



UNIVERSIDAD  
NACIONAL  
DE COLOMBIA  

---

S E D E   B O G O T Á

**DOCUMENTO FINAL - ACASI**

Integrantes:

Layonel Camilo Leal Díaz  
Santiago Jose Salgado López  
Cristian Camilo Torres Sánchez  
Carol Jinneth Barahona Salgado

Docente: Néstor Germán Bolívar Pulgarín  
Programación orientada a objetos

<b>Resumen</b>	<b>3</b>
<b>Introducción</b>	<b>3</b>
<b>Estado del arte</b>	<b>4</b>
<b>Tecnologías asociadas</b>	<b>4</b>
<b>Diseño del prototipo de software</b>	<b>4</b>
<b>Retos vigentes y análisis prospectivo</b>	<b>5</b>
<b>Conclusiones</b>	<b>5</b>
<b>Controladores</b>	<b>6</b>
Package Controlador	6
clase LoginController	6
Package Controlador.Corresponsabilidad	7
Clase ConCrearCorresponsabilidad	7
Package Controlador.usuarios	8
Clase ConAU_BuscarUsuario	8
Clase ConAU_CrearUsuario	9
Clase ConAU_InfoUsuario	10
Package Controlador.Estadisticas	11
Clase ControladorEstadisticas	11
Clase ControladorGraficoDiasDelAño	13
Clase ControladorGraficoAño	15
<b>Modelos</b>	<b>17</b>
Package Modelo	17
Clase Servicio	17
Clase Computador	18
Clase Sala	19
Clase Sesión	21
Package Modelo.Users	21
Clase Usuario	21
Clase Estudiante	21
Clase profesor	22
Clase AbstractUser	22
<b>Vistas</b>	<b>23</b>
Package Vista	23
Vista Login	23
Vista Application	24
Vista AU_CrearUsuario	24
Vista AU_BuscarUsuario	25
Vista AU_InfoUsuario	26
Package Vista.components	27

Package Vista.recursos.imagenes	27
Package Vista.utils	27

## Resumen

ACASI nace para reemplazar una aplicación anticuada la cual está en funcionamiento en estos momentos en la sala de cómputo de “ciencias humanas”. ACASI resuelve los problemas presentados actualmente como errores de conexión con el servidor, la búsqueda de datos guardados en la base de datos. También permite que el profesor cargo de administrar las salas tenga una interfaz más amigable e intuitiva para gestionar las diversas funciones y métodos que se realizan a la hora de administrar la sala. El reto de esta aplicación es lograr la optimización de los diversos procesos de la sala, empleando funcionalidades de la aplicación anterior y agregando nuevas funciones a nuestro programa. Para la creación de ACASI se utilizó Java como lenguaje principal y Supabase. Para la realización de la aplicación se hizo una investigación sobre las herramientas que nos ofrece Java como sus interfaces gráficas. Y una base de datos para su funcionamiento.

## Introducción

Este documento presenta un proyecto que se da a raíz de una problemática en a la hora de gestionar las diversas salas de equipos de computación de la facultad de “Ciencias Humanas”, y su dificultad frente la organización, control y recopilación de datos, del uso de las salas y sus componentes.

La finalidad de este proyecto es brindar una aplicación, que facilite el trabajo del docente encargado de la administración de las salas, como de los diversos estudiantes de corresponsabilidad que ayudan en el manejo de estas; Esto se realizará mediante una aplicación con una interfaz más intuitiva y amigable, brindando nuevas herramientas, como la creación de gráficas, una búsqueda de estudiantes más rápida, una creación de usuarios más simple y la gestión de estos mismos.

Este proyecto se realizó en el lenguaje de programación Java y NetBeans como asistente. Para complementar la aplicación se utilizó la base de datos Supabase, una base de datos SQL en la nube, con el fin de poder recuperar, modificar y utilizar los datos de manera más sencilla.

## Estado del arte

ACASI toma como referencia la aplicación con la cual se maneja actualmente las salas de equipos, una aplicación que lleva en funcionamiento hace más de una década. Esta aplicación tienen funciones para crear usuarios, hacer búsquedas generales y asignarles equipos, esta última función es hecha mediante texto, lo cual dificulta la asignación de los equipos como el conocer su posición en la sala. Su base de datos funciona mediante un equipo local al cual ya no es posible acceder.



### Tecnologías asociadas

La tecnología que utilizamos en este proyecto es Java, es un lenguaje de programación de propósito general, orientado a objetos y basado en clases. Java es ampliamente utilizado en el desarrollo de aplicaciones empresariales, aplicaciones web, aplicaciones móviles (particularmente en Android), y sistemas integrados, además gracias a la JVM, el mismo código Java puede ejecutarse en diferentes sistemas operativos sin modificación, lo que lo hace altamente portátil. Junto a esto, Java proporciona una rica biblioteca estándar que facilita el desarrollo de aplicaciones en diversas áreas, desde la manipulación de datos hasta la creación de interfaces gráficas de usuario.

Como base de datos decidimos utilizar Supabase que es una plataforma de desarrollo backend que se presenta como una alternativa de código abierto a Firebase. Ofrece funcionalidades como una base de datos en tiempo real, autenticación, almacenamiento de archivos, y funciones serverless. Está diseñada para facilitar la creación y gestión de aplicaciones completas sin necesidad de manejar infraestructura compleja.

### Diseño del prototipo de software

El software de ACASI se diseñó, teniendo en mente, una lógica y visuales que sean simples y fáciles de entender para aquellos que utilizaran la app, además de brindar opciones para la personalización y accesibilidad del usuario. Todo esto se realizó con el pensamiento de no poner más carga en los administradores de las salas de cómputo, facilitando su trabajo.

A continuación mostraremos imágenes del prototipo:

ACASI		SALAS		
Salas	Usuarios	Test	Ocho	Sala1
		Administrar	Administrar	Administrar
		Sociologia	Sala3	Sala4
		Administrar	Administrar	Administrar
				+

### Retos vigentes y análisis prospectivo

Como una aplicación en la que se utilizó una base de datos en la nube, uno de los principales retos es el funcionamiento de la aplicación cuando el equipo en el que se está ejecutando no tiene conexión a internet. Esto debido a que, ciertas funciones de esta dependen de una respuesta de la base de datos, la cual no podría enviar datos sin conexión.

Lo ideal sería que los procesos que no requieren conexión se siguieran ejecutando, como también dejar en claro al usuario que si no está conectado a una red, no podrá hacer uso de ciertas funciones.

### Conclusiones

El reto de adaptarnos al código a lo largo del avance del proyecto, el cambio de base de datos, como también el cambio de nuestro ambiente de desarrollo del visual es cuál teníamos planeado usar JavaFX nos llevó a más complicaciones, lo cual nos forzó a adaptar las herramientas que nos brindaba NetBeans para aplicarlas en nuestra aplicación.

Así mismo, los problemas al dividir código y unirlo también se hizo presente, lo cual nos mostró que tener un sistema organizado y metódico, pudo habernos dado mejores resultados, para evitar problemas, simples y en los cuales perdemos tiempo valioso en resolver.

A lo largo de la realización del proyecto hemos aplicado todo lo visto en clase más conocimiento, aparte de herramientas empleadas en la aplicación tales como supabase.

## Controladores

### Package Controlador

#### clase LoginController

**Método LoginController:** Crea un objeto del tipo **Login** en el cual se ejecutarán lo demás métodos de la clase, los demás métodos que se ejecutan, esta requiere que se defina como se ejecuta el controlador (en este caso mediante el **btoLogin**) este requiere de las librerías **ActionEvent** y **ActionListener**

```
public LoginController(Login login) {  
    this.login = login;  
    this.login.btnLogin.addActionListener((java.awt.event.ActionEvent evt) -> {  
        btnLoginActionPerformed(evt);  
    });  
}
```

**Método StartView:** Define el tamaño y ubicación en el panel en el que se aplicara el controlador

```
public void startView() {  
    login.setTitle("Login ACASI");  
    login.setLocationRelativeTo(null);  
}
```

**Método btnLoginActionPerformed:** Captura los datos enviados por el visual **Login** usando Métodos **GetText**, se asegura que el usuario haya ingresado los datos en los campos de texto, así que coincida su formato, llama al método de servicio **iniciarSesion**, se asegura que exista un usuario que coincida con las credenciales ingresadas y crea un nuevo visual **Aplication**.

```

private void btnLoginActionPerformed(ActionEvent e) {
    login.isLoading(true);
    String passwordFormated = new String(login.password.getPassword());
    String usernameText = login.username.getText();

    if (usernameText.isEmpty() | passwordFormated.isEmpty()){
        login.errorMessage.setText("Recuerda completar todos los campos.");
        login.isLoading(false);
        return;
    }

    if (!usernameText.contains("@")){
        login.errorMessage.setText("El usuario no es valido.");
        login.isLoading(false);
        return;
    }

    JSONObject data = Servicio.iniciarSesion(usernameText, passwordFormated);

    if (data.has("errorMessage")){
        login.errorMessage.setText(data.getString("errorMessage"));
        login.isLoading(false);
        return;
    }

    Application application = new Application();
    application.setVisible(true);
    login.setVisible(false);
    login.errorMessage.setText("");
    login.isLoading(false);
}

```

## Package Controlador.Corresponsabilidad

### Clase ConCrearCorresponsabilidad

**Método ConCrearCorresponsabilidad:** Crea un objeto del tipo **CrearCorresponsabilidad** en el cual se ejecutarán lo demás métodos de la clase, los demás métodos que se ejecutan esta requiere que se defina como se ejecuta el controlador (en este caso mediante el **BtoAñadirCorresponsabilidad**) este requiere de las librerías **ActionEvent** y **ActionListener**

```

public ConCrearCorresponsabilidad(CrearCorresponsabilidad crearCorresponsabilidad) {
    this.crearCorresponsabilidad = crearCorresponsabilidad;
    this.crearCorresponsabilidad.BtnAñadirCorresponsabilidad.addActionListener((ActionListener) this);
}

```

**Método Inicio:** Define el tamaño y ubicación en el panel en el que se aplicara el controlador

```

public void Inicio() {
    crearCorresponsabilidad.setLocation(0, 0);
    crearCorresponsabilidad.setSize(0,0);

}

