BITACORA MANTENIMIENTO DE SALONES DE EVENTOS

UNIVERSIDAD POLITÉCNICA SALESIANA

INTEGRANTES:

ROMMEL INGA

EDWIN QUISHPE

RICARDO POZO

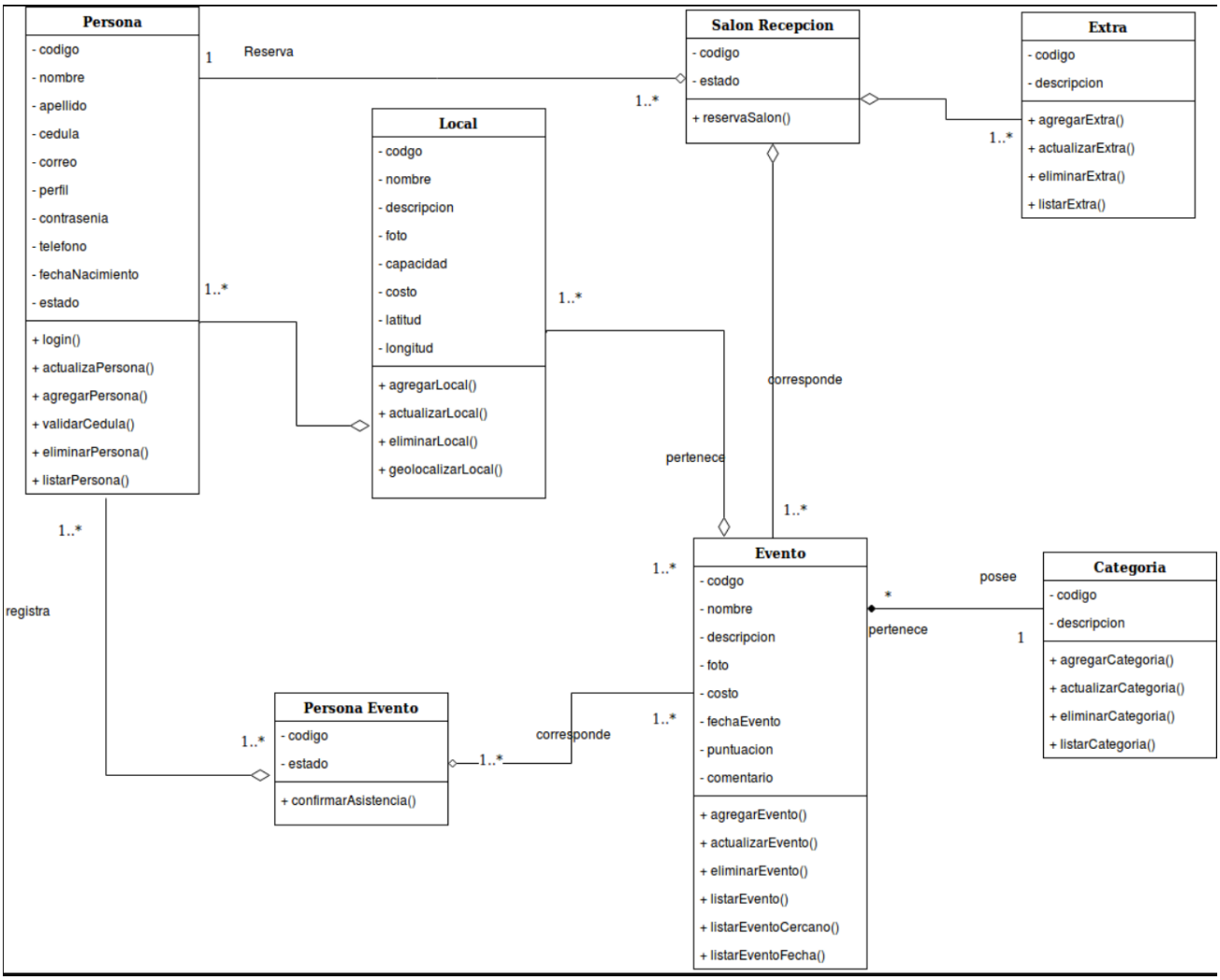
BYRON CALLE

5 DE DICIEMBRE DE 2017

APLICACIONES DISTRIBUIDAS

ING. CRISTIAN TIMBI

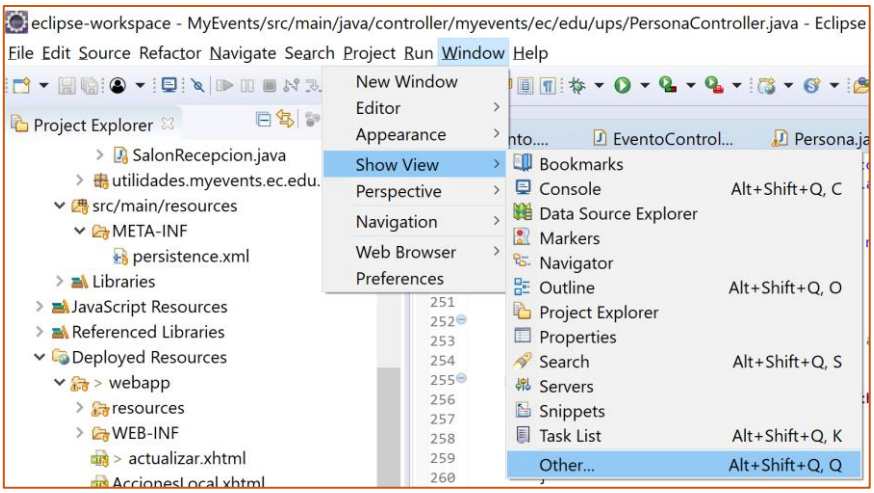
1. Diagrama de clases MyEvents



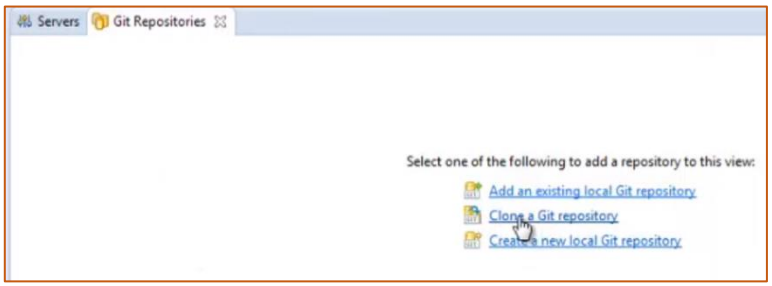
2. MANEJO DEL REPOSITORIO GITHUB

Para sincronizar un proyecto desde github con nuestro eclipse oxigen, vamos a realizar los siguientes pasos.

