

DOCUMENTACIÓN PROYECTO 2 EVALUACIÓN

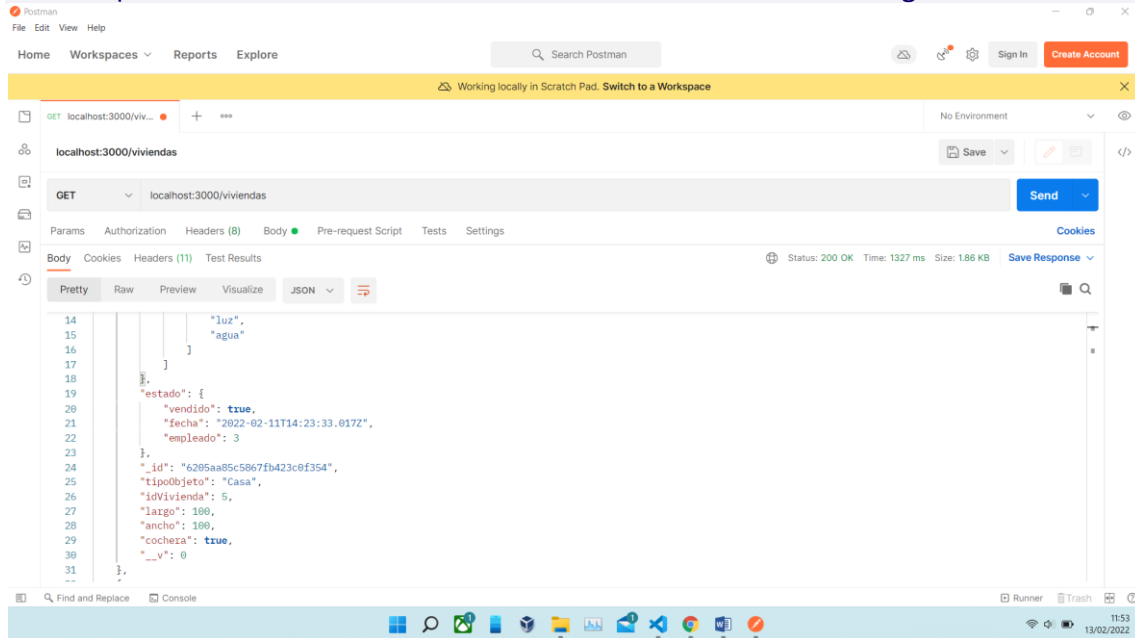
INDICE

1. RUTAS
2. COLECCIONES
3. CLASES
4. VISTA DEL USUARIO FINAL
5. ESTRUCTURA ANGULAR

1. RUTAS

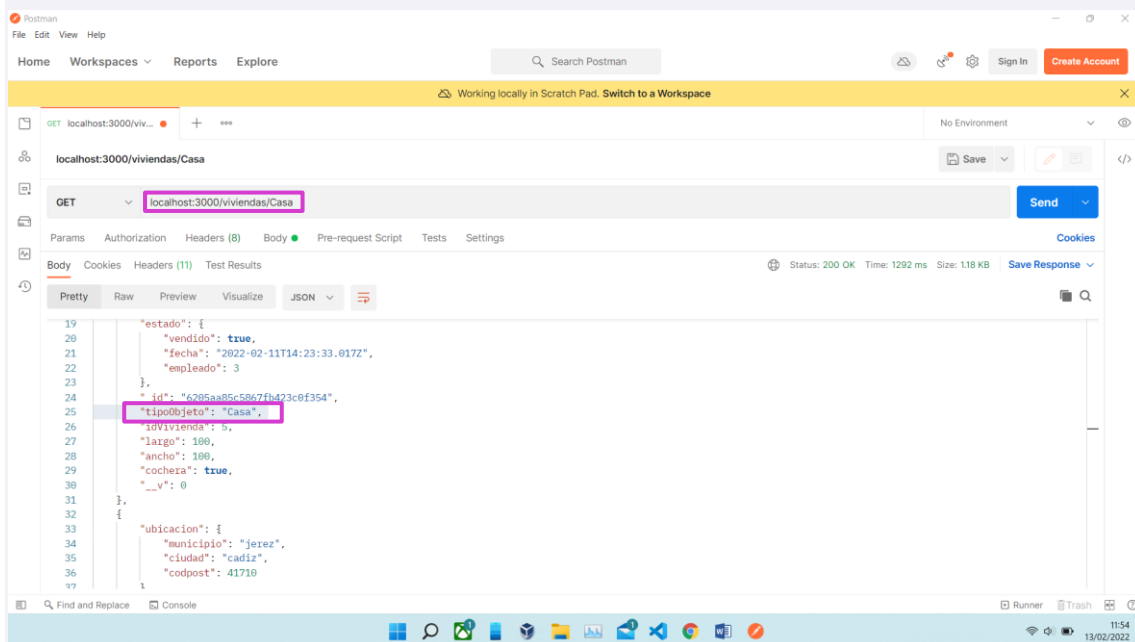
```
this._router.get('/viviendas', this.getViviendas)
```

Se usa para obtener todas las viviendas almacenadas en MongoDB



```
this._router.get('/viviendas/:type', this.getTypes)
```

Se usa para obtener todas las viviendas filtradas por el tipo de objeto (Casa o Chalet)