```

**Método actionPerformed:**Captura los datos enviados por el visual  
**CrearCorresponsabilidad** usando M  todos **GetText** y **GetItem**, se asegura que el usuario haya ingresado todos los datos de los campos de texto, llama al m  todo de servicio.

```

public void actionPerformed(ActionEvent e) {
    String name = crearCorresponsabilidad.TxtNewCorName.getText();
    String last_name = crearCorresponsabilidad.TxtNewCorApe.getText();
    String document_number = crearCorresponsabilidad.TxtNewCorDoc.getText();
    String email = crearCorresponsabilidad.TxtNewCorCorreo.getText();
    String time = crearCorresponsabilidad.TxtNewCorTime.getText();
    String faculty = crearCorresponsabilidad.BoxNewCorFaculty.getSelectedItem().toString();
    String career = crearCorresponsabilidad.TxtNewCorCareer.getText().toLowerCase();
    crearCorresponsabilidad.ErrorNewCor.setForeground(Color.red);

    if (name.isEmpty()|last_name.isEmpty()|document_number.isEmpty()|email.isEmpty()){
        crearCorresponsabilidad.ErrorNewCor.setText("Rellene todos los espacios");
        return;
    }

    Servicio.createUser(name, last_name, document_number, email, time, faculty, career);
    crearCorresponsabilidad.ErrorNewCor.setForeground(Color.GREEN);
    crearCorresponsabilidad.ErrorNewCor.setText("Usuario Creado Correctamente");
    crearCorresponsabilidad.TxtNewCorName.setText("");
    crearCorresponsabilidad.TxtNewCorApe.setText("");
    crearCorresponsabilidad.TxtNewCorTime.setText("");
    crearCorresponsabilidad.TxtNewCorDoc.setText("");
    crearCorresponsabilidad.TxtNewCorCorreo.setText("");
    crearCorresponsabilidad.TxtNewCorCareer.setText("");
}

```

## Package Controlador.usuarios

### Clase ConAU\_BuscarUsuario

**M  todo ConAU\_BuscarUsuario:** Crea un objeto del tipo **Au\_BuscarUsuario** en el cual se ejecutar  n los dem  s m  todos de la clase, esta requiere que se defina como se ejecuta el controlador (en este caso mediante el **BtoBuscar**) este requiere de las librer  as **ActionEvent** y **ActionListener**.

```

public ConAU_BuscarUsuario(AU_BuscarUsuario VerBusUsuario) {
    this.VerBusUsuario = VerBusUsuario;
    this.VerBusUsuario.BtoBuscar.addActionListener(this);
}

```

**M  todo Inicio:** Define el tama  o y ubicaci  n en el panel en el que se aplicara el controlador

```

public void Inicio() {
    VerBusUsuario.setLocation(0, 0);
    VerBusUsuario.setSize(0, 0);

}

```

**Método actionPerformed:** Captura los datos enviados por el visual **AU\_BuscarUsuario** usando Métodos **GetText**, se asegura que el usuario haya ingresado los datos en alguno de los campos de texto, llama al método de servicio **SearchUser** y crea un nuevo visual **AU\_InfoUser**.

```

@Override
public void actionPerformed(ActionEvent e) {
    String NomUsuario = VerBusUsuario.TxtSearchName.getText();
    String ApeUsuario = VerBusUsuario.TxtSearchApe.getText();
    String DocUsuario = VerBusUsuario.TxtSearchDoc.getText();
    if ((NomUsuario.isEmpty()) && (ApeUsuario.isEmpty()) && (DocUsuario.isEmpty())){
        VerBusUsuario.errorSearch.setText("Ingrese un dato en alguno de los espacios");
        return;
    }
    findUser = Servicio.searchUser(NomUsuario, ApeUsuario, DocUsuario);

    //ejecuta el visual de AU_InfoUsuario al ser el boton que lo muestra
    // a su vez tambien ejecuta el controlador para mostrar los datos del usuario
    // buscado

    AU_InfoUsuario ViewInfoUser = new AU_InfoUsuario();
    ConAU_InfoUsuario ConInUser = new ConAU_InfoUsuario(ViewInfoUser, findUser);
    ConInUser.Inicio();

    VerBusUsuario.PanBuscarUsu.removeAll();
    VerBusUsuario.PanBuscarUsu.add(ViewInfoUser);
    VerBusUsuario.PanBuscarUsu.revalidate();
    VerBusUsuario.PanBuscarUsu.repaint();
}

```

### Clase ConAU\_CrearUsuario

**Método ConAU\_CrearUsuario:** Crea un objeto del tipo **Au\_CrearUsuario** en el cual se ejecutarán los demás métodos de la clase, esta requiere que se defina como se ejecuta el controlador (en este caso mediante el **BtoAñadirUsuario**) éste requiere de las librerías **ActionEvent** y **ActionListener**

```

public ConAU_CrearUsuario(AU_CrearUsuario crearUser){
    this.crearUser = crearUser;
    this.crearUser.BtnAñadirUsuario.addActionListener((ActionListener) this);
}

```

**Método Inicio:** Define el tamaño y ubicación en el panel en el que se aplicará el controlador

```
public void Inicio() {
    crearUser.setLocation(0, 0);
    crearUser.setSize(0, 0);
}
```

**Método actionPerformed:** Captura los datos enviados por el visual **AU\_CrearUsuario** usando Métodos **GetText** y **GetItem**, se asegura que el usuario haya llenado todos los datos de los campos de texto, llama al método de servicio **CreateUser** y genera un **setText** en un label para mostrar la confirmación de que el usuario se creó correctamente.

```
//Ios envia a el metodo createUser en el modelo Service
@Override
public void actionPerformed(ActionEvent e) {
    String name = crearUser.TxtNewUserName.getText();
    String last_name = crearUser.TxtNewUserApe.getText();
    String document_number = crearUser.TxtNewUserDoc.getText();
    String email = crearUser.TxtNewUserCorreo.getText();
    String vinculation = crearUser.BoxNewUserVin.getSelectedItem().toString();
    String faculty = crearUser.BoxNewUserFaculty.getSelectedItem().toString();
    String career = crearUser.TxtNewUserCareer.getText().toLowerCase();
    crearUser.ErrorNewUser.setForeground(Color.red);

    if (name.isEmpty() | last_name.isEmpty() | document_number.isEmpty() | email.isEmpty()) {
        crearUser.ErrorNewUser.setText("Rellene todos los espacios");
        return;
    }

    Servicio.createUser(name, last_name, document_number, email, vinculation, faculty, career);
    crearUser.ErrorNewUser.setForeground(Color.GREEN);
    crearUser.ErrorNewUser.setText("Usuario Creado Correctamente");
    crearUser.TxtNewUserName.setText("");
    crearUser.TxtNewUserApe.setText("");
    crearUser.TxtNewUserDoc.setText("");
    crearUser.TxtNewUserCorreo.setText("");
    crearUser.TxtNewUserCareer.setText("");
}
```

### Clase ConAU\_InfoUsuario

**Método ConAU\_InfoUsuario:** Crea un objeto del tipo **Au\_InfoUsuario** en el cual se ejecutarán los demás métodos de la clase, esta requiere que se defina como se ejecuta el controlador (en este caso mediante el Una vez se ejecute **ConAU\_BuscarUsuario**), se asegura que el usuario encontrado exista y finalmente se imprimen los datos del usuario encontrado.

```
public ConAU_InfoUsuario(AU_InfoUsuario ViewInfoUser, Usuario findUser){  
    this.ViewInfoUser = ViewInfoUser;  
  
    if (findUser.getId().isEmpty()){  
        ViewInfoUser.ErrorUserNotFound.setText("No se encontro ningun usuario que coincidia con los datos ingresados");  
        return;  
    }  
  
    ViewInfoUser.ShowUserName.setText(findUser.getName() + " " + findUser.getLast_name());  
    Integer docNum = findUser.getDocument_number();  
    ViewInfoUser.ShowUserDoc.setText( docNum.toString());  
    ViewInfoUser.ShowUserOtherInfo.setText(findUser.getEmail());  
}  
  
public void Inicio(){  
    ViewInfoUser.setLocation(0, 0);  
    ViewInfoUser.setSize(0,0);  
}
```

**Método Inicio:** Define el tamaño y ubicación en el panel en el que se aplicará el controlador

```
public void Inicio(){  
    ViewInfoUser.setLocation(0, 0);  
    ViewInfoUser.setSize(0,0);  
}
```

## Package Controlador.Estadisticas

### Clase ControladorEstadisticas

**Método Constructor ControladorEstadisticas:** Crea un objeto del tipo **VistaEstadisticas** en el cual se ejecutarán los demás métodos de la clase. Los demás métodos que se ejecutan requieren que se defina cómo se ejecuta el controlador (en este caso mediante el **buttonGenerarGrafico** y **comboBoxSeleccionTipoGrafico**). Este método requiere de las librerías **ActionEvent** y **ComboBoxModel**.

```

public ControladorEstadisticas(VistaEstadisticas vistaEstadisticas) {
    this.vistaEstadisticas = vistaEstadisticas;
    this.currentSalas = Servicio.getRooms();

    String[] keySalas = new String[this.currentSalas.length];
    Integer contador = 0;
    for (Sala sala : this.currentSalas) {
        keySalas[contador] = sala.getName();
        contador++;
        System.out.println("-----");
        System.out.println(Arrays.toString(keySalas));
    }

    ComboBoxModel cbm = new DefaultComboBoxModel(items: keySalas);
    vistaEstadisticas.comboBoxSeleccionSala.setModel(aModel: cbm);

    this.vistaEstadisticas.buttonGenerarGrafico.addActionListener((java.awt.event.ActionEvent evt) -> {
        buttonCreateChartActionPerformed(e: evt);
    });
    this.vistaEstadisticas.comboBoxSeleccionTipoGrafico.addActionListener((java.awt.event.ActionEvent evt) -> {
        comboBoxChangeTypeOFChartActionPerformed(e: evt);
    });
}

```

**Método ObtenerSalaSeleccionada:** Obtiene la sala seleccionada basada en el nombre proporcionado. Este método compara el nombre de la sala seleccionada con las salas disponibles y devuelve el ID correspondiente.

```

private Integer obtenerSalaSeleccionada(String selectedSala) {
    for (Sala sala : currentSalas) {
        if (sala.getName().equals(anObject: selectedSala)) {
            return sala.getId();
        }
    }
    return null;
}

```

**Método ConfigurarVistaGrafico:** Define el tamaño y ubicación en el panel en el que se aplicará el controlador.

```

private void configurarVistaGrafico(VistaGrafico VG, VistaGraficoFacultades VGF) {
    VG.setSize(width: 0, height: 0);
    VG.setLocation(x: 0, y: 0);
    VG.setSize(width: 0, height: 0);
    VG.setLocation(x: 0, y: 0);

    vistaEstadisticas.panelSolicitaRango.removeAll();
    vistaEstadisticas.panelSolicitaRango.add(comp: VG);
    vistaEstadisticas.panelSolicitaRango.revalidate();
    vistaEstadisticas.panelSolicitaRango.repaint();

    vistaEstadisticas.panelSolicitaSala.removeAll();
    vistaEstadisticas.panelSolicitaSala.add(comp: VGF);
    vistaEstadisticas.panelSolicitaSala.revalidate();
    vistaEstadisticas.panelSolicitaSala.repaint();
}

