



Clase 41. Programación Backend

Desarrollo de un servidor web basado en capas completo



OBJETIVOS DE LA CLASE

- Identificar los marcos de MERN stack.
- Configurar CORS.
- Crear una aplicación completa con API RESTful y un front-end simple.
- Conocer sobre las pruebas de servidores y los tipos.

CRONOGRAMA DEL CURSO

Clase 40



**Arquitectura del servidor:
Persistencia**

Clase 41



**Desarrollo de un
servidor web basado en
capas**

Clase 42



**Testeo de
funcionalidades**

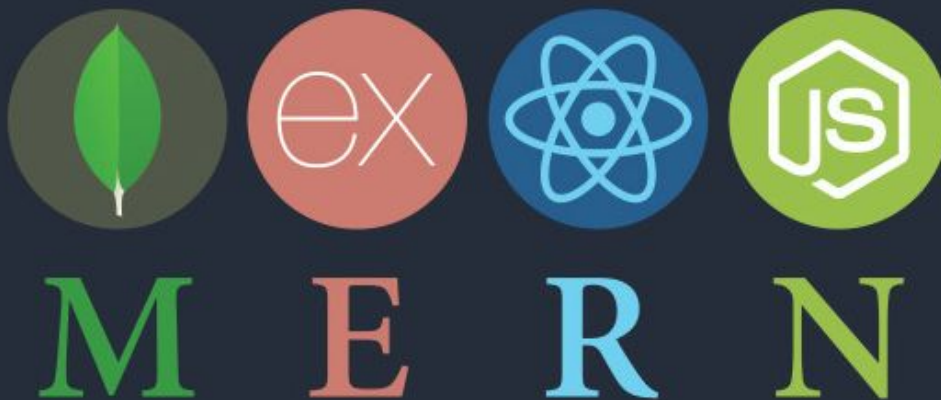
STACK MERN

CODER HOUSE

¿De qué se trata?



- Como vimos al principio del curso, el stack MERN es un conjunto de marcos/tecnologías utilizados para el desarrollo web de aplicaciones que consta de MongoDB, React JS, Express JS y Node JS como sus componentes.
- La combinación de estas cuatro tecnologías nos permite como desarrolladores crear sitios web (y aplicaciones) completos usando React (con JavaScript o TypeScript) del lado del cliente (front-end) y Node JS del lado del servidor (back-end). Así podremos dominar tanto la parte visual (la experiencia del usuario) como la parte lógica del servidor.
- Entonces con este stack, usamos Javascript tanto del lado del cliente como del lado del servidor.



Características de sus tecnologías



MongoDB

- Es una base de datos no relacional.
- Los registros son similares a un objeto JSON.
- Es una base de datos muy flexible.

ExpressJS

- Es un framework de Node diseñado para escribir aplicaciones simplificadas, rápidas y seguras, del lado del servidor.

Características de sus tecnologías



ReactJS

- Es una biblioteca de código abierto, que se utiliza para desarrollar la parte de front-end (interfaces de usuario).
- Permite a los desarrolladores modificar y actualizar la página para ver los cambios, sin tener que reiniciarla.
- Aplicaciones rápidas y escalables.

NodeJS

- Es un entorno de ejecución de JavaScript de código abierto, multiplataforma y diseñado para ejecutarse en el lado del servidor.
- Orientado a eventos asíncronos y diseñado para crear aplicaciones escalables.



Ventajas de MERN



- El stack MERN es un marco sólido para desarrollar aplicaciones dinámicas, interactivas y avanzadas. Tiene alta flexibilidad y escalabilidad.
- Su uso reduce los gastos, necesitando menos personal para obtener el mismo resultado, ya que el stack completo se programa con JavaScript.
- Facilita el proceso de trabajar con una arquitectura modelo vista controlador (MVC) haciendo que el desarrollo fluya sin problemas.
- Ayuda a evitar el trabajo pesado innecesario, por lo que mantiene el desarrollo de la aplicación web muy organizado.
- Frameworks basados en código abierto y con el respaldoado por los apoyos de su comunidad.

CORS

¿De qué se trata?



- El Intercambio de Recursos de Origen Cruzado, CORS, es un mecanismo para permitir o restringir los recursos solicitados en un servidor web dependiendo de dónde se inició la solicitud HTTP.
- Esto se utiliza para proteger un determinado servidor web del acceso de otro sitio web o dominio. Por ejemplo, solo los dominios permitidos podrán acceder a los archivos alojados en un servidor, como una hoja de estilo, una imagen o un script.
- Por razones de seguridad, los navegadores restringen las solicitudes HTTP de origen cruzado iniciadas dentro de un script.

¿Cómo se utiliza?



- Por ejemplo, si nos encontramos en **http://example.com/page1** y estamos haciendo referencia a una imagen de **http://image.com/myimage.jpg**, no podremos recuperar esa imagen a menos que `http://image.com` permita compartir orígenes cruzados con `http://example.com`.
- Hay un encabezado HTTP llamado *origin* en cada solicitud HTTP el cual define desde dónde se originó la solicitud de dominio. Podemos usar la información del encabezado para restringir o permitir que los recursos de nuestro servidor web los protejan.

Configurando CORS



- Npm tiene un módulo llamado CORS, para poder configurar fácilmente las cabeceras, y decidir si permitimos o no el acceso a ciertas solicitudes de dominio cruzado.
- En primer lugar, instalamos el módulo con el comando:
- Luego, lo requerimos en el archivo *server.js*.

```
$ npm install cors
```

```
var express = require('express')  
var cors = require('cors')  
var app = express()
```

Configurando CORS

Ejemplo
en vivo



- Si deseamos habilitar CORS para todas las solicitudes, simplemente podemos usar el middleware cors antes de configurar las rutas, configurándolo a nivel global:

```
const express = require('express');
const cors = require('cors');

const app = express();

app.use(cors())

.....
```

- ★ Esto nos permitirá acceder a todas las rutas desde cualquier lugar de la web si eso es lo que necesitamos. Entonces, las rutas que configuremos serán accesibles para todos los dominios.

Configurando CORS

Ejemplo
en vivo



- Si necesitamos que una determinada ruta sea accesible y no otras rutas, podemos configurar cors en una determinada ruta como middleware en lugar de configurarlo para toda la aplicación:

```
app.get('/', cors(), (req, res) => {  
  res.json({  
    message: 'Hello World'  
  });  
});
```

- Esto permitirá que una determinada ruta sea accesible por cualquier dominio. Entonces, en este caso, solo la ruta “/” será accesible para cada dominio. Las demás rutas solo serán accesibles para las solicitudes que se iniciaron en el mismo dominio que la API en la que estén definidas.

