Sprint #2

MY APP PERSISTENTE

Nicolás Palermo

SPRINT #1 VS SPRINT #2

Etapa #1 - Data Store

1. LA INFORMACIÓN DEBE ESTAR ALMACENADA EN UNA DB - 21PTS
2. EL USUARIO DEBE TENER UN ESTADO DE SUSPENSION (CONDITIONAL ATTRIBUTE)
3. CREAR UN CRUD PARA "METODOS DE PAGOS" - 5PTS
4. LOS USUARIOS DEBEN PODER AGENDAR SUS DIRECIONES Y PODER SELECIONARLAS CUANDO REALIZAN UN PEDIDO - 8PTS
5. REALIZAR UN MODELADO DE LAS SIGUIENTES TABLAS "USUARIO, PRODUCTOS, PEDIDOS, METODOS DE PAGO, AGENDA"

Sequelize - Mysql

Se creo un archivo en **src/database/index.js** con las siguientes funciones propuestas

a. Function Connect

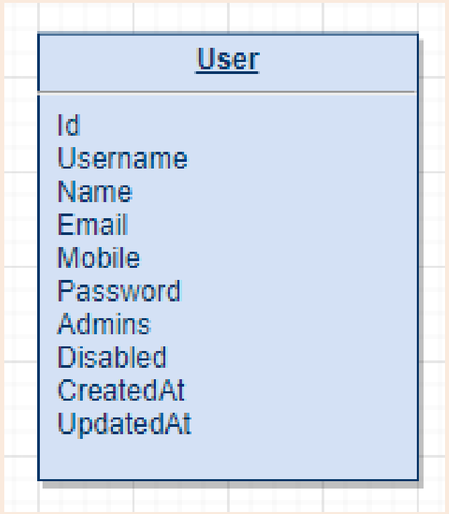
b. Function createSyncTables

c. Function getModel

d. Function dbConexion

Modelos de la DB

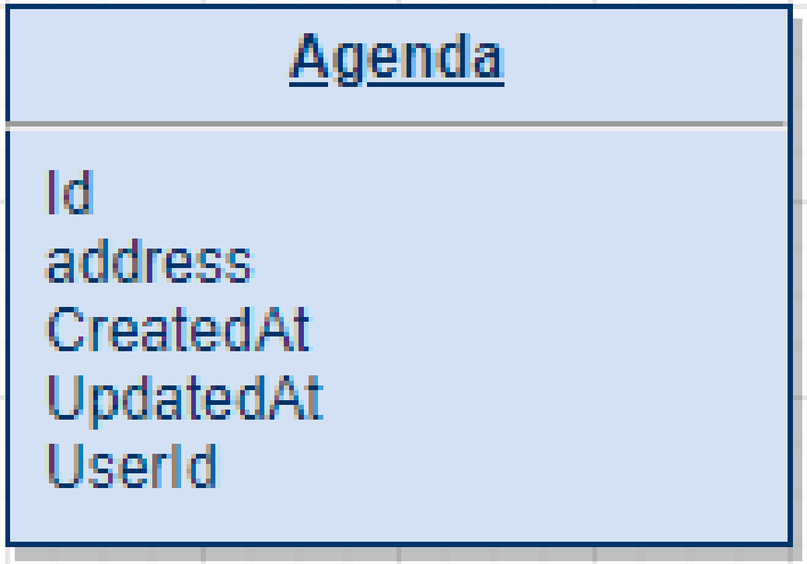
ROUND 2




```
const User = connection.define('User', {
    username: {
      type: DataTypes.STRING,
      unique: true,
      allowNull: false
    },
    name: {
      type: DataTypes.STRING,
      allowNull: false
    },
    email: {
      type: DataTypes.STRING,
      unique: true,
      allowNull: false
    },
    mobile:{
        type: DataTypes.STRING,
        allowNull: false
    },
    password: {
      type: DataTypes.STRING,
      allowNull: false
    },
    admins: {
      type: DataTypes.BOOLEAN,
      allowNull: false
    },
    disabled: {
        type: DataTypes.BOOLEAN,
        allowNull: false
    }
  }, {
    modelName: 'User',
    tableName: 'User'
  });
```

