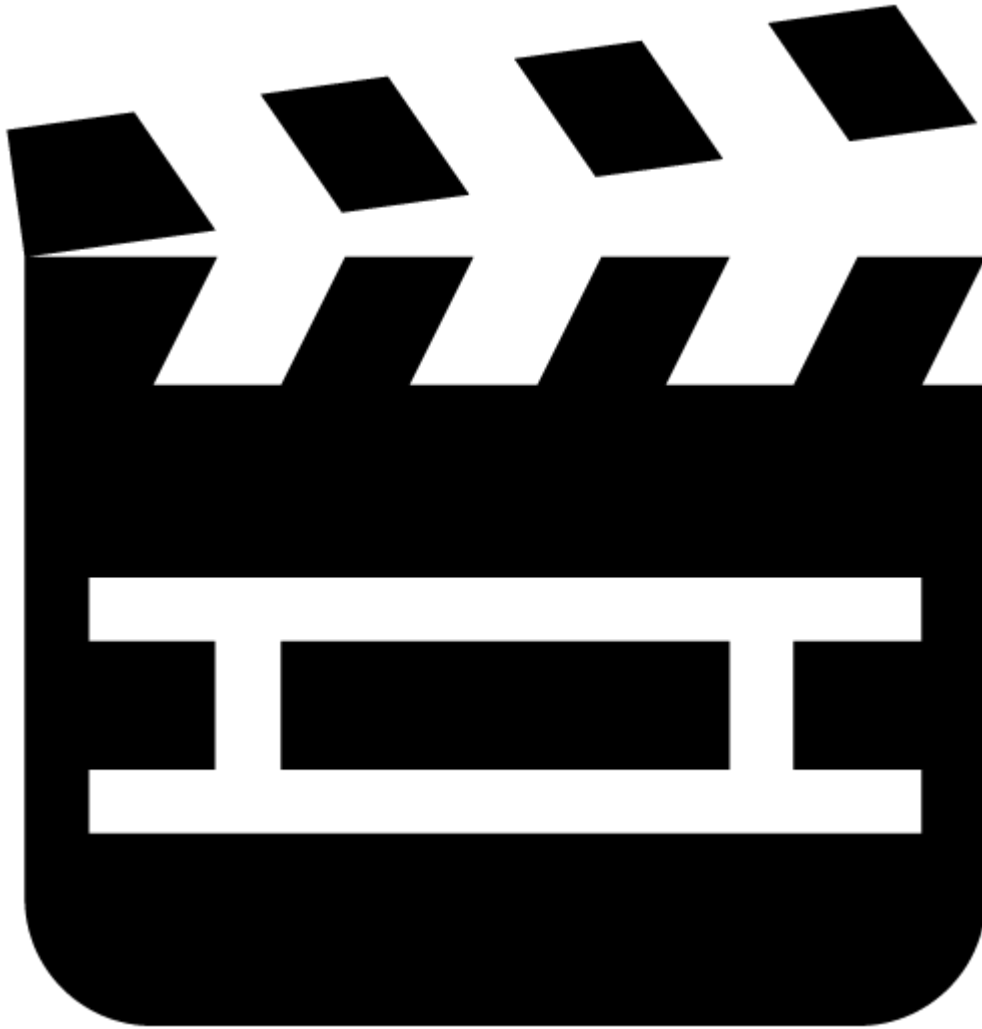


PROJECT CINELMO



David Naranjo García

2º Dam Tarde

Curso 2017-18

Índice

Introducción

UML y explicación de las clases

Diagramas E/R y base de datos

Bocetos/Mockups

Servicio RESTFul

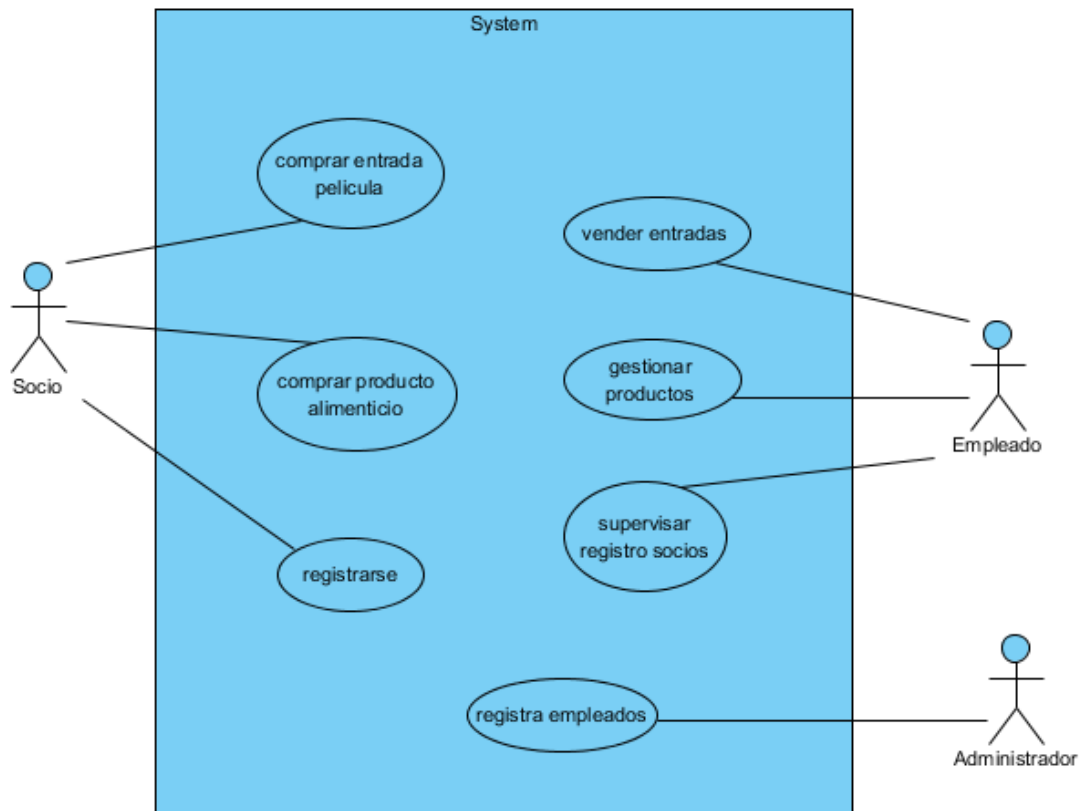
CRUD dentro del Servicio

Aplicación cliente

<https://github.com/Davnargar>

Introducción

El proyecto se basa en la gestión de un cine donde un cliente podrá comprar la entrada o pase de una película y además pagar si quiere algún tipo de alimento con la entrada, bien siendo comida, bebida o ambas. El programa se centrará en la función de dos usuarios o “actores” que realizarán distintas actividades y que veremos en la siguiente imagen.



