CRUD Node.JS/MongoDB

Este apunte es una guia rápida de como realizar una aplicación web sencilla usando las siguientes tecnologías.

- Node.Js. Crea el servidor por medio del framework Express.js
- MongoDB. Base de datos para persistencia.
- Bootstrap 4. Framework de css para estilizar las vistas.
- Pug. Motor de plantillas.

Fuente:

1 – Primeros pasos

1) Creamos el archivo de metadatos del proyecto **package.json**

```
npm init
```

2) Instalamos los modulos necesarios para el proyecto

```
npm i express pug mongoose morgan --save
```

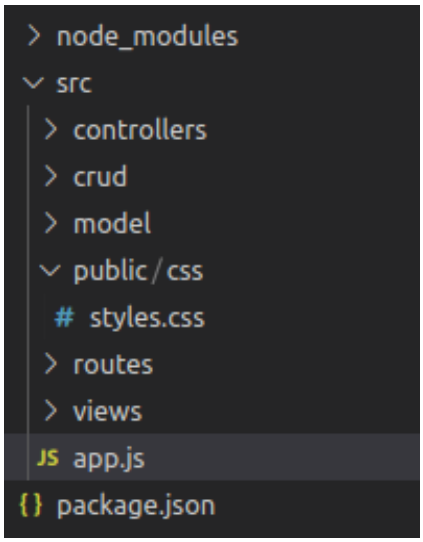
3) Para facilitar el desarrollo instalamos nodemon como dependencia de desarrollador.

```
Npm i nodemon -D
```

En la sección scripts del package.json declaramos el script para corre el proyecto con nodemon

```
"scripts": {
   "start": "node src/app.js",
   "dev": "nodemon src/app.js"
},
```

4) Creamos la estructura del proyecto con los siguientes archivos y carpetas



5) Antes de empezar con el modelo, creamos el código general del servidor. Importamos los módulos necesarios, inicializamos express, configuramos la conexión a la base de datos, seteamos algunas configuraciones de express (puerto, vistas y motor de plantillas) y la ubicación de la carpeta de archivos estáticos, algunos middlewares (morgan, urlEncoded) y por ultimo inicializamos el servidor.

app.js

```
//module imports
const express = require('express');
const path = require('path');
const morgan = require('morgan');
const mongoose = require('./config/connection');

const app = express();

//Routes imports
const indexRoutes = require('./routes/index');

//settings
app.set('port', process.env.PORT || 3000); //Puerto asignado por el host, sino localhost 3000
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

//static files
app.use(express.static(path.join(__dirname, 'public')));

//Middlewares
app.use(morgan('dev')); //Mostrar peticiones por consola
app.use(express.urlencoded({extended: false})); // codificación de datos json

//routes
app.use('/', indexRoutes);

//server start
app.listen(app.get('port'), ()=>{
    console.log (`Server on port ${app.get('port')}`)
})
```

6) para una primer prueba de funcionalidad creamos los siguientes archivos con su respectivo contenido.

router/index.js

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

router.get('/', (req, res)=>{
```

```
    res.render('index', {title: 'default route', msg: 'Crud Nodejs/mongoDB'})
});
module.exports = router;
```

views/template/template.pug

```
<!DOCTYPE html>
html(lang="en")
    head
        meta(charset="UTF-8")
        meta(name="viewport", content="width=device-width, initial-scale=1.0")
        //Bootstrap 4
        link(rel="stylesheet", href="/bootstrap/css/bootstrap.min.css")
        //Styles
        link(rel="stylesheet", href="/css/styles.css")
        title= title
    body
        block content
            include ../partials/footer.pug
```

views/partials/footer.pug

