Ariel Alejandro Bautista M.
Guatemala, octubre 29 de 2020

Manual Técnico

Prueba técnica para los postulantes a la Consultoría

A continuación, se describe de forma detallada lo realizado durante la prueba técnica que solicitaba un CRUD de fiscalías del Ministerio Público en donde era necesario conocer su ubicación, así como su información de contacto, utilizando el entorno Node.Js para la capa del servidor, ReacJS como librería para realizar la interfaz del usuario y SQL Server Express como base de datos para garantizar la integridad de la información.

Seguidamente, se detallarán los paquetes utilizados en Node.Js, así como la versión de estos para tener una idea más general de los recursos empleados para la elaboración de esta aplicación.

- body-parser v.1.19.0
  Analiza el cuerpo de las solicitudes entrantes en un middleware antes que los controladores, disponible en la propiedad req.body.

- cors: v.2.8.5
  CORS es un paquete node.js para proporcionar un middleware Connect / Express que se puede usar para habilitar CORS con varias opciones.

- express: v.4.17.1
  Marco web minimalista, rápido y sin supervisión para node. La filosofía de Express es proporcionar herramientas pequeñas y sólidas para servidores HTTP, lo que la convierte en una gran solución para aplicaciones de una sola página, sitios web, híbridos o API HTTP públicas.

- mssql: v.6.2.3
  Cliente de Microsoft SQL Server para Node.js

- nodemon: v.2.0.6
  nodemon es una herramienta que ayuda a desarrollar aplicaciones basadas en node.js al reiniciar automáticamente la aplicación de nodo cuando se detectan cambios de archivo en el directorio.

Diagrama Entidad Relación

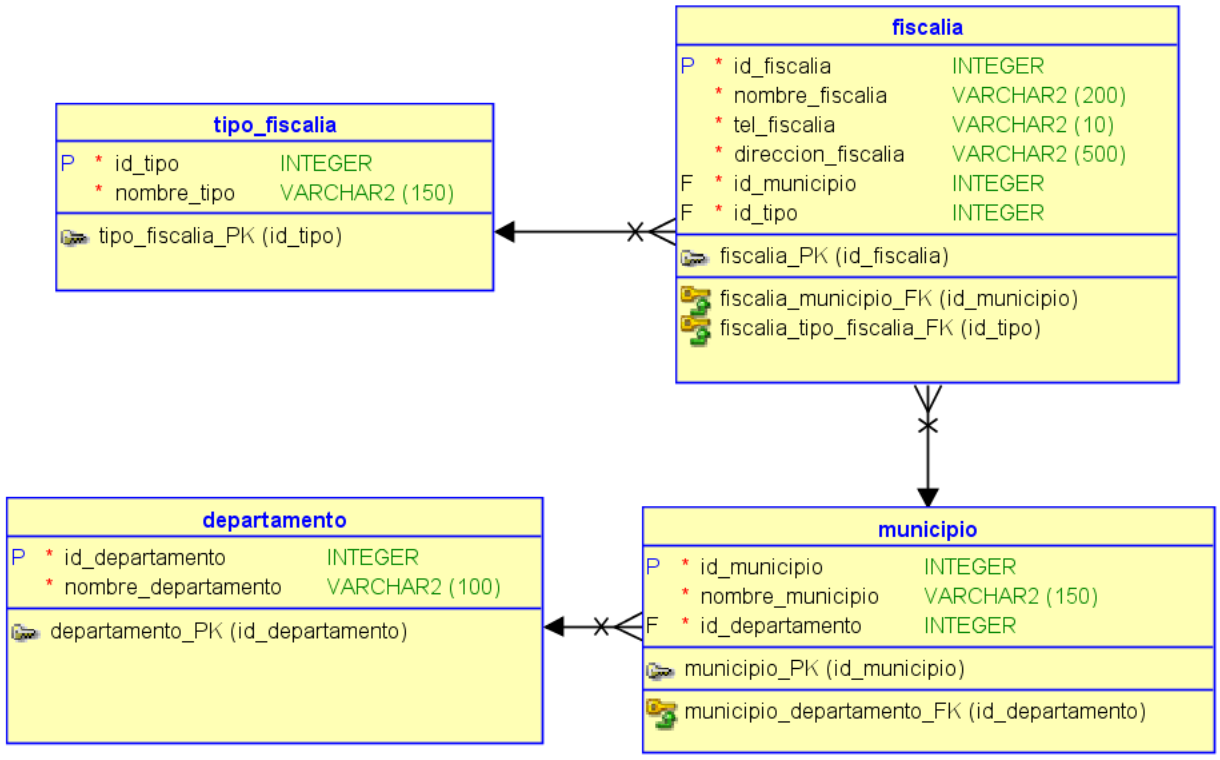


Tabla Fiscalía:

En este caso el núcleo de nuestro modelo es la tabla **"fiscalía"** que cuenta con los atributos de: id, nombre, número de teléfono, dirección, así como el identificador del municipio y el identificador de tipo de fiscalía, ambas llaves foráneas de otras tablas que se explicarán a continuación.

Tabla de Tipos de Fiscalías:

Se creó una tabla para almacenar los distintos tipos de fiscalías existentes con posibilidad a ampliarse sin ningún inconveniente. En este caso la tabla solo cuenta con dos atributos, un identificador y la descripción del tipo como tal.

Tabla Departamento:

Dado que es importante conocer la ubicación geográfica de las fiscalías se agregó esta tabla para organizar a los municipios por departamento, en este caso la tabla cuenta con los atributos de identificador y nombre del departamento.

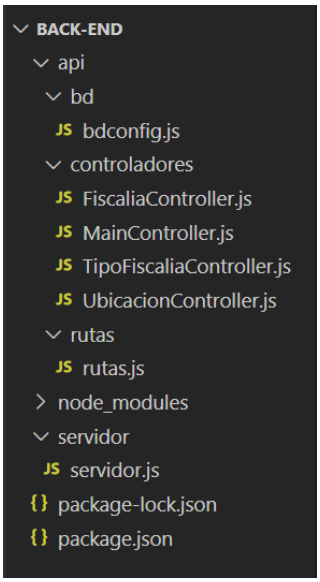
Tabla Municipio:

Finalmente se creó una tabla municipio, la cual contiene un atributo que la relaciona con la tabla departamento, para finalmente asociarse a la tabla fiscalía en donde asocia una fiscalía con un municipio en concreto. Además, cuenta con los atributos de identificador y nombre del municipio.