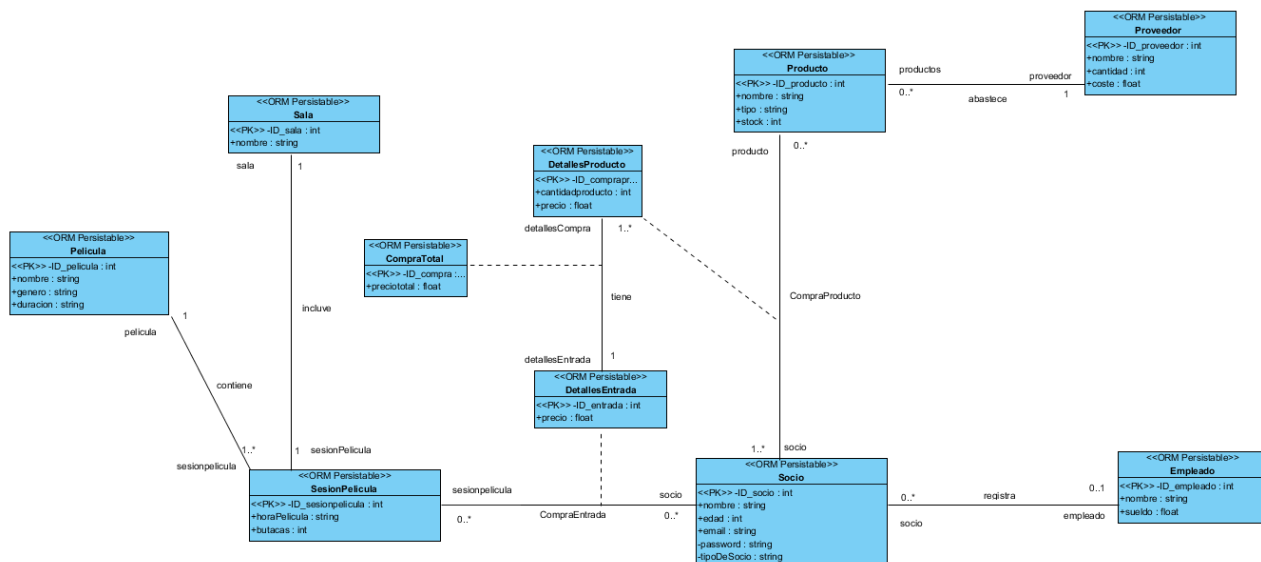
Como podemos comprobar que realmente hay tres actores pero no tendremos registro del administrador por lo que “administrador” no estará en la base de datos, así que encontramos los actores “Socio” y “Empleado”.

El socio será el cliente que se registra en la aplicación y una vez registrado podrá comprar tanto la entrada de la película como los alimentos. Un socio no podrá acceder a ninguna de estas funciones si no se ha registrado, la manera de registrarse puede ser por la aplicación misma o un empleado puede registrar a dicho cliente desde su cuenta.

La función que desarrolla el empleado en este proyecto principalmente será la de vender las entradas, gestionar los productos (controlar que hay stock, poner un precio y entregarlos a los clientes) y supervisar el registro de socios. Vamos a centrarnos en la supervisión del registro ya que antes se habló de otra función; un empleado podrá ver el número de clientes que hay registrado todas sus características por si ocurre algún problema poder facilitar la solución, pero además si un cliente accede al cine, compra una entrada manualmente y dicho cliente quiere registrarse en ese momento, el empleado puede hacerlo sin problemas.

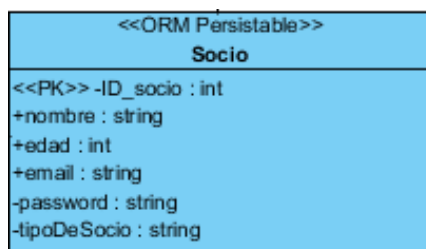
Construcción del programa

El programa está compuesto por las siguientes clases:



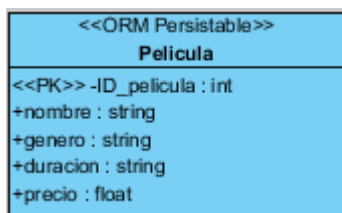
Ahora procederemos a explicar el contenido de cada clase por separado:

SOCIO



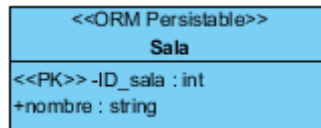
En esta clase registraremos al cliente en la que contendrá los datos que identifican a cada socio, la aplicación le pedirá un nombre, su edad ya que tiene que ser mayor de edad para estar registrado, su correo electrónico, contraseña y puede elegir el tipo de socio que quiere ser “socio estándar o premium”, una vez se ha registrado se le asignará un identificador autoincrementable para que los empleados puedan hacer un seguimiento más sencillo este identificador al ser autoincrementable no podrá repetirse.

PELICULA



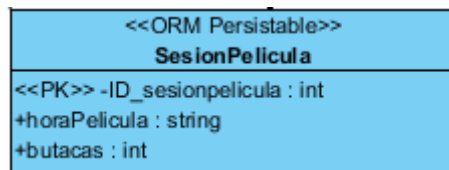
Aquí registraremos las películas que estarán disponibles en el cine, las añadiremos por nombre, género (comedia, drama, acción, etc.), la duración que tendrá y el precio, el identificador de la película es del mismo tipo que la clase SOCIO.

SALA



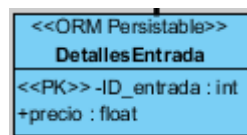
La clase sala solo contiene el nombre de la sala y su correspondiente identificador ya que su nombre nos servirá para determinar los detalles de la compra de la entrada que se verá más adelante.

SESION PELICULA



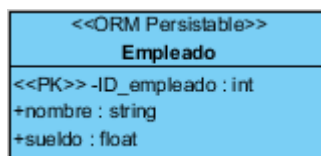
La clase de la sesión de la película solo contendrá la hora de la película para determinar cuando una sala está disponible o no y las butacas, más adelante mostraremos las relaciones que tienen las clases para evitar dudas.

DETALLES ENTRADA



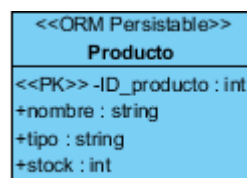
La clase “DetallesEntrada” es una clase de asociación entre el socio y la sesión de la película por lo que la clase únicamente añadirá el precio de la entrada, más los datos de SesionPelicula y Socio.

EMPLEADO



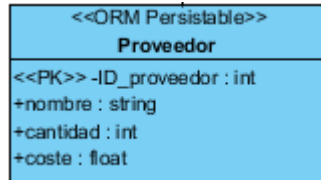
La clase de empleado contiene el nombre de los empleados, el sueldo que tiene cada uno y su identificador.

PRODUCTO



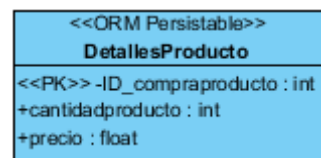
Registramos el nombre del producto, su tipo (comida, bebida) y el stock que tienen en el recinto, además del identificador como tienen todas las clases.

PROVEEDOR



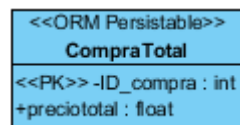
El proveedor no contará como actor ya que solo necesitamos saber quién distribuye (nombre), la cantidad que solicitamos y lo que nos costará el abastecimiento, más su identificador.