Configurando CORS con Options

Ejemplo
en vivo



- Podemos usar las opciones de configuración con CORS para personalizar ésto aún más.
- Podemos usar la configuración para permitir el acceso de un solo dominio o subdominios, configurar métodos HTTP que estén permitidos, como GET y POST, según nuestros requisitos.
- Así es como podemos permitir el acceso de un solo dominio usando las opciones de CORS:

```
var corsOptions = {  
  origin: 'http://localhost:8080',  
  optionsSuccessStatus: 200 // For Legacy browser support  
}  
  
app.use(cors(corsOptions));
```


Configurando CORS con Options

Ejemplo
en vivo



- También podemos configurar los métodos HTTP que estén permitidos:

```
var corsOptions = {  
  origin: 'http://localhost:8080',  
  optionsSuccessStatus: 200 // For legacy browser support  
  methods: "GET, PUT"  
}  
  
app.use(cors(corsOptions));
```

- Si enviamos una solicitud POST desde `http://localhost: 8080`, el navegador la bloqueará, ya que solo se admiten GET y PUT según los métodos especificados en esta configuración.

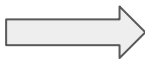
Configurando CORS dinámico con Function

Ejemplo
en vivo



- Si las configuraciones no satisfacen nuestros requisitos, podemos crear una función personalizada para CORS.

- Por ejemplo, supongamos que deseamos permitir el uso compartido de CORS para archivos .jpg http://something.com y http://example.com.



```
const allowlist = ['http://something.com', 'http://example.com'];

const corsOptionsDelegate = (req, callback) => {
  let corsOptions;

  let isDomainAllowed = whitelist.indexOf(req.header('Origin')) !== -1;
  let isExtensionAllowed = req.path.endsWith('.jpg');

  if (isDomainAllowed && isExtensionAllowed) {
    // Enable CORS for this request
    corsOptions = { origin: true }
  } else {
    // Disable CORS for this request
    corsOptions = { origin: false }
  }
  callback(null, corsOptions)
}

app.use(cors(corsOptionsDelegate));
```

Configurando CORS dinámico con Function

Ejemplo
en vivo



- El *callback* acepta dos parámetros:
El primero es un error donde pasamos **null** y el segundo son opciones donde pasamos **{origin: false}**.
- Por lo tanto, una aplicación web alojada en `http://something.com` o `http://example.com` podría hacer referencia a una imagen con la extensión `.jpg` desde el servidor, como hemos configurado en nuestra función personalizada.

```
const allowlist = ['http://something.com', 'http://example.com'];

const corsOptionsDelegate = (req, callback) => {
  let corsOptions;

  let isDomainAllowed = whitelist.indexOf(req.header('Origin')) !== -1;
  let isExtensionAllowed = req.path.endsWith('.jpg');

  if (isDomainAllowed && isExtensionAllowed) {
    // Enable CORS for this request
    corsOptions = { origin: true }
  } else {
    // Disable CORS for this request
    corsOptions = { origin: false }
  }
  callback(null, corsOptions)
}

app.use(cors(corsOptionsDelegate));
```

APLICACIÓN COMPLETA

Ejemplo de aplicación completa



- A continuación, vamos a ver un ejemplo de una aplicación completa, hecha con el stack MERN. Tenemos el lado del servidor como API RESTful y por separado el lado del cliente, que consume esa API. La aplicación está configurada en capas como vimos las clases anteriores.
- La misma, es una aplicación para leer noticias, generar las noticias, marcarlas como leídas, obtener las noticias y borrarlas.
- Empezamos creando un proyecto de Node con Express. Instalamos además el módulo de cors, mongodb y dotenv.
- Por otro lado, creamos un proyecto de React para el lado del cliente.

LADO SERVIDOR: API RESTful

Configuración API RESTful

Ejemplo
en vivo



JS server.js X

server > JS server.js > ...

```
1 import config from './config.js';
2 import express from 'express'
3 import cors from 'cors'
4 import RouterNoticias from './router/noticias.js'
5
6 const app = express()
7
8 if(config.NODE_ENV == 'development') app.use(cors())
9
10 app.use(express.static('public'))
11 app.use(express.json())
12
13 const routerNoticias = new RouterNoticias()
14
15 /* ----- */
16 /*           ZONA DE RUTAS MANEJADAS POR EL ROUTER           */
17 /* ----- */
18 app.use('/noticias', routerNoticias.start())
```

- Comenzamos con el archivo **server.js** en el cual configuramos el servidor y todo lo básico que tendrá nuestra aplicación. Tendremos Express, Cors, y las rutas para las noticias.

```
20 /* ----- */
21 /*           Servidor LISTEN           */
22 /* ----- */
23 const PORT = config.PORT || 8000
24 const server = app.listen(PORT,
25   () => console.log(
26     `Servidor express escuchando en el puerto ${PORT} (${config.NODE_ENV} - ${config.TIPO_PERSISTENCIA})`
27   ))
28 server.on('error', error => console.log('Servidor express en error:', error))
```

CODER HOUSE

Configuración API RESTful

Ejemplo
en vivo



JS config.js X

server > JS config.js > ...

```
1 // config.js
2 import dotenv from 'dotenv';
3 import path from 'path';
4
5 dotenv.config({
6   path: path.resolve(process.cwd(), process.env.NODE_ENV + '.env')
7 });
8
9 export default {
10   NODE_ENV: process.env.NODE_ENV || 'development',
11   HOST: process.env.HOST || 'localhost',
12   PORT: process.env.PORT || 8080,
13   //MEM - FILE - MONGO
14   TIPO_PERSISTENCIA: process.env.TIPO_PERSISTENCIA || 'MEM'
15 }
```

- Tenemos el archivo **config.js** que lo requerimos en el *server*, donde configuramos las variables de entorno con el módulo *dotenv*.
- Las variables de entorno que tenemos son: `NODE_ENV`, `HOST`, `PORT` y `TIPO_PERSISTENCIA`.

Configuración API RESTful

Ejemplo
en vivo



```
development.env X
server > development.env
1  NODE_ENV=development
2  HOST=localhost
3  PORT=8080
4  //MEM - FILE - MONGO
5  TIPO_PERSISTENCIA=FILE
```

```
production.env X
server > production.env
1  NODE_ENV=production
2  HOST=localhost
3  PORT=9000
4  //MEM - FILE - MONGO
5  TIPO_PERSISTENCIA=MONGO
```

- Tenemos los archivos donde definimos las variables de entorno.
- Vamos a tener dos ambientes: desarrollo (**development**) y producción (**production**). Para cada una de estas, tenemos los valores de las variables de entorno que se muestran.