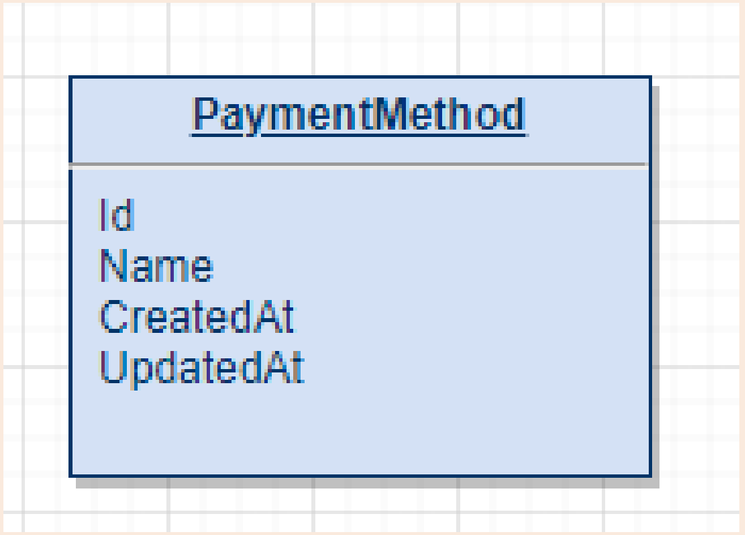
Modelado de User


```
const Agenda = connection.define('Agenda', {
    address: {
      type: DataTypes.STRING,
      allowNull: false
    }
  }, {
    modelName: 'Agenda',
    tableName: 'Agenda'
  });
```



Modelado de Agenda

PaymentMethod



Modelado de PaymentMethod

```
const PaymentMethod = connection.define('PaymentMethod', {
    name: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    }
  }, {
    modelName: 'PaymentMethod',
    tableName: 'PaymentMethod'
  });
```

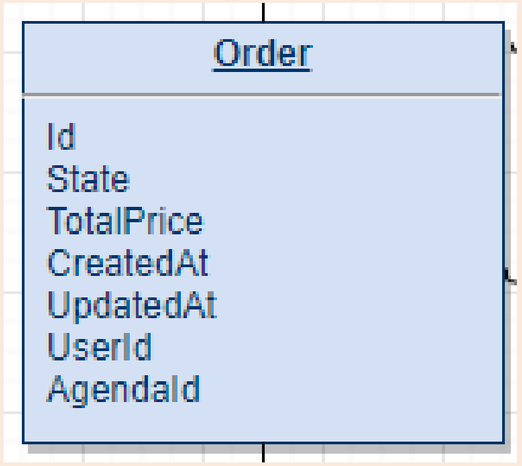
Products

Product
Name Description Picture Price CreatedAt UpdatedAt

```
const Product = connection.define('Product', {
    name: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    },
    description:{
        type: DataTypes.STRING,
        allowNull: false
    },
    picture:{
        type: DataTypes.STRING,
        allowNull: false
    },
    price: {
        type: DataTypes.INTEGER,
        allowNull: false
    }
  }, {
    modelName: 'Product',
    tableName: 'Product'
  });
```

Modelado de Product

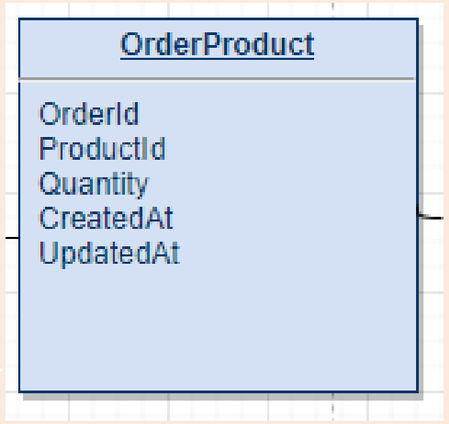
Order




```
const Order = connection.define('Order', {
    state: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    },
   totalPrice:{
        type: DataTypes.INTEGER,
        allowNull: false
    }
  }, {
    modelName: 'Order',
    tableName: 'Order'
  });
```