Back-End

Estructura

Para la parte del servidor, se distribuyeron los archivos en dos carpetas, la carpeta **“api”** y la carpeta **“servidor”**. A su vez, la carpeta **“api”** se desglosa en tres directorios importantes, los cuales son **“bd”**, **“controladores”** y **“rutas”**



servidor/servidor.js

Este archivo es el que se encarga de poner al servidor de pie, en él encontramos las importaciones a los módulos importantes como express, body-parser, cors entre otros. Es acá donde también se le coloca el número de puerto en donde el servidor escuchará las peticiones. En las líneas 6 y 7 se realizan las configuraciones para que express use body-parser para procesar el cuerpo de las peticiones. A partir de la línea 9 hasta la 14 se configuran los cors necesarios para que no nos impida conectarnos a la aplicación. En la línea 16 creamos una variable rutas que es donde se encuentran los endpoints de nuestra aplicación y en la línea 17 se le envía por parámetro la aplicación de express para que pueda hacer uso de sus configuraciones. Finalmente, en la línea 19 a la 20 se hace el llamado para levantar el servidor.

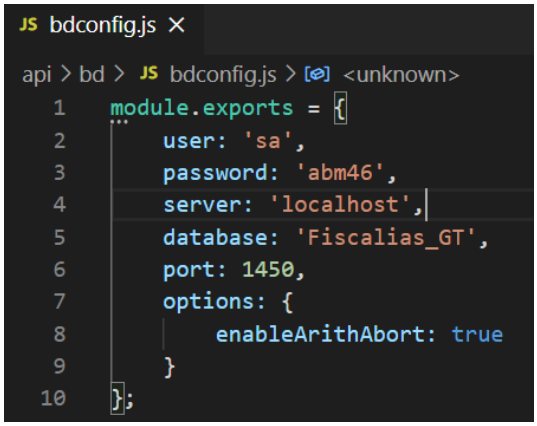
servidor.js

servidor > servidor.js

```
1  var express = require('express');
2  var app = express();
3  var bodyParser = require('body-parser');
4  var port = process.env.PORT || 3000;
5
6  app.use(bodyParser.urlencoded({ extended: true }));
7  app.use(bodyParser.json());
8
9  const cors = require('cors');
10 var corsOptions = {
11     origin: '*',
12     optionsSuccessStatus: 200
13 }
14 app.use(cors(corsOptions));
15
16 var routes = require('../api/rutas/rutas.js');
17 routes(app);
18
19 app.listen(port, () => {
20     console.log(`El servidor está corriendo en: http://localhost:${port}`);
21 });
```

api/bd/bdconfig.js

Este archivo contiene los parámetros de configuración para poder conectarnos a nuestra base de datos, en este caso SQL SERVER EXPRESS



api/controladores/FiscaliaController.js

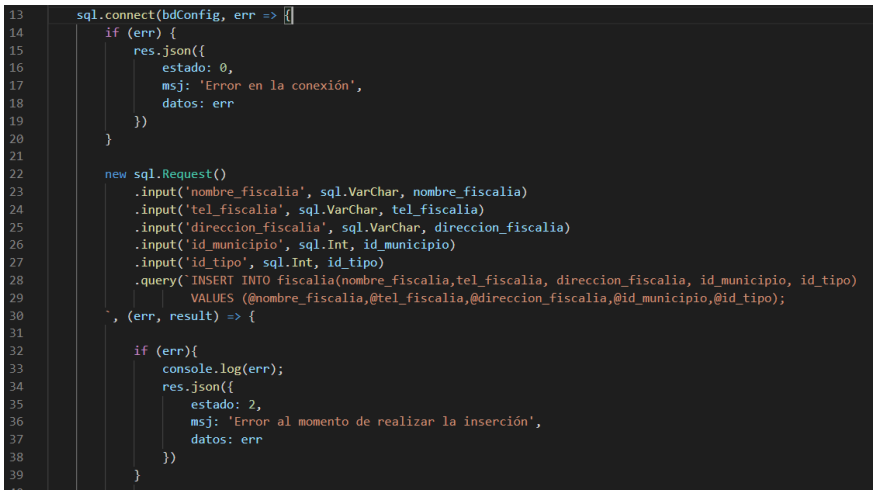
Dentro de este archivo se configuran las acciones de consulta, eliminación, creación y modificación de la información de las fiscalías, en este caso contamos con las siguientes funciones:

- crearFiscalia:

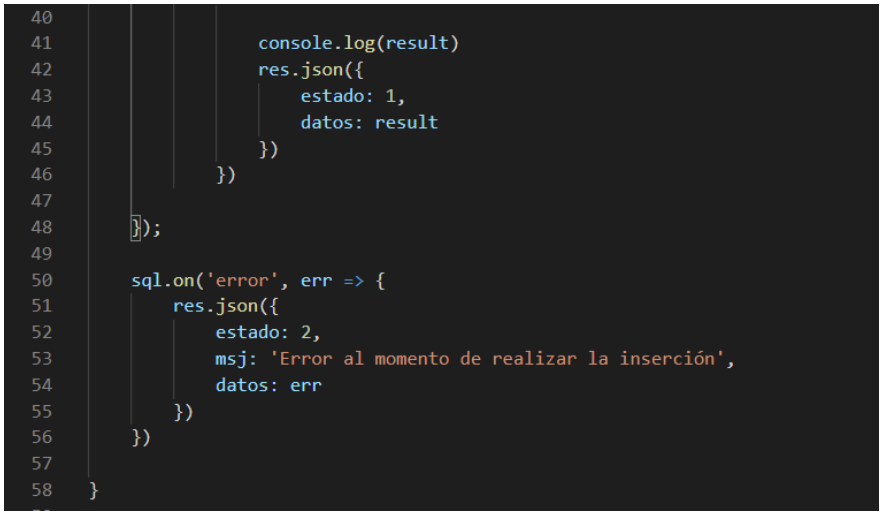
  - ✓ **Endpoint:** /crearFiscalia
  - ✓ **Tipo:** POST
  - ✓ **Parámetros:** nombre_fiscalia, tel_fiscalia, direccion_fiscalia, id_municipio, id_tipo
  - ✓ **Descripción:** En esta función se hace el llamado a la creación de una nueva fiscalía, se utilizan los parámetros para pasárselos a la base de datos y que esta cree una nueva fiscalía utilizando los mismos.
  - ✓ **Respuestas:** En condiciones normales, es decir cuando se lleva a cabo la inserción correctamente, se retornan dos parámetros, uno llamado **"estado"** que tendrá el valor de **"1"** y en el parámetro **"datos"** devolverá los resultados de la consulta.

