

Proyecto hospital

Jairo Rastrojo Rodríguez

2º ASIR

Administración de sistemas gestores de bases de datos

1. Introducción

En este documento se verán datos y explicaciones de diversos elementos de la res api, entre ellos su funcionamiento y como se han configurado ciertas cosas para solucionar errores.

La idea es crear con visual studio code una res api, que mediante rutas usando postman podamos trabajar con la base de datos de mongo atlas, y poder trabajar de forma remota con nuestra res api utilizando heroku

En ultima instancia, se utilizara angular para crear una página web operativa con la que poder trabajar en nuestro proyecto desde un entorno visual en la nube.

Tecnologías y programas a usar:

- Visual Studio Code
- Postman
- Mongo Atlas
- Heroku
- Angular
- Stacblitz
- Typescript
- Node.js

2. Res Api Hospital

Vamos a comenzar creando en la carpeta model, los schemas de las clases:

```
src > model > TS empleados.ts > [e] empleSchema > [j] _idiomas > [j] de src > model > TS pacientes.ts > [e] pacienteSchema > [j] _id > [j] type
1 import {Schema, model } from 'mongoose'
2
3 const empleSchema = new Schema({
4   _id:{
5     type: Number,
6     unique: true
7   },
8   _nombre: {
9     type: String
10  },
11  _apellido: {
12    type: String
13  },
14  _contacto: {
15    type: Number,
16    unique: true,
17  },
18  _sueldo: {
19    type: Number
20  },
21  _especialidad: {
22    type: String,
23  },
24  _idiomas: [
25    type: Array,
26    default: 'String'
27  ]
28 })
export const Trabajadores = model('empleados', empleSchema)

1 import {Schema, model } from 'mongoose'
2
3 const pacienteSchema = new Schema({
4   _id: {
5     type: Number,
6     unique: true
7   },
8   _nombre: {
9     type: String
10  },
11  },
12  _apellido1: {
13    type: String
14  },
15  _apellido2: {
16    type: String
17  },
18  _edad: {
19    type: Number
20  },
21  _dni: {
22    type: String,
23    unique: true
24  },
25  _telefono: {
26    type: Number
27  },
28  _medico: {
29    type: String
30  }
31 })
export const Atendidos = model('pacientes', pacienteSchema)
```

Tras crear los schemas, vamos a crear la carpeta DataBase y un archivo con el mismo nombre donde vamos a poner nuestro link de mongo atlas, para poder subir a el todos los datos que queramos

```
src > database > TS database.ts > [e] DataBase > [j] (set) cadenaConexion
1 import mongoose from 'mongoose';
2
3 class DataBase {
4
5   public _cadenaConexion: string = 'mongodb+srv://jairo:1234@cluster0.dynne.mongodb.net/hospital?retryWrites=true&w=majority'
6   constructor(){
7
8   }
9   set cadenaConexion(_cadenaConexion: string){
10     this._cadenaConexion = _cadenaConexion
11   }
12 }
```

Ahora haremos las rutas que se encargarán de llevar a cabo las acciones que queramos hacer.

Nuevo paciente:

```
//Añadir un nuevo paciente
private postpacientes = async (req: Request, res: Response) => {
  const { id, nombre, apellido1, apellido2, edad, dni, telefono, medico, urgencia, tipo, prueba, test } = req.body
  await db.conectarBD()
  const dSchema={
    _id: id,
    _nombre: nombre,
    _apellido1: apellido1,
    _apellido2: apellido2,
    _edad: edad,
    _dni: dni,
    _telefono: telefono,
    _medico: medico,
    _urgencia: urgencia,
    _tipo: tipo,
    _prueba: prueba,
    _test: test
  }
  const oSchema = new Atendidos(dSchema)
  await oSchema.save()
    .then( (doc: any) => res.send(doc))
    .catch( (err: any) => res.send('Error: ' + err))
  await db.desconectarBD()
}
```