`this._router.get('/vivienda/:idVivienda', this.getVivienda)`

Se usa para obtener una vivienda con el ID dado

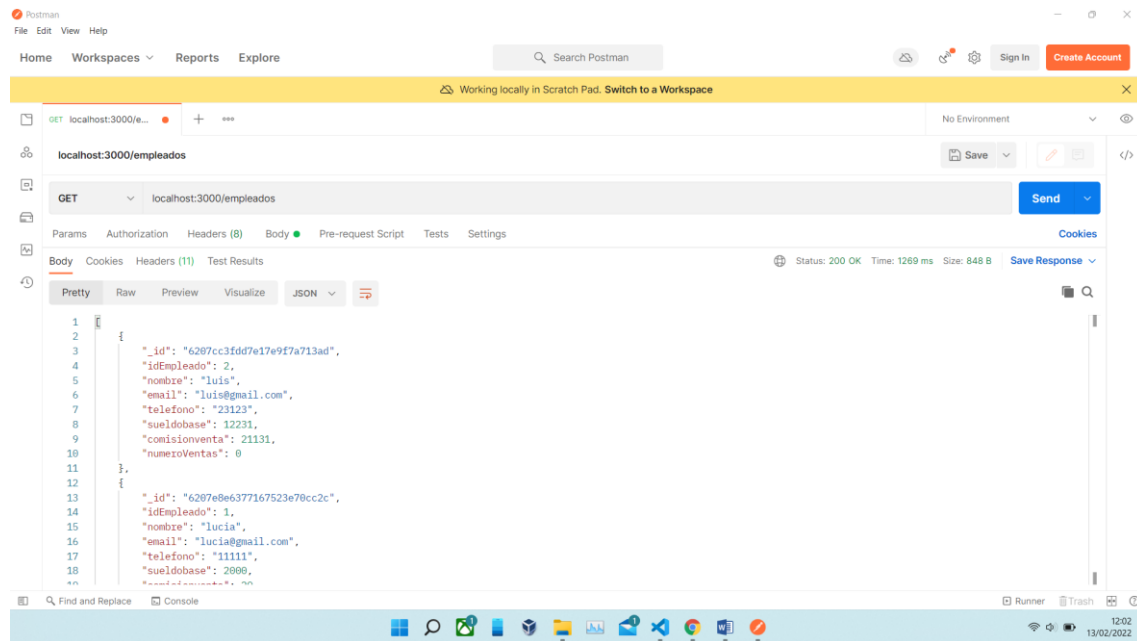
The screenshot shows the Postman application interface. At the top, a text box contains the code `this._router.get('/vivienda/:idVivienda', this.getVivienda)` and the explanation "Se usa para obtener una vivienda con el ID dado". Below this, the Postman window displays a GET request to `localhost:3000/vivienda/5`. The request is successful, with a status of 200 OK. The response body is a JSON object, which is displayed in the "Body" tab. The JSON object contains the following data:

```
{  "agua": true,  "estado": {    "vendido": true,    "fecha": "2022-02-11T14:23:33.017Z",    "empleado": 3  },  "id": "6205aa85c5867fb423c8f354",  "idVivienda": 5,  "latigo": 100,  "ancho": 100,  "cocheza": true,  "v": 0}
```

The value `"idVivienda": 5` is highlighted with a red box. The Postman interface also shows the "Params", "Authorization", "Headers (8)", "Body", "Pre-request Script", "Tests", and "Settings" tabs. The "Body" tab is selected, and the response is displayed in the "Pretty" format. The status bar at the bottom shows the time as 11:59 and the date as 13/02/2022.

```
this._router.get('/empleados', this.getEmpleados)
```