    En situaciones de error, el parámetro **"estado"** será **"0"** o **"2"** en donde el primero indica un error al momento de conectar con la bd, y el segundo nos indicará que existió un error mientras se realizaba la consulta.

```
JS FiscaliaController.js ×
api > controladores > JS FiscaliaController.js > crearFiscalia > crearFiscalia > sql.connect() callback
1   var sql = require('mssql')
2   var bdConfig = require('../bd/bdconfig')
3
4
5   exports.crearFiscalia = function (req, res) {
6
7       var nombre_fiscalia = req.body.nombre_fiscalia;
8       var tel_fiscalia = req.body.tel_fiscalia;
9       var direccion_fiscalia = req.body.direccion_fiscalia;
10      var id_municipio = req.body.id_municipio;
11      var id_tipo = req.body.id_tipo;
12
```



```
13      sql.connect(bdConfig, err => {
14          if (err) {
15              res.json({
16                  estado: 0,
17                  msj: 'Error en la conexión',
18                  datos: err
19              })
20          }
21
22          new sql.Request()
23              .input('nombre_fiscalia', sql.VarChar, nombre_fiscalia)
24              .input('tel_fiscalia', sql.VarChar, tel_fiscalia)
25              .input('direccion_fiscalia', sql.VarChar, direccion_fiscalia)
26              .input('id_municipio', sql.Int, id_municipio)
27              .input('id_tipo', sql.Int, id_tipo)
28              .query(`INSERT INTO fiscalia(nombre_fiscalia,tel_fiscalia, direccion_fiscalia, id_municipio, id_tipo)
29                      VALUES (@nombre_fiscalia,@tel_fiscalia,@direccion_fiscalia,@id_municipio,@id_tipo);
30          `, (err, result) => {
31
32              if (err){
33                  console.log(err);
34                  res.json({
35                      estado: 2,
36                      msj: 'Error al momento de realizar la inserción',
37                      datos: err
38                  })
39              }
```



```
40
41                      console.log(result)
42                      res.json({
43                          estado: 1,
44                          datos: result
45                      })
46                  })
47
48          });
49
50          sql.on('error', err => {
51              res.json({
52                  estado: 2,
53                  msj: 'Error al momento de realizar la inserción',
54                  datos: err
55              })
56          })
57
58      }
```

- verFiscalia:

  - ✓ **Endpoint:** /verFiscalia/:id_fiscalia
  - ✓ **Tipo:** GET
  - ✓ **Parámetros:** id_fiscalia
  - ✓ **Descripción:** Esta función recibe por parámetro el id de una fiscalía y devuelve los detalles de esta.

- ✓ **Respuestas:** Si la consulta resulta exitosa, el parámetro **"estado"** tiene el valor de 1 y retorna en el parámetro **"datos"** la información solicitada

  En situaciones de error, el parámetro **"estado"** será **"0"** o **"2"** en donde el primero indica un error al momento de conectar con la bd, y el segundo nos indicará que existió un error mientras se realizaba la consulta. En Ambos casos el parámetro **"datos"** contiene información sobre el error.

```
exports.verFiscalia = function (req, res) {

    var id_fiscalia = req.params.id_fiscalia;

    sql.connect(bdConfig, err => {
        if (err) {
            res.json({
                estado: 0,
                msj: 'Error en la conexión',
                datos: err
            })
        }

        new sql.Request()
            .input('id_fiscalia', sql.Int, id_fiscalia)
            .query(`SELECT d.id_departamento, d.nombre_departamento, m.id_municipio, m.nombre_municipio, f.id_fiscalia, f.nombre_
                    FROM fiscalia f, tipo_fiscalia t, municipio m, departamento d
                    WHERE t.id_tipo = f.id_tipo
                    AND m.id_municipio = f.id_municipio
                    AND d.id_departamento = m.id_departamento
                    AND f.id_fiscalia = @id_fiscalia;`,
                (err, result) => {
                    res.json({
                        estado: 1,
                        datos: result['recordset']
                    })
                })

    });

    sql.on('error', err => {
        res.json({
            estado: 2,
            msj: 'Error al momento de realizar la consulta',
            datos: err
        })
    })

}
```

- obtenerFiscalias:

  - ✓ **Endpoint:** /obtenerFiscalias
  - ✓ **Tipo:** GET.
  - ✓ **Parámetros:** Ninguno.
  - ✓ **Descripción:** Retorna todas las fiscalías registradas en el sistema.
  - ✓ **Respuestas:** Si la consulta resulta exitosa, el parámetro **"estado"** tiene el valor de 1 y retorna en el parámetro **"datos"** la información solicitada

    En situaciones de error, el parámetro **"estado"** será **"0"** o **"2"** en donde el primero indica un error al momento de conectar con la bd, y el segundo nos indicará que existió un error mientras se realizaba la consulta. En Ambos casos el parámetro **"datos"** contiene información sobre el error.

```
JS rutas.js   JS FiscaliaController.js ●
api > controladores > JS FiscaliaController.js > verFiscalia > verFiscalia
102  exports.obtenerFiscalias = function (req, res) {
103
104      sql.connect(bdConfig, err => {
105          if (err) {
106              res.json({
107                  estado: 0,
108                  msj: 'Error en la conexión',
109                  datos: err
110              })
111          }
112
113          new sql.Request()
114              .query(`SELECT d.id_departamento, d.nombre_departamento, m.id_municipio, m.nombre_municipio, f.id_fiscalia, f.nomb
115                      FROM fiscalia f, tipo_fiscalia t, municipio m, departamento d
116                      WHERE t.id_tipo = f.id_tipo
117                      AND m.id_municipio = f.id_municipio
118                      AND d.id_departamento = m.id_departamento;`,
119                  (err, result) => {
120                      res.json({
121                          estado: 1,
122                          datos: result['recordset']
123                      })
124                  })
125      });
126
127      sql.on('error', err => {
128          res.json({
129              estado: 2,
130              msj: 'Error al momento de realizar la consulta',
131              datos: err
132          })
133      })
134  }
```

- modificarFiscalia:

  - ✓ **Endpoint:** /modificarFiscalia
  - ✓ **Tipo:** PUT.
  - ✓ **Parámetros:** id_fiscalia, nombre_fiscalia, tel_fiscalia, direccion_fiscalia, id_municipio, id_tipo.
  - ✓ **Descripción:** Actualiza el registro de una fiscalía con el id señalado en los parámetros.
  - ✓ **Respuestas:** Si la modificación resulta exitosa, el parámetro **“estado”** tiene el valor de 1 y retorna en el parámetro **“datos”** los resultados de la actualización

