# Universidad Politécnica de Victoria

# Sistema del Control de Sesiones de CAI

**(Versión: 1.0.0)**

# Manual Técnico

M.S.I. MARIO HUMBERTO RODRÍGUEZ CHÁVEZ

JOSE ANTONIO MOLINA DE LA FUENTE
SERGIO GIOVANNY PÉREZ PICAZO
ERICK ELIZONDO RODRIGUEZ

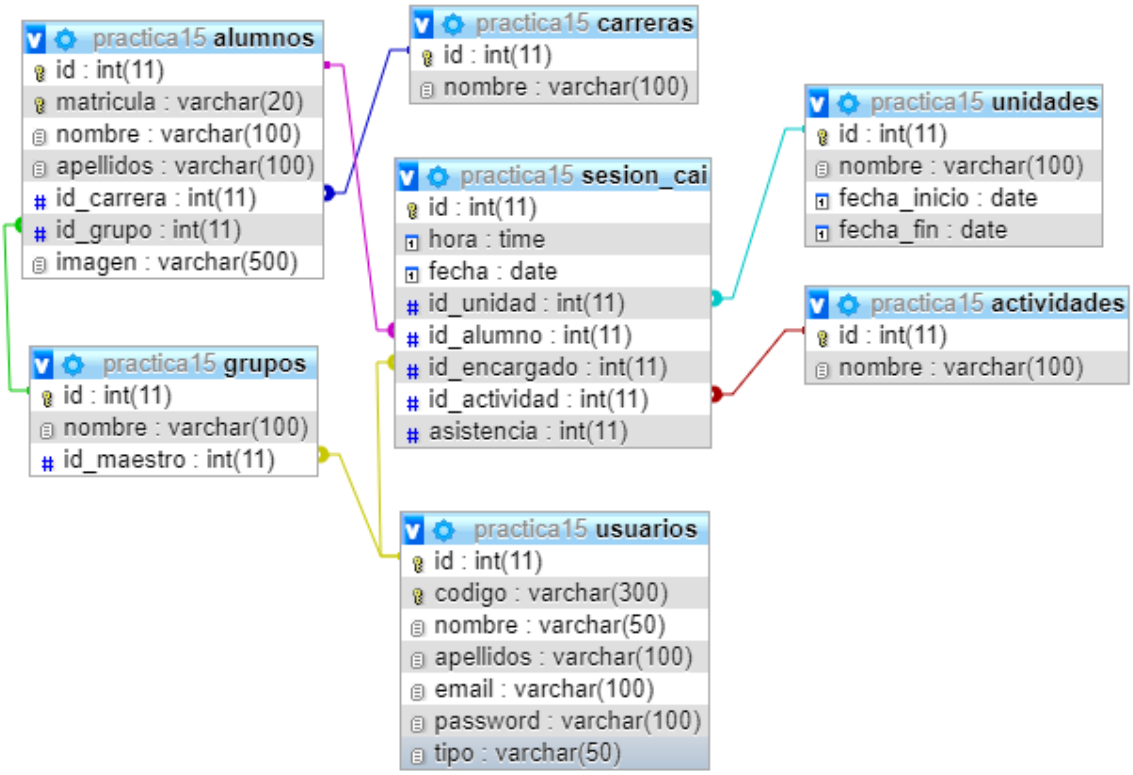
12 de agosto de 2018

# Contenido

# Introducción

El sistema de control de sesiones de CAI, permite llevar una mejor administración de las horas realizadas por los alumnos durante el cuatrimestre en curso. Además, el sistema ofrece diferentes módulos dependiendo del tipo de empleado que quiera acceder al sistema. Dentro de los usuarios que pueden acceder se encuentran, el superadmin, el cual tiene acceso a cada uno de los módulos y realizar todo tipo de tareas, el encargado, el cual permite agregar alumnos a la sesión de CAI y revisar las horas de los alumnos, y por último se encuentran los maestros, que solamente pueden acceder a las horas realizadas por los alumnos de sus grupos.

# Diseño de la Base de Datos



# Modelo

La carpeta del modelo contiene:

- Dos carpetas
- Cinco archivos php.

### Carpeta images

La carpeta images contiene el icono del logo de la universidad.

### Carpeta uploads

La carpeta uploads contendrá todas las imágenes que se suban al sistema.

### Archivo conexion.php

El archivo conexion.php realiza la conexión mediante PDO a la base de datos.

```
1. <?php
2. //Clase de conexión con la BD
3. class Conexion{
4.         //función que se conecta mediante PDO a la base de datos
   indicada en el parámetro
5.         public function conectar(){
6.                 $link = new PDO("mysql:host=localhost;dbname=practica15
   ","root","");
7.                 return $link;
8.         }
9. }
10.    ?><?php
11.    //Clase de conexión con la BD
12.    class Conexion{
13.            //función que se conecta mediante PDO a la base de datos
   indicada en el parámetro
14.            public function conectar(){
15.                    $link = new PDO("mysql:host=localhost;dbname=practi
   ca15","root","");
16.                    return $link;
17.            }
18.    }
19.    ?>
```

**Archivo borrar.php**

El archivo borrar.php requiere de los archivos crud.php y controller.php, después hace una instancia del controlador y llama la función borrarController,

```
1. <?php
2.          //se incluyen los archivos de modelo y controlador
3.          require_once("model/crud.php");
4.          require_once("controller/controller.php");
5.
6.          //se crea la instancia del controlador
7.          $controller = new MVC();
8.
9.          //se ejecuta el método borrar de la clase del controlador
10.             $controller->borrarController($_GET['id'],$_GET['tipo']);
11.    ?>
```

**Archivo logout.php**

El archivo logout.php destruye la sesión actual y redirecciona al index.

```
1. <?php
2.          //se destruye la sesión actual
3.          session_destroy();
4.          //se redirecciona al index
5.          echo "<script>window.location='index.php';</script>";
6.
7. ?>
```

**Archivo enlaces.php**

El archivo enlaces.php contiene la clase Enlaces() que contiene el método publico enlacesPaginasModel que dado un nombre de enlace redirige hacia el modulo que quiere ser mostrado.

```
1. <?php
2. //Clase de enlaces de pagina
3. class Enlaces{
4.          //metodo publico que dado un nombre de enlace, retorna el
   modulo que sera incluido o mostrado
5.          public function enlacesPaginasModel($enlace){
6.
7.                  if($enlace== "usuarios" || $enlace == "grupos" || $enl
   ace == "alumnos" || $enlace == "carreras" || $enlace =="sesion_cai"||
   $enlace == "actividades" || $enlace == "unidades"){
8.                          $module = "view/$enlace/$enlace.php";
```

```
9.                  }else if($enlace == "editar_usuario" || $enlace == "re
   gistro_usuario"){
10.                             $module = "view/usuarios/$enlace.php";
11.                     }else if($enlace== "editar_grupo" || $enlace == "re
   gistro_grupo"){
12.                             $module = "view/grupos/$enlace.php";
13.                     }else if($enlace== "editar_alumno" || $enlace == "r
   egistro_alumno"){
14.                             $module = "view/alumnos/$enlace.php";
15.                     }else if($enlace== "editar_carrera" || $enlace == "
   registro_carrera"){
16.                             $module = "view/carreras/$enlace.php";
17.                     }else if($enlace == "editar_actividad" || $enlace =
   = "registro actividad"){
18.                             $module = "view/actividades/$enlace.php";
19.                     }else if($enlace == "editar_unidad" || $enlace == "
   registro_unidad"){
20.                             $module = "view/unidades/$enlace.php";
21.                     }else if($enlace == "login"){
22.                             $module = "view/login.php";
23.                     }else if($enlace == "borrar"){
24.                             $module = "model/borrar.php";
25.                     }else if($enlace == "logout"){
26.                             $module = "model/logout.php";
27.                     }else if($enlace == "reportes" || $enlace == "detal
   le_reporte"){
28.                             $module = "view/reportes/$enlace.php";
29.                     }
30.                     else{
31.                             $module = "view/sesion_cai/sesion_cai.php";
32.                     }
33.                     return $module;
34.             }
35.     }
36.     ?>
```

**Archivo crud.php**

El archivo crud.php requiere del archivo de conexion.php, además se encuentra la clase modelo llamada Crud que hereda las propiedades y métodos de la clase Conexión. Dentro de la clase Crud, se encuentran los diferentes métodos que se usaron en el sistema, a continuación, se explicaran:

**Método ingresoUsuarioModel**

Utiliza como parametros el nombre de usuario y la contraseña, de esta manera se realiza una consulta a la base de datos para verificar que el usuario existe y que pueda acceder al sistema.

```
1. public function ingresoUsuarioModel($user, $password){
2.                 $stmt = Conexion::conectar()->prepare("SELECT * FROM
   usuarios WHERE codigo = :user and password = :password"); //se prepara
   la conexion
3.                 //definicion de parametros
4.                 $stmt->bindParam(":user", $user);
5.                 $stmt->bindParam(":password",$password);
6.                 $stmt->execute(); //ejecucion mediante pdo
7.                 return $stmt->fetch(); //se retorna lo asociado a la
   consulta
8.                 $stmt->close();
9.         }
```

**Método contarAlumnosController**

Esta consulta retorna el total de alumnos que están en la sesión actual de CAI y que no tengan asistencia. Retorna lo asociado a la consulta.

```
1.         public function contarAlumnosController(){
2.                 $stmt = Conexion::conectar()->prepare("SELECT COUNT(*)
   FROM sesion_cai WHERE fecha = CURDATE() AND hora >= DATE_FORMAT(NOW(),
   '%k:00:00') AND hora <= DATE_FORMAT(NOW(), '%k:59:59') AND
   asistencia=0"); //se prepara la conexion
3.                 //definicion de parametros
4.                 $stmt->execute(); //ejecucion mediante pdo
5.                 return $stmt->fetch()[0]; //se retorna lo asociado a la
   consulta
6.                 $stmt->close();
7.         }
```

**Método deleteSesionesSinAsistenciaModel**

Esta consulta elimina la asistencia de los alumnos en las sesiones, es decir que, si un alumno no recibió asistencia durante el tiempo indicado, el sistema lo borrara en automático. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.         public function deleteSesionesSinAsistenciaModel(){
2.                 $stmt = Conexion::conectar()->prepare("DELETE FROM
   sesion_cai WHERE asistencia=0 AND ( ( hora < DATE_FORMAT(NOW(),
   '%k:00:00') OR hora > DATE_FORMAT(NOW(), '%k:59:59') ) OR
   fecha!=CURDATE())"); //se prepara la conexion
3.                 //definicion de parametros
```

```
4.                  if($stmt->execute())
5.                          return "success";
6.                  else
7.                          return "error";
8.                  $stmt->close();
9.          }
```

**Método vistaXTablaModel**

Este método recibe como parámetro el nombre de la tabla, por lo tanto, el método retornara todos los registros de la tabla indicada.