```

**Método ButtonCreateChartActionPerformed:** Maneja la acción del botón **buttonGenerarGrafico** para generar el gráfico correspondiente a la selección del usuario. Este método obtiene las fechas, sala y tipo de visualización seleccionados, y luego llama a la configuración del gráfico.

```
public void buttonCreateChartActionPerformed(ActionEvent e) {
    System.out.println("Se presionó el botón");
    Date start_time = vistaEstadisticas.fechaInicialDateChooser.getDate();
    Date finish_time = vistaEstadisticas.fechaFinalDateChooser.getDate();
    String selectedSala = vistaEstadisticas.comboBoxSelecciónSala.getSelectedItem().toString();
    Integer sala_id = obtenerSalasSeleccionadas(selectedSala);
    String selectedVisualization = vistaEstadisticas.comboBoxSelecciónVisualización.getSelectedItem().toString();
    VistaGrafico VG = new VistaGrafico();
    VistaGraficoFacultades VGF = new VistaGraficoFacultades();

    switch (selectedVisualization) {
        case "Año" -> {
            controladorGraficoAño = new ControladorGraficoAño(vistaGraficoMeses: VG, start_time, finish_time, sala_id);
            controladorGraficoFacultades = new ControladorGraficoFacultades(vistaGraficoFacultades: VGF, start_time, finish_time, sala_id);
            configurarVistaGrafico(VG, VGF);
        }
        case "Meses" -> {
            controladorGraficoMeses = new ControladorGraficoMeses(vistaGraficoMeses: VG, start_time, finish_time, sala_id);
            controladorGraficoFacultades = new ControladorGraficoFacultades(vistaGraficoFacultades: VGF, start_time, finish_time, sala_id);
            configurarVistaGrafico(VG, VGF);
        }
        case "Días del Año" -> {
            controladorGraficoDíasDelAño = new ControladorGraficoDíasDelAño(vistaGraficoMeses: VG, start_time, finish_time, sala_id);
            controladorGraficoFacultades = new ControladorGraficoFacultades(vistaGraficoFacultades: VGF, start_time, finish_time, sala_id);
            configurarVistaGrafico(VG, VGF);
        }
        default -> {}
    }
}
```

**Método ComboBoxChangeTypeOfChartActionPerformed:** Maneja la acción del **comboBoxSeleccionTipoGrafico** para cambiar el tipo de gráfico según la selección del usuario. Este método es similar al anterior, pero permite cambiar el tipo de gráfico sin necesidad de generar uno nuevo desde cero.

```
public void comboBoxChangeTypeOfChartActionPerformed(ActionEvent e) {
    System.out.println("Se presionó el botón para cambiar de gráfico");
    Date start_time = vistaEstadisticas.fechaInicialDateChooser.getDate();
    Date finish_time = vistaEstadisticas.fechaFinalDateChooser.getDate();
    String selectedSala = vistaEstadisticas.comboBoxSelecciónSala.getSelectedItem().toString();
    Integer sala_id = obtenerSalasSeleccionadas(selectedSala);
    String selectedVisualization = vistaEstadisticas.comboBoxSelecciónVisualización.getSelectedItem().toString();
    String selectedChartVisualization = vistaEstadisticas.comboBoxSelecciónTipoGrafico.getSelectedItem().toString();

    VistaGrafico VG = new VistaGrafico();
    VistaGraficoFacultades VGF = new VistaGraficoFacultades();

    switch (selectedVisualization) {
        case "Año" -> {
            controladorGraficoAño = new ControladorGraficoAño(vistaGraficoMeses: VG, start_time, finish_time, sala_id, tipoDeGrafico: selectedChartVisualization);
            controladorGraficoFacultades = new ControladorGraficoFacultades(vistaGraficoFacultades: VGF, start_time, finish_time, sala_id);
            configurarVistaGrafico(VG, VGF);
        }
        case "Meses" -> {
            controladorGraficoMeses = new ControladorGraficoMeses(vistaGraficoMeses: VG, start_time, finish_time, sala_id, tipoDeGrafico: selectedChartVisualization);
            controladorGraficoFacultades = new ControladorGraficoFacultades(vistaGraficoFacultades: VGF, start_time, finish_time, sala_id);
            configurarVistaGrafico(VG, VGF);
        }
        case "Días del Año" -> {
            controladorGraficoDíasDelAño = new ControladorGraficoDíasDelAño(vistaGraficoMeses: VG, start_time, finish_time, sala_id, tipoDeGrafico: selectedChartVisualization);
            controladorGraficoFacultades = new ControladorGraficoFacultades(vistaGraficoFacultades: VGF, start_time, finish_time, sala_id);
            configurarVistaGrafico(VG, VGF);
        }
        default -> {}
    }
}
```

## Clase ControladorGraficoDiasDelAño

**Método Constructor ControladorGraficoDiasDelAño :** Este constructor inicializa un objeto ControladorGraficoDiasDelAño utilizando la vista VistaGrafico, las fechas de inicio y fin, y el ID de la sala. Al inicializarse, el constructor llama al método showBarChartDaysOfYear() para mostrar un gráfico de barras con los datos correspondientes a los días del año.

```
public ControladorGraficoDiasDelAño(VistaGrafico vistaGraficoMeses, Date start_time, Date finish_time, Integer sala_id) {
    this.vistaGraficoMeses = vistaGraficoMeses;
    this.start_time = start_time;
    this.finish_time = finish_time;
    this.sala_id = sala_id;

    try {
        showBarChartDaysOfYear();
    } catch (ParseException ex) {
        Logger.getLogger(ControladorGraficoDiasDelAño.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

**Método Constructor ControladorGraficoDiasDelAño :** Este constructor adicional permite seleccionar el tipo de gráfico a mostrar (barras o líneas). Dependiendo de la selección, se llama a showBarChartDaysOfYear() o showLinechartDaysOfYear() para generar el gráfico correspondiente.

```
public ControladorGraficoDiasDelAño(VistaGrafico vistaGraficoMeses, Date start_time, Date finish_time, Integer sala_id, String tipoDeGrafico) {
    this.vistaGraficoMeses = vistaGraficoMeses;
    this.start_time = start_time;
    this.finish_time = finish_time;
    this.sala_id = sala_id;
    this.tipoDeGrafico = tipoDeGrafico;

    if (this.tipoDeGrafico.equals("Gráfico de Barras")){
        try {
            showBarChartDaysOfYear();
        } catch (ParseException ex) {
            Logger.getLogger(ControladorGraficoMeses.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    if (this.tipoDeGrafico.equals("Gráfico de Líneas")){
        try {
            showLinechartDaysOfYear();
        } catch (ParseException ex) {
            Logger.getLogger(ControladorGraficoMeses.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

**Método showBarChartDaysOfYear:** Este método genera un gráfico de barras basado en los datos de sesiones por día del año. Los datos se obtienen a través del servicio Servicio.getDaysOfYear, que devuelve un LinkedHashMap con los días del año y el número de sesiones correspondientes. El gráfico se configura y se muestra en la vista VistaGrafico.

```

public JFreeChart showBarChartDaysOfYear() throws ParseException{
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    LinkedHashMap<String, Integer> monthData = Servicio.getDaysOfYear(start_time, finish_time, sala_id);

    for(Map.Entry<String, Integer> month : monthData.entrySet()){
        String key = month.getKey();
        Integer value = month.getValue();

        // Imprimir la clave y el valor
        System.out.println("Clave: " + key + ", Valor: " + value);
        dataset.setValue(value, rowKey: "Fecha:", columnKey: key);
    }

    JFreeChart chart = ChartFactory.createBarChart(title: "Sesiones/Dia del Año", categoryAxisLabel: "Días del Año", valueAxisLabel: "Sesiones",
        dataset, orientation: PlotOrientation.VERTICAL, legend: false, tooltips: true, urls: false);

    CategoryPlot categoryPlot = chart getCategoryPlot();
    //categoryPlot.setRangeGridlinePaint(Color.BLUE);
    categoryPlot.setBackgroundPaint(paint: Color.WHITE);
    BarRenderer renderer = (BarRenderer) categoryPlot.getRenderer();
    Color clr3 = new Color(r: 204, g: 0, b: 51);
    renderer.setSeriesPaint(series: 0, paint: clr3);
    ChartPanel barpChartPanel = new ChartPanel(chart);
    vistaGraficoMeses.panelGraficoMes.removeAll();
    vistaGraficoMeses.panelGraficoMes.add(comp: barpChartPanel, constraints: BorderLayout.CENTER);
    vistaGraficoMeses.panelGraficoMes.validate();

    return chart;
}

```

**Método showLinechartDaysOfYear:** Este método genera un gráfico de líneas basado en los datos de sesiones por día del año. Al igual que en showBarChartDaysOfYear, los datos se obtienen a través del servicio Servicio.getDaysOfYear. El gráfico de líneas se configura y se muestra en la vista VistaGrafico.

```

public JFreeChart showLinechartDaysOfYear() throws ParseException{
    DefaultCategorydataset dataset = new DefaultCategorydataset();
    LinkedHashMap<String, Integer> monthData = Servicio.getDaysOfYear(start_time, finish_time, sala_id);

    for(Map.Entry<String, Integer> month : monthData.entrySet()){
        String key = month.getKey();
        Integer value = month.getValue();

        // Imprimir la clave y el valor
        System.out.println("Clave: " + key + ", Valor: " + value);
        dataset.setValue(value, rowKey: "Amount", columnKey: key);
    }

    //create chart
    JFreeChart linechart = ChartFactory.createLineChart(title: "Sesiones/Dia del Año", categoryAxisLabel: "Días del Año", valueAxisLabel: "Sesiones",
        dataset, orientation: PlotOrientation.VERTICAL, legend: false, tooltips: true, urls: false);

    //create plot object
    CategoryPlot lineCategoryPlot = linechart getCategoryPlot();
    lineCategoryPlot.setBackgroundPaint(paint: Color.white);

    //create render object to change the modify the line properties like color
    LineAndShapeRenderer lineRenderer = (LineAndShapeRenderer) lineCategoryPlot.getRenderer();
    Color lineChartColor = new Color(r: 204, g: 0, b: 51);
    lineRenderer.setSeriesPaint(series: 0, paint: lineChartColor);

    //create chart panel to display chart(graph)
    ChartPanel lineChartPanel = new ChartPanel(chart: linechart);
    vistaGraficoMeses.panelGraficoMes.removeAll();
    vistaGraficoMeses.panelGraficoMes.add(comp: lineChartPanel, constraints: BorderLayout.CENTER);
    vistaGraficoMeses.panelGraficoMes.validate();

    return linechart;
}

```

## Clase ControladorGraficoAño

**Método Constructor ControladorGraficoAño:** Este constructor inicializa un objeto ControladorGraficoAño utilizando la vista VistaGrafico, las fechas de inicio y fin, y el ID de la sala. Al inicializarse, el constructor llama al método showBarChartYear() para mostrar un gráfico de barras con los datos correspondientes al año.

```

public ControladorGraficoAño(VistaGrafico vistaGraficoMeses, Date start_time, Date finish_time, Integer sala_id) {
    this.vistaGraficoMeses = vistaGraficoMeses;
    this.start_time = start_time;
    this.finish_time = finish_time;
    this.sala_id = sala_id;

    try {
        showBarChartYear();
    } catch (ParseException ex) {
        Logger.getLogger(name:ControladorGraficoAño.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
    }
}
}

```

**Método Constructor ControladorGraficoAño:** Este constructor adicional permite seleccionar el tipo de gráfico a mostrar (barras o líneas). Dependiendo de la selección, se llama a showBarChartYear() o showLinechartYear() para generar el gráfico correspondiente.

```

public ControladorGraficoAño(VistaGrafico vistaGraficoMeses, Date start_time, Date finish_time, Integer sala_id, String tipoDeGrafico) {
    this.vistaGraficoMeses = vistaGraficoMeses;
    this.start_time = start_time;
    this.finish_time = finish_time;
    this.sala_id = sala_id;
    this.tipoDeGrafico = tipoDeGrafico;

    if (this.tipoDeGrafico.equals(anObject:"Gráfico de Barras")){
        try {
            showBarChartYear();
        } catch (ParseException ex) {
            Logger.getLogger(name:ControladorGraficoMeses.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        }
    }

    if (this.tipoDeGrafico.equals(anObject:"Gráfico de Líneas")){
        try {
            showLinechartYear();
        } catch (ParseException ex) {
            Logger.getLogger(name:ControladorGraficoMeses.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        }
    }
}
}

```

**Método showBarChartYear:** Este método genera un gráfico de barras basado en los datos de sesiones por año. Los datos se obtienen a través del servicio Servicio.getYear, que devuelve un LinkedHashMap con los años y el número de sesiones correspondientes. El gráfico se configura y se muestra en la vista VistaGrafico.

```

public JFreeChart showBarChartYear() throws ParseException{
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    LinkedHashMap<String, Integer> monthData = Servicio.getYear(start_time, finish_time, sala_id);

    for(Map.Entry<String, Integer> month : monthData.entrySet()){
        String key = month.getKey();
        Integer value = month.getValue();

        // Imprimir la clave y el valor
        System.out.println("Clave: " + key + ", Valor: " + value);
        dataset.setValue(value, rowKey: "Amount", columnKey: key);
    }

    JFreeChart chart = ChartFactory.createBarChart(title: "Sesiones/Año", categoryAxisLabel: "Años", valueAxisLabel: "Sesiones",
        dataset, orientation:PlotOrientation.VERTICAL, legend: false, tooltips: true, urls: false);

    CategoryPlot categoryPlot = chart.getCategoryPlot();
    //categoryPlot.setRangeGridlinePaint(Color.BLUE);
    categoryPlot.setBackgroundPaint(paint: Color.WHITE);
    BarRenderer renderer = (BarRenderer) categoryPlot.getRenderer();
    Color clr3 = new Color(r: 204,g: 0,b: 51);
    renderer.setSeriesPaint(series: 0, paint: clr3);
    ChartPanel barpChartPanel = new ChartPanel(chart);
    vistaGraficoMeses.panelGraficoMes.removeAll();
    vistaGraficoMeses.panelGraficoMes.add(comp: barpChartPanel, constraints:BorderLayout.CENTER);
    vistaGraficoMeses.panelGraficoMes.validate();

    return chart;
}
}

```

**Método showLinechartYear:** Este método genera un gráfico de líneas basado en los datos de sesiones por año. Al igual que en showBarChartYear, los datos se obtienen a través del servicio Servicio.getYear. El gráfico de líneas se configura y se muestra en la vista VistaGrafico.

```

public JFreeChart showLinechartYear() throws ParseException{
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    LinkedHashMap<String, Integer> monthData = Servicio.getYear(start_time, finish_time, sala_id);

    for(Map.Entry<String, Integer> month : monthData.entrySet()){
        String key = month.getKey();
        Integer value = month.getValue();

        // Imprimir la clave y el valor
        System.out.println("Clave: " + key + ", Valor: " + value);
        dataset.setValue(value, rowKey: "Amount", columnKey: key);
    }

    //create chart
    JFreeChart linechart = ChartFactory.createLineChart(title: "Sesiones/Año", categoryAxisLabel: "Meses", valueAxisLabel:"Sesiones",
        dataset, orientation:PlotOrientation.VERTICAL, legend: false, tooltips: true, urls: false);

    //create plot object
    CategoryPlot lineCategoryPlot = linechart.getCategoryPlot();
    lineCategoryPlot.setBackgroundPaint(paint: Color.white);

    //create render object to change the modify the line properties like color
    LineAndShapeRenderer lineRenderer = (LineAndShapeRenderer) lineCategoryPlot.getRenderer();
    Color lineChartColor = new Color(r: 204, g: 0, b: 51);
    lineRenderer.setSeriesPaint(series: 0, paint: lineChartColor);

    //create chart panel to display chart(graph)
    ChartPanel lineChartPanel = new ChartPanel(chart: linechart);
    vistaGraficoMeses.panelGraficoMes.removeAll();
    vistaGraficoMeses.panelGraficoMes.add(comp: lineChartPanel, constraints: BorderLayout.CENTER);
    vistaGraficoMeses.panelGraficoMes.validate();

    return linechart;
}

```

## Clase ControladorGraficoMeses

**Método Constructor ControladorGraficoMeses:** Este constructor inicializa un objeto ControladorGraficoMeses utilizando la vista VistaGrafico, las fechas de inicio y fin, y el ID de la sala. Al inicializarse, el constructor llama al método showBarChartMonth() para mostrar un gráfico de barras con los datos correspondientes al mes.

```

public ControladorGraficoMeses(VistaGrafico vistaGraficoMeses, Date start_time, Date finish_time, Integer sala_id) {
    this.vistaGraficoMeses = vistaGraficoMeses;
    this.start_time = start_time;
    this.finish_time = finish_time;
    this.sala_id = sala_id;

    try {
        showBarChartMonth();
    } catch (ParseException ex) {
        Logger.getLogger(ControladorGraficoMeses.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        System.out.println("no hay grafico para ti");
    }
}

```

**Método Constructor ControladorGraficoMeses :** Este constructor adicional permite seleccionar el tipo de gráfico a mostrar (barras o líneas). Dependiendo de la selección, se

llama a showBarChartMonth() o showLinechartMonth() para generar el gráfico correspondiente.

```
public ControladorGraficoMeses(VistaGrafico vistaGraficoMeses, Date start_time, Date finish_time, Integer sala_id, String tipoDeGrafico) {
    this.vistaGraficoMeses = vistaGraficoMeses;
    this.start_time = start_time;
    this.finish_time = finish_time;
    this.sala_id = sala_id;
    this.tipoDeGrafico = tipoDeGrafico;

    if (this.tipoDeGrafico.equals(anObject:"Gráfico de Barras")){
        try {
            showBarChartMonth();
        } catch (ParseException ex) {
            Logger.getLogger(ControladorGraficoMeses.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        }
    }

    if (this.tipoDeGrafico.equals(anObject:"Gráfico de Líneas")){
        try {
            showLinechartMonth();
        } catch (ParseException ex) {
            Logger.getLogger(ControladorGraficoMeses.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        }
    }
}
```

**Método showBarChartMonth:** Este método genera un gráfico de barras basado en los datos de sesiones por mes. Los datos se obtienen a través del servicio Servicio.getMonth, que devuelve un LinkedHashMap con los meses y el número de sesiones correspondientes. El gráfico se configura y se muestra en la vista VistaGrafico.

```
public JFreeChart showBarChartMonth() throws ParseException{
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    LinkedHashMap<String, Integer> monthData = Servicio.getMonth(start_time, finish_time, sala_id);

    for(Map.Entry<String, Integer> month : monthData.entrySet()){
        String key = month.getKey();
        Integer value = month.getValue();

        // Imprimir la clave y el valor
        System.out.println("Clave: " + key + ", Valor: " + value);
        dataset.setValue(value, rowKey: "Amount", columnKey: key);
    }

    JFreeChart chart = ChartFactory.createBarChart(title: "Sesiones/Mes", categoryAxisLabel: "Meses", valueAxisLabel: "Sesiones",
        dataset, orientation:PlotOrientation.VERTICAL, legend: false, tooltips: true, urls: false);

    CategoryPlot categoryPlot = chart.getCategoryPlot();
    //categoryPlot.setRangeGridlinePaint(Color.BLUE);
    categoryPlot.setBackgroundPaint(paint: Color.WHITE);
    BarRenderer renderer = (BarRenderer) categoryPlot.getRenderer();
    Color clr3 = new Color(r: 204, g: 0, b: 51);
    renderer.setSeriesPaint(series: 0, paint: clr3);
    ChartPanel barpChartPanel = new ChartPanel(chart);
    vistaGraficoMeses.panelGraficoMes.removeAll();
    vistaGraficoMeses.panelGraficoMes.add(comp:barpChartPanel, constraints:BorderLayout.CENTER);
    vistaGraficoMeses.panelGraficoMes.validate();

    return chart;
}
```

**Método showLinechartMonth:** Este método genera un gráfico de líneas basado en los datos de sesiones por mes. Al igual que en showBarChartMonth, los datos se obtienen a través del servicio Servicio.getMonth. El gráfico de líneas se configura y se muestra en la vista VistaGrafico.

```

public JFreeChart showLinechartMonth() throws ParseException{
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    LinkedHashMap<String, Integer> monthData = Servicio.getMonth(start_time, finish_time, sala_id);

    for(Map.Entry<String, Integer> month : monthData.entrySet()){
        String key = month.getKey();
        Integer value = month.getValue();

        // Imprimir la clave y el valor
        System.out.println("Clave: " + key + ", Valor: " + value);
        dataset.setValue(value, rowKey: "Amount", columnKey: key);
    }

    //create chart
    JFreeChart linechart = ChartFactory.createLineChart(title: "Sesiones/Mes", categoryAxisLabel: "Meses", valueAxisLabel:"Sesiones",
        |dataset, orientation:PlotOrientation.VERTICAL, legend: false, tooltips: true, urls: false);

    //create plot object
    CategoryPlot lineCategoryPlot = linechart.getCategoryPlot();
    lineCategoryPlot.setBackgroundPaint(paint: Color.white);

    //create render object to change the modify the line properties like color
    LineAndShapeRenderer lineRenderer = (LineAndShapeRenderer) lineCategoryPlot.getRenderer();
    Color lineChartColor = new Color(r: 204, g: 0, b: 51);
    lineRenderer.setSeriesPaint(series: 0, paint: lineChartColor);

    //create chart panel to display chart(graph)
    ChartPanel lineChartPanel = new ChartPanel(chart: linechart);
    vistaGraficoMeses.panelGraficoMes.removeAll();
    vistaGraficoMeses.panelGraficoMes.add(comp: lineChartPanel, constraints: BorderLayout.CENTER);
    vistaGraficoMeses.panelGraficoMes.validate();

    return linechart;
}

```

## Clase ControladorGraficoFacultades

**Método Constructor ControladorGraficoFacultades :** Este constructor inicializa un objeto ControladorGraficoFacultades utilizando la vista VistaGraficoFacultades, las fechas de inicio y fin, y el ID de la sala. Al inicializarse, el constructor llama al método showPieChart() para mostrar un gráfico de pastel con los datos correspondientes a las facultades.

```

public ControladorGraficoFacultades(VistaGraficoFacultades vistaGraficoFacultades, Date start_time, Date finish_time, Integer sala_id) {
    this.vistaGraficoFacultades = vistaGraficoFacultades;
    this.start_time = start_time;
    this.finish_time = finish_time;
    this.sala_id = sala_id;
    try {
        showPieChart();
    } catch (ParseException ex) {
        Logger.getLogger(ControladorGraficoFacultades.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
    }
}

```

**Método showPieChart:** Este método genera un gráfico de pastel basado en los datos de sesiones por facultad. Los datos se obtienen a través del servicio Servicio.getFaculty, que devuelve un LinkedHashMap con las facultades y el número de sesiones correspondientes. El gráfico se configura y se muestra en la vista VistaGraficoFacultades.

```

public JFreeChart showPieChart() throws ParseException{
    //create dataset
    DefaultPieDataset barDataset = new DefaultPieDataset();
    LinkedHashMap<String, Integer> facultyData = Servicio.getFaculty(start_time, finish_time, sala_id);

    for(Map.Entry<String, Integer> month : facultyData.entrySet()){
        String key = month.getKey();
        Integer value = month.getValue();

        // Imprimir la clave y el valor
        System.out.println("Clave: " + key + ", Valor: " + value);
        barDataset.setValue(key, value);
    }

    //create chart
    JFreeChart piechart = ChartFactory.createPieChart(title: "Sesiones/Facultad", dataset:barDataset, legend: false, tooltips: true, urls: false);

    PiePlot piePlot =(PiePlot) piechart.getPlot();

    //changing pie chart blocks colors
    piePlot.setSectionPaint(key: "iPhone 5s", new Color(r: 255,g: 255,b: 102));
    piePlot.setSectionPaint(key: "SamSung Grand", new Color(r: 102,g: 255,b: 102));
    piePlot.setSectionPaint(key: "MotoG", new Color(r: 255,g: 102,b: 153));
    piePlot.setSectionPaint(key: "Nokia Lumia", new Color(r: 0,g: 204,b: 204));

    piePlot.setBackgroundPaint(paint: Color.white);

    //create chartPanel to display chart(graph)
    ChartPanel barChartPanel = new ChartPanel(chart: piechart);
    vistaGraficoFacultades.panelGraficoFacultades.removeAll();
    vistaGraficoFacultades.panelGraficoFacultades.add(comp: barChartPanel, constraints:BorderLayout.CENTER);
    vistaGraficoFacultades.panelGraficoFacultades.validate();
    return piechart;
}

```

## Modelos

### Package Modelo

#### Clase Servicio

**Método iniciarSesion:** Inicia sesión para un usuario utilizando su correo electrónico y contraseña. Dependiendo del rol del usuario, se asigna un objeto de tipo Profesor o Estudiante. Si las credenciales son incorrectas, se retorna un mensaje de error.

```

static public JSONObject iniciarSesion(String email, String password){
    JSONObject data = supabase.auth.LoginWithEmailAndPassword(email, password);
    if (data.has("user")){
        //El login fue correcto
        sessionRole = data.getJSONObject("user").getString("role");
        if (sessionRole.equals("admin")){
            teacher = new Profesor(data);
        }else{
            student = new Estudiante(data);
        }
    }else{
        //El login falló
        HashMap<String, Object> errorMap = new HashMap<>();
        errorMap.put("errorMessage", "Tu usuario o contraseña no son correctos.");
        JSONObject error = new JSONObject(errorMap);

        return error;
    }
    return data;
}

```

**Método updateRooms:** Actualiza y devuelve una lista de las salas (Sala) disponibles en la base de datos.

```
static public Sala[] updateRooms() {
    JSONArray data = supabase.from("salas").select("*");
    sessionSalas = new Sala[data.length()];
    System.out.println(data.toString());
    for (int i = 0; i < data.length(); i++) {
        sessionSalas[i] = new Sala(data.getJSONObject(i));
    }
    return sessionSalas;
}
```

**Método getRooms:** Devuelve una lista de las salas (Sala) disponibles. Si no hay salas en la sesión actual, las actualiza primero.

```
static public Sala[] getRooms() {
    if (sessionSalas.length == 0) {
        sessionSalas = updateRooms();
        return sessionSalas;
    }
    return sessionSalas;
}
```

**Método getComputers:** Devuelve una lista de computadoras (Computador) en función del ID de la sala.

```
static public Computador[] getComputers(Integer sala_id) {
    JSONArray response = supabase.from("computers").select(eq("*", "sala_id", sala_id));
    computers = new Computador[response.length()];
    for (int i = 0; i < response.length(); i++) {
        computers[i] = new Computador(response.getJSONObject(i));
    }
    return computers;
}
```

**Método createComputers:** Inserta una nueva lista de computadoras en la base de datos.

```
public static void createComputers(JSONArray computers_array) {
    supabase.from("computers").insert(computers_array);
}
```

**Método getActiveSessions:** Devuelve una lista de sesiones activas (Sesion) en una sala específica.

```
static public Sesion[] getActiveSessions(Sala dataSala){
    JSONArray data = supabase.from("active_sessions").selecteq("*", "users(*)", "computers(*)", "sala_id", dataSala.getId());
    active_sessions = new Sesion[data.length()];
    System.out.println(data.toString());
    for (int i = 0; i < data.length(); i++) {
        Computador session_computer = new Computador(data.getJSONObject(i).getJSONObject("computers"));
        Usuario session_user = new Usuario(data.getJSONObject(i).getJSONObject("users"));
        active_sessions[i] = new Sesion(data.getJSONObject(i), dataSala, session_user, session_computer);
    }
    return active_sessions;
}
```

**Método updateActiveSessions:** Actualiza la lista de sesiones activas en una sala específica.

```
static public void updateActiveSessions(int sala_id){
    JSONArray data = supabase.from("active_sessions").selecteq("*", "users(*)", "computers(*)", "sala_id", sala_id);
    active_sessions = new Sesion[data.length()];

    for (int i = 0; i < data.length(); i++) {
        Computador session_computer = new Computador(data.getJSONObject(i).getJSONObject("computers"));
        Usuario session_user = new Usuario(data.getJSONObject(i).getJSONObject("users"));
        active_sessions[i] = new Sesion(data.getJSONObject(i), currentManagementSalas, session_user, session_computer);
    }
}
```

**Método addActiveSession:** Añade una nueva sesión activa a la sala utilizando el número de documento del usuario, el ID de la sala, y el ID de la computadora.

```
static public void addActiveSession(String docNumber, Integer sala_id, String computer_id){
    JSONArray findUser = supabase.from("users").selecteq("*", "document_number", docNumber);
    if (findUser.length() == 0) {
        System.out.println("No se encontró el usuario");
        return;
    }
    Usuario findUsuario = new Usuario(findUser.getJSONObject(0));
    JSONArray body = new JSONArray("{\"sala_id\":\"" + sala_id + "\", \"computer_id\":\"" + computer_id + "\", \"user_id\":\"" + findUsuario.getId() + "\"}");
    System.out.println(body.toString());
    supabase.from("active_sessions").insert(body);
    updateActiveSessions(sala_id);
}
```

**Método deleteActiveSession:** Elimina una sesión activa de la sala basándose en el UUID de la computadora y actualiza las sesiones activas.

```
static public void deleteActiveSession(String uuid, int sala_id){
    System.out.println(uuid);
    JSONArray response = supabase.from("active_sessions").delete(new Filter("computer_id", "eq", uuid));
    System.out.println(response.toString());
    updateActiveSessions(sala_id);
}
```

**Método createRoom:** Crea una nueva sala (Sala) y la inserta en la base de datos.

```
static public Sala createRoom(String name, Integer rows, Integer cols, Map<Integer, int[]> computers_arrays) {
    Sala newSala = new Sala(name, rows, cols, computers_arrays);
    Sala[] salas = new Sala[]{newSala};
    JSONArray body = new JSONArray(salas);
    System.out.println(body);
    JSONArray response = supabase.from("salas").insert(body);
    return new Sala(response.getJSONObject(0));
}
```

**Método getRoom:** Devuelve una sala específica utilizando su ID.

```

public static Sala getRoom(Integer id){
    System.out.println("Buscando una Sala en especial..");
    JSONArray salas = supabase.from("salas").selectEq("*", "id", id);
    JSONObject sala = salas.getJSONObject(0);
    Sala createSala = new Sala(sala);
    return createSala;
}

```

**Método getSessionsByRange:** Devuelve una lista de sesiones (Sesion) filtradas por el ID de la sala, la hora de inicio, y la hora de fin.

```

public static Sesion[] getSessionsByRange(Integer sala_id, String start_time, String finish_time){
    Filter filterorder = new Filter("order", "start_time", "asc");
    Filter filterTipo = new Filter("sala_id", "eq", sala_id);
    Filter filterFechaInicio = new Filter("start_time", "gte", start_time);
    Filter filterFechaFin = new Filter("finish_time", "lte", finish_time);
    Filter[] filters = new Filter[]{filterFechaInicio, filterFechaFin, filterorder, filterTipo};

    JSONArray filteredSessions = supabase.from("sessions").select("*", users("*"), salas("*"), filters);
    System.out.println(filteredSessions);
    Sesion[] arraySessions = new Sesion[filteredSessions.length()];
    System.out.println(filteredSessions.toString());

    for (int i = 0; i < filteredSessions.length(); i++) {
        arraySessions[i] = new Sesion(filteredSessions.getJSONObject(i));
    }

    return arraySessions;
}

```

**Método getUsersBySessionAndRange:** Devuelve una lista de sesiones (Sesion) filtradas por el ID de la sala, la hora de inicio, y la hora de fin utilizando objetos Date.

```

public static Session[] getUsersBySessionAndRange(Integer sala_id, Date start_time, Date finish_time) throws ParseException{
    String finishTimeString = getDateFormatString("yyyy-MM-dd HH:mm:ss", finish_time);
    String startTimeString = getDateFormatString("yyyy-MM-dd HH:mm:ss", start_time);

    Filter filterTipo = new Filter("sala_id", "eq", sala_id);
    Filter filterorder = new Filter("order", "start_time", "asc");
    Filter filterFechaInicio = new Filter("start_time", "gte", URLEncoder.encode(startTimeString, StandardCharsets.UTF_8));
    Filter filterFechaFin = new Filter("finish_time", "lte", URLEncoder.encode(finishTimeString, StandardCharsets.UTF_8));
    Filter[] filters = new Filter[]{filterFechaInicio, filterFechaFin, filterorder, filterTipo};

    JSONArray filteredSessions = supabase.from("sessions").select("*", salas("*"), users("*"), filters);
    System.out.println(filteredSessions);
    Session[] arraySessions = new Session[filteredSessions.length()];
    System.out.println(filteredSessions.toString());

    for (int i = 0; i < filteredSessions.length(); i++) {
        arraySessions[i] = new Session(filteredSessions.getJSONObject(i));
    }

    return arraySessions;
}

```

**Método getDateFormat:** Devuelve un objeto Date formateado a partir de una cadena de texto y un patrón.

```

public static Date getDateFormat(String formatPattern, String date) throws ParseException {
    SimpleDateFormat formatter = new SimpleDateFormat(formatPattern);
    return formatter.parse(date);
}

```

**Método getDateFormatString:** Devuelve una cadena de texto formateada a partir de un objeto Date y un patrón.

```

public static String getDateFormatString(String formatPattern, Date date) throws ParseException {
    SimpleDateFormat formatter = new SimpleDateFormat(formatPattern);
    formatter.setTimeZone(TimeZone.getDefault().getTimeZone("COT"));
    //String formattedDateStr = formatter.format(date);
    return formatter.format(date);
}

```

**Método getDaysOfYear:** Devuelve un LinkedHashMap que contiene los días del año y la cantidad de usuarios que usaron las sesiones en esos días.

```

public static LinkedHashMap<String, Integer> getDaysOfYear(Date start_time, Date finish_time, Integer sala_id) throws ParseException{
    Sesion[] response = getUsersBySessionAndRange(sala_id, start_time, finish_time);

    LinkedHashMap<String, Integer> formatedSession = new LinkedHashMap<>();

    for (Sesion sesion : response) {
        System.out.println("_____");
        Date date = sesion.getHoraEntrada();
        System.out.println(date);
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);

        // Obtenemos el día y el mes
        Integer day = calendar.get(Calendar.DAY_OF_MONTH);
        Integer month = calendar.get(Calendar.MONTH) + 1; // Los meses en Calendar son 0-based

        // Formateamos la fecha como "dd/MM/yyyy"
        String dayStr = String.format("%02d/%02d", day, month);

        if (formatedSession.containsKey(dayStr)) {
            Integer aux = formatedSession.get(dayStr);
            formatedSession.put(dayStr, aux + 1);
        } else {
            System.out.println("No existe aún el día: " + dayStr);
            formatedSession.put(dayStr, 1);
        }
    }

    System.out.println(formatedSession.toString());
    return formatedSession;
}

```

**Método getMonth:** Devuelve un LinkedHashMap que contiene los meses y la cantidad de usuarios que usaron las sesiones en esos meses.

```

public static LinkedHashMap<String, Integer> getMonth(Date start_time, Date finish_time, Integer sala_id) throws ParseException{
    //String month = null;

    Sesión[] response = getUsersBySessionAndRange(sala_id, start_time, finish_time);

    LinkedHashMap <String, Integer> formatedSession = new LinkedHashMap<>();

    for (Sesión sesión : response){
        System.out.println("_____");
        Date date = sesión.getHoraEntrada();
        System.out.println(date);
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        String month = calendar.getDisplayName(Calendar.MONTH, Calendar.LONG, Locale.forLanguageTag("es-ES"));

        if (formatedSession.containsKey(month)){
            Integer aux = formatedSession.get(month);
            formatedSession.put(month, aux + 1);
        }else{
            System.out.println("No existe aún el mes: " + month);
            formatedSession.put(month, 1);
        }
    }

    System.out.println(formatedSession.toString());
    return formatedSession;
}

```

**Método getYear:** Devuelve un LinkedHashMap que contiene los años y la cantidad de usuarios que usaron las sesiones en esos años.

```

public static LinkedHashMap<String, Integer> getYear(Date start_time, Date finish_time, Integer sala_id) throws ParseException{
    //String month = null;

    Sesión[] response = getUsersBySessionAndRange(sala_id, start_time, finish_time);

    LinkedHashMap <String, Integer> formatedSession = new LinkedHashMap<>();

    for (Sesión sesión : response){
        System.out.println("_____");
        Date date = sesión.getHoraEntrada();
        System.out.println(date);
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        Integer year = calendar.get(Calendar.YEAR);
        String yearStr = String.valueOf(year);

        if (formatedSession.containsKey(yearStr)){
            Integer aux = formatedSession.get(yearStr);
            formatedSession.put(yearStr, aux + 1);
        }else{
            System.out.println("No existe aún el año: " + yearStr);
            formatedSession.put(yearStr, 1);
        }
    }

    System.out.println(formatedSession.toString());
    return formatedSession;
}

```

**Método getFaculty:** Devuelve un LinkedHashMap que contiene las facultades y la cantidad de usuarios que pertenecen a cada una.

```

public static LinkedHashMap<String, Integer> getFaculty(Date start_time, Date finish_time, Integer sala_id) throws ParseException{
    Sesión[] response = getUsersBySessionAndRange(sala_id, start_time, finish_time);

    LinkedHashMap <String, Integer> formatedSession = new LinkedHashMap<>();

    for (Sesión session : response){
        String faculty = session.getUsuario().getFaculty();
        //System.out.println(session.getHoraEntrada().toString());
        if (formatedSession.containsKey(faculty)){
            Integer aux = formatedSession.get(faculty);
            formatedSession.put(faculty, aux + 1);
        }else{
            formatedSession.put(faculty, 1);
        }
    }

    System.out.println(formatedSession);

    return formatedSession;
}

```

**Método searchUser:** Busca un usuario en la base de datos basado en el nombre, apellido o número de documento.

```

public static JSONObject searchUser(String NomUsuario, String ApeUsuario, String DocUsuario){
    JSONObject findUser = new JSONObject();

    if (!NomUsuario.isBlank()){
        JSONArray response = supabase.from("users").select("*", new Filter("name", "eq", NomUsuario));
        findUser = response.getJSONObject(0);
        System.out.println(response.toString());
    }

    if (!ApeUsuario.isBlank()){
        JSONArray response = supabase.from("users").select("*", new Filter("last_name", "eq", ApeUsuario));
        findUser = response.getJSONObject(0);
        System.out.println(response.toString());
    }

    if (!DocUsuario.isBlank()){
        JSONArray response = supabase.from("users").select("*", new Filter("document_number", "eq", DocUsuario));
        findUser = response.getJSONObject(0);
        System.out.println(response.toString());
    }

    return findUser;
}

```

**Método createUser:** Crea un nuevo usuario (Usuario) en la base de datos.

```

public static String createUser(String name, String last_name, String document_number, String email, String vinculation, String faculty, String career){
    Usuario[] arraysUsers = new Usuario[1];
    arraysUsers[0] = new Usuario(career, faculty, Integer.valueOf(document_number), name, last_name, email);
    System.out.println("intentando crear usuario");
    JSONArray body = new JSONArray(arraysUsers);
    System.out.println(body.toString());
    JSONArray usuarioCreado = supabase.from("users").insert(body);
    return usuarioCreado.toString();
}

```

## Clase Computador

Es una clase que define las características que tienen un computador

**Método Computador:** Es el método constructor de la clase Computador en la que se define sus características

```
public Computador(JSONObject computer_data) {
    this.id = computer_data.getString("id");
    this.name = computer_data.getString("name");
    this.cont_sala = computer_data.getInt("cont_sala");
    this.estado = "libre";
}
```

**Método Computador:** Constructor que recibe el nombre del computador, el número de la sala (cont\_sala), y el sala\_id, y utiliza estos valores para inicializar los atributos correspondientes.

```
public Computador(String name, Integer cont_sala, Integer sala_id) {
    this.name = name;
    this.cont_sala = cont_sala;
    this.sala_id = sala_id;
}
```

**Método Show:** imprime algunos de los datos del computador con el fin de utilizarlos en otros métodos exteriores

```
public void show() {
    System.out.println("Computador: " + name);
    System.out.println("ID computador" + id.toString());
}
```

**Métodos get:** Estos métodos se encargan de obtener las características de un usuario, una función que es utilizada en otros métodos y controladores para imprimir información u obtenerla

```

public String getId() {
    return id;
}

public String getName() {
    return name;
}

public String getEstado() {
    return estado;
}

public Integer getCont_sala() {
    return cont_sala;
}

```

### Clase Sala

**Método Sala:** Constructor por defecto que inicializa la variable id en -1. No requiere parámetros adicionales.

```

public Sala() {
    this.id = -1;
}

```

**Método Sala:** Constructor que recibe el nombre de la sala, número de filas, columnas y un Map con la información de los computadores. Inicializa estos valores en la clase.

```

public Sala(String name, Integer rows, Integer cols, Map<Integer, int[]> computersData) {
    this.name = name;
    this.rows = rows;
    this.cols = cols;
    this.computers_arrays = computersData;
}

```

**Método Sala:** Constructor que recibe un objeto JSONObject y lo utiliza para inicializar los atributos de la sala, incluyendo la conversión de un JSONObject a un Map<Integer, int[]> para la variable computers\_arrays.

```

public Sala(JSONObject data){
    this.id = data.getInt("id");
    this.name = data.getString("name");
    JSONObject test = data.getJSONObject("computers_arrays");
    Map<Integer, int[]> computers_data = new HashMap<>();

    // Recorrer las claves del JSONObject
    for (String key : test.keySet()) {
        int keyInt = Integer.parseInt(key); // Convertir la clave a Integer

        // Obtener el JSONArray asociado a la clave
        JSONArray jsonArray = test.getJSONArray(key);

        // Convertir el JSONArray a un array de enteros
        int[] values = new int[jsonArray.length()];
        for (int i = 0; i < jsonArray.length(); i++) {
            values[i] = jsonArray.getInt(i);
        }

        // Poner la clave y el array de enteros en el Map
        computers_data.put(keyInt, values);
    }
    this.computers_arrays = computers_data;
    this.rows = data.getInt("rows");
    this.cols = data.getInt("cols");
}

```

**Método Sala:** Constructor que recibe un objeto JSONObject y un JSONArray, y los utiliza para inicializar los atributos name, id y computadores.

```

public Sala(JSONObject data, JSONArray computers_data) {
    System.out.println(data.toString());
    System.out.println(computers_data.toString());

    this.name = (String) data.getString("name");
    this.id = (Integer) data.getInt("id");
    this.computadores = new JSONArray(computers_data);
}

```

**Método createSession:** Crea una nueva sesión con un usuario y un computador específicos, y la retorna.

```

public Sesion createSession(Usuario usuario, Computador computador) {
    Sesion newSession = new Sesion(this, usuario, computador);
    return newSession;
}

```

**Método isAvailable:** Verifica si la sala está disponible revisando si el id es distinto de -1.

```
public Boolean isAvailable () {  
    return this.id != -1;  
}
```

**Método getComputadores:** Retorna el arreglo de computadores en formato JSONArray.

```
public JSONArray getComputadores () {  
    return computadores;  
}
```

**Método getId:** Retorna el id de la sala.

```
public Integer getId () {  
    return id;  
}
```

**Método getName:** Retorna el nombre de la sala.

```
public String getName () {  
    return name;  
}
```

**Método getRows:** Retorna el número de filas de la sala.

```
public Integer getRows () {  
    return rows;  
}
```

**Método getCols:** Retorna el número de columnas de la sala.

```
public Integer getCols () {  
    return cols;  
}
```

**Método getComputers\_arrays:** Retorna el Map que contiene los arrays de computadores.

```
public Map<Integer, int[]> getComputers_arrays () {  
    return computers_arrays;  
}
```

## Clase Sesión

**Método Sesión:** Constructor que recibe una sala, un usuario, y un computador, y utiliza estos parámetros para inicializar los atributos correspondientes. Además, establece la hora de entrada como la fecha actual y el estado de la sesión como "activa".

```
public Sesión(Sala sala, Usuario usuario, Computador computador) {  
    this.usuario = usuario;  
    this.computador = computador;  
    this.sala = sala;  
    this.horaEntrada = new Date();  
    this.estadoSesión = "activa";  
}
```

**Método Sesión:** Constructor que recibe un objeto JSONObject, extrae la información de las fechas de inicio y fin, y las convierte a formato Date. Inicializa los atributos horaEntrada, horaSalida, usuario, y sala utilizando los datos proporcionados en el JSONObject.

**Método Sesión:** Constructor que recibe un objeto JSONObject, una sala, un usuario, y un computador, y utiliza estos parámetros para inicializar los atributos correspondientes. También establece la hora de entrada como la fecha actual y el estado de la sesión como "activa".

```
public Sesión(JSONObject session_data, Sala sala, Usuario usuario, Computador computador) {  
    this.usuario = usuario;  
    this.computador = computador;  
    this.sala = sala;  
    this.horaEntrada = new Date();  
    this.estadoSesión = "activa";  
}
```

**Método getDateFormat:** Método privado que convierte una cadena de texto en un objeto Date según un patrón de formato especificado. Si ocurre un error en la conversión, retorna la fecha actual.

```
private Date getDateFormat(String formatPattern, String date) {  
    SimpleDateFormat formatter = new SimpleDateFormat(formatPattern);  
    try {  
        return formatter.parse(date);  
    } catch (ParseException ex) {  
        return new Date();  
    }  
}
```

**Método getEstadoSesion:** Retorna el estado actual de la sesión.

```
public String getEstadoSesion() {  
    return estadoSesion;  
}
```

**Método getHoraEntrada:** Retorna la hora de entrada de la sesión.

```
public Date getHoraEntrada() {  
    return horaEntrada;  
}
```

**Método getHoraSalida:** Retorna la hora de salida de la sesión.

```
public Date getHoraSalida() {  
    return horaSalida;  
}
```

**Método getComputador:** Retorna el computador asociado a la sesión.

```
public Computador getComputador() {  
    return computador;  
}
```

**Método getUsuario:** Retorna el usuario asociado a la sesión.

```
public Usuario getUsuario() {  
    return usuario;  
}
```

**Método getSala:** Retorna la sala asociada a la sesión.

```

public Sala getsala() {
    return sala;
}

```

## Package Modelo.Users

### Clase Usuario

La clase usuario se encarga de dar las características que tienen un usuario registrado en la base de datos

**Metodo Usuario:** Es el constructor de la clase Usuario

```

public Usuario(String career, String faculty, Integer document_number, String name, String last_name, String email) {
    this.career = career;
    this.faculty = faculty;
    this.document_number = document_number;
    this.name = name;
    this.last_name = last_name;
    this.email = email;
}

```

**Métodos Get:** Estos métodos se encargan de obtener las características de un usuario, una función que es utilizada en otros métodos y controladores para imprimir información u obtenerla

```

public String getEmail() {
    return email;
}

public void show(){
    System.out.println("Nombre: " + this.name);
}

public String getCareer() {
    return career;
}

public String getFaculty() {
    return faculty;
}

public Integer getDocument_number() {
    return document_number;
}

public String getName() {
    return name;
}

public String getLast_name() {
    return last_name;
}

public String getId() {
    return id;
}

```

### Clase Estudiante

La clase usuario define las características que tendrá un estudiante de corresponsabilidad, en este caso hereda todas las características de un abstracUser y agrega una propia

**Método Estudiante:** Es el constructor de la clase Estudiante, define cuáles son sus valores base y se asegura que todo usuario tenga esos datos.

```
public class Estudiante extends AbstractUser{  
    private Integer horasCorresponsabilidad;  
  
    public Estudiante(JSONObject userData) {  
        super(userData);  
        this.horasCorresponsabilidad = 1;  
    }  
  
}
```

### Clase profesor

La clase usuario define las características que tendrá un profesor/Administrado, en este caso hereda todas las características de un abstractUser, pero es definida con un objeto diferente a abstractUse y Estudiante debido a los accesos que este tiene

**Método Profesor:** Es el constructor de la clase Profesor, define cuáles son sus valores base y se asegura que todo usuario tenga esos datos.

```
public class Profesor extends AbstractUser{  
  
    public Profesor(JSONObject userData) {  
        super(userData);  
    }  
  
}
```

### Clase AbstractUser

La clase usuario se encarga de dar las características que tienen un usuarioAbstracto las cuales heredada a profesores y estudiantes, que serán registrados en la base de datos

**Metodo AbstractUser:** Es el constructor de la clase abstractUser, define cuáles son sus valores base y se asegura que todo usuario tenga esos datos.

```
public AbstractUser () {
    this.token = "";
    this.email = "";
    this.role = "";
    this.document = 0;
    this.name = "";
    this.lastname = "";
    this.facultad = "";
}

public AbstractUser (JSONObject userData) {
    this.email = (String) userData.getJSONObject("user").get("email");
    this.token = (String) userData.get("access_token");
    this.role = (String) userData.getJSONObject("user").getString("role");
}

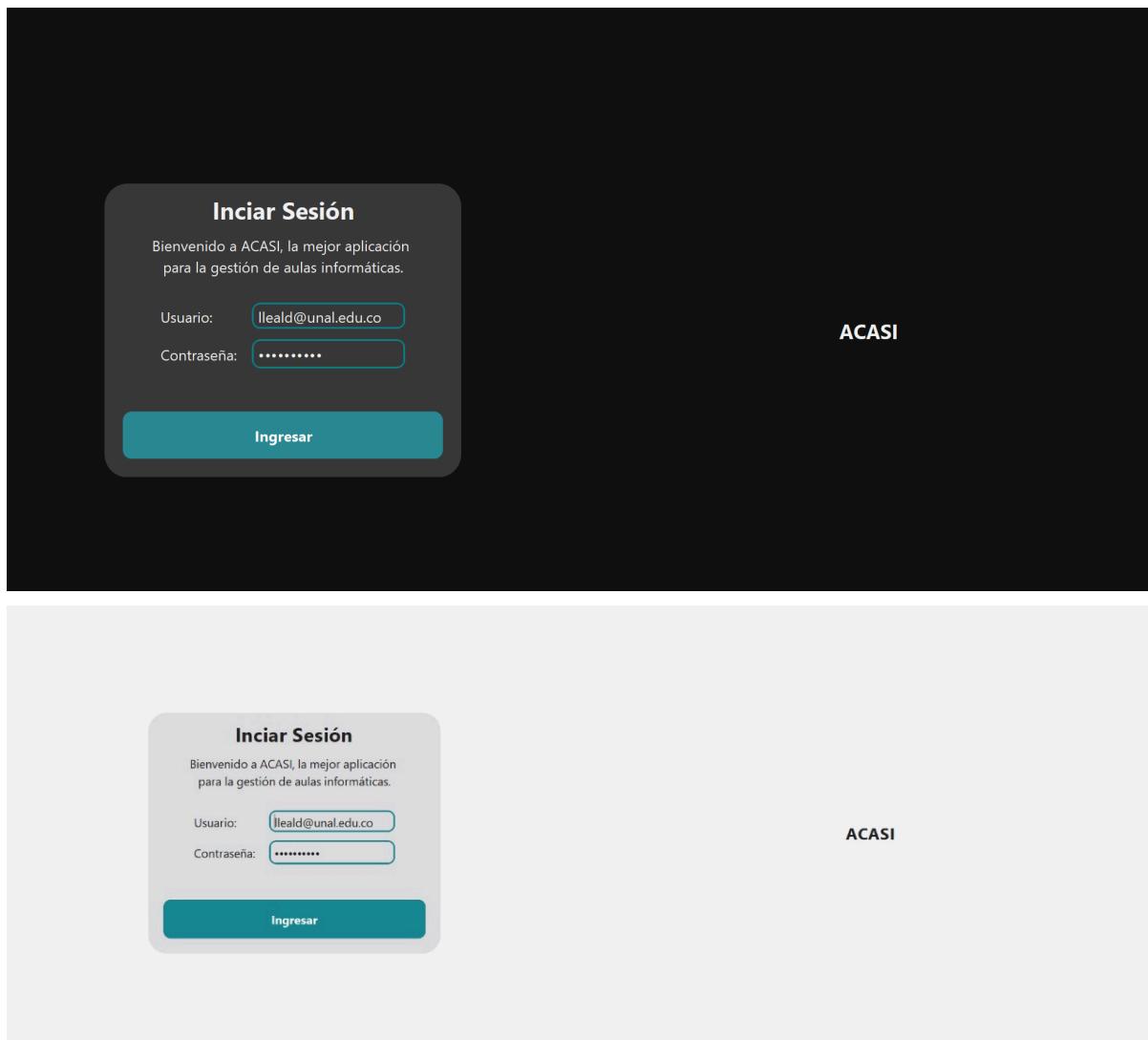
public AbstractUser(JSONObject user_data, Boolean isComplete) {
    if (isComplete){
        this.token = (String) user_data.getJSONObject("user").get("email");
        this.email = (String) user_data.getJSONObject("user").get("email");
        this.role = (String) user_data.getJSONObject("user").get("email");
        this.document = (Integer) user_data.getJSONObject("user").get("email");
        this.name = (String) user_data.getJSONObject("user").get("email");
        this.lastname = (String) user_data.getJSONObject("user").get("email");
        this.facultad = (String) user_data.getJSONObject("user").get("email");
    }
}
```

## Vistas

### Package Vista

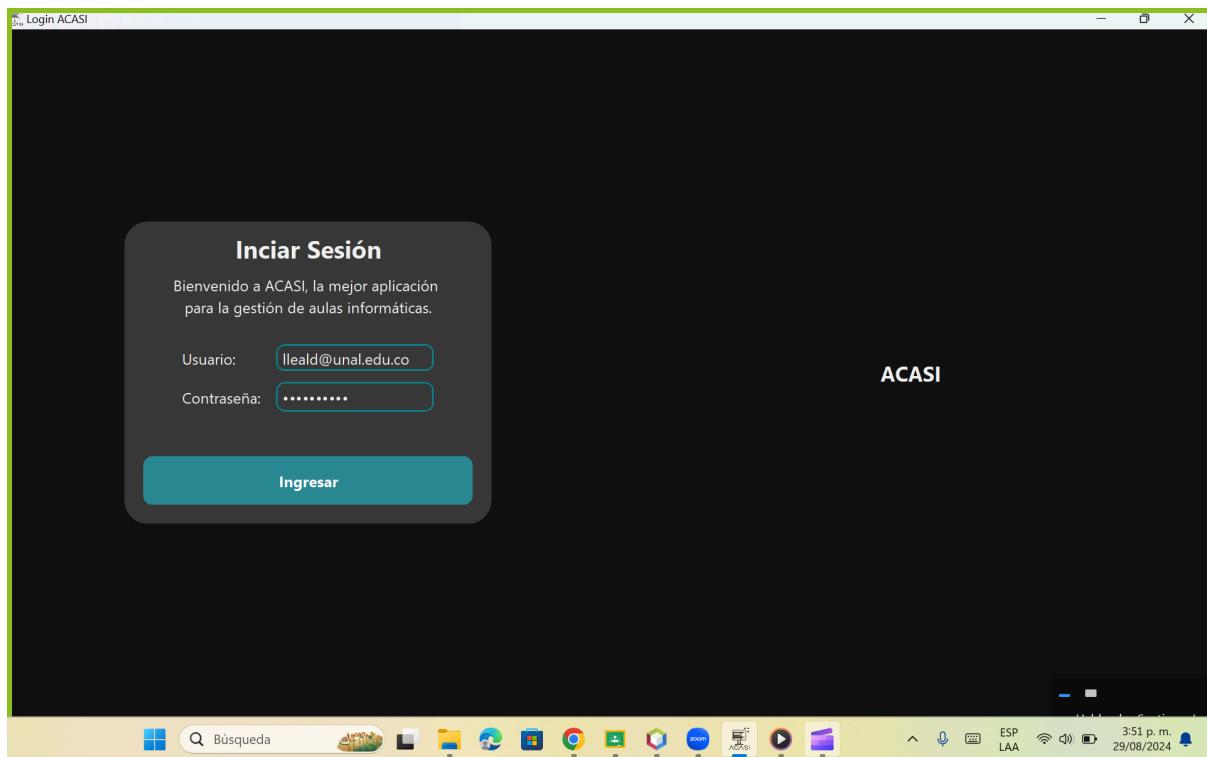
#### Vista Login

Vista **Login** será la vista que se ejecute al inicio de la aplicación, en la cual el usuario podrá ingresar los datos de **Usuario** y **Contraseña** por medio de dos **JTextField** y un **Button ingresar** el cual ejecuta el controlador de **LoginController**, está visual se encarga de identificar que tipo de usuario intenta ingresar y que el usuario esté en la base de datos.



## Vista Application

La vista Application es la vista que contiene el menú el cual el usuario podrá elegir la opción a realizar, la cual está compuesta por 5 Button las opciones son **Salas**, **Usuario**, **Analíticas**, **Corresponsabilidad** y **Configuración**.

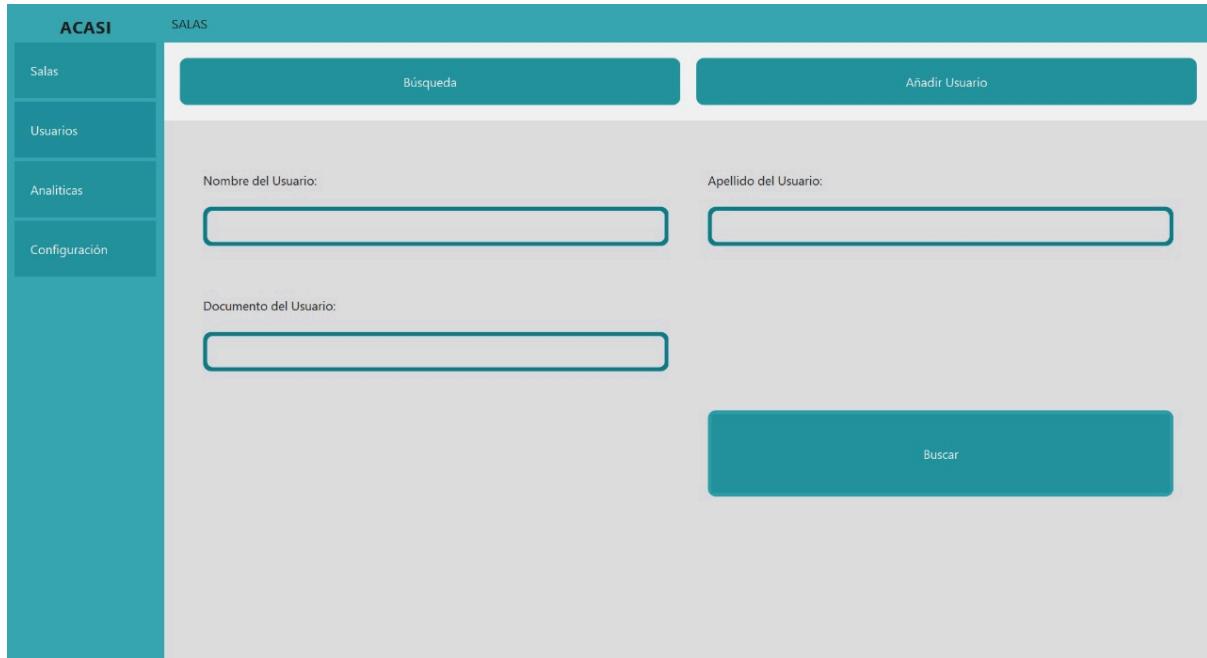


### Vista AU\_CrearUsuario

En la vista Vista AU\_CrearUsuario, encontramos cuatro **JTextField** en donde mediante **Inputs custoomizados** se recolectan los datos de **Nombre/s**, **Apellidos**, **Documento** y **Correo electrónico** del usuario y tres **Combo Box**, **BoxNewUserFaculty** en el cual se selecciona la facultad a la cual pertenece el usuario, **BoxNewUserCareer** en el cual se selecciona la carrera a la que pertenece el usuario y **BoxNewUserVin**, se recolectan los datos y son enviados al controlador **ConAU\_CrearUsuario** y Button **AÑADIR** el cual ejecuta el código .

### Vista AU\_BuscarUsuario

Dentro de la vista **AU\_BuscarUsuario**, encontramos tres tipos de **JTextField** en los cuales mediante **Inputs custoomizados** se recolectan los datos de **Nombre**, **Apellidos** y **Documento** del usuario, para luego enviarlos al controlador **ConAU\_BuscarUsuario**.

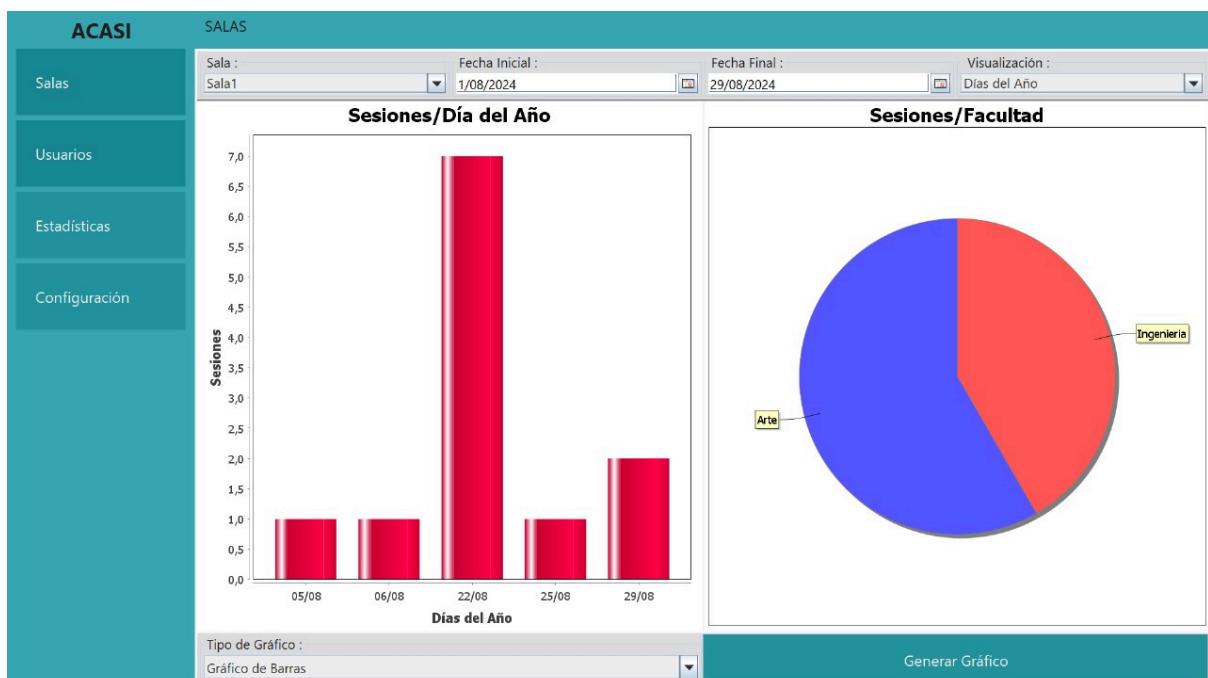


### Vista AU\_InfoUsuario

En la vista **InfoUsuario** se visualizará la información del usuario buscado por la vista **AU\_BuscarUsuario** por medio de tres **JTextField** los cuales mostraran el **Nombre**, **Documento** e **Información extra**.

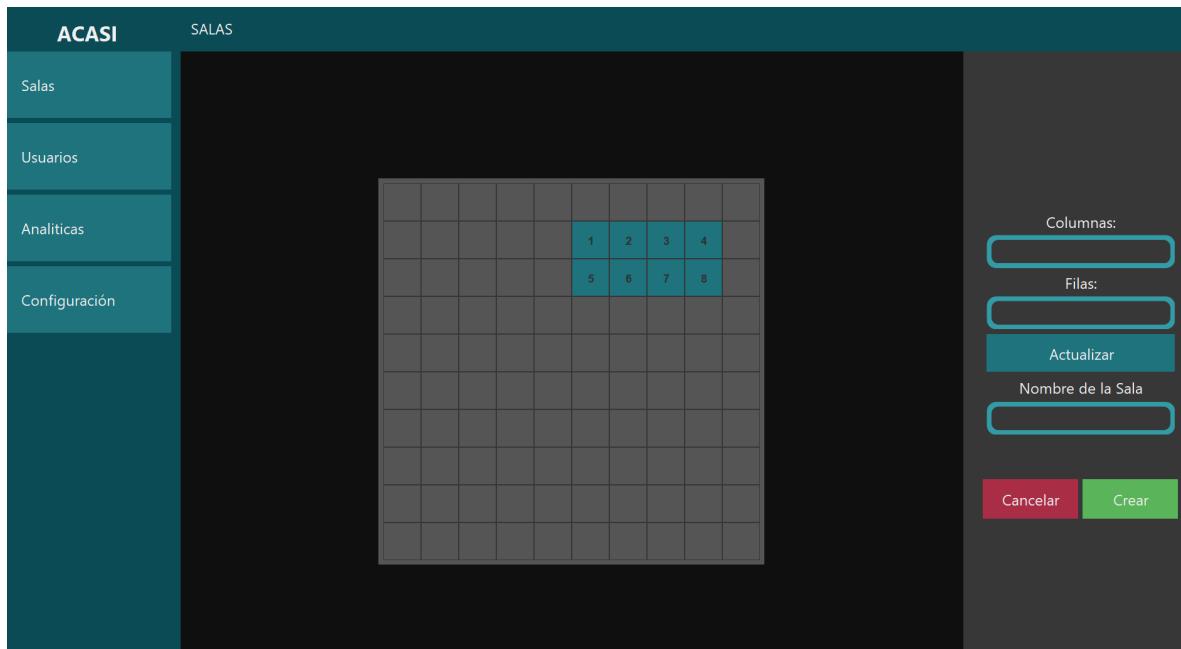
## Vista Estadísticas

vista **Estadísticas** tiene componentes de combo box, **comboBoxSelecciónSalaActionPerformed** permite que el usuario elija la sala a la cual realizar la estadística, **comboBoxSeleccionVisualizacionActionPerformed** en el cual se muestra **Año Mes y Día del año**, **comboBoxSeleccionTipoGraficoActionPerformed** en el cual el usuario tendrá la opción de seleccionar el tipo de gráfica que desea y un **Button Crear grafico** el cual realiza la opción de crear los gráficos.