Nuevo Empleado:

```
//Añadir un nuevo Empleado
private postempleados = async (req: Request, res: Response) => {
  const { id, nombre, apellido, contacto, puesto, especialidad, idiomas, sueldo } = req.body
  await db.conectarBD()
  const dSchema={
    _id: id,
    _nombre: nombre,
    _apellido: apellido,
    _contacto: contacto,
    _puesto: puesto,
    _especialidad: especialidad,
    _idiomas: idiomas,
    _sueldo: sueldo
  }
  const oSchema = new Trabajadores(dSchema)
  await oSchema.save()
    .then( (doc: any) => res.send(doc))
    .catch( (err: any) => res.send('Error: ' + err))
  await db.desconectarBD()
}
```

Listar Todos los pacientes:

```
//Listar todos los pacientes de la base de datos
private getPacientes = async (req: Request, res: Response) => {
  const promise = new Promise<any>( async (resolve, reject) => {
    await db.conectarBD()
    .then( async () => {
      Atendidos.find({})
      .then( (pacientes) => {
        db.desconectarBD()
        .then( () => resolve(pacientes) )
        .catch( (error) => reject(`Error desconectando de ${db._cadenaConexion}: ${error}`) )
      })
      .catch( (error) => reject(`Error consultando a ${db._cadenaConexion}: ${error}`) )
    })
    .catch( (error) => reject(`Error conectando a ${db._cadenaConexion}: ${error}`) )
  })
  res.json(await promise)
  db.desconectarBD()
}
```

Listar Todos los empleados:

```
//Listar todos los empleados de la BD
private getEmpleados = async (req: Request, res: Response) => {
  await db.conectarBD()
  .then( async ()=> {
    const query = await Trabajadores.aggregate([
      {
        $lookup: {
          from: 'pacientes',
          localField: '_apellido',
          foreignField: '_medico',
          as: "pacientes"
        }
      }
    ])
    res.json(query)
  })
  .catch((mensaje) => {
    res.send(mensaje)
  })
  await db.desconectarBD()
}
```

Buscar paciente:

```
//Buscar un paciente especifico
private getbuspaciente = async (req:Request, res: Response) => {
  const { id } = req.params
  await db.conectarBD()
  .then( async ()=> {
    const pac = await Atendidos.findOne({
      _id: id
    })
    res.json(pac)
  })
  .catch((mensaje) => {
    res.send(mensaje)
  })
  await db.desconectarBD()
}
```

Buscar empleado

```
//Buscar un empleado especifico
private getbusempleado = async(req: Request, res: Response)=>{
  const { apellido } = req.params
  await db.conectarBD()
  .then( async ()=> {
    const emp = await Trabajadores.aggregate([
      {
        $match: {
          "_apellido": apellido
        },
      },{
        $lookup: {
          from: 'pacientes',
          localField: '_apellido',
          foreignField: '_medico',
          as: "pacientes"
        }
      }
    ])
    res.json(emp)
  })
  .catch((mensaje) => {
    res.send(mensaje)
  })
  await db.desconectarBD()
}
```

Actualizar los datos de los pacientes

```
//Actualizar o cambiar los datos de un paciente especifico
private updatepaciente = async (req: Request, res: Response) => {
  const {id} = req.params
  const {nombre, apellido1, apellido2, dni, telefono, medico, tipo, pruebas, test} = req.body
  await db.conectarBD()
  await Atendidos.findOneAndUpdate({
    _id: id
  },{
    _nombre: nombre,
    _apellido1: apellido1,
    _apellido2: apellido2,
    _dni: dni,
    _telefono: telefono,
    _medico: medico,
    _tipo: tipo,
    _prueba: pruebas,
    _test: test
  },{
    new:true,
    runValidators:true
  })
  .then((doc: any) => res.send(doc))
  .catch((err: any) => res.send ('Error: ' + err))
  await db.desconectarBD()
}
```