RUTAS

Rutas

Ejemplo
en vivo



JS noticias.js X

server > router > JS noticias.js > ...

```
1  import express from 'express'
2  const router = express.Router()
3
4  import ControladorNoticias from '../controlador/noticias.js'
5
6  class RouterNoticias {
7
8      constructor() {
9          this.controladorNoticias = new ControladorNoticias()
10     }
11
12     start() {
13         router.get('/:id?', this.controladorNoticias.obtenerNoticias)
14         router.post('/', this.controladorNoticias.guardarNoticia)
15         router.put('/:id', this.controladorNoticias.actualizarNoticia)
16         router.delete('/:id', this.controladorNoticias.borrarNoticia)
17
18         return router
19     }
20 }
21
22 export default RouterNoticias
```

- En la carpeta de **routes**, tenemos el archivo **noticias.js** con la clase *RouterNoticias* que está instanciada en el *server.js*.
- Dentro de esta clase, definimos el método **start()** el cual contiene la definición de las rutas por GET, POST, PUT y DELETE para las noticias que ejecutan los métodos del archivo de controlador.

CONTROLADOR

Controlador

Ejemplo
en vivo



```
JS noticias.js X
server > controlador > JS noticias.js > ...
1  import ApiNoticias from '../api/noticias.js'
2
3  class ControladorNoticias {
4
5      constructor() {
6          this.apiNoticias = new ApiNoticias()
7      }
8
9      obtenerNoticias = async (req,res) => {
10         try {
11             let id = req.params.id
12             //console.log(id)
13             let Noticias = await this.apiNoticias.obtenerNoticias(id)
14
15             res.send(Noticias)
16         }
17         catch(error) {
18             console.log('error obtenerNoticias', error)
19         }
20     }
21
22     guardarNoticia = async (req,res) => {
23         try {
24             let Noticia = req.body
25             //console.log(Noticia)
26             let NoticiaGuardada = await this.apiNoticias.guardarNoticia(Noticia)
27
28             res.json(NoticiaGuardada)
29         }
30         catch(error) {
31             console.log('error obtenerNoticias', error)
32         }
33     }
34 }
```

- Luego, en la carpeta **controlador** tenemos el archivo **noticias.js** con la clase **ControladorNoticias**, instanciada en el archivo de rutas.
- En esta clase, definimos los métodos de controlador para las rutas definidas.
- El método **obtenerNoticias()** lista las noticias guardadas.
- El método **guardarNoticia()** guarda una nueva noticia creada.

Controlador

Ejemplo
en vivo



```
35 actualizarNoticia = async (req,res) => {
36   try {
37     let Noticia = req.body
38     let id = req.params.id
39     //console.log(Noticia)
40     let NoticiaActualizada = await this.apiNoticias.actualizarNoticia(id,Noticia)
41     res.json(NoticiaActualizada)
42   }
43   catch(error) {
44     console.log('error obtenerNoticias', error)
45   }
46 }
47
48 borrarNoticia = async (req,res) => {
49   try {
50     let id = req.params.id
51     let NoticiaBorrada = await this.apiNoticias.borrarNoticia(id)
52     res.json(NoticiaBorrada)
53   }
54   catch(error) {
55     console.log('error obtenerNoticias', error)
56   }
57 }
58 }
59
60 export default ControladorNoticias
```

- El método `actualizarNoticia()` guarda los cambios realizados sobre una noticia.
- El método `borrarNoticia()` borra una noticia por su id.
- Todos estos métodos, usan métodos del archivo de Api.

LÓGICA DE NEGOCIO

Lógica de negocio

Ejemplo
en vivo



```
JS noticias.js X
server > api > JS noticias.js > ...
1 import config from '../config.js';
2 import NoticiasFactoryDAO from '../model/DAOs/noticiasFactory.js'
3 import Noticias from '../model/models/noticias.js';
4
5 class ApiNoticias {
6
7   constructor() {
8     this.noticiasDAO = NoticiasFactoryDAO.get(config.TIPO_PERSISTENCIA)
9   }
10
11   async obtenerNoticias(id) { return await this.noticiasDAO.obtenerNoticias(id) }
12
13   async guardarNoticia(noticia) {
14     ApiNoticias.asegurarNoticiaValida(noticia,true)
15     return await this.noticiasDAO.guardarNoticia(noticia)
16   }
17
18   async actualizarNoticia(id,noticia) {
19     ApiNoticias.asegurarNoticiaValida(noticia,false)
20     return await this.noticiasDAO.actualizarNoticia(id,noticia)
21   }
22
23   async borrarNoticia(id) { return await this.noticiasDAO.borrarNoticia(id) }
24
25   static asegurarNoticiaValida(noticia,requerido) {
26     try {
27       Noticias.validar(noticia,requerido)
28     } catch (error) {
29       throw new Error('la noticia posee un formato json invalido o faltan datos: ' + error.details[0].message)
30     }
31   }
32 }
33
34 export default ApiNoticias
```

- La lógica de negocio la tenemos en la carpeta **api**, en el archivo **noticias.js**.
- En este archivo, se realizan las peticiones a la capa de persistencia.
- Están definidos los distintos métodos. Estos, piden la información al DAO, según el tipo de persistencia definido por el entorno de ejecución.

→ Nota: el método asegurarNoticiaValida lo veremos más adelante.

Capa de persistencia

Ejemplo
en vivo



- Vamos entonces a la carpeta **model**. En esta, tenemos la carpeta **DAOs**, la cual tiene el archivo **noticiasFactory.js**.
- En este, tenemos la clase NoticiasFactoryDAO, instanciada en el archivo anterior.
- En esta clase, tenemos el método **get()** que instancia los modelos según el tipo de persistencia.

```
JS noticiasFactory.js X
server > model > DAOs > JS noticiasFactory.js > ...
1  import NoticiasMemDAO from './noticiasMem.js'
2  import NoticiasFileDAO from './noticiasFile.js'
3  import NoticiasDBMongo from './noticiasDBMongo.js'
4
5  class NoticiasFactoryDAO {
6      static get(tipo) {
7          switch(tipo) {
8              case 'MEM': return new NoticiasMemDAO()
9              case 'FILE': return new NoticiasFileDAO(process.cwd() + '/noticias.json')
10             case 'MONGO': return new NoticiasDBMongo('mibase','noticias')
11             default: return new NoticiasMemDAO
12         }
13     }
14 }
15
16 export default NoticiasFactoryDAO
```

