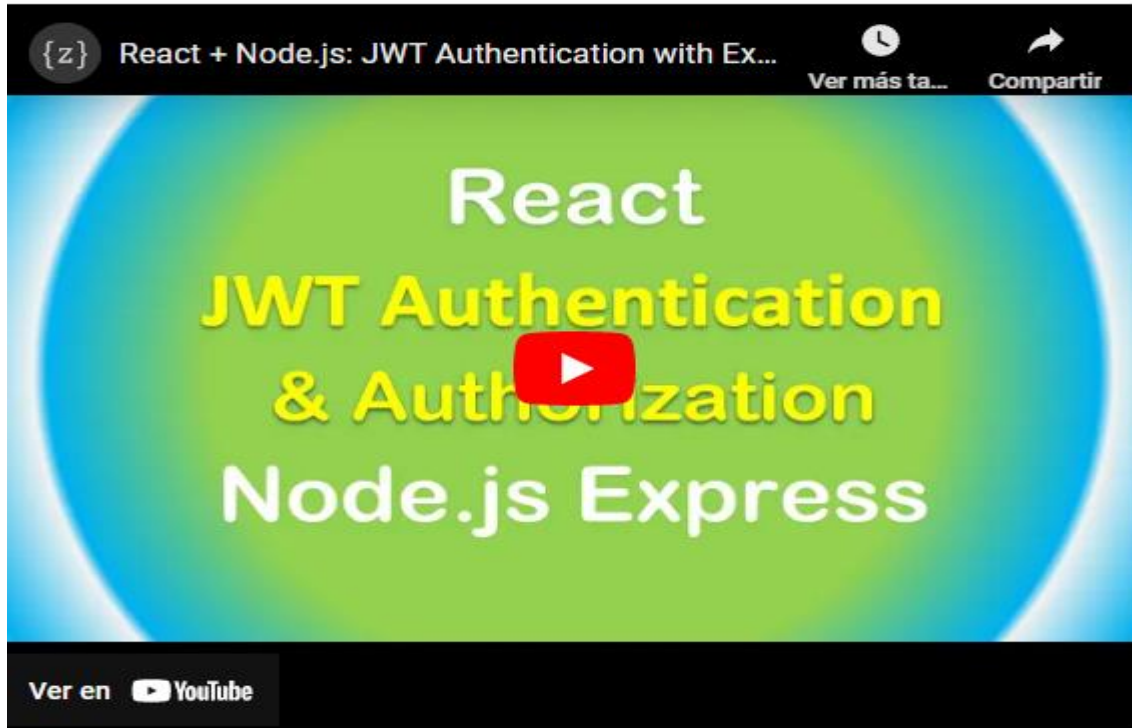
Node.js + PostgreSQL: autenticación y autorización de usuarios con JWT

<https://www.bezkoder.com/node-js-jwt-authentication-postgresql/>

Interfaz con React, React Router

=====

Video demostración



<https://www.youtube.com/watch?v=tNcWX9qPcCM>

=====

Ejemplo de autenticación React JWT (sin Redux)

Última modificación: 22 de septiembre de 2022 bezkoder Reaccionar , Seguridad

En este tutorial, vamos a crear un ejemplo de autenticación: inicio de sesión y registro de React.js JWT con LocalStorage, React Router, Axios y Bootstrap (sin Redux). Yo te mostraré:

- Flujo de autenticación JWT para registro de usuario e inicio de sesión de usuario
- Estructura del proyecto para la autenticación React JWT (sin Redux) con LocalStorage, React Router y Axios
- Creación de componentes React con validación de formulario
- Componentes de React para acceder a Recursos protegidos (Autorización)
- Barra de navegación dinámica en la aplicación React

Contenidos

- Descripción general del ejemplo de autenticación de React JWT
- Registro de usuario y flujo de inicio de sesión de usuario
- Diagrama de componentes de React con enrutador, Axios y almacenamiento local
- Tecnología
- Estructura del proyecto
- Configurar el proyecto React.js
- Agregar enrutador React
- Importar Bootstrap
- Crear servicios
 - Servicio de autenticación
 - Servicio de datos
- Crear componentes de reacción para la autenticación
 - Descripción general de la validación de formularios
 - Página de inicio de sesión
 - Página de registro
 - Página de perfil
- Crear componentes de React para acceder a los recursos
 - Página de inicio
 - Páginas basadas en roles
- Agregar barra de navegación y definir rutas
- Cerrar sesión cuando el token haya caducado
- Agregar estilo CSS para componentes React
- Configure el puerto para el cliente de autenticación React JWT con API web
- Conclusión
- Otras lecturas

Descripción general del ejemplo de autenticación de React JWT

Construiremos una aplicación React en eso:

- Hay páginas de inicio/cierre de sesión, registro.
- Los datos del formulario serán validados por el front-end antes de enviarse al back-end.
- Dependiendo de las funciones del usuario (administrador, moderador, usuario), la barra de navegación cambia sus elementos automáticamente.

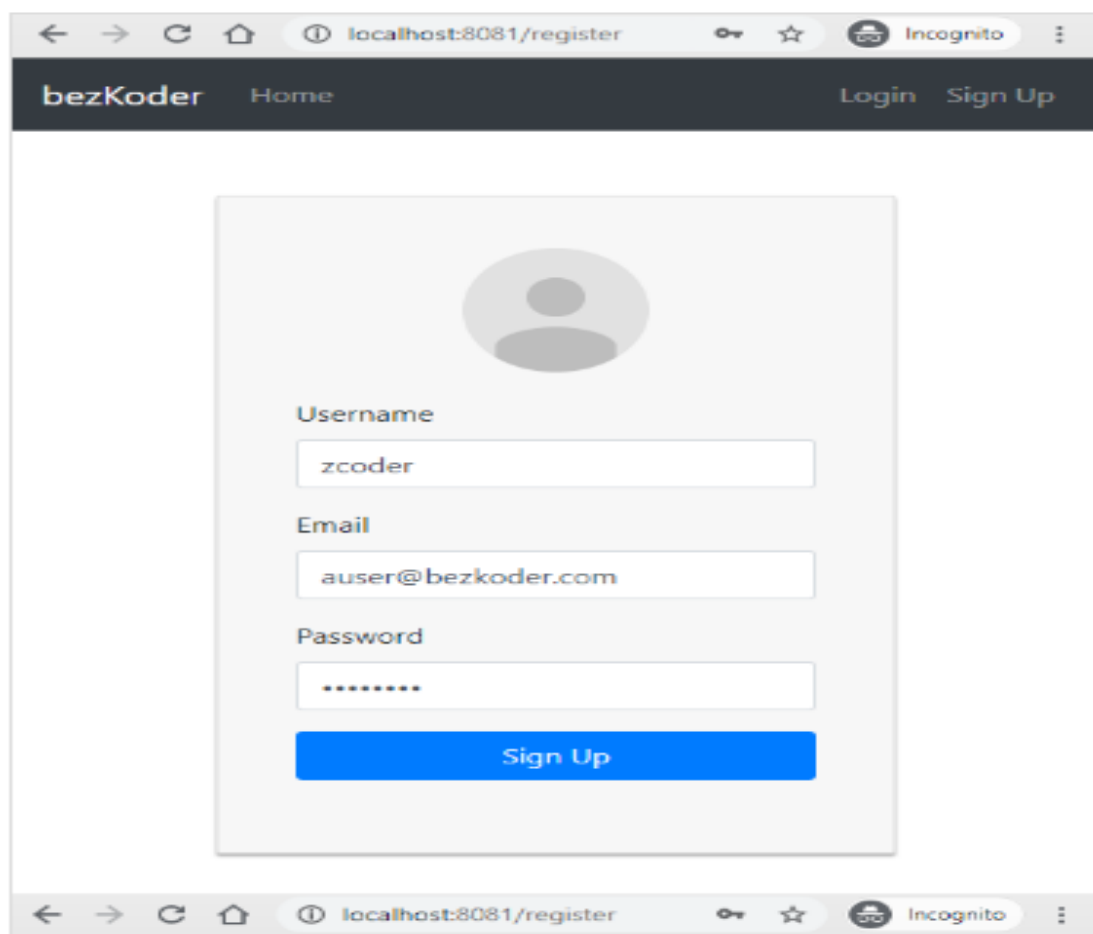
Aquí están las capturas de pantalla:

– Página de registro:

– Cualquiera puede acceder a una página pública antes de iniciar sesión:

Public Content.

Un nuevo Usuario puede registrarse:



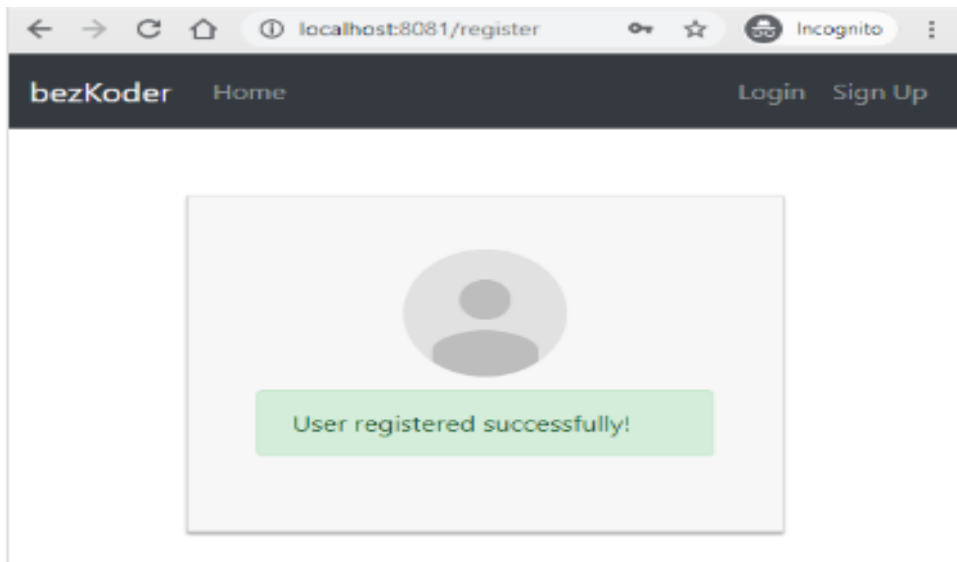
The screenshot shows a web browser window with the URL `localhost:8081/register`. The page features a dark header with the bezKoder logo and navigation links. The main content area contains a registration form with a user icon placeholder, input fields for Username, Email, and Password, and a blue Sign Up button.

Username

Email

Password

[Sign Up](#)



– La validación del registro de formulario será así:

A registration form with a light gray background and a circular user icon placeholder at the top. It contains three input fields, each with a validation error message in a red box below it. The first field is 'Username' with the value 'zk' and the error 'The username must be between 3 and 20 characters.' The second field is 'Email' with the value 'user@bezkoder' and the error 'This is not a valid email.' The third field is 'Password' with five dots and the error 'The password must be between 6 and 40 characters.' At the bottom is a blue 'Sign Up' button.

– Después de que el registro sea exitoso, el usuario puede iniciar sesión:

A screenshot of a web browser window. The address bar shows 'localhost:8081/login'. The browser is in Incognito mode. The page has a dark header with 'bezKoder' and 'Home' on the left, and 'Login' and 'Sign Up' on the right. The main content area is a light gray box containing a gray user icon, a 'Username' label, a text input field with 'zcoder', a 'Password' label, a password input field with dots, and a blue 'Login' button.

- Después de iniciar sesión, la aplicación dirige al usuario a la página **de perfil** :

← → ↻ 🏠 ⓘ localhost:8081/profile ☆ 🍷 Incognito ⋮

bezKoder Home User zcoder LogOut

zcoder Profile

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZWl0eSIsImN1b250IjoiZm9udCIsImV4cCI6MTY1MjQ0MDAwfQ.eyJ1b250IjoiZm9udCIsImN1b250IjoiZm9udCIsImV4cCI6MTY1MjQ0MDAwfQ.9nNDFCiap0dP3jWKJKXM

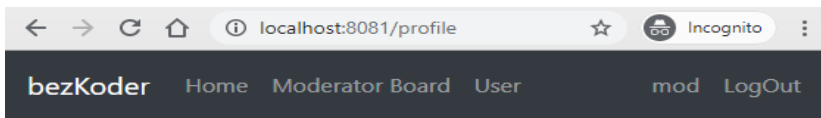
Id: 3

Email: auser@bezkoder.com

Authorities:

- `ROLE_USER`

– Interfaz de usuario para el inicio de sesión **del moderador** (la barra de navegación cambiará según las autoridades):



mod Profile

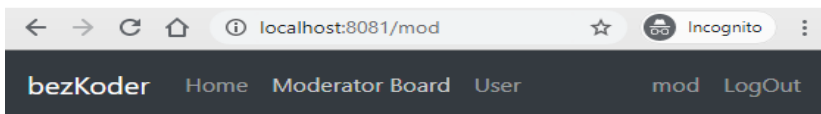
Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLT...

Id: 2

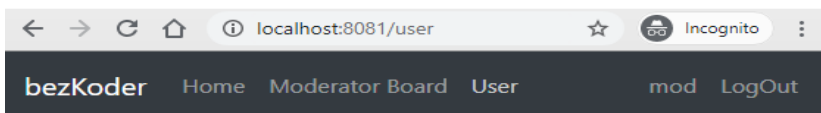
Email: mod@bezkoder.com

Authorities:

- ROLE_USER
- ROLE_MODERATOR

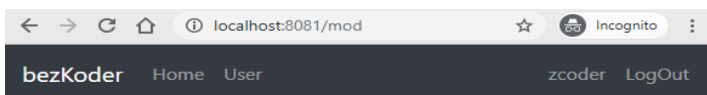


Moderator Content.