```
1.          public function vistaXTablaModel($table){
2.                  $stmt = Conexion::conectar()->prepare("SELECT *
   FROM $table"); //preparacion de la consulta SQL
3.                  $stmt->execute(); //ejecucion de la consulta
4.                  return $stmt->fetchAll(); //se retorna en un array
   asociativo el resultado de la consulta
5.                  $stmt->close();
6.          }
```

**Método getSesionesControllerModel**

Este método permite obtener todas las sesiones de la hora y la fecha ingresada. El método retorna como resultado de la consulta un array asociativo.

```
1.          public function getSesionesControllerModel(){
2.                  $stmt = Conexion::conectar()->prepare("SELECT a.id as
   id_alumno, a.imagen as imagen, sc.id as id, a.matricula as matricula,
   CONCAT(a.nombre,' ',a.apellidos) as nombre, ac.nombre as actividad,
   sc.hora as hora, sc.asistencia as asistencia FROM sesion_cai AS sc
   INNER JOIN alumnos AS a ON a.id=sc.id_alumno INNER JOIN actividades AS
   ac ON ac.id = sc.id_actividad WHERE fecha = CURDATE() AND hora >=
   DATE_FORMAT(NOW(), '%k:00:00') AND hora <= DATE_FORMAT(NOW(),
   '%k:59:59') AND asistencia=0"); //preparacion de la consulta SQL
3.                  $stmt->execute(); //ejecucion de la consulta
4.                  return $stmt->fetchAll(); //se retorna en un array
   asociativo el resultado de la consulta
5.                  $stmt->close();
6.          }
```

**Método setAsistenciaModel**

Este método permite actualizar la asistencia del alumno. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function setAsistenciaModel($id){
2.                  $stmt = Conexion::conectar()->prepare("UPDATE
   sesion_cai SET asistencia = 1 WHERE id = :id");
3.                  $stmt->bindParam(":id", $id);
4.                  if($stmt->execute())
5.                          return "success";
6.                  else
7.                          return "error";
8.                  $stmt->close();
9.          }
```

**Método registroUsuarioModel**

Este método permite el registro de maestros o encargados a través de un arreglo asociativo de datos que se inserta en la base de datos. Si la inserción es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function registroUsuarioModel($data){
2.                  $stmt = Conexion::conectar()->prepare("INSERT INTO
   usuarios(codigo,nombre,apellidos, email, password, tipo)
   VALUES(:codigo, :nombre, :apellidos, :email, :password, :tipo)");
3.                  //preparacion de parametros
4.                  $stmt->bindParam(":codigo", $data['codigo']);
5.                  $stmt->bindParam(":nombre", $data['nombre']);
6.                  $stmt->bindParam(":apellidos", $data['apellidos']);
7.                  $stmt->bindParam(":email", $data['email']);
8.                  $stmt->bindParam(":password", $data['password']);
9.                  $stmt->bindParam(":tipo", $data['tipo']);
10.                     if($stmt->execute()) //ejecucion
11.                             return "success"; //respuesta
12.                     else
13.                             return "error";
14.                     $stmt->close();
15.             }
```

**Método registroAlumnoModel**

Este método permite el registro de alumnos a través de un arreglo asociativo de datos que se inserta en la base de datos. Si el registro se guardó correctamente, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function registroAlumnoModel($data){
2.                  $stmt = Conexion::conectar()->prepare("INSERT INTO
   alumnos(matricula,nombre,apellidos, id_carrera, id_grupo, imagen)
   VALUES(:matricula, :nombre, :apellidos, :carrera, :id_grupo,
   :imagen)");
3.                  //preparacion de parametros
4.                  $stmt->bindParam(":matricula", $data['matricula']);
5.                  $stmt->bindParam(":nombre", $data['nombre']);
6.                  $stmt->bindParam(":apellidos", $data['apellidos']);
7.                  $stmt->bindParam(":carrera", $data['carrera']);
8.                  $stmt->bindParam(":id_grupo", $data['id_grupo']);
9.                  $stmt->bindParam(":imagen", $data['imagen']);
10.                     if($stmt->execute()) //ejecucion
11.                             return "success"; //respuesta
12.                     else{
13.                             //echo print_r($stmt->errorInfo());
14.                             return "error";
15.                     }
16.                     $stmt->close();
17.             }
```

**Método registroGrupoModel**

Este método permite el registro de grupos a través de un arreglo asociativo de datos que se inserta en la base de datos. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function registroGrupoModel($data){
2.                  $stmt = Conexion::conectar()->prepare("INSERT INTO
   grupos(nombre,id_maestro) VALUES(:nombre, :id_maestro)");
3.                  //preparacion de parametros
4.                  $stmt->bindParam(":nombre", $data['nombre']);
5.                  $stmt-
   >bindParam(":id_maestro", $data['id_maestro']);
6.                  if($stmt->execute()) //ejecucion
7.                          return "success"; //respuesta
8.                  else
9.                          return "error";
10.                        $stmt->close();
11.             }
```

**Método registroActividadModel**

Este método permite el registro de actividades que el alumno puede llevar a cabo en CAI, a través de un arreglo asociativo de datos que se inserta en la base de datos. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function registroActividadModel($data){
2.                  $stmt = Conexion::conectar()->prepare("INSERT INTO
   actividades(nombre) VALUES(:nombre)");
3.                  //preparacion de parametros
4.                  $stmt->bindParam(":nombre", $data['nombre']);
5.                  if($stmt->execute()) //ejecucion
6.                          return "success"; //respuesta
7.                  else
8.                          return "error";
9.                  $stmt->close();
10.             }
```

**Método registroUnidadModel**

Permite el registro de unidades dentro del cuatrimestre, a través de un arreglo asociativo de datos que se inserta en la base de datos. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function registroUnidadModel($data){
2.                  $stmt = Conexion::conectar()->prepare("INSERT INTO
   unidades(nombre, fecha_inicio, fecha_fin) VALUES(:nombre,
   :fecha_inicio, :fecha_fin)");
3.                  //preparacion de parametros
4.                  $stmt->bindParam(":nombre", $data['nombre']);
5.                  $stmt-
   >bindParam(":fecha_inicio", $data['fecha_inicio']);
6.                  $stmt->bindParam(":fecha_fin", $data['fecha_fin']);
7.                  if($stmt->execute()) //ejecucion
8.                          return "success"; //respuesta
9.                  else
10.                                 return "error";
11.                         $stmt->close();
12.             }
```

**Método registroSesionCaiModel**

Este método permite el registro de la sesión de CAI, a través de un arreglo asociativo de datos que se inserta en la base de datos. Si la consulta es exitosa, entonces realiza una segunda consulta que regresa una respuesta, en caso contrario, retorna "error".

```
1.          public function registroSesionCaiModel($data){
2.                  $stmt = Conexion::conectar()->prepare("INSERT INTO
   sesion_cai(hora, fecha, id_unidad, id_alumno, id_encargado,
   id_actividad, asistencia) VALUES(DATE_FORMAT(NOW(), '%k:%i:%s'),
   CURDATE(), :id_unidad, :id_alumno, :id_encargado, :id_actividad,
   :asistencia)");
3.                  //preparacion de parametros
4.                  $stmt->bindParam(":id_unidad", $data['id_unidad']);
5.                  $stmt->bindParam(":id_alumno", $data['id_alumno']);
6.                  $stmt-
   >bindParam(":id_encargado", $data['id_encargado']);
7.                  $stmt-
   >bindParam(":id_actividad", $data['id_actividad']);
8.                  $stmt-
   >bindParam(":asistencia", $data['asistencia']);
9.                  if($stmt->execute()){ //ejecucion
10.                                 $stmt = Conexion::conectar()-
   >prepare("SELECT MAX(id) AS id, DATE_FORMAT(NOW(), '%k:%i:%s') AS
   hora FROM sesion_cai");
11.                                 $stmt->execute();
12.                                 return json_encode($stmt-
   >fetch()); //respuesta
13.                         }
14.                         else{
15.                                 return "error";
16.                         }
17.                         $stmt->close();
18.             }
```

**Método getServerHour**

Este método permite obtener la hora del servidor, para ser mostrada en la interfaz. Retorna el resultado de la consulta realizada.