DETALLESPRODUCTO



Es la clase resultante entre la asociación de “Producto” y “Socio” en el que el Socio podrá elegir la cantidad del producto que quiere y el precio que le costará la compra.

COMPRATOTAL



Como bien indica su nombre, esta clase reunirá el total entre “DetallesProducto” y “DetallesEntrada”.

Diagrama Entidad/Relación simplificado

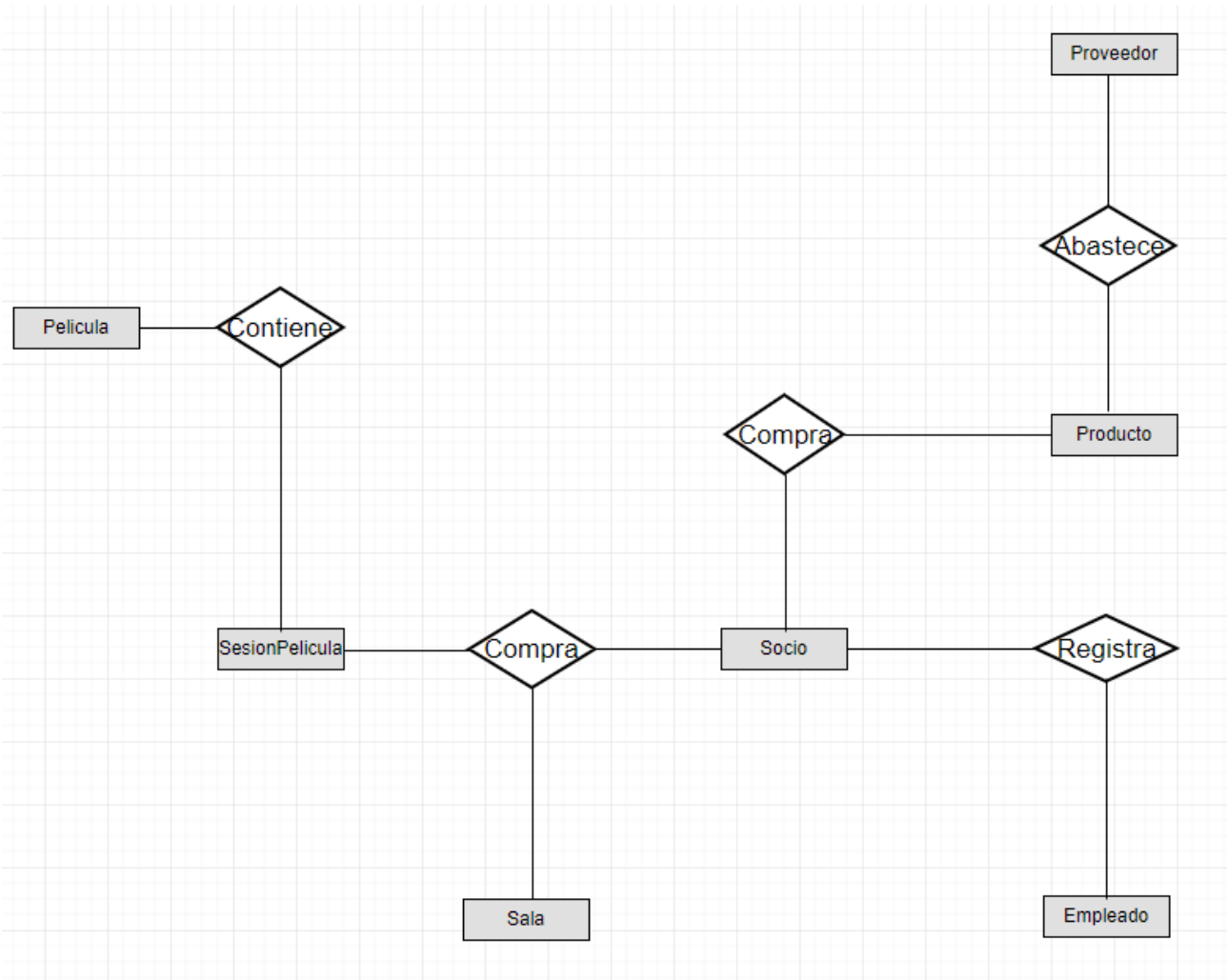


Diagrama Entidad/Relación con conectores en Visual Paradigm

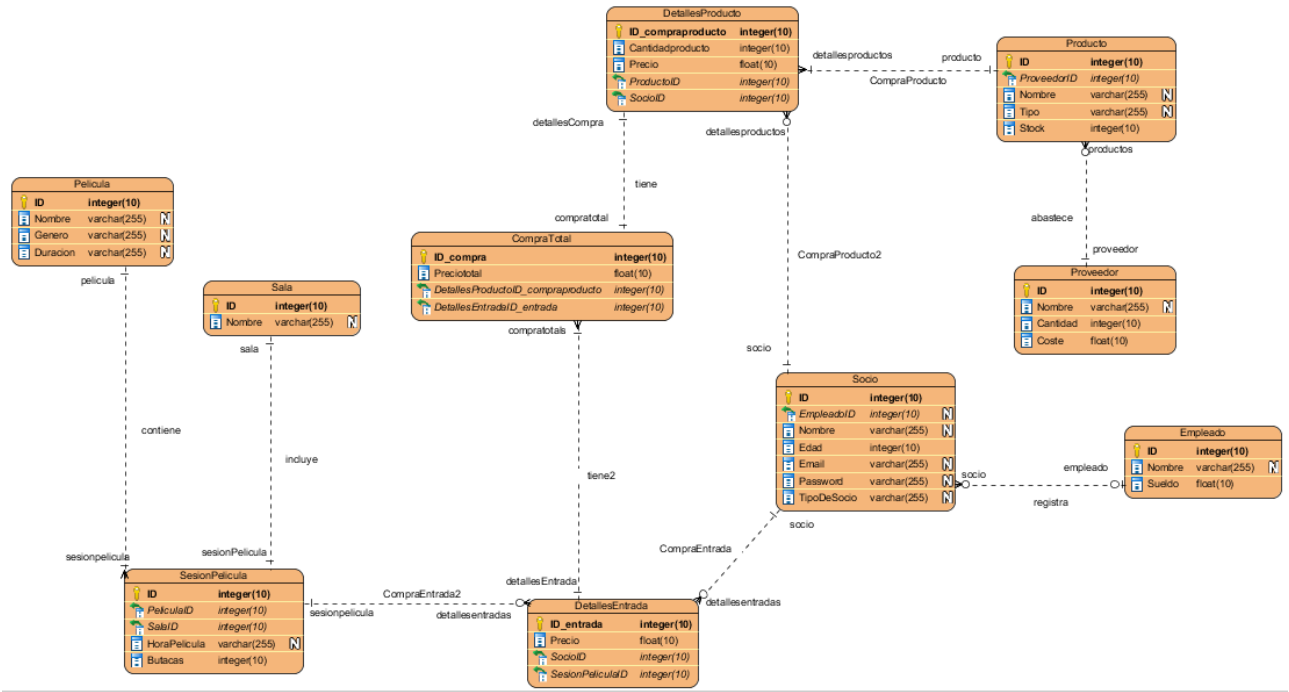
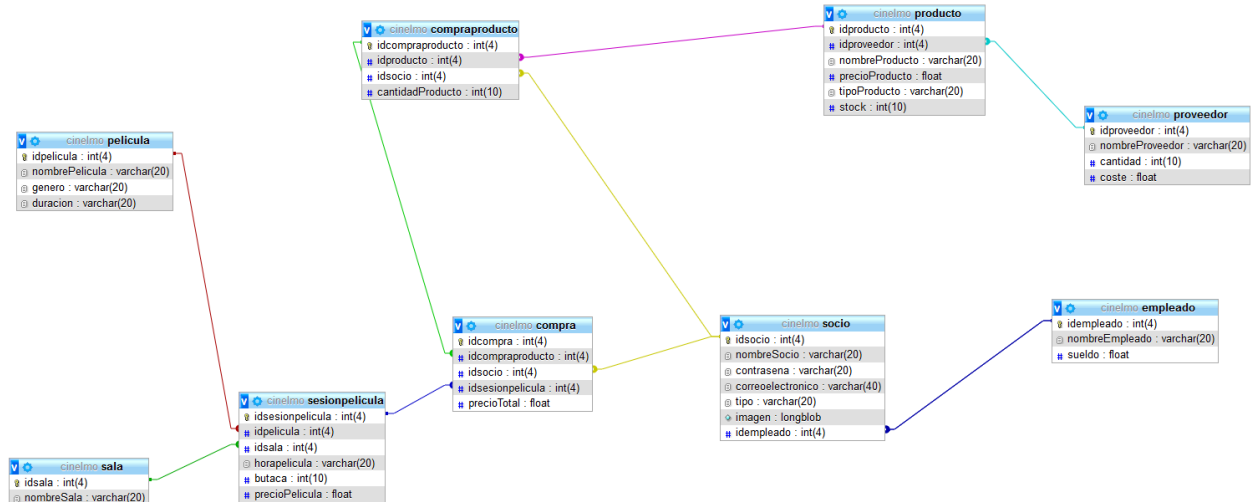