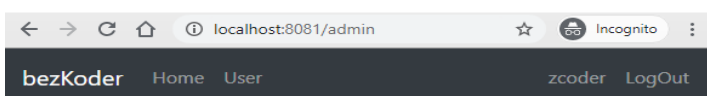


User Content.

– Si un usuario que no tiene la función de administrador intenta acceder a la página del panel de **administración / moderador** :

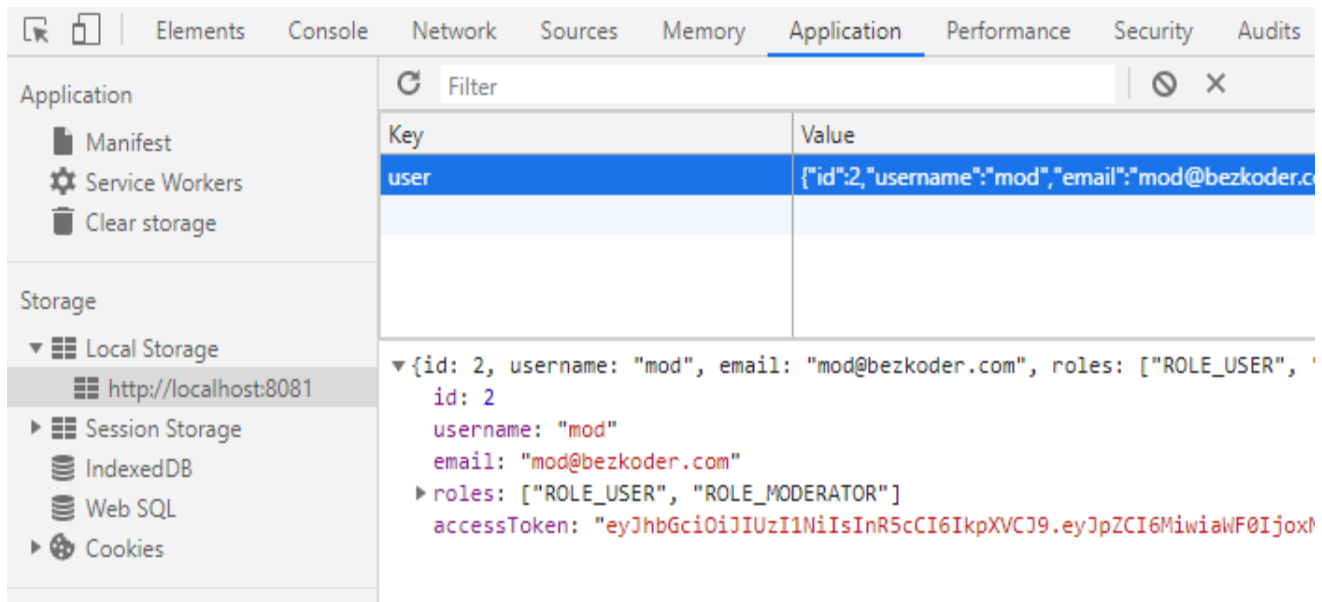


Require Moderator Role!



Require Admin Role!

– Compruebe el almacenamiento local del navegador:



Si desea agregar un token de actualización, visite:

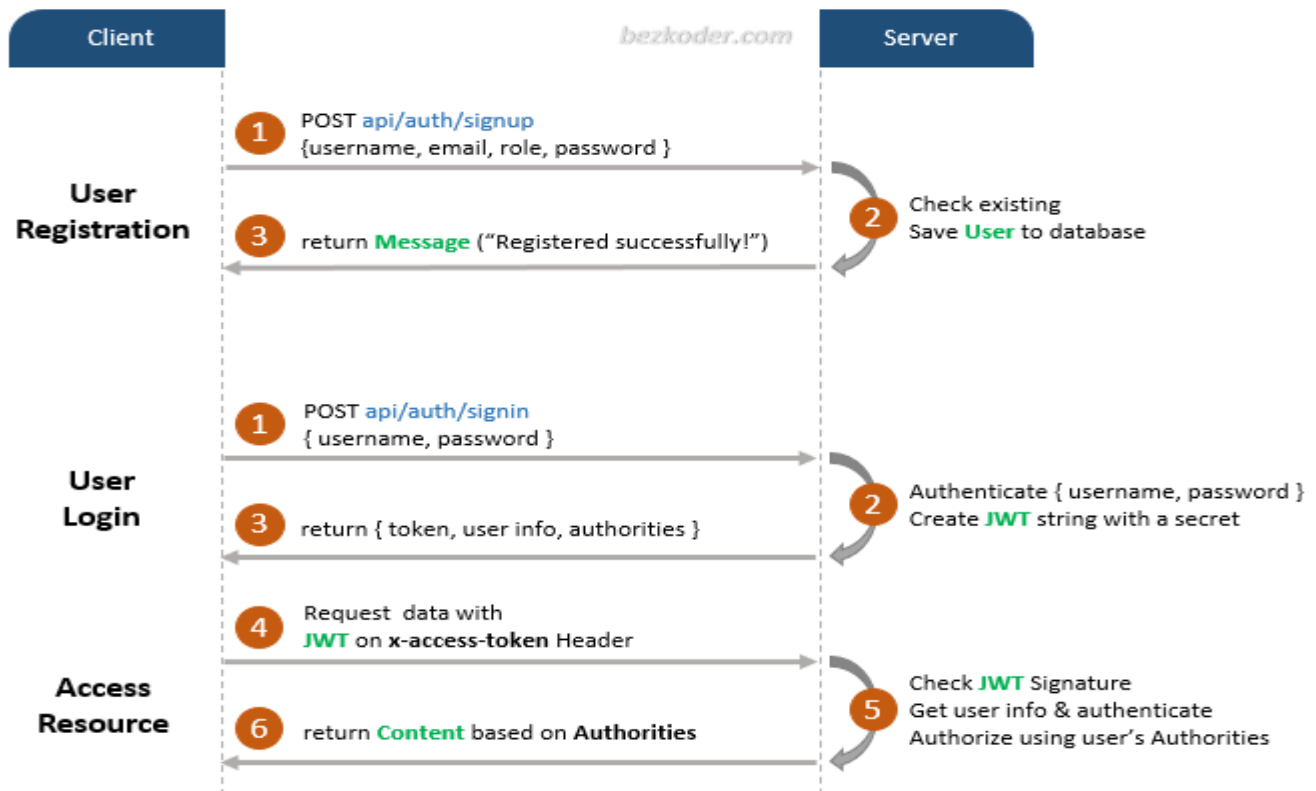
React Refresh Token con JWT y Axios Interceptors

<https://www.bezkoder.com/react-refresh-token/>



Flujo para registro de usuario e inicio de sesión de usuario

El diagrama muestra el flujo del proceso de registro de usuario, inicio de sesión de usuario y autorización.