Modelado de Order

OrderProduct




```
const OrderProduct = connection.define('OrderProduct', {
    OrderId: {
      type: DataTypes.INTEGER,
      allowNull: false,
      references: {
        model: orderModel,
        key: "id"
      }
  },
  ProductId:{
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: producModel,
      key: "id"
    }
  },
   quantity:{
        type: DataTypes.INTEGER,
        allowNull: false
    }
  }, {
    modelName: 'OrderProduct',
    tableName: 'OrderProduct'
  });
```

Modelado de OrderProduct

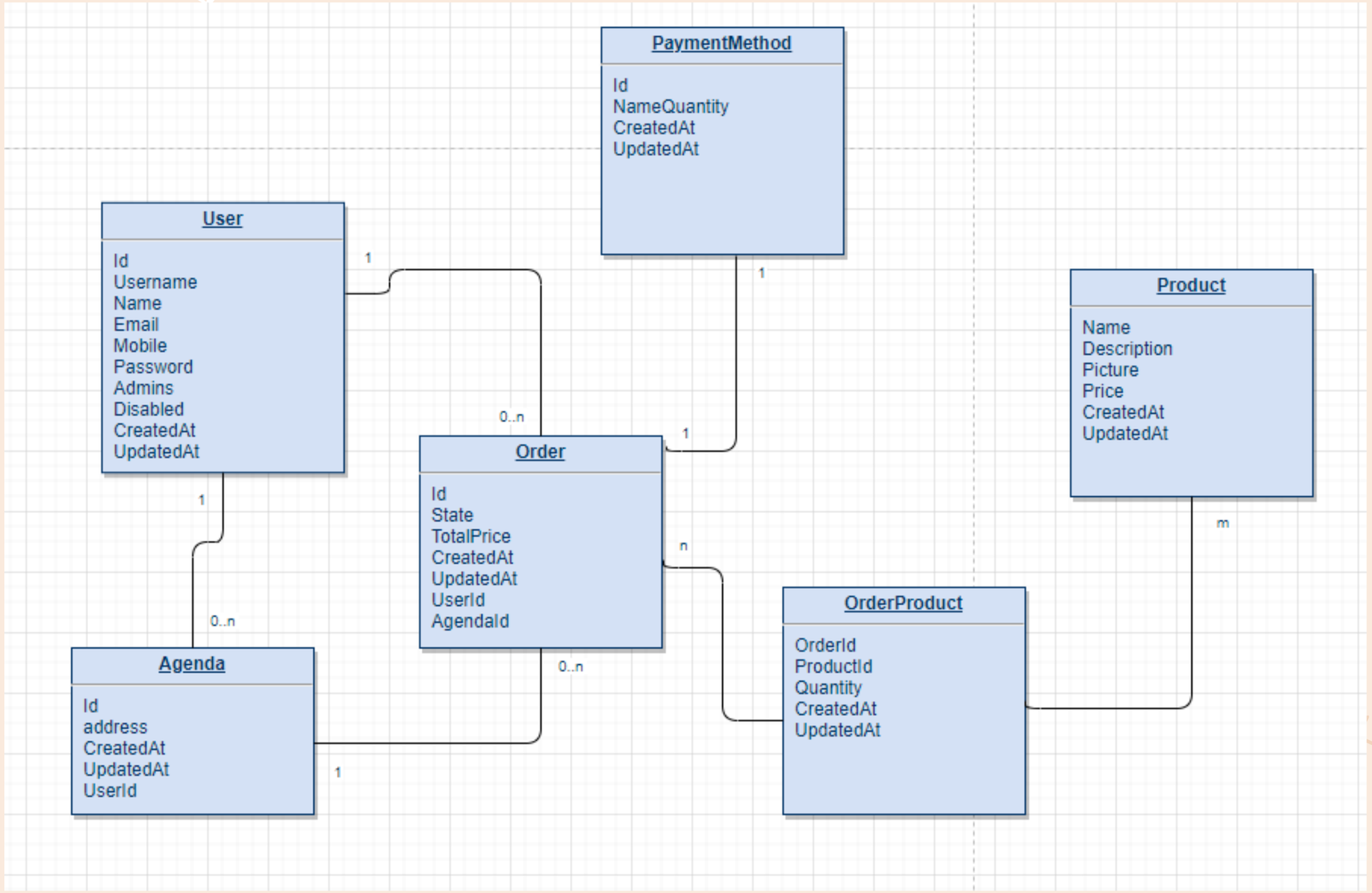


