

DESARROLLO WEB CON MEAN

Curso: DESARROLLO WEB CON MEAN (WEB FULL STACK DEVELOPER)

MEAN Stack

Introducción a MEAN

Solución de desarrollo web en un solo lenguaje

Validación fácil y flexible con MongoDB y Mongoose

Sin hilos gracias a NodeJS

Plantillas del lado del cliente dinámicas con Angular

MEAN Stack

Introducción a MEAN



Introducción a MEAN

Tanto MongoDB, como NodeJS y Angular usan el mismo tipo de objetos

```
{ _id: ObjectId("1627a72b62738f7a66000003"),  
  Usuario: "Ruben" }
```

MEAN Stack

Instalación

Prerequisitos:

Nodejs y npm : NodeJS del lado del servidor y npm como gestor de paquetes

MongoDB: Como base de datos nosql

Bower: como gestor de paquetes frontend

```
$ npm install -g bower
```

Grunt: Para automatización de tareas de desarrollo.

```
$ npm install -g grunt-cli
```

MEAN Stack

Instalación

Para instalar mean podemos usar el repositorio de GitHub

```
$ git clone https://github.com/meanjs/mean.git meanjs
```

Dentro del directorio que habremos renombrado, resolvemos las dependencias

```
$ npm install
```

Para lanzar la aplicación

```
$ grunt
```

MEAN Stack

Estructura de carpetas

App: Contiene todos los ficheros del servidor. En su interior se encuentran las siguientes carpetas

controllers: Almacén de los controladores Express, aquí se encuentra la lógica de negocio del backend

models: Donde se almacenan los modelos de Mongoose

routes: Los ficheros de enrutamiento. Se definen las rutas Express. Deben ser ficheros “.js” y son cargados automáticamente

tests: Carpeta de pruebas de Mocha

views: Fichero de las vistas. Con Angular no es necesario las vistas, pero si las plantillas

templates: Directorio para agregar plantillas

MEAN Stack

Estructura de carpetas

Config: Contiene todos los ficheros de configuración del servidor.
En su interior se encuentran las siguientes carpetas

env: Los ficheros cargados por config.js según el entorno.

strategies: Ficheros de configuración para el fichero
passport.js

config.js: Cargador de configuración que establece los
ficheros a cargar según el entorno

express.js: Configuración de Express

init.js: Fichero de inicialización

passport.js: Configuración de las estrategias de autenticación
que obtiene de la carpeta strategies

MEAN Stack

Estructura de carpetas

Public: Contiene todos los ficheros de configuración del servidor. En su interior se encuentran las siguientes carpetas

dist: Almacen de css y javascript compilados.

modules: Almacenamiento de módulos de AngularJS

config.js: Módulo de configuración de AngularJS. Comienza con dos propiedades globales y un método

 applicationModuleName: Nombre principal del modulo

 applicationModuleVendorDependencies:

application.js: Fichero principal de aplicación AngularJS

MEAN Stack

Estructura de carpetas

Ficheros en raíz: Contiene todos los ficheros de configuración del servidor. En su interior se encuentran las siguientes carpetas

server.js: Inicialización de app de Node.JS.

bower.json: Fichero de definición de bower. Configuración de componentes front-end

Dockerfile: Configuración de comandos para crear una imagen de Docker.

fig.yml: Fichero de configuración del entorno de desarrollo de Docker.

gruntfile.js: Definición de tareas

karma.js: Configuración de tests karma

package.json: Fichero de definición npm

Procfile: Fichero de procesos heroku

MEAN Stack

Enrutados con Express

La configuración de Express se almacena en `config/express.js`.
Existen una serie de rutas predefinidas que se encuentran en
`app/routes/users.server.routes.js`

GET <http://localhost:3000/users/me> : Devuelve el usuario actual de autenticación