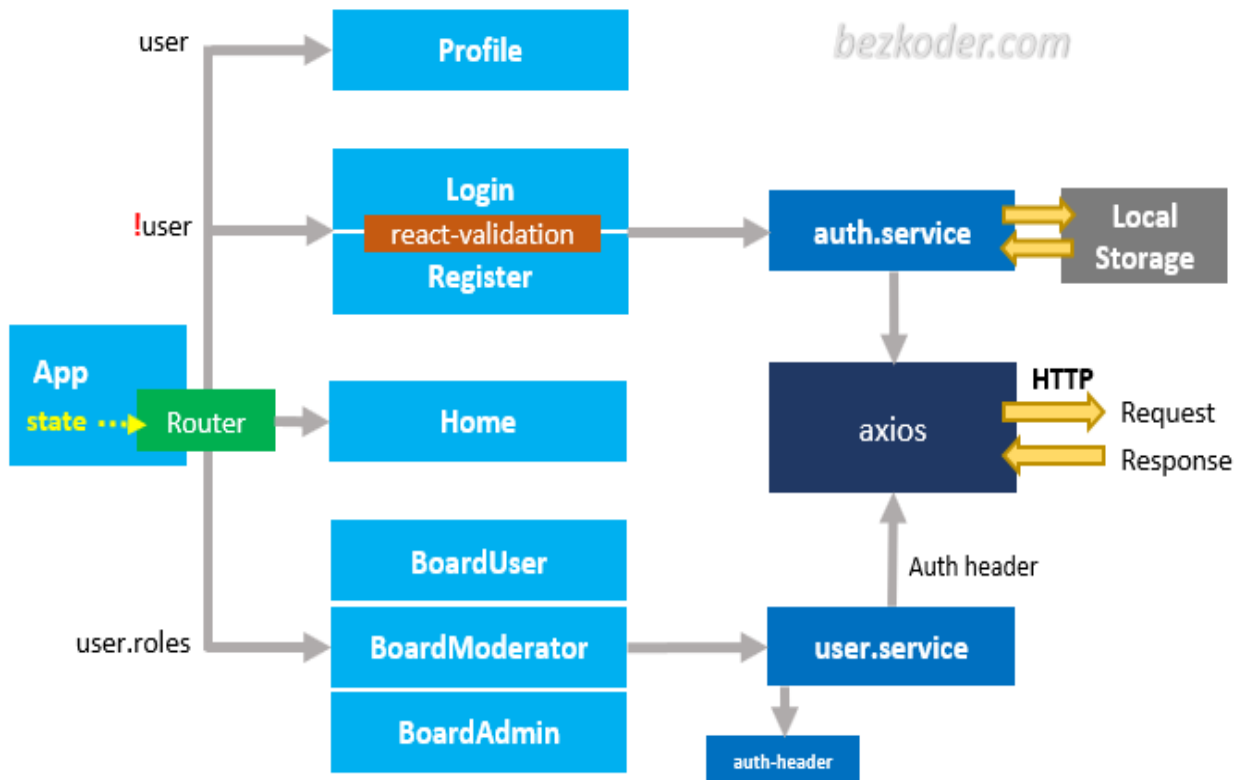
Hay 2 puntos finales para la autenticación:

- `api/auth/signup` para registro de usuario
- `api/auth/signin` para inicio de sesión de usuario

Si el cliente desea enviar una solicitud a puntos finales/datos protegidos, agrega JWT legal al encabezado HTTP **x-access-token**.

Descripción general

Veamos el siguiente diagrama.



- El App componente es un contenedor con React Router (BrowserRouter). Según el estado, la barra de navegación puede mostrar sus elementos.
- Login & Register los componentes tienen formulario para el envío de datos (con soporte de react-validation biblioteca). Llamamos a métodos desde auth.service para realizar una solicitud de inicio de sesión/registro.
- auth.service métodos utilizados axios para realizar solicitudes HTTP. También almacena u obtiene **JWT** del almacenamiento local del navegador dentro de estos métodos.
- Home el componente es público para todos los visitantes.
- Profile el componente muestra la información del usuario después de que la acción de inicio de sesión sea exitosa.
- BoardUser, BoardModerator, BoardAdmin los componentes se mostrarán por estado user.roles. En estos componentes, usamos user.service para acceder a recursos protegidos desde Web API.
- user.service utiliza auth-header() la función auxiliar para agregar JWT al encabezado HTTP. auth-header() devuelve un objeto que contiene el JWT del usuario actualmente conectado desde el almacenamiento local.

Si desea utilizar HttpOnly Cookie para almacenar JWT, visite:

Ejemplo de inicio de sesión y registro de React.js: JWT y HttpOnly Cookie

<https://www.bezkoder.com/react-login-example-jwt-hooks/>

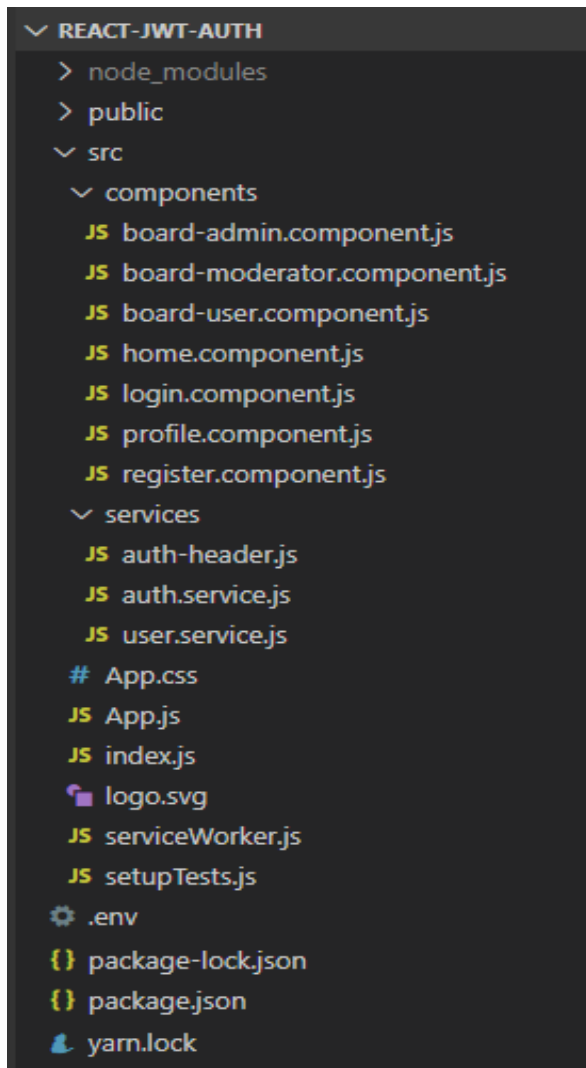
Tecnología

Vamos a usar estos módulos:

- React 18/17
- react-router-dom 6
- axios 0.27.2
- react-validation 3.0.7
- Bootstrap 4
- validator 13.7.0

Estructura del proyecto

Esta es la estructura de carpetas y archivos para esta aplicación React:



Con la explicación en el diagrama anterior, puede comprender fácilmente la estructura del proyecto.