```
footer.text-center
    p Carlossmarts@gmail.com
    script(src="/bootstrap/js/bootstrap.min.js")
```

views/index.pug (Esta vista se renderiza desde la ruta '/')

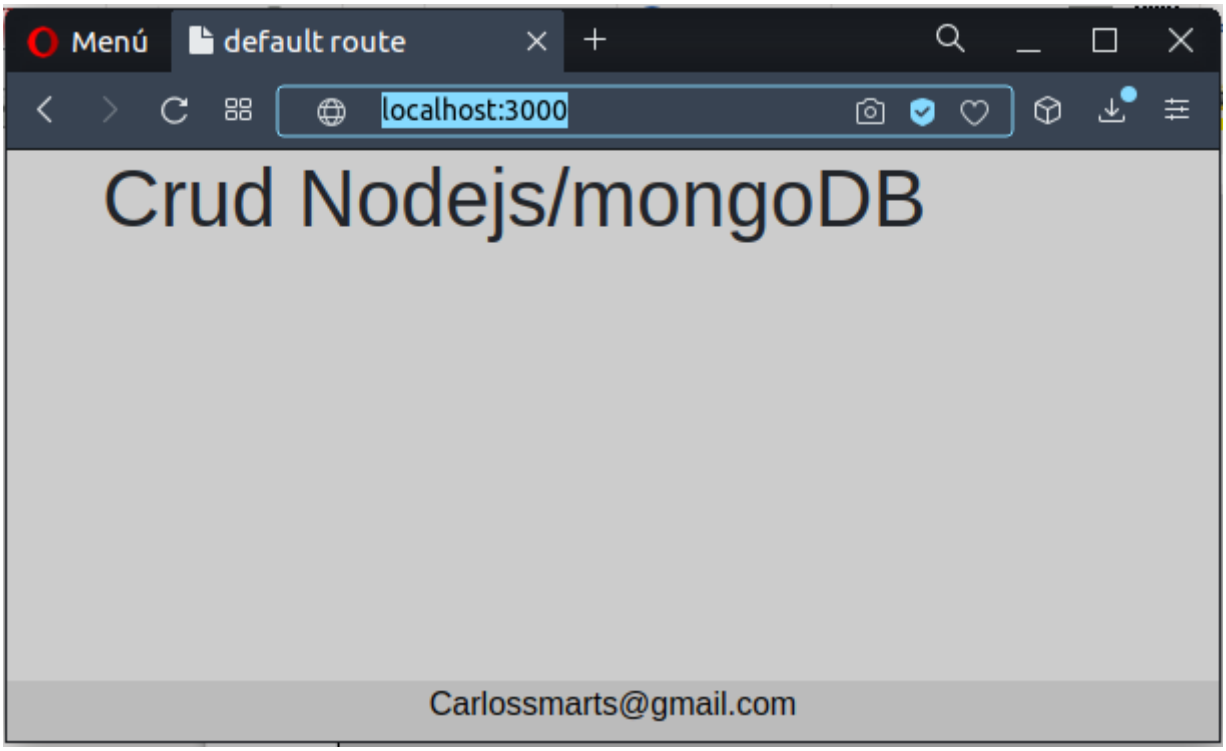
```
extends ./template/template
prepend content
        h1= msg
```

public/css/styles.css

```
body{
    background-color: #ccc;
}
html{
    min-height: 100%;
```

```
    position: relative;
}
footer{
    background-color: #bbb;
    position: absolute;
    bottom: 0;
    width: 100%;
    height: 30px;
    color: black;
}
```

Con esta estos archivos , al acceder desde el navegador a **http://localhost:3000/** se obtiene la siguiente vista



2 – Vistas

Primero vamos a generar las vistas y luego les damos funcionalidad y generamos la conexión a base de datos. (en realidad estos pasos deberían hacerse en simultaneo para cada caso de uso, pero como son pocos sirve mejor como ejemplo).

1) Agregamos en el archivo template.pug una navegación para que se mueste como encabezado en todas las páginas.

views/template/template.pug

```
<!DOCTYPE html>
html(lang="en")
    head
        meta(charset="UTF-8")
        meta(name="viewport", content="width=device-width, initial-scale=1.0")
        //FontAwesome
        link(rel="stylesheet", href="/fontawesome/css/all.css")
        //Bootstrap 4
        link(rel="stylesheet", href="/bootstrap/css/bootstrap.min.css")
        //Styles
        link(rel="stylesheet", href="/css/styles.css")
        title CRUD | Node.js MongoDB
    body
        nav.navbar.navbar-dark.bg-dark.mb-4
            a.navbar-brand(href="/") CRUD NodeJS/MongoDB
        block content
            include ../partials/footer.pug
```

Creamos en index.pug una tabla en la que se mostrarán los datos del esquema y un formulario para las altas y modificaciones. Estos últimos son escencialmente el mismo, pero varían en la información que reciben. Mientras el formulario de alta no recibe ningún dato, el de modificación requiere el elemento a modificar, por lo que muestra sus valores y tiene el id como input:hidden.

views/index.pug

```
extends ./template/template
prepend content
    .container
        .row
            .col-md-8.offset-2.text-info
                a(href='/add')
                    i.fas.fa-plus-circle Nueva tarea
        .row
            .col-md-8.offset-2
                //tabla
```