```
1.          public function getServerHour(){
2.                  $stmt = Conexion::conectar()->prepare("SELECT
   DATE_FORMAT(NOW(), '%k:%i:%s') AS hora");
3.                  $stmt->execute();
```

```
4.                  return $stmt->fetch()[0]; //respuesta
5.          }
```

**Método getServerDate**

Obtener la fecha del servidor, para ser mostrada en la interfaz. Retorna el resultado de la consulta.

```
1.          public function getServerDate(){
2.                  $stmt = Conexion::conectar()->prepare("SELECT
   CURDATE() AS fecha");
3.                  $stmt->execute();
4.                  return $stmt->fetch()[0]; //respuesta
5.          }
```

**Método registroCarreraModel**

Este método permite el registro de carreras a través de un arreglo asociativo de datos que se inserta en la base de datos. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function registroCarreraModel($data){
2.                  $stmt = Conexion::conectar()->prepare("INSERT INTO
   carreras(nombre) VALUES(:nombre)");
3.                  //preparacion de parametros
4.                  $stmt->bindParam(":nombre", $data['nombre']);
5.                  if($stmt->execute()) //ejecucion
6.                          return "success"; //respuesta
7.                  else
8.                          return "error";
9.                  $stmt->close();
10.             }
```

**Método getRegModel**

Este método obtiene la informacion de un registro que coincida con el id que se manda como parámetro. Se retorna el resultado de la consulta.

```
1.          public function getRegModel($id, $table){
2.                  //se prepara la consulta sql
```

```
3.                  $stmt = Conexion::conectar()->prepare("SELECT *
   FROM $table WHERE id = :id");
4.                  $stmt->bindParam(":id",$id); //se asocia el
   parametro
5.                  $stmt->execute(); //se ejecuta la consulta
6.                  return $stmt->fetch(); //se retorna el resultado de
   la consulta
7.                  $stmt->close();
8.          }
```

**Método actualizarUsuarioModel**

Este método permite actualizar la información del usuario (maestros o encargados) que coincida con el id que se manda como parámetro. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function actualizarUsuarioModel($data){
2.                  $stmt = Conexion::conectar()->prepare("UPDATE
   usuarios SET codigo = :codigo, nombre=:nombre,
   apellidos=:apellidos, email=:email, password=:password, tipo=:tipo
   WHERE id = :id");
3.                  $stmt->bindParam(":codigo", $data['codigo']);
4.                  $stmt->bindParam(":nombre", $data['nombre']);
5.                  $stmt->bindParam(":apellidos", $data['apellidos']);
6.                  $stmt->bindParam(":email", $data['email']);
7.                  $stmt->bindParam(":password", $data['password']);
8.                  $stmt->bindParam(":tipo", $data['tipo']);
9.                  $stmt->bindParam(":id", $data['id']);
10.                     if($stmt->execute())
11.                             return "success";
12.                     else
13.                             return "error";
14.                     $stmt->close();
15.             }
```

**Método actualizarAlumnoModel**

Actualiza la información del alumno que coincida con el codigo del alumno. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function actualizarAlumnoModel($data){
```

```
2.                 $stmt = Conexion::conectar()->prepare("UPDATE
   alumnos SET matricula=:matricula, nombre=:nombre,
   apellidos=:apellidos, id_carrera=:id_carrera, id_grupo=:id_grupo
   WHERE id = :id");
3.                 $stmt->bindParam(":matricula", $data['matricula']);
4.                 $stmt->bindParam(":nombre", $data['nombre']);
5.                 $stmt->bindParam(":apellidos", $data['apellidos']);
6.                 $stmt-
   >bindParam(":id_carrera", $data['id_carrera']);
7.                 $stmt->bindParam(":id_grupo", $data['id_grupo']);
8.                 $stmt->bindParam(":id", $data['id']);
9.                 if($stmt->execute())
10.                             return "success";
11.                     else
12.                             return "error";
13.                     $stmt->close();
14.             }
```

**Método actualizarGrupoModel**

Este método permite actualizar la información del grupo que coincida con el codigo del grupo. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.         public function actualizarGrupoModel($data){
2.                 $stmt = Conexion::conectar()->prepare("UPDATE
   grupos SET nombre=:nombre, id_maestro = :id_maestro WHERE id=:id");
3.                 $stmt-
   >bindParam(":id_maestro", $data['id_maestro']);
4.                 $stmt->bindParam(":nombre", $data['nombre']);
5.                 $stmt->bindParam(":id", $data['id']);
6.                 if($stmt->execute())
7.                         return "success";
8.                 else
9.                         return "error";
10.                     $stmt->close();
11.             }
```

**Método actualizarActividadModel**

Permite actualizar la información de la actividad que coincida con el codigo de la actividad. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function actualizarActividadModel($data){
2.                  $stmt = Conexion::conectar()->prepare("UPDATE
   actividades SET nombre=:nombre WHERE id=:id");
3.                  $stmt->bindParam(":nombre", $data['nombre']);
4.                  $stmt->bindParam(":id", $data['id']);
5.                  if($stmt->execute())
6.                          return "success";
7.                  else
8.                          return "error";
9.                  $stmt->close();
10.             }
```

**Método actualizarUnidadModel**

El método permite actualizar la información de la unidad que coincida con el codigo de la unidad. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function actualizarUnidadModel($data){
2.                  $stmt = Conexion::conectar()->prepare("UPDATE
   unidades SET nombre=:nombre, fecha_inicio = :fecha_inicio,
   fecha_fin = :fecha_fin WHERE id=:id");
3.                  $stmt->bindParam(":nombre", $data['nombre']);
4.                  $stmt-
   >bindParam(":fecha_inicio", $data['fecha_inicio']);
5.                  $stmt->bindParam(":fecha_fin", $data['fecha_fin']);
6.                  $stmt->bindParam(":id", $data['id']);
7.                  if($stmt->execute())
8.                          return "success";
9.                  else
10.                             return "error";
11.                     $stmt->close();
12.             }
```

**Método actualizarCarreraModel**

Este método permite actualizar la información de la carrera que coincida con el codigo de la carrera. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function actualizarCarreraModel($data){
2.                  $stmt = Conexion::conectar()->prepare("UPDATE
   carreras SET nombre=:nombre WHERE id=:id");
```

```
3.                  $stmt->bindParam(":nombre", $data['nombre']);
4.                  $stmt->bindParam(":id", $data['id']);
5.                  if($stmt->execute())
6.                          return "success";
7.                  else
8.                          return "error";
9.                  $stmt->close();
10.             }
```

### Método borrarXModel

Este método permite borrar el registro dependiendo del id y el nombre de la tabla que se manden como parámetros. Si la consulta es exitosa, entonces retorna como respuesta “success”, en caso contrario, retorna “error”.

```
1.          public function borrarXModel($id, $table){
2.                  $stmt = Conexion::conectar()->prepare("DELETE
   FROM $table WHERE id = $id"); //preparar consulta para eliminar el
   registro con el id y la tabla mandada como parametro
3.                  if($stmt->execute()) //se ejecuta la consulta
4.                          return "success"; //se retorna la respuesta
5.                  else
6.                          return "error";
7.                  $stmt->close();
8.          }
```

### Método checkAvailabilityOfUserCodeModel

Este método permite verificar la disponibilidad de una columna en una tabla, para ello, recibe como parametros el codigo y nombre de la tabla, por lo tanto, retorna un valor de verdadero o falso, dependiendo del resultado de la consulta.

```
1.          public function checkAvailabilityOfUserCodeModel($code, $ta
   ble, $nid){
2.                  $stmt = Conexion::conectar()->prepare("SELECT
   EXISTS(SELECT * FROM $table WHERE $nid = :codigo) as a");
3.                  //echo $table . " " . $code . " ".$nid;
4.                  $stmt->bindParam(":codigo", $code);
5.                  $stmt->execute();
6.
7.                  return $stmt->fetch();
8.
9.                  $stmt->close();
10.             }
```

### Método getDetailsFromAlumnoModel

El método permite obtener informacion del alumno que coincida con el codigo que se manda como parámetro. Retorna el resultado de la consulta.

```
1.          public function getDetailsFromAlumnoModel($code, $table){
2.                  $stmt = Conexion::conectar()->prepare("SELECT id,
   CONCAT(nombre,' ',apellidos) as nombre, imagen FROM $table WHERE
   matricula=:matricula");
3.                  //echo $table . " " . $code . " ".$nid;
4.                  $stmt->bindParam(":matricula", $code);
5.                  $stmt->execute();
6.
7.                  return $stmt->fetch();
8.
9.                  $stmt->close();
10.             }
```

### Método getActividades

Obtiene todas las actividades que se tengan registradas en la base de datos.

```
1.          public function getActividades(){
2.                  $stmt = Conexion::conectar()->prepare("SELECT id,
   nombre FROM actividades");
3.                  $stmt->execute();
4.
5.                  return $stmt->fetch();
6.
7.                  $stmt->close();
8.          }
```

### Método getUnidadIdModel

Obtiene el id de la unidad que se encuentre entre un rango de fechas.

```
1.          public function getUnidadIdModel(){
2.                  $stmt = Conexion::conectar()->prepare("SELECT id
   FROM `unidades` WHERE fecha_fin>=NOW() AND fecha_inicio<=NOW()");
3.                  $stmt->execute();
4.
5.                  return $stmt->fetch();
6.
7.                  $stmt->close();
8.          }
```