Primero debemos hacer clic en la pestaña de Windows --> luego nos dirigimos a la opción de dice Show View -> por ultimo aquí se despliega otro menú y seleccionamos en other.



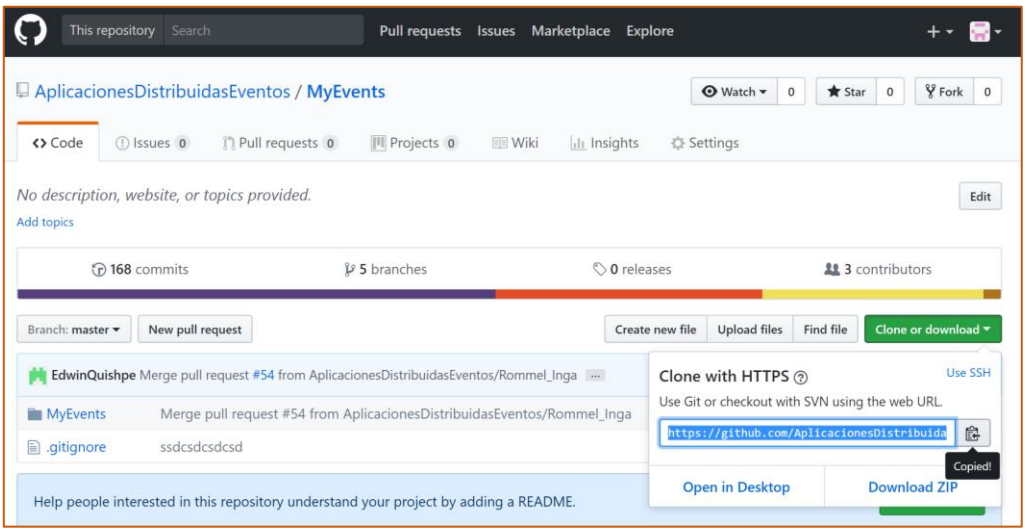
Ya aquí vamos a seleccionar los repositorios de Github, para que este pueda enlazar a un nuevo proyecto que tengamos creado en nuestro Github, por defecto primero iremos a la opción Choose a Git reposity.



