

## DESARROLLO DE APLICACIONES WEB AVANZADO

### LABORATORIO N° 04

## Drives para Node.js de MongoDB



<b>Alumno(s):</b>					<b>Nota</b>	
<b>Grupo:</b>			<b>Ciclo:V</b>			
<b>Criterio de Evaluación</b>	<b>Excelente (4pts)</b>	<b>Bueno (3pts)</b>	<b>Requiere mejora (2pts)</b>	<b>No accept. (0pts)</b>	<b>Puntaje Logrado</b>	
Entiende cómo funciona mongoose						
Utiliza mixins para reutilización de código						
Desarrolla aplicaciones web						
Realiza con éxito lo propuesto en la tarea.						
Es puntual y redacta el informe adecuadamente						

## Laboratorio 04: Drives para Node.js de MongoDB

### Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Entender el funcionamiento de Jade
- Desarrollar aplicaciones web con el framework Express
- Implementación correcta de mongoose

### Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

### Equipos y Materiales:

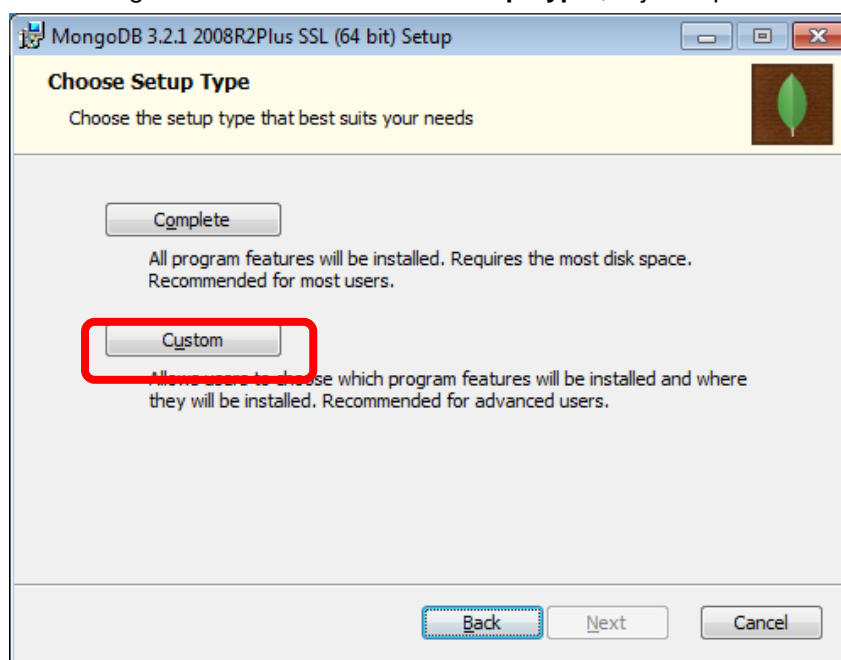
- Una computadora con:
  - Windows 7 o superior
  - VMware Workstation 10+ o VMware Player 7+
  - Conexión a la red del laboratorio
- Máquinas virtuales:
  - Windows 7 Pro 64bits Español - Plantilla
- Instalador de node.js

