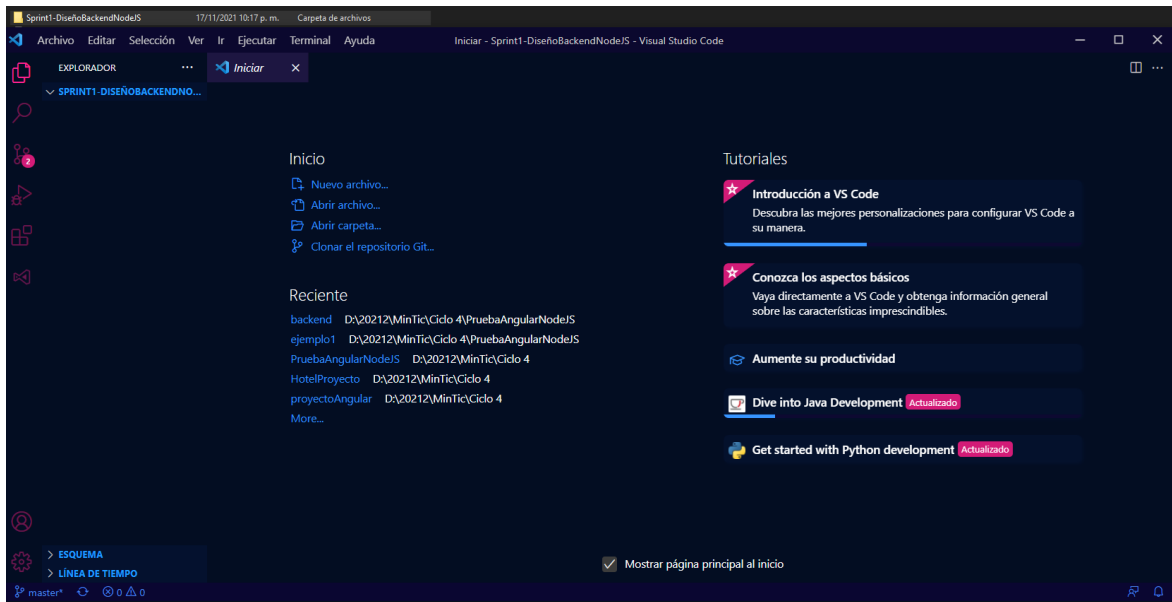
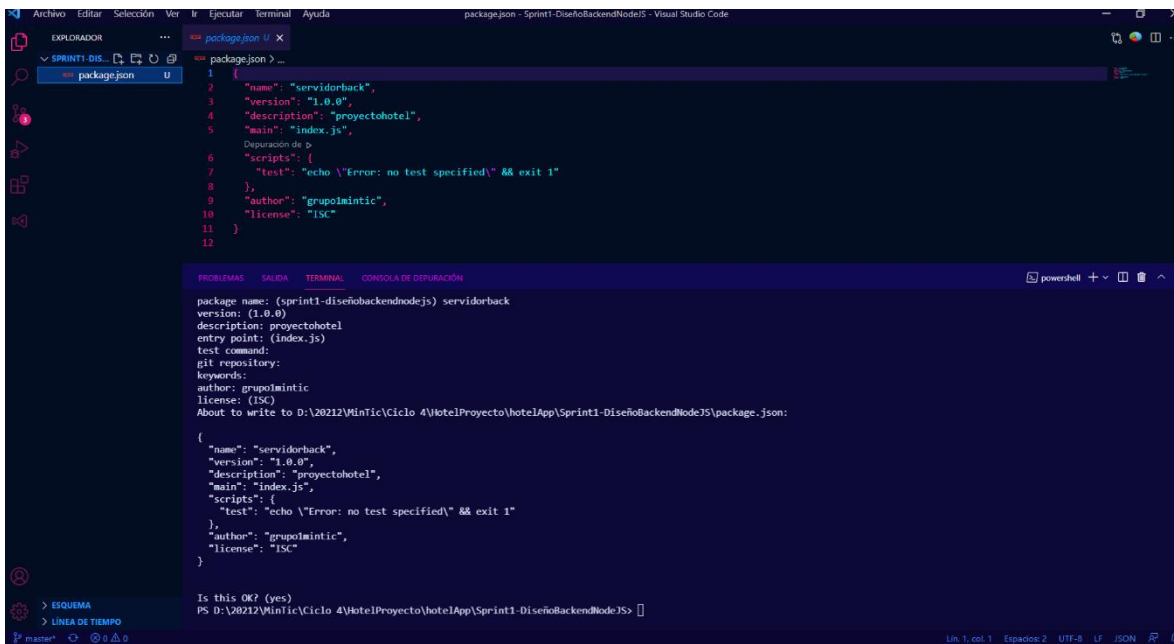


## 1. Creación de la carpeta del back-end y abierta en Visual Studio Code.



## 2. Iniciamos el servidor.



### 3. Instalación de nodemon.

```
package.json - Sprint1-DesarrolloBackendNodeJS - Visual Studio Code
EXPLORADOR
package.json
package-lock.json
package.json

package.json
4 "description": "proyecto:hotel",
5 "main": "index.js",
6 "scripts": {
7   "test": "echo \"Error: no test specified\" && exit 1"
8 },
9 "author": "grupointinc",
10 "license": "ISC",
11 "dependencies": {
12   "nodemon": "^2.0.15"
13 }
14
15

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACION
powerhell + ~ ~ ~ ~ ~
}
"author": "grupointinc",
"license": "ISC"
}

Is this OK? (yes)
PS D:\20212\MiniCiclo 4\HotelProyecto\hotelApp\Sprint1-DesarrolloBackendNodeJS> npm install -o nodemon

> nodemon@2.0.15 postinstall: D:\20212\MiniCiclo 4\HotelProyecto\hotelApp\Sprint1-DesarrolloBackendNodeJS\node_modules\nodemon
> node bin/postinstall || exit 0

npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN servidorback@1.0.0 No repository field.

+ nodemon@2.0.15
added 116 packages from 53 contributors and audited 117 packages in 15.447s

15 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

PS D:\20212\MiniCiclo 4\HotelProyecto\hotelApp\Sprint1-DesarrolloBackendNodeJS>
```

#### 4. Instalación de express, mongoose y dotenv

The screenshot displays the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left shows the project structure:

- SPRINT1-DISEÑOBACKENDONOLINE
- node\_modules
- package-lock.json
- package.json

The main editor area shows the contents of `package.json`:

```
{  
  "description": "proyectoHotel",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "grupomintic",  
  "license": "ISC",  
  "dependencies": {  
    "dotenv": "^10.0.0",  
    "express": "^4.17.1",  
    "mongoose": "^6.0.13",  
    "nodemon": "^2.0.15"  
  }  
}
```

The bottom panel contains the integrated terminal with the following command and output:

```
PS D:\20212\MinTic\Ciclo 4\VotelProyecto\hotelApp\Sprint1-DiseñoBackendNodeJS> npm install express mongoose dotenv  
npm WARN deprecated server-destroy@1.0.0 No repository field.  
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fs-events@2.3.2 (node_modules\fs-events):  
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fs-events@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})  
  
+ dotenv@10.0.0  
+ express@4.17.1  
+ mongoose@6.0.13  
added 84 packages from 96 contributors and audited 201 packages in 58.733s  
  
19 packages are looking for funding  
run `npm fund` for details  
  
found 0 vulnerabilities  
  
PS D:\20212\MinTic\Ciclo 4\VotelProyecto\hotelApp\Sprint1-DiseñoBackendNodeJS>
```

## 5. Conexión a la base de datos.

The screenshot shows the Visual Studio Code editor with a project named 'Sprint1-DisenoBackendNodeJS'. The file explorer on the left shows the project structure, including 'config', 'controllers', 'models', 'node\_modules', 'routes', 'index.js', 'package-lock.json', and 'package.json'. The main editor displays the 'db.js' file in the 'config' directory, which contains the following code:

```
1 const mongoose = require('mongoose');
2
3 //
4 const conectarDB = async () => {
5   try {
6     await mongoose.connect('mongodb+srv://jean:12345@cluster0.r0xf.mongodb.net/hotel', {
7       useNewUrlParser: true,
8       useUnifiedTopology: true,
9     });
10    console.log('Base de datos conectada correctamente');
11  } catch (error) {
12    console.log('Error de conexion');
13    process.exit(1);
14  }
15 }
16
17 module.exports = conectarDB;
```

The terminal at the bottom shows the output of running the application with Node.js. It indicates that the server is connected to the database and is listening on port 3000.

```
[nodemon] app crashed - waiting for file changes before starting...
[nodemon] restarting due to changes...
[nodemon] starting node .
El servidor está conectado
Base de datos conectada correctamente
[nodemon] restarting due to changes...
[nodemon] starting node .
[nodemon] restarting due to changes...
[nodemon] starting node .
El servidor escuchando
Base de datos conectada correctamente
[nodemon] restarting due to changes...
[nodemon] starting node .
[nodemon] restarting due to changes...
[nodemon] starting node .
El servidor escuchando
```

## 6. Creación de los módulos.

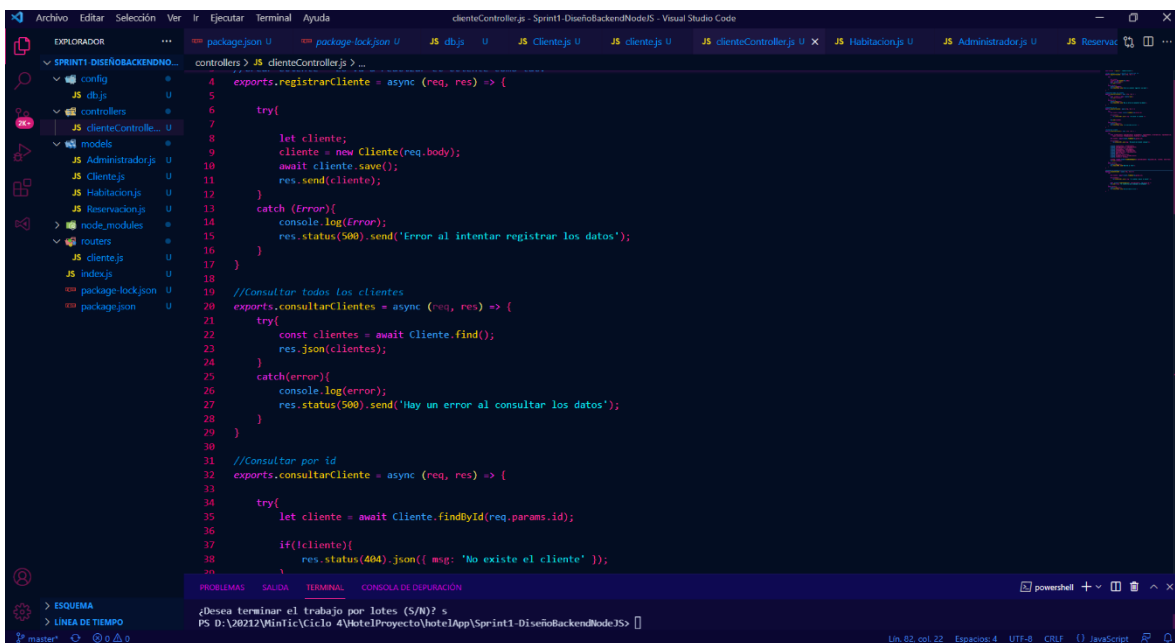
The screenshot shows the Visual Studio Code editor with the same project. The file explorer on the left shows the project structure, including 'config', 'controllers', 'models', 'node\_modules', 'routes', 'index.js', 'package-lock.json', and 'package.json'. The main editor displays the 'Cliente.js' file in the 'models' directory, which contains the following code:

```
1 const mongoose = require('mongoose');
2
3 //Esquema de la clase cliente
4 const ClienteSchema = mongoose.Schema({
5
6   tipoDocumento: {
7     type: String,
8     required: true
9   },
10
11   numeroDocumento: {
12     type: String,
13     required: true
14   },
15
16   primerNombre: {
17     type: String,
18     required: true
19   },
20
21   segundoNombre: {
22     type: String,
23     required: false
24   },
25
26   primerApellido: {
27     type: String,
28     required: true
29   },
30
31   segundoApellido: {
32     type: String,
33     required: false
34   },
35
36   email: {
37     type: String,
38     required: true
39   }
40 });
```

The terminal at the bottom shows the output of running the application with Node.js. It indicates that the server is connected to the database and is listening on port 3000.

```
[nodemon] app crashed - waiting for file changes before starting...
[nodemon] restarting due to changes...
[nodemon] starting node .
El servidor está conectado
Base de datos conectada correctamente
[nodemon] restarting due to changes...
[nodemon] starting node .
[nodemon] restarting due to changes...
[nodemon] starting node .
El servidor escuchando
Base de datos conectada correctamente
[nodemon] restarting due to changes...
[nodemon] starting node .
[nodemon] restarting due to changes...
[nodemon] starting node .
El servidor escuchando
```

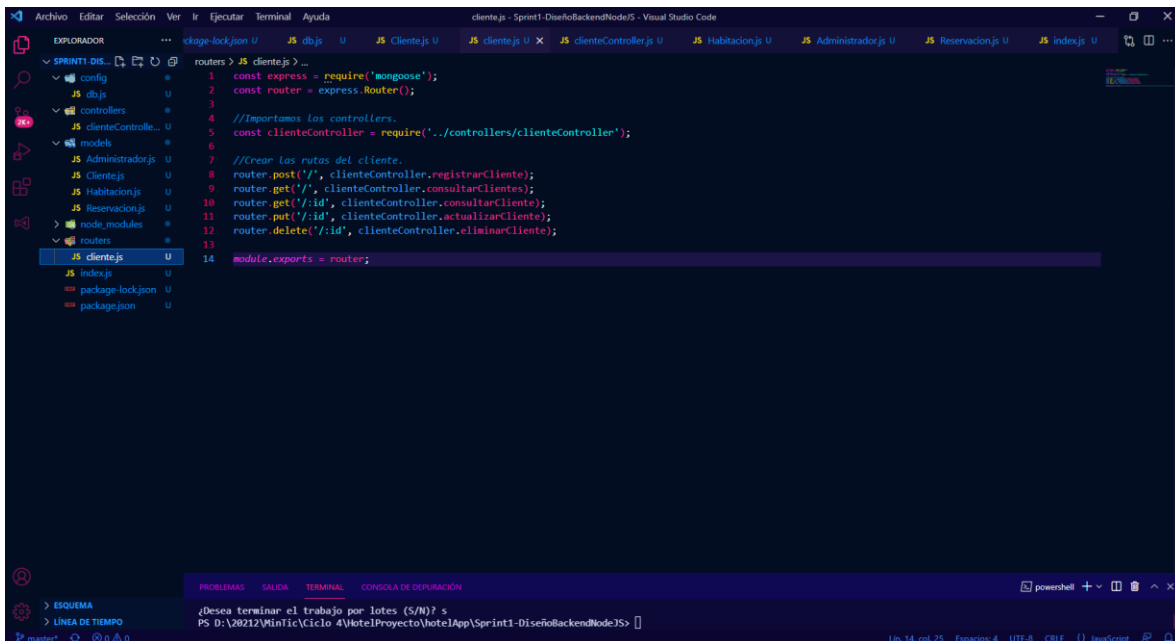
## 7. CONTROLLER completo de la clase cliente



The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the code for `clienteController.js`. The code defines three asynchronous functions: `registrarCliente`, `consultarClientes`, and `consultarCliente`. The `registrarCliente` function takes a request and response, creates a new `Cliente` object from the request body, saves it, and sends it back. The `consultarClientes` function finds all clients and returns them as JSON. The `consultarCliente` function finds a client by ID and returns it as JSON, or a 404 error if it doesn't exist. The bottom status bar shows the file is at line 82, column 22.

```
1 exports.registrarCliente = async (req, res) => {
2   try{
3     let cliente;
4     cliente = new Cliente(req.body);
5     await cliente.save();
6     res.send(cliente);
7   } catch (Error){
8     console.log(Error);
9     res.status(500).send('Error al intentar registrar los datos');
10  }
11 }
12 //Consultar todos los clientes
13 exports.consultarClientes = async (req, res) => {
14   try{
15     const clientes = await Cliente.find();
16     res.json(clientes);
17   } catch(error){
18     console.log(error);
19     res.status(500).send('Hay un error al consultar los datos');
20   }
21 }
22 //Consultar por id
23 exports.consultarCliente = async (req, res) => {
24   try{
25     let cliente = await Cliente.findById(req.params.id);
26     if(!cliente){
27       res.status(404).json({ msg: 'No existe el cliente' });
28     }
29   }
30 }
31
```


## 8. ROUTERS de la clase cliente.



The screenshot shows the Visual Studio Code editor with the file explorer on the left. The main editor window shows the code for `cliente.js`. The code sets up an Express application, requires the `mongoose` library, and creates an Express router. It then imports the `clienteController` and registers routes for the client: a POST route for registration, a GET route for listing all clients, a GET route for getting a client by ID, a PUT route for updating a client, and a DELETE route for deleting a client. The router is then exported. The bottom status bar shows the file is at line 14, column 25.

```
1 const express = require('express');
2 const router = express.Router();
3 //Importamos los controllers.
4 const clienteController = require('../controllers/clienteController');
5 //Crear las rutas del cliente.
6 router.post('/', clienteController.registrarCliente);
7 router.get('/', clienteController.consultarClientes);
8 router.get('/:id', clienteController.consultarCliente);
9 router.put('/:id', clienteController.actualizarCliente);
10 router.delete('/:id', clienteController.eliminarCliente);
11 module.exports = router;
```

## 9. Configuración del archivo index.js para las rutas del cliente.



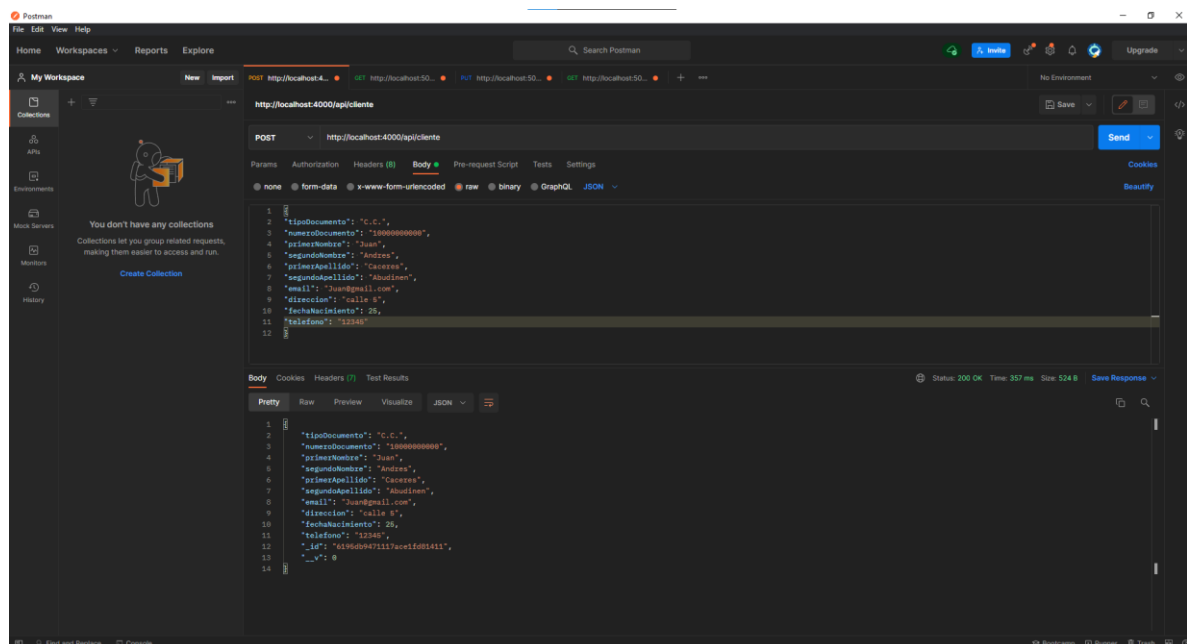
```
1 const express = require("express");
2 const conectarDB = require("./config/db");
3
4 const app = express();
5
6 //Conexión a la base de datos.
7 conectarDB();
8 app.use(express.json());
9
10 //Llamamos las rutas.
11 app.use("/api/cliente", require("./routes/cliente"));
12
13
14 app.listen(4000, () => {
15   console.log("El servidor escuchando");
16 })
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
¿Desea terminar el trabajo por lotes (Y/N)? s
PS D:\20212\MinTic\Ciclo 4\hotelProyecto\hotelApp\Sprint1-DiseñoBackendNodeJS>
```

## Pruebas unitarias de las rutas con la herramienta POSTMAN

### Prueba POST:



POST http://localhost:4000/api/cliente

Body

```
1 {
2   "tipoDocumento": "C.C.",
3   "numeroDocumento": "10000000000",
4   "primerNombre": "Juan",
5   "segundoNombre": "Andres",
6   "primerApellido": "Caceres",
7   "segundoApellido": "Abudines",
8   "email": "Juan@gmail.com",
9   "direccion": "calle 8",
10  "fechaNacimiento": 28,
11  "telefono": "82345"
12 }
```

Body

```
1 {
2   "tipoDocumento": "C.C.",
3   "numeroDocumento": "10000000000",
4   "primerNombre": "Juan",
5   "segundoNombre": "Andres",
6   "primerApellido": "Caceres",
7   "segundoApellido": "Abudines",
8   "email": "Juan@gmail.com",
9   "direccion": "calle 8",
10  "fechaNacimiento": 28,
11  "telefono": "12345",
12  "id": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50aWZlIiwiaWF0IjoxNjM0MjE1MjE1fQ",
13  "v": 0
14 }
```

Status: 200 OK Time: 357 ms Size: 524 B Save Response

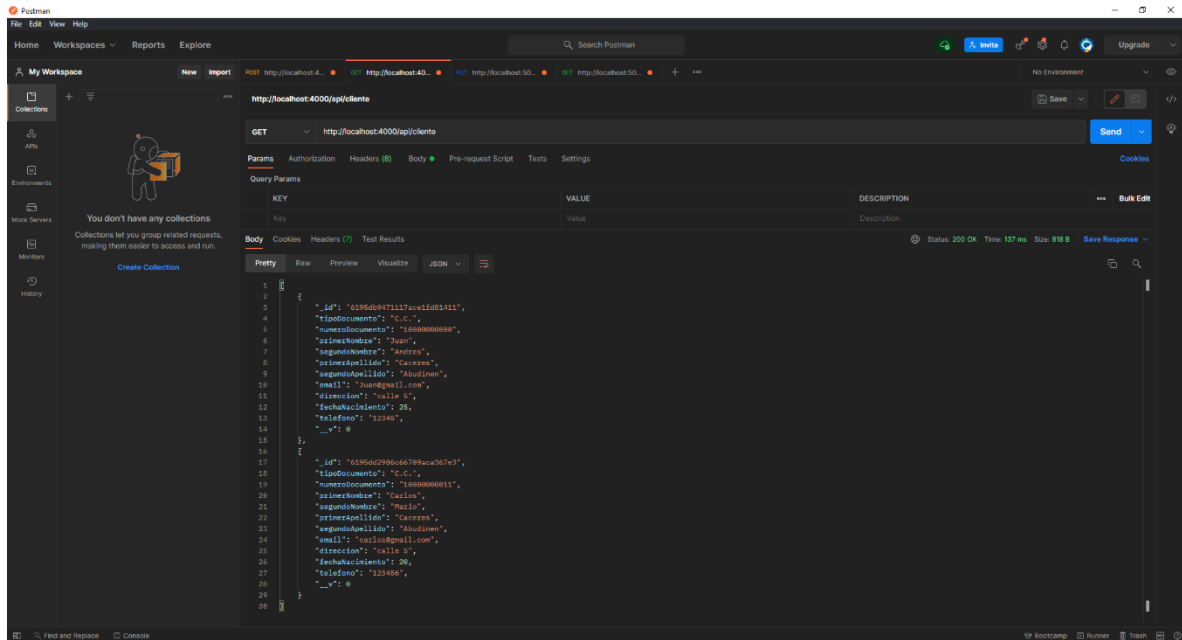
## Resultado POST en la base de datos:

The screenshot displays the MongoDB Compass web interface. On the left sidebar, the 'Local' connection is selected, showing the 'Hotel' database and the 'clientes' collection. The main panel shows the 'Documents' tab for the 'clientes' collection. A single document is displayed, representing a client record. The document structure is as follows:

```
{
  "_id": "001ec1487a189009471327a45f084113",
  "tipoDocumento": "C",
  "numeroDocumento": "18000000000",
  "primerNombre": "Juan",
  "segundoNombre": "Andrés",
  "primerApellido": "García",
  "segundoApellido": "Rodríguez",
  "email": "juan@gmail.com",
  "direccion": "Calle 5ª",
  "fechaNacimiento": "25/01/1990",
  "telefono": "33945",
  "..."
}
```

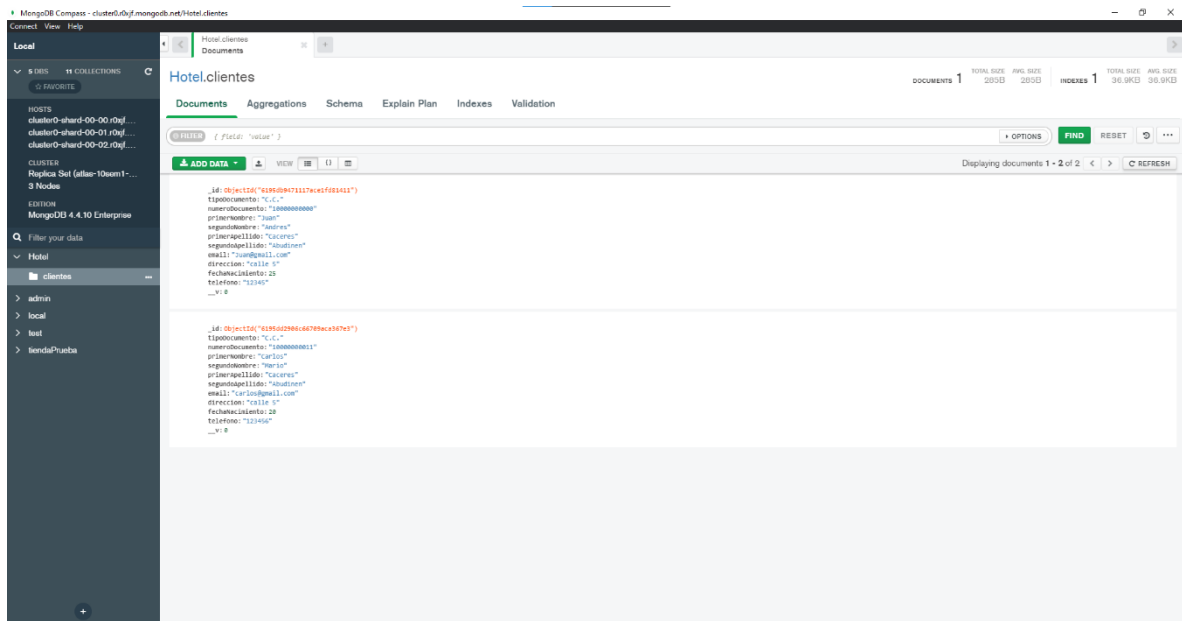
The interface also shows a filter bar with the filter '{ "field": "value" }' and a status bar indicating 'Displaying documents 1 - 1 of 1'.

Prueba GET de todos los registros:



Postman interface showing a GET request to `http://localhost:4000/api/clientes`. The response is a JSON array of two client records.

```
1 [
2   {
3     "_id": "6196d0971117acc1d081411",
4     "tipoDocumento": "C.C.",
5     "numeroDocumento": "1888888888888",
6     "primerNombre": "Juan",
7     "segundoNombre": "Andrés",
8     "primerApellido": "Caceres",
9     "segundoApellido": "Abudinen",
10    "email": "juan@gmail.com",
11    "direccion": "calle 5",
12    "fechaNacimiento": 28,
13    "telefono": "12345",
14    "_v": 0
15  },
16  {
17    "_id": "6196d02996c64789aca307e3",
18    "tipoDocumento": "C.C.",
19    "numeroDocumento": "1888888888888",
20    "primerNombre": "Carlos",
21    "segundoNombre": "Mario",
22    "primerApellido": "Caceres",
23    "segundoApellido": "Abudinen",
24    "email": "carlos@gmail.com",
25    "direccion": "calle 5",
26    "fechaNacimiento": 30,
27    "telefono": "123456",
28    "_v": 0
29  }
30 ]
```

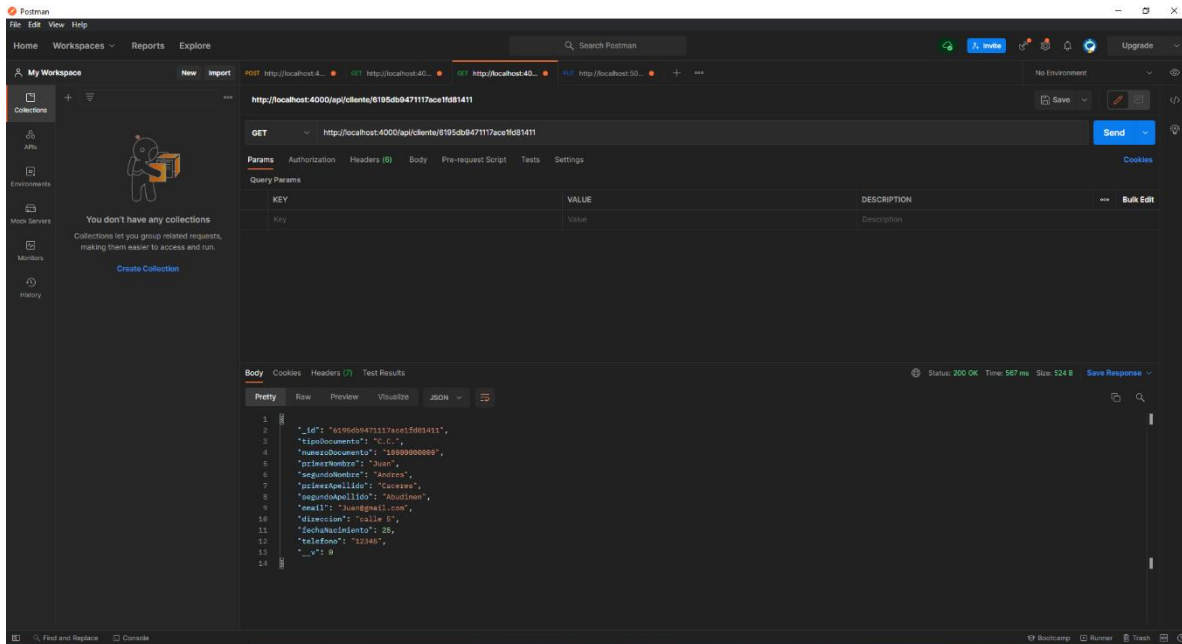


MongoDB Compass interface showing the `clientes` collection in the `Hotel` database. Two documents are displayed.

```
{ "_id": "6196d0971117acc1d081411",
  "tipoDocumento": "C.C.",
  "numeroDocumento": "1888888888888",
  "primerNombre": "Juan",
  "segundoNombre": "Andrés",
  "primerApellido": "Caceres",
  "segundoApellido": "Abudinen",
  "email": "juan@gmail.com",
  "direccion": "calle 5",
  "fechaNacimiento": 28,
  "telefono": "12345",
  "_v": 0 }

{ "_id": "6196d02996c64789aca307e3",
  "tipoDocumento": "C.C.",
  "numeroDocumento": "1888888888888",
  "primerNombre": "Carlos",
  "segundoNombre": "Mario",
  "primerApellido": "Caceres",
  "segundoApellido": "Abudinen",
  "email": "carlos@gmail.com",
  "direccion": "calle 5",
  "fechaNacimiento": 30,
  "telefono": "123456",
  "_v": 0 }
```

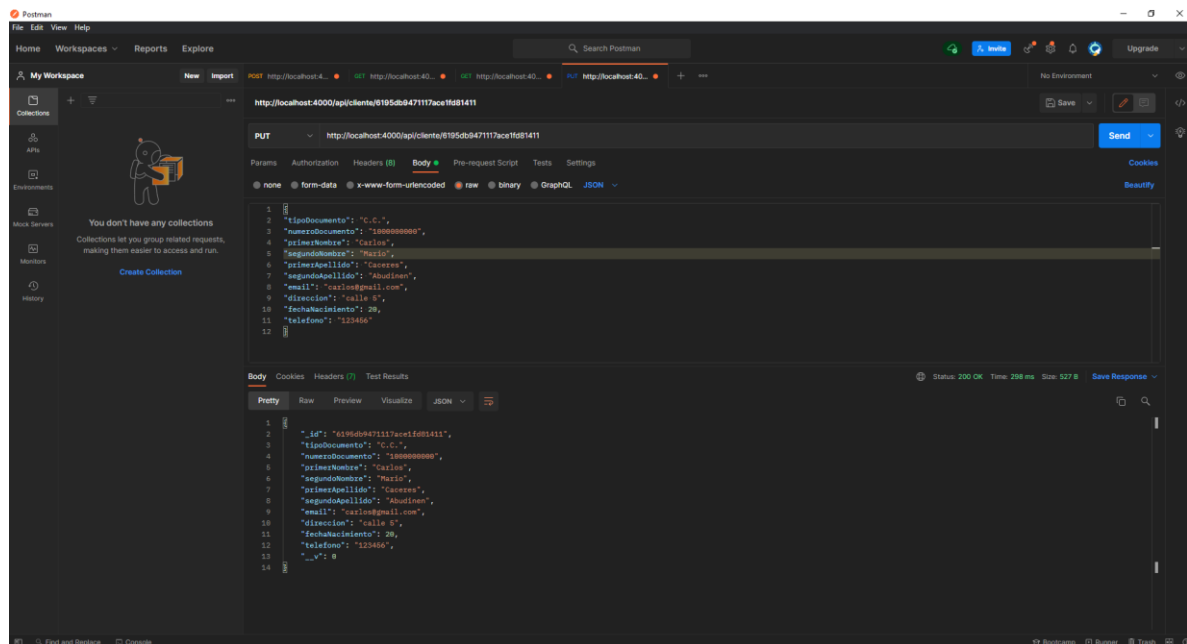
## Prueba GET por id:



The screenshot shows the Postman interface with a GET request to `http://localhost:4000/api/cliente/6195db947117ace1f681411`. The response is a JSON object with the following structure:

```
1 {
2   "id": "6195db947117ace1f681411",
3   "tipoDocumento": "C.C.",
4   "numeroDocumento": "1989888888",
5   "primerNombre": "Dora",
6   "segundoNombre": "Andres",
7   "primerApellido": "Caceres",
8   "segundoApellido": "Abudinen",
9   "email": "dura@gmail.com",
10  "direccion": "calle 6",
11  "fechaNacimiento": 26,
12  "telefono": "123456",
13  "_v": 0
14 }
```

## Prueba PUT:



The screenshot shows the Postman interface with a PUT request to `http://localhost:4000/api/cliente/6195db947117ace1f681411`. The request body is a JSON object with the following structure:

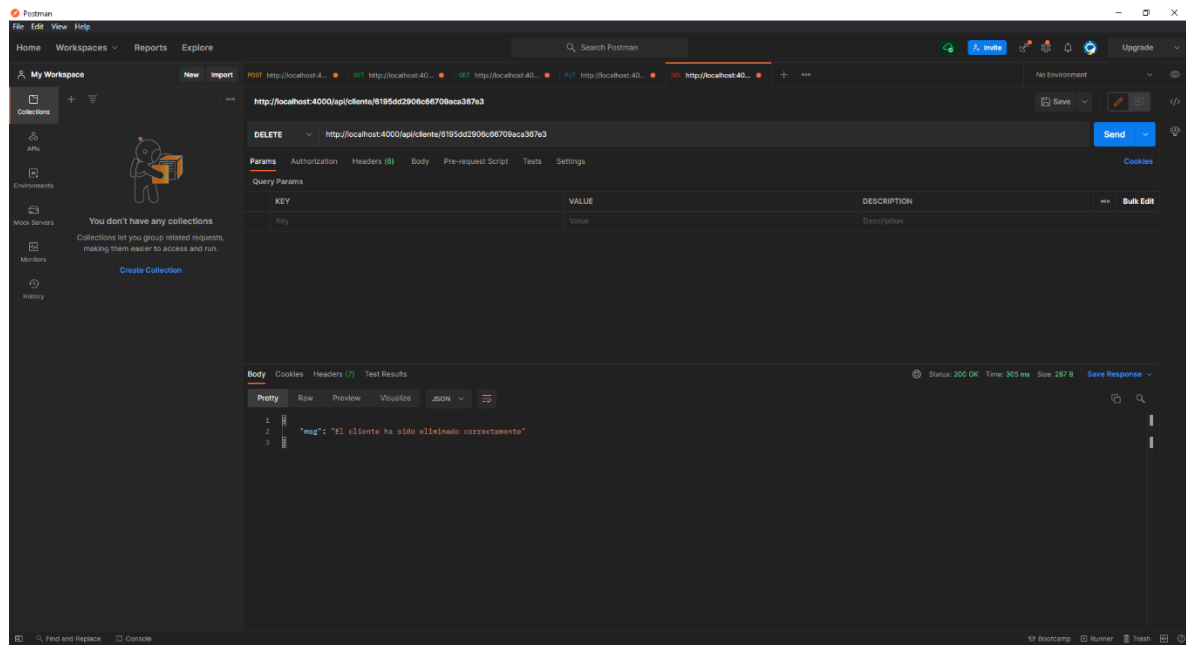
```
1 {
2   "tipoDocumento": "C.C.",
3   "numeroDocumento": "1989888888",
4   "primerNombre": "Carlos",
5   "segundoNombre": "Mario",
6   "primerApellido": "Caceres",
7   "segundoApellido": "Abudinen",
8   "email": "carlos@gmail.com",
9   "direccion": "calle 6",
10  "fechaNacimiento": 26,
11  "telefono": "123456"
12 }
```

The response is a JSON object with the following structure:

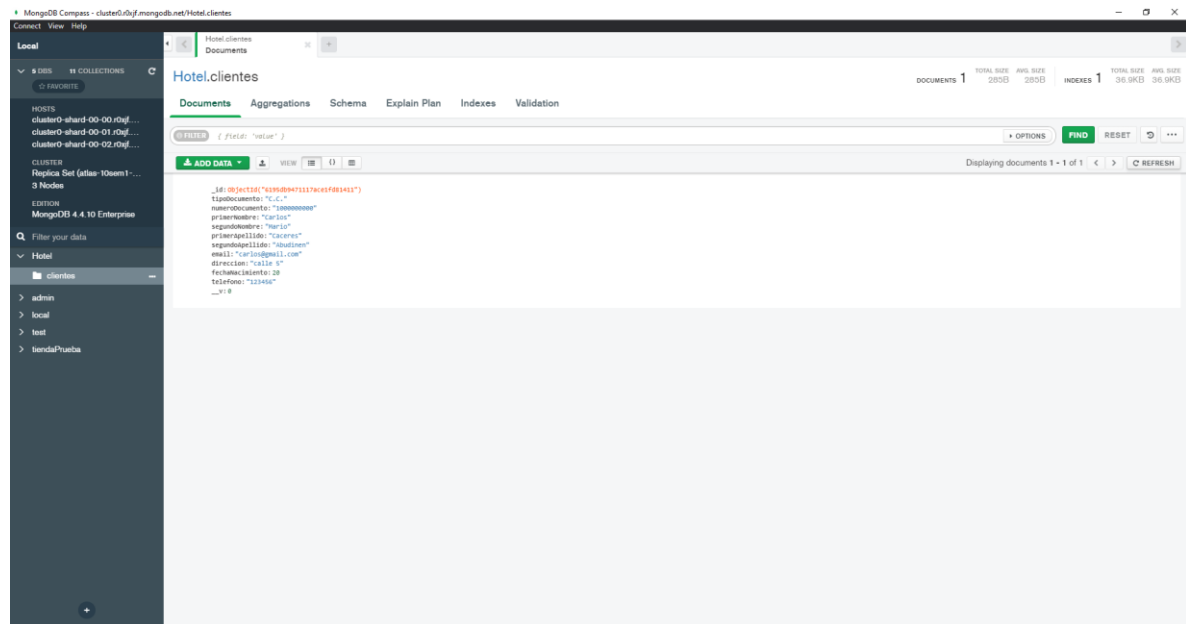
```
1 {
2   "id": "6195db947117ace1f681411",
3   "tipoDocumento": "C.C.",
4   "numeroDocumento": "1989888888",
5   "primerNombre": "Carlos",
6   "segundoNombre": "Mario",
7   "primerApellido": "Caceres",
8   "segundoApellido": "Abudinen",
9   "email": "carlos@gmail.com",
10  "direccion": "calle 6",
11  "fechaNacimiento": 26,
12  "telefono": "123456",
13  "_v": 0
14 }
```



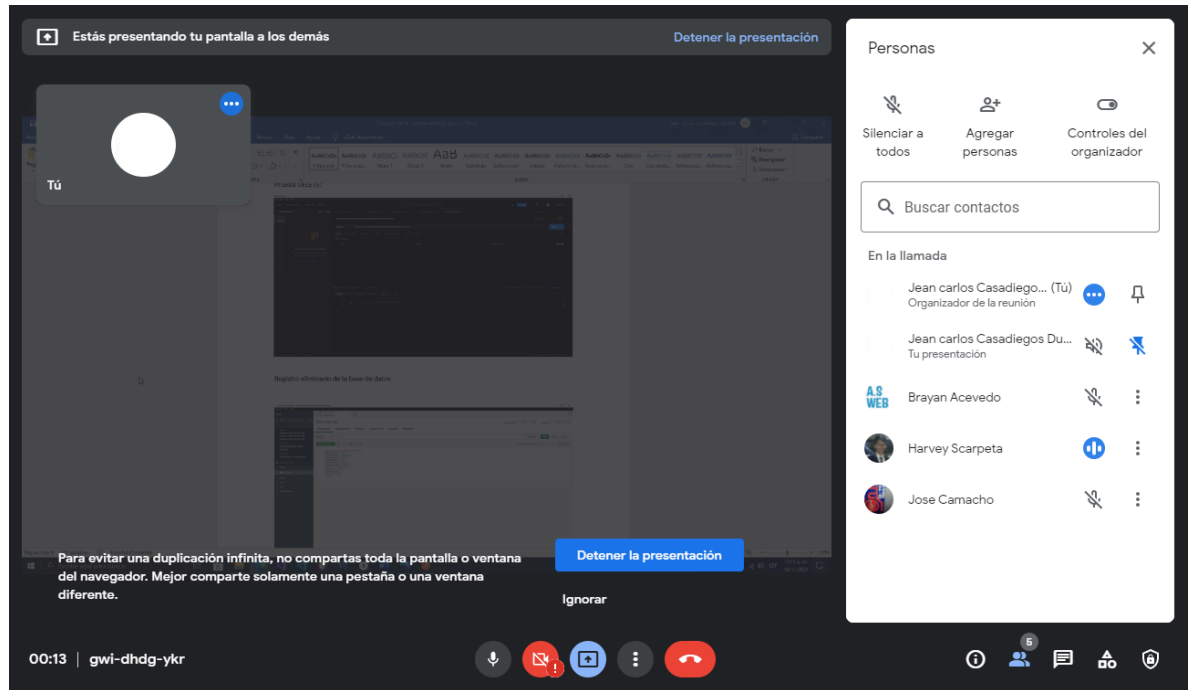
## Prueba DELETE:



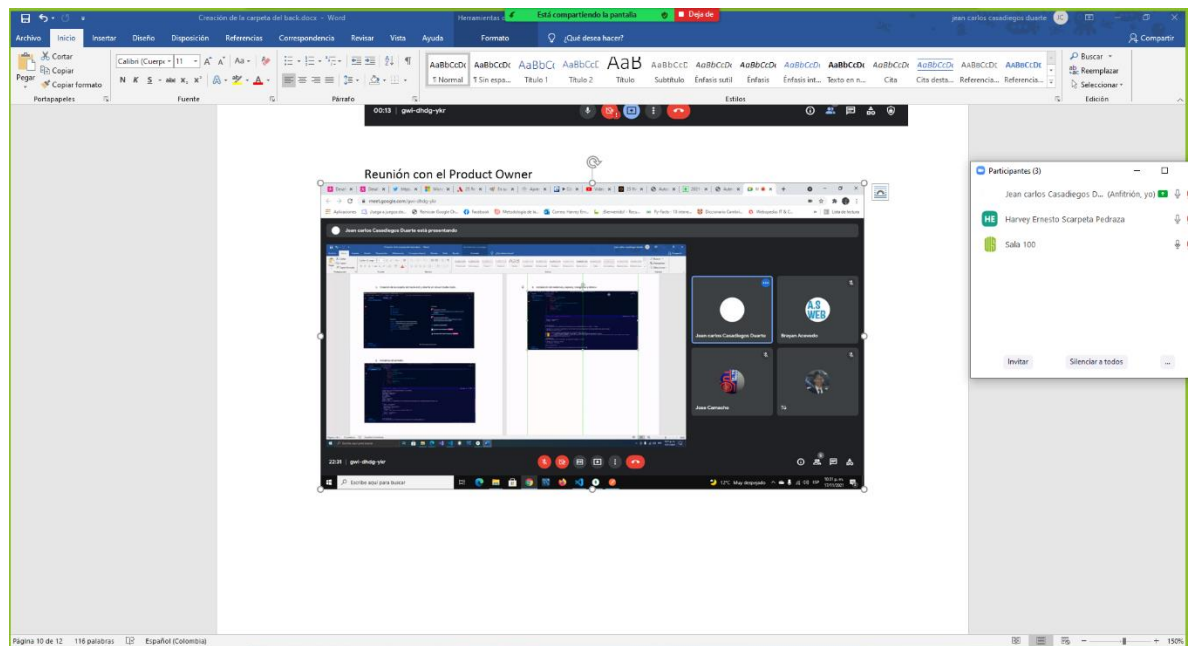
## Registro eliminado de la base de datos



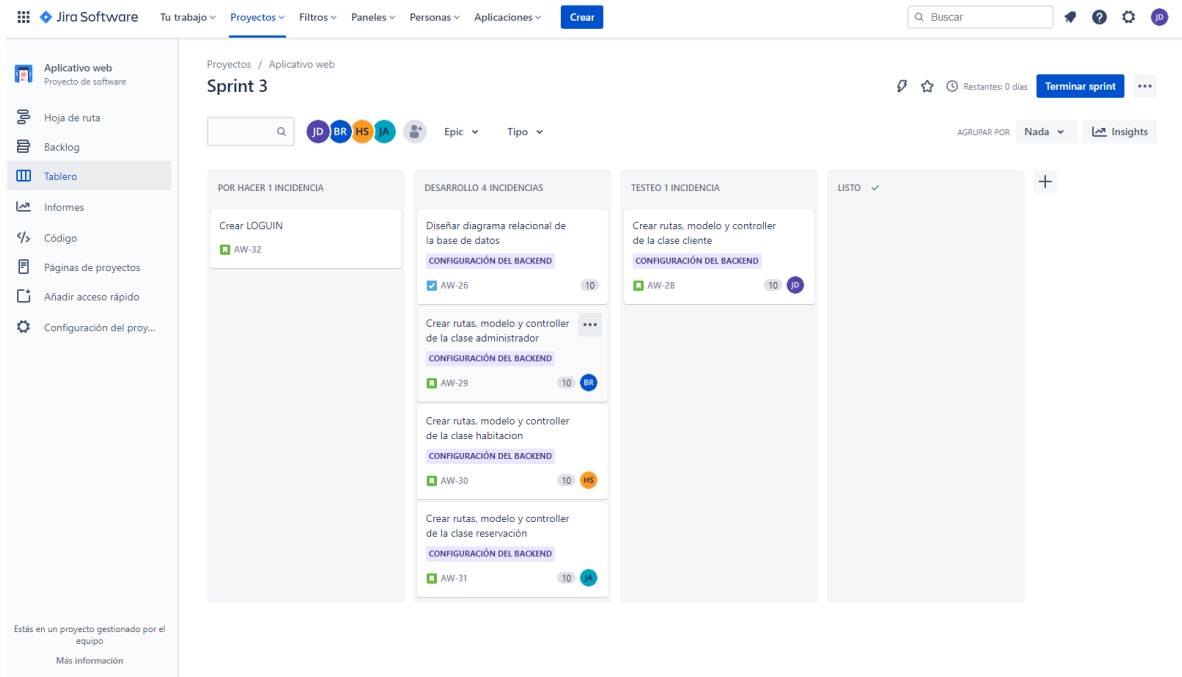
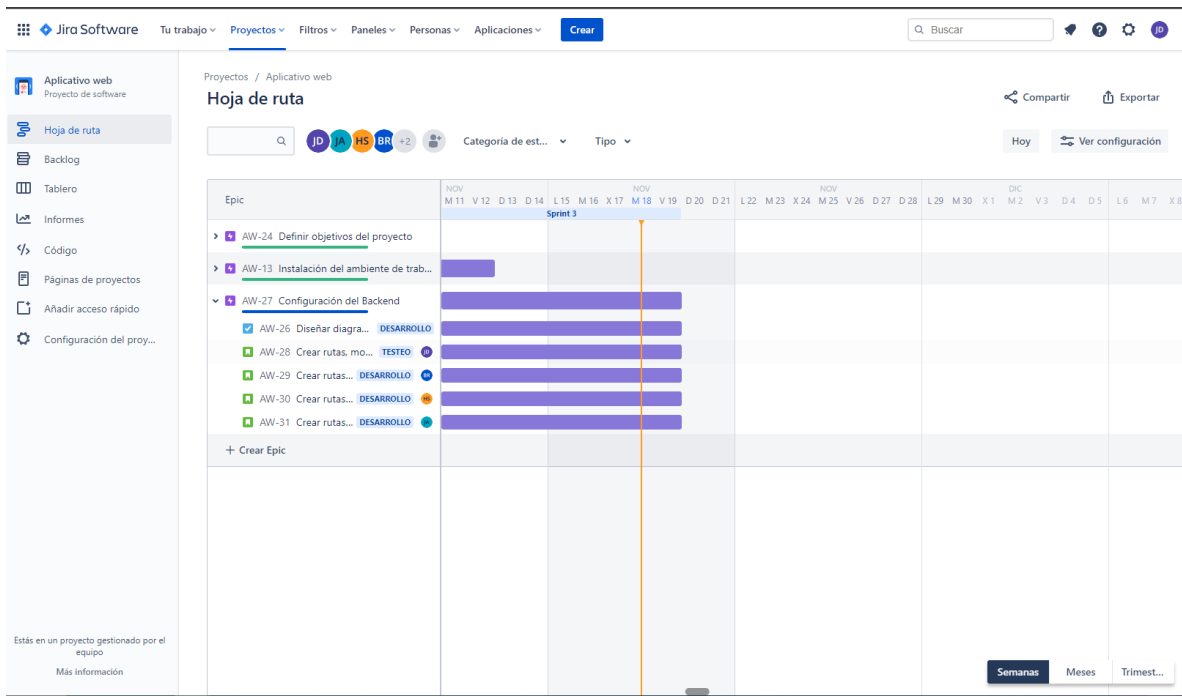
## Reunión de grupo:



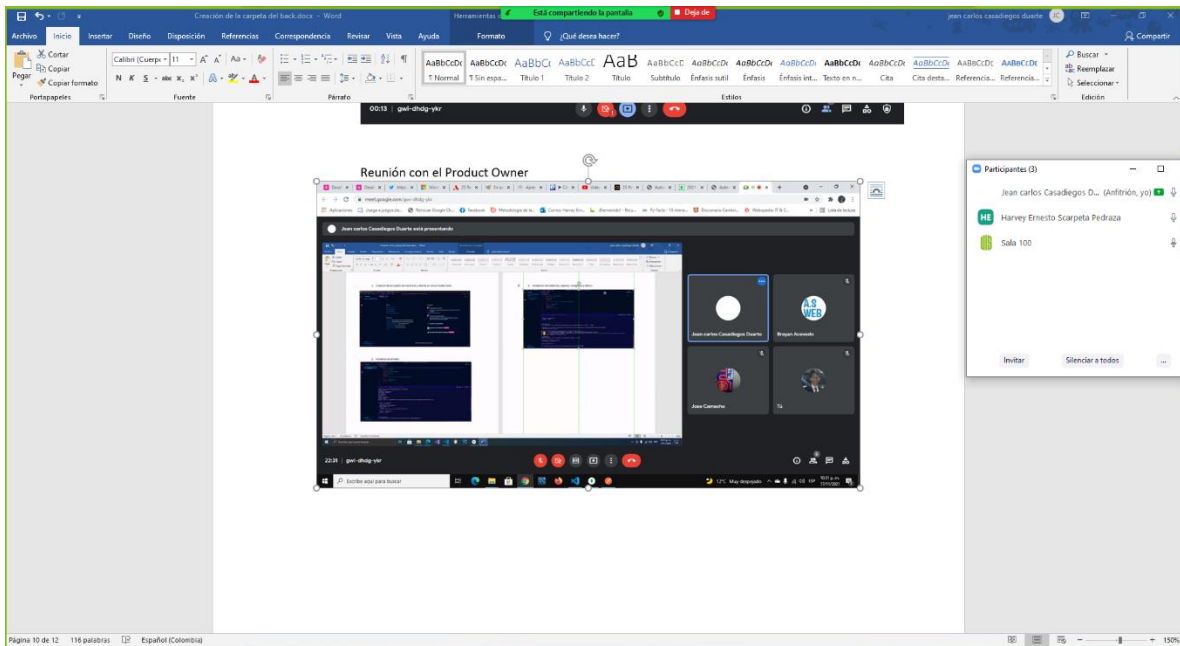
## Reunión con el Product Owner



## Plataforma JIRA:



## Reunión con el tutor:



Repositorio de GIT: <https://github.com/SrJean97/hotelApp.git>