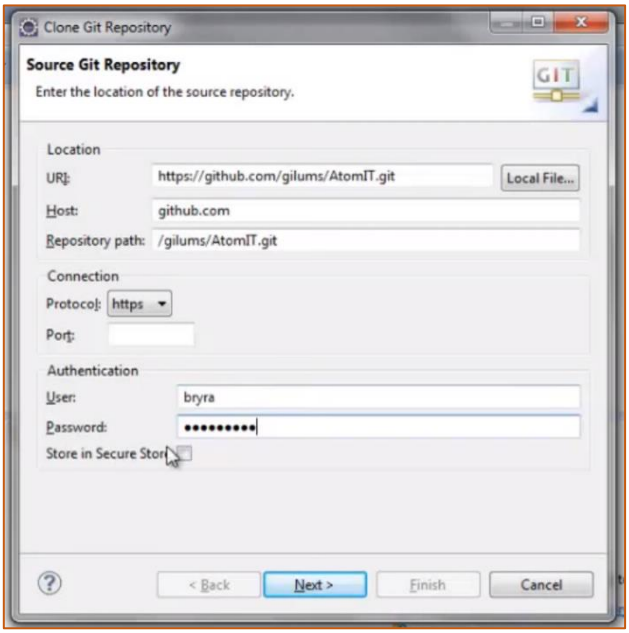
Ya una vez dentro se nos despliega otra ventana en la cual tendremos que pegar el directorio de donde se encuentra ubicado nuestro proyecto.

Pero para ello debemos realizar lo siguiente:

Nos dirigimos a nuestra página principal de Github y escogemos el proyecto a ejecutarse ese momento -> ya identificado el proyecto nos dirigimos a la sección que dice Clone or Download y copiamos el link de ese proyecto

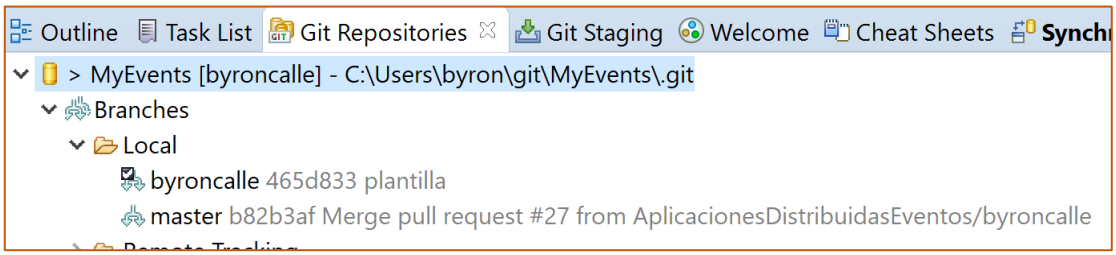


Este link que acabamos de copiar lo vamos a pegar en la ventana siguiente

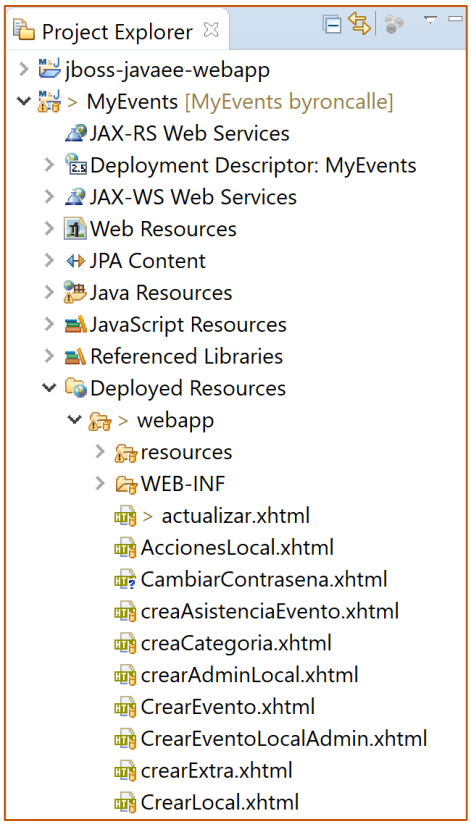


Ya una vez hecho esto ya solo nos queda en dar en siguiente y finalizar nuestra sincronización con el eclipse.