Actualizar los datos de los empleados

```
//Actualizar o cambiar los datos de un empleado especifico
private updateempleado = async (req: Request, res: Response) => {
  const {id} = req.params
  const {nombre, apellido, contacto, sueldo, idiomas} = req.body
  await db.conectarBD()
  await Trabajadores.findOneAndUpdate({
    _id: id
  },{
    _nombre: nombre,
    _apellido: apellido,
    _contacto: contacto,
    _sueldo: sueldo,
    _idiomas: idiomas,
  },{
    new:true,
    runValidators:true
  })
  .then((doc: any) => res.send(doc))
  .catch((err: any) => res.send ('Error: ' + err))
  await db.desconectarBD()
}
```

Eliminar un paciente:

```
//Eliminar un paciente de la base de datos
private deletepaciente = async (req: Request, res: Response) => {
  const {id} = req.params
  await db.conectarBD()
  await Atendidos.findOneAndDelete(
    {
      _id:id
    }
  )
  .then( (doc: any) => {
    if (doc == null) {
      res.send(`No encontrado`)
    }else {
      res.json({"Borrado": true})
    }
  })
  .catch ((err: any) => res.send('Error: ' + err))
  db.desconectarBD()
}
```

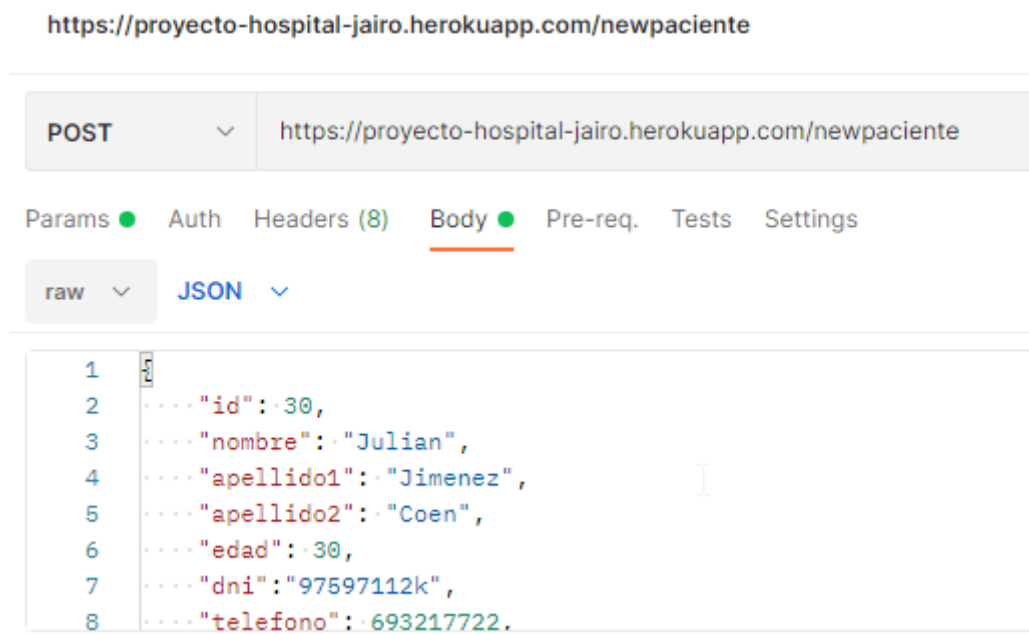
Eliminar un empleado

```
//Eliminar un empleado de la base de datos
private deleteempleado = async (req: Request, res: Response) => {
  const {id} = req.params
  await db.conectarBD()
  await Trabajadores.findOneAndDelete(
    {
      _id:id
    }
  )
  .then( (doc: any) => {
    if (doc == null) {
      res.send(`No encontrado`)
    }else {
      res.json({"Borrado": true})
    }
  })
  .catch ((err: any) => res.send('Error: ' + err))
  db.desconectarBD()
}
```