Se usa para obtener todos los empleados de MongoDB

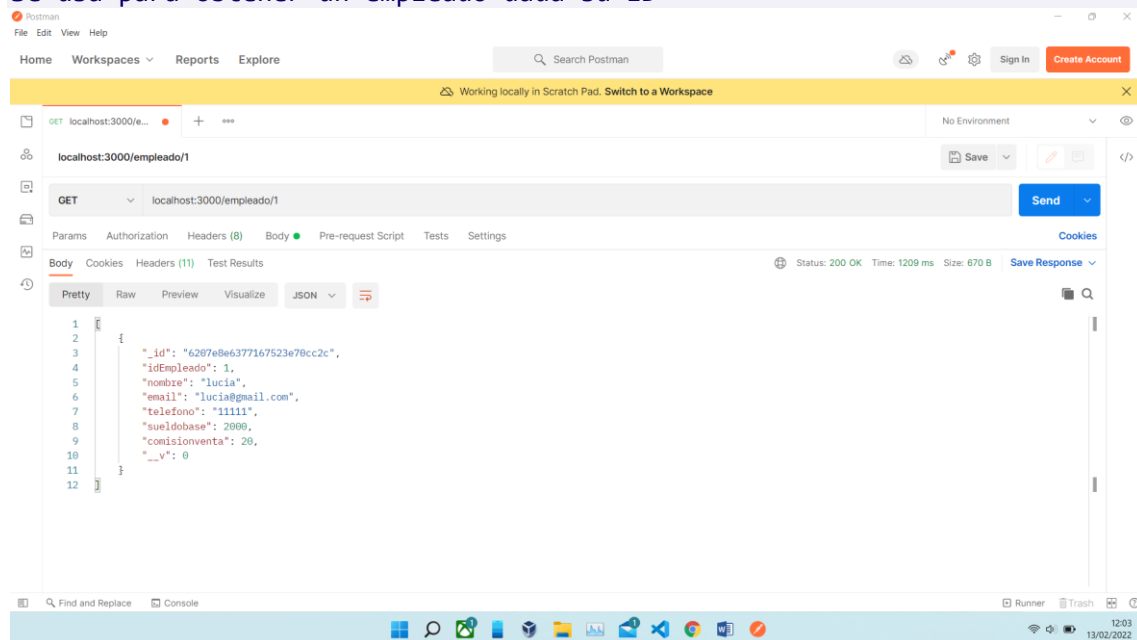


Postman interface showing a GET request to `localhost:3000/empleados`. The response is a JSON array of two employee objects. The status is 200 OK, time is 1269 ms, and size is 848 B.

```
1 {
2   "id": "6287cc3fdd7e17e9f7a713ad",
3   "idEmpleado": 2,
4   "nombre": "Luis",
5   "email": "luis@gmail.com",
6   "telefono": "23123",
7   "suelabase": 12231,
8   "comisionventa": 21131,
9   "numeroVentas": 0
10 },
11 {
12   "id": "6287e8e6377167523e78cc2c",
13   "idEmpleado": 1,
14   "nombre": "Lucia",
15   "email": "lucia@gmail.com",
16   "telefono": "11111",
17   "suelabase": 2800,
18   "comisionventa": 20,
19   "numeroVentas": 0
20 }
```

```
this._router.get('/empleado/:idEmpleado', this.getEmpleado)
```

Se usa para obtener un empleado dada su ID



Postman interface showing a GET request to `localhost:3000/empleado/1`. The response is a JSON object for the employee with ID 1. The status is 200 OK, time is 1209 ms, and size is 670 B.

```
1 {
2   "id": "6287e8e6377167523e78cc2c",
3   "idEmpleado": 1,
4   "nombre": "Lucia",
5   "email": "lucia@gmail.com",
6   "telefono": "11111",
7   "suelabase": 2800,
8   "comisionventa": 20,
9   "numeroVentas": 0
10 }
```

```
this._router.post('/vivienda', this.postVivienda)
```

Se usa para añadir viviendas a MongoDB tanto casas como chalets

A screenshot of the Postman application interface. The top bar shows 'POST localhost:3000/vivienda'. The 'Body' tab is selected, displaying a JSON object with the following structure:

```
{
  "tipoObjeto": "Casa",
  "idVivienda": 10,
  "largo": 12,
  "ancho": 10,
  "municipio": "utreza",
  "ciudad": "sevilla",
  "codpost": 41710,
  "habitaciones": 4,
  "baños": 3,
  "ascensor": false,
  "enuniamiento": false
}
```

The response status is '200 OK' with a time of '1389 ms' and a size of '933 B'. The response body is displayed in 'Pretty' format, showing the same JSON object.

```
this._router.post('/empleados', this.postEmpleado)
```

Se usa para añadir un nuevo empleado a MongoDB

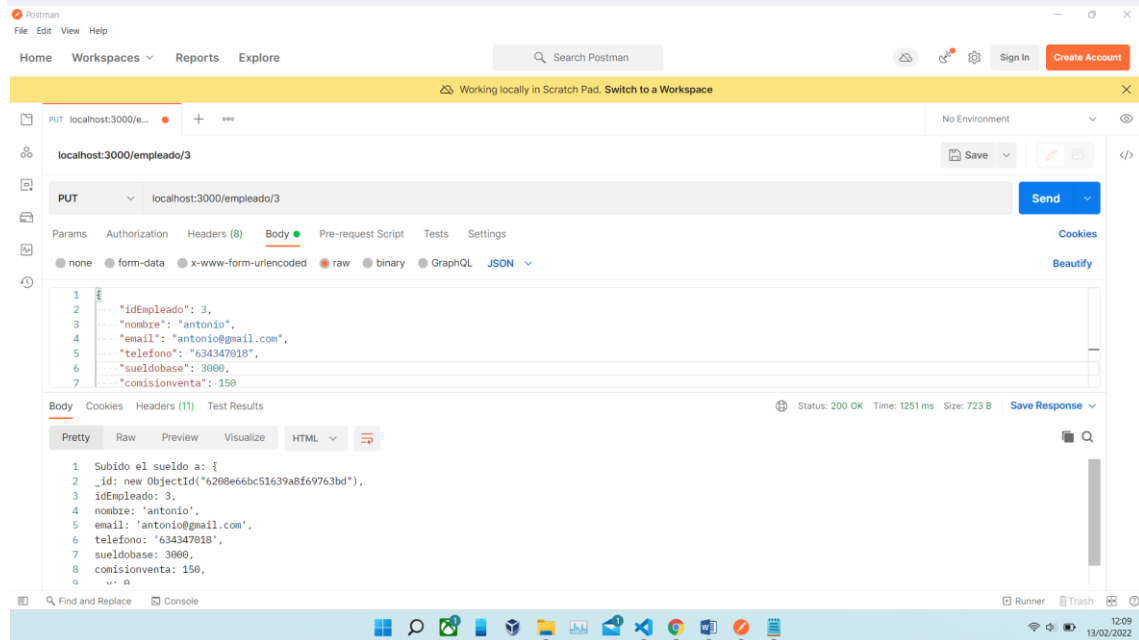
A screenshot of the Postman application interface. The top bar shows 'GET localhost:3000/empleados'. The 'Body' tab is selected, displaying a JSON object with the following structure:

```
{
  "idEmpleado": 3,
  "nombre": "antonio",
  "email": "antonio@gmail.com",
  "telefono": "634347018",
  "sueidobase": 1200,
  "comisionventa": 150
}
```

The response status is '200 OK' with a time of '1241 ms' and a size of '1.16 KB'. The response body is displayed in 'Pretty' format, showing the same JSON object.

```
this._router.put('/empleado/:idEmpleado', this.modificarEmpleado)
```

Se usa para modificar empleados, en este caso se le modifica el sueldo base de 1200 a 300



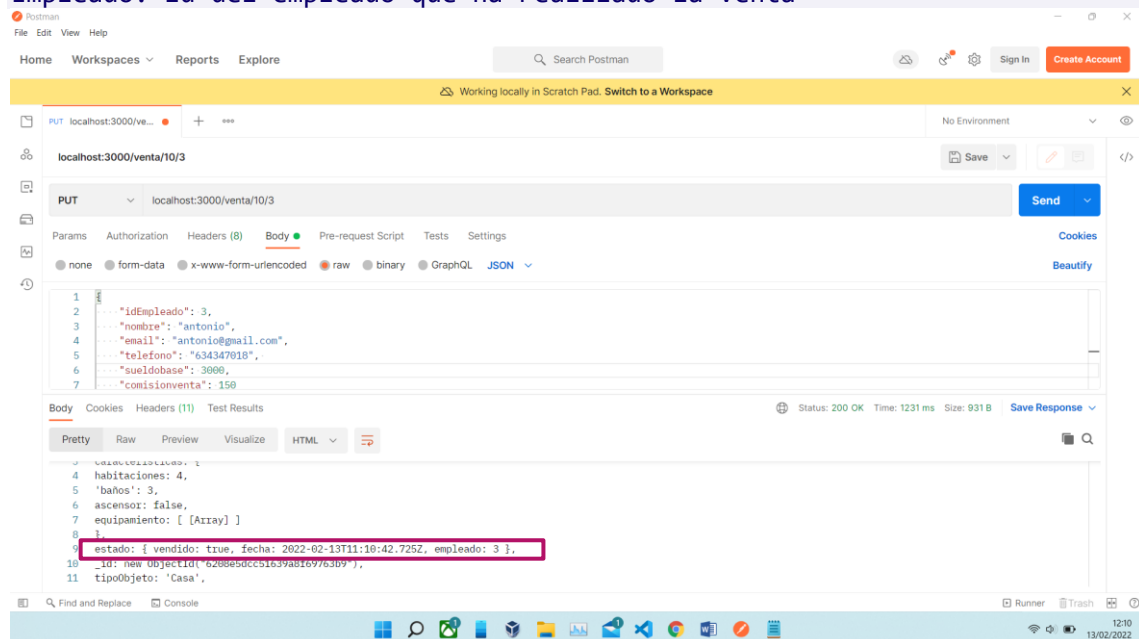
```
this._router.put('/venta/:idVivienda/:idEmpleado', this.updateEstado)
```

Esta función es la más interesante, se usa para realizar una venta. Añadiremos al campo estado de la colección vivienda los siguientes valores:

Vendido: pasa de ser false a true

Fecha: fecha y hora en la que se realiza la venta

Empleado: id del empleado que ha realizado la venta



```

1 Vivienda vendida {
2   ubicacion: { municipio: 'utrera', ciudad: 'sevilla', codpost: 41710 },
3   características: {
4     habitaciones: 4,
5     'baños': 3,
6     ascensor: false,
7     equipamiento: [ [Array] ]
8   },
9   estado: { vendido: true, fecha: 2022-02-13T11:10:42.725Z, empleado: 3 },
10  _id: new ObjectId("6208e5dcc51639a8f69763b9"),
11  tipoObjeto: 'Casa',
12  idVivienda: 10,
13  largo: 12,
14  ancho: 10,
15  cochera: true,
16  __v: 0
17 }

```

`this._router.delete('/deleteEmpleado/:idEmpleado', this.deleteEmpleado)`

Se usa para borrar un empleado

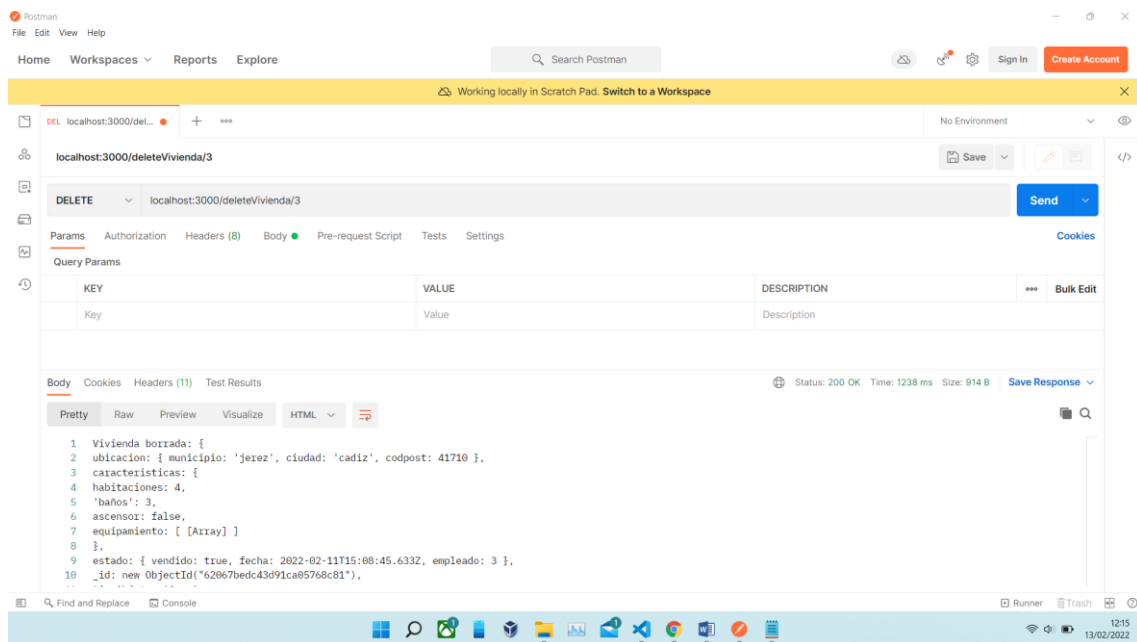
The screenshot shows the Postman interface with a DELETE request to `localhost:3000/deleteEmpleado/1`. The response status is 200 OK. The response body is a JSON object:

```

{
  "_id": "6208e5dcc51639a8f69763b9",
  "idEmpleado": 1,
  "nombre": "Lucia",
  "email": "lucia@gmail.com",
  "telefono": "11111",
  "sueldoBase": 2000,
  "comisionVenta": 20,
  "__v": 0
}

```

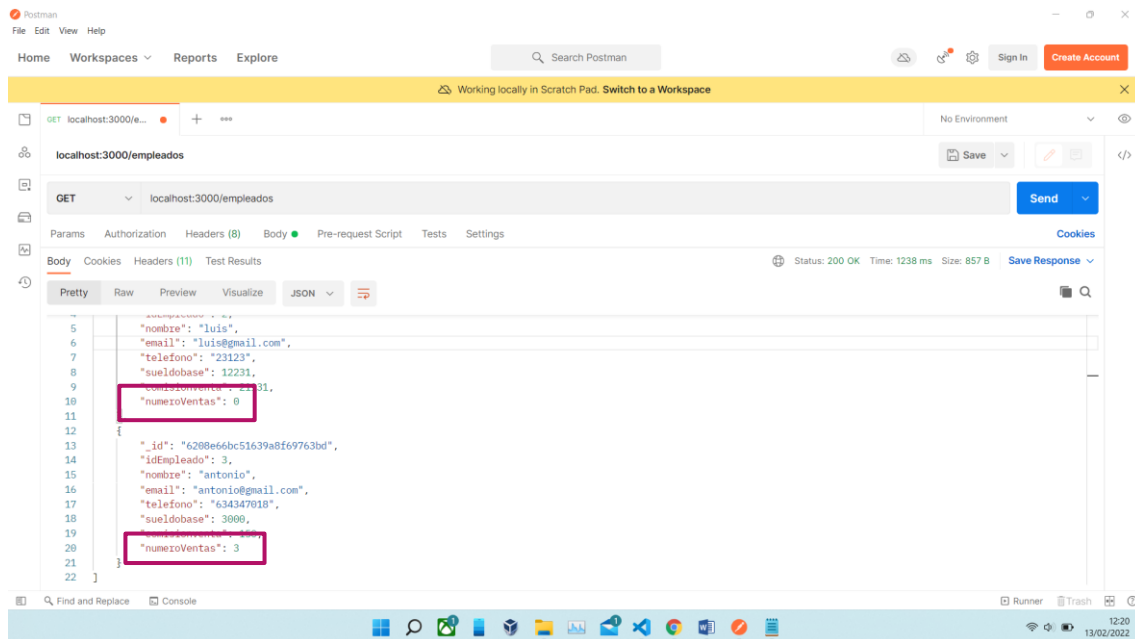
`this._router.delete('/deleteVivienda/:idVivienda', this.deleteVivienda)`
Se usa para eliminar una vivienda



Por último, remarcar esta función:

`this._router.get('/empleados', this.getEmpleado)`

```
private getEmpleados = async (req: Request, res: Response) => {
  await db.conectarBD()
  .then(async () => {
    const query = await modeloEmpleado.aggregate([
      {
        '$lookup': {
          'from': 'viviendas',
          'localField': 'idEmpleado',
          'foreignField': 'estado.empleado',
          'as': 'numeroVentas'
        }
      }, {
        '$project': {
          'idEmpleado': 1,
          'nombre': 1,
          'email': 1,
          'telefono': 1,
          'sueldobase': 1,
          'comisionventa': 1,
          'numeroVentas': {
            '$size': '$numeroVentas'
          }
        }
      }
    ])
    res.json(query)
  })
  .catch((mensaje) => {
    res.send(mensaje)
  })
  await db.desconectarBD()
}
```

Añade un nuevo campo al empleado, que nos muestra cuantas viviendas ha vendido.

2. COLECCIONES

En primer lugar, veremos la colección viviendas.

En ella hacemos uso de las validaciones de mongoose:

Required: agrega un validador requerido para el campo

Unique: Es un ayudante para construir índices únicos de MongoDB

Lowercase: convierte las cadenas de string en minúscula.

Default: establece un valor predeterminado para el campo

Mensajes de errores personalizados.

```
const viviendaSchema = new Schema({
  tipoObjeto: {
    type: String,
    required: 'Que no se te olvide, a ver como lo identificas luego'
  },
  idVivienda: {
    type: Number,
    unique: true,
  },
  largo: Number,
  ancho: Number,
  ubicacion: {
    municipio: String,
    ciudad: {
      type: String,
      lowercase: true,
      required: [true, 'Se te olvida la ciudad!']
    },
    codpost: Number,
  },
  características: {
    habitaciones: Number,
    baños: Number,
    ascensor: Boolean,
    equipamiento: Array
  },
  estado: {
    vendido: {
      type: Boolean,
      default: false
    },
    fecha: Date,
    empleado: Number,
  },
  piscina: Boolean,
```

```
    largojardin: Number,  
    anchojardin: Number,  
    cochera: Boolean  
  })
```

Para continuar veremos la colección de los empleados:

```
const empleadoSchema = new Schema({  
  idEmpleado: {  
    type: Number,  
    unique: true  
  },  
  nombre: String,  
  email: {  
    type: String,  
    required: true  
  },  
  telefono: {  
    type: String,  
    required: true  
  },  
  sueldobase: {  
    type: Number,  
    default: 950  
  },  
  comisionventa: Number,  
})
```


3. CLASES

3.1 CLASE VIVIENDA (PADRE)

La clase Vivienda tiene algo particular, para hacerlo más "profesional" he recurrido a las interfaces para ahorrar líneas de código, las podemos ver en los campos Ubicación y Características.

```
export abstract class Vivienda {  
  private _idVivienda: number;  
  private _largo: number;  
  private _ancho: number;  
  private _ubicacion: Ubicacion;  
  private _caracteristicas: Caracteristicas;  
  private _estado: {  
    vendido: boolean;  
    fecha: Date | null;  
    empleado: number  
  };  
};
```

Las interfaces están formuladas en otro directorio llamados interfaces.ts y serían de tal forma:

```
export interface Ubicacion {  
  municipio: string;  
  ciudad: string;  
  codpost: number;  
}  
  
export interface Caracteristicas {  
  habitaciones: number;  
  baños: number;  
  ascensor: boolean;  
  equipamiento: Array<string>;  
}
```

Los métodos de la clase vivienda son:

```

ubi() {
  return (
    'Municipio: ' +
    this._ubicacion.municipio +
    ', Ciudad: ' +
    this._ubicacion.ciudad +
    ', Codigo postal: ' +
    this._ubicacion.codpost
  );
}

est() {
  return (
    'Vendido: ' + this._estado.vendido +
    ', Fecha: ' + this._estado.fecha +
    ', Empleado: ' + this._estado.empleado
  );
}

carac() {
  return (
    '\nHabitaciones: ' +
    this._caracteristicas.habitaciones +
    ', Baños: ' +
    this._caracteristicas.baños +
    ', ¿Tiene ascensor?: ' +
    this._caracteristicas.ascensor +
    '\nEquipamiento: ' +
    this._caracteristicas.equipamiento
  );
}

```

```

//Calcular el precio de la vivienda, según la ubicación el precio del m2 varía.
preciom2() {
  let preciom2: any;
  if (this.ubicacion.ciudad == 'sevilla') {
    preciom2 = 1386 * this.superficie();
  } else if (this.ubicacion.ciudad == 'almeria') {
    preciom2 = 1088 * this.superficie();
  } else if (this.ubicacion.ciudad == 'jaen') {
    preciom2 = 823 * this.superficie();
  } else if (this.ubicacion.ciudad == 'malaga') {
    preciom2 = 2442 * this.superficie();
  } else if (this.ubicacion.ciudad == 'granada') {
    preciom2 = 1375 * this.superficie();
  } else if (this.ubicacion.ciudad == 'cadiz') {
    preciom2 = 1555 * this.superficie();
  } else if (this.ubicacion.ciudad == 'cordoba') {
    preciom2 = 1220 * this.superficie();
  } else if (this.ubicacion.ciudad == 'huelva') {
    preciom2 = 1253 * this.superficie();
  }
  return Math.round(preciom2);
}

superficie(): number {
  let superficie: number;
  superficie = this._ancho * this.largo;
  return superficie;
}

```

```

    todo() {
      return `Superficie: ${this.superficie()},
        Precio: ${this.preciom2()},
        Ubicación: [ ${this.ubi()} ],
        Características: [ ${this.carac()} ],
        Estado: [ ${this.est()} ]`;
    }
  }

```

3.2 CLASE CHALET Y CASA (HIJAS)

```

export class Chalet extends Vivienda {
  private _piscina: boolean;
  private _largojardin: number;
  private _anchojardin: number;
}

```

```

export class Casa extends Vivienda {
  private _cochera: boolean;
}

```

Polimorfismo en la clase Casa:

```

    preciom2() {
      let preciom2 = super.preciom2();
      if (this._cochera == true) {
        preciom2 += 1.000;
      }
      return Math.round(preciom2);
    }
  }

```

```

    todo() {
      return super.todo() + "Dispone de cochera: " + this._cochera;
    }
  }

```

Polimorfismo en la clase Chalet:

```

    preciom2() {
      let preciom2 = super.preciom2();
      let preciojardin = this.m2jardin();
      preciom2 = preciom2 + preciojardin;
      if (this._piscina == true) {
        preciom2 += 200;
      }
      return Math.round(preciom2);
    }
  }

```

```

    todo() {
      let resultado: string;
      resultado = `${super.todo()}, ¿Tiene piscina?: ${
        this._piscina
      }, Superficie del jardín(m2): ${this.sjardin()},`;
      return resultado;
    }
  }

```

4. VISTA DEL USUARIO FINAL

Hablaremos en primer lugar del menú CRUD (CREATE, READ, UPDATE AND DELETE).

IDEALISTA VIVIENDAS EMPLEADOS ABOUT ME

Listado de Empleados

Agregar

Id	Nombre	Email	Telefono	Sueldo Base	Comision Venta	Numero Ventas	
2	luis	luis@gmail.com	23123	\$12,231.00	\$21,131.00	0	 
3	antonio	antonio@gmail.com	634347018	\$3,000.00	\$150.00	3	 
1	Francisco	francisco@gmail.com	955486752	\$1,100.00	\$200.00	0	 
4	Lola	lola@hotmail.com	662137012	\$1,400.00	\$70.00	0	 

AGREGAR UN EMPLEADO

IDEALISTA VIVIENDAS EMPLEADOS ABOUT ME











CREAR EMPLEADO

Agregar

IDEALISTA VIVIENDAS EMPLEADOS ABOUT ME

Listado de Empleados

Agregar

Id	Nombre	Email	Telefono	Sueldo Base	Comision Venta	Numero Ventas	
2	luis	luis@gmail.com	23123	\$12,231.00	\$21,131.00	0	 
3	antonio	antonio@gmail.com	634347018	\$3,000.00	\$150.00	3	 
1	Francisco	francisco@gmail.com	955486752	\$1,100.00	\$200.00	0	 
4	Lola	lola@hotmail.com	662137012	\$1,400.00	\$70.00	0	 
5	pepe	pepe@hotmail.com	99898226	\$1,250.00	\$100.00	0	 

Si dejamos los campos vacíos nos saltará un error de validación:

IDEALISTA VIVIENDAS EMPLEADOS ABOUT ME

CREAR EMPLEADO

Pro favor, ingrese todos los campos

Agregar

¿Y como se consigue esto?

Pues bien, hay dos partes imprescindibles:

La primera en el documento HTML:

Añadimos la directiva *ngIf, es decir se activará si...

Submitted: significa que hayamos dado click al boton de submit

crearEmpleado.invalid: significa que el formulario de crear empleado tiene que ser invalido

```
<div class="container">
  <div class="row">
    <div class="col-lg-6 offset-lg-3">
      <div class="card text-center">
        <div class="card-body">
          <h3>{{titulo}}</h3>
          <span *ngIf="submitted && crearEmpleado.invalid" class="badge badge-danger">Pro favor, ingrese todos los campos</span>
          <form [formGroup]="crearEmpleado" (ngSubmit)="agregarEmpleado()">
            <div class="row">
              <input type="number" formControlName="idEmpleado" class="mt-2 form-control form-control-lg" placeholder="ID">
            </div>
            <div class="row">
              <input type="text" formControlName="nombre" class="form-control form-control-lg mt-3" placeholder="Nombre">
            </div>
            <div class="row">
              <div class="col"><input type="text" formControlName="email" class="form-control form-control-lg mt-3" placeholder="Email"></div>
              <div class="col"><input type="text" formControlName="telefono" class="form-control form-control-lg mt-3" placeholder="Telefono"></div>
            </div>
            <div class="row">
              <div class="col"><input type="number" formControlName="sueldobase" class="form-control form-control-lg mt-3" placeholder="Sueldo Base"></div>
              <div class="col"><input type="number" formControlName="comisionventa"></div>
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
```

¿Qué quiere decir que el formulario sea invalido?











Pues significa que el formulario no cumple estos requisitos que se encuentran en el archivo ts del componente:


```
export class CrearEmpleadoComponent implements OnInit {  
  crearEmpleado: FormGroup;  
  submitted = false;  
  id: string | null  
  titulo = 'CREAR EMPLEADO'  
  constructor(  
    private fb: FormBuilder,  
    private empleadoService: EmpleadoService,  
    private activeRoute: ActivatedRoute) {  
    this.crearEmpleado = this.fb.group({  
      idEmpleado: ['', Validators.required],  
      nombre: ['', Validators.required],  
      email: ['', [Validators.required, Validators.email]],  
      telefono: ['', Validators.required],  
      sueldobase: ['', Validators.required],  
      comisionventa: ['', Validators.required]  
    })  
    this.id = this.activeRoute.snapshot.paramMap.get("idEmpleado")  
  }  
}
```

BORRAR EMPLEADO

Listado de Empleados

Agregar


Id	Nombre	Email	Telefono	Sueldo Base	Comision Venta	Numero Ventas	
2	luis	luis@gmail.com	23123	\$12,231.00	\$21,131.00	0	 
3	antonio	antonio@gmail.com	634347018	\$3,000.00	\$150.00	3	 
1	Francisco	francisco@gmail.com	955486752	\$1,100.00	\$200.00	0	 
4	Lola	lola@hotmail.com	662137012	\$1,400.00	\$70.00	0	 
5	pepe	pepe@hotmail.com	99898226	\$1,250.00	\$100.00	0	 



Como vemos en la siguiente foto, pepe ha sido eliminado.

Listado de Empleados

Agregar

Id	Nombre	Email	Telefono	Sueldo Base	Comision Venta	Numero Ventas	
2	luis	luis@gmail.com	23123	\$12,231.00	\$21,131.00	0	 
3	antonio	antonio@gmail.com	634347018	\$3,000.00	\$150.00	3	 
1	Francisco	francisco@gmail.com	955486752	\$1,100.00	\$200.00	0	 
4	Lola	lola@hotmail.com	662137012	\$1,400.00	\$70.00	0	 

EDITAR EMPLEADO

Vamos a editar a Lola:

IDEALISTA









VIVIENDAS

EMPLEADOS

ABOUT ME

Listado de Empleados

Agregar

Id	Nombre	Email	Telefono	Sueldo Base	Comision Venta	Numero Ventas	
2	luis	luis@gmail.com	23123	\$12,231.00	\$21,131.00	0	 
3	antonio	antonio@gmail.com	634347018	\$3,000.00	\$150.00	3	 
1	Francisco	francisco@gmail.com	955486752	\$1,100.00	\$200.00	0	 
4	Lola	lola@hotmail.com	662137012	\$1,400.00	\$70.00	0	 

Como podemos ver, al haber hecho click en LOLA su ID viene automaticamente al formulario de edición.

IDEALISTA

VIVIENDAS

EMPLEADOS

ABOUT ME

EDITAR EMPLEADO

4

Nombre

Email

Telefono

Sueldo Base

Comision Venta

Editar

Cambiaremos el email:

EDITAR EMPLEADO

4

Lola

lolita@hotmail.com

662137012

1400

70

Editar

Como podemos ver, el email de LOLA ha cambiado.

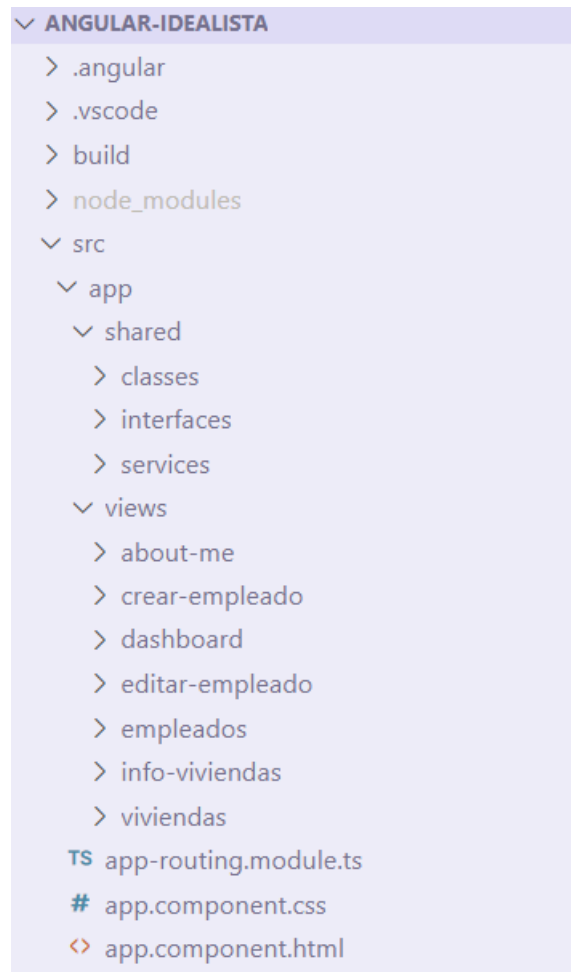
Listado de Empleados

Agregar

Id	Nombre	Email	Telefono	Sueldo Base	Comision Venta	Numero Ventas	
2	luis	luis@gmail.com	23123	\$12,231.00	\$21,131.00	0	 
3	antonio	antonio@gmail.com	634347018	\$3,000.00	\$150.00	3	 
1	Francisco	francisco@gmail.com	955486752	\$1,100.00	\$200.00	0	 
4	Lola	lolita@hotmail.com	662137012	\$1,400.00	\$70.00	0	 

5. ESTRUCTURA ANGULAR

El proyecto está estructurado de la siguiente forma:



Como podemos ver, dentro del directorio APP existen dos ramas,

SHARED: donde encontraremos toda la parte ''logica''

VIEWS: donde encontraremos toda la parte ''vista''

Es una estructura pensada en el MVC. Separando las partes vistas de los ''controladores''.

Dentro del directorio SHARED encontraremos:

- Clases
- Interfaces
- Servicios

```
▼ app
  ▼ shared
    ▼ classes
      TS casa.ts
      TS chalet.ts
      TS empleado.ts
      TS usuario.ts
      TS vivienda.ts
    ▼ interfaces
      TS interfaces.ts
    ▼ services
      TS empleado.service.spec.ts
      TS empleado.service.ts
      TS vivienda.service.spec.ts
      TS vivienda.service.ts
```

Dentro del directorio VIEWS encontraremos:

- Componentes

```
▼ app
  > shared
  ▼ views
    > about-me
    > crear-empleado
    > dashboard
    > editar-empleado
    > empleados
    > info-viviendas
    > viviendas
```