Ahora como siguiente paso debemos desplegar las características que vienen y procedemos a crear una rama para poder editar nuestro proyecto desde las ramas hijos y no desde el usuario root, así se tendrá un mejor control de quienes conformen el grupo de trabajo, con sus roles bien definidos a modificar sin alterar los cambios de los demás integrantes.



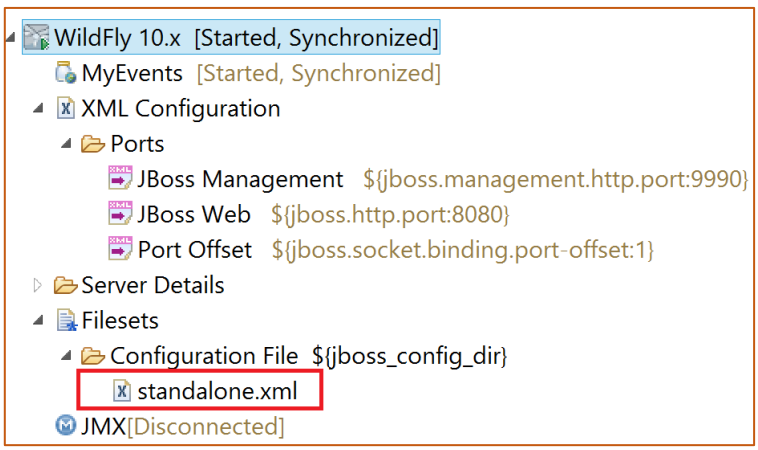
Ya hecho esto, si lo hacemos por primera vez ya estaría cargado en nuestro eclipse el proyecto que vamos a modificar, caso contrario tendremos que sincronizarlo, manualmente.



3. CONFIGURACION DRIVERS EN EL STANDALONE PARA LA CONEXION JAVA EE.

Los drivers para la conexión con los diferentes tipos de bases de datos se debe de configurar un archivo de del servidor Wildfly denominado standalone.

La ubicación del archivo standalone está en:



La configuración de los drivers con el nombre de la base de datos, usuario y contraseña se encuentra en:

```
<datasource jta="true" jndi-name="java:jboss/datasources/mysql" pool-name="mysql" enabled="true" use-ccm="true">
    <connection-url>jdbc:mysql://localhost:3306/bdevento</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <driver>mysql-connector-java-5.1.44.jar_com.mysql.jdbc.Driver_5_1</driver>
    <security>
        <user-name>root</user-name>
    </security>
    <validation>
        <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
        <background-validation>true</background-validation>
        <exception-sorter class-name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
</datasource>
<drivers>
    <driver name="oracle-driver" module="com.oracle">
        <driver-class>oracle.jdbc.OracleDriver</driver-class>
    </driver>
    <driver name="postgresql-driver" module="org.postgresql">
        <driver-class>org.postgresql.Driver</driver-class>
    </driver>
    <driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
    </driver>
</drivers>
</datasources>
```

4. CREACION DE ENTIDADES

Las entidades de datos son modelos de datos que se encuentra definidas en la base de datos y que representa una tabla en esa base.

Para la creación del proyecto **MyEVENT**, se crearan las entidades de negocios en eclipse Oxigen, mediante el cual se hará uso del API ***JPA*** para establecer las tablas en la base de datos MySQL.

Para la creación de ello nos basaremos en un modelo E-R y Diagrama de clases que contiene información sobre la estructura del sistema.

Una vez ya alcanzado los puntos anteriores, procederemos a realizar la inserción de código para la creación de las entidades, por lo tanto tenemos:

Entidad: Persona

Package: modelo.myevents.ec.edu.ups

```
/*ENTIDAD PERSONA
 * */
@Entity
@Table(name = "PERSONA")
public class Persona {
    @Id
    @Column(name = "per_id")
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int id;

    @Column(name = "per_nombre")
    @NotBlank(message = "Por favor ingrese el nombre")
    private String nombre;

    @Column(name = "per_apellido")
    @NotBlank(message = "Por favor ingrese el apellido")
    private String apellido;

    @Column(name = "per_cedula")
    @Pattern(regexp = "[\\s]*[0-9]*[1-9]+",message="Solo debe ingresar numeros")
    @NotBlank(message = "Por favor ingrese la cedula")
    private String cedula;
```

```
/*
 * Respectivos getters y setters
 */
```

La clase Persona para que se una entidad se le agrega la anotación **@Entity** y un **@Id** la cual serán almacenadas en la base de datos. Las demás anotaciones son validaciones y nombres que se les coloca a las propiedades de esa entidad. Además la entidad deberá contener los métodos getter y setter ya que por tener propiedades privadas son la vía de acceso a dichas propiedades.