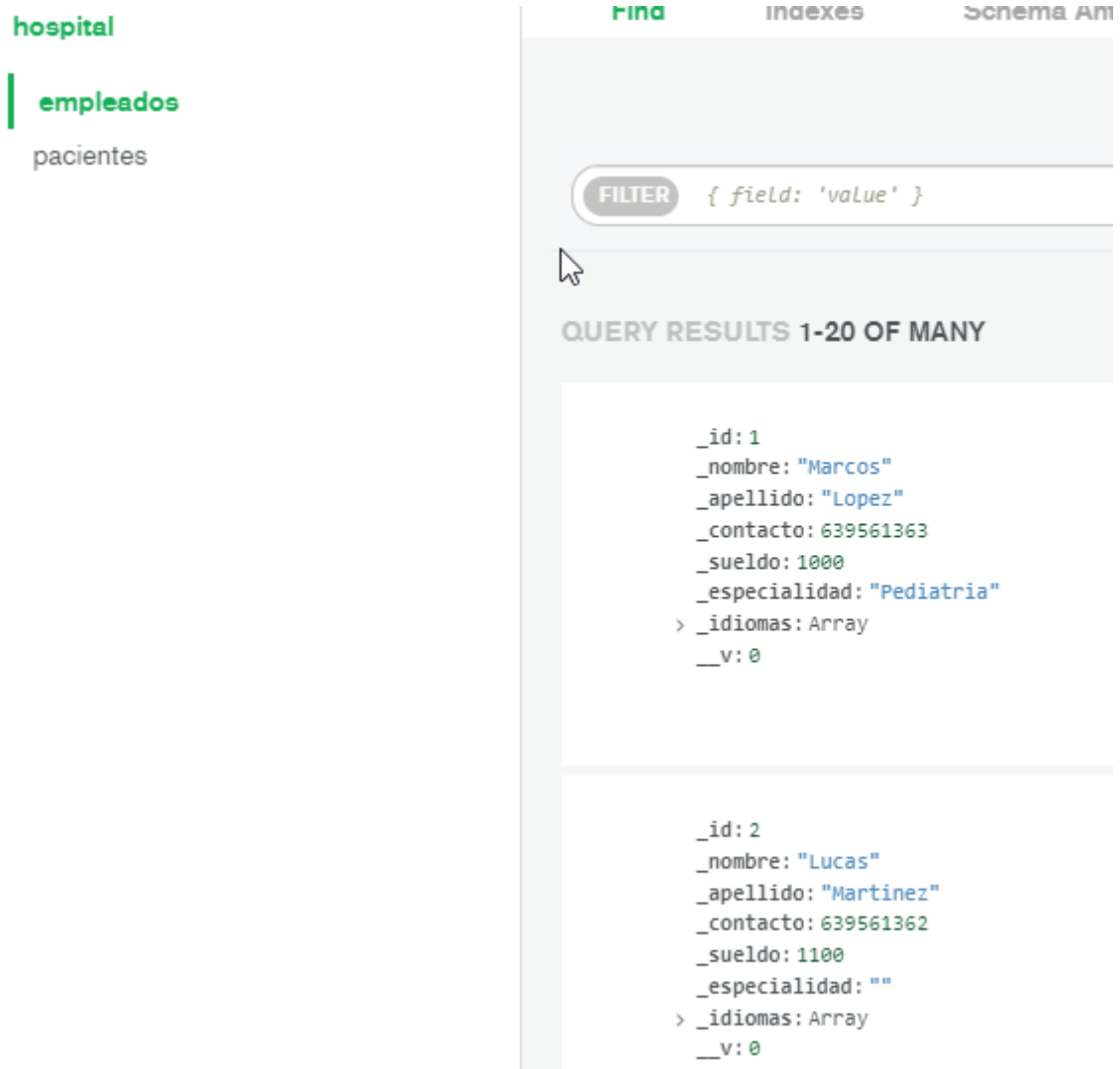

Todas las rutas:

```
    }
    misRutas(){
      this._router.post('/newpaciente', this.postpacientes),
      this._router.post('/newempleado', this.postempleados),
      this._router.get('/verpaciente', this.getPacientes),
      this._router.get('/verempleado', this.getEmpleados),
      this._router.get('/buspaciente/:id', this.getbuspaciente),
      this._router.get('/busempleado/:apellido', this.getbusempleado),
      this._router.put('/actualizarpaciente/:id', this.updatepaciente),
      this._router.put('/actualizarepleado/:id', this.updateempleado),
      this._router.delete('/eliminarpaciente/:id', this.deletepaciente),
      this._router.delete('/eliminarpleado/:id', this.deleteempleado)
    }
  }
  const obj = new Routes()
  obj.misRutas()
  export const routes = obj.router
```

Dichas rutas serán usadas con postman para ser subida a mongo atlas



Y vemos que se han subido a mongo atlas



3. Heroku

Para que podamos seguir debemos subir el proyecto a github:

https://github.com/rastrojorodriguezjairo2/RES_API_Hospital

En el enlace anterior podemos acceder a dicho repositorio

Deberemos vincular nuestro repositorio de github a heroku

GitHubrastrojorodriguezjairo2/RES_API_Hospitalmain

OverviewResourcesDeployMetricsActivityAccessSettings

Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them.
[Learn more.](#)

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests.
[Learn more.](#)

Choose a pipeline

Deployment method

Heroku Git

Use Heroku CLI

GitHub

Connected

Container Registry

Use Heroku CLI

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to rastrojorodriguezjairo2/RES_API_Hospital by rastrojorodriguezjairo2

Disconnect...

Releases in the [activity feed](#) link to GitHub to view commit diffs

Automatically deploys from main

Y desplegaremos nuestra res api

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Disable Automatic Deploys

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more.](#)

Choose a branch to deploy

main