### Procedimiento:

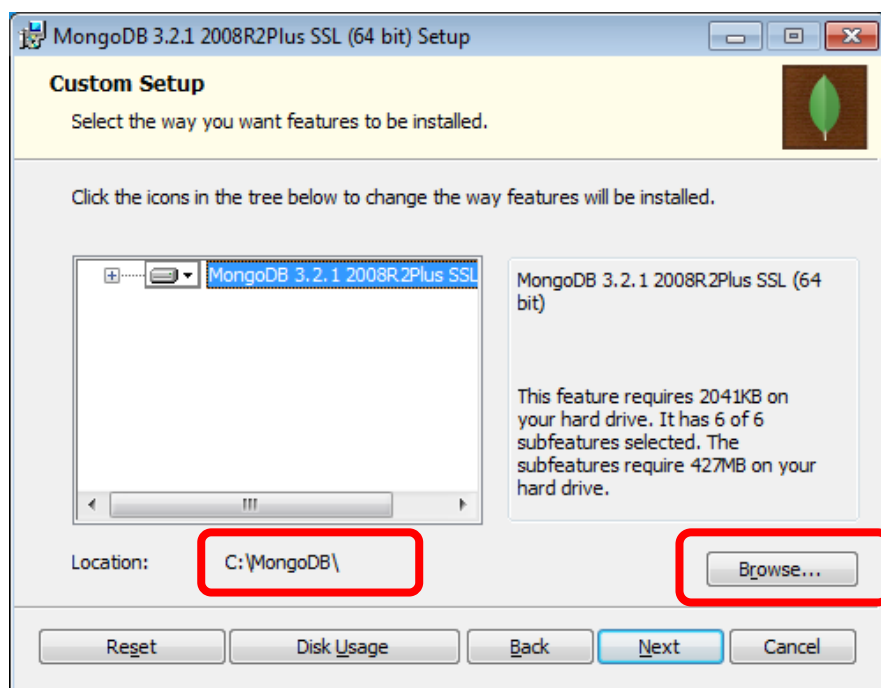
#### Lab Setup

#### 1. Proceso de instalación de MongoDB

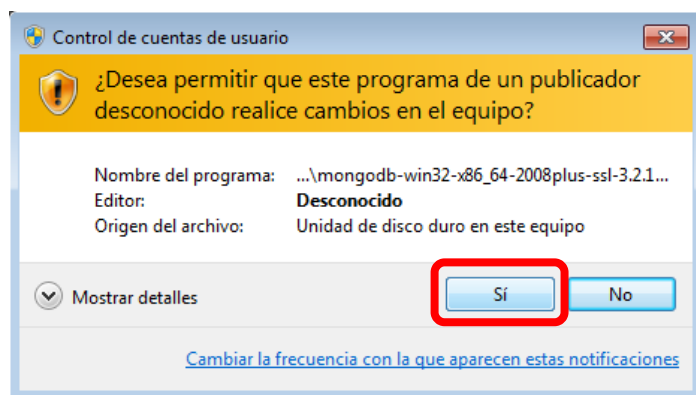
- 1.1. Solicite al instructor, el archivo de instalación de MongoDB  
**mongodb-win32-x86\_64-2008plus-ssl-3.2.1-signed.msi**
- 1.2. Copie el archivo instalador al escritorio del equipo virtual
- 1.3. Doble clic en el instalador para iniciar el proceso de instalación.
- 1.4. Cuando llegue a la ventana “**Choose Setup Type**”, elija la opción “**Custom**”



- 1.5. En la ventana “**Custom Setup**”, empleando el botón **Browse**, cambie la ruta de instalación a la carpeta **C:\MongoDB**



- 1.6. Proceda con la instalación del programa  
1.7. En caso se solicite autorizar cambios que realice el programa. Clic en el botón **Si**



- 1.8. Espere que finalice el proceso de **instalación**. Clic en el botón **Finish**

## Configuración de MongoDB

### 2. Creación de la carpeta para almacenamiento de datos

- 2.1. Crear una carpeta de nombre **Data** dentro de la carpeta **c:\MongoDB**  
2.2. Crear una carpeta de nombre **db** dentro de la carpeta **c:\MongoDB\Data**  
2.3. Ejecutar el siguiente comando dentro de **C:\MongoDB\bin**

```
>mongod --dbpath c:\MongoDB\Data\db
```

- 2.4. Esto iniciará la instancia de MongoDB a la que accederemos durante el laboratorio. Para conectarse a dicha instancia, en otra consola ubicada en **C:\MongoDB\bin** ejecutamos:

```
>mongo
```

- 2.5. Estamos conectados a la base de datos **test**, para validar que todo está ok ingresaremos un documento de prueba y luego haremos la consulta para ver su existencia.

```
> db.producto.insert({nombre:'prueba',descripcion:'este es un producto de prueba',precio:'5'});  
  
> db.producto.find()
```

### 3. Continuación de laboratorio anterior

- 3.1. En el laboratorio anterior usted creó un sitio web con el Framework Express, para poder representar el trabajo de su grupo de proyecto. Usará el mismo código para nuestra sesión actual.
- 3.2. Ya que tenemos node.js instalado y express configurado, procederemos a instalar mongoose para continuar con nuestra implementación.

```
>npm install mongoose --save
```

- 3.3. Usaremos también body-parser, el cual nos permite hacer un tratamiento de los argumentos recibidos durante las peticiones.

```
>npm install body-parser --save
```

- 3.4. En la raíz de nuestro proyecto, crearemos el archivo lab10.js con el siguiente contenido

```
var express = require('express'),  
app = express(),  
bodyParser = require('body-parser'),  
producto = require('./models/prod');  
  
app.use(bodyParser.json());  
  
app.set('view engine','jade');  
  
app.get('/',function(req,res){  
    res.send('Hola mundo');  
})_  
  
app.get('/producto', producto.show);  
app.post('/producto', producto.create);  
app.post('/producto/update', producto.update);  
app.post('/producto/delete', producto.delete);  
  
app.listen(9090,function(){  
    console.log('Iniciando!');  
});
```

- 3.5. Estamos creando un tratamiento de datos para nuestra colección producto a través de las cuatro operaciones principales del CRUD, indique que líneas corresponden a cada una de estas.