    En situaciones de error, el parámetro **“estado”** será **“0”** o **“2”** en donde el primero indica un error de conexión, y el segundo nos indicará que existió un error mientras se realizaba la consulta. En Ambos casos el parámetro **“datos”** contiene información sobre el error.

```
138  exports.modificarFiscalia = function (req, res) {
139
140      var id_fiscalia = req.body.id_fiscalia
141      var nombre_fiscalia = req.body.nombre_fiscalia;
142      var tel_fiscalia = req.body.tel_fiscalia;
143      var direccion_fiscalia = req.body.direccion_fiscalia;
144      var id_municipio = req.body.id_municipio;
145      var id_tipo = req.body.id_tipo;
146
147      sql.connect(bdConfig, err => {
148          if (err) {
149              res.json({
150                  estado: 0,
151                  msj: 'Error en la conexión',
152                  datos: err
153              })
154          }
```

- eliminarFiscalia:

  - ✓ **Endpoint:** /eliminarFiscalia/id_fiscalia
  - ✓ **Tipo:** DELETE.
  - ✓ **Parámetros:** id_fiscalia
  - ✓ **Descripción:** Elimina el registro con el id señalado en los parámetros.
  - ✓ **Respuestas:** Si la eliminación resulta exitosa, el parámetro **"estado"** tiene el valor de 1 y retorna en el parámetro **"datos"** los resultados de la actualización

    En situaciones de error, el parámetro **"estado"** será **"0"** o **"2"** en donde el primero indica un error de conexión, y el segundo nos indicará que existió un error mientras se realizaba la consulta. En Ambos casos el parámetro **"datos"** contiene información sobre el error.

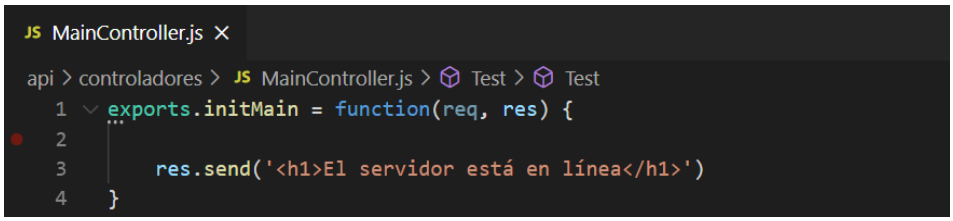
```
exports.eliminarFiscalia = function (req, res) {

    var id_fiscalia = req.params.id_fiscalia;

    sql.connect(bdConfig, err => {
        if (err) {
            res.json({
                estado: 0,
                msj: 'Error en la conexión',
                datos: err
            })
        }
```

```
        new sql.Request()
            .input('id_fiscalia', sql.Int, id_fiscalia)
            .query(`DELETE FROM fiscalia
                    WHERE id_fiscalia=@id_fiscalia;`,
                (err, result) => {
                    if (err){
                        console.log(err);
                        res.json({
                            estado: 2,
                            msj: 'Error al momento de realizar la consulta',
                            datos: err
                        })
                    }
                    res.json({
                        estado: 1,
                        datos: result['recordset']
                    })
                })

    });
```

```
    sql.on('error', err => {
        res.json({
            estado: 2,
            msj: 'Error al momento de realizar la consulta',
            datos: err
        })
    })

}
```

api/controladores/MainController.js

Archivo con la lógica de la pantalla de inicio para que al levantar el servidor muestre el mensaje “El servidor está en línea” en el puerto indicado.



api/controladores/TipoFiscaliaController.js

En este archivo se encuentra el método para obtener todos los tipos de fiscalías que han sido registrado en el sistema.

- obtenerTipo:

  - ✓ **Endpoint:** /tipoFiscalia
  - ✓ **Tipo:** GET.
  - ✓ **Parámetros:** Ninguno
  - ✓ **Descripción:** Obtiene todos los tipos de fiscalías que han sido agregados al sistema.
  - ✓ **Respuestas:** Si la consulta resulta exitosa, el parámetro **“estado”** tiene el valor de 1 y retorna en el parámetro **“datos”** la información consultada.

    En situaciones de error, el parámetro **“estado”** será **“0”** o **“2”** en donde el primero indica un error de conexión, y el segundo nos indicará que existió un error mientras se realizaba la consulta. En Ambos casos el parámetro **“datos”** contiene información sobre el error.

api/controladores/UbicacionController.js

En este archivo se encuentran los métodos para obtener la información de los departamentos y los municipios según el identificador de un determinado departamento

- obtenerDeptos:

  - ✓ **Endpoint:** /departamentos
  - ✓ **Tipo:** GET.
  - ✓ **Parámetros:** Ninguno
  - ✓ **Descripción:** Obtiene todos los departamentos que han sido agregados al sistema.
  - ✓ **Respuestas:** Si la consulta resulta exitosa, el parámetro **"estado"** tiene el valor de 1 y retorna en el parámetro **"datos"** la información consultada.

    En situaciones de error, el parámetro **"estado"** será **"0"** o **"2"** en donde el primero indica un error de conexión, y el segundo nos indicará que existió un error mientras se realizaba la consulta. En Ambos casos el parámetro **"datos"** contiene información sobre el error.

```
var sql = require('mssql')
var bdConfig = require('../bd/bdconfig')

exports.obtenerDeptos = function(req, res) {

    sql.connect(bdConfig, err => {
        if (err) {
            res.json({
                estado: 0,
                msj: 'Error en la conexión',
                datos: err
            })
        }

        new sql.Request().query('SELECT * FROM departamento;', (err, result) => {
            res.json({
                estado: 1,
                datos: result['recordset']
            })
        })
    });

    sql.on('error', err => {
        res.json({
            estado: 2,
            msj: 'Error al momento de realizar la consulta',
            datos: err
        })
    })
}
```

- obtenerMunicipios:

  - ✓ **Endpoint:** /municipios/: id_depto
  - ✓ **Tipo:** GET.
  - ✓ **Parámetros:** id_depto

- ✓ **Descripción:** Obtiene todos los municipios que pertenecen al departamento con el identificador indicado.

- ✓ **Respuestas:** Si la consulta resulta exitosa, el parámetro **"estado"** tiene el valor de 1 y retorna en el parámetro **"datos"** la información consultada.

