Modelación del proyecto(backend)

momento 2:

Diagrama de un proyecto relacional completo :

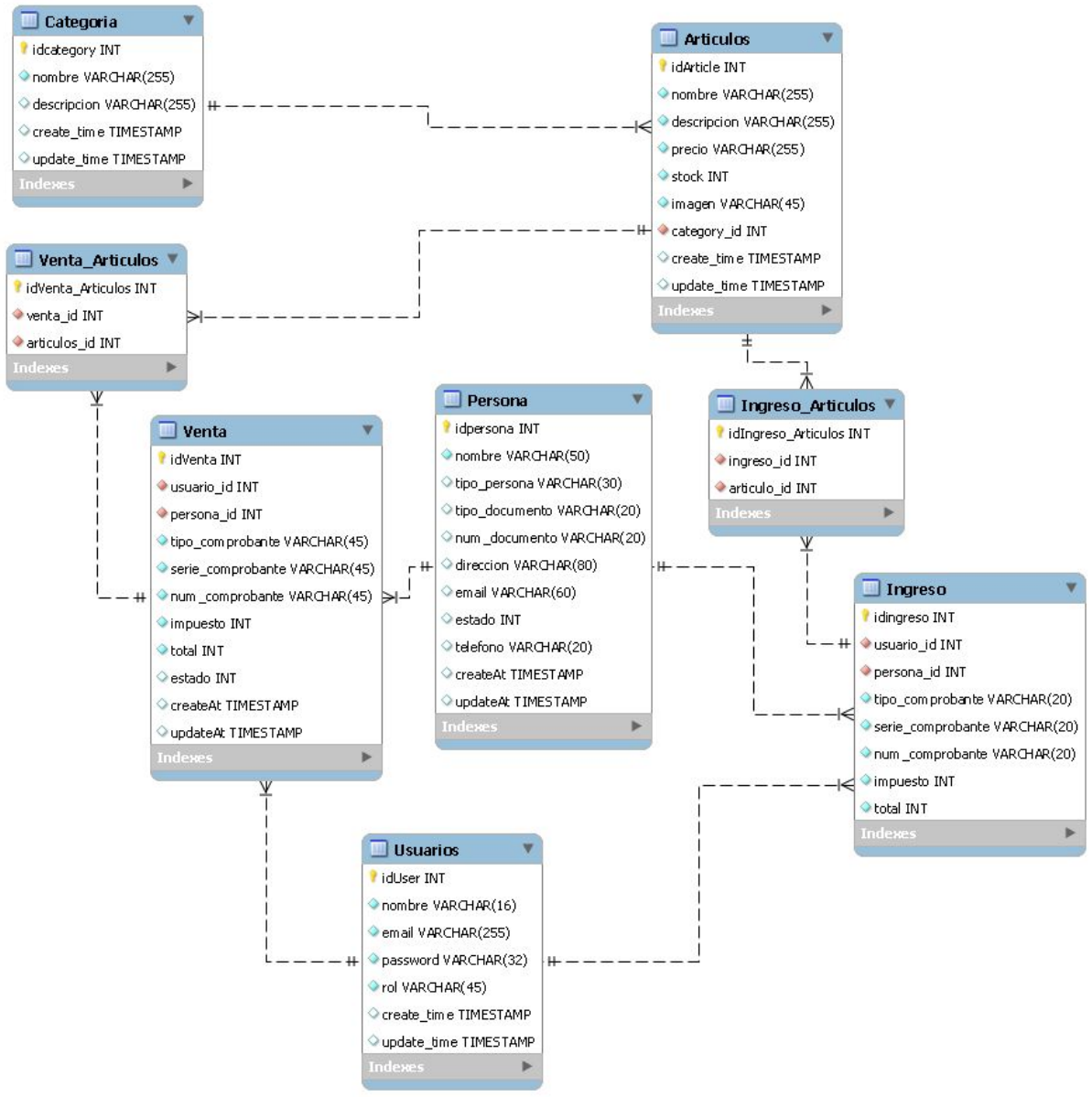
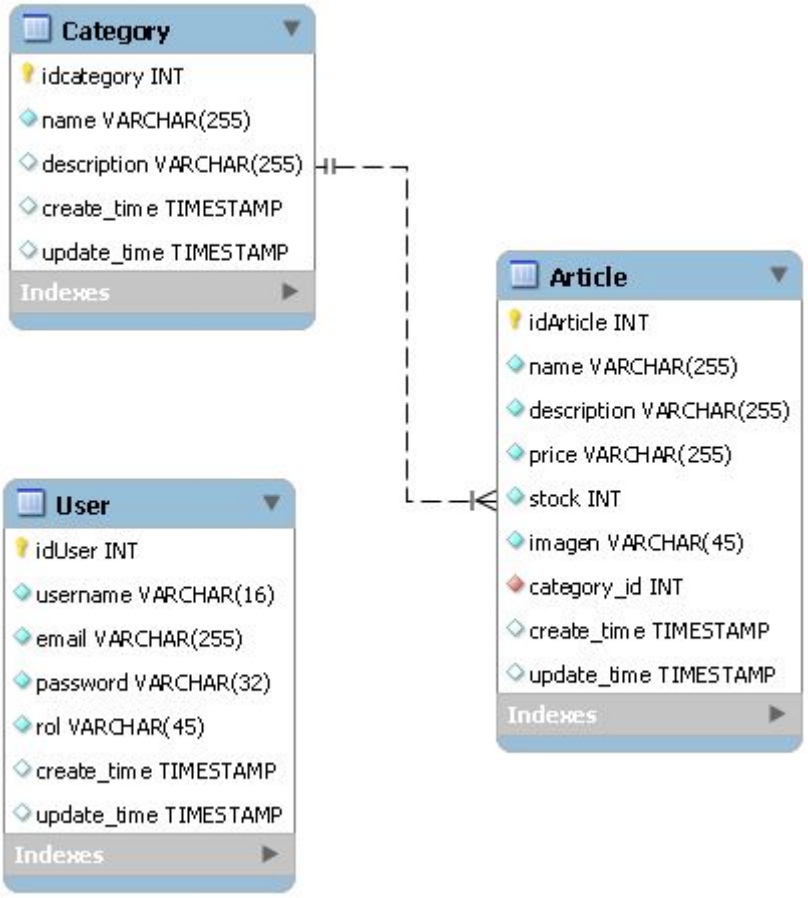