Entidad: Local

Package: modelo.myevents.ec.edu.ups

```
/*ENTIDAD LOCAL SALON DE EVENTOS
 * */
@Entity
@Table(name="LOCAL")
public class Local {

    @Id
    @Column(name="local_codigo")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int codigo;

    @Column(name="local_nombre")
    @NotNull
    @Size(min=5,max=50)
    private String nombre;

    @Column(name="local_descripcion")
    @NotNull
    @Size(min=5,max=50)
    private String descripcion;

    @Column(name="local_capacidad")
    @NotNull
    private String capacidad;
```

```
/*
 * Respectivos getters y setters
 */
```

De la misma manera tenemos a la entidad Local que contendrá toda la información necesaria para completar la tabla dentro de la base de datos.

Ésta es la forma de crear entidades de datos en java EE, de igual manera se crearán las demás tablas basándose en el diagrama de clases siguiendo la misma lógica aplicada anteriormente.

4.1.Relaciones entre Entidades

Como se puede observar en el diagrama de clases, la entidad Persona puede realizar la reserva de uno o más locales y cada local puede ser reservado por una sola persona. Explicado lo anterior se procederá a realizar la relación de Maestro-detalle en Java EE entre estas dos entidades.

Para realizar la relación de uno a muchos se hará uso de la anotación **@OneToMany**, la cual la entidad Persona tendrá una colecciones de locales, por ende se le agrega un objeto de tipo List< > de Local llamado locales.

Tenemos en la entidad Persona:

```
@OneToMany(cascade=(javax.persistence.CascadeType.ALL),fetch=FetchType.EAGER)
@JoinColumn(name="per_loc_fk", referencedColumnName="per_id")
private List<Local> locales;
```

CasccadeType.**ALL** ésta anotación se agregan los tipos de cascada, para incluir solo actualizaciones y fusiones en la operación cascada para una relación de uno a muchos.

FetchType.**EAGER,** éste comando fuerza la carga de los datos al momento de iniciar.

@JoinColumn es una anotación utilizada para referenciarse a la columna de la otra entidad la cual tendrá la FK, el primer campo dentro del *JoinColumn* será el nombre que se le dará a esa columna en este caso a la entidad Local, el campo *referencedColumnName* es el código o id de la entidad Persona.

5. CREACION DE **DAO** (OBJETO DE ACCESO A DATOS)

Los objetos de acceso a datos DAO son los únicos encargados de conectarse con los orígenes de datos y recuperar estructura de datos, son los encargados de enviar sentencias SQL. Para que los DAO sea un objeto persistente es necesario realizar la llamada a los EntityManager que se utiliza para crear, eliminar instancias de entidad, para encontrar entidades por su ID, y para realizar consultas sobre esas entidades.

En nuestro proyecto crearemos una clase Recursos el cual contendrá el API de persistencia y será inyectado a través de la anotación **@Inyect.**

Clase: EntityManager

Package: utilidades.myevents.ec.edu.ups

```
public class Resources {

   @Produces
   @PersistenceContext
   private EntityManager em;

   @Produces
   public Logger produceLog(InjectionPoint injectionPoint) {
      return Logger.getLogger(injectionPoint.getMember().getDeclaringClass().getName());
   }

   @Produces
   @RequestScoped
   public FacesContext produceFacesContext() {
      return FacesContext.getCurrentInstance();
   }

}
```

Se crea un objeto **em**, el cual será utilizado por los objetos de acceso a datos.

Clase: PersonaDAO

Package: dao.myevents.ec.edu.ups

```java
@Stateless
public class PersonaDAO {

    @Inject
    private EntityManager em;
```

@Stateles es un bean de sesion con estado, utilizado para mantener una conversación es decir el bean mantendrá los datos del usuario hasta que cierre sesión.

