

## Models

```
const { DataTypes } = require('sequelize');
// Exportamos una funcion que define el modelo
// Luego le injectamos la conexion a sequelize.
module.exports = (sequelize) => {
  // defino el modelo
  sequelize.define('country', {
    id: {
      type: DataTypes.STRING(3),
      allowNull: false,
      primaryKey: true,
      validate: {
        isAlpha: true,
      },
    },
    name: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    img_flag: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        isUrl: true,
      },
    },
    continent: {
      type: DataTypes.STRING,
      allowNull: false
    },
    capital: {
      type: DataTypes.STRING,
      allowNull: false
    },
    subregion: {
      type: DataTypes.STRING
    },
    area: {
      type: DataTypes.REAL
    },
    population: {
```

definimos el modelo de “country”, estructuramos de sequelize DataTypes para usarlo y especificar el tipo de valores que permitirá en ese campo.

Usamos constrains(restricciones) como allowNull: false

```
1 const { DataTypes } = require('sequelize');
2 // Exportamos una función que define el modelo
3 // Luego le inyectamos la conexión a sequelize.
4 module.exports = (sequelize) => {
5   // defino el modelo
6   sequelize.define('activity', {
7     name: {
8       type: DataTypes.STRING,
9       allowNull: false,
10      validate: {
11        isAlpha: true
12      }
13    },
14    difficulty: {
15      type: DataTypes.ENUM(["", "1", "2", "3", "4", "5"])
16    },
17    duration: {
18      type: DataTypes.ENUM(["", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"]),
19    },
20    season: {
21      type: DataTypes.ENUM(["", "summer", "autumn", "winter", "spring"]),
22    }
23  });
24 };
```

Definir modelo(representación de una tabla) llamado “activity” que luego se pluralizara(debió a una librería) en la BD, con los campos(atributos) de nombre, dificultad, duración y temporada. El id lo genera automáticamente. Exportamos dicho modelo. Permite valores vacíos ya que no es un campo obligatorio.

Usamos validate: {} para validar que solo reciba letras

## DB

```
require('dotenv').config();
const { Sequelize } = require('sequelize');
const fs = require('fs');
const path = require('path');
const {
  DB_USER, DB_PASSWORD, DB_HOST,
} = process.env;

const sequelize = new Sequelize(`postgres://${DB_USER}:${DB_PASSWORD}@${DB_HOST}/countries`, {
  logging: false, // set to console.log to see the raw SQL queries
  native: false, // lets Sequelize know we can use pg-native for ~30% more speed
});
const basename = path.basename(__filename);

const modelDefiners = [];
```

Acá nos conectamos a la base de datos dando nuestros datos de conexión los cuales está en .env

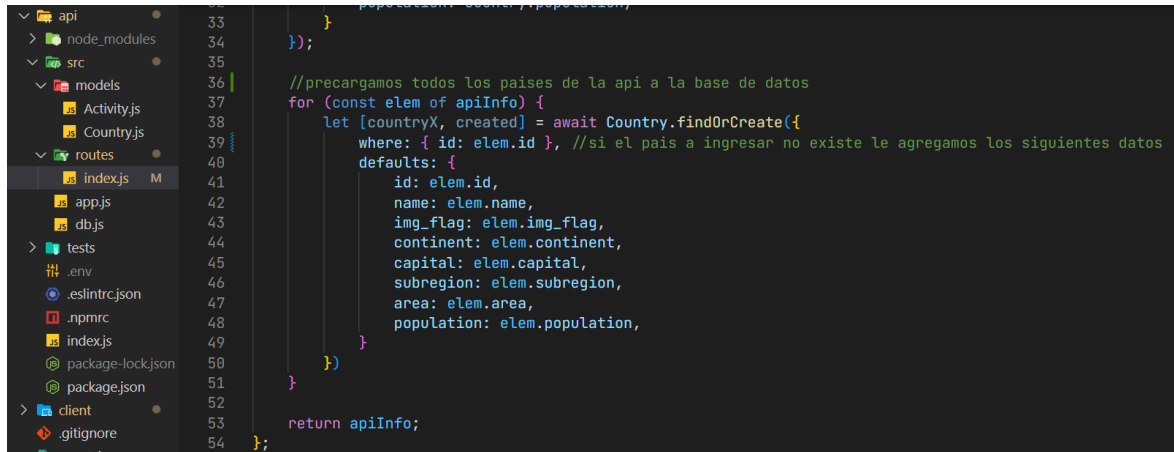
```
17 // Leemos todos los archivos de la carpeta Models, los requerimos y agregamos al arreglo modelDefiners
18 fs.readdirSync(path.join(__dirname, '/models'))
19   .filter((file) => (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js'))
20   .forEach((file) => {
21     modelDefiners.push(require(path.join(__dirname, '/models', file)));
22   });
23
24 // Inyectamos la conexión (sequelize) a todos los modelos
25 modelDefiners.forEach(model => model(sequelize));
26 // Capitalizamos los nombres de los modelos ie: product => Product
27 let entries = Object.entries(sequelize.models);
28 let capsEntries = entries.map((entry) => [entry[0].toUpperCase() + entry[0].slice(1), entry[1]]);
29 sequelize.models = Object.fromEntries(capsEntries);
30
31 // En sequelize.models están todos los modelos importados como propiedades
32 // Para relacionarlos hacemos un destructuring
33 const { Country, Activity } = sequelize.models;
34
35 // Aca vendrian las relaciones
36 // Product.hasMany(Reviews);
37 Country.belongsToMany(Activity, { through: 'country_activity' });
38 Activity.belongsToMany(Country, { through: 'country_activity' });
39
40 module.exports = {
41   ...sequelize.models, // para poder importar los modelos así: const { Product, User } = require('./db.js');
42   conn: sequelize,     // para importart la conexión { conn } = require('./db.js');
43 };
```