```
            table.table.table-bordered.table-hover
                thead
                    tr
                        th N°
                        th Titulo
                        th Descripcion
                        th Acciones
                tbody
                    - let i = 1
                    each task in tasks
                        tr
                            td= `${i++}`
                            td= task.title
                            td= task.description
                            td
                                .row.justify-content-around
                                    a.btn.btn-sm(href="/done/" + task._id
                                                class= !task.done? "btn-dark" : "btn-success") Hecho
                                    a.btn.btn-danger.btn-sm(href="/delete/"+task._id) Eliminar
                                    a.btn.btn-info.btn-sm(href="/update/"+task._id) Modificar
```

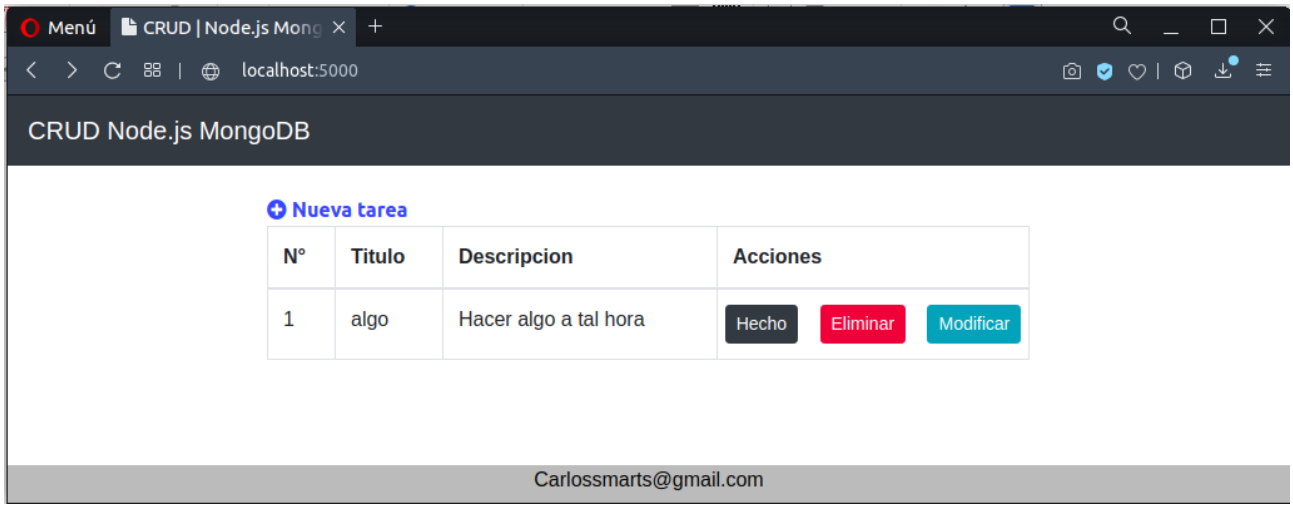
views/updForm

```
extends ./template/template
prepend content
    .row.justify-content-center
        //formulario
        .col-md-6.mb-5.col-sm-8
            .card
                .card-body
                    form(action="/update", method="POST")
                        input(type="hidden", name="_id" value= ""+ task._id)
                        .form-group
                            input.form-control(type="text" name="title" value= `${task.title}`)
                        .form-group
                            textarea.form-control(name="description" cols="80")
                                = `${task.description}`
                        button.btn.btn-primary.btn-block(type="submit") Guardar
```

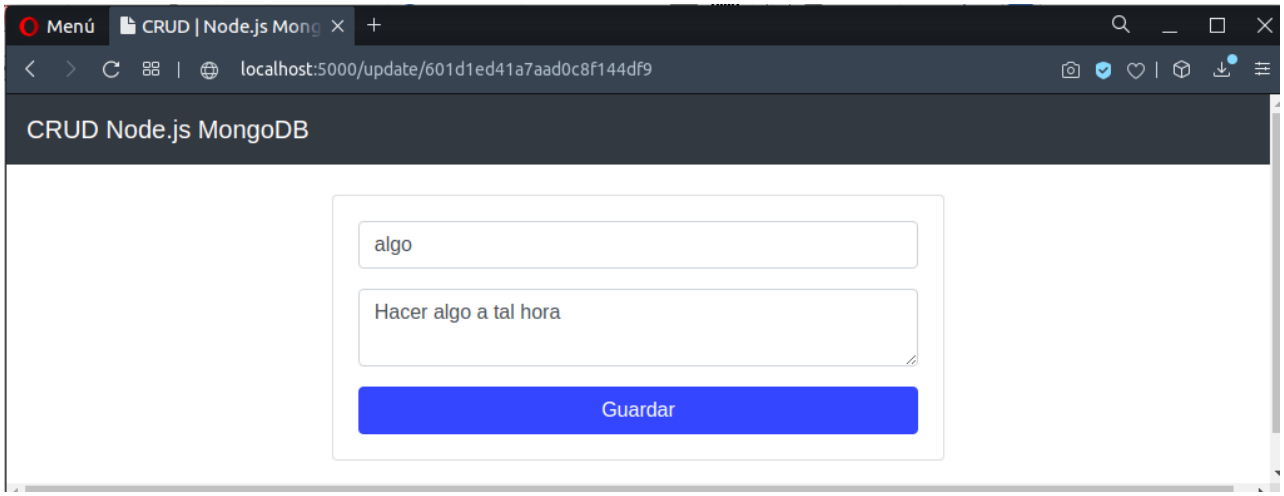
*Los elementos del formulario deben llamarse como los atributos del esquema

Estos documentos generan las siguientes vistas

pagina principal



Formulario de actualización



3 – Controladores y conexión a base de datos

Una vez generada la vista principal pasamos a crear las rutas y los controladores que gestionan cada petición. Como son pocos y todos hacen referencia a un mismo modelo de datos, creamos un solo archivo para las rutas y uno para los controladores.

1) Primero creamos un archivo de configuración para la conexión a la base de datos

config/connection.js (Este se importa desde app.js)

```
const mongoose = require('mongoose');

//DB connection
const uri = 'mongodb://localhost/agenda';
mongoose.connect(uri,{
    useNewUrlParser: true,
```

```
    useUnifiedTopology: true
})
    .then(db => console.log('DB connected'))
    .catch(err => console.log(err));
module.exports = mongoose;
```

2) Creamos un esquema simple de tareas a realizar

model/task.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const taskSchema = new Schema({
    title: String,
    description : String,
    done : {
        type: Boolean,
        default: false
    }
});
module.exports = mongoose.model('Tasks', taskSchema);
```

3) Creamos el elemento router y asignamos las rutas que se deben gestionar. En lugar de implementar directamente los callbacks usamos funciones async/await en un archivo aparte.

routes/index.js

```
const express = require('express');
const taskController = require('../controllers/taskControllers');
const router = express.Router();

//traer
router.get('/', taskController.getTasks);

//alta
router.get('/add', taskController.addForm);
router.post('/create', taskController.createTask);

//baja
router.get('/delete/:id', taskController.deleteTask);
```

```
//modificacion
router.get('/update/:id', taskController.updateForm);
router.post('/update', taskController.updateTask);

//hecho
router.get('/done/:id', taskController.done);

module.exports = router;
```

4) Creamos los controladores con funciones asíncronas.

```
const Task = require('../model/Task'); //const { findByIdAndDelete, findById } = require('../model/Task');modelo


//Traer
const getTasks = async (req, res)=>{
    const tasks = await Task.find(); //trae todas las tareas
    res.render('index', {tasks});
}

//alta
const addForm = async(req, res)=>{
    res.render('addForm', )
}

const createTask = async (req, res)=>{
    const newTask = new Task(req.body);
    await newTask.save();
    res.redirect('/');
}

//baja
const deleteTask = async (req, res)=>{
    let { id } = req.params;
    await Task.findByIdAndDelete(id);
    res.redirect('/');
}

//modificacion
const updateForm = async (req, res)=>{
    let { id } = req.params;
```

```
    let task = await Task.findById(id);
    res.render('updForm', {task});
}
const updateTask = async (req, res)=>{
    let id = req.body._id;
    const updTask = await Task.findByIdAndUpdate(id, req.body);
    res.redirect('/');
}

const done = async (req, res)=>{
    let { id } = req.params;
    const task = await Task.findById(id);
    console.log(task);
    task.done = !task.done; //cambio de estado
    await task.save();
    res.redirect('/');
}

module.exports = {
    getTasks,
    createTask,
    addForm,
    deleteTask,
    updateForm,
    updateTask,
    done
}
```

*Hasta este punto el ABM Se encuentra funcionando. El siguiente paso es mejorar un poco la aplicación usando ajax y ventanas modales.