  En situaciones de error, el parámetro **"estado"** será **"0"** o **"2"** en donde el primero indica un error de conexión, y el segundo nos indicará que existió un error mientras se realizaba la consulta. En Ambos casos el parámetro **"datos"** contiene información sobre el error.

```
exports.obtenerMunicipios = function(req, res) {

    var id_depto = req.params.id_depto;

    sql.connect(bdConfig, err => {
        if (err) {
            res.json({
                estado: 0,
                msj: 'Error en la conexión',
                datos: err
            })
        }

        new sql.Request()
            .input('id_depto', sql.Int, id_depto)
            .query(`SELECT m.id_municipio, m.nombre_municipio FROM municipio m
                WHERE m.id_departamento = @id_depto;
            `, (err, result) => {
                res.json({
                    estado: 1,
                    datos: result['recordset']
                })
            })
    });

    sql.on('error', err => {
        res.json({
            estado: 2,
            msj: 'Error al momento de realizar la consulta',
            datos: err
        })
    })
}
```

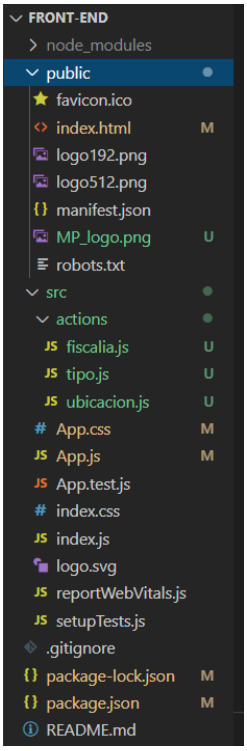
api/rutas/rutas.js

En este archivo se asigna un **"Endpoint"**, a cada controlador creado, así como también se asigna el tipo de petición que aceptará dicha función (POST, GET, DELETE, etc.):

```
module.exports = function(app) {

    var main = require('../controladores/MainController.js');
    var tipo_fiscalia = require('../controladores/TipoFiscaliaController.js');
    var ubicacion = require('../controladores/UbicacionController');
    var fiscalia = require('../controladores/FiscaliaController');

    app.route('/').get(main.initMain);

    app.route('/tipoFiscalia').get(tipo_fiscalia.obtenerTipo);
    app.route('/departamentos').get(ubicacion.obtenerDeptos);
    app.route('/municipios/:id_depto').get(ubicacion.obtenerMunicipios);
    app.route('/crearFiscalia').post(fiscalia.crearFiscalia);
    app.route('/obtenerFiscalias').get(fiscalia.obtenerFiscalias);
    app.route('/verFiscalia/:id_fiscalia').get(fiscalia.verFiscalia);
    app.route('/modificarFiscalia').put(fiscalia.modificarFiscalia);
    app.route('/eliminarFiscalia/:id_fiscalia').delete(fiscalia.eliminarFiscalia);
}
```

Front-End

Estructura



Para este caso los directorios fueron creados mediante la instrucción en consola: **“create-react-app front-end”**, creando la carpeta **“public”** en donde colocaremos nuestros archivos estáticos, y lo importante sería el index.html, que es donde inicia nuestra aplicación, sin embargo, si lo abrimos en un navegador sin estar corriendo en React nuestra página aparecerá vacía.

public/index.html

Parece un archivo html cualquiera, pero es el lugar en donde será renderizada nuestra interfaz de React, específicamente dentro de la etiqueta div cuyo id es **“root”**

```
public > index.html > html > body
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta name="description" content="Web site created using create-react-app" />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />

    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <title>Directorio de Fiscalías</title>
</head>

<body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div style="width: 100%;" id="root"></div>

    -->
</body>

</html>
```

src/app.js

Se podría decir que es el archivo más importante de ReactJS, ya que es aquí, el punto de partida de nuestra interfaz.

```
1
2  import React from "react";
3  import './App.css';
4  import 'bootstrap/dist/css/bootstrap.min.css'
5  import {
6      Table,
7      Button,
8      Container,
9      Modal,
10     ModalHeader,
11     ModalBody,
12     FormGroup,
13     ModalFooter,
14 } from "reactstrap";
15 import { obtenerFiscalias, crearFiscalia, eliminarFiscalia, modificarFiscalia, verFiscalia } from './actions/fiscalia'
16 import { obtenerTipos } from './actions/tipo'
17 import { obtenerDepartamentos, obtenerMunicipios } from './actions/ubicacion';
```

En primera instancia tenemos los imports en donde agregamos módulos como **“React”**, **“Bootstrap”** y algunos componentes de **“reactstrap”** como lo son las tablas, botones, contenedores y ventanas modales que es lo que se utilizó en este paso para hacer la creación y modificación de fiscalías. Se importan además todos los módulos de la carpeta **“actions”** ya que estos contienen la lógica para realizar las peticiones al servidor y serán explicados más adelante.

```
19 class App extends React.Component {
20
21     state = {
22         fiscalias: [],
23         deptos: [],
24         municipios: [],
25         tipos: [],
26         modalActualizar: false,
27         modalInsertar: false,
28         form: {
29             direccion_fiscalia: "",
30             id_departamento: 0,
31             id_fiscalia: 0,
32             id_municipio: 0,
33             id_tipo: 0,
34             nombre_departamento: "",
35             nombre_fiscalia: "",
36             nombre_municipio: "",
37             nombre_tipo: "",
38             tel_fiscalia: ""
39
40         },
41     };
42
```

Declaramos **“App”** como una clase que extiende de **”React.Component”** y creamos los estados que nos ayudarán a que nuestra interfaz vaya de la mejor manera posible. En primer lugar, declaramos un arreglo llamado **“fiscalias”** donde guardaremos la información de estas para mostrarla en formato de tabla más adelante. La misma idea va para los estados **“deptos”**, **“municipios”** y **“tipos”**. El estado **“modalActualizar”** y **“modalInsertar”** son de tipo booleano que activarán los modales en cuanto estas alcancen el valor de verdadero. Finalmente declaramos un **“form”** con todos los campos necesarios para poder crear, modificar y eliminar sin mayores inconvenientes, es acá donde se almacenará de forma temporal la información hasta que sean enviadas para crear o modificar una fiscalía.

- Función mostrarModalActualizar:

  ✓ **Parámetros:** dato

  ✓ **Descripción:** Esta función cambia el valor del estado **"modalActualizar"** para poder observar el modal de actualización y así modificar los datos de un usuario, los cuales se encuentran contenidos en el parámetro **"dato"**, sin embargo, el cambio del estado lo hace una vez obtenga los municipios del departamento cuyo identificador se encuentra dentro de **"dato"** en la propiedad **"id_departamento"** para ser exacto.

```
43      mostrarModalActualizar = (dato) => {
44          obtenerMunicipios(dato.id_departamento)
45          .then(municipios =>{
46              console.log(municipios)
47              this.setState({municipios});
48              this.setState({
49                  form: dato,
50                  modalActualizar: true,
51              });
52          })
53      };
```

- Función cerrarModalActualizar:

  ✓ **Parámetros:** Ninguno.