Diagrama de relacional del proyecto final:



Momento 3:
Creación de modelos
antes de crear los modelos debemos inicializar nuestro proyecto como lo hemos explicado en las sesiones anteriores:
creramos el directorio donde se aloja nuestro proyecto:
mkdir semana4_proyecto_final
cd semana4_proyecto_final

comandos:

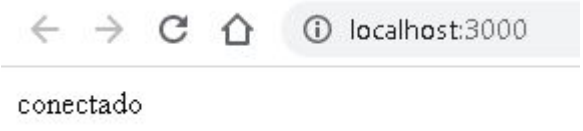
- npm init -y

paquetes:

- npm install cors jsonwebtoken bcryptjs express-promise-router --save
- npm install --save express body-parser sequelize sequelize-cli sqlite3 nodemon mysql2 morgan
- iniciacializamos sequelize-cli con:: **sequelize-cli init**
- creamos archivo index.js:

```
const express = require('express');
const morgan = require('morgan');
const cors = require('cors');
const path = require('path');


const bodyParser = require('body-parser');


const app = express();
app.use(morgan('dev'));
app.use(cors());
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({ extended: true }));

app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'public')))

app.get('/', function(req, res) {
     res.send("conectado");
});


app.set('port', process.env.PORT || 3000);

app.listen(app.get('port'), () => {
    console.log('Server on port ' + app.get('port') + ' on dev');
});
```

ejecutamos el comando node index.js para observar que todo haya salido bien:

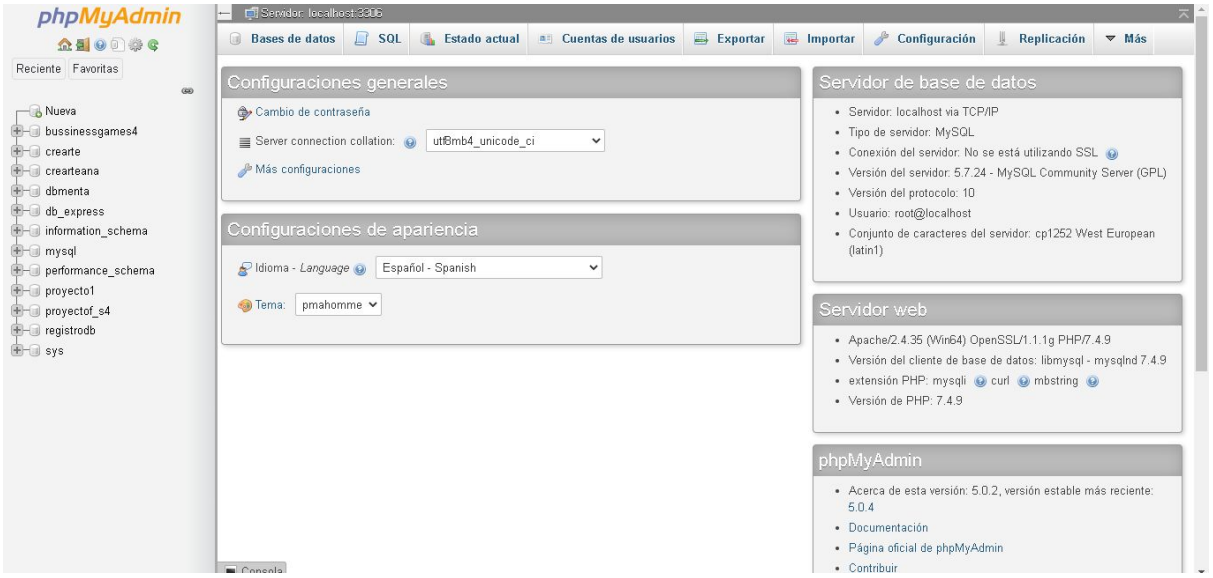


conexión bases de datos:
para simplicidad y facilidad al momento de crear la base de datos de nuestro proyecto se realizará con phpmyadmin , pero queda a criterio de cada uno escoger su herramienta que mas le guste

phpmyadmin:phpMyAdmin es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web, utilizando un navegador web. Actualmente puede crear y eliminar Bases de Datos, crear, eliminar y alterar tablas, borrar, editar y añadir

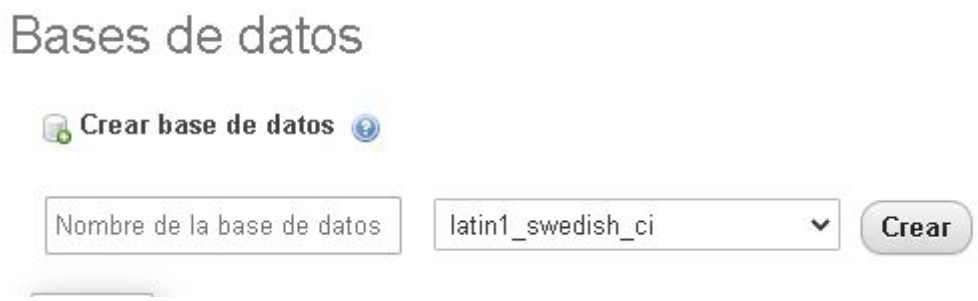
campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos y está disponible en 72 idiomas.

iniciamos el servidor de bases de datos ,abrimos phpmyadmin en el navegador y iniciamos sesión ,nos encontramos con la siguiente interfaz.