CAPA DE PERSISTENCIA

Capa de persistencia

Ejemplo
en vivo



- Dentro de esta misma carpeta, en el archivo **noticias.js**, tenemos la clase **NoticiasBaseDAO**.
- Esta clase, la van a heredar las clases de cada uno de los tipos de persistencia que definimos. Por lo tanto, tendrán esto básico en común.

```
JS noticias.js X
server > model > DAOs > JS noticias.js > ...
1  class NoticiasBaseDAO {
2
3      getNext_Id(noticias) {
4          let lg = noticias.length
5          return lg? parseInt(noticias[lg-1]._id) + 1 : 1
6      }
7
8      getIndex(_id,noticias) {
9          return noticias.findIndex(noticia => noticia? noticia._id == _id: true)
10     }
11 }
12
13 export default NoticiasBaseDAO
```

Capa de persistencia - En memoria

Ejemplo
en vivo



```
JS noticiasMem.js X
server > model > DAOs > JS noticiasMem.js > ...
1 import noticiaDTO from '../DTOs/noticias.js'
2 import NoticiasBaseDAO from './noticias.js'
3
4 class NoticiasMemFileDAO extends NoticiasBaseDAO {
5
6   constructor() {
7     super()
8     this.noticias = []
9   }
10
11   obtenerNoticias = async _id => {
12     try {
13       if(_id) {
14         let index = this.noticias.findIndex(noticia => noticia._id == _id)
15         return index >= 0 ? [this.noticias[index]] : []
16       }
17       else {
18         return this.noticias
19       }
20     }
21     catch(error) {
22       console.log('error en obtenerNoticias', error)
23       return []
24     }
25   }
26
27   guardarNoticia = async noticia => {
28     try {
29       let _id = this.getNext_Id(this.noticias)
30       let fyh = new Date().toLocaleString()
31       let noticiaGuardada = noticiaDTO(noticia._id, fyh)
32       this.noticias.push(noticiaGuardada)
33
34       return noticiaGuardada
35     }
36     catch(error) {
37       console.log('error en guardarNoticia:', error)
38       let noticia = {}
39
40       return noticia
41     }
42   }
43 }
```

```
44
45 actualizarNoticia = async (_id, noticia) => {
46   try {
47     let fyh = new Date().toLocaleString()
48     let noticiaNew = noticiaDTO(noticia._id, fyh)
49
50     let indice = this.getIndex(_id, this.noticias)
51     let noticiaActual = this.noticias[indice] || {}
52
53     let noticiaActualizada = {...noticiaActual, ...noticiaNew}
54
55     indice >= 0 ?
56       this.noticias.splice(indice, 1, noticiaActualizada) :
57       this.noticias.push(noticiaActualizada)
58
59     return noticiaActualizada
60   }
61   catch(error) {
62     console.log('error en actualizarNoticia:', error)
63     let noticia = {}
64
65     return noticia
66   }
67 }
68
69 borrarNoticia = async _id => {
70   try {
71     let indice = this.getIndex(_id, this.noticias)
72     let noticiaBorrada = this.noticias.splice(indice, 1)[0]
73
74     return noticiaBorrada
75   }
76   catch(error) {
77     console.log('error en borrarNoticia:', error)
78     let noticia = {}
79
80     return noticia
81   }
82 }
83
84 export default NoticiasMemFileDAO
```

- Si el tipo de persistencia es “MEM” entonces, llamamos al archivo **noticiasMem.js** e instanciamos la clase *NoticiasMemDAO* como extensión de la clase de base.
- En esta clase, están los métodos para persistir las noticias en memoria.

Capa de persistencia - En memoria

Ejemplo
en vivo



```
guardarNoticia = async noticia => {  
  try {  
    let _id = this.getNext_Id(this.noticias)  
    let fyh = new Date().toLocaleString()  
    let noticiaGuardada = noticiaDTO(noticia,_id,fyh)  
    this.noticias.push(noticiaGuardada)  
  
    return noticiaGuardada  
  }  
}
```

- Al persistir en memoria, no tenemos un id ni fecha y hora de creación automáticas como pasa con Mongo por ejemplo.
- Entonces, en el método que vimos de `guardarNoticia()` de la clase `NoticiasMemDAO`, tenemos que crear un id y un objeto de fecha y hora.
- Con ésto, tenemos en la carpeta **DTOs**, archivo **noticias.js**, una función que agrega al objeto de la nueva noticia el id y la fecha y hora de creación de la misma.

```
JS noticias.js X  
server > model > DTOs > JS noticias.js > ...  
1  function noticiaDTO(noticia,_id,fyh) {  
2    return {  
3      ...noticia,  
4      _id,  
5      fyh  
6    }  
7  }  
8  
9  export default noticiaDTO
```

Capa de persistencia - En archivo



- Si el tipo de persistencia es “FILE” llamamos al archivo ***noticiasFile.js*** el cual tiene la clase *NoticiasFileDAO* como extensión de la clase base.
- En éste, debemos especificarle el nombre y ubicación del archivo en cuál vamos a persistir la información (lo hacemos al instanciar la clase en el archivo *noticiasFactory.js*).
- Tenemos entonces en esta clase todos los métodos para persistir las noticias en un archivo.
- Vemos el código de como nos quedan los distintos métodos en la siguiente diapositiva.

Capa de persistencia - En archivo

Ejemplo
en vivo



1.

```
JS noticiasFileJs X
server > model > DAOs > JS noticiasFileJs > ...
1 import fs from 'fs'
2 import noticiaDTO from '../DTOs/noticias.js'
3 import NoticiasBaseDAO from './noticias.js'
4
5 class NoticiasFileDAO extends NoticiasBaseDAO {
6
7   constructor(nombreArchivo) {
8     super()
9     this.nombreArchivo = nombreArchivo
10   }
11
12   async leer(archivo) {
13     return JSON.parse(await fs.promises.readFile(archivo, 'utf-8'))
14   }
15
16   async guardar(archivo, noticias) {
17     await fs.promises.writeFile(archivo, JSON.stringify(noticias, null, '\t'))
18   }
19
20   obtenerNoticias = async _id => {
21     try {
22       if(_id) {
23         let noticias = await this.leer(this.nombreArchivo)
24         let index = noticias.findIndex(noticia => noticia._id == _id)
25
26         return index >= 0 ? [noticias[index]] : []
27       } else {
28         let noticias = await this.leer(this.nombreArchivo)
29         return noticias
30       }
31     } catch(error) {
32       console.log('error en obtenerNoticias:', error)
33       let noticias = []
34
35       //guardo archivo
36       await this.guardar(this.nombreArchivo, noticias)
37       return noticias
38     }
39   }
40 }
41
```

