Protección de aplicación web MEVN Stack mediante Passport

Tutorial completo paso a paso sobre cómo proteger MEVN (MongoDB, Express.js, Vue.js 2, Node.js) Stack Web Application usando Passport.js.

Tabla de contenido:

Prerrequisitos:

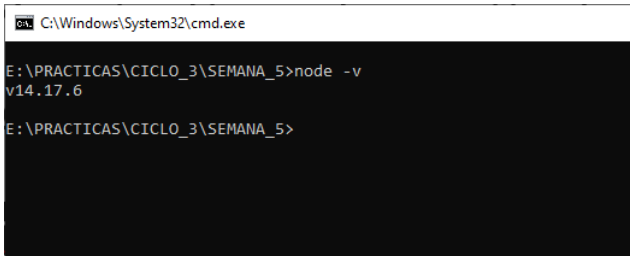
Se requieren las siguientes herramientas, frameworks y módulos para este tutorial:

1. Node.js
2. Express.js
3. MongoDB
4. Mongoose.js
5. Vue.js
6. Vue-CLI
7. Passport.js
8. Terminal o línea de comando de Node.js
9. IDE o editor de texto

Suponemos que ya ha instalado Node.js y MongoDB. Asegúrese de que la línea de comandos de Node.js funcione (en Windows) o se pueda ejecutar en el terminal Linux / OS X. Puede ejecutar MongoDB en la terminal o línea de comando diferente. Escriba este comando en la terminal o en la línea de comandos de Node para verificar la versión de Node.js.

```
node -v
```

Y aquí está nuestra versión de Node.js.



Ahora, instale Vue-CLI escribiendo este comando.

```
npm install -g @vue/cli
```

Luego

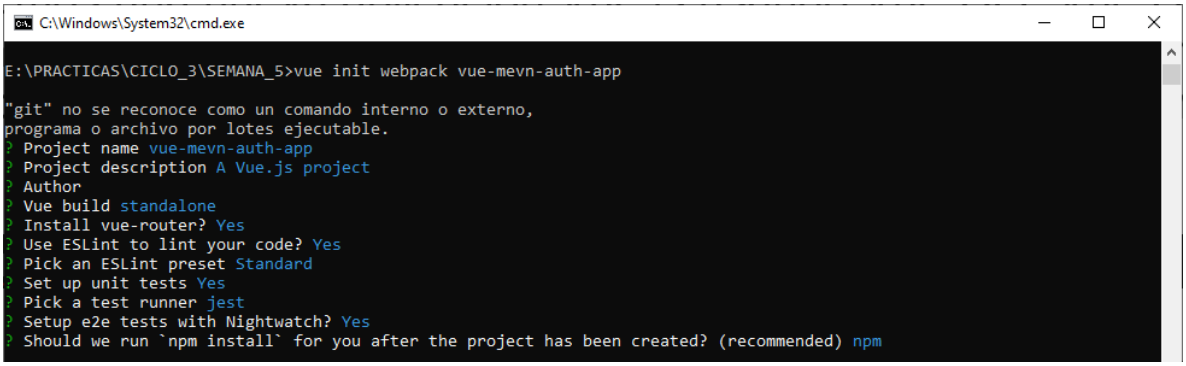
```
npm i -g @vue/cli-init
```

Cree una nueva aplicación Vue.js

Crearemos una nueva aplicación Vue.js usando Vue-CLI. Vue CLI es un sistema completo para el desarrollo rápido de Vue.js.
Para crear una nueva aplicación Vue.js usando “vue-cli”, simplemente escriba este comando desde la terminal o la línea de comandos.

```
vue init webpack vue-mevn-auth-app
```

De respuesta con “y” o <ENTER> a las preguntas hechas en la consola. A saber:



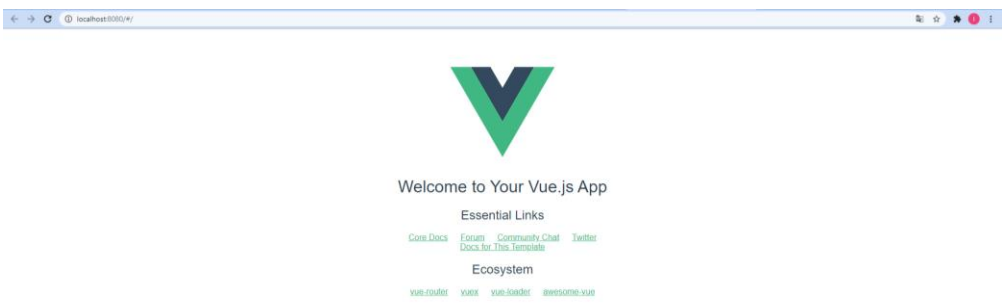
Como es habitual, tenemos que asegurarnos de que la aplicación Vue.js se esté ejecutando correctamente. Vaya al nuevo directorio de la aplicación Vue.js y luego ejecute la aplicación. A saber:

```
cd vue-mevn-auth-app
```

luego, ejecute el siguiente comando:

```
npm run dev
```

Ahora, abra el navegador y vaya a esta dirección http://localhost:8080



Instale Express.js como servidor API REST

Instalamos o agregamos Express.js a la aplicación Vue existente. Primero cierre la aplicación Vue.js en ejecución presionando "`ctrl + c`" y luego escriba este comando para agregar el módulo "Express.js" y sus dependencias (body-parser, morgan, serve-favicon).

```
npm install express body-parser morgan serve-favicon --save
```

A continuación, cree una nueva carpeta llamada "`bin`" en la raíz de la carpeta del proyecto Vue.js, luego, cree un archivo llamado "`www`".

