|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Carrera** | **Plan de estudios** | **Clave materia** | **Nombre de la materia** | **Semestre** | **Grupo** | **Periodo** |
| **ISC** | **ISIC-2010-** | DAM-1904 | Desarrollo de Aplicaciones para el  Comercio Electrónico | **9** |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Practica No.** | **Laboratorio de:** | **Nombre de la practica** | **Duración (Hora)** |
| **3** | **CÓMPUTO** | APLICACIONES PARA EL LADO DEL SERVIDOR | **10HRS** |

|  |  |
| --- | --- |
| **Unidad temática: 1** | **Subtema:**  CONCEPTOS BÁSICOS DE LA HERRAMIENTA DE DESARROLLO DEL LADO DEL SERVIDOR (mvc web services) |
| **Competencia especifica materia:**  Desarrolla aplicaciones para comercio electrónico en la nube utilizando software libre. | |
| **Competencia de la práctica:**  Crea y aplica funcionalidades especiales para aplicaciones de comercio electrónico  como proyecto final | |
| **Materiales:**  Visual Code, librerías API, Dependencias, REST | |
| **Desarrollo**  Construcción del controlador de la API de nuestro proyecto, para ello crea un nuevo folder llamado “controlador” dentro del proyecto con el siguiente código  'use strict'  function **prueba**(req, res) {      res.**status**(200).**send**({          mesagge: 'Probando una accion del controlador de usuarios del api REST con node y mongo'      });  }  module.exports = {  **prueba**  };  Llamaremos al archivo usuarioControl.js, observaras que el método puede recibir o enviar peticiones, en este caso particular responde el siguiente mensaje en el navegador cuando es invocado el método prueba 'Probando una accion del controlador de usuarios del api REST con node y mongo'  El status 200 significa que el servicio web se ha cargado correctamente, y usamos un módulo para usar este js en otros archivos js.  Ahora, viene algo interesante vamos hacer un especie de middleware, es decir, vamos a usar la api express para lanzar el servicio web (pudimos hacer todo en un archivo) el uso de middleware es para ordenar mejor los procesos en el mvc.  El siguiente código manda a llamar la funcionalidad del usuarioControl y gracias a express podemos hacer la petición get desde el navegador e invocar el método prueba. Finalmente, agregamos module para que nuestro js sea usado en todo el proyecto, principalmente en app.js. Crea un archivo llamado usuarioRuta.js e ingresa el código descrito anteriormente  'use strict'  var **express** = **require**('express');  var UsuarioControl = **require**('../controlador/usuarioControl');  var api = **express**.**Router**();  api.**get**('/probando-controlador', UsuarioControl.**prueba**);  module.exports = api;  Ahora, realizamos una modificación en app.js para invocar al api de ruta que hemos creado, observa que ingrese “/api” para ejecutar el servicio para tener un mejor orden de las ejecuciones. También ya no necesitaremos a otro método get que se usaba de forma local, ahora tenemos un diseño mvc más en forma.  'use strict'  var **express** = **require**('express');  var **bodyParser** = **require**('body-parser');  var app = **express**();  var user\_routes = **require**('./rutas/usuariosRuta');  app.**use**(**bodyParser**.**urlencoded**({ extended: false }));  app.**use**(**bodyParser**.**json**());  *//configurar cabeceras http*  *// rutas base*  app.**use**('/api', user\_routes);  *//app.get('/pruebas', function(req, res) {*  *//    res.status(200).send({ mesage: 'Bienvenido  al curso Ivan Azamar' });*  *//});*  module.exports = app;  Ahora, ya estamos preparados para realizar operaciones con los usuarios, como por ejemplo: registro, eliminar, actualizar o consultar. Realizaremos la primera operación  Uniendo el modelo con el control, observemos los cambios en el código del control en usuarioControl.js  Vamos a modificar el Control de los =os, observa que necesitamos al modelo y también requerimos a bycript para encriptar la constraseña. Analiza el método registrarUsuario, en donde creamos una instancia del modelo (asocia los códigos), y recibimos datos (req) del curpo de la web (body) es decir por el método POST, posteriormente, guardamos los datos y verificamos el password  'use strict'  var bcrypt = **require**('bcrypt');  var usuariosModelo = **require**('../modelo/usuarios');  function **prueba**(req, res) {      res.**status**(200).**send**({          mesagge: 'Probando una accion del controlador de usuarios del api REST con node y mongo'      });  }  function **registrarUsuario**(req, res) {      var usuario = new **usuariosModelo**();      var params = req.body; *//recibe todos los datos por POST*      console.