

# Manual Técnico

Para el desarrollo del proyecto se desarrolló un frontend y backend en donde se creó y empleo el uso de una API (Application Programming Interface) para que estos dos lados se puedan comunicar y pasar información, del lado del backend levantamos un servidor en el siguiente archivo

```
backend > src > js server.js > ...
1  const express = require('express');           // Importando express
2  const app = express();                         // Inicializando express que va a ser nuestro servidor
3  const cors = require('cors');                 // Importando cors para poder hacer peticiones desde el frontend
4
5  // Configuraciones
6  app.use(express.json());
7  app.use(cors());
8
9  // Mis rutas
10 const rutasUsuarios = require('./rutas/usuarios');
11
12 app.set('port', process.env.PORT || 4000);     // Configurando el puerto que vamos a utilizar
13
14 app.get('/', (req, res) => {                    // Ruta inicial de mi servidor
15   res.send('Servidor funcionando y listo para recibir peticiones en puerto 4000');
16 });
17
18 app.use('/usuarios', rutasUsuarios);
19
20 app.listen(app.get('port'), () => {             // Inicializando el servidor, aqui estoy levantando el servidor
21   console.log('Server on port ${app.get('port')}');
22 });
23
24 module.exports = app;                          // Exportando mi servidor para poder utilizarlo en mis pruebas unitarias
```

Definimos un número de rutas para que se puedan comunicar nuestro frontend al backend por medio de peticiones http las cuales nos permiten pedir información a la API

```
backend > src > rutas > js usuarios.js > ...
1  const express = require('express');           // Importando express
2  const router = express.Router();               // Inicializando express y Router sera el encargado de las rutas
3
4
5  // Controlador
6  const controlador = require('../controladores/usuarios');
7
8  // Ruta para traer usuarios
9  router.get('/', controlador.leerUsuarios);
10 router.get('/citass', controlador.leerCitas);
11 router.get('/medicinas', controlador.leerMedicinas);
12 router.get('/medicinasreportes', controlador.leerMedicinasReportes);
13 router.get('/doctoresreportes', controlador.leerDoctoresReportes);
14
15 // Ruta para crear usuarios
16 router.post('/', controlador.crearUsuario);
17 router.post('/citass', controlador.crearCita);
18
19 // Ruta para editar usuarios
20 router.put('/:id', controlador.actualizarUsuario); // :id es un parametro que se envia por la url el cual indica el id del usu
21 router.put('/citass/:id', controlador.actualizarCita);
22 router.put('/medicinas/:id', controlador.actualizarMedicina);
23 router.put('/medicinasreportes/:id', controlador.actualizarMedicinaReportes);
24 router.put('/doctoresreportes/:id', controlador.actualizarDoctoresReportes);
25
26 // Ruta para eliminar usuarios
27 router.delete('/:id', controlador.eliminarUsuario); // :id es un parametro que se envia por la url el cual indica el id del usu
28 router.delete('/citass/:id', controlador.eliminarCita);
29
30
31 module.exports = router;                        // Exportando mi servidor para poder utilizarlo en mis pruebas unitarias
```

Definimos los métodos que queremos que nuestra API haga como las operaciones CRUD (crear, leer, modificar y eliminar) para cada uno de los archivos que queremos guardar, estos archivos son de tipo binario

```
frontend > src > libs > http.js > put
1 // En este archivo definimos las operaciones CRUD para la entidad "usuarios" y las rutas para cada operación, en este caso solo tenemos una
2
3 const http = {}; // Creando mi objeto http que contendrá las funciones para hacer peticiones
4
5 http.get = async (url) => {
6   try {
7     const response = await fetch(url);
8     const data = await response.json();
9     return data;
10  } catch (error) {
11    console.log('Error en la petición GET 7771: ', error);
12    return null;
13  }
14 }
15
16
17 http.post = async (url, data) => {
18   try {
19     const response = await fetch(url, {
20       method: "POST",
21       headers: {
22         "Content-Type": "application/json",
23       },
24       body: JSON.stringify(data),
25     });
26     return response;
27   } catch (error) {
28     console.log('Error en la petición POST 777: ', error);
29     return null;
30   }
31 }
32
```

Definimos las funciones y sus parámetros que se tienen que enviar para ejecutar una petición al backend para que luego esta modifique los archivos binario y nos de una respuesta

```
// Controlador para citas y operaciones CRUD
✓ controlador.crearCita = (req, res) => {
  const data = req.body;

  ✓ if (fs.existsSync('citas.bin')) {
    const buffer = fs.readFileSync('citas.bin');
    const dataAnterior = JSON.parse(buffer.toString());

    dataAnterior.push(data);
    const bufferNuevo = Buffer.from(JSON.stringify(dataAnterior));

    fs.writeFileSync('citas.bin', bufferNuevo);
    return res.send('Creando citas desde el controlador si existe');
  }

  const arreglo = [];
  arreglo.push(data);

  const buffer = Buffer.from(JSON.stringify(arreglo));
  fs.writeFileSync('citas.bin', buffer);

  return res.send('Creando citas desde el controlador');
}
```

Usamos la extensión de VSCode ‘Thunder Client’ para probar si nuestras peticiones devuelven el formato correcto el cual tiene que ser en formato JSON

GET ⌵ http://localhost:4000/usuarios Send

Query Headers <sup>2</sup> Auth Body <sup>1</sup> Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "id": 7,
3   "nombre": "Fernando",
4   "password": "f"
5 }
```

Status: 200 OK Size: 1.57 KB Time: 27 ms

Response Headers <sup>7</sup> Cookies Results Docs

```
1 [
2   {
3     "id": 7,
4     "userType": "paciente",
5     "userName": "fernando",
6     "password": "f",
7     "nombre": "Fernando",
8     "apellido": "Orozco",
9     "fechaNacimiento": "2020-10-03",
10    "gender": "Masculino",
11    "phone": "12345678"
12  },
13  {
14    "id": 1,
15    "userType": "doctor",
16    "userName": "Dr. Reyes",
17    "password": "d",
18    "nombre": "Marco",
19    "apellido": "Reyes",
20    "fechaNacimiento": "2022-2-22",
21    "gender": "Masculino",
22    "phone": "12345678"
23  },
24  {
25    "id": 2,
26    "userType": "doctor",
27    "userName": "Dra. Salazar",
28    "password": "d2",
29    "nombre": "Samantha",
30    "apellido": "Salazar",
31    "fechaNacimiento": "2022-2-22",
32    "gender": "Femenino",
33    "phone": "12345678"
34  },
35  {
36    "id": 3,
37    "userType": "enfermera",
38    "userName": "enfe1",
39    "password": "e",
40    "nombre": "Brenda",
41    "apellido": "Castillo",
42    "fechaNacimiento": "2022-2-22",
43    "gender": "Femenino",
```