@Inject es una inyección de dependencias que permite suministrar objetos a una clase.

En esta clase inyectamos la persistencia **em** proveniente del **EntityManager**.

```java
/*
 *Metodo para Insertar
 */
public void insertPersona(Persona p) {
    em.persist(p);
}

/*
 *Metodo para Actualizar
 */
public void updatePersona(Persona p) {
    System.out.println("Updating..."+p.getId()+p.getCorreo()+p.getContrasenia());
    em.merge(p);
}
/*
 *Metodo para buscar
 */
public Persona selectPersona(int id) {
    Persona p = em.find(Persona.class, id);
    return p;
}

/*
 *Metodo para eliminar
 */
public void deletePersona(int id) {
    Persona p = selectPersona(id);
    em.remove(p);
}

/*
 *SQl consulta a la base
 */
public List<Persona> listPersonas() {
    String sql = "select p from Persona p";
    TypedQuery<Persona> query = em.createQuery(sql, Persona.class);
    System.out.println("2");
    List<Persona> lpersonas = query.getResultList();

    return lpersonas;
}
```

La imagen anterior muestra la forma de realizar un CRUD a una entidad, se aplica la misma lógica para la creación de los demás CRUD en las diferentes entidades y poder persistir en la base de datos.

Clase: LocalDAO

Package: dao.myevents.ec.edu.ups

Para la entidad Local.java se mantiene con la misma estructura en el DAO que Persona.java. Ahora para poder realizar una edición de un local y persistir la misma entidad se lo realiza a través de un método que utiliza uno de los métodos del em para buscar por el id, si el objeto

de Local.java es diferente de null encuentra el código y actualiza esa tabla, de lo contrario realiza una nueva inserción.

Método para editar y guarda en LocalDAO.java

```
@Stateless
public class LocalDAO {

    @Inject
    private EntityManager em;


    //Metodo para editar y guardar

    public void guardarLocal(Local l) {

        Local auxlocal = leerLocal(l.getCodigo());
        if(auxlocal!=null) {
            updateLocal(l);

        }else {
            insertarLocal(l);
        }
    }
```

6. CREACIÓN DE ACTION CONTROLERS

Los ActionController de cada entidad se encuentra ubicada en la lógica de negocios necesaria para interactuar con el cliente de la aplicación, realizar validaciones, estos son encargados de utilizar los métodos CRUD y simplificar la complejidad en la programación. Además de ser utilizado para la navegación en páginas JSF.

Para la entidad Personas tenemos:

Clase: PersonaController

Package: controller.myevents.ec.edu.ups

```java
@ManagedBean
@SessionScoped
public class PersonaController {

    /*
     * Variable para la validacion de la cedula
     */
    private static final String PATTERN_EMAIL = "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@"
            + "[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";

    private Persona personas = null;

    private int id;
    private String pactual;

    /*Definicion de variables para la validacion-coincidencia del numero de cedula ingresado
     * */
    @NotBlank(message = "Ingrese las contrasenias")
    private String contrasenia;
    private String conincidencia;
    private String Loginexiste;

    /*Varibles donde almaceno los valores de la consulta maestro-detalles
     * */
    private String nusuario;
    private String nlocal;
    private String ndescripcion;
    private String ncapacidad;
    private String ncosto;
```

@ManagedBean es un bean de Java Beans administrado por el Framewrok JSF, estos contienen los getter y setter, lógica comercialo respaldos.

@SessionScoped es uno de los ámbitos del ManagedBean que vive mientras viva la sesión HTTP.

```java
public void crear() {
    if (coincidirContrasenia() == true) {
        if (validarCedula() == true) {
            if(validarCorreo()==true) {
                personas.setPerfil("USUARIO");
                personas.setEstado("A");
                pdao.guardar(personas);
                inicializar();
                init();
                this.conincidencia = "Grabado exitoso!";
            }else {
                this.conincidencia = "El formato del correo es incorrecto";
            }
        } else {
            System.out.println("Cedula incorrecta");
            this.conincidencia = "La cedula es incorrecta";
        }
    } else {
        this.conincidencia = "Ingrese las mismas contrasenias";
    }
}
```