2.

```
43 guardarNoticia = async noticia => {
44   try {
45     //leo archivo
46     let noticias = await this.leer(this.nombreArchivo)
47
48     let _id = this.getNext_Id(noticias)
49     let fyh = new Date().toLocaleString()
50     let noticiaGuardada = noticiaDTO(noticia, _id, fyh)
51     noticias.push(noticiaGuardada)
52
53     //guardo archivo
54     await this.guardar(this.nombreArchivo, noticias)
55
56     return noticiaGuardada
57   } catch(error) {
58     console.log('error en guardarNoticia:', error)
59     let noticia = {}
60
61     return noticia
62   }
63 }
64
```

CODER HOUSE

Capa de persistencia - En archivo

Ejemplo
en vivo



3.

```
66 actualizarNoticia = async (_id, noticia) => {
67   try {
68     //leo archivo
69     let noticias = await this.leer(this.nombreArchivo)
70
71     let fyh = new Date().toLocaleString()
72     let noticiaNew = noticiaDTO(noticia, _id, fyh)
73
74     let indice = this.getIndex(_id, noticias)
75     let noticiaActual = noticias[indice] || {}
76
77     let noticiaActualizada = {...noticiaActual, ...noticiaNew}
78
79     indice >= 0 ?
80       noticias.splice(indice, 1, noticiaActualizada) :
81       noticias.push(noticiaActualizada)
82
83     //guardo archivo
84     await this.guardar(this.nombreArchivo, noticias)
85
86     return noticiaActualizada
87   }
88   catch(error) {
89     console.log('error en actualizarNoticia:', error)
90     let noticia = {}
91
92     return noticia
93   }
94 }
95 }
```

4.

```
97 borrarNoticia = async _id => {
98   try {
99     //leo archivo
100    let noticias = await this.leer(this.nombreArchivo)
101
102    let indice = this.getIndex(_id, noticias)
103    let noticiaBorrada = noticias.splice(indice, 1)[0]
104
105    //guardo archivo
106    await this.guardar(this.nombreArchivo, noticias)
107
108    return noticiaBorrada
109  }
110  catch(error) {
111    console.log('error en borrarNoticia:', error)
112    let noticia = {}
113
114    return noticia
115  }
116 }
117 }
118
119 export default NoticiasFileDAO
```

CODER HOUSE

Capa de persistencia - En MongoDB

Ejemplo
en vivo



JS noticiasDBMongo.js X

server > model > DAOs > JS noticiasDBMongo.js > ...

```
1  import noticiaDTO from '../DTOs/noticias.js'
2  import NoticiasBaseDAO from './noticias.js'
3
4  import mongodb from 'mongodb';
5  const { MongoClient,ObjectId } = mongodb;
6
7  class NoticiasDBMongoDAO extends NoticiasBaseDAO {
8
9      constructor(database, collection) {
10         super()
11         ;( async () => {
12             console.log('Conectando a la Base de datos...')
13             /* ----- */
14             /*           Conexión a la base de datos warriors           */
15             /* ----- */
16             // connecting at mongoClient
17             const connection = await MongoClient.connect('mongodb://localhost',{
18                 useNewUrlParser: true,
19                 useUnifiedTopology: true
20             });
21             const db = connection.db(database);
22             this._collection = db.collection(collection);
23             /* ----- */
24             console.log('Base de datos conectada')
25         })()
26     }
```

- Si el tipo de persistencia es “MONGO” llamamos al archivo **noticiasDBMongo.js** el cual tiene la clase *NoticiasDBMongoDAO* como extensión de la clase base.
- Al instanciar la clase en el archivo *noticiasFactory.js* debemos definirle el nombre de la base de datos a la cual se va a conectar y el nombre de la colección.
- Usamos *MongoClient* y en primer lugar creamos la conexión a la base de datos.

Capa de persistencia - En MongoDB

Ejemplo
en vivo



```
28 obtenerNoticias = async _id => {
29   try {
30     if(_id) {
31       console.log(_id)
32       const noticia = await this._collection.findOne({_id: ObjectId(_id)})
33       console.log(noticia)
34       return [noticia]
35     }
36     else {
37       const noticias = await this._collection.find({}).toArray()
38       return noticias;
39     }
40   }
41   catch(error) {
42     console.log('obtenerNoticias error', error)
43   }
44 }
45
46 guardarNoticia = async noticia => {
47   try{
48     await this._collection.insertOne(noticia);
49     return noticia
50   }
51   catch(error) {
52     console.log('guardarNoticia error', error)
53     return noticia
54   }
55 }
56
57
58 actualizarNoticia = async (_id, noticia) => {
59   try {
60     await this._collection.updateOne({_id: ObjectId(_id)}, {$set: noticia});
61     return noticia
62   }
63   catch(error) {
64     console.log('actualizarNoticia error', error)
65     return noticia
66   }
67 }
```

- Luego, tenemos todos los métodos para persistir las noticias en MongoDB.
- En el método de `borrarNoticia()`, antes de borrarla, usamos su id en el archivo `noticiasDTO`, para luego poder devolver como retorno de este método un objeto con el id de la noticia eliminada.

```
69 borrarNoticia = async _id => {
70   let noticiaBorrada = noticiaDTO({}, _id, null)
71   try {
72     await this._collection.deleteOne({_id: ObjectId(_id)})
73     return noticiaBorrada
74   }
75   catch(error) {
76     console.log('borrarNoticia error', error)
77     return noticiaBorrada
78   }
79 }
80
81
82 export default NoticiasDBMongoDAO
```

CODER HOUSE

Capa de persistencia - En MongoDB

Ejemplo
en vivo



- Para validar que los campos requeridos no lleguen vacíos, en el archivo de lógica de negocio, en la carpeta api, tenemos en los métodos de guardar y actualizar noticia el llamado a un método de validación.

```
async guardarNoticia(noticia) {  
  ApiNoticias.asegurarNoticiaValida(noticia,true)  
  return await this.noticiasDAO.guardarNoticia(noticia)  
}  
  
async actualizarNoticia(id,noticia) {  
  ApiNoticias.asegurarNoticiaValida(noticia,false)  
  return await this.noticiasDAO.actualizarNoticia(id,noticia)  
}
```

Capa de persistencia - Validación

Ejemplo
en vivo



- En ese mismo archivo, tenemos definido este método para validar:

```
static asegurarNoticiaValida(noticia,requerido) {  
  try {  
    Noticias.validar(noticia,requerido)  
  } catch (error) {  
    throw new Error('la noticia posee un formato json invalido o faltan datos: ' + error.details[0].message)  
  }  
}
```

- Vemos en este método, que como parámetro pasamos la noticia a validar y si es requerido o no.
- Este método llama al modelo *Noticias*, el método *validar* y le pasa estos dos parámetros. Esta clase está definida en el archivo que veremos a continuación.