---

---

---

---

- 3.6. Necesitamos crear la carpeta models donde ubicaremos nuestros archivos de conexión a MongoDB. Dentro de esta carpeta crearemos el archivo prod.js con el siguiente contenido.

```
var mongoose = require('mongoose'),
    Schema = mongoose.Schema;

mongoose.connect('mongodb://localhost/test');

var producto_schema = new Schema({
  nombre: String,
  descripcion: String,
  precio: String
});
prod_model = mongoose.model('producto', producto_schema, 'producto');

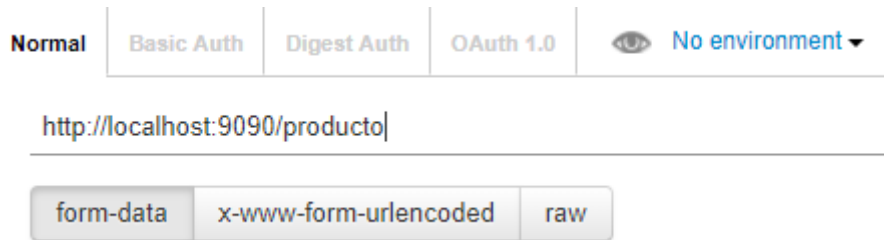
module.exports = {
  show: function(req, res){
    prod_model.find({}, function(err, items){
      if (!err) {
        res.send(items);
      } else {
        return console.log(err);
      }
    });
  },
  create: function(req, res){
    //
  },
  update: function(req, res){
    //
  },
  delete: function(req, res){
    //
  },
};
```

- 3.7. Para poder hacer operaciones sobre las colecciones de MongoDB, debemos declarar un Schema. Nosotros hemos declarado uno con los campos de producto. Así como MongoDB proporciona distintos tipos de variables, mongoose (el encargado de la conexión con node.js) también soporta varios tipos de variables en la declaración del Schema.

```
var schema = new Schema({
  name: String,
  binary: Buffer,
  living: Boolean,
  updated: { type: Date, default: Date.now },
  age: { type: Number, min: 18, max: 65, required: true },
  mixed: Schema.Types.Mixed,
  _someId: Schema.Types.ObjectId,
  array: [],
  ofString: [String],
  nested: { stuff: { type: String, lowercase: true, trim: true } }
});
```

- 3.8. En el módulo exportado del archivo prod.js, hemos creado una función show. Esta hace referencia a prod\_model, que es un modelo del Schema antes creado. Nosotros hacemos operaciones sobre los modelos, quienes a su vez utilizan los esquemas para poder parsear la data recibida; es decir, para comunicarnos con MongoDB desde node.js necesitamos de un modelo, y el modelo necesita un Schema para poder conocer la estructura de datos a recibir.

En el laboratorio de la semana 4 instalamos el navegador Chrome con la extensión Postman REST. Haga uso de él para hacer las pruebas pertinentes.



- 3.9. Adjunte una captura del resultado de ejecutar un GET en la url localhost:9090/producto. Ya hemos probado que tenemos conexión con nuestra base de datos, procederemos a crear las funciones restantes para tratar con toda la data. El archivo prod.js terminará luciendo de la siguiente manera.

```
var mongoose = require('mongoose'),
    Schema = mongoose.Schema;

mongoose.connect('mongodb://localhost/test');

var producto_schema = new Schema({
  nombre: String,
  descripcion: String,
  precio: String
});
prod_model = mongoose.model('producto', producto_schema, 'producto');

module.exports = {
  show: function(req, res){
    if(req.query._id==null){
      prod_model.find({},function(err,items){
        if (!err) {
          res.send(items);
        }else{
          return console.log(err);
        }
      });
    }else{
      prod_model.findOne({_id: req.query._id},function(err,items){
        if (!err) {
          res.send(items);
        }else{
          return console.log(err);
        }
      });
    }
  },
}
```

```

    create: function(req,res){
      var item = {
        nombre: req.query.nombre,
        descripcion: req.query.descripcion,
        precio: req.query.precio
      };
      var nuevo = new prod_model(item).save();
      res.send(nuevo);
    },
    update: function(req,res){
      prod_model.findOne({_id: req.query._id},function(err,producto){
        producto.nombre = req.query.nombre;
        producto.descripcion = req.query.descripcion;
        producto.precio = req.query.precio;
        producto.save();
        res.send(producto);
      });
    },
    delete: function(req,res){
      prod_model.findOne({_id: req.query._id},function(err,producto){
        producto.remove();
        res.send({status:true});
      });
    },
  },
};