El código arriba del paréntesis rojo se encarga de traer los archivos de los modelos, leerlos y conectarlos con Sequelize

Dentro de las llaves rojas tomamos los modelos importados y realizamos la asociación que en este caso es de muchos a muchos y creamos una tabla intermedia llamada `country_activity` que tendrá los id del modelo `Activity` y `Country`.

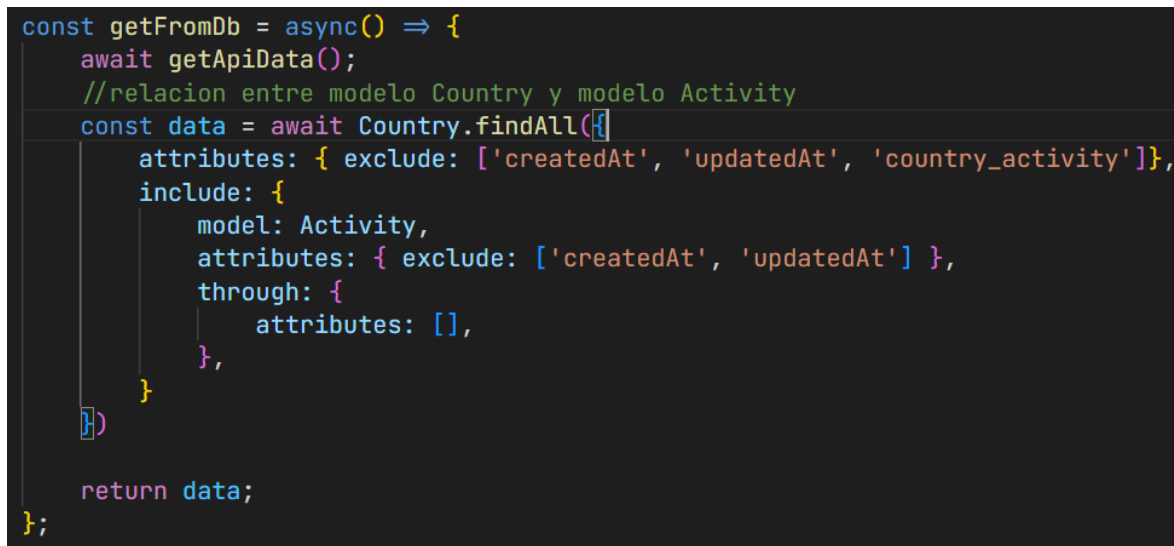
Dentro de las llaves azules exportamos los modelos sequelize y la conexión de la base de datos.

## Routes



```
33 }
34 });
35
36 //precargamos todos los paises de la api a la base de datos
37 for (const elem of apiInfo) {
38   let [countryX, created] = await Country.findOrCreate({
39     where: { id: elem.id }, //si el pais a ingresar no existe le agregamos los siguientes datos
40     defaults: {
41       id: elem.id,
42       name: elem.name,
43       img_flag: elem.img_flag,
44       continent: elem.continent,
45       capital: elem.capital,
46       subregion: elem.subregion,
47       area: elem.area,
48       population: elem.population,
49     }
50   });
51 }
52
53 return apiInfo;
54 };
```

Agregar la data del api de los países, de manera individual en cada una de las filas.



```
const getFromDb = async() => {
  await getApiData();
  //relacion entre modelo Country y modelo Activity
  const data = await Country.findAll({
    attributes: { exclude: ['createdAt', 'updatedAt', 'country_activity'] },
    include: {
      model: Activity,
      attributes: { exclude: ['createdAt', 'updatedAt'] },
      through: {
        attributes: [],
      },
    },
  });
  return data;
};
```

Traemos todos los países previamente cargados a la base de datos, y le decimos que de la información traída me ignore los campos de momento de agregado y actualizado a la vez que la tabla intermedia. A su vez me incluya la información en la tabla de actividades, está se van a comunicar vía su id de cada uno que las conecta. Y finalmente la función hace un return con dicha data.



```
//end-points
router.get("/countries" , async(req, res) => {
  const { name } = req.query;
  const myDB = await getFromDb();
  if (name) {
    const filteredCountry = myDB.filter(c => c.name.toLowerCase().includes(name.toLowerCase()));
    filteredCountry.length ? res.status(200).send(filteredCountry) : res.status(404).send("Country not found");
  } else {
    res.status(200).send(myDB);
  }
});
```