Capa de persistencia - Validación



- Siguiendo dentro de la carpeta **model**, tenemos la carpeta **models**, y dentro el archivo **noticias.js**.
- En este archivo, tenemos la clase Noticias que recién mencionamos, con el método **validar()**. Este método chequea si los campos requeridos tiene información al guardar o actualizar una noticia.
- Si todos los campos están correctos, entonces valida la noticia. Si alguno llega vacío, entonces da error. Utiliza el módulo de npm **Joi** para validar.
- En la siguiente diapositiva vemos el código de este archivo.

Capa de persistencia - Validación

Ejemplo
en vivo



```
JS noticias.js X
server > model > models > JS noticias.js > ...
1  import Joi from 'joi'
2
3  class Noticias {
4
5      constructor(titulo, cuerpo, autor, imagen, email, vista) {
6          this.titulo = titulo
7          this.cuerpo = cuerpo
8          this.autor = autor
9          this.imagen = imagen
10         this.email = email
11         this.vista = vista
12     }
13
14     equals(otroNoticias) {
15         if (!(otroNoticias instanceof Noticias)) {
16             return false
17         }
18         if (this.titulo !== otroNoticias.titulo) {
19             return false
20         }
21         if (this.cuerpo !== otroNoticias.cuerpo) {
22             return false
23         }
24         if (this.autor !== otroNoticias.autor) {
25             return false
26         }
27         if (this.imagen !== otroNoticias.imagen) {
28             return false
29         }
30         if (this.email !== otroNoticias.email) {
31             return false
32         }
33         if (this.vista !== otroNoticias.vista) {
34             return false
35         }
36         return true
37     }
38 }
```

```
39     static validar(noticia, requerido) {
40         //console.log(noticia, requerido)
41         const NoticiaSchema = Joi.object({
42             titulo: requerido? Joi.string().required() : Joi.string(),
43             cuerpo: requerido? Joi.string().required() : Joi.string(),
44             autor: requerido? Joi.string().required() : Joi.string(),
45             imagen: requerido? Joi.string().required() : Joi.string(),
46             email: requerido? Joi.string().required() : Joi.string(),
47             vista: requerido? Joi.boolean().required() : Joi.boolean()
48         })
49
50         const { error } = NoticiaSchema.validate(noticia)
51         if (error) {
52             throw error
53         }
54     }
55 }
56
57 export default Noticias
```

Package.JSON e inicio del servidor

Ejemplo
en vivo



```
package.json X
server > npm package.json > ...
1 {
2   "type": "module",
3   "name": "backend-noticias",
4   "version": "1.0.0",
5   "description": "",
6   "main": "server.js",
7   "scripts": {
8     "test": "echo \\\"Error: no test specified\\\" && exit 1",
9     "start": "node server.js",
10    "watch": "nodemon server.js --ignore noticias.json",
11    "dev": "set NODE_ENV=development&& node server.js",
12    "prod": "set NODE_ENV=production&& node server.js"
13  },
14  "keywords": [],
15  "author": "",
16  "license": "MIT",
17  "dependencies": {
18    "cors": "^2.8.5",
19    "dotenv": "^10.0.0",
20    "express": "^4.17.1",
21    "joi": "^17.4.0",
22    "mongodb": "^3.6.9"
23  }
24 }
```

- Vemos que en los scripts tenemos definidas diferentes formas de iniciar el servidor.
- Con los comandos `npm run dev` y `npm run prod`, lo iniciamos con el ambiente de desarrollo y producción respectivamente.
- Con `npm start`, lo iniciamos con las variables que especificamos por default (persistencia en memoria).
- Con `npm run watch`, iniciamos en modo watch, con nodemon. En este caso ignoramos el archivo `noticias.json` que es en el que persiste el tipo "FILE" porque sino, al ser nodemon, va a reiniciar el servidor cada vez que se modifique este archivo.



BREAK

¡5/10 MINUTOS Y VOLVEMOS!



SERVIDOR MVC COMPLETO

Tiempo: 15 a 20 minutos

SERVIDOR MVC COMPLETO

Desafío
generico



Tiempo: 15 a 20 minutos

Realizar un esqueleto de servidor MVC basado en Node.js y express.

Este debe tener separado en capas, donde se encuentren carpetas para resolver:

- La capa de ruteo
- El controlador
- La lógica de negocio
- Las validaciones de nuestros datos
- La capa de persistencia (DAO, DTO)

Realizar una simple ruta get y una post para pedir e incorporar palabras a un array de strings persistidos en memoria, siguiendo la lógica de la separación del proceso en capas.

SERVIDOR MVC COMPLETO

Desafío
generico



Tiempo: 15 a 20 minutos

Cada palabra que ingrese por post se debe almacenar en el array dentro de un objeto que contenga un timestamp. Ej.

```
[  
  { id: 1, palabra: "Hola", timestamp: 1624450180112 },  
  { id: 2, palabra: "que", timestamp: 1624450189685 },  
  { id: 3, palabra: "tal", timestamp: 1624450195068 }  
  ...  
]
```

Con el get se traerá la frase completa en formato string.

Probar la operación con postman.

LADO CLIENTE: PROYECTO EN REACT

Configuración del front-end



- Creamos un proyecto en React para nuestro front-end de la aplicación.
- En éste, vamos a consumir la API que creamos, y crear los componentes que necesitamos para poder mostrar el listado de noticias, crear nuevas, actualizarlas y borrarlas.
- Vamos a usar *Axios* para hacer los llamados a la API RESTful.
- Vamos a usar el módulo *faker* para crear noticias de forma aleatoria.