**Método getReporteModel**

Este método permite que, dado un arreglo asociativo, se pueda obtener los alumnos de un grupo y filtrar el número de horas según la unidad y el id de grupo que se enviaron como parametros. El método retorna un array asociativo con el resultado de la consulta.

```
1.         public function getReporteModel($data){
2.                 //Consulta SQL para traer las horas de los alumnos
   del grupo seleccionado segun el id de la unidad
3.                 $stmt = Conexion::conectar()->prepare("SELECT a.id
   as id, a.matricula as matricula, a.nombre as alumnoN, a.apellidos
   as alumnoA, u.nombre as unidad ,SUM(sc.asistencia) as numero_horas,
   g.nombre as grupo
4.                         FROM grupos as g
5.                         INNER JOIN alumnos as a ON g.id =
   a.id_grupo
6.                         INNER JOIN sesion_cai as sc on a.id =
   sc.id_alumno
7.                         INNER JOIN unidades as u on u.id =
   sc.id_unidad
8.                         WHERE g.id = :id_grupo and u.id =
   :id_unidad and sc.asistencia = 1
9.                         GROUP BY sc.id_alumno");
10.                      //preparacion de parametros
11.                      $stmt-
   >bindParam(":id_grupo", $data['id_grupo']);
12.                      $stmt-
   >bindParam(":id_unidad", $data['id_unidad']);
13.                      $stmt->execute(); //ejecucion de la consulta
14.                      return $stmt->fetchAll(); //se retorna en un
   array asociativo el resultado de la consulta
15.                      $stmt->close();
16.             }
```

**Método getDetalleDeReporteModel**

Este método obtiene las actividades realizadas en cada hora en una unidad, mediante el id de alumno y el id de unidad que se mandaron como parametros. Retorna un array asociativo con el resultado de la consulta.

```
1.         public function getDetalleDeReporteModel($data){
2.                 //Consulta SQL para traer las horas de los alumnos
   del grupo seleccionado segun el id de la unidad
3.                 $stmt = Conexion::conectar()->prepare("SELECT
   al.matricula as matricula, al.nombre as nombre, al.apellidos as
   apellidos, g.nombre as nombreGrupo, u.nombre as
   nombreUnidad,  a.nombre as nombreActividad, sc.fecha as fecha,
   sc.hora as hora
4. FROM sesion_cai as sc INNER JOIN actividades as a ON
   sc.id_actividad = a.id
5. INNER JOIN unidades as u on u.id = sc.id_unidad
6. INNER JOIN alumnos as al on al.id = sc.id_alumno
7. INNER JOIN grupos as g on g.id = al.id_grupo
8. WHERE sc.id_alumno = :id_alumno and sc.id_unidad = :id_unidad and
   sc.asistencia = 1
9.
10.                                 ");
11.                     //preparacion de parametros
12.                     $stmt-
   >bindParam(":id_alumno", $data['id_alumno']);
13.                     $stmt-
   >bindParam(":id_unidad", $data['id_unidad']);
14.                     $stmt->execute(); //ejecucion de la consulta
15.                     return $stmt->fetchAll(); //se retorna en un
   array asociativo el resultado de la consulta
16.                     $stmt->close();
17.             }
```

**Método vistaGruposModel**

El método permite filtrar los grupos de un maestro mediante el id de maestro y se retorna como resultado un array asociativo con el resultado de la consulta.

```
1.         public function vistaGruposModel($id_usuario){
2.                 if(empty($id_usuario)){ //verificar si esta vacio
   el id de usuario (maestro), se obtienen todos los grupos
3.                         $stmt = Conexion::conectar()-
   >prepare("SELECT * FROM grupos"); //preparacion de la consulta SQL
4.                 }
```

```
5.                 else{ //si no esta vacio, se filtra por el id
   mandado como parametro
6.                         $stmt = Conexion::conectar()-
   >prepare("SELECT * FROM grupos WHERE id_maestro =
   :id"); //preparacion de la consulta SQL
7.                         $stmt->bindParam(":id", $id_usuario);
8.                 }
9.                 $stmt->execute(); //ejecucion de la consulta
10.                       return $stmt->fetchAll(); //se retorna en un
   array asociativo el resultado de la consulta
11.                       $stmt->close();
12.
13.             }
```

# Controlador

El archivo controller.php contiene la clase MVC que a su vez contiene diferentes funciones que son necesarias para el funcionamiento del sistema, a continuación, se explicaran cada uno de los métodos empleados:

**Método showTemplate**

El método showTemplate permite que se muestre la plantilla base cuando la sesión es iniciada, por lo tanto, dentro del método incluye el archivo template.php.

```
1.          public function showTemplate(){
2.                  session_start();
3.                  include "view/template.php";
4.          }
```

**Método enlacePaginasController**

Este método es el encargado de capturar la variable action mediante el método get y a su vez hace la petición al modelo para que redireccione a las vistas correspondientes.

```
1.          public function enlacePaginasController(){
2.                  if(isset($_GET['action'])){
3.                          $enlace = $_GET['action'];
4.                  }else{
5.                          $enlace = 'index';
6.                  }
7.
8.                  if(isset($_SESSION["user_info"])){
9.                          if( ($enlace!= "dashboard" || $enlace != "index
   " || $enlace != "sesion_cai" || $enlace !="reportes_sesiones") && $_SES
   SION["user_info"]["tipo"] == "encargado" && $enlace != "logout" && $enl
   ace != "reportes" && $enlace != "detalle_reporte"){
10.                                             $enlace = "index";
11.                         }
12.
13.                         if($enlace != "reportes_sesiones" && $_SESSI
   ON["user_info"]["tipo"] == "teacher" && $enlace != "logout" && $enlace
   != "reportes" && $enlace != "detalle_reporte"){
14.                                     $enlace = "reportes";
15.                         }
16.                 }
17.
18.                 //peticion al modelo
19.                 $peticion = Enlaces::enlacesPaginasModel($enlace);
20.           //mostrar peticion
21.                 include $peticion;
22.           }
```

**Método verificarLoginController**

Método de tipo público que verifica si un usuario ha iniciado sesión, en caso contrario, redirecciona al login.

```
1.          public function verificarLoginController(){
2.                  if(isset($_SESSION)){
3.                          if(isset($_SESSION['login'])){
4.                  if(!$_SESSION['login']){
5.                                  echo "<script>window.location='index.p
   hp?action=login';</script>";
6.                  }
7.        }else{
8.          echo "<script>window.location='index.php?action=login';</scrip
   t>";
9.        }
10.                   }else{
11.                             echo "<script>window.location='index.php?ac
   tion=login';</script>";
12.                   }
13.           }
```

**Método showNav**

Es un método público que trabaja con el archivo header.php, el cual verifica que el usuario este logueado, de ser asi muestra el menú de acuerdo a los privilegios que se tengan, por ejemplo, como se mencionó al principio del documento, existen tres tipos de personas que pueden acceder al sistema, pero cada una de ellas puede usar ciertos módulos.

**Método ingresoUsuarioController**

Es el método encargado de ingresar los valores del login e iniciar sesión.

```
1.          public function ingresoUsuarioController(){
2.                  if(isset($_POST['username']) && isset($_POST['passw
   ord']) && isset($_POST['btn_add'])){
3.
4.                          $resultado = Crud::ingresoUsuarioModel($_PO
   ST['username'], $_POST['password']); //se ejecuta la funcion del
   modelo
5.                          //se verifica que lo retornado por el
   modelo no este vacio
6.                          if(!empty($resultado)){
7.                                  $_SESSION['login']=true; //
   iniciar la variable de sesion login
```

```
8.                                          $_SESSION['user_info']= $re
   sultado; //guardar los datos del usuario en una sesion
9.                                          echo "<script>window.locati
   on='index.php?action=index'</script>";
10.                                 }else{
11.                                         //mostrar mensaje en caso de
   no existir el usuario
12.                                         echo "<script>swal('Usuario o
   contraseña incorrectos', 'No existe un usuario registrado con esas
   credenciales', 'error');</script>";
13.                                 }
14.                         }
15.                 }
```

**Método mostrarInicioController**

Imprime un mensaje en el inicio con el nombre del maestro dependiendo de la variable sesión.

```
1.          public function mostrarInicioController(){
2.                  if(isset($_SESSION['user_info'])){
3.                          echo "<i class='nav-icon fa fa-user'></i> [
   ".$_SESSION['user_info']['nombre']." ]";
4.                  }
5.          }
```

**Método getUsuariosController**

Método encargado de crear una tabla con los usuarios registrados en la base de datos, se le envía como parámetro el usuario actual.

```
1.          public function getUsuariosController($idUser){ //parametro
   $idUser (el usuario actual)
2.                  $informacion = Crud::vistaXTablaModel("usuarios");/
   /ejecucion del metodo del modelo
3.                  if(!empty($informacion)){
4.                          //si el resultado no esta vacio, imprimir
   los datos de los usuarios
5.          foreach ($informacion as $row => $item) {
6.            if($item['id']!=$idUser){ //no mostrar el usuario actual
7.            echo "<tr>";
8.            echo "<td>".$item['codigo']."</td>";
9.            echo "<td>".$item['nombre']."</td>";
10.                 echo "<td>".$item['apellidos']."</td>";
11.                 echo "<td>".$item['email']."</td>";
12.                 echo "<td>".strtoupper($item['tipo'])."</td>";
13.                 echo "<td>"."<a class='btn btn-secondary fa fa-
   edit'
```

```
   href='index.php?action=editar_usuario&id=".$item['id']."'></a></td>
   ";
14.                    //mandar por propiedad onclick el id del
   elemento tag a para eleminarlo
15.                                          echo "<td>"."<a class='btn
   btn-danger fa fa-trash' id='borrar_btn".$item["id"]."'
   onclick='b(".$item["id"].");'
   href='index.php?action=borrar&tipo=usuarios&id=".$item['id']."'></a
   ></td>";
16.
17.                   echo "</tr>";
18.                 }
19.
20.                                  }
21.           }
22.               }
```