Configurar el proyecto React.js

Abra cmd en la carpeta en la que desea guardar la carpeta del proyecto, ejecute el comando:
npx create-react-app react-jwt-auth

Agregar enrutador React

- Ejecute el comando: npm install react-router-dom.
- Abra **src** / *index.js* y ajuste App por BrowserRouter objeto.

```
import React from "react";
import { createRoot } from "react-dom/client";
import { BrowserRouter } from "react-router-dom";

import App from "./App";

const container = document.getElementById("root");
const root = createRoot(container);

root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

Importar Bootstrap

Ejecutar comando: npm install bootstrap@4.6.2.

Abra **src** / *App.js* y modifique el código que contiene de la siguiente manera:

```
import React, { Component } from "react";
import "bootstrap/dist/css/bootstrap.min.css";

class App extends Component {
  render() {
    // ...
  }
}

export default App;
```

Crear servicios

Vamos a crear dos servicios en la carpeta **src** / **services** :

servicios

auth-header.js

auth.service.js (*servicio de autenticación*)

user.service.js (*servicio de datos*)

Antes de trabajar con estos servicios, necesitamos instalar Axios con el comando:
npm install axios

Servicio de autenticación

El servicio utiliza Axios para solicitudes HTTP y almacenamiento local para información de usuario y JWT.

Proporciona los siguientes métodos importantes:

- `login()`: POSTE {nombre de usuario, contraseña} y guárdelo JWT en el almacenamiento local
- `logout()`: eliminar JWT del almacenamiento local
- `register()`: POST {nombre de usuario, correo electrónico, contraseña}
- `getCurrentUser()`: obtener información de usuario almacenada (incluido JWT)

```
import axios from "axios";

const API_URL = "http://localhost:8080/api/auth/";

class AuthService {
  login(username, password) {
    return axios
      .post(API_URL + "signin", {
        username,
        password
      })
      .then(response => {
        if (response.data.accessToken) {
          localStorage.setItem("user", JSON.stringify(response.data));
        }

        return response.data;
      });
  }

  logout() {
    localStorage.removeItem("user");
  }

  register(username, email, password) {
    return axios.post(API_URL + "signup", {
      username,
      email,
      password
    });
  }

  getCurrentUser() {
    return JSON.parse(localStorage.getItem('user'));
  }
}

export default new AuthService();
```

Servicio de datos

También tenemos métodos para recuperar datos del servidor. En el caso de que accedamos a recursos protegidos, la solicitud HTTP necesita un encabezado de autorización.

Vamos a crear una función auxiliar llamada `authHeader()` dentro de `auth-header.js` :

```
export default function authHeader() {
  const user = JSON.parse(localStorage.getItem('user'));

  if (user && user.accessToken) {
    return { Authorization: 'Bearer ' + user.accessToken };
  } else {
    return {};
  }
}
```

El código anterior verifica el almacenamiento local para `user` el artículo. Si hay un inicio de sesión `user` con `accessToken` (JWT), devuelva el encabezado de autorización HTTP. De lo contrario, devuelve un objeto vacío.

Nota: Para el back-end de Node.js Express, utilice el encabezado **x-access-token** como este:

```
export default function authHeader() {
  const user = JSON.parse(localStorage.getItem('user'));

  if (user && user.accessToken) {
    // for Node.js Express back-end
    return { 'x-access-token': user.accessToken };
  } else {
    return {};
  }
}
```

Ahora definimos un servicio para acceder a los datos en `user.service.js` :

```
import axios from 'axios';
import authHeader from './auth-header';

const API_URL = 'http://localhost:8080/api/test/';

class UserService {
  getPublicContent() {
    return axios.get(API_URL + 'all');
  }

  getUserBoard() {
    return axios.get(API_URL + 'user', { headers: authHeader() });
  }

  getModeratorBoard() {
    return axios.get(API_URL + 'mod', { headers: authHeader() });
  }

  getAdminBoard() {
    return axios.get(API_URL + 'admin', { headers: authHeader() });
  }
}
```

```
export default new UserService();
```

Puede ver que agregamos un encabezado HTTP con la ayuda de `authHeader()` la función al solicitar un recurso autorizado.

Crear componentes de reacción para la autenticación

En la carpeta **src** , cree una nueva carpeta con el nombre de **componentes** y agregue varios archivos de la siguiente manera:

componentes

login.component.js

registro.componente.js

perfil.componente.js

Descripción general de la validación de formularios

Ahora necesitamos una biblioteca para la validación de formularios, por lo que agregaremos la biblioteca *de validación de reacción* a nuestro proyecto.

Ejecute el comando: `npm install react-validation validator`

Para usar la validación de react en este ejemplo, debe importar los siguientes elementos:

```
import Form from "react-validation/build/form";
import Input from "react-validation/build/input";
import CheckButton from "react-validation/build/button";
```

```
import { isEmail } from "validator";
```

También usamos `isEmail()` la función del **validador** para verificar el correo electrónico. Así es como los ponemos en `render()` método con `validations` atributo:

```
const required = value => {
  if (!value) {
    return (
      <div className="alert alert-danger" role="alert">
        This field is required!
      </div>
    );
  }
};
```

```
const email = value => {
  if (!isEmail(value)) {
    return (
      <div className="alert alert-danger" role="alert">
        This is not a valid email.
      </div>
    );
  }
};
```

```
render() {
  return (
    ...
    <Form
      onSubmit={this.handleLogin}
      ref={c => {this.form = c;}}
    >
    ...
  );
}
```

```

    <Input
      type="text"
      className="form-control"
      ...
      validations={[required, email]}
    />

    <Checkbox
      style={{ display: "none" }}
      ref={c => {this.checkBtn = c;}}
    />
  </Form>
  ...
);
}

```

Vamos a llamar `validateAll()` al método `Form` para verificar las funciones de validación en `validations`. Luego `Checkbox` nos ayuda a verificar si la validación del formulario es exitosa o no. Por lo tanto, este botón no se mostrará en el formulario.