Diagrama Entidad/Relación implementado en phpmyadmin



Código de la base de datos

COMPRA

```

CREATE TABLE `compra` (
  `idcompra` int(4) NOT NULL,
  `idcompraproducto` int(4) NOT NULL,
  `idsocio` int(4) NOT NULL,
  `idsesionpelicula` int(4) NOT NULL,
  `precioTotal` float NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
    
```

COMPRAPRODUCTO

```

CREATE TABLE `compraproducto` (
  `idcompraproducto` int(4) NOT NULL,
  `idproducto` int(4) NOT NULL,
  `idsocio` int(4) NOT NULL,
  `cantidadProducto` int(10) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
    
```

EMPLEADO

```

CREATE TABLE `empleado` (
  `idempleado` int(4) NOT NULL,
  `nombreEmpleado` varchar(20) NOT NULL,
  `sueldo` float NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
    
```

PELICULA

```

CREATE TABLE `pelicula` (
  `idpelicula` int(4) NOT NULL,
  `nombrePelicula` varchar(20) NOT NULL,
  `genero` varchar(20) NOT NULL,
  `duracion` varchar(20) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
    
```


PRODUCTO

```
CREATE TABLE `producto` (  
  `idproducto` int(4) NOT NULL,  
  `idproveedor` int(4) NOT NULL,  
  `nombreProducto` varchar(20) NOT NULL,  
  `precioProducto` float NOT NULL,  
  `tipoProducto` varchar(20) NOT NULL,  
  `stock` int(10) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

PROVEEDOR

```
CREATE TABLE `proveedor` (  
  `idproveedor` int(4) NOT NULL,  
  `nombreProveedor` varchar(20) NOT NULL,  
  `cantidad` int(10) NOT NULL,  
  `coste` float NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

SALA

```
CREATE TABLE `sala` (  
  `idsala` int(4) NOT NULL,  
  `nombreSala` varchar(20) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

SESIONPELICULA

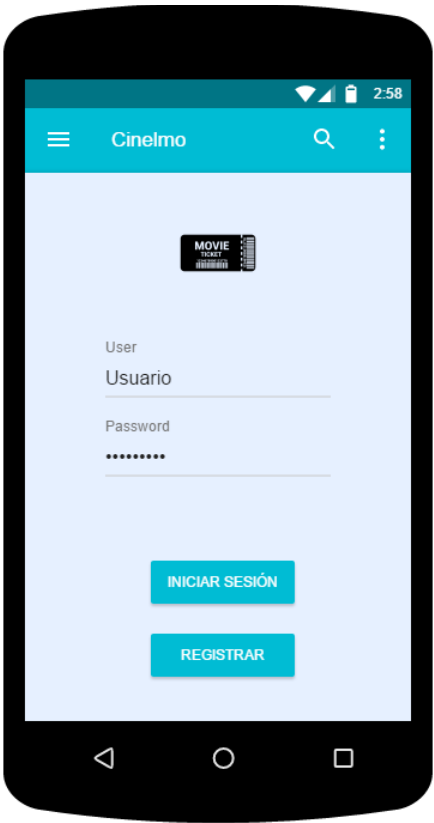
```
CREATE TABLE `sesionpelicula` (  
  `idsesionpelicula` int(4) NOT NULL,  
  `idpelicula` int(4) NOT NULL,  
  `idsala` int(4) NOT NULL,  
  `horapelicula` varchar(20) NOT NULL,  
  `butaca` int(10) NOT NULL,  
  `precioPelicula` float NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