Puede utilizar su propio nombre de archivo y carpeta para el punto de partida de la aplicación Express.js.

Abra y edite el archivo "`www`" y luego agregue estas líneas de códigos.

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require("../app");
var debug = require("debug")("mean-app:server");
var http = require("http");

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || "3000");
app.set("port", port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on("error", onError);
```

```
server.on("listening", onListening);

/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== "listen") {
    throw error;
  }

  var bind = typeof port === "string" ? "Pipe " + port : "Port " + port;

  // handle specific listen errors with friendly messages
  switch (error.code) {
    case "EACCES":
      console.error(bind + " requires elevated privileges");
      process.exit(1);
      break;
    case "EADDRINUSE":
      console.error(bind + " is already in use");
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
```

```
  var addr = server.address();
  var bind = typeof addr === "string" ? "pipe " + addr : "port " + addr.port;
  debug("Listening on " + bind);
}
```

A continuación, cambie el servidor predeterminado que se ejecuta con el comando “npm”. Abra y edite “package.json” y luego reemplace el valor de ”start” dentro de “scripts”.

```
"scripts": {
    "dev": "webpack-dev-server --inline --progress --
config build/webpack.dev.conf.js",
    "start": "npm run build && node ./bin/www",
    "unit": "cross-env BABEL_ENV=test karma start test/unit/karma.conf.js --
single-run",
    "e2e": "node test/e2e/runner.js",
    "test": "npm run unit && npm run e2e",
    "lint": "eslint --ext .js,.vue src test/unit/specs test/e2e/specs",
    "build": "node build/build.js"
},
```

Ahora, cree “app.js” en la raíz de la carpeta del proyecto.

Abra y edite “app.js” y luego agregue estas líneas de códigos.

```
var express = require("express");
var path = require("path");
var favicon = require("serve-favicon");
var logger = require("morgan");
var bodyParser = require("body-parser");

var book = require("./routes/book");
var app = express();

app.use(logger("dev"));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: "false" }));
app.use(express.static(path.join(__dirname, "dist")));
app.use("/books", express.static(path.join(__dirname, "dist")));
app.use("/book", book);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error("Not Found");
  err.status = 404;
  next(err);
});

// restful api error handler
```

```
app.use(function(err, req, res, next) {
  console.log(err);

  if (req.app.get("env") !== "development") {
    delete err.stack;
  }

  res.status(err.statusCode || 500).json(err);
});

module.exports = app;
```

A continuación, cree una carpeta de rutas en la raíz del proyecto, a saber:

```
mkdir routes
```

Seguidamente, cree, abra y edite el archivo "`routes/book.js`" y luego agregue estas líneas de códigos para manejar la solicitud de API REST.

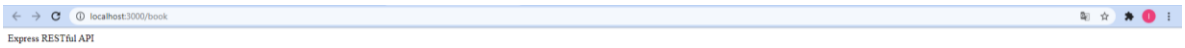
```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.send('Express RESTful API');
});

module.exports = router;
```

Ahora, ejecute el servidor con este comando.

```
npm start
```

Entonces, verá la página de inicio de Vue.js anterior cuando apunte su navegador a la dirección: http://localhost:3000

Cuando cambie la dirección a http://localhost:3000/book, vera la siguiente página:



Instalar y configurar Mongoose.js

Necesitamos acceder a los datos de MongoDB. Para eso, instalaremos y configuraremos Mongoose.js.

Mongoose proporciona una solución sencilla basada en esquemas para modelar los datos de su aplicación. Incluye conversión de tipos incorporada, validación, creación de consultas, `hooks` de lógica empresarial y más, listos para usar.

En la terminal, escriba este comando para instalar mongoose y bluebird después de detener el servidor Express en ejecución.

```
npm install mongoose bluebird --save
```

Abra y edite "`app.js`" y agregue estas líneas después de la última línea de variables.

```
var mongoose = require("mongoose");

mongoose.Promise = require("bluebird");
mongoose
  .connect(
    "mongodb+srv://lmolero:1126254560@cluster0.bpms5.mongodb.net/vuemevnauthapp?
retryWrites=true&w=majority",
    {
      promiseLibrary: require("bluebird")
    }
  )
  .then(() => console.log("connection succesful"))
  .catch(err => console.error(err));
```

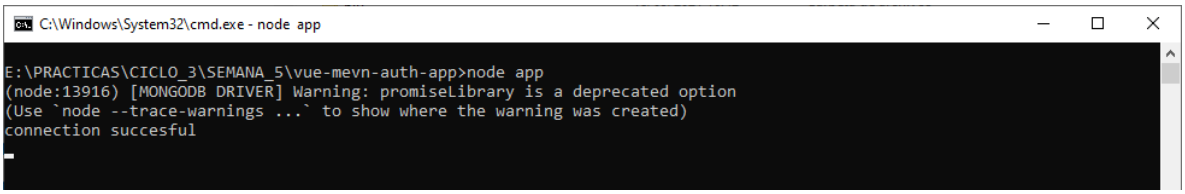
Ahora, ejecute el servidor MongoDB en una pestaña de terminal diferente o línea de comando o ejecute desde el servicio.