```
this.form.validateAll();
```

```

if (this.checkBtn.context._errors.length === 0) {
  // do something when no error
}

```

Esta es otra forma de implementar la validación de formularios:
[ejemplo de validación de formularios de React con Formik y Yup](#)

Admite funciones de enrutador

Desde `react-router-dom` v6, el soporte para `history` ha quedado obsoleto. Entonces necesitamos un contenedor (HOC) que pueda usar nuevos ganchos útiles.

En la carpeta **src**, cree un archivo **common** / *with-router.js* con el siguiente código:

```

import { useLocation, useNavigate, useParams } from "react-router-dom";

export const withRouter = (Component) => {
  function ComponentWithRouterProp(props) {
    let location = useLocation();
    let navigate = useNavigate();
    let params = useParams();
    return <Component {...props} router={{ location, navigate, params }} />;
  }
  return ComponentWithRouterProp;
};

```

Página de inicio de sesión

Esta página tiene un Formulario con `username` & `password`.

- Los vamos a verificar como campo *obligatorio*.
- Si la verificación está bien, llamamos al `AuthService.login()` método, luego dirigimos al usuario a la página **de Perfil**: `this.props.router.navigate("/profile");` o mostramos `message` con error de respuesta.

login.component.js

```
import React, { Component } from "react";
import Form from "react-validation/build/form";
import Input from "react-validation/build/input";
import CheckButton from "react-validation/build/button";

import AuthService from "../services/auth.service";

import { withRouter } from '../common/with-router';

const required = value => {
  if (!value) {
    return (
      <div className="alert alert-danger" role="alert">
        This field is required!
      </div>
    );
  }
};

class Login extends Component {
  constructor(props) {
    super(props);
    this.handleLogin = this.handleLogin.bind(this);
    this.onChangeUsername = this.onChangeUsername.bind(this);
    this.onChangePassword = this.onChangePassword.bind(this);

    this.state = {
      username: "",
      password: "",
      loading: false,
      message: ""
    };
  }

  onChangeUsername(e) {
    this.setState({
      username: e.target.value
    });
  }

  onChangePassword(e) {
    this.setState({
      password: e.target.value
    });
  }

  handleLogin(e) {
    e.preventDefault();

    this.setState({
      message: "",
      loading: true
    });

    this.form.validateAll();

    if (this.checkBtn.context._errors.length === 0) {
      AuthService.login(this.state.username, this.state.password).then(
        () => {
          this.props.router.navigate("/profile");
        }
      );
    }
  }
}
```

```

        window.location.reload();
    },
    error => {
        const resMessage =
            (error.response &&
                error.response.data &&
                error.response.data.message) ||
            error.message ||
            error.toString();

        this.setState({
            loading: false,
            message: resMessage
        });
    }
    );
} else {
    this.setState({
        loading: false
    });
}
}

render() {
    return (
        <div className="col-md-12">
            <div className="card card-container">
                

                <Form
                    onSubmit={this.handleLogin}
                    ref={c => {
                        this.form = c;
                    }}
                >
                    <div className="form-group">
                        <label htmlFor="username">Username</label>
                        <Input
                            type="text"
                            className="form-control"
                            name="username"
                            value={this.state.username}
                            onChange={this.onChangeUsername}
                            validations={[required]}
                        />
                    </div>

                    <div className="form-group">
                        <label htmlFor="password">Password</label>
                        <Input
                            type="password"
                            className="form-control"
                            name="password"
                            value={this.state.password}
                            onChange={this.onChangePassword}
                            validations={[required]}
                        />
                    </div>
                </Form>
            </div>
        </div>
    );
}

```

```

    <div className="form-group">
      <button
        className="btn btn-primary btn-block"
        disabled={this.state.loading}
      >
        {this.state.loading && (
          <span className="spinner-border spinner-border-sm"></span>
        )}
        <span>Login</span>
      </button>
    </div>

    {this.state.message && (
      <div className="form-group">
        <div className="alert alert-danger" role="alert">
          {this.state.message}
        </div>
      </div>
    )}
    <CheckButton
      style={{ display: "none" }}
      ref={c => {
        this.checkBtn = c;
      }}
    />
  </Form>
</div>
</div>
);
}
}

export default withRouter(Login);

```

Página de registro

Esta página es similar a **la página de inicio de sesión** .

Para la validación de formularios, hay algunos detalles más:

- username: requerido, entre 3 y 20 caracteres
- email: requerido, formato de correo electrónico
- password: requerido, entre 6 y 40 caracteres

Vamos a llamar `AuthService.register()` al método y mostrar el mensaje de respuesta (exitoso o error).

registro.componente.js

```

import React, { Component } from "react";
import Form from "react-validation/build/form";
import Input from "react-validation/build/input";
import CheckButton from "react-validation/build/button";
import { isEmail } from "validator";

import AuthService from "../services/auth.service";

const required = value => {
  if (!value) {
    return (
      <div className="alert alert-danger" role="alert">

```

```

        This field is required!
    </div>
  );
}
};

const email = value => {
  if (!isEmail(value)) {
    return (
      <div className="alert alert-danger" role="alert">
        This is not a valid email.
      </div>
    );
  }
};

const vusername = value => {
  if (value.length < 3 || value.length > 20) {
    return (
      <div className="alert alert-danger" role="alert">
        The username must be between 3 and 20 characters.
      </div>
    );
  }
};

const vpassword = value => {
  if (value.length < 6 || value.length > 40) {
    return (
      <div className="alert alert-danger" role="alert">
        The password must be between 6 and 40 characters.
      </div>
    );
  }
};

export default class Register extends Component {
  constructor(props) {
    super(props);
    this.handleRegister = this.handleRegister.bind(this);
    this.onChangeUsername = this.onChangeUsername.bind(this);
    this.onChangeEmail = this.onChangeEmail.bind(this);
    this.onChangePassword = this.onChangePassword.bind(this);

    this.state = {
      username: "",
      email: "",
      password: "",
      successful: false,
      message: ""
    };
  }

  onChangeUsername(e) {
    this.setState({
      username: e.target.value
    });
  }

  onChangeEmail(e) {
    this.setState({
      email: e.target.value

```

```

    });
}

onChangePassword(e) {
    this.setState({
        password: e.target.value
    });
}

handleRegister(e) {
    e.preventDefault();

    this.setState({
        message: "",
        successful: false
    });

    this.form.validateAll();

    if (this.checkBtn.context._errors.length === 0) {
        AuthService.register(
            this.state.username,
            this.state.email,
            this.state.password
        ).then(
            response => {
                this.setState({
                    message: response.data.message,
                    successful: true
                });
            },
            error => {
                const resMessage =
                    (error.response &&
                     error.response.data &&
                     error.response.data.message) ||
                    error.message ||
                    error.toString();

                this.setState({
                    successful: false,
                    message: resMessage
                });
            }
        );
    }
}

render() {
    return (
        <div className="col-md-12">
            <div className="card card-container">
                

                <Form
                    onSubmit={this.handleRegister}
                    ref={c => {
                        this.form = c;
                    }}
                >

```



```

        ref={c => {
          this.checkBtn = c;
        }}
      />
    </Form>
  </div>
</div>
);
}
}

```

Página de perfil

Esta página obtiene el usuario actual del almacenamiento local llamando al `AuthService.getCurrentUser()` método y muestra la información del usuario (con token). Si el usuario no ha iniciado sesión, redirigir a `/home` la página.

perfil.componente.js

```

import React, { Component } from "react";
import { Navigate } from "react-router-dom";
import AuthService from "../services/auth.service";

export default class Profile extends Component {
  constructor(props) {
    super(props);

    this.state = {
      redirect: null,
      userReady: false,
      currentUser: { username: "" }
    };
  }

  componentDidMount() {
    const currentUser = AuthService.getCurrentUser();

    if (!currentUser) this.setState({ redirect: "/home" });
    this.setState({ currentUser: currentUser, userReady: true })
  }

  render() {
    if (this.state.redirect) {
      return <Navigate to={this.state.redirect} />
    }
  }
}

```

```

const { currentUser } = this.state;

return (
  <div className="container">
    {(this.state.userReady) ?
      <div>
        <header className="jumbotron">
          <h3>
            <strong>{currentUser.username}</strong> Profile
          </h3>
        </header>
        <p>
          <strong>Token:</strong>{" "}
          {currentUser.accessToken.substring(0, 20)} ...{" "}
          {currentUser.accessToken.substr(currentUser.accessToken.length - 20)}
        </p>
        <p>
          <strong>Id:</strong>{" "}
          {currentUser.id}
        </p>
        <p>
          <strong>Email:</strong>{" "}
          {currentUser.email}
        </p>
        <strong>Authorities:</strong>
        <ul>
          {currentUser.roles &&
            currentUser.roles.map((role, index) => <li key={index}>{role}</li>)}
        </ul>
      </div>: null}
    </div>
  );
}
}

```

Crear componentes de React para acceder a los recursos

Estos componentes se utilizarán UserService para solicitar datos de la API.

inicio.componente.js

placa-usuario.component.js

junta-moderador.component.js

tablero-admin.component.js

Página de inicio

Esta es una página pública que muestra contenido público. Las personas no necesitan iniciar sesión para ver esta página.

inicio.componente.js

```
import React, { Component } from "react";

import UserService from "../services/user.service";

export default class Home extends Component {
  constructor(props) {
    super(props);

    this.state = {
      content: ""
    };
  }

  componentDidMount() {
    UserService.getPublicContent().then(
      response => {
        this.setState({
          content: response.data
        });
      },
      error => {
        this.setState({
          content:
            (error.response && error.response.data) ||
            error.message ||
            error.toString()
        });
      }
    );
  }

  render() {
    return (
      <div className="container">
        <header className="jumbotron">
          <h3>{this.state.content}</h3>
        </header>
      </div>
    );
  }
}
```

Páginas basadas en roles

Vamos a tener 3 páginas para acceder a datos protegidos:

- Llamadas a la página **BoardUser**UserService.getUserBoard()
- Llamadas de la página **BoardModerator**UserService.getModeratorBoard()
- Llamadas de la página **BoardAdmin**UserService.getAdminBoard()

Le mostraré la página de usuario, por ejemplo, otras páginas son similares a esta página.

placa-usuario.component.js

```
import React, { Component } from "react";

import UserService from "../services/user.service";

export default class BoardUser extends Component {
  constructor(props) {
    super(props);

    this.state = {
      content: ""
    };
  }

  componentDidMount() {
    UserService.getUserBoard().then(
      response => {
        this.setState({
          content: response.data
        });
      },
      error => {
        this.setState({
          content:
            (error.response &&
              error.response.data &&
              error.response.data.message) ||
            error.message ||
            error.toString()
        });
      }
    );
  }

  render() {
    return (
      <div className="container">
        <header className="jumbotron">
          <h3>{this.state.content}</h3>
        </header>
      </div>
    );
  }
}
```

Puede simplificar la declaración de importación con:
Importación absoluta en React

Agregar barra de navegación y definir rutas

Ahora agregamos una barra de navegación en Appel componente. Este es el contenedor raíz de nuestra aplicación.

La barra de navegación cambia dinámicamente según el estado de inicio de sesión y las funciones del usuario actual.

- **Casa** : siempre
- **Iniciar sesión y registrarse** : si el usuario aún no ha iniciado sesión
- **Usuario** : AuthService.getCurrentUser() devuelve un valor
- **Moderador de la Junta** : las funciones incluyenROLE_MODERATOR
- **Administrador de la junta** : los roles incluyenROLE_ADMIN

src/Aplicación.js

```
import React, { Component } from "react";
import { Routes, Route, Link } from "react-router-dom";
import "bootstrap/dist/css/bootstrap.min.css";
import "./App.css";

import AuthService from "../services/auth.service";

import Login from "../components/login.component";
import Register from "../components/register.component";
import Home from "../components/home.component";
import Profile from "../components/profile.component";
import BoardUser from "../components/board-user.component";
import BoardModerator from "../components/board-moderator.component";
import BoardAdmin from "../components/board-admin.component";

class App extends Component {
  constructor(props) {
    super(props);
    this.logout = this.logout.bind(this);

    this.state = {
      showModeratorBoard: false,
      showAdminBoard: false,
      currentUser: undefined,
    };
  }

  componentDidMount() {
    const user = AuthService.getCurrentUser();

    if (user) {
      this.setState({
        currentUser: user,
        showModeratorBoard: user.roles.includes("ROLE_MODERATOR"),
        showAdminBoard: user.roles.includes("ROLE_ADMIN"),
      });
    }
  }

  logout() {
    AuthService.logout();
    this.setState({
```

```

    showModeratorBoard: false,
    showAdminBoard: false,
    currentUser: undefined,
  });
}

```

```

render() {
  const { currentUser, showModeratorBoard, showAdminBoard } = this.state;

  return (
    <div>
      <nav className="navbar navbar-expand navbar-dark bg-dark">
        <Link to="/" className="navbar-brand">
          bezKoder
        </Link>
        <div className="navbar-nav mr-auto">
          <li className="nav-item">
            <Link to="/home" className="nav-link">
              Home
            </Link>
          </li>

          {showModeratorBoard && (
            <li className="nav-item">
              <Link to="/mod" className="nav-link">
                Moderator Board
              </Link>
            </li>
          )}

          {showAdminBoard && (
            <li className="nav-item">
              <Link to="/admin" className="nav-link">
                Admin Board
              </Link>
            </li>
          )}

          {currentUser && (
            <li className="nav-item">
              <Link to="/user" className="nav-link">
                User
              </Link>
            </li>
          )}
        </div>

        {currentUser ? (
          <div className="navbar-nav ml-auto">
            <li className="nav-item">
              <Link to="/profile" className="nav-link">
                {currentUser.username}
              </Link>
            </li>
            <li className="nav-item">
              <a href="/login" className="nav-link" onClick={this.logOut}>
                LogOut
              </a>
            </li>
          </div>
        ) : (
          <div className="navbar-nav ml-auto">

```