:

damos clic en donde dice nueva :



ingresamos el nombre que le queremos dar en mi caso **proyectof_s4** y le damos crear ,nos aparecerá en la lista del lado izquierdo.

ahora ya creada nuestra base de datos vamos a conectarla a nuestro proyecto , vamos al directorio config/config.json y en el objeto **development** ponemos las credenciales de la bd creada anteriormente:

```
{
    "development": {
        "username": "root",
        "password": null,
        "database": "proyectof_s4",
        "host": "127.0.0.1",
        "dialect": "mysql"
    },
    "test": {
        "username": "root",
        "password": null,
        "database": "database_test",
        "host": "127.0.0.1",
        "dialect": "mysql"
    },
    "production": {
        "username": "root",
        "password": null,
        "database": "database_production",
        "host": "127.0.0.1",
        "dialect": "mysql"
    }
}
```

Momento 4:
Creación de modelos:
para creación de nuestro modelos ,observamos cada una de las tablas con sus atributos del diagrama relacional para implementarlos en nuestro proyecto , esto se h realiza por medio del comando :
sequelize model:create --name nombre_tabla --attributes
en nuestro caso el conjunto de comandos de acuerdo con el diagrama relacional anterior quedaría así :

Modelo Usuario:
sequelize model:generate --name Usuario --attributes
rol:string,nombre:string,password:string,email:string,tipo_documento:string,
num_documento:string,direccion:string,telefono:string,estado:integer
como podemos observar tenemos nuevos atributos uno de ellos es rol , lo utilizaremos para manejar restricciones de usuario según su rol, los roles serán:
Administrador:con acceso total al sistema
Vendedor:solo para acceso al módulo de ventas.
Almacenero:solo acceso al módulo de ingresos ,artículos y categorías.

Modelo Persona:

sequelize model:generate --name Persona --attributes tipo_persona:string,nombre:string,tipo_documento:string,num_documento:string,direccion:string,telefono:string,email:string,estado:integer

Modelo Categoría:

sequelize model:generate --name Categoria --attributes nombre:string,descripcion:string,estado:integer

Modelo Artículos:

sequelize model:generate --name Articulo --attributes codigo:string,nombre:string,descripcion:string,precio_venta:integer,stock:integer,estado:integer,categoriaId:integer

Modelo Venta:

sequelize model:create --name Venta --attributes usuarioId:integer,personaId:integer,tipo_comprobante:string,serie_comprobante:string,num_comprobante:string,impuesto:integer,total:integer,estado:integer

Modelo para la relación de mucho a muchos Venta y artículos:

sequelize model:create --name VentaArticulos --attributes ventaId:integer,articuloId:integer,articulo:string,cantidad:integer,precio:integer,descuento:integer

Modelo Ingreso:

sequelize model:create --name Ingreso --attributes usuarioId:integer,personaId:integer,tipo_comprobante:string,serie_comprobante:string,num_comprobante:string,impuesto:integer,total:integer,estado:integer

Modelo para la relación de mucho a muchos Ingresos y artículos:

sequelize model:create --name IngresoArticulos --attributes ingresoId:integer,articuloId:integer,articulo:string,cantidad:integer,precio:integer,descuento:integer

Nota:para el proyecto solo vamos a desarrollar los modelo usuario,categoría,artículos por limitaciones de tiempo.