SOCIO

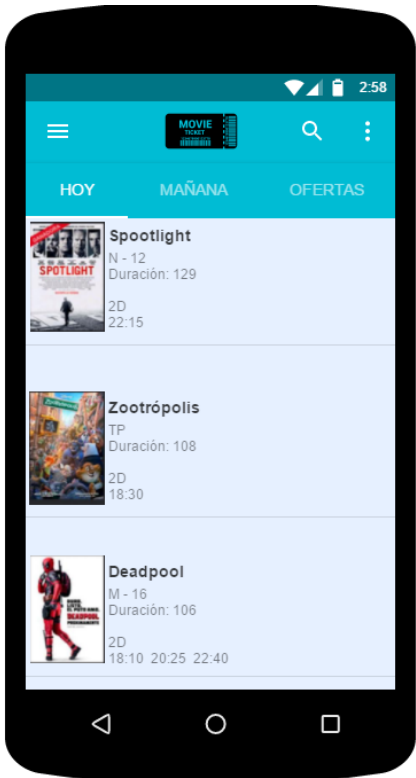
```
CREATE TABLE `socio` (  
  `idsocio` int(4) NOT NULL,  
  `nombreSocio` varchar(20) NOT NULL,  
  `contrasena` varchar(20) NOT NULL,  
  `correoelectronico` varchar(40) NOT NULL,  
  `tipo` varchar(20) NOT NULL,  
  `imagen` longblob,  
  `idempleado` int(4) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Bocetos de aplicación Web/Móvil: Modo Día y Modo Nocturno

Aplicación versión móvil modo diurno: Inicio de sesión.



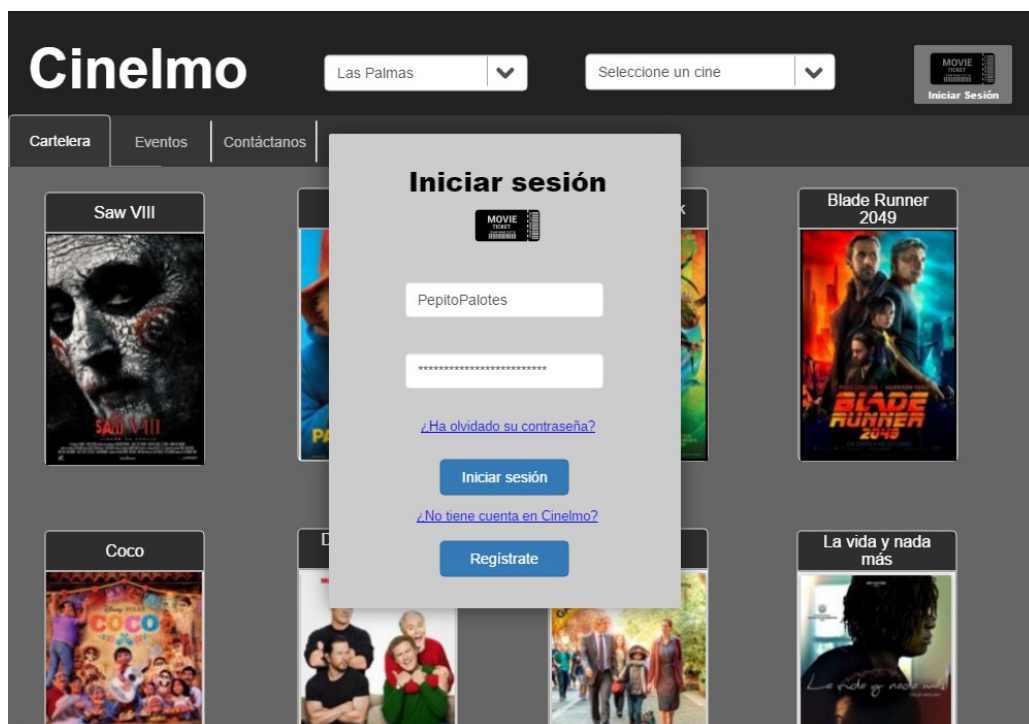
Aplicación versión móvil modo diurno: Pantalla principal.



Aplicación versión móvil modo diurno: Selección de butaca.



Aplicación versión web modo nocturno: Inicio de sesión.



Aplicación versión web modo nocturno: Pantalla de película 1.

Cineldo

Las Palmas


Selecione un cine

PepitoPalotes

Cartelera

Eventos

Contáctanos



Saw VIII

M-18 Terror

Sesión:

18:0019:1020:2021:3022:40

Tráiler:

SAW VIII - Tráiler oficial - En cines 24 noviembre

Butacas:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Aplicación versión web modo nocturno: Pantalla de película 2.

Cineldo

Las Palmas


Selecione un cine

PepitoPalotes

Cartelera

Eventos

Contáctanos



Saw VIII

M-18 Terror

Sesión:

18:0019:1020:2021:3022:40

Tráiler:

SAW VIII - Tráiler oficial - En cines 24 noviembre

Butacas:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Precio total: 5€

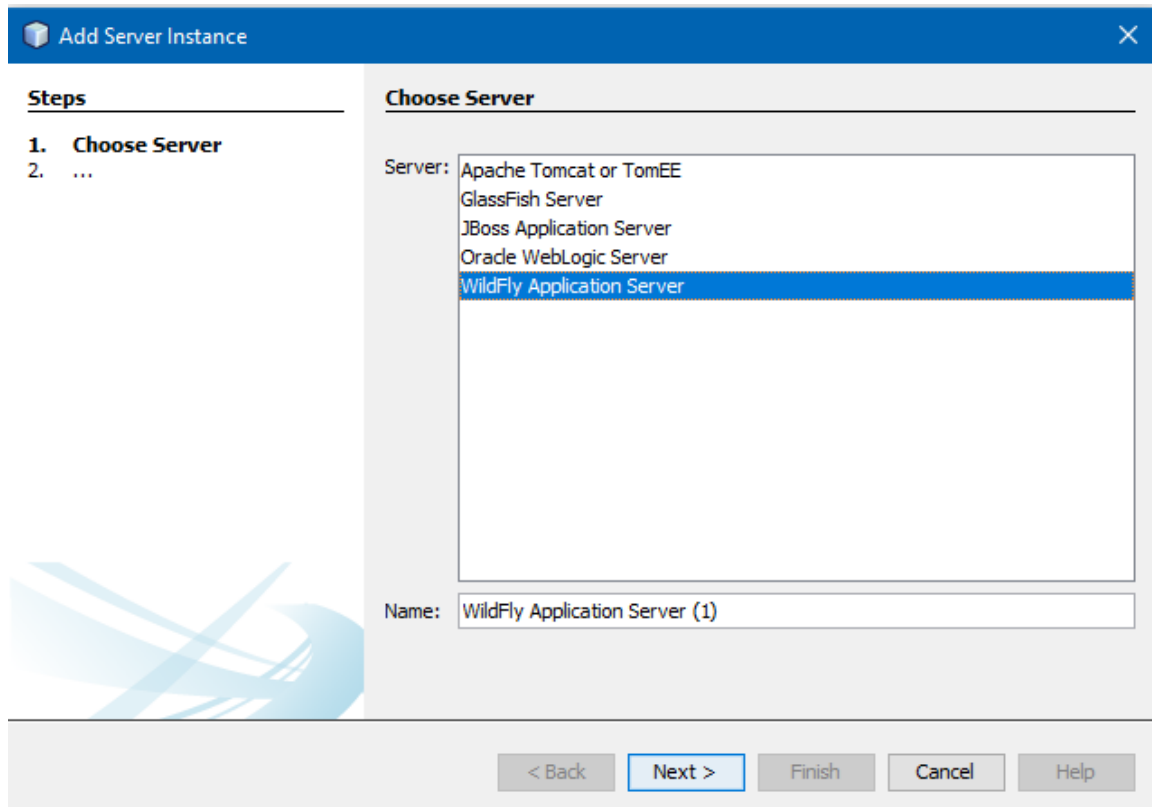
Aceptar

Rechazar

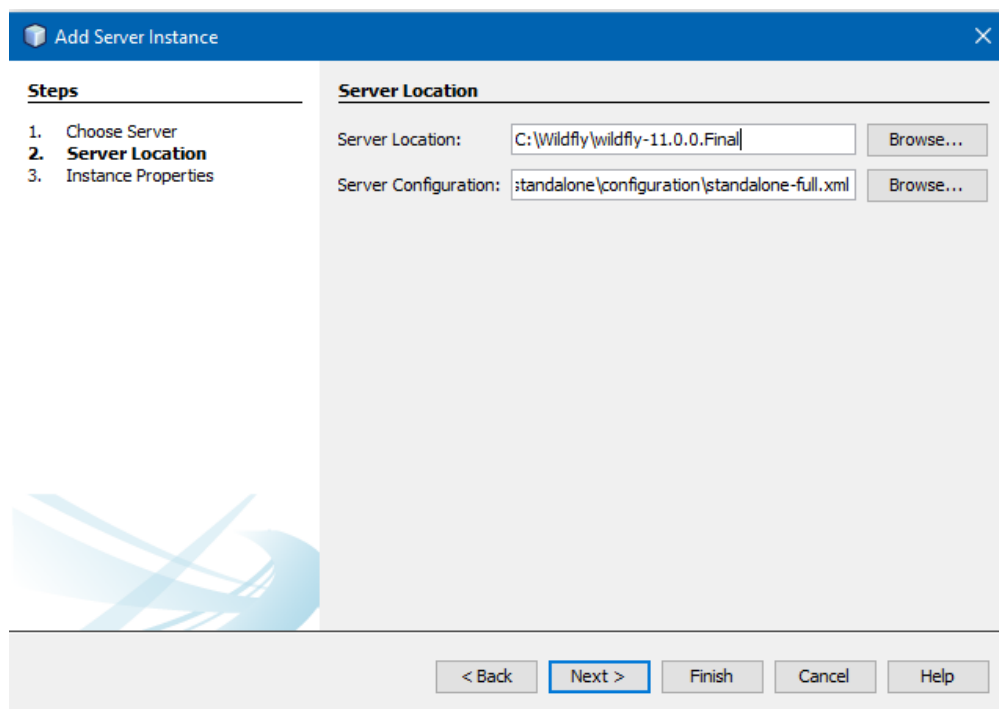