Deploy Branch

Receive code from GitHub

Build main 3a3a6783

Release phase

Deploy to Heroku

✓

✓


✓

✓

Your app was successfully deployed.

View

Como podemos ver, al usar las rutas podemos acceder a los datos alojados en mongo atlas



The screenshot shows a web browser window with the address bar displaying 'proyecto-hospital-jairo.herokuapp.com/verpaciente'. Below the address bar, there are several tabs: 'GamesFull | Juegos...', 'Ver Sekai Seifuku: B...', 'Welcome - CircleCI', and 'Your Repositor...'. The main content area of the browser shows a JSON response from a REST client. The response is an array of two objects, each representing a patient record. The first object has an '_id' of 1, a name 'Berto', and a last name 'Bello'. The second object has an '_id' of 2, a name 'Alfonso', and a last name 'Belizo'. Both objects have an 'edad' (age) and a 'dni' (ID number). The 'telefono' (phone number) is the same for both: 612619716. The 'medico' (doctor) is 'Teclado' for the first and 'Lumici' for the second. The 'urgencia' (emergency) is 'Brecha en la cabeza' for the first and 'Torcedura de tobillo' for the second. The 'tipo' (type) is 'urgencias' for both. The 'prueba' (test) is 'Curas' and 'Resonancia' for the first, and 'Radiografia' for the second. The 'test' field is empty for both, and the '_v' (version) is 0 for both.

```
[
  {
    "_id": 1,
    "_nombre": "Berto",
    "_apellido1": "Bello",
    "_apellido2": "Baeza",
    "_edad": 23,
    "_dni": "90531024d",
    "_telefono": 612619716,
    "_medico": "Teclado",
    "_urgencia": "Brecha en la cabeza",
    "_tipo": "urgencias",
    "_prueba": [
      "Curas",
      "Resonancia"
    ],
    "_test": "",
    "_v": 0
  },
  {
    "_id": 2,
    "_nombre": "Alfonso",
    "_apellido1": "Alitas",
    "_apellido2": "Belizo",
    "_edad": 12,
    "_dni": "90531024d",
    "_telefono": 612619716,
    "_medico": "Lumici",
    "_urgencia": "Torcedura de tobillo",
    "_tipo": "urgencias",
    "_prueba": [
      "Radiografia"
    ],
    "_test": "",
    "_v": 0
  }
]
```