**Método getAlumnosController**

Método encargado de crear una tabla con los alumnos registrados en la base de datos, se le envía como parámetro el usuario actual

```
1.          public function getAlumnosController($idUser){ //parametro
   $idUser (el usuario actual)
2.                  $informacion = Crud::vistaXTablaModel("alumnos");//
   ejecucion del metodo del modelo
3.                  if(!empty($informacion)){
4.                          //si el resultado no esta vacio, imprimir
   los datos de los usuarios
5.          foreach ($informacion as $row => $item) {
6.                  $grupo = Crud::getRegModel($item["id_grupo"],"grupo
   s");
7.                  $carrera = Crud::getRegModel($item["id_carrera"],"c
   arreras");
8.                  echo "<tr>";
9.                  echo "<td>".$item['matricula']."</td>";
10.                       echo "<td>".$item['nombre']."</td>";
11.                       echo "<td>".$item['apellidos']."</td>";
12.                       echo "<td>".$carrera['nombre']."</td>";
13.                       echo "<td>".$grupo['nombre']."</td>";
14.                       echo '<td><a href="'.$item["imagen"].'"
   id="ver_btn'.$item["id"].'"
   onclick="ver_imagen('.$item["id"].');">Ver</a></td>';
15.                       echo "<td>"."<a class='btn btn-secondary fa
   fa-edit'
   href='index.php?action=editar_alumno&id=".$item['id']."'></a></td>"
   ;
16.                    //mandar por propiedad onclick el id del
   elemento tag a para eleminarlo
17.                                  echo "<td>"."<a class='btn btn-danger
   fa fa-trash' id='borrar_btn".$item["id"]."'
   onclick='b(".$item["id"].");'
```

```
   href='index.php?action=borrar&tipo=alumnos&id=".$item['id']."'></a>
   </td>";
18.
19.                   echo "</tr>";
20.                 }
21.
22.             }
23.
24.               }
```

**Método getSesionesController**

Este método es de tipo público y su función es obtener las sesiones que se tienen registradas en la base de datos, para obtener las sesiones, necesita del método getSesionesControllerModel de la clase Crud.

```
1.   public function getSesionesController(){
2.     $informacion = Crud::getSesionesControllerModel();//ejecucion
   del metodo del modelo
3.     if(!empty($informacion)){
4.       //si el resultado no esta vacio, imprimir los datos de los
   usuarios
5.         foreach ($informacion as $row => $item) {
6.             echo "<tr>";
7.               echo "<td>".$item['matricula']."</td>";
8.               echo "<td>".$item['nombre']."</td>";
9.               echo "<td>".$item['actividad']."</td>";
10.                   echo "<td>".$item['hora']."</td>";
11.                   echo '<td><a href="'.$item["imagen"].'"
   id="ver_btn'.$item["id_alumno"].'"
   onclick="ver_imagen('.$item["id_alumno"].');">Ver</a></td>';
12.                   echo "<td><button type='button' id='borrar-
   ".$item['id']."-".$item['matricula']."' class='btn btn-danger'><i
   class='fa fa-close'></i></button></td>";
13.                   echo "<td><button type='button' id='asistencia-
   ".$item['id']."-".$item['matricula']."' class='btn btn-success'><i
   class='fa fa-check'></i></button></td>";
14.                   //echo "<td><div class='checkbox'><label><input
   class='form_control' type='checkbox'></div></td>";
15.                 echo "</tr>";
16.               }
17.
18.           }
19.       }
```

**Método getMatriculasSesionesController**

Este método, hace uso del método getSesionesControllerModel de la clase Crud, con la finalidad de obtener las matrículas.

```
1.   public function getMatriculasSesionesController(){
2.     $informacion = Crud::getSesionesControllerModel();//ejecucion
   del metodo del modelo
3.     if(!empty($informacion)){
4.       //si el resultado no esta vacio, imprimir los datos de los
   usuarios
5.       for($i=0;$i<sizeof($informacion);$i++){
6.         if($i!=sizeof($informacion)-1)
7.           echo "'".$informacion[$i]['matricula']."',";
8.         else
9.           echo "'".$informacion[$i]['matricula']."'";
10.            }
11.
12.          }
13.        }
```

**Método getGruposController**

Método público que permite la creación de una tabla con los grupos registrados en la base de datos, requiere del método vistaXTablaModel de la clase Crud, llevando como parámetro el nombre de la tabla.

```
1.         public function getGruposController(){ //parametro $idUser
   (el usuario actual)
2.                 $informacion = Crud::vistaXTablaModel("grupos");//e
   jecucion del metodo del modelo
3.                 if(!empty($informacion)){
4.                         //si el resultado no esta vacio, imprimir
   los datos de los usuarios
5.         foreach ($informacion as $row => $item) {
6.                 $encargado = Crud::getRegModel($item['id_maestro'],
    "usuarios");
7.           echo "<tr>";
8.           echo "<td>".$item['id']."</td>";
9.           echo "<td>".$item['nombre']."</td>";
10.                echo "<td>".$encargado['nombre']."
   " . $encargado["apellidos"]."</td>";
11.                echo "<td>"."<a class='btn btn-secondary fa fa-
   edit'
   href='index.php?action=editar_grupo&id=".$item['id']."'></a></td>";
12.                   //mandar por propiedad onclick el id del
   elemento tag a para eleminarlo
13.                                         echo "<td>"."<a class='btn
   btn-danger fa fa-trash' id='borrar_btn".$item["id"]."'
   onclick='b(".$item["id"].");'
```

```
   href='index.php?action=borrar&tipo=grupos&id=".$item['id']."'></a><
   /td>";
14.
15.                   echo "</tr>";
16.
17.
18.                                 }
19.             }
20.
21.               }
```

**Método getActividadesController**

Es un método de tipo público que permite crear una tabla con las actividades que se tienen registradas en la base de datos, para ello, hace uso del método vistaXTablaModel de la clase Crud, solo manda como parámetro el nombre de la tabla de la base de datos.

```
1.   public function getActividadesController(){ //parametro $idUser
   (el usuario actual)
2.     $informacion = Crud::vistaXTablaModel("actividades");//ejecucio
   n del metodo del modelo
3.     if(!empty($informacion)){
4.       //si el resultado no esta vacio, imprimir los datos de los
   usuarios
5.         foreach ($informacion as $row => $item) {
6.           echo "<tr>";
7.           echo "<td>".$item['id']."</td>";
8.           echo "<td>".$item['nombre']."</td>";
9.           echo "<td>"."<a class='btn btn-secondary fa fa-edit'
   href='index.php?action=editar_actividad&id=".$item['id']."'></a></t
   d>";
10.                   //mandar por propiedad onclick el id del
   elemento tag a para eleminarlo
11.                echo "<td>"."<a class='btn btn-danger fa fa-trash'
   id='borrar_btn".$item["id"]."' onclick='b(".$item["id"].");'
   href='index.php?action=borrar&tipo=actividades&id=".$item['id']."'>
   </a></td>";
12.
13.                   echo "</tr>";
14.
15.
16.             }
17.             }
18.
19.         }
```

**Método getUnidadesController**

Es un método que se encarga de imprimir una tabla con las unidades registradas en la base de datos, para mostrar la informacion, se requiere de la clase Crud, específicamente del método vistaXTablaModel, con el nombre de la tabla como parámetro.

```
1.   public function getUnidadesController(){ //parametro $idUser (el
   usuario actual)
2.     $informacion = Crud::vistaXTablaModel("unidades");//ejecucion
   del metodo del modelo
3.     if(!empty($informacion)){
4.       //si el resultado no esta vacio, imprimir los datos de los
   usuarios
5.         foreach ($informacion as $row => $item) {
6.           echo "<tr>";
7.           echo "<td>".$item['id']."</td>";
8.           echo "<td>".$item['nombre']."</td>";
9.           echo "<td>".$item['fecha_inicio']."</td>";
10.                echo "<td>".$item['fecha_fin']."</td>";
11.                echo "<td>"."<a class='btn btn-secondary fa fa-
   edit'
   href='index.php?action=editar_unidad&id=".$item['id']."'></a></td>"
   ;
12.                  //mandar por propiedad onclick el id del
   elemento tag a para eleminarlo
13.                echo "<td>"."<a class='btn btn-danger fa fa-trash'
   id='borrar_btn".$item["id"]."' onclick='b(".$item["id"].");'
   href='index.php?action=borrar&tipo=unidades&id=".$item['id']."'></a
   ></td>";
14.
15.                  echo "</tr>";
16.
17.
18.            }
19.            }
20.
21.        }
```

**Método getCarrerasController**

Crea un a tabla con las carreras registradas en la base de datos, para ello, se requiere del método vistaXTablaModel con el nombre de la tabla como parámetro.

```
1.         public function getCarrerasController(){ //parametro
   $idUser (el usuario actual)
2.                 $informacion = Crud::vistaXTablaModel("carreras");/
   /ejecucion del metodo del modelo
3.                 if(!empty($informacion)){
```

```
4.                          //si el resultado no esta vacio, imprimir
   los datos de los usuarios
5.          foreach ($informacion as $row => $item) {
6.            echo "<tr>";
7.            echo "<td>".$item['id']."</td>";
8.            echo "<td>".$item['nombre']."</td>";
9.            echo "<td>"."<a class='btn btn-secondary fa fa-edit'
   href='index.php?action=editar_carrera&id=".$item['id']."'></a></td>
   ";
10.                    //mandar por propiedad onclick el id del
   elemento tag a para eleminarlo
11.                                          echo "<td>"."<a class='btn
   btn-danger fa fa-trash' id='borrar_btn".$item["id"]."'
   onclick='b(".$item["id"].");'
   href='index.php?action=borrar&tipo=carreras&id=".$item['id']."'></a
   ></td>";
12.
13.                   echo "</tr>";
14.
15.
16.                                 }
17.            }
18.
19.              }
```