```
mongod
```

A continuación, puede probar la conexión a MongoDB. Ejecute el siguiente comando:

```
node app
```

Y vera:



Si todavía está utilizando la biblioteca Mongoose Promise incorporada, recibirá esta advertencia obsoleta en el terminal.

```
(node:13916) [MONGODB DRIVER] Warning: promiseLibrary is a deprecated option
(Use `node --trace-warnings ...` to show where the warning was created)
```

Esa es la razón por la que agregamos módulos `bluebird` y los registramos como biblioteca Mongoose Promise.

Crear modelo Mongoose.js

Agregue la carpeta de modelos en la raíz de la carpeta del proyecto para guardar los archivos del modelo Mongoose.js.

```
mkdir models
```

Ahora, dentro de la carpeta “`models`” cree el archivo “`Book.js`” para el modelo de Mongoose.js. Crearemos un modelo de colección de libros.

Ahora, abra y edite ese archivo y agregue el siguiente código:

```
var mongoose = require("mongoose");

var BookSchema = new mongoose.Schema({
```

```
  isbn: String,
  title: String,
  author: String,
  description: String,
  published_year: String,
  publisher: String,
  updated_date: { type: Date, default: Date.now }
});

module.exports = mongoose.model("Book", BookSchema);
```

Ese esquema se asignará a las colecciones de MongoDB llamadas “Book”.

Cree rutas exprés para la API REST

Abra nuevamente el archivo “routes/book.js” y reemplace todos los códigos con este enrutador Express tiene agregando los métodos GET, POST, PUT y DELETE que acceden a los datos de MongoDB.

```
var express = require("express");
var router = express.Router();
var mongoose = require("mongoose");
var Book = require("../models/Book.js");

/* GET ALL BOOKS */
router.get("/", function(req, res, next) {
  Book.find(function(err, products) {
    if (err) return next(err);
    res.json(products);
  });
});

/* GET SINGLE BOOK BY ID */
router.get("/:id", function(req, res, next) {
  Book.findById(req.params.id, function(err, post) {
    if (err) return next(err);
    res.json(post);
  });
});

/* SAVE BOOK */
router.post("/", function(req, res, next) {
  Book.create(req.body, function(err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
```

```
/* UPDATE BOOK */
router.put("/:id", function(req, res, next) {
  Book.findByIdAndUpdate(req.params.id, req.body, function(err, post) {
    if (err) return next(err);
    res.json(post);
  });
});

/* DELETE BOOK */
router.delete("/:id", function(req, res, next) {
  Book.findByIdAndRemove(req.params.id, req.body, function(err, post) {
    if (err) return next(err);
    res.json(post);
  });
});

module.exports = router;
```

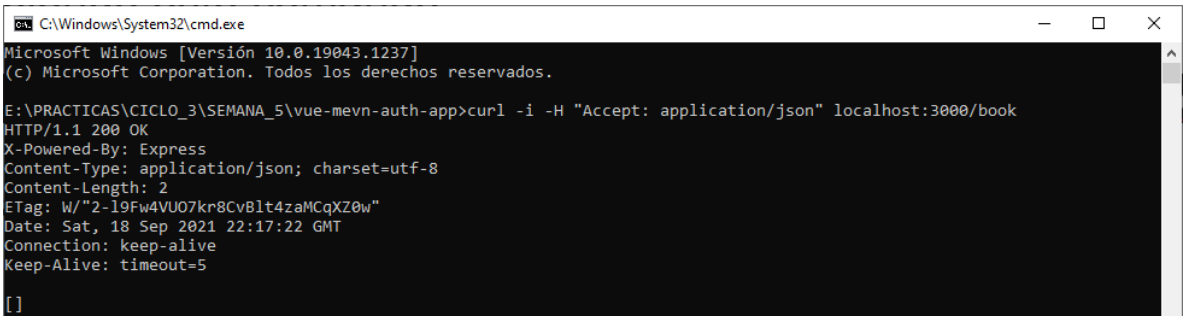
Ejecute nuevamente el servidor Express, de la forma:

```
npm start
```

Luego, abra otra terminal o línea de comando para probar la API Restful escribiendo este comando:

```
curl -i -H "Accept: application/json" localhost:3000/book
```

Si ese comando devuelve una respuesta como la que se muestra a continuación, la API REST está lista para funcionar.



Ahora, probemos la colección de libros con algunos datos iniciales enviando una solicitud POST desde la API REST, para lo cual, abramos Postman.

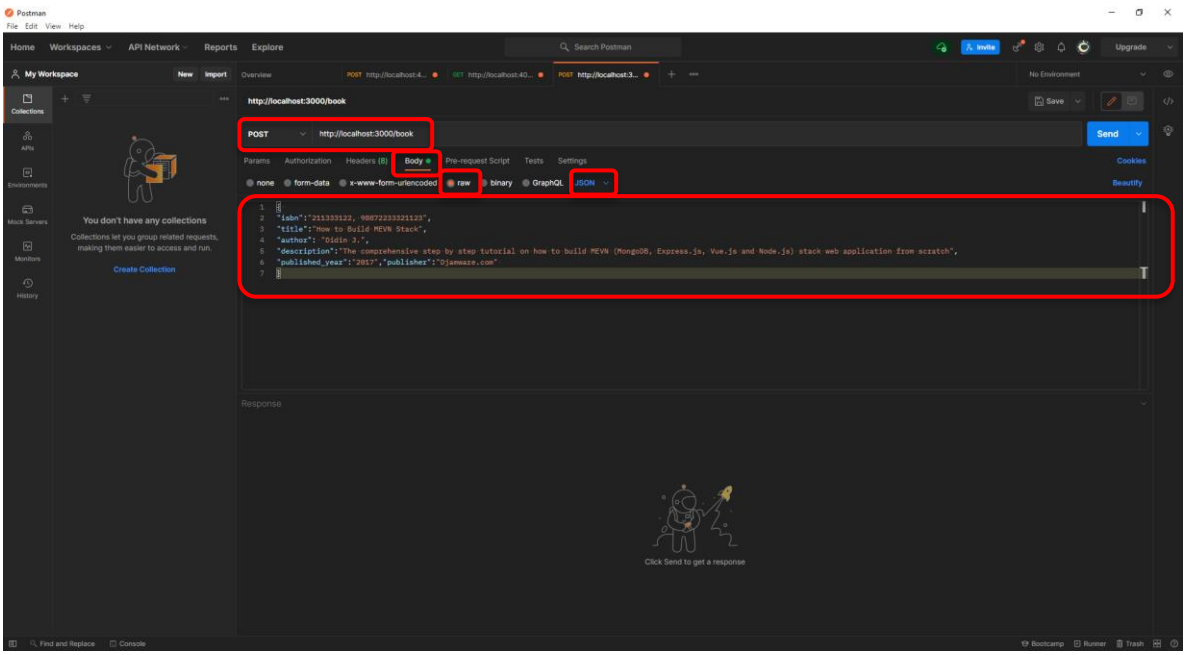
Creamos un “test” y generamos una solicitud HTTP POST, de la forma:

Solicitud: POST.
Dirección: `http://localhost:3000/book`

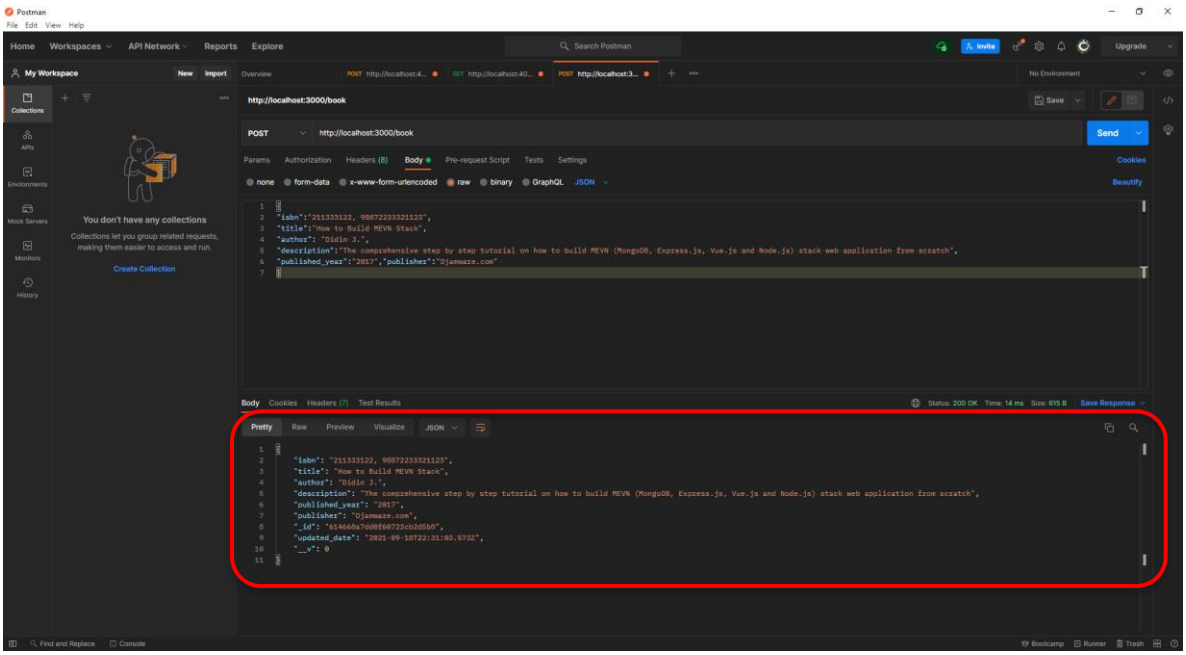
Ahora, creamos el siguiente registro para enviar la solicitud

```
{
"isbn":"211333122, 98872233321123",
"title":"How to Build MEVN Stack",
"author": "Didin J.",
"description":"The comprehensive step by step tutorial on how to build MEVN (Mon
goDB, Express.js, Vue.js and Node.js) stack web application from scratch",
"published_year":"2017","publisher":"Djamware.com"
}
```

Tal como se aprecia en la imagen, registramos los datos de la siguiente forma:



En consecuencia, verá la siguiente respuesta en la consola de Postman:



Instalar y configurar Passport.js

Es hora de asegurar la API REST usando Passport.js. Passport es un middleware de autenticación para Node.js. Extremadamente flexible y modular, que se puede colocar discretamente en cualquier aplicación web basada en Express.

Primero, instale el módulo Passport.js y sus dependencias (bcrypt-nodejs, jsonwebtoken, morgan, passport-jwt).