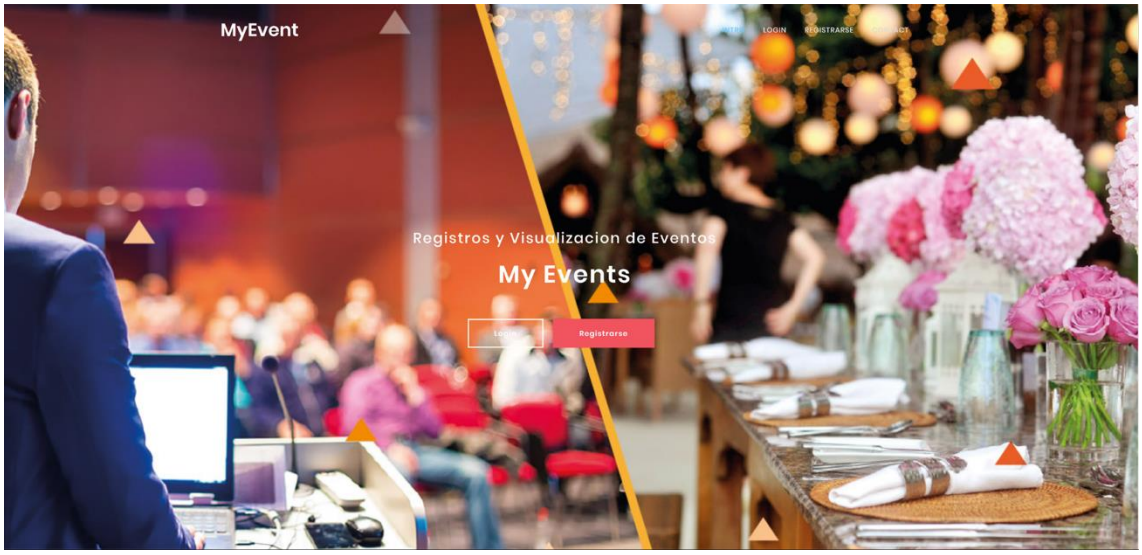
También se varios objetos necesarios para la clase PersonaController, una de ellas es la de crear la persona, además se debe validar todos los campos que fueran necesarios para evitar conflictos con los usuarios que están familiarizándose con la app.

El administrador de la app debe tener la opción de realizar un crud en las clases que sean necesarios de su modificación, para ello mostramos parte del código, en el cual se muestra como creando el objeto para modificación.

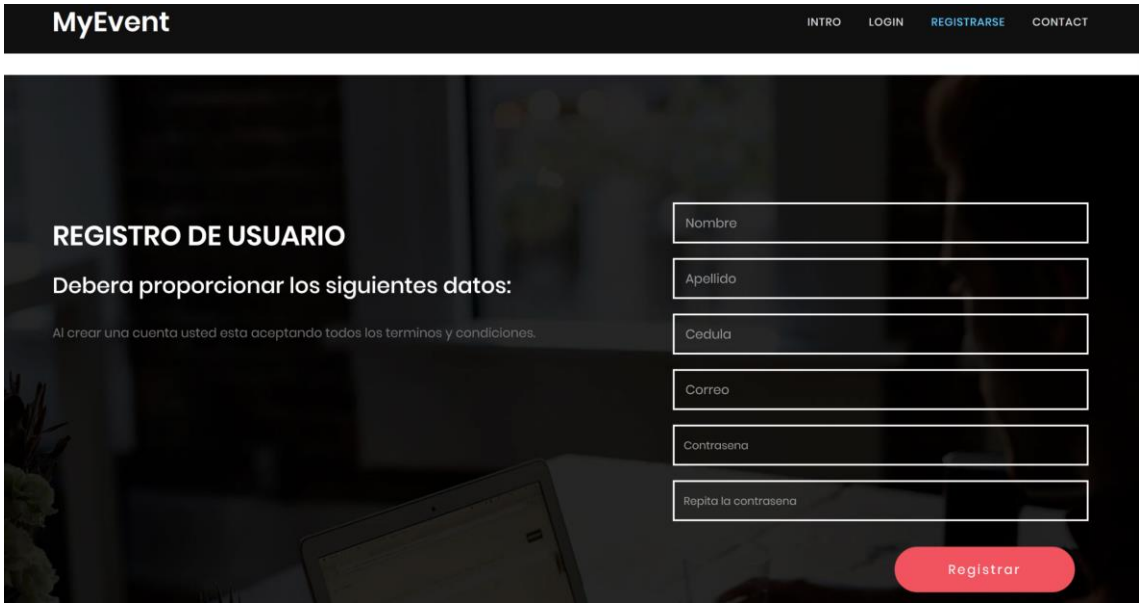
```
public String modificar() {
    try{
        System.out.println(personas.getPerfil());
        if(myUser.getPerfil().equals("USUARIO")) {
            personas.setContrasenia(pactual);
            pdao.updatePersona(personas);
            return "mainUser";
        }else if(myUser.getPerfil().equals("ADMIN")) {
            personas.setContrasenia(pactual);
            System.out.println("ACTUALIZAR ADMIN :"+personas.getCedula());
            System.out.println("ELSE IF ADMIN");
            pdao.updatePersona(personas);
            return "mainAdmin";
}return null;
    }catch (NullPointerException e) {
    // TODO: handle exception
        System.out.print("NullPointerException caught");
        return null;
    }
    finally {
            System.out.println("error" + pactual);
    }
}
```