**log**(params);      usuario.nombre = params.nombre;      usuario.apellido = params.apellido;      usuario.email = params.email;      usuario.password = params.password;      usuario.rol = 'ROLE\_USER';      usuario.imagen = 'null';      if (params.password) {  *//encriptar ocntraseña y guradar datos*      } else {          res.**status**(500).**send**({ mesagge: 'Introduce la contraseña' })      }  }  module.exports = {  **prueba**  };  Ejecuta con nodemon y sigue comprobando en el navegador, nada nuevo al momento. Incluso lo podemos hacer con PostMan la prueba del método GET como se observa en la imagen    Recuera, lo que programamos fue la recepción de parámetros con el método POST, usaremos a POSTMAN para ello en el control exportaremos el método que hemos creado, realiza el siguiente cambio:  module.exports = {  **prueba**, **registrarUsuario**  };  Ahora, debemos ir a la rutas y crear una nueva pero en esta ocasión usaremos el método post, modifica el código de rutas de usuario y agrega  api.**post**('/registro', UsuarioControl.**registrarUsuario**);  Guarda cambios, y vamos al postman y prueba el ingreso de datos como se muestra en la imagen siguiente        Como puedes observar se han recibido los parámetros pro el método POST mediante la librera express y se han guardado en el modelo a través de un esquema y con el console.log añadido podemos ver que se imprimen en visual code. También body-parser convierte la información en json y utilizable  Ahora, vamos a encriptar el password para poder guardar el registro en la collection, para ello revisa muy bien la documentación en <https://www.npmjs.com/package/bcrypt>, ya que las instrucciones pueden cambiar con el tiempo.  De la misma forma revisa los status de error del protocolo http en <https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP>  Ahora, usaremos también un métodos de mongoose llamado sabe, que usa un callback tipo flecha para manejar un error o guardar el usuario  'use strict'  const bcrypt = **require**('bcrypt');  const usuarios = **require**('../modelo/usuarios');  var usuariosModelo = **require**('../modelo/usuarios');  function **prueba**(req, res) {      res.**status**(200).**send**({          mesagge: 'Probando una accion del controlador de usuarios del api REST con node y mongo'      });  }  function **registrarUsuario**(req, res) {      var usuario = new **usuariosModelo**();      var params = req.body; *//recibe todos los datos por Por el Metodo POST*      console.**log**(params);      usuario.nombre = params.nombre;      usuario.apellido = params.apellido;      usuario.email = params.email;      usuario.rol = 'ROLE\_USER';      usuario.imagen = 'null';      if (params.password) {          bcrypt.**hash**(params.password, 10, function(err, hash) {              usuario.password = hash;              if (usuario.nombre != null && usuario.apellido != null && usuario.email != null) {  *//guardar el ususario en BD*                  usuario.**save**((err, usuarioAlmacenado) => {                      if (err) {                          res.**status**(500).**send**({ mesagge: 'Error al guardar el usuario' });                      } else {                          if (!usuarioAlmacenado) {                              res.**status**(404).**send**({ mesagge: 'No se ha registrado el ususario' });                          } else {  *//nos devuelve un objeto con los datos del ususario guardado*                              res.**status**(200).**send**({ usuarios: usuarioAlmacenado });                          }                      }                  });              } else {                  res.**status**(200).**send**({ mesagge: 'Introduce todos los campos' });              }          });      } else {          res.**status**(500).**send**({ mesagge: 'Introduce la contraseña' });      }  }  module.exports = {  **prueba**,  **registrarUsuario**  };    Se insertaron algunos campos, ya que los demás no coinciden, realiza modificaciones al modelo para que esto sea posible, no es necesario tener construida la colección para que el código cree la misma, recuerda que con model estamos creando la colección y con save guardamos en la BD  En la siguiente imagen se muestra como quedo la el modelo    Si le realizaste modificaciones al modelo, muestra la colección y observa el password con hash, ahora de aquí en adelante haremos los tratamientos necesarios para controlar el password, hemos terminado una sección en donde se ha creado el servidor web, la conexión a la base de datos con inserción, la función para recibir y mandar un mensaje a través del web services y encriptar información. | |
| **Resultados** | |
| **Conclusiones:**  Realiza tus conclusiones una cuartilla completa | |
| **Referencias bibliográficas:**  Anota la referencias de tres libros que correspondan al tema | |
| **Anexos** | |

**FECHA DE ELABORACIÓN: 01-09-2021**