```

- 3.10. Adjunte capturas haciendo pruebas de las cinco operaciones, tome en cuenta que la url localhost:9090/producto ahora acepta un argumento \_id opcional, en caso de ser recibido, devolverá el documento que coincida con ese \_id.
- 3.11. Implementaremos una vista de login, por lo que agregaremos el siguiente middleware en nuestro archivo principal.

```

app.get('/login',function(req,res){
  res.render('login');
});

```

- 3.12. Seguidamente, en la carpeta views, crearemos el archivo login.jade con el siguiente contenido.

```

html
  head
    meta(name='viewport', content='width=device-width, initial-scale=1')
    title Sistema
    link(rel='stylesheet', href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css')
  body
    block content
      .container
        form.form-signin(action="/login", method="post")
          h2.form-signin-heading Ingrese sus datos
          input.input-block-level(type="text", name="username", placeholder="Email")
          input.input-block-level(type="password", name="password", placeholder="Password")
          button.btn.btn-large.btn-primary(type="submit") Iniciar sesion

```

- 3.13. Antes de continuar con nuestra implementación de una validación de usuarios, haremos pruebas con jade y mongoose. Implementemos el middleware table para renderizar una tabla en bootstrap.

```

app.get('/table',function(req,res){
  res.render('table');
});

```

- 3.14. Cree la vista table.jade en la carpeta views.

```

table .table
  head
    tr
      th ID
      th Nombre
      th Descripcion
      th Precio
  tbody
    tr
      td _id
      td nombre_ejemplo
      td descr_ejemplo
      td precio_caro
  
```

- 3.15. Adjunte una captura de la vista generada en el navegador.
- 3.16. Ahora mezclaremos lo aprendido. Modifique la función show, ubicada en prod.js, de tal forma que luzca así.

```

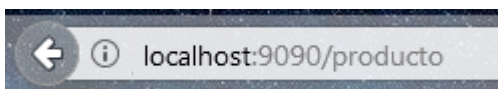
show: function(req,res){
  prod_model.find({},function(err,items){
    if (!err) {
      //res.send(items);
      res.render('table',{data: items});
    }else{
      return console.log(err);
    }
  });
},
  
```

- 3.17. Ahora modificaremos la vista table.jade para que luzca de la siguiente manera

```

table.table
  head
    tr
      th ID
      th Nombre
      th Descripcion
      th Precio
  tbody
    for item in data
      tr
        td= item._id
        td= item.nombre
        td= item.descripcion
        td= item.precio
  
```

- 3.18. Adjunte capturas con el resultado de ingresar a la siguiente URL.



- 3.19. Implemente un archivo user.js en la carpeta models con toda la lógica necesaria para acceder al sistema (hacer que los usuarios validen ante esa vista login).



- 3.20. Implemente un CRUD para la tabla de productos. Para su facilidad, se adjunta un ejemplo de cómo luce un formulario en jade.

```
.main.container
  .row
    .col-md-6.col-md-offset-3
      h1.display-4.m-b-2 Crear Producto
      form(method='POST' action='/producto')
        div.form-group
          label(for='nombre') Nombre:
          input#nombre.form-control(type='text' name='nombre')
        div.form-group
          label(for='descripcion') Descripción:
          input#descripcion.form-control(type='text' name='descripcion')
        div.form-group
          label(for='precio') Precio:
          input#precio.form-control(type='text' name='precio')
        button.btn.btn-primary(type='submit') Grabar
```

#### 4. Finalizar la sesión

- 4.1. Apagar el equipo virtual
- 4.2. Apagar el equipo

## **Tarea**

Implementar el CRUD de al menos tres maestros generales de su proyecto en MongoDB con vistas basadas en Jade y Bootstrap. Adjuntar capturas del código utilizado y de la estructura del proyecto.

## **Conclusiones:**

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.