  ✓ **Descripción:** Cierra el modal de actualización, en este caso es llamado cuando se hace clic sobre el botón de **"cancelar"** dentro del modal de actualización.

```
55      cerrarModalActualizar = () => {
56          this.setState({ modalActualizar: false });
57      };
```

- Función mostrarModalInsertar:

  ✓ **Parámetros:** Ninguno.

  ✓ **Descripción:** Cambia el estado **"modalInsertar"** para que el modal de inserción sea visible y podamos crear un nuevo usuario.}

```
59      mostrarModalInsertar = () => {
60          this.setState({
61              modalInsertar: true,
62          });
63      };
```

- Función cerrarModalInsertar:

  ✓ **Parámetros:** Ninguno.

  ✓ **Descripción:** Cierra el modal de creación, en este caso es llamado cuando se hace clic sobre el botón de **"cancelar"** dentro del modal de creación.

```
65      cerrarModalInsertar = () => {
66          this.setState({ modalInsertar: false });
67      };
```

- Función componentDidMount:

  ✓ **Parámetros:** Ninguno.

  ✓ **Descripción:** Esta función se ejecuta al momento de montarse el componente, y dentro de él se llama a **"obtenerFiscalias()"** que se encarga de hacer la consulta al servidor por las fiscalías existente, al hallarlas modifica el estado **"fiscalías"** que al contener por lo menos un elemento genera la tabla donde se despliegan los detalles de dicho elemento. También se se cargan a los respectivos estados, los departamentos y tipos de fiscalías existentes para posteriormente utilizar estos estados en la modificación o creación de un nuevo registro.

```
69      componentDidMount() {
70          obtenerFiscalias()
71              .then(fiscalias => {
72                  console.log(fiscalias)
73                  this.setState({ fiscalias })
74              })
75
76          obtenerTipos()
77          .then(tipos => {
78              console.log(tipos)
79              this.setState({ tipos })
80          })
81
82          obtenerDepartamentos()
83          .then(deptos => {
84              console.log(deptos)
85              this.setState({deptos})
86          })
87
88      }
```

- Función eliminar:

  ✓ **Parámetros:** dato.

  ✓ **Descripción:** Esta función se llama al momento de querer eliminar un registro, se encuentra dentro de una tabla a la cuál se le manda por parámetro un objeto **"dato"** que incluye la información del registro que se desea eliminar. Dentro de él se llama a **"eliminarFiscalia(id_fiscalia)"** que es el encargado de hacer la petición **"delete"** al servidor y que necesita llevar por parámetro el identificador de la fiscalía a eliminar.

  Si el registro se elimina de forma satisfactoria se vuelve a llamar a **"obtenerFiscalias()"**para actualizar el estado actual que guarda nuestra información.

```
eliminar = (dato) => {
    var opcion = window.confirm("Está seguro que desea eliminar el elemento " + dato.nombre_fiscalia);
    if (opcion == true) {

        eliminarFiscalia(dato.id_fiscalia)
            .then(result => {
                console.log(result)
                alert('Elemento eliminado correctamente')
                obtenerFiscalias()
                    .then(fiscalias => {
                        console.log(fiscalias)
                        this.setState({ fiscalias })
                    })
            })
            .catch(err => {
                console.log(err);
                alert('Ocurrió un error al momento de eliminar el registro, por favor vuelva a intentar')
            })
        this.setState({ modalActualizar: false });
    }
};
```

- Función insertar:

  - ✓ **Parámetros:** Ninguno.

  - ✓ **Descripción:** Esta función se llama al momento de querer crear un nuevo registro, en este caso se va a buscar los datos al estado **"form"** que está atento a los cambios que ocurran en los campos dentro de los modales de inserción y actualización. Luego este estado es almacenado en una variable la cual es pasada como parámetro **"crearFiscalia()"** que se encarga de realizar la petición post al servidor para crear un nuevo registro. De igual forma si la inserción se realiza correctamente se llama a **"obtenerFiscalias()"** para actualizar el estado **"fiscalías"** y así refrescar la vista de nuestra tabla.

    Finalmente se modifica al estado **"modalInsertar"** para que se oculte el modal utilizado para agregar los datos de la inserción.

```
insertar = () => {
    var valorNuevo = { ...this.state.form };
    console.log(valorNuevo)
    crearFiscalia(valorNuevo)
        .then(result => {
            console.log(result)
            obtenerFiscalias()
                .then(fiscalias => {
                    console.log(fiscalias)
                    this.setState({ fiscalias })
                })

        })
        .catch(err => {
            alert('Ocurrió un error al momento de insertar el registro, por favor intentelo de nuevo')
        })


    this.setState({ modalInsertar: false });
}
```

- Función handleChange:

  - ✓ **Parámetros:** e.

  - ✓ **Descripción:** Es el manejador de los cambios dentro de los modales, recibe un parámetro que contiene la información del evento como el nombre del componente que lo activó, así como el valor que sufrió el cambio, en este caso cada vez que percibe una alteración actualiza el estado dentro del **"form"** para mantener la información más reciente y posteriormente

poder ser enviada ya sea para una petición de actualización o inserción. Cabe destacar que este manejador se encuentra atento al momento en que cambia el estado **“id_departamento”** ya que, dependiendo del identificador actual, así serán los municipios cargados en el estado **“municipios”** por lo que se realiza una llamada a **“obtenerMunicipios(id_departamento)”** que nos retorna los municipios asociados a un determinado departamento.

```
153     handleChange = (e) => {
154
155         if (e.target.name === 'id_departamento')
156         {
157             obtenerMunicipios(e.target.value)
158             .then(municipios =>{
159                 this.setState({
160                     municipios
161                 })
162             })
163         }
164
165
166
167         this.setState({
168             form: {
169                 ...this.state.form,
170                 [e.target.name]: e.target.value,
171             },
172         });
173     };
```

- Función render:

  ✓ **Parámetros:** Ninguno.