Diagrama UML

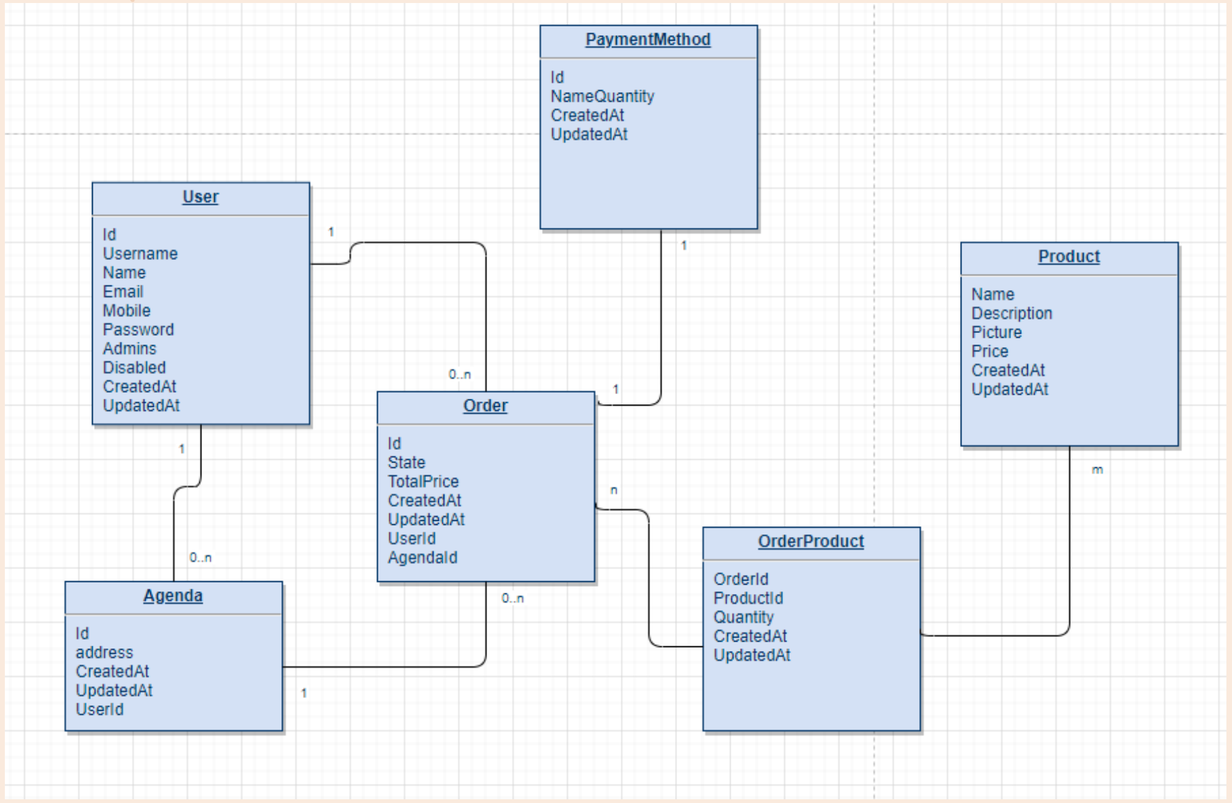


Diagrama UML

```
// Síncronizamos las relaciones
//1
models.User.hasMany(models.Agenda);  // 1-N
models.Agenda.belongsTo(models.User) // N-1
//2
models.User.hasMany(models.Order) // 1-N
models.Order.belongsTo(models.User) // N-1
//3
models.Agenda.hasMany(models.Order) // 1-N
models.Order.belongsTo(models.Agenda) // N-1
//4
models.Order.belongsToMany(models.Product, {through: models.OrderProduct});
models.Product.belongsToMany(models.Order, {through: models.OrderProduct});
```