En el primer end-point va a mostrar todos los países traídos de la BD y vamos a validar si viene algún valor por medio de la query, este será en caso de que el usuario desee buscar un país en específico y no toda la lista de países en la BD, a su vez tendrá un método `toLowerCase` para que no importe si lo busca en minúscula o mayúscula.

```
router.get("/countries/:idCountry", async(req, res) => {
  const { idCountry } = req.params;
  const myDB = await getFromDb();
  const countryDetails = myDB.filter(coun => coun.id.toLowerCase() === idCountry.toLowerCase());

  countryDetails.length ? res.status(200).send(countryDetails) : res.status(404).send("Country not found");
});
```

Por medio de `params` recibimos el `id` de cada país, y este a su vez nos devuelve la data de dicho país en la BD, esto lo usaremos para ver los detalles de los países.

```
router.get("/activity", async(req, res) => {
  const getActivity = await Activity.findAll({
    attributes: ["name", "id"]
  });
  if (getActivity.length) {
    res.status(200).send(getActivity);
  } else {
    res.json([
      {
        id: 000,
        name: "No hay actividades"
      }
    ]);
  }
});
```

End point que nos trae las actividades de la BD de la tabla `activities`, donde solo trae el `name` y el `id`. Si no hay actividades trae un arreglo con un solo elemento que dice "No hay actividades".

```

router.post("/activity", async(req, res) => {
  let {
    name,
    difficulty,
    duration,
    season,
    countries_id
  } = req.body

  let [activityX, created] = await Activity.findOrCreate({ //creamos una fila en activity con estos datos
    where: {name: name},
    defaults: {
      name: name,
      difficulty: difficulty,
      duration: duration,
      season: season
    }
  });

  const allDataCountries = await Country.findAll( { where: {id: countries_id} } );
  //contiene la info de los paises

  for (let value of allDataCountries) {
    //values contiene el total de data de los paises
    await value.addActivity(activityX.dataValues.id); //fooInstance.addbar();
    //vinculo instancia del pais seleccionada a la tabla de actividad por medio del id de la tabla
    console.log(activityX);
  }

  res.status(200).send("activity created succesfully");
});

```

Recibimos unos valores por body(vendrán del formulario). Si el nombre de la actividad(se especifica en el where) no existe en la base de datos cree una fila con dicha información que asignara a su correspondiente columna las cuales son nombre, dificultad, duración y temporada.

Una vez creada la actividad le pido que vincule dicha actividad con una o múltiples ciudades especificadas por su id.

No olviddr sobre las funciones asíncronas(async await).

SERVER

```

app.js M X index.js ... routes M Activity.js Country.js db.js index.js api
api > src > app.js > server.use() callback
1  const express = require('express');
2  const cookieParser = require('cookie-parser');
3  const bodyParser = require('body-parser');
4  const morgan = require('morgan');
5  const routes = require('./routes/index.js');
6
7  require('./db.js');
8
9  const server = express();
10
11  server.name = 'API';
12
13  server.use(bodyParser.urlencoded({ extended: true, limit: '50mb' }));
14  server.use(bodyParser.json({ limit: '50mb' }));
15  server.use(cookieParser());
16  server.use(morgan('dev'));
17  server.use((req, res, next) => {
18    res.header('Access-Control-Allow-Origin', 'http://localhost:3000'); // update to match the domain you will make the request from
19    res.header('Access-Control-Allow-Credentials', 'true');
20    res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');
21    res.header('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, DELETE');
22    next();
23  });
24
25  server.use('/', routes);
26
27  // Error catching endware.
28  server.use((err, req, res, next) => { // eslint-disable-line no-unused-vars
29    const status = err.status || 500;
30    const message = err.message || err;
31    console.error(err);
32    res.status(status).send(message);
33  });
34
35  module.exports = server;

```

Requerimos express, requerimos los middlewares necesarios y los aplicamos a express, a su vez que asignamos dominios para el acceso y evitar incidentes de CORS (Intercambio de Recursos de Origen Cruzado). Y al final mensajes de error en caso de tener problemas con la conexión.

## INDEX API

```

index.js X app.js M Activity.js Country.js db.js
api > index.js > ...
17  // ~~~~~
20  const server = require('./src/app.js');
21  const { conn } = require('./src/db.js');
22
23  // Syncing all the models at once.
24  conn.sync({ force: true }).then(() => {
25    server.listen(3001, () => {
26      console.log('%s listening at 3001'); // eslint-disable-line no-console
27    });
28  });
29

```

Requerimos el módulo de conexión a la base de datos(conn) y el servidor con express. force: true elimina los registros en la base de datos cada vez que se cargue el servidor. Va a correr el servidor en el puerto 3001 y devolverá un mensaje una vez conectado o que falle.