POST <http://localhost:3000/auth/signup> : Da de alta un usuario por user y pass

POST <http://localhost:3000/auth/signin> : Logea a un usuario con user y pass

GET <http://localhost:3000/auth/signout> : Cierra la sesión de un usuario

GET <http://localhost:3000/auth/facebook> : Inicio del proceso OAUTH de facebook

GET <http://localhost:3000/auth/facebook/callback> : Uri Callback del OAUTH de facebook

MEAN Stack

Enrutados con Express

GET <http://localhost:3000/auth/twitter> : Inicio del proceso OAUTH de twitter

GET <http://localhost:3000/auth/twitter/callback> : Uri Callback del OAUTH de twitter

GET <http://localhost:3000/auth/linkedin> : Inicio del proceso OAUTH de LinkedIn

GET <http://localhost:3000/auth/linkedin/callback> : Uri Callback del OAUTH de LinkedIn

GET <http://localhost:3000/auth/google> : Inicio del proceso OAUTH de Google

GET <http://localhost:3000/auth/google/callback> : Uri Callback del OAUTH de Google

GET <http://localhost:3000/auth/github> : Inicio del proceso OAUTH de GitHub

GET <http://localhost:3000/auth/github/callback> : Uri Callback del OAUTH de GitHub

MEAN Stack

Enrutados de Aplicación:

Se encuentran en `app/routes/core.server.routes.js`

GET <http://localhost:3000>

Solo sirve una página de aplicación por medio de Angular

MEAN Stack

Passport

Se trata de un Middleware de Aplicación que permite usar distintos métodos de autenticación

Contiene 6 tipos distintos por defecto

Local

Facebook

LinkedIn

Twitter

Google

Github

MEAN Stack

Generación de aplicación con MEAN Stack

Una vez generado el sistema, podemos comprobar el fichero `express.js` para definir la uri de conexión a MongoDB y la base de datos, aunque es más recomendable definir variables y exportar según el entorno.

```
var uri = 'mongodb://localhost/todos';  
var db = require('mongoose').connect(uri);
```

Dentro del `Server.js` viene definido el servidor. Podemos indicar también el puerto donde se ejecuta, o leer del fichero `mongoose.js`

Podemos definir logs para aprovechar y ver que el estado es correcto.

Dentro de Mongoose, podemos aprovechar y definir la dirección del servidor

MEAN Stack

Generación de aplicación con MEAN Stack

Dentro del modelo, en app/models, definimos el modelo de mongoose

```
mongoose.model('Persona', {nombre: String,apellido:String,fecha_nacimiento:Date});
```

Dentro del fichero routes.js definimos los endpoints del API Rest, usando los verbos GET, POST PUT DELETE:

```
var users = require('../app/controllers/personas.server.controller');

module.exports = function(app) {
  app.route('/personas').post(personas.create);
};
```


MEAN Stack

Generación de aplicación con MEAN Stack

Dentro de `personas.server.controller.js`

Por ejemplo:

```
var User = require('mongoose').model('User');
```

```
exports.create = function(req, res, next) {  
  var user = new Persona(req.body);  
  user.save(function(err) {  
    if (err) {  
      return next(err);  
    }  
    else {  
      res.json(persona);  
    }  
  });  
};
```

MEAN Stack

Generación de aplicación con MEAN Stack

Dentro del módulo `HttpModule` de Angular se encuentran las peticiones del API REST. Hay que importarlo en el módulo raíz de la aplicación.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpModule } from '@angular/http';
/* Imports de componentes, pipes, directivas ... */
@NgModule({
  declarations: [
    // Componentes, pipes y directivas
  ],
  imports: [
    BrowserModule,
    HttpModule
  ],
  // Resto de propiedades: providers, bootstrap...
})
export class AppModule { }
```

MEAN Stack

Generación de aplicación con MEAN Stack

Por ejemplo para listar todas las personas (debemos crear el servicio claro!)