luego al ejecutar cada uno de estos comandos para la creación de modelos ,observamos el directorio models con cada uno de estos y el directorio migrations con sus respectivas migraciones de cada modelo, antes de migrar los modelos debemos actualizarlos los modelos y las migraciones con las relaciones que se observan en nuestro modelo:

como por ejemplo

Modelo articulo:

en este caso tenemos una relación de uno a muchos con el modelo categoría (como se observa en el diagrama)

actualizamos el función associate con la relación

:

```
static associate(models) {
            this.belongsTo(models.Categoria, { foreignKey: 'categoriaId', as: 'categoria'
})
        }
```

y en su respectiva migración para tener una buena indexación en nuestra bd actualizamos el campo categoriaid:

```
categoriaId: {
                type: Sequelize.INTEGER,
                allowNull: false,
                references: { // User belongsTo Company 1:1
                    model: 'Categoria',
                    key: 'id'
                }
            }
```

y hacemos lo mismo para cada uno de los modelos que tengan esta relación:

para relaciones de muchos a muchos como demostración realizaremos el modelo ingreso que está relacionado muchos a muchos con el modelo artículos , uno a muchos los modelos persona y usuarios:

actualizamos el modelo Ingreso con el siguiente código:

```
'use strict';
const {
    Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
    class Ingreso extends Model {
        static associate(models) {
            this.belongsToMany(models.Articulo, {
                through: 'IngresoArticulos',
                as: 'detalles',
                foreignKey: 'ingresoId',
                otherKey: 'articuloId'
            });

            this.belongsTo(models.Persona, { foreignKey: 'personaId', as: 'persona' });
            this.belongsTo(models.Usuario, { foreignKey: 'usuarioId', as: 'usuario' });

        }
    };
    Ingreso.init({
        usuarioId: DataTypes.INTEGER,
        personaId: DataTypes.INTEGER,
        tipo_comprobante: DataTypes.STRING,
        serie_comprobante: DataTypes.STRING,
        num_comprobante: DataTypes.STRING,
        impuesto: DataTypes.INTEGER,
        total: DataTypes.INTEGER,
        estado: DataTypes.INTEGER

    }, {
        sequelize,
```

```
        modelName: 'Ingreso',
    });
    return Ingreso;
};
```

y la migración:

```
'use strict';
module.exports = {
    up: async(queryInterface, Sequelize) => {
        await queryInterface.createTable('Ingresos', {
            id: {
                allowNull: false,
                autoIncrement: true,
                primaryKey: true,
                type: Sequelize.INTEGER
            },
            tipo_comprobante: {
                type: Sequelize.STRING
            },
            serie_comprobante: {
                type: Sequelize.STRING
            },
            num_comprobante: {
                type: Sequelize.STRING
            },
            impuesto: {
                type: Sequelize.INTEGER
            },
            total: {
                type: Sequelize.INTEGER
            },
            estado: {
                type: Sequelize.INTEGER
            },
            usuarioId: {
                type: Sequelize.INTEGER,
                allowNull: false,
                references: { // User belongsTo Company 1:1
                    model: 'Usuarios',
                    key: 'id'
                }
            },
            personaId: {
                type: Sequelize.INTEGER,
                allowNull: false,
                references: { // User belongsTo Company 1:1
                    model: 'Personas',
                    key: 'id'
                }
            },
            createdAt: {
                allowNull: false,
                type: Sequelize.DATE
```

```
            },
            updatedAt: {
                allowNull: false,
                type: Sequelize.DATE
            }
        });
    },
    down: async(queryInterface, Sequelize) => {
        await queryInterface.dropTable('Ingresos');
    }
};
```

para establecer la relación de muchos a mucho con el modelo artículos vimos que tenemos una tabla pivote y un modelo llamado IngresosArticulos éstos tienen la siguiente estructura:
Modelo:

```
'use strict';
const {
    Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
    class IngresoArticulos extends Model {
        /**
         * Helper method for defining associations.
         * This method is not a part of Sequelize lifecycle.
         * The `models/index` file will call this method automatically.
         */
        static associate(models) {
            // define association here
        }
    };
    IngresoArticulos.init({
        ingresoId: DataTypes.INTEGER,
        articuloId: DataTypes.INTEGER,
        articulo: DataTypes.STRING,
        cantidad: DataTypes.INTEGER,
        precio: DataTypes.INTEGER,
        descuento: DataTypes.INTEGER
    }, {
        sequelize,
        modelName: 'IngresoArticulos',
    });
    return IngresoArticulos;
};
```