  ✓ **Descripción:** En esta función es donde se escribe el código para el diseño de nuestra interfaz. En este caso contamos con un pequeño encabezado que incluye un título representativo y un botón para crear una nueva fiscalía.

```
178                 <div className="centerContent">
179                     <img className="header-logo" src="MP_logo.png" height="100" width="auto" alt="Ministerio Publico" />
180                 </div>
181                 <br />
182                 <div className="header-buttons">
183                     <Button onClick={() => this.mostrarModalInsertar()}>
184                         Crear Fiscalía
185                     </Button>
186                 </div>
187                 <br />
188                 <br />
```

Posteriormente creamos la tabla donde se visualizará la información detalla de cada una de estas. Primero creamos los encabezados:

```
189                 <div className="center-title">
190                     <h1 >Fiscalías Dentro del Sistema</h1>
191                     <br />
192                     <Table>
193                         <thead>
194                             <tr>
195                                 <th>Departamento</th>
196                                 <th>Municipio</th>
197                                 <th>Nombre de la Fiscalía</th>
198                                 <th>Teléfono</th>
199                                 <th>Dirección</th>
200                                 <th>Tipo de Fiscalía</th>
201                                 <th>Acciones</th>
202                             </tr>
203                         </thead>
```

Luego en el cuerpo de la tabla agregamos el código necesario para mapear los registros contenidos en el estado **"fiscalías"** para desglosar la información contenida en él, así como también presentamos los botones con las opciones de actualización y eliminación de registros.

```
<tbody>
    {
        this.state.fiscalias.map((dato) => (
            <tr key={dato.id_fiscalia}>
                <td>{dato.nombre_departamento}</td>
                <td>{dato.nombre_municipio}</td>
                <td>{dato.nombre_fiscalia}</td>
                <td>{dato.tel_fiscalia}</td>
                <td>{dato.direccion_fiscalia}</td>
                <td>{dato.nombre_tipo}</td>
                <td>
                    <Button
                        color="primary"
                        onClick={() => this.mostrarModalActualizar(dato)}
                    >

                        Editar
                    </Button>
                    <br/>
                    <br/>
                    <Button color="danger" onClick={() => this.eliminar(dato)}>Eliminar</Button>
                </td>
            </tr>
        ))
    }
</tbody>
```

Pasamos a la parte de los modales, en este caso contamos con dos, uno para la inserción de registros y otro para la actualización. En el Modal de inserción primero se le indica el estado al cual estará asociado para ser mostrado o no, en el **"ModalHeader"** nada más le colocamos un título al modal. Luego en el cuerpo creamos un **"FormGroup"** para cada campo solicitado.

```

<Modal isOpen={this.state.modalInsertar}>
    <ModalHeader>
        <div><h3>Crear Nueva Fiscalía</h3></div>
    </ModalHeader>

```

En este caso los campos son: nombre de la fiscalía, tipo de fiscalía el cual es un componente select, por lo que asociamos las opciones con el estado **"tipos"** y se mapean todas las opciones dentro de este componente.

```
<FormGroup>
    <label>
        Nombre de la Fiscalía:
    </label>
    <input
        className="form-control"
        name="nombre_fiscalia"
        type="text"
        onChange={this.handleChange}
    />
</FormGroup>

<FormGroup>
    <label>
        Tipo de Fiscalía:
    </label>
    <select
        className="form-control"
        name="id_tipo"
        value={this.state.form.id_tipo}
        onChange={this.handleChange}
    >
        <option value={0}></option>
        {
            this.state.tipos.map( tipo => <option key={tipo.id_tipo} value={tipo.id_tipo} >{tipo.nombr
        }

    </select>
</FormGroup>
```

Contamos también con los campos de: número telefónico y departamentos, este último también se trata de un componente select asociado al estado **“deptos”** que se llenó con la información al inicio de la carga de la página.

```
<FormGroup>
    <label>
        No. de Teléfono:
    </label>
    <input
        className="form-control"
        name="tel_fiscalia"
        type="text"
        onChange={this.handleChange}
    />
</FormGroup>

<FormGroup>
    <label>
        Departamento:
    </label>
    <select
        className="form-control"
        name="id_departamento"
        value={this.state.form.id_departamento}
        onChange={this.handleChange}
    >
        <option value={0}></option>
        {
            this.state.deptos.map( tipo => <option key={tipo.id_departamento} value={tipo.id_departame
        }

    </select>
</FormGroup>
```

El campo de municipio también se trata de un select que depende directamente del departamento seleccionado, de esta forma mostrará únicamente los municipios pertenecientes a dicho departamento. También contamos con el campo de dirección.

```
<FormGroup>
    <label>
        Municipio:
    </label>
    <select
        className="form-control"
        name="id_municipio"
        value={this.state.form.id_municipio}
        onChange={this.handleChange}
    >
        <option value={0}></option>
        {
            this.state.municipios.map( tipo => <option key={tipo.id_municipio} value={tipo.id_municipi
        }

    </select>
</FormGroup>

<FormGroup>
    <label>
        Dirección:
    </label>
    <input
        className="form-control"
        name="direccion_fiscalia"
        type="text"
        onChange={this.handleChange}
    />
</FormGroup>
```

Finalmente, en el **“ModalFooter”** contamos con los botones que nos permiten aceptar o cancelar la acción solicitada

```
<ModalFooter>
    <Button
        color="primary"
        onClick={() => this.insertar()}
    >
        Insertar
    </Button>
    <Button
        className="btn btn-danger"
        onClick={() => this.cerrarModalInsertar()}
    >
        Cancelar
    </Button>
</ModalFooter>
```

El modal de modificación sigue la misma estructura con la salvedad de que los campos no se encuentran vacíos, por lo que se hace uso de la propiedad **“value”** para colocarles la información correspondiente según el campo, y esta información está almacenada en el estado **“form”**

Cabecera del modal, con el respectivo título

```
<Modal isOpen={this.state.modalActualizar}>
    <ModalHeader>
        <div><h3>Editar Fiscalía</h3></div>
    </ModalHeader>
```

Cuerpo del Modal con los capos nombre de fiscalía y tipo de fiscalía.

```
<ModalBody>

    <FormGroup>
        <label>
            Nombre de la Fiscalía:
        </label>
        <input
            className="form-control"
            name="nombre_fiscalia"
            type="text"
            value={this.state.form.nombre_fiscalia}
            onChange={this.handleChange}
        />
    </FormGroup>

    <FormGroup>
        <label>
            Tipo de Fiscalía:
        </label>
        <select
            className="form-control"
            name="id_tipo"
            value={this.state.form.id_tipo}
            onChange={this.handleChange}
        >
            <option value={0}></option>
            {
                this.state.tipos.map( tipo => <option key={tipo.id_tipo} value={tipo.id_tipo} >{tipo.nombr
            }

        </select>
    </FormGroup>
```