### Método getSelectForUsuarios

Permite la creación de un Select con los usuarios registrados, dependiendo del tipo de usuario, ya sea maestro o encargado.

```
1.          public function getSelectForUsuarios($tipo, $firstID){
2.                   $informacion = Crud::vistaXTablaModel("usuarios");
   //se obtienen todos los usuarios de la bd mediante la conexion al
   modelo
3.                  if(!empty($informacion)){
4.                          if($firstID==""){
5.                                  foreach ($informacion as $row => $i
   tem) {
6.                                          if($item["tipo"]==$tipo){
7.                                                  echo "<option
   value='".$item['id']."'>".$item['nombre']. "
   " .$item["apellidos"] ."</option>";
8.                                          }
9.                                  }
10.                                }else{
11.                                         $reg = Crud::getRegModel($fir
   stID, "usuarios");
12.                                         //se coloca primero la opcion
   del select del usuario
13.                                         echo "<option
   value='".$reg['id']."'>".$reg['nombre']."
   ".$reg["apellidos"]."</option>";
```

```
                                        foreach ($informacion as $row
    => $item) { //se imprimen los usuarios restantes
                                            if($item['id']!=$firs
   tID && $item["tipo"]==$tipo)
                                                echo "<option
   value='".$item['id']."'>".$item['nombre']. "
   " .$item["apellidos"] ."</option>";
                                        }
                                }


                        }
                }
```

### Método getSelectForGrupos

Este método permite la creación de un Select con los grupos registrados en la base de datos.

```
        public function getSelectForGrupos($firstID){
                $informacion = Crud::vistaXTablaModel("grupos"); //
   se obtienen todos los grupos de la bd mediante la conexion al
   modelo
                if(!empty($informacion)){
                        if($firstID==""){
                                foreach ($informacion as $row => $i
   tem) {
                                                $petic = Crud::getR
   egModel($item["id_maestro"], "usuarios");
           echo "<option
   value='".$item['id']."'>".$item['nombre']."  ". $petic["nombre"]."
   " .$petic["apellidos"] ."</option>";
                                }
                        }else{
                                        $reg = Crud::getRegModel($fir
   stID, "grupos");
                                        //se coloca primero la opcion
   del select del grupo
              $petic = Crud::getRegModel($reg["id_maestro"], "usuar
   ios");

                                        echo "<option
   value='".$reg['id']."'>".$reg['nombre']."  ". $petic["nombre"]."
   " .$petic["apellidos"] ."</option>";
                                        foreach ($informacion as $row
    => $item) { //se imprimen los grupos restantes
                                            if($item['id']!=$firs
   tID){
                                                $petic = Crud
   ::getRegModel($item["id_maestro"], "usuarios");
                    echo "<option
   value='".$item['id']."'>".$item['nombre']."  ". $petic["nombre"]."
   " .$petic["apellidos"] ."</option>";
```

```
                }
                                        }
                                }


                        }
                }
```

**Método getSelectForCarreras**

Crea un Select con las carreras que se tienen registradas en la base de datos, en caso de requerir un Select con un valor al principio, ya sea cuando se edita, se ingresa en una variable llamada $firstID.

```
        public function getSelectForCarreras($firstID){
                $informacion = Crud::vistaXTablaModel("carreras");
  //se obtienen todos los grupos de la bd mediante la conexion al
  modelo
                if(!empty($informacion)){
                        if($firstID==""){
                                foreach ($informacion as $row => $i
  tem) {
                                                echo "<option
  value='".$item['id']."'>".$item['nombre']."</option>";
                                }
                        }else{
                                $reg = Crud::getRegModel($firstID,
  "carreras");
                                        //se coloca primero la opcion
  del select del grupo
                                        echo "<option
  value='".$reg['id']."'>".$reg['nombre']."</option>";
                                        foreach ($informacion as $row
   => $item) { //se imprimen los grupos restantes
                                                if($item['id']!=$firs
  tID)
                                                        echo "<option
  value='".$item['id']."'>".$item['nombre']."</option>";
                                        }
                                }
                        }
                }
```

**Método getUnidadId**

Es un método que retorna en id de la unidad actual, esto a través del método getUnidadIdModel de la clase Crud.

```
1.   public function getUnidadId(){
2.     $id_unidad = Crud::getUnidadIdModel(); //se obtienen todos los
   grupos de la bd mediante la conexion al modelo
3.
4.     echo $id_unidad[0];
5.   }
```

**Método registroUsuarioController**

Método encargado de verificar si es que se presionó el botón de registro, en caso de ser asi, se toma la informacion de los controles y se ejecuta el método de registro en el modelo. Si la inserción retorna success, entonces imprime una alerta de éxito, en caso contrario, imprime una alerta de error.

```
1.         public function registroUsuarioController(){ //registro de
   usuario (maestro o encargado)
2.                 if(isset($_POST['btn_agregar'])){//verificar clic
   en el boton
3.                         //crear array con los datos a registrar
   tomados de los controles
4.                         $data = array('codigo'=> $_POST['codigo'],
5.                                                 'nombre'=> $_POST['
   nombre'],
6.                                                 'apellidos'=> $_POS
   T['apellidos'],
7.                                                 'email'=> $_POST['e
   mail'],
8.                                                 'password'=> $_POST
   ['password'],
9.                                                 'tipo'=> $_POST['ti
   po']
10.                                             );
11.                             //peticion al modelo del reigstro del
   producto mandando como param la informacion de este
12.                             $registro = Crud::registroUsuarioMode
   l($data);
13.                             if($registro == "success"){ //verific
   ar la respuesta del modelo
14.                 echo "<script>swal('Exito!','Usuario
   registrado!','success');
15.                 window.location='index.php?action=usuarios';</script>"
   ;
16.                             }else{
```

```
                                        echo "<script>swal('Error','El codigo ingresado ya fue usado!. Por favor, ingresa otro','error');</script>";
                                    }
                            }
                }
```

**Método registroSesionCai**

Es el método encargado de realizar el registro de una sesión de CAI en la base de datos, la informacion pasa al método de registroSesionCaiModel, de la clase Crud.

```
    public function registroSesionCai($data_val){ //registro de sesion cai
        $data = $data_val;  //Valores mandados de la peticion POST

        //peticion al modelo del registro de sesion
        $registro = Crud::registroSesionCaiModel($data);
        echo $registro;
    }
```

**Método registroAlumnoController**

Este método permite verificar si se presionó el botón de registro, en caso de ser asi, se toman los datos de los campos y se ejecuta el método que permite la inserción del alumno a la base de datos. Si la inserción retorna success, entonces imprime una alerta de éxito, en caso contrario, imprime una alerta de error.

**Método registroGrupoController**

Es un método que verifica que cuando se presione el botón de registro, tome la informacion del grupo y la guarde como un registro más en la base de datos. Si la inserción retorna success, entonces imprime una alerta de éxito, en caso contrario, imprime una alerta de error.

```
        public function registroGrupoController(){ //registro de grupo
                if(isset($_POST['btn_agregar'])){//verificar clic en el boton
                        //crear array con los datos a registrar tomados de los controles
                        $data = array('nombre'=> $_POST['nombre'],
                                      'id_maestro'=> $_POST['id_maestro']
```

```
                                        );
                        //peticion al modelo del reigstro del producto mandando como param la informacion de este
                        $registro = Crud::registroGrupoModel($data);
                        if($registro == "success"){ //verificar la respuesta del modelo
                echo "<script>swal('Exito!','Grupo registrado!','success');
                window.location='index.php?action=grupos';</script>";
                                }else{
                                        echo "<script>swal('Error','Ocurrio un error al registrar el grupo','error');</script>";
                                }
                    }
                }
```

**Método registroActividadController**

Es un método que verifica que cuando se presione el botón de registro, tome la informacion de la actividad a registrar y la guarde como un registro más en la base de datos. Si la inserción retorna success, entonces imprime una alerta de éxito, en caso contrario, imprime una alerta de error.

```
  public function registroActividadController(){ //registro de actividad
    if(isset($_POST['btn_agregar'])){//verificar clic en el boton
      //crear array con los datos a registrar tomados de los controles
      $data = array('nombre'=> $_POST['nombre']
          );
      //peticion al modelo del reigstro del producto mandando como param la informacion de este
      $registro = Crud::registroActividadModel($data);
      if($registro == "success"){ //verificar la respuesta del modelo
        echo "<script>swal('Exito!','Actividad registrada!','success');
            window.location='index.php?action=actividades';</script>";
          }else{
            echo "<script>swal('Error','Ocurrio un error al registrar la actividad','error');</script>";
          }
        }
      }
```

**Método registroUnidadController**

Es un método que verifica que cuando se presione el botón de registro, tome la informacion de la unidad que quiere registrar y la guarde como un registro más en la base de datos. Si la inserción retorna success, entonces imprime una alerta de éxito, en caso contrario, imprime una alerta de error.

```
1.   public function registroUnidadController(){ //registro de unidad
2.     if(isset($_POST['btn_agregar'])){//verificar clic en el boton
3.       $fecha_inicio = date('Y-m-
   d', strtotime($_POST['fecha_inicio']));
4.       $fecha_fin = date('Y-m-d', strtotime($_POST['fecha_fin']));
5.       //crear array con los datos a registrar tomados de los
   controles
6.       $data = array('nombre'=> $_POST['nombre'],
7.         'fecha_inicio'=> $fecha_inicio,
8.         'fecha_fin'=> $fecha_fin
9.           );
10.            //peticion al modelo del reigstro del producto mandando
   como param la informacion de este
11.            $registro = Crud::registroUnidadModel($data);
12.            if($registro == "success"){ //verificar la respuesta
   del modelo
13.              echo "<script>swal('Exito!','Unidad
   registrada!','success');
14.             window.location='index.php?action=unidades';</script>"
   ;
15.            }else{
16.              echo "<script>swal('Error','Ocurrio un error al
   registrar la unidad','error');</script>";
17.            }
18.          }
19.        }
```