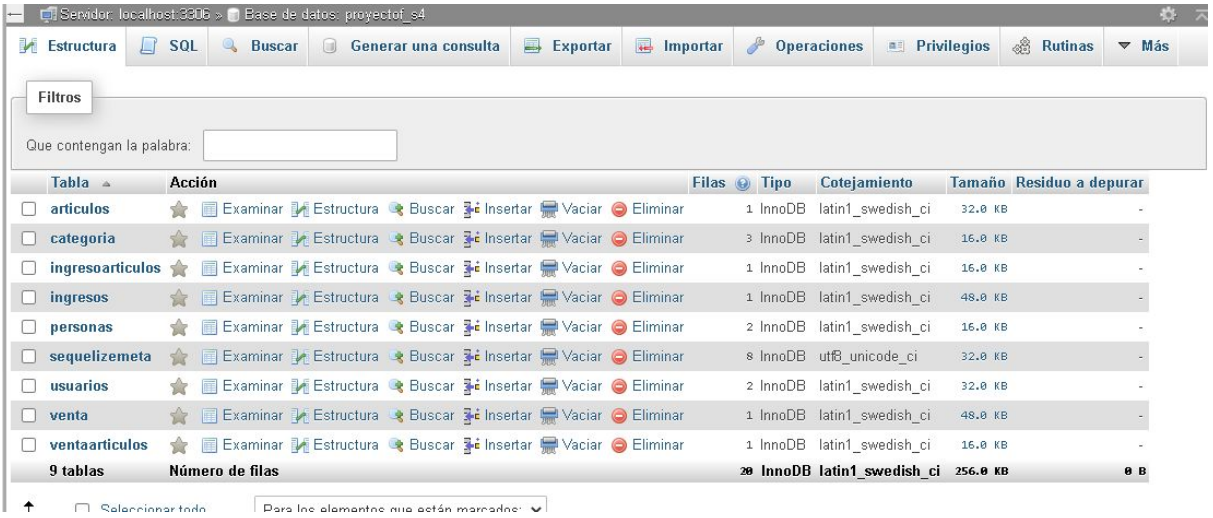
migración:

```
'use strict';
module.exports = {
    up: async(queryInterface, Sequelize) => {
        await queryInterface.createTable('IngresoArticulos', {
            id: {
                allowNull: false,
                autoIncrement: true,
                primaryKey: true,
                type: Sequelize.INTEGER
            },
            ingresoId: {
                type: Sequelize.INTEGER
            },
            articuloId: {
                type: Sequelize.INTEGER
            },
            articulo: {
                type: Sequelize.STRING

            },
            cantidad: {
                type: Sequelize.INTEGER
            },
            precio: {
                type: Sequelize.INTEGER
            },
            descuento: {
                type: Sequelize.INTEGER
            },
            createdAt: {
                allowNull: false,
                type: Sequelize.DATE
            },
            updatedAt: {
                allowNull: false,
                type: Sequelize.DATE
            }
        });
    },
    down: async(queryInterface, Sequelize) => {
        await queryInterface.dropTable('IngresoArticulos');
    }
};
```

Este mismo proceso se realiza para los demás modelos y sus debidas migraciones dependiendo de su relación.

el tener listo los modelos y las migraciones ejecutamos el siguiente comando:
sequelize db:migrate

si todo sale bien en las migraciones , vamos a nuestro navegador en phpmyadmin y lo actualizamos para ver los cambios,damos click en la base de datos del proyecto,podemos observar que se han migrado cada uno de nuestro modelos con sus respectivos atributos .



referencias:
https://sequelize.org/master/manual/migrations.html
https://github.com/sequelize/cli#documentation