Campos para el número de teléfono y la selección del departamento

```
<FormGroup>
    <label>
        No. de Teléfono:
    </label>
    <input
        className="form-control"
        name="tel_fiscalia"
        type="text"
        value={this.state.form.tel_fiscalia}
        onChange={this.handleChange}
    />
</FormGroup>

<FormGroup>
    <label>
        Departamento:
    </label>
    <select
        className="form-control"
        name="id_departamento"
        value={this.state.form.id_departamento}
        onChange={this.handleChange}
    >
        <option value={0}></option>
        {
            this.state.deptos.map( tipo => <option key={tipo.id_departamento} value={tipo.id_departame
        }

    </select>
</FormGroup>
```

Campos para la selección del municipio y la dirección

```
<FormGroup>
    <label>
        Municipio:
    </label>
    <select
        className="form-control"
        name="id_municipio"
        value={this.state.form.id_municipio}
        onChange={this.handleChange}
    >
        <option value={0}></option>
        {
            this.state.municipios.map( tipo => <option key={tipo.id_municipio} value={tipo.id_municipi
        }

    </select>
</FormGroup>

<FormGroup>
    <label>
        Dirección:
    </label>
    <input
        className="form-control"
        name="direccion_fiscalia"
        type="text"
        value={this.state.form.direccion_fiscalia}
        onChange={this.handleChange}
    />
</FormGroup>
```

Finalmente, en el pie del modal tenemos los botones que confirman la edición o cancelan el proceso.

```
<ModalFooter>
    <Button
        color="primary"
        onClick={() => this.editar(this.state.form)}
    >
        Editar
    </Button>
    <Button
        color="danger"
        onClick={() => this.cerrarModalActualizar()}
    >
        Cancelar
    </Button>
</ModalFooter>
```

Por último, pero no menos importante (literalmente) debemos exportar la App como tal para poder ser renderizada. Eso lo hacemos mediante esta línea

```
476   export default App;
```

src/actions/fiscalía.js

Dentro de este archivo es donde se realizan las peticiones hacia el servidor, sean estas POST, PUT, GET o DELETE mediante la librería axios.

- crearFiscalia:

  - ✓ **Parámetros:** data: objeto en formato json con los datos necesarios para poder realizar una inserción dentro de la base de datos.

  - ✓ **Descripción:** Realiza una petición POST al endpoint **"/crearFiscalia"**, que es el servicio que nos crea una nueva fiscalía, y le manda por parámetro el objeto data. En este caso no se retorna respuesta.

```
3   export const crearFiscalia = async data => {
4       axios.post('http://' + window.location.hostname + ':3000/crearFiscalia' , data)
5       .then(result =>{
6           console.log(result)
7       })
8       .catch(err => {
9           console.log(err);
10      })
11  }
```

- obtenerFiscalia:

  - ✓ **Parámetros:** Ninguno.

  - ✓ **Descripción:** Realiza una petición GET al endpoint **"/obtenerFiscalias"**, servicio que nos retorna todas las fiscalías en el sistema, y no envía ni recibe parámetros, al final retorna la respuesta del servidor.

```
14  export const obtenerFiscalias = async () => {
15
16      return await axios.get('http://' + window.location.hostname + ':3000/obtenerFiscalias')
17      .then( result => result.data.datos)
18      .catch( err => {
19          console.log(err);
20      })
21
22  }
```

- verFiscalia:

  - ✓ **Parámetros:** id.

  - ✓ **Descripción:** Realiza una petición GET al endpoint **"/verFiscalia"**, servicio que nos retorna la información de una determinada fiscalía dependiendo del parámetro recibido y al final retorna la respuesta del servidor.

```
export const verFiscalia = async id => {

    return await axios.get('http://' + window.location.hostname + ':3000/verFiscalia/' + id)
    .then( result => result.data.datos)
    .catch( err => {
        console.log(err);
    })

}
```

- modifcarFiscalia:

  - ✓ **Parámetros:** data: objeto en formato json con los datos necesarios para poder realizar una actualización de fiscalía dentro de la base de datos.

  - ✓ **Descripción:** Realiza una petición PUT al endpoint **"/modifcarFiscalia"**, servicio que nos permite actualizar la información de un registro en base a la información recibida y que no retorna ninguna respuesta.

```
export const modificarFiscalia = async data => {
    axios.put('http://' + window.location.hostname + ':3000/modificarFiscalia' , data)
    .then(result =>{
        console.log(result)
    })
    .catch(err => {
        console.log(err);
    })
}
```

- eliminarFiscalia:

  - ✓ **Parámetros:** id

  - ✓ **Descripción:** Realiza una petición DELETE al endpoint **"/eliminarFiscalia"** que es el servicio que nos permite retirar una fiscalía del sistema, en este caso elimina la fila con el identificador indicado mediante el parámetro de entrada.

```
export const eliminarFiscalia = async id => {

    return await axios.delete('http://' + window.location.hostname + ':3000/eliminarFiscalia/' + id)
    .then( result => result.data.datos)
    .catch( err => {
        console.log(err);
    })
```

src/actions/tipo.js

Dentro de este archivo se realiza la petición al servidor para obtener los tipos de fiscalía disponibles, mediante la librería axios.

- obtenerTipos:

  - ✓ **Parámetros:** Ninguno.

  - ✓ **Descripción:** Realiza una petición GET al endpoint **"/tipoFiscalia"** que es el servicio que nos permite obtener todos los tipos de fiscalía existentes en el sistema.

```
export const obtenerTipos = async () => {

    return await axios.get('http://' + window.location.hostname + ':3000/tipoFiscalia')
    .then( result => result.data.datos)
    .catch( err => {
        console.log(err);
    })
```

src/actions/ubicacion.js

Dentro de este archivo se realizan las peticiones al servidor correspondientes a los municipios y departamentos, al igual que en los anteriores, las peticiones se realizan mediante la librería axios.

- obtenerDepartamentos:

  - ✓ **Parámetros:** Ninguno.

  - ✓ **Descripción:** Realiza una petición GET al endpoint **"/departamentos"** que es el servicio que nos permite obtener todos los departamentos existentes en el sistema.

```
export const obtenerDepartamentos = async () => {
    return await axios.get('http://' + window.location.hostname + ':3000/departamentos')
    .then(result => result.data.datos)
    .catch( err => {
        console.log(err);
    })
}
```

- obtenerMunicipios:

  - ✓ **Parámetros:** id_depto.

  - ✓ **Descripción:** Realiza una petición GET al endpoint **"/municipios"** que es el servicio que nos permite obtener los municipios de un departamento en concreto en base al identificador que se recibe como parámetro de entrada.

```
export const obtenerMunicipios = async id_depto => {
    return await axios.get('http://' + window.location.hostname + ':3000/municipios/' + id_depto)
    .then(result => result.data.datos)
    .catch( err => {
        console.log(err);
    });
}
```