**Método registroCarreraController**

Es un método que verifica que cuando se presione el botón de registro, tome la informacion de la nueva carrera a registrar y la guarde como un registro más en la base de datos. Si la inserción retorna success, entonces imprime una alerta de éxito, en caso contrario, imprime una alerta de error.

```
1.         public function registroCarreraController(){ //registro de
   carrera
2.                 if(isset($_POST['btn_agregar'])){//verificar clic
   en el boton
3.                         //crear array con los datos a registrar
   tomados de los controles
4.                         $data = array('nombre'=> $_POST['nombre']
```

```
                                        );
                        //peticion al modelo del reigstro del producto mandando como param la informacion de este
                        $registro = Crud::registroCarreraModel($data);
                        if($registro == "success"){ //verificar la respuesta del modelo
        echo "<script>swal('Exito!','Carrera registrada!','success');
            window.location='index.php?action=carreras';</script>";
                                }else{
                                        echo "<script>swal('Error','Ocurrio un error al registrar la carrera','error');</script>";
                                }
                    }
             }
```

### Método getCategoriaController

Este método permite, dado un id de categoría, obtener los datos de la base de datos de ese id, para posteriormente, mostrarlos en los controles y de esta manera poder editar la información.

```
        public function getCategoriaController(){
                $id = (isset($_GET['id'])) ? $_GET['id'] : ""; //verificacion del id
                $peticion = Crud::getRegModel($id, 'categorias', $_SESSION['tienda']); //peticion al modelo del registro especificado por el id
                if(!empty($peticion)){
                        echo "
                                <div class='form-group'>
                   <p>
                   <label>Id</label>
                   <input type='text' class='form-control' name='id' value='".$peticion['id']."' placeholder='' required='' readonly='true'>
                        </p>
                        </div>
                        <div class='form-group'>
                          <p>
                          <label>Nombre</label>
                          <input type='text' class='form-control' name='nombre' value='".$peticion['nombre']."' placeholder='Ingresa el nombre de la categoria' required=''>
                        </p>
                        </div>
                        ";
                        }
             }
```

**Método getUsuarioController**

Es el método encargado de que, dado un id de usuario, se obtengan los datos de la base de datos y se impriman los controles con la informacion del usuario con ese id y de esta manera poder modificarlos.

**Método getAlumnoController**

Es el encargado de obtener la informacion o datos de un alumno, esto por medio de un id proporcionado, de esta manera se busca la informacion en la base de datos y se imprime la informacion para su posterior edición.

**Método getGrupoController**

Método encargado de obtener los datos de un grupo mediante su id, a través de la base de datos, se imprimen los controles con los datos y los valores o informacion registrada, para editarlos posteriormente.

**Método getActividadController**

Este método permite que dado un id de una actividad, se obtenga la informacion de la base de datos y se imprima en los controles, de esta manera se puede editar.

**Método getUnidadController**

Es un método de tipo público que permite obtener la informacion de la base de datos de una unidad en específico, la informacion se obtiene mediante el id de la unidad, de esta manera la informacion será mostrada en los controles para su posterior edición.

**Método getCarreraController**

Método que permite, dado un id de una carrera, obtener los datos de la base de datos e imprimir la informacion en los controles para que se pueda editar posteriormente.

**Método actualizarUsuarioController**

Es un método que verifica si se dio clic en el botón de actualización, de esta manera realiza la actualización mediante la ejecución del método actualizarUsuarioModel que se encuentra en la clase Crud. Si el método del modelo, retorna un success, se muestra una alerta de éxito, en caso contrario, imprime una alerta de error

```
1.         public function actualizarUsuarioController(){ //actualizac
   ion de usuario
2.                 if(isset($_POST['btn_actualizar'])){ //verificacion
   de clic en el boton
3.                         //se toman los valores de los controles y
   se guardan en un array
4.                         $data = array(
5.                                 "codigo"=>$_POST['codigo'],
6.                                 "nombre"=>$_POST['nombre'],
7.                                 "apellidos"=>$_POST['apellidos'],
8.                                 "email"=>$_POST['email'],
9.                                 "password"=>$_POST['password'],
10.                                      "tipo"=>$_POST['tipo'],
11.             "id"=>$_POST['id']
12.                                );
13.
14.                                //se realiza la ejecucion del metodo
   que actualiza un alumno en el modelo, mandando los parametros
   correspondientes, datos y matricula
15.                                $peticion = Crud::actualizarUsuarioMo
   del($data);
16.                                if($peticion == "success"){ //verific
   acion de la respuesta por el modelo
17.                                echo "<script>
18.                                swal('Exito!','Usuario
   actualizado!','success');
19.                                window.location='index.php?action=usu
   arios';</script>";
20.                                }else{
21.                                        echo "<script>swal('Error',
   'Ocurrio un error al guardar los cambios', 'error');</script>";
22.                                }
23.                        }
24.             }
```

**Método actualizarAlumnoController**

Es un método que verifica si se dio clic en el botón de actualización, de esta manera realiza la actualización mediante la ejecución del método actualizarAlumnoModel que se encuentra en la clase Crud. Si el método del modelo, retorna un success, se muestra una alerta de éxito, en caso contrario, imprime una alerta de error.

```
1.          public function actualizarAlumnoController(){ //actualizar
   alumno
2.                  if(isset($_POST['btn_actualizar'])){ //verificacion
   de clic en el boton
3.                          //se toman los valores de los controles y
   se guardan en un array
4.                          $data = array(
5.                                  "matricula"=>$_POST['matricula'],
6.                                  "nombre"=>$_POST['nombre'],
7.                                  "apellidos"=>$_POST['apellidos'],
8.                                  "id_carrera"=>$_POST['id_carrera'],
9.                                  "id_grupo"=>$_POST['id_grupo'],
10.             "id"=>$_POST['id']
11.
12.                                 );
13.
14.                                 //se realiza la ejecucion del metodo
   que actualiza un alumno en el modelo, mandando los parametros
   correspondientes, datos y matricula
15.                                 $peticion = Crud::actualizarAlumnoMod
   el($data);
16.                                 if($peticion == "success"){ //verific
   acion de la respuesta por el modelo
17.                                 echo "<script>
18.                                 swal('Exito!','Alumno
   actualizado!','success');
19.                                 window.location='index.php?action=alu
   mnos';</script>";
20.                                 }else{
21.                                         echo "<script>swal('Error',
   'Ocurrio un error al guardar los cambios', 'error');</script>";
22.                                 }
23.                         }
24.             }
```

### Método setAsistenciaController

Actualiza la asistencia de determinado registro en la lista de la sesión de CAI. Hace uso del método setAsistenciaModel de la clase Crud.

```
1.    public function setAsistenciaController($id){
2.      $peticion = Crud::setAsistenciaModel($id);
3.      echo $peticion;
4.    }
```

**Método actualizarGrupoController**

Es un método que verifica si se dio clic en el botón de actualización, de esta manera realiza la actualización mediante la ejecución del método actualizarGrupoModel que se encuentra en la clase Crud. Si el método del modelo, retorna un success, se muestra una alerta de éxito, en caso contrario, imprime una alerta de error

```
1.         public function actualizarGrupoController(){ //se actualiza
   un grupo
2.                 if(isset($_POST['btn_actualizar'])){ //verificacion
   de clic en el boton
3.                         //se toman los valores de los controles y
   se guardan en un array
4.                         $data = array(
5.                                 "nombre"=>$_POST['nombre'],
6.                                 "id_maestro"=>$_POST['id_maestro'],
7.                                 "id"=>$_POST['id']
8.                         );
9.
10.                                 //se realiza la ejecucion del metodo
   que actualiza un alumno en el modelo, mandando los parametros
   correspondientes, datos y matricula
11.                                 $peticion = Crud::actualizarGrupoMode
   l($data);
12.                                 if($peticion == "success"){ //verific
   acion de la respuesta por el modelo
13.                                 echo "<script>
14.                                 swal('Exito!','Grupo
   actualizado!','success');
15.                                 window.location='index.php?action=gru
   pos';</script>";
16.                                 }else{
17.                                         echo "<script>swal('Error',
   'Ocurrio un error al guardar los cambios', 'error');</script>";
18.                                 }
19.                         }
20.             }
```

**Método getServerHourController**

Método que se encarga de obtener la hora actual del servidor mediante una consulta, esta última se realiza en el método getServerHour de la clase Crud.

```
1.   public function getServerHourController(){
2.     $peticion = Crud::getServerHour();
3.     echo $peticion;
4.   }
```

**Método getServerDateController**

Método que se encarga de obtener la fecha actual del servidor mediante una consulta, esta última se realiza en el método getServerDate de la clase Crud.

```
1.   public function getServerDateController(){
2.     $peticion = Crud::getServerDate();
3.     echo $peticion;
4.   }
```

**Método actualizarActividadController**

Método que verifica si se presionó el botón de actualización, de ser asi, realiza la actualización mediante la ejecución del método actualizarActividadModel de la clase Crud. Si el método del modelo, retorna un success, se muestra una alerta de éxito, en caso contrario, imprime una alerta de error.

**Método actualizarUnidadController**

Método que verifica si se presionó el botón de actualización, de ser asi, realiza la actualización mediante la ejecución del método actualizarUnidadModel de la clase Crud. Si el método del modelo, retorna un success, se muestra una alerta de éxito, en caso contrario, imprime una alerta de error.

**Método actualizarCarreraController**

Método que verifica si se presionó el botón de actualización, de ser asi, realiza la actualización mediante la ejecución del método actualizarCarreraModel de la clase Crud. Si el método del modelo, retorna un success, se muestra una alerta de éxito, en caso contrario, imprime una alerta de error.

**Método borrarController**

Método que dado un id y nombre de la tabla, ejecuta el método borrarXModel del modelo y borra el registro especificado en base a la tabla e id que se proporcionó. Si el método del modelo, retorna un success, se muestra una alerta de éxito, en caso contrario, imprime una alerta de error.

**Método borrarSesionController**

Método que dado un id y un nombre de tabla, ejecuta el método del modelo y borra el registro especificado en base a la tabla que se proporcionó, este método utiliza un método de la clase Crud y manda como parámetro el id y como nombre de la tabla “sesion_cai”.

**Método checkAvailabilityOfUserCodeController**

Este método verifica la disponibilidad de una columna en una tabla, hace uso del método checkAvailabilityOfUserCodeModel que se encuentra en la clase Crud en el modelo.

```
1.   public function checkAvailabilityOfUserCodeController($code, $tab
   le, $nid){
2.
3.     $respuesta = Crud::checkAvailabilityOfUserCodeModel($code, $tab
   le, $nid);
4.
5.
6.     echo($respuesta['a']);
7.
8.   }
```

**Método getDetailsFromAlumno**

Método que, mediante una consulta, obtiene los detalles del alumno, para ello, es necesario el codigo y nombre de la tabla.

```
1.   public function getDetailsFromAlumno($code, $table){
2.
3.     $respuesta = Crud::getDetailsFromAlumnoModel($code, $table);
4.
5.     echo(json encode($respuesta));
6.   }
```

**Método getCantidadAlumnos**

Es un método encargado de contar los alumnos de la sesión actual, hace uso del método contarAlumnosController de la clase Crud.

```
1.   public function getCantidadAlumnos(){
2.     $total = Crud::contarAlumnosController();
```

```
3.      return $total;
4.    }
```

**Método sesion_caiHeaderAlumnosController**

Función que imprime el widget de alumnos en el aula, hace uso del método contarAlumnosController de la clase Crud, para obtener el total de alumnos en la sesión actual.

**Método sesion_caiHeaderController**

Método que es el encargado de poner la interfaz de ingreso de alumnos a la sesión, cuenta con un campo de texto para la matrícula y para la actividad a realizar durante la sesión.

**Método getSelectForActividades**

Método que crea un Select con las actividades registradas, utiliza el método vistaXTablaModel de la clase Crud para obtener todos los registros de la tabla Actividades.

```
1.    public function getSelectForActividades(){
2.      $informacion = Crud::vistaXTablaModel("actividades"); //se
   obtienen todos los grupos de la bd mediante la conexion al modelo
3.      if(!empty($informacion)){
4.        foreach ($informacion as $row => $item) {
5.            echo "<option
   value='".$item['id']."'>".$item['nombre']."</option>";
6.        }
7.      }
8.    }
```

**Método getSelectForAlumnos**

Método que crea un Select con las actividades registradas, utiliza el método vistaXTablaModel de la clase Crud para obtener todos los registros de la tabla Alumnos.

```
1.    public function getSelectForAlumnos(){
2.      $informacion = Crud::vistaXTablaModel("alumnos"); //se obtienen
   todos los grupos de la bd mediante la conexion al modelo
```

```
3.      if(!empty($informacion)){
4.        foreach ($informacion as $row => $item) {
5.            echo "<option
   value='".$item['matricula']."'>".$item['matricula']." |
   ". $item["nombre"]." ".$item["apellidos"]."</option>";
6.        }
7.      }
8.    }
```

**Método deleteSesionesSinAsistenciaController**

Método que borra las sesiones que no tengan asistencia o con valor 0 en la base de datos y que además no pertenezcan a la sesión actual. Utiliza el método deleteSesionesSinAsistenciaModel de la clase Crud.

```
1.    public function deleteSesionesSinAsistenciaController(){
2.      $peticion = Crud::deleteSesionesSinAsistenciaModel();
3.      echo $peticion;
4.    }
```

**Método getSelectForGruposReportes**

Método que crea un Select con los grupos registrados en la base de datos filtrados por el tipo de usuario en sesión.

```
1.    public function getSelectForGruposReportes(){
2.      if($_SESSION["user_info"]["tipo"]=="teacher") //si es maestro,
   mandar el id como parametro para filtrar los grupos
3.          $informacion = Crud::vistaGruposModel($_SESSION["user_info"
   ]["id"]); //se obtienen todos los grupos de la bd mediante la
   conexion al modelo
4.      else
5.        $informacion = Crud::vistaGruposModel(""); //se obtienen
   todos los grupos de la bd mediante la conexion al modelo
6.        foreach ($informacion as $row => $item) { //se imprimen los
   grupos
7.            $petic = Crud::getRegModel($item["id_maestro"], "usuarios
   ");
8.            echo "<option
   value='".$item['id']."'>".$item['nombre']."  ". $petic["nombre"]."
   " .$petic["apellidos"] ."</option>";
9.      }
10.       }
```

**Método getSelectForUnidades**

Método que crea un Select con las unidades registradas en la base de datos.

```
1.   public function getSelectForUnidades(){
2.     $informacion = Crud::vistaXTablaModel("unidades"); //se
   obtienen todos las las unidades de la bd mediante la conexion al
   modelo
3.     foreach ($informacion as $row => $item) { //se imprimen las
   unidades a manera de select
4.         echo "<option
   value='".$item['id']."'>".$item['nombre']."</option>";
5.     }
6.   }
```

**Método getReporteController**

Método que crea una tabla con los alumnos de un grupo seleccionado y según la unidad seleccionada, filtra el número de horas registradas en la tabla “sesion_cai” de la base de datos.

**Método getDetalleDeReporteController**

Método que dado un id de alumno y el id de una unidad, obtiene las actividades realizadas en cada una de las horas en esa unidad hechas por el alumno, la informacion se muestra en forma de tabla.

# Vista

La carpeta de la vista contiene:

- Diez carpetas
- Seis archivos php.

**Carpeta actividades**

La carpeta actividades contiene tres archivos php que pertenecen al módulo de actividades del sistema, contiene el archivo que tiene la vista de las actividades registradas, el formulario para la edición y registro de una actividad.

**Carpeta alumnos**

La carpeta alumnos contiene tres archivos php que son parte del módulo de alumnos del sistema de CAI, contiene el archivo que es la vista de todos los alumnos registrados en la base de datos, además de los formularios de editar y registrar alumnos.

**Carpeta carreras**

La carpeta carreras contiene tres archivos php que pertenecen al módulo de carreras del sistema, contiene el archivo que tiene la vista de las carreras registradas, el formulario para la edición y registro de una carrera.

**Carpeta dist**

Esta carpeta es parte fundamental en cuanto al diseño del sistema de CAI, contiene archivos css y js.

**Carpeta grupos**

La carpeta grupos contiene tres archivos php que pertenecen al módulo de gestión de grupos del sistema, contiene el archivo que tiene la vista de las actividades registradas, el formulario para la edición y registro de un grupo.

**Carpeta plugins**

Esta carpeta permite hacer uso de diferentes widgets, asi como herramientas que mejoran la interfaz del sistema.

**Carpeta reportes**

La carpeta grupos contiene tres archivos php que pertenecen al módulo de gestión de grupos del sistema, contiene el archivo que tiene la vista de las actividades registradas, el formulario para la edición y registro de un grupo.

**Carpeta sesion_cai**

Esta carpeta contiene el archivo de la interfaz principal de la sesión actual, donde se puede observar cuantos alumnos han entrado, a la sesión, además donde será el pase de lista de asistencia.

**Carpeta unidades**

La carpeta unidades contiene tres archivos php que pertenecen al módulo de gestión de unidades del sistema, contiene el archivo que tiene la vista de las unidades registradas, el formulario para la edición y registro de una unidad.

**Carpeta usuarios**

La carpeta usuarios contiene tres archivos php que pertenecen al módulo de gestión de usuarios del sistema, contiene el archivo que tiene la vista de los usuarios registrados, el formulario para la edición y registro de un usuario.

**Archivo template.php**

El archivo template.php contiene algunos métodos como getDate que permite obtener la fecha, retorna una cadena de texto con el formato de la fecha (dd/mm/aaaa). Asi como también el método getTime que actualiza la hora en el reloj de la página en la parte superior. El método confirmLogout envía una confirmación de cierre de sesión. Por último, se encuentran los métodos b y c que pide la

confirmación al editar o eliminar un registro y el método ver_imagen que muestra en un sweet alert la imagen del alumno.

**Archivo header.php**

El archivo header.php crea una instancia de la clase MVC que está en el controlador y manda llamar al método showNav.

**Archivo footer.php**

El archivo footer.php hace referencia a algunos archivos js.

**Archivo login.php**

El archivo login.php representa la vista del inicio de sesión, muestra los campos para el codigo de empleado y la contraseña para acceder al sistema.

**Archivo inicio.php**

El archivo inicio.php manda llamar al método de verificarLoginController y mostrarInicioController de la clase MVC.

**Archivo logout.php**

El archivo logout.php destruye la sesión y redirige en el index.php.