```
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import 'rxjs/Rx';

@Injectable()
export class HttpService {
  constructor(private http: Http) { }
  getPersonas() {
    return this.http.get('/personas').map((resp: Response) => {
      return resp.json();
    });
  }
}
```

MEAN Stack

Generación de aplicación con MEAN Stack

Hay que llamar al servicio para obtener las personas.

```
import { Component, OnInit } from '@angular/core';
import { HttpService } from './http.service';

@Component({ /* Propiedades de este decorador: selector, templateUrl... */ })
export class ListaPersonasComponent implements OnInit {
  personas: object[];
  constructor(private httpService: HttpService) {
    this.personas = [];
  }
  ngOnInit() {
    this.httpService.getPersonas().subscribe((datos) => {
      this.personas = datos;
    });
  }
  selectPerson(persona) {
    // ...
  }
}
```

MEAN Stack

Generación de aplicación con MEAN Stack

Dentro de la plantilla del componente ponemos una tabla para mostrar la lista de personas

```
<table class="table table-striped">
  <tr class="info">
    <td>Nombre</td>
    <td>Apellidos</td>
    <td>Fecha de nacimiento</td>
  </tr>
  <tr *ngFor="persona of personas" (click)="selectPerson(persona)">
    <td>{{ persona.nombre }}</td>
    <td>{{ persona.apellido }}</td>
    <td>{{ persona.fecha_nacimiento }}</td>
  </tr>
</table>
```

MEAN Stack

Generación de aplicación con MEAN Stack

Podemos usar passport para una estrategia de autenticación local

```
var passport = require('passport'),
    LocalStrategy = require('passport-local').Strategy,
    User = require('mongoose').model('User');

module.exports = function() {
  passport.use(new LocalStrategy(function(username, password, done) {
    User.findOne(
      {username: username},
      function(err, user) {
        if (err) {
          return done(err);
        }

        if (!user) {
          return done(null, false, {message: 'Usuario desconocido'});
        }

        if (!user.authenticate(password)) {
          return done(null, false, {message: 'Contraseña incorrecta'});
        }

        return done(null, user);
      }
    );
  }));
};
```

MEAN Stack

Generación de aplicación con MEAN Stack

Creamos un fichero passport.js en la carpeta de configuración

```
var passport = require('passport'),
    mongoose = require('mongoose');

module.exports = function() {
  var User = mongoose.model('User');

  passport.serializeUser(function(user, done) {
    done(null, user.id);
  });

  passport.deserializeUser(function(id, done) {
    User.findOne(
      { _id: id },
      '-password',
      function(err, user) {
        done(err, user);
      }
    );
  });

  require('./strategies/local.js')();
};
```

MEAN Stack

Ahora debemos actualizar el modelo para que primero autentique.

```
UserSchema.methods.authenticate = function(password) {  
  var md5 = crypto.createHash('md5');  
  md5 = md5.update(password).digest('hex');  
  
  return this.password === md5;  
};
```

Definiendo si es correcto antes de guardar:

```
UserSchema.pre('save',  
  function(next) {  
    if (this.password) {  
      var md5 = crypto.createHash('md5');  
      this.password = md5.update(this.password).digest('hex');  
    }  
  
    next();  
  }  
);
```


MEAN Stack: Ejercicio

SE PRETENDE REALIZAR UNA APLICACIÓN DE GESTIÓN DE LIBROS

PARA ELLO SE DARÁ DE ALTA EL MODEL LIBRO Y SE CREARÁ EL ACCESO CRUD
EL LIBRO TENDRÁ LOS SIGUIENTES CAMPOS

TITULO

AUTOR

SINOPSIS

ISBN

CATEGORIA[TERROR, DRAMA, NOVELA HISTORICA, ETC...]

SE PODRÁN LISTAR LOS LIBROS, INSERTAR UN NUEVO LIBRO, ACTUALIZAR
LIBROS EXISTENTES Y ELIMINAR LIBROS