Configuración del front-end

Ejemplo
en vivo



JS generador.js X

cliente > src > JS generador.js > ...

```
1  import faker from 'faker'
2
3  faker.locale = "es";
4
5  const generarNoticia = () => ({
6    titulo: faker.hacker.phrase(),
7    cuerpo: faker.lorem.paragraph(),
8    autor: faker.name.findName(),
9    imagen: faker.image.avatar(),
10   email: faker.internet.email(),
11   vista: false
12 })
13
14 export default generarNoticia
```

- Dentro de la carpeta src creamos un archivo llamado generador.js.
- En este, y con el módulo faker, creamos una función que genera una noticia aleatoria.
- Definimos para la noticia un *título*, su *cuerpo*, *autor*, una *imagen* y un *email*. Además, si fue marcada como *vista* inicializa siempre en *false*.



Componente Noticia

- Creamos el componente de **Noticia.js**, el cuál creará la tarjeta en que se va a mostrar cada una de las noticias.
- Como props le pasamos la *noticia*, el método *borrar*, el método *marcaLeida* y el *index*, para el número de noticia.

```
JS Noticia.js X
cliente > src > componentes > JS Noticia.js > ...
1 import './Noticia.css'
2
3 function Noticia(props) {
4   let { noticia, marcarLeida, borrar, index } = props
5   return (
6     <div className="Noticia" style={{opacity: noticia.vista? '0.5': '1'}}>
7       <div className="media alert alert-primary my-4">
8         <img src={noticia.imagen} style={{width: '350px', borderRadius: '15px'}} alt={noticia.title} />
9         <div className="media-body ml-4">
10
11           /* ----- */
12           /* Botón de leer */
13           /* ----- */
14           <button className="btn btn-warning float-right" onClick={
15             () => marcarLeida(noticia._id)}><i className="fab fa-readme"> Leida</i></button>
16
17           /* ----- */
18           /* Botón de borrar */
19           /* ----- */
20           <button className="btn btn-danger float-left" onClick={
21             () => borrar(noticia._id)}>
22           <i className="far fa-trash-alt"> Borrar</i></button>
23
24           /* ----- */
25           /* Representación de la noticia */
26           /* ----- */
27           <h3 className="text-center font-italic text-uppercase"><u>Noticia Nro. {index+1}</u></h3>
28           <br />
29           <h3>{noticia.titulo}</h3>
30           <p><i>{noticia.cuerpo}</i></p>
31           <p><b>{noticia.autor}</b></p>
32           <p><b><i>{noticia.email}</i></b></p>
33           <p><b>ID: </b><i>{noticia._id}</i></p>
34         </div>
35       </div>
36     </div>
37   )
38 }
39
40 export default Noticia
```

Componente Noticias

Ejemplo
en vivo



- Luego, creamos el componente **Noticias.js**. Este es un componente de estado y es donde vamos a hacer los llamados a la Api usando Axios.
- Comenzamos importando Axios, el generador de noticias, el componente Noticia.
- Creamos la URL genérica para hacer los llamados. Vimos que la ruta general es `/noticias`. Si el ambiente es producción, la URL es `"/noticias"` ya que envía las rutas como relativas. En cambio, en desarrollo, la URL quedará `"http://localhost:8080/noticias"`.

```
JS Noticias.js X
cliente > src > componentes > JS Noticias.js > ...
1  import React from 'react'
2  import Noticia from './Noticia'
3
4  import axios from 'axios'
5
6  import './Noticias.css'
7  import generarNoticia from '../generador'
8
9  const URL_NOTICIAS = (process.env.NODE_ENV === 'production'? '': 'http://localhost:8080') + '/noticias/'
```


Componente Noticias

Ejemplo
en vivo



```
11 class Noticias extends React.Component {
12
13   state = {
14     noticias : [],
15     idObtenerNoticia : '',
16     pedidas : false
17   }
18
19   /* ----- */
20   /*      GET noticia      */
21   /* ----- */
22   async obtenerNoticias(_id) {
23     try {
24       let response = await axios(URL_NOTICIAS+(_id?_id:'))
25       let { data:noticias } = response
26       this.setState({noticias: noticias? noticias : []})
27     }
28     catch(error) {
29       console.error(error)
30       this.setState({noticias: []})
31     }
32     this.setState({pedidas : true, idObtenerNoticia: ''})
33   }
34 }
```

- Creamos el componente de estado y definimos en *state* las variables *noticias* como un *array*, *idObtenerNoticia* como un *string* y *pedidas* como *boolean* en *false*.
- El primer método que tenemos es el GET de una noticia por id. De forma asíncrona buscamos la noticia con Axios y luego guardamos en la variable de estado *noticias* la noticia que encuentra si es que lo hace.

Componente Noticias

Ejemplo
en vivo



```
/* ----- */
/*      POST noticia      */
/* ----- */
async incorporarNoticia() {
  try {
    let noticia = generarNoticia()

    let response = await axios.post(URL_NOTICIAS, noticia)
    let { data: noticiaIncorporada } = response
    console.log(noticiaIncorporada)

    let noticias = [...this.state.noticias]
    noticias.push(noticiaIncorporada)
    this.setState({noticias})
  }
  catch(error) {
    console.error('incorporarNoticia', error)
  }
}

/* ----- */
/*      UPDATE noticia    */
/* ----- */
async actualizarComoLeida(_id) {
  try {
    let { data: noticia } = await axios.put(URL_NOTICIAS+_id, {vista: true})
    console.log(noticia)
    let noticias = [...this.state.noticias]
    noticias.find(noticia => noticia._id === _id).vista = true
    this.setState({noticias})
  }
  catch(error) {
    console.error(error)
  }
}
```

- El método por POST para crear una nueva noticia. La generamos con el generador que creamos.
- El método por PUT para actualizar una noticia por id marcándola como leída.
- Y finalmente el método por DELETE para eliminar una noticia por id.

```
/* ----- */
/*      DELETE noticia    */
/* ----- */
async borrarNoticia(_id) {
  try {
    let { data: noticia } = await axios.delete(URL_NOTICIAS+_id)
    console.log(noticia)

    let noticias = [...this.state.noticias]
    let index = noticias.findIndex(noticia => noticia._id === _id)

    noticias.splice(index, 1)
    this.setState({noticias})
  }
  catch(error) {
    console.error('borrarNoticia', error)
  }
}
```