```
npm install bcrypt-nodejs jsonwebtoken morgan passport passport-jwt --save
```

Ahora, cree los siguientes archivos de configuración dentro de la carpeta "`config`"

`settings.js`
`passport.js`

Seguidamente, abra “config/settings.js” y agregue estas líneas de códigos.

```
module.exports = {
  secret: "mevnsecure"
};
```

Ese archivo contiene un código secreto para generar un token JWT.

A continuación, abra “config/passport.js” y agregue estas líneas de códigos:

```
var JwtStrategy = require("passport-jwt").Strategy,
  ExtractJwt = require("passport-jwt").ExtractJwt;

// load up the user model
var User = require("../models/user");
var settings = require("../config/settings"); // get settings file

module.exports = function(passport) {
  var opts = {};
  opts.jwtFromRequest = ExtractJwt.fromAuthHeaderWithScheme("jwt");
  opts.secretOrKey = settings.secret;
  passport.use(
    new JwtStrategy(opts, function(jwt_payload, done) {
      User.findOne({ id: jwt_payload.id }, function(err, user) {
        if (err) {
          return done(err, false);
        }
        if (user) {
          done(null, user);
        } else {
          done(null, false);
        }
      });
    })
  );
};
```

Esta configuración se utiliza para obtener al usuario haciendo coincidir el token JWT con el token obtenido del cliente. Esta configuración necesita crear un modelo de usuario. Para eso, cree el archivo “User.js” dentro de la carpeta de “models”.

A continuación, abra “`models/User.js`” y agregue estas líneas de códigos.

```
var mongoose = require("mongoose");
var Schema = mongoose.Schema;
var bcrypt = require("bcrypt-nodejs");

var UserSchema = new Schema({
  username: {
    type: String,
    unique: true,
    required: true
  },
  password: {
    type: String,
    required: true
  }
});

UserSchema.pre("save", function(next) {
  var user = this;
  if (this.isModified("password") || this.isNew) {
    bcrypt.genSalt(10, function(err, salt) {
      if (err) {
        return next(err);
      }
      bcrypt.hash(user.password, salt, null, function(err, hash) {
        if (err) {
          return next(err);
        }
        user.password = hash;
        next();
      });
    });
  } else {
    return next();
  }
});

UserSchema.methods.comparePassword = function(passw, cb) {
  bcrypt.compare(passw, this.password, function(err, isMatch) {
    if (err) {
      return cb(err);
    }
    cb(null, isMatch);
  });
};

module.exports = mongoose.model("User", UserSchema);
```

Los diferentes modelos de usuario son una función adicional para crear una contraseña encriptada usando “Bcrypt” y una función para comparar contraseñas encriptadas.

Cree enrutadores express para el inicio de sesión de la API REST

Ahora, crearemos un enrutador para autenticar al usuario y restringir los recursos. En la carpeta “routes” cree un archivo llamado “auth.js”

Seguidamente, abra “routes/auth.js” y escriba el siguiente código:

```
var mongoose = require("mongoose");
var passport = require("passport");
var settings = require("../config/settings");
require("../config/passport")(passport);
var express = require("express");
var jwt = require("jsonwebtoken");
var router = express.Router();
var User = require("../models/user");

router.post("/register", function(req, res) {
  if (!req.body.username || !req.body.password) {
    res.json({ success: false, msg: "Please pass username and password." });
  } else {
    var newUser = new User({
      username: req.body.username,
      password: req.body.password
    });
    // save the user
    newUser.save(function(err) {
      if (err) {
        return res.json({ success: false, msg: "Username already exists." });
      }
      res.json({ success: true, msg: "Successful created new user." });
    });
  }
});

router.post("/login", function(req, res) {
  User.findOne(
    {
      username: req.body.username
    },
    function(err, user) {
      if (err) throw err;

      if (!user) {
```

```
        res
          .status(401)
          .send({
            success: false,
            msg: "Authentication failed. User not found."
          });
      } else {
        // check if password matches
        user.comparePassword(req.body.password, function(err, isMatch) {
          if (isMatch && !err) {
            // if user is found and password is right create a token
            var token = jwt.sign(user.toJSON(), settings.secret);
            // return the information including token as JSON
            res.json({ success: true, token: "JWT " + token });
          } else {
            res
              .status(401)
              .send({
                success: false,
                msg: "Authentication failed. Wrong password."
              });
          }
        });
      }
    }
  );
});

module.exports = router;
```

Ahora, abra "`app.js`" nuevamente para asignar el enrutador de autenticación a la URL.

Agregue esta variable requerida:

```
var auth = require('./routes/auth');
```

Y esta sentencia "`.use`" después de "`.use(book)`".

```
app.use('/api/auth', auth);
```

Protección del enrutador de la API REST del libro

Es hora de proteger los recursos de la API REST. Agregaremos una restricción para la API REST de Book. Para eso, abra “`routes/book.js`” agregando la variable “`passport`", el nuevo enrutador GET y POST y la función para obtener y extraer el token JWT. El código quedarà de la siguiente forma:

```
var express = require("express");
var router = express.Router();
var mongoose = require("mongoose");
var Book = require("../models/Book.js");
var passport = require("passport");
require("../config/passport")(passport);

/* GET ALL BOOKS */
router.get("/", passport.authenticate("jwt", { session: false }), function(
  req,
  res
) {
  var token = getToken(req.headers);
  if (token) {
    Book.find(function(err, books) {
      if (err) return next(err);
      res.json(books);
    });
  } else {
    return res.status(403).send({ success: false, msg: "Unauthorized." });
  }
});

/* GET SINGLE BOOK BY ID */
router.get("/:id", function(req, res, next) {
  Book.findById(req.params.id, function(err, post) {
    if (err) return next(err);
    res.json(post);
  });
});

/* SAVE BOOK */
router.post("/", passport.authenticate("jwt", { session: false }), function(
  req,
  res
) {
  var token = getToken(req.headers);
  if (token) {
    Book.create(req.body, function(err, post) {
      if (err) return next(err);
      res.json(post);
```

```
    });
  } else {
    return res.status(403).send({ success: false, msg: "Unauthorized." });
  }
});

/* UPDATE BOOK */
router.put("/:id", function(req, res, next) {
  Book.findByIdAndUpdate(req.params.id, req.body, function(err, post) {
    if (err) return next(err);
    res.json(post);
  });
});

/* DELETE BOOK */
router.delete("/:id", function(req, res, next) {
  Book.findByIdAndRemove(req.params.id, req.body, function(err, post) {
    if (err) return next(err);
    res.json(post);
  });
});

getToken = function(headers) {
  if (headers && headers.authorization) {
    var parted = headers.authorization.split(" ");
    if (parted.length === 2) {
      return parted[1];
    } else {
      return null;
    }
  } else {
    return null;
  }
};

module.exports = router;
```

Ahora, ejecute nuevamente el servidor Express

```
npm start
```

Desarrollemos la misma prueba de API REST. A saber:

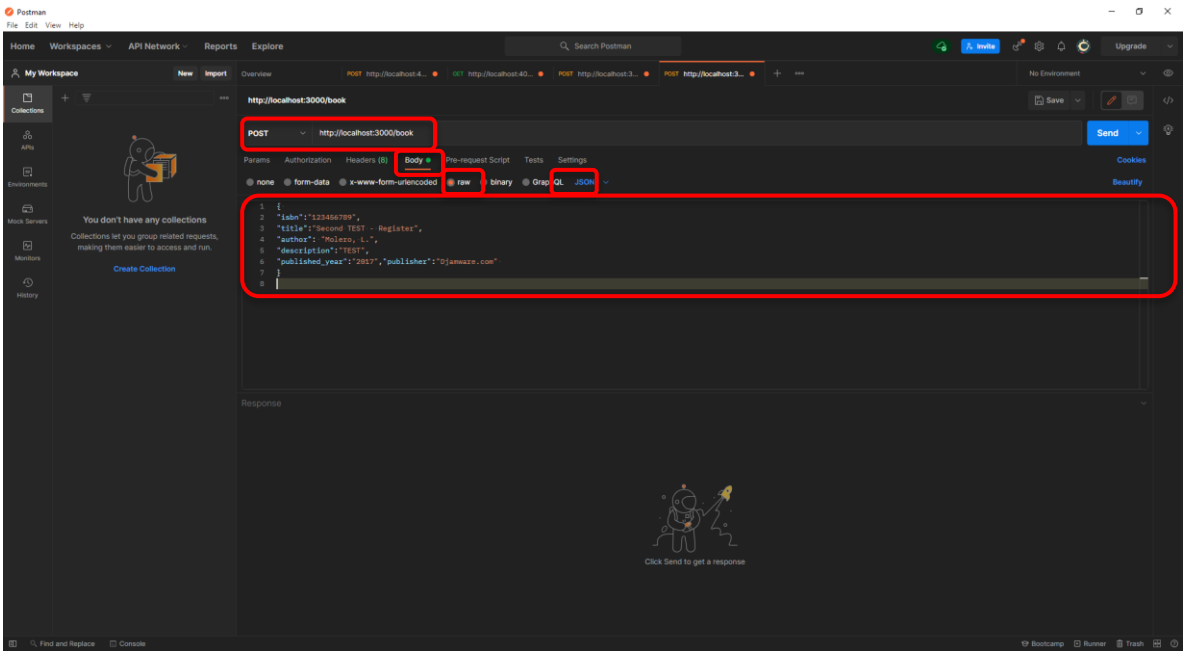
Creamos de nuevo un “test” y generamos una solicitud HTTP POST, de la forma:

Solicitud: POST.
Dirección: `http://localhost:3000/book`

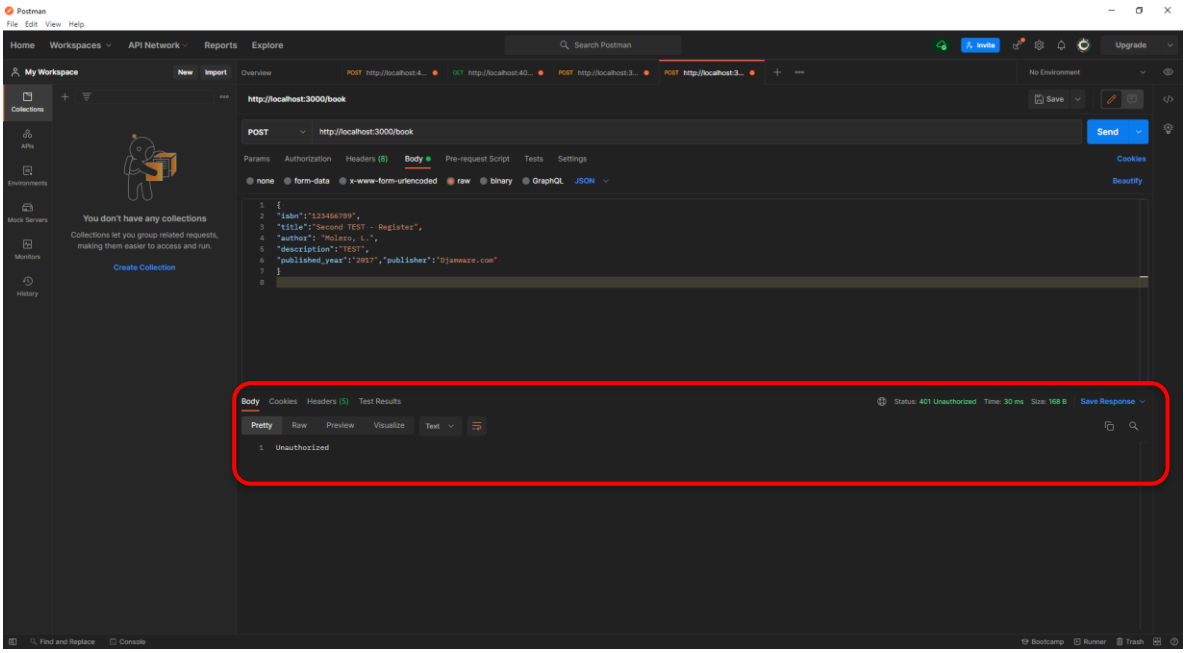
Ahora, creamos el siguiente registro para enviar la solicitud

```
{
"isbn":"123456789",
"title":"Second TEST - Register",
"author": "Molero, L.",
"description":"TEST",
"published_year":"2017","publisher":"Djamware.com"
}
```

Tal como se aprecia en la imagen, registramos los datos de la siguiente forma:



En consecuencia, verá la siguiente respuesta en la consola de Postman:



Eso significa que la API RESTful de Book es segura y accesible para un usuario autorizado.

Instalar y configurar los módulos necesarios para Vue.js

Necesitamos módulos adicionales para acceder a la API REST y diseñar la interfaz. Para navegar o enrutar entre vistas, ya está instalado el módulo "`vue-router`".

Escriba estos comandos en la raíz del proyecto para instalar los módulos necesarios de Bootstrap-Vue, Bootstrap para UI / UX y Axios para consumir la API REST.

```
npm i bootstrap-vue bootstrap@4.0.0-beta.2
```

Ahora, instalar axios

```
npm i axios --save
```

Abra “src/main.js” y agregue las importaciones para Bootstrap-Vue.

```
import Vue from "vue";
import BootstrapVue from "bootstrap-vue";
import App from "./App";
import router from "./router";
import "bootstrap/dist/css/bootstrap.css";
import "bootstrap-vue/dist/bootstrap-vue.css";
```

Agregue esta línea después de `Vue.config`.

```
Vue.use(BootstrapVue);
```

Para registrar o crear rutas para toda la navegación de la aplicación, abra “src/router/index.js” y luego reemplace todos los códigos con esto.

```
import Vue from "vue";
import Router from "vue-router";
import BookList from "@/components/BookList";
import Login from "@/components/Login";
import Register from "@/components/Register";

Vue.use(Router);

export default new Router({
  routes: [
    {
      path: "/",
      name: "BookList",
      component: BookList
    },
    {
      path: "/login",
      name: "Login",
      component: Login
    },
    {
      path: "/register",
      name: "Register",
      component: Register
    }
  ]
});
```

Ahora, vamos a crear los siguientes componentes de la aplicación Vue, a saber:

src/components/BookList.vue
src/components/Login.vue
src/components/Register.vue

Modificar el componente BookList.vue

Ahora, abra "`src/components/BookList.vue`" y agregue estas líneas de códigos.

```
<template>
  <b-row>
    <b-col cols="12">
      <h2>
        Book List
        <b-link href="#/add-book">(Add Book)</b-link>
      </h2>
      <b-table striped hover :items="books" :fields="fields">
        <template slot="actions" scope="row">
          <b-btn size="sm" @click.stop="details(row.item)">Details</b-btn>
        </template>
      </b-table>
      <ul v-if="errors && errors.length">
        <li v-for="error of errors">
          {{ error.message }}
        </li>
      </ul>
    </b-col>
  </b-row>
</template>

<script>
import axios from "axios";

export default {
  name: "BookList",
  data() {
    return {
      fields: {
        isbn: { label: "ISBN", sortable: true, class: "text-center" },
        title: { label: "Book Title", sortable: true },
        actions: { label: "Action", class: "text-center" }
      },
      books: [],
      errors: []
    };
  },
  created() {
    axios.defaults.headers.common["Authorization"] = localStorage.getItem(
      "jwtToken"
    );
    axios
      .get(`http://localhost:3000/book`)
      .then(response => {
        this.books = response.data;
```