Servicio RESTful con Wildfly y Maven

Una vez tengamos implementada la base de datos en phpmyadmin e interconectada, pasamos a crear el servidor restful en netbeans utilizando Wildfly.

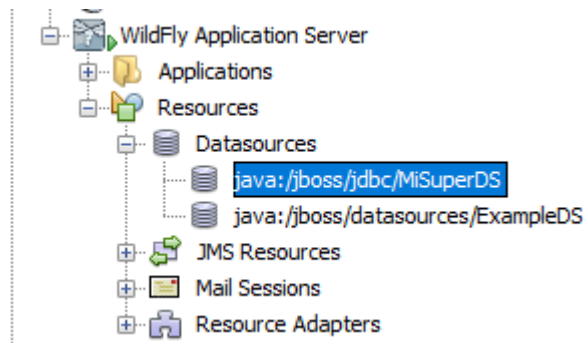
Añadimos primero el servidor en la pestaña “Services”>”Servers”>Add server.



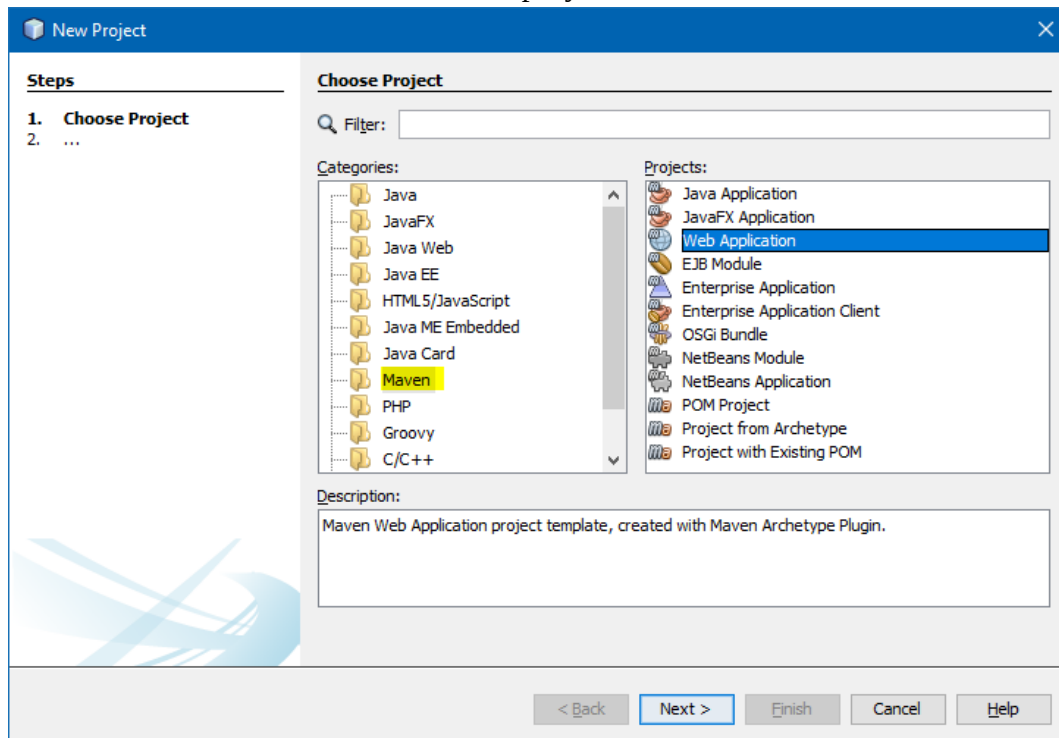
Hacemos clic en siguiente y buscamos la carpeta de wildfly que debemos tener descargado previamente.



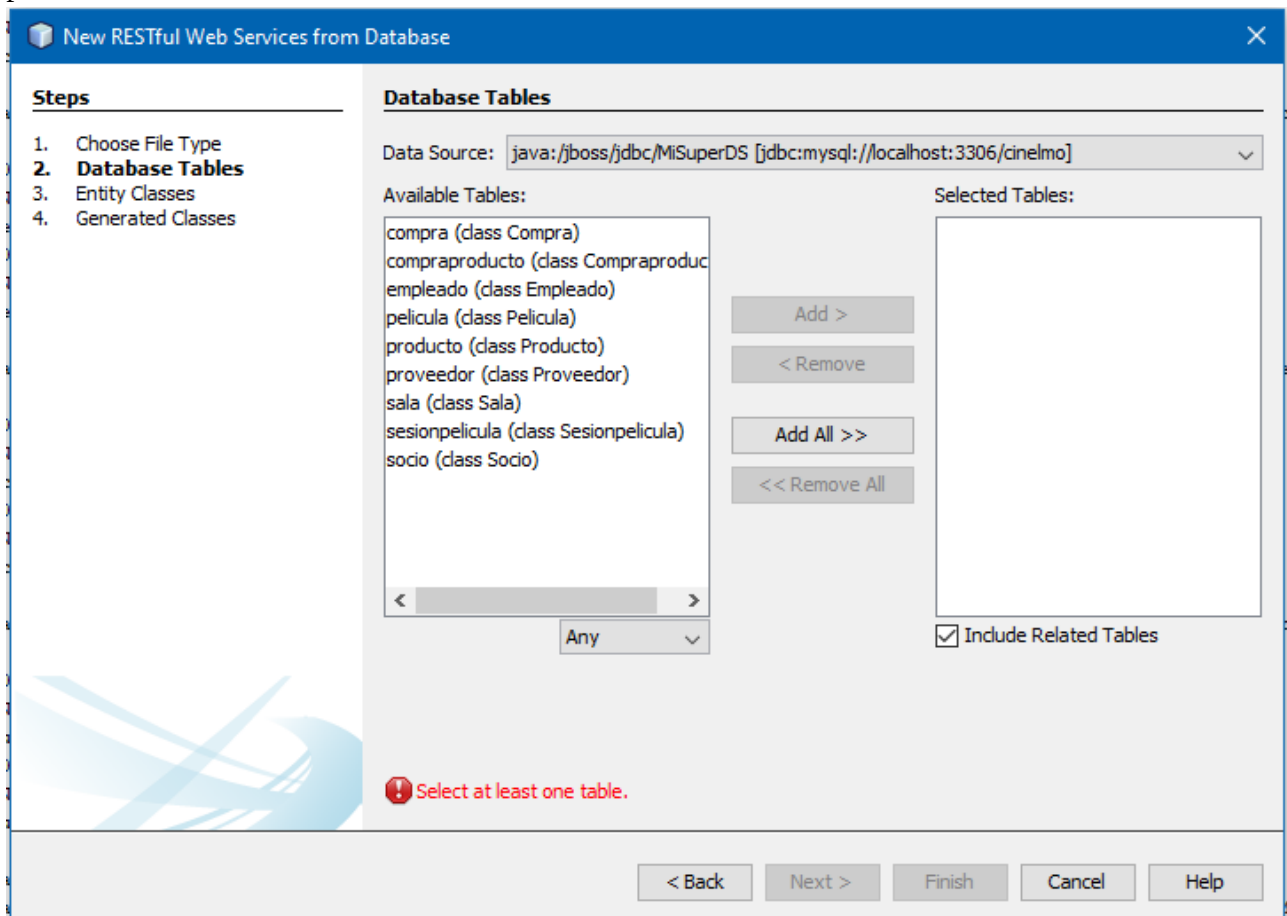
Hacemos clic en siguiente y en “Finalizar”. Una vez que tengamos creado el servidor wildfly, podremos iniciarlo con clic derecho “Start” y una vez iniciado nos aparecerán unos menús desplegables en el que podemos implementar los recursos de datos. En nuestro caso “java:/jboss/jdbc/MiSuperDS”.



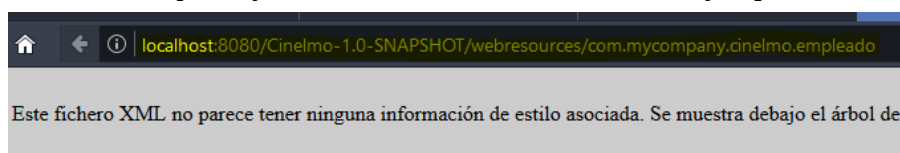
Con el recurso de datos creado, vamos a crear el proyecto web con Maven.



Nosotros lo llamaremos “Cineldo”, dentro del proyecto nos dirigimos al recurso de paquetes y creamos un nuevo servicio RESTful para la base de datos para así unirlo con el recurso creado hace poco.



Cuando hayamos creado el servicio restful podemos comprobarlo en cualquiera de las tablas, yendo a RESTful Web Services, clic derecho en cualquiera de las opciones que aparecen y elegimos “Test Resource Uri”. Cuando se ejecute nos llevará al navegador y aparecerán los datos que estén introducidos en la tabla que hayamos seleccionado, usaremos de ejemplo la tabla “empleado”.