4. Angular

Ahora vamos a usar angular para crear nuestro entorno de trabajo para ello usaremos en el terminal la siguiente línea:

ng new angular-hospital

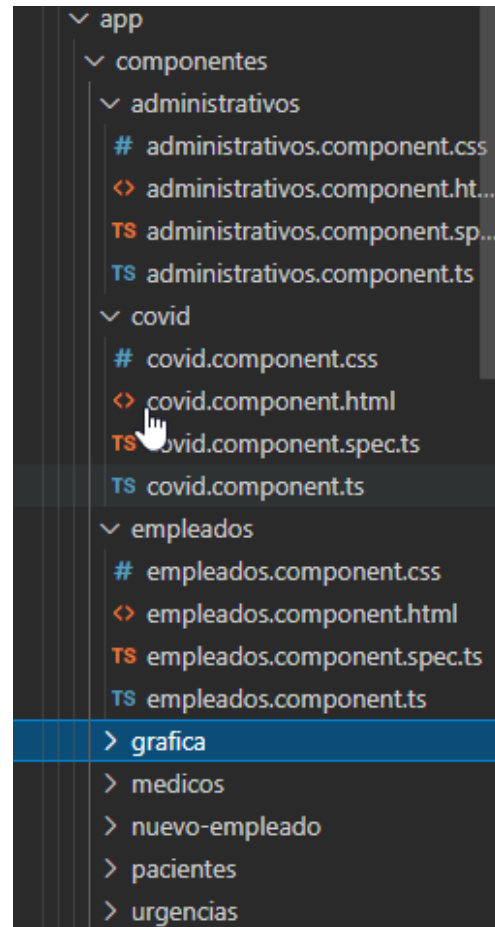
para crear nuestro proyecto, a continuación crearemos los componentes que usaremos:

ng generate component _____

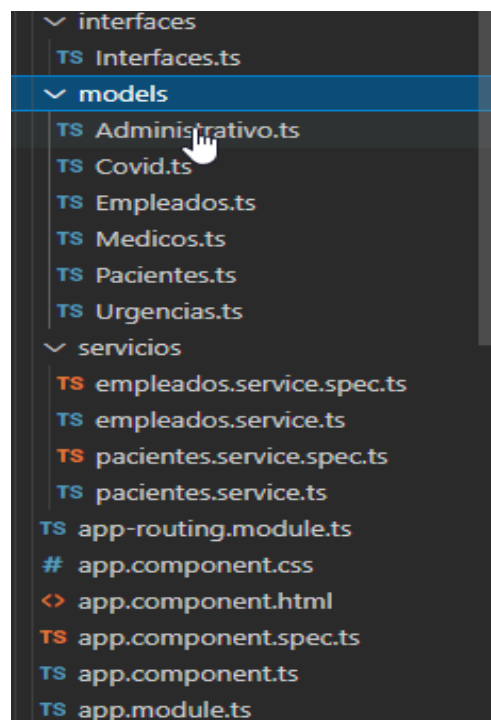
Para crear los componentes que vamos a usar lo que crea tres archivos para editar el componente, un html, un css y un archivo ts, ademas creamos manualmente las clases en una carpeta llamada models, ademas de crear una interfaz para mostrar datos a la hora de trabajar con los archivos html de los componentes. Creamos los servicios de las clases principales

ng generate service _____

Componentes:



Servicios, interfaces y clases:



Servicio de Urgencias Hospital Healty Fox

Empleados
Medicos
Administrativos
Pacientes
Gráficas