CODER HOUSE

Componente Noticias

Ejemplo
en vivo



- Tenemos luego el render en donde creamos los botones para crear nueva noticia y obtener noticias.

```
render() {
  let { noticias,idObtenerNoticia,pedidas } = this.state
  return (
    <div className="Noticias">
      <div className="container mt-3">
        <div className="jumbotron">
          <h1>Mis Noticias - API REST Full</h1>
          <hr />
          {/----- */}
          {/ Obtener noticias */}
          {/----- */}
          <button className="btn btn-info my-3 float-left" onClick={
            () => this.obtenerNoticias(idObtenerNoticia)}>
            <i className="fas fa-file-alt"> Obtener</i></button>

          <input value={idObtenerNoticia} className="mt-3 ml-1 form-control w-25 float-left" onChange={
            e => this.setState({idObtenerNoticia: e.target.value})
          } placeholder="todas ó ingrese ID" type="text" />

          {/----- */}
          {/ Crear noticia */}
          {/----- */}
          <button className="btn btn-success my-3 float-right" onClick={
            () => this.incorporarNoticia()}>
            <i className="fas fa-envelope-open-text"> Generar</i></button>

          <div className="clearfix">

            {/ Cartel de no hay noticias */}
            {
              !noticias.length && pedidas &&
              <h3 className="alert alert-danger">
                No hay noticias para mostrar
              </h3>
            }
          </div>
        </div>
      </div>
    </div>
  )
}
```

- Finalmente vemos la representación de las noticias, con el componente Noticia al que le pasamos las props.

```
{/ Representación de las noticias */}
{
  noticias.map( (noticia,index) => {
    return (
      <Noticia
        noticia={noticia}
        index={index}
        marcarLeida={_id => this.actualizarComoLeida(_id) }
        borrar={_id => this.borrarNoticia(_id) }
        key={noticia._id}
      />
    )
  })
}
</div>
</div>
}

export default Noticias
```

Build - package.json



- Con el comando `npm run build`, creamos en nuestro proyecto de React, una carpeta llamada **build** con los archivos correspondientes para poder preparar nuestro proyecto y luego utilizarlo en otro.
- Debemos copiar los archivos que se generan con este comando, a la carpeta **public** de nuestra API RESTful para conectar así el front y el back.
- Luego de modificar nuestro proyecto de React, una vez que ya queramos subir esos cambios a producción, debemos ejecutar nuevamente el comando para generar la carpeta de *build* e ir a nuestro proyecto de back, borrar los archivos de *public* y pegar los nuevos generados.

Build - package.json

Ejemplo
en vivo



- Para no tener que hacer esto manualmente cada vez que hagamos modificaciones al proyecto, configuramos los scripts del package.json del proyecto de React para hacerlo de forma automática ejecutando un solo comando.

```
"scripts": {  
  "start": "react-scripts start",  
  "build": "react-scripts build",  
  "copy": "xcopy /E /I /Y build ..\\server\\public",  
  "delArchivos": "del /S /Q /F ..\\server\\public\\*.\"",  
  "delCarpetas": "rd /S /Q ..\\server\\public\\",  
  "buildCopy": "npm run build && npm run delArchivos && npm run delCarpetas && npm run copy",  
  "test": "react-scripts test",  
  "eject": "react-scripts eject"  
},
```

Build - package.json



- `npm run delArchivos` borra los archivos de la carpeta public de la API REST.
- `npm run delCarpetas` borras las carpetas de la carpeta public de la API REST.
- `npm run copy` copia las carpetas y archivos de la carpeta build a public en la API REST.
- Con `npm run buildCopy` hacemos los comandos juntos incluyendo `npm run build`.

Levantar la aplicación



- Para levantar la aplicación en **desarrollo** tenemos distintas formas. Por un lado, podemos levantar la aplicación en desarrollo, con el comando `npm run dev` en la API. Como copiamos los archivos del proyecto de React a la carpeta public, si entramos a <http://localhost:8080> podemos usar nuestra aplicación de forma correcta.

Levantar la aplicación



- Por otro lado, también podemos ejecutar el front con `npm start`, teniendo levantado el back con `npm run dev`. En este caso, entramos a <http://localhost:3000> y también usamos nuestra aplicación. Esta forma funciona debido a las CORS que configuramos en nuestra API, ya que de otra forma se hubiera bloqueado el acceso a las rutas desde el origen por las políticas de CORS.
- Para levantar la aplicación en **producción**, usamos el comando `npm run prod` en la consola de la API REST. Ingresamos luego a <http://localhost:9000> (porque pusimos ese puerto en `.env`) y allí podremos usar nuestra aplicación de forma correcta.

Vista de la aplicación



- Con el botón “Obtener” obtenemos todas las noticias que tengamos almacenadas. Si especificamos Id, nos traerá la noticia con ese Id si es que existe.
- Con el botón “Generar” generamos una nueva noticia, de forma aleatoria.

Mis Noticias - API REST Full

Obtener

todas ó ingrese ID

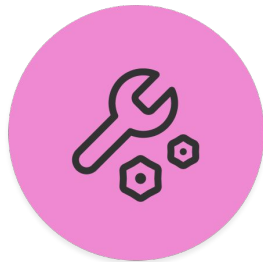
Generar

Vista de la aplicación



- Esta es la tarjeta que se crea con cada noticia que agregamos.
- En cada una tenemos el botón de “Leída” para marcar como leída la noticia.
- Además, tenemos el botón de “Eliminar” para borrarla.



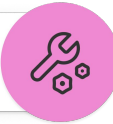


CONSUMIR NUESTRA API REST

Tiempo: 5 minutos

CONSUMIR NUESTRA API REST

Desafío
generico



Tiempo: 5 minutos

- Realizar una sencilla página web front en HTML/JS (send.html) que al ejecutarse dentro del navegador, en un proceso independiente al servidor del desafío anterior (puede estar servida por el live server de visual studio code), le envíe a este por post una palabra al azar.

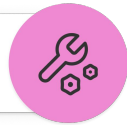
No hace falta realizar la vista, el HTML estará para contener el script de ejecución.

- Utilizar axios en el front para emitir dicho request.

CONSUMIR NUESTRA API REST

Tiempo: 5 minutos

Desafío
generico



- Así mismo, realizaremos otra página web (receive.html) similar a la anterior, que al ejecutar su script interno, genere un request al mismo servidor en su ruta get para obtener la frase completa almacenada, representando por consola o en la vista del documento dicha frase.

Considerar el uso de CORS en el servidor para permitir los request de dominios cruzados.

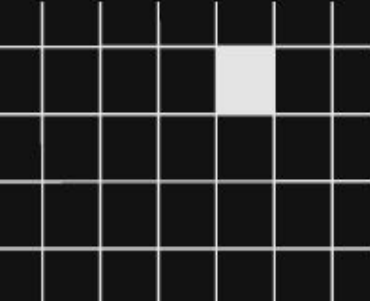
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- MERN stack
 - CORS
 - Aplicación con API RESTful en el lado servidor y un front-end simple en lado cliente.
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE