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 servidos en el siguiente archivo

backend > src > server.js > ...

```
const express = require('express');                    // Importando express
const app = express();                                 // Inicializando express que va a ser nuestro servidor
const cors = require('cors');                          // Importando cors para poder hacer peticiones desde el frontend

// Configuraciones
app.use(express.json());
app.use(cors());

// Mis rutas
const rutasUsuarios = require('./rutas/usuarios');

app.set('port', process.env.PORT || 4000);             // Configurando el puerto que vamos a utilizar

app.get('/', (req, res) => {                           // Ruta inicial de mi servidor
  res.send('Servidor funcionando y listo para recibir peticiones en puerto 4000');
});
                                                                                    ~/Desktop/UHospital/backend
app.use('/usuarios', rutasUsuarios);

app.listen(app.get('port'), () => {                    // Inicializando el servidor, aqui estoy levantanado el servidor
  console.log(`Server on port ${app.get('port')}`);
});

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 > usuarios.js > ...

```
const express = require('express');              // Importando express
const router = express.Router();                 // Inicializando express y Router sera el encargado de las rutas


//  Controlador
const controlador = require('../controladores/usuarios');

//  Ruta para traer usuarios
router.get('/', controlador.leerUsuarios);
router.get('/citas', controlador.leerCitas);
router.get('/medicinas', controlador.leerMedicinas);
router.get('/medicinasreportes', controlador.leerMedicinasReportes);
router.get('/doctoresreportes', controlador.leerDoctoresReportes);

//  Ruta para crear usuarios
router.post('/', controlador.crearUsuario);
router.post('/citas', controlador.crearCita);

//  Ruta para editar usuarios
router.put('/:id', controlador.actualizarUsuario);        // :id es un parametro que se envia por la url el cual indica el id del usu
router.put('/citas/:id', controlador.actualizarCita);
router.put('/medicinas/:id', controlador.actualizarMedicina);
router.put('/medicinasreportes/:id', controlador.actualizarMedicinaReportes);
router.put('/doctoresreportes/:id', controlador.actualizarDoctoresReportes);

//  Ruta para eliminar usuarios
router.delete('/:id', controlador.eliminarUsuario);       // :id es un parametro que se envia por la url el cual indica el id del usu
router.delete('/citas/:id', controlador.eliminarCita);


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 lo archivos que queremos guardar, estos archivos son de tipo binario

frontend > src > libs > http.js > put

```js
// En este archivo definimos las operacion CRUD para la entidad "usuarios" y las rutas para cada operación, en este caso solo tenemos una

const http = {};                          // Creando mi objeto http que contendra las funciones para hacer peticiones

http.get = async (url) => {
  try {
    const response = await fetch(url);
    const data = await response.json();
    return data;
  } catch (error) {
    console.log('Error en la peticion GET 7771: ', error)
    return null
  }
}


http.post = async (url, data) => {
  try {
    const response = await fetch(url, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });
    return response;
  } catch (error) {
    console.log('Error en la peticion POST 777: ', error)
    return null
  }
}
```

~/Desktop/UHospit

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

```js
//  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