```
-<collection>
-  <empleado>
    <idempleado>1</idempleado>
    <nombreEmpleado>DavidNaranjo</nombreEmpleado>
    <sueldo>1500.0</sueldo>
  </empleado>
-  <empleado>
    <idempleado>2</idempleado>
    <nombreEmpleado>Carlos</nombreEmpleado>
    <sueldo>1250.0</sueldo>
  </empleado>
-  <empleado>
    <idempleado>5</idempleado>
    <nombreEmpleado>PedroRamos</nombreEmpleado>
    <sueldo>1550.0</sueldo>
  </empleado>
</collection>
```

En la parte subrayada apreciamos el enlace dedicado a empleado, junto con el puerto de Wildfly “localhost:8080” y la información viene recogida en tipo xml.

Si queremos que aparezca como un json debemos de quitarlo manualmente en “resource packages” el paquete del proyecto, “service” y en el caso del ejemplo “EmpleadoFacadeRest.java”. El resultado sería el siguiente:

```
@PersistenceContext(unitName = "com.mycompany_Cinelmo_war_1.0-SNAPSHOTPU")
private EntityManager em;

public EmpleadoFacadeREST() {
    super(Empleado.class);
}

@POST
@Override
@Consumes(MediaType.APPLICATION_JSON)
public void create(Empleado entity) {
    super.create(entity);
}

@PUT
@Path("/{id}")
@Consumes(MediaType.APPLICATION_JSON)
public void edit(@PathParam("id") Integer id, Empleado entity) {
    super.edit(entity);
}

@DELETE
@Path("/{id}")
public void remove(@PathParam("id") Integer id) {
    super.remove(super.find(id));
}

@GET
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
public Empleado find(@PathParam("id") Integer id) {
    return super.find(id);
}

@GET
@Override
@Produces(MediaType.APPLICATION_JSON)
public List<Empleado> findAll() {
    return super.findAll();
}
```


Aplicación cliente con ReactJS y Meteor

Una vez que tenemos tanto la base de datos como el servidor, nos dedicaremos a crear el cliente, en el CMD crearemos el proyecto con el comando “meteor create cinelmoProject” (el nombre variará dependiendo de cómo se quiera nombrar el proyecto. Por ejemplo, con prueba sería así:

```
C:\Users\david\proyecto-react>meteor create prueba
Created a new Meteor app in 'prueba'.

To run your new app:
  cd prueba
  meteor

If you are new to Meteor, try some of the learning resources here:
  https://www.meteor.com/tutorials

meteor create --bare to create an empty app.
meteor create --full to create a scaffolded app.
```

Al tener ya creado el proyecto, entramos en su carpeta y le instalamos los paquetes que queramos implementar gracias a la librería NPM. Se implementarán en este proyecto los siguientes paquetes:

- meteor npm install --save react react-dom (Para trabajar con react en meteor)
- meteor add twbs:bootstrap (Bootstrap)
- meteor add http (Para acceder al servicio web)
- meteor npm install --save react-router-dom (Para el enrutamiento)

Una vez se hayan instalado ejecutamos el cliente añadiendo “meteor” o “meteor npm start”.

```
=> Started proxy.
=> A patch (Meteor 1.6.0.1) for your current release is available!
    Update this project now with 'meteor update --patch'.
=> Started MongoDB.
=> Started your app.