```

    <li className="nav-item">
      <Link to={"/login"} className="nav-link">
        Login
      </Link>
    </li>

    <li className="nav-item">
      <Link to={"/register"} className="nav-link">
        Sign Up
      </Link>
    </li>
  </div>
)}
</nav>

<div className="container mt-3">
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/home" element={<Home />} />
    <Route path="/login" element={<Login />} />
    <Route path="/register" element={<Register />} />
    <Route path="/profile" element={<Profile />} />
    <Route path="/user" element={<BoardUser />} />
    <Route path="/mod" element={<BoardModerator />} />
    <Route path="/admin" element={<BoardAdmin />} />
  </Routes>
</div>
</div>
);
}
}

export default App;

```

Cerrar sesión cuando el token haya caducado

Hay dos maneras. Para obtener más detalles, visite:

[Reaccionar: cómo cerrar la sesión cuando el token JWT está vencido](#)

Agregar estilo CSS para componentes React

Abra **src / App.css** y escriba un código CSS de la siguiente manera:

```

label {
  display: block;
  margin-top: 10px;
}

.card-container.card {
  max-width: 350px !important;
  padding: 40px 40px;
}

.card {
  background-color: #f7f7f7;
  padding: 20px 25px 30px;
  margin: 0 auto 25px;
  margin-top: 50px;
  -moz-border-radius: 2px;
  -webkit-border-radius: 2px;
  border-radius: 2px;
  -moz-box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);

```

```
-webkit-box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);
box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);
}

.profile-img-card {
width: 96px;
height: 96px;
margin: 0 auto 10px;
display: block;
-moz-border-radius: 50%;
-webkit-border-radius: 50%;
border-radius: 50%;
}
```

Configure el puerto para el cliente de autenticación React JWT con API web

Debido a que la mayoría del servidor HTTP usa la configuración CORS que acepta el uso compartido de recursos restringido a algunos sitios o puertos, también debemos configurar el puerto para nuestra aplicación.

En la carpeta del proyecto, cree un archivo `.env` con el siguiente contenido:

`PORT=8081`

- **Back-end** : <https://www.bezkoder.com/node-js-jwt-authentication-mysql/>
- **Front-end** : <https://www.bezkoder.com/react-jwt-auth/>



tutoriales y mas : <https://www.bezkoder.com/>

Fullstack (ejemplo de autenticación y autorización de JWT):

- [React + Spring Boot](https://www.bezkoder.com/spring-boot-react-jwt-auth/): <https://www.bezkoder.com/spring-boot-react-jwt-auth/>
- [Node.js + MongoDB: autenticación y autorización de usuarios con JWT](https://www.bezkoder.com/node-js-mongodb-auth-jwt/)
<https://www.bezkoder.com/node-js-mongodb-auth-jwt/>
- [Node.js + PostgreSQL: autenticación y autorización de usuarios con JWT](https://www.bezkoder.com/node-js-jwt-authentication-postgresql/)
<https://www.bezkoder.com/node-js-jwt-authentication-postgresql/>
- [React Typescript: ejemplo de autenticación JWT \(sin Redux\)](https://www.bezkoder.com/react-typescript-login-example/)
<https://www.bezkoder.com/react-typescript-login-example/>
- [React Redux: ejemplo de autenticación y autorización JWT](https://www.bezkoder.com/react-redux-jwt-auth/)
<https://www.bezkoder.com/react-redux-jwt-auth/>

Uso de Hooks:

- [React Hooks: ejemplo de autenticación JWT \(sin Redux\)](https://www.bezkoder.com/react-hooks-jwt-auth/)
<https://www.bezkoder.com/react-hooks-jwt-auth/>
- [React Hooks + Redux: ejemplo de autenticación y autorización JWT](https://www.bezkoder.com/react-hooks-redux-login-registration-example/)
<https://www.bezkoder.com/react-hooks-redux-login-registration-example/>

Fullstack CRUD:

Java -React

- [React + Spring Boot + MySQL](https://www.bezkoder.com/react-spring-boot-crud/): <https://www.bezkoder.com/react-spring-boot-crud/>
- [Ejemplo de Spring Boot + React Typescript](https://www.bezkoder.com/spring-boot-react-typescript/): <https://www.bezkoder.com/spring-boot-react-typescript/>
- [React + Spring Boot + PostgreSQL](https://www.bezkoder.com/spring-boot-react-postgresql/): <https://www.bezkoder.com/spring-boot-react-postgresql/>
- [React + Spring Boot + MongoDB](https://www.bezkoder.com/react-spring-boot-mongodb/): <https://www.bezkoder.com/react-spring-boot-mongodb/>

Node-React

React Redux Login, Logout, Registration example with Hooks:

<https://www.bezkoder.com/react-hooks-redux-login-registration-example/>

- [React + Node.js Express + MySQL](https://www.bezkoder.com/react-node-express-mysql/): <https://www.bezkoder.com/react-node-express-mysql/>
- [React + Node.js Express + PostgreSQL](https://www.bezkoder.com/react-node-express-postgresql/): <https://www.bezkoder.com/react-node-express-postgresql/>

– [React.js + Node.js Express + MongoDB](https://www.bezkoder.com/react-node-express-mongodb-mern-stack/): <https://www.bezkoder.com/react-node-express-mongodb-mern-stack/>

– [Reaccionar + Django](https://www.bezkoder.com/django-react-axios-rest-framework/): <https://www.bezkoder.com/django-react-axios-rest-framework/>

React Typescript con Axios y Web API: <https://www.bezkoder.com/react-typescript-axios/>

React Material UI examples with a CRUD Application

<https://www.bezkoder.com/react-material-ui-examples-crud/>

Otra forma de implementar la validación de formulario:

[ejemplo de validación de formulario React con Formik y Yup](#)

<https://www.bezkoder.com/react-form-validation-example-formik/>

Dockerize:

– [Docker Compose: React, Node.js, ejemplo de MySQL](#):

<https://www.bezkoder.com/docker-compose-react-nodejs-mysql/>

Firebase

Sin servidor:

– React Firebase CRUD con Realtime Firebase: <https://www.bezkoder.com/react-firebase-crud/>

– [Ejemplo de la aplicación React Firestore CRUD | Firebase en la nube Firestore](#)

<https://www.bezkoder.com/react-firestore-crud/>

– Typescript version: [React Typescript Firestore CRUD example | Firebase Firestore](#)

<https://www.bezkoder.com/react-typescript-firestore/>

Java-Angular

Angular 14 + Spring Boot + MySQL: CRUD example:

<https://www.bezkoder.com/spring-boot-angular-14-mysql/>

Angular 16 + Spring Boot: JWT Authentication & Authorization example

<https://www.bezkoder.com/angular-16-spring-boot-jwt-auth/>

Java backend

• Spring JPA + PostgreSQL: <https://www.bezkoder.com/spring-boot-postgresql-example/>