NOTA: se debe controlar algunas excepciones probables a darse en la ejecución de la app.

7. DISEÑO DE LA INTERFAZ (MANEJO DE JSF)

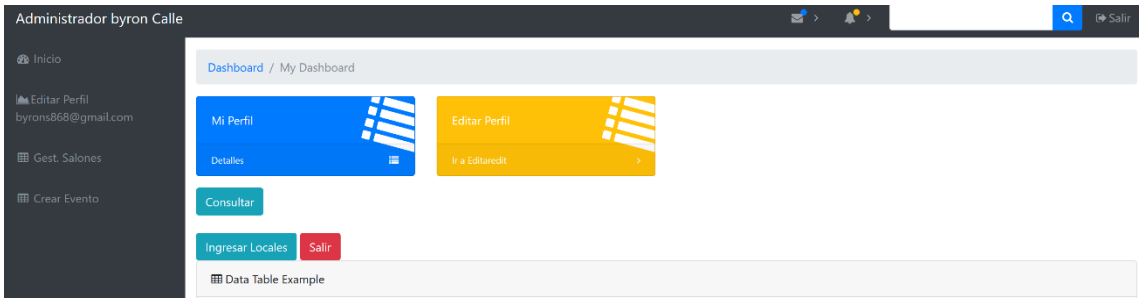


Como se puede observar esta es la interfaz principal de nuestra aplicación, la que los usuarios podrán acceder a la misma luego de un registro y luego deben loguearse a la misma.



Interfaz de la aplicación por la cual un usuario se podrá registrar a la aplicación MyEvent. Como se puede ver se tienen los campos necesarios para identificar la autenticidad del usuario nuevo.

Ya una vez logueado a la aplicación, el administrador del nuevo local a crearse podrá ver la siguiente interfaz.



Como se puede observar la platilla se trató de hacerla lo más amigable al usuario, el cual tiene las opciones de:

- Editar el perfil del usuario (nombre, correo, etc).
- Gestionar los salones (Descripción, nombre local, Capacidad, puntuación, ubicación, etc)
- Lita los eventos cargados en la aplicación.

Aquí el administrador del local tiene la funcionalidad de CRUD.

BITACORA EJECCUCIÓN PROYECTO MYEVENT																			
ACTIVIDADES	SESIONES																		OBSERVACIONES
	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	
Recopilación de información																			Se buscó información relevante para resolver problema y tener una idea de negocio aceptable
análisis del problema																			Relacionar todos los eventos en una sola aplicación, cruzar información de los locales, ubicar los locales por medio de google Maps (GPS).
Desarrollo																			Se comenzó a desarrollar la app con arquitectura Java EE, para optimizar el desarrollo de la misma (Capa de Presentación, capa de Negocio, Capa de Servidor) Iniciamos con la ejecución de las tablas en nuestra bd, luego se iniciamos con el desarrollo de las plantillas, después se comenzó a ejecutar las tablas de los usuarios luego se procedió con procesos de llenado de las tablas con datos, a su vez se validó los datos, luego se partió con la cruce de la información.
Pruebas																			Probar que los datos se creen en la bd, luego ingresar datos y que se relacionen entre ellos(FK), Validar
Implementación																			Ejecución de mantenimiento de las tablas para el administrador.