=> App running at: http://localhost:3000/
    Type Control-C twice to stop.
```

Debemos tener el puerto 3000 libre para ello.

CRUD de Socios

Para el ejemplo de cómo se ha creado Cinelmo usaremos la tabla de los socios dentro de la base de datos.

Página principal.

Cinelmo	Inicio	Listado	Insertar	Sign Up	Login
Nombre	Correo	Tipo	Acciones		
Tiburcio	tiburcio@correo.com	Premium			
David	david.naranjogarcia@outlook.es	Premium			
Pedro	pedro@correo.com	Premium			
Federico	federico@correo.com	Estándar			
Rafa	rafa1234@gmail.com	Premium			

CREATE o POST en la api RESTful.

Para crear un socio nos dirigiremos a la pestaña de Insertar que aparece en la cabecera de la página. Para ello creamos un formulario donde introducimos 4 campos de texto y 1 botón, los cuales 2 de los campos “Contraseña” y “Correo Electrónico” están adaptados y validados como se refieren sus nombres.

```
render() {
  return (
    <div>
      <div className="container">
        <form onSubmit={this.insert}>
          <div className="form-group">
            <label htmlFor="name">Nombre:</label>
            <input type="text" className="form-control" id="name" name="name" />
          </div>
          <div className="form-group">
            <label htmlFor="passw">Contraseña:</label>
            <input type="password" className="form-control" id="passw" name="passw" />
          </div>
          <div className="form-group">
            <label htmlFor="email">Correo Electrónico:</label>
            <input type="email" className="form-control" id="email" name="email" />
          </div>
          <div className="form-group">
            <label htmlFor="type">Tipo:</label>
            <input type="text" className="form-control" id="type" name="type" />
          </div>
          <button type="submit" className="btn btn-primary">Insertar</button>
        </form>
      </div>
      { this.state.insertSuccess &&
        <Redirect to="/" />
      }
    </div>
  );
}
```

Nombre:

Contraseña:

Correo Electrónico:

Tipo:

Insertar

Una vez creado los campos del formulario, añadimos los datos del socio que queramos insertar y hacemos clic el botón. Una vez que se realice el clic realizará la siguiente función.

```
insert(event){
  event.preventDefault();
  var formInsert = event.target;
  HTTP.call('POST', 'http://localhost:8080/Cinelmo-1.0-5NAPSHOT/webresources/com.mycompany.cinelmo.socio/', {
    data: { nombreSocio: formInsert.name.value, contrasena: formInsert.passw.value, correoelectronico: formInsert.email.value, tipo: formInsert.type.value }
  }, (error, result) => {
    if (!error) {
      this.setState({
        insertSuccess: true
      });
    }
  });
}
```

Para que entre en funcionamiento la función “insert” debemos tener los campos añadidos correctamente, por ejemplo si en el campo “Correo Electrónico” no establecemos una dirección de correo real, o un @, aparecerá que hay un error y hay que introducir un dato válido.

Correo Electrónico:

socio

Tipo:

Estándar













Incluye un signo "@" en la dirección de correo electrónico. La dirección "socio" no incluye el signo "@".

READ o GET en la api RESTful.

Como se apreció en la página principal, lo que tenemos es el READ o GET, establecido como una tabla para ordenar los socios introducidos por su identificador.

```
render() {
  return (
    <div>
      <div className="container">
        <table className="table table-hover">
          <thead>
            <tr>
              <th>Nombre</th><th>Correo</th><th>Tipo</th><th>Acciones</th>
            </tr>
          </thead>
          <tbody>
            {this.showSocio()}
          </tbody>
        </table>
      </div>
      { this.state.editSelected &&
        <Redirect to={"/edit/" + this.state.idsocio} />
      }
    </div>
  );
}
```

Nombre	Correo	Tipo	Acciones
Tiburcio	tiburcio@correo.com	Premium	 
David	david.naranjogarcia@outlook.es	Premium	 
Pedro	pedro@correo.com	Premium	 
Federico	federico@correo.com	Estándar	 
Rafa	rafa1234@gmail.com	Premium	 

Los botones de acciones están establecidos en la función socios, los cuales dan lugar a las funciones que nos faltan, Editar y Eliminar.













```
showSocio(){
  return this.state.socios.map((socio) => (
    <tr key={socio.idsocio}>
      <td>{socio.nombreSocio}</td><td>{socio.correoelectronico}</td><td>{socio.tipo}</td>
      <td>
        <button className="btn btn-danger" onClick={() => this.delete(socio.idsocio)}>
          <span className="glyphicon glyphicon-trash"></span>
        </button>
        <button className="btn btn-info" onClick={() => this.edit(socio.idsocio)}>
          <span className="glyphicon glyphicon-edit"></span>
        </button>
      </td>
    </tr>
  ));
}
```

UPDATE o PUT en la api RESTful.

Para mostrar cómo editar un socio hemos creado uno de ejemplo y para acceder a la ventana de editar debemos pulsar el botón azul, el cual dentro de showSocio llama a otra función llamada “edit” en la que nos redirecciona a otra pantalla.

```
edit(idsocio){
  this.setState({
    editSelected: true,
    idsocio: idsocio
  });
}
```

```
{ this.state.editSelected &&
  <Redirect to={"/edit/" + this.state.idsocio} />
}
```

Nombre	Correo	Tipo	Acciones
Tiburcio	tiburcio@correo.com	Premium	 
David	david.naranjogarcia@outlook.es	Premium	 
Pedro	pedro@correo.com	Premium	 
Federico	federico@correo.com	Estándar	 
Rafa	rafa1234@gmail.com	Premium	 
Socio	socio@correo.com	Estándar	 

Editar un dato es bastante similar a insertarlo salvo por la variante de comenzar con los campos a modificar rellenos con dichos datos a editar.

Nombre:

Socio

Contraseña:

.....

Correo Electrónico:

socio@correo.com

Tipo:

Estándar

Modificar

Nombre:

Leticia

Contraseña:

.....

Correo Electrónico:

leticiarm@gmail.com

Tipo:

Prémium

Modificar

```
edit(event){
  event.preventDefault();
  HTTP.call('PUT',
    'http://localhost:8080/Cineldo-1.0-SNAPSHOT/webresources/com.mycompany.cineldo.socio/'
    + this.state.idsocio, {
    data: { idsocio: this.state.idsocio, nombreSocio: this.state.name, contrasena: this.state.passw, correoelectronico: this.state.email, tipo: this.state.type }
  }, (error, result) => {
    if (!error) {
      this.setState({
        editSuccess: true
      });
    }
  });
}
```

DELETE.

Por último, eliminar un usuario o socio, en la página principal se puede realizar de una manera bastante sencilla, seleccionamos el botón rojo con el dibujo de la papelera del socio que queremos eliminar y hacemos clic.

Leticia

leticiarm@gmail.com

Prémium



```
delete(idsocio){
  HTTP.call('DELETE', 'http://localhost:8080/Cineldo-1.0-SNAPSHOT/webresources/com.mycompany.cineldo.socio/' + idsocio
  , (error, result) => {
    if (!error) {
      //JSON.stringify(result);
      this.fetch();
    }
  });
}
```