Sequelize Associations

SERVER UP

Procedemos a la elaboración del archivo **src/server.js**. El cual contendrá los middlewares globales necesarios y los routers

Router (Folder)

Aqui tendemos los archivos destinados a los diferentes routers propuestos para el sprint#2

a. user.js

b. paymentMethod.js

c. order.js

d. agenda.js

e. product.js

Creamos la función Main donde:

- Levantará la DB

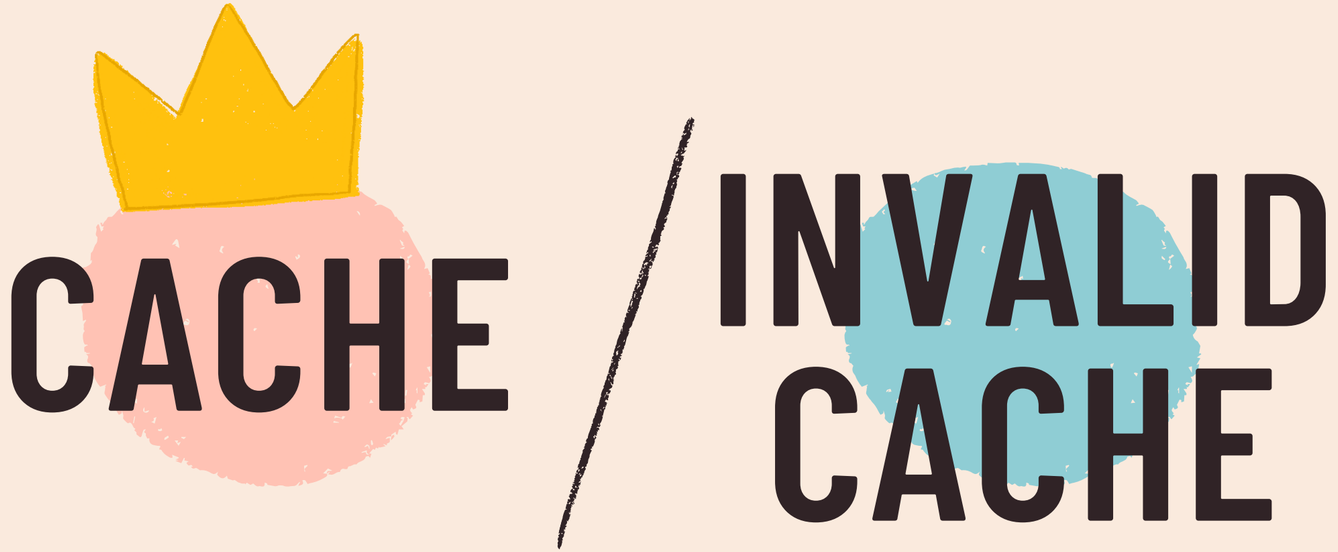
- Creará unos registros default

- Levantará el Servidor

Etapa #2 Funcionabilidad

* No haya usuarios duplicados -3pts
* Uso de Helmet - 3pts
* Acceso a los endpoints con JWT -5pts
* Password debe estar encriptada
* Usuario puede modificar-eliminar pedido antes de cerrarlo - 13pts

Etapa #3 - Cache



- Capa Cache para mostrar todos los productos -13pts

- Limpiar Cache al momento de actualizar y crear producto - 3pts

Etapa #4 - test y otros

1. RUTINA TEST PARA EL REGISTRO DE NUEVOS USUARIOS - 5PTS
2. TENER TODA LA DOCUMENTACIÓN EN SWAGGER - 2PTS
3. REALIZAR UN MERGEA A MASTER - 3TPS
4. ACTUALIZAR LAS INSTRUCCIONES DEL READM.MD - 1PT