```
      })
      .catch(e => {
        this.errors.push(e);
        if (e.response.status === 401) {
          this.$router.push({
            name: "Login"
          });
        }
      });
  },
  methods: {
    details(book) {
      this.$router.push({
        name: "ShowBook",
        params: { id: book._id }
      });
    }
  }
};
</script>
```

Como puede ver, hay una llamada a la API REST de Book cuando se carga la página. Si la API llama al estado de error de la respuesta “401”, será redirigido a la página de inicio de sesión. También colocamos un botón de cierre de sesión dentro de la condición donde el token JWT no existe en el almacenamiento local.

Modificar el componente Login.vue

Anteriormente, hemos creado un archivo Vue para el componente de inicio de sesión. Ahora, abra “`src/components/Login.vue`` y agregue el siguiente código

```
<template>
  <b-row class="justify-content-md-center">
    <b-col cols="6">
      <div v-if="errors && errors.length">
        <div v-for="error of errors">
          <b-alert show>{{ error.message }}</b-alert>
        </div>
      </div>
      <b-form @submit="onSubmit">
        <b-form-group
          id="fieldsetHorizontal"
          horizontal
          :label-cols="4"
          breakpoint="md"
          label="Enter Username"
```

```
        >
          <b-form-input
            id="username"
            :state="state"
            v-model.trim="login.username"
          ></b-form-input>
        </b-form-group>
        <b-form-group
          id="fieldsetHorizontal"
          horizontal
          :label-cols="4"
          breakpoint="md"
          label="Enter Password"
        >
          <b-form-input
            type="password"
            id="password"
            :state="state"
            v-model.trim="login.password"
          ></b-form-input>
        </b-form-group>
        <b-button type="submit" variant="primary">Login</b-button>
        <b-button type="button" variant="success" @click.stop="register()"
          >Register</b-button
        >
      </b-form>
    </b-col>
  </b-row>
</template>

<script>
import axios from "axios";

export default {
  name: "Login",
  data() {
    return {
      login: {},
      errors: []
    };
  },
  methods: {
    onSubmit(evt) {
      evt.preventDefault();
      axios
        .post(`http://localhost:3000/api/auth/login/`, this.login)
        .then(response => {
          localStorage.setItem("jwtToken", response.data.token);
          this.$router.push({
            name: "BookList"
          });
        })
        .catch(e => {
```

```
          console.log(e);
          this.errors.push(e);
        });
    },
    register() {
      this.$router.push({
        name: "Register"
      });
    }
  }
};
</script>
```

Modificar el componente Register.vue

Anteriormente, hemos creado un archivo para el componente de registro. Ahora, abra "`src/components/Register.vue`" y agregue el siguiente código.

```
<template>
  <b-row class="justify-content-md-center">
    <b-col cols="6">
      <h2>Please Register</h2>
      <div v-if="errors && errors.length">
        <div v-for="error of errors">
          <b-alert show>{{ error.message }}</b-alert>
        </div>
      </div>
      <b-form @submit="onSubmit">
        <b-form-group
          id="fieldsetHorizontal"
          horizontal
          :label-cols="4"
          breakpoint="md"
          label="Enter Username"
        >
          <b-form-input
            id="username"
            :state="state"
            v-model.trim="register.username"
          ></b-form-input>
        </b-form-group>
        <b-form-group
          id="fieldsetHorizontal"
          horizontal
          :label-cols="4"
          breakpoint="md"
          label="Enter Password"
```

```
        >
          <b-form-input
            type="password"
            id="password"
            :state="state"
            v-model.trim="register.password"
          ></b-form-input>
        </b-form-group>
        <b-button type="submit" variant="primary">Register</b-button>
        <b-button type="button" variant="success" @click="$router.go(-1)"
          >Cancel</b-button
        >
      </b-form>
    </b-col>
  </b-row>
</template>

<script>
import axios from "axios";

export default {
  name: "Register",
  data() {
    return {
      register: {},
      errors: []
    };
  },
  methods: {
    onSubmit(evt) {
      evt.preventDefault();
      axios
        .post(`http://localhost:3000/api/auth/register/`, this.register)
        .then(response => {
          alert("Registered successfully");
          this.$router.push({
            name: "Login"
          });
        })
        .catch(e => {
          console.log(e);
          this.errors.push(e);
        });
    }
  }
};
</script>
```

Finalmente, para la función de cierre de sesión, simplemente agregue el botón de cierre de sesión dentro de la página Lista de libros y configure la acción solo para borrar el almacenamiento local con el elemento "`jwtToken`". Abra nuevamente "`src/components/BookList.vue`" y luego agregue la función de cierre de sesión al método.

```
methods: {
    details(book) {
      this.$router.push({
        name: "ShowBook",
        params: { id: book._id }
      });
    },
    logout() {
      localStorage.removeItem("jwtToken");
      this.$router.push({
        name: "Login"
      });
    }
  }
};
```

También agregue el botón al lado del encabezado.

```
      <h2>
        Book List
        <b-link @click="logout()">(Logout)</b-link>
      </h2>
```

Ejecute y pruebe la aplicación segura MEVN Stack

```
npm start
```

Abra el navegador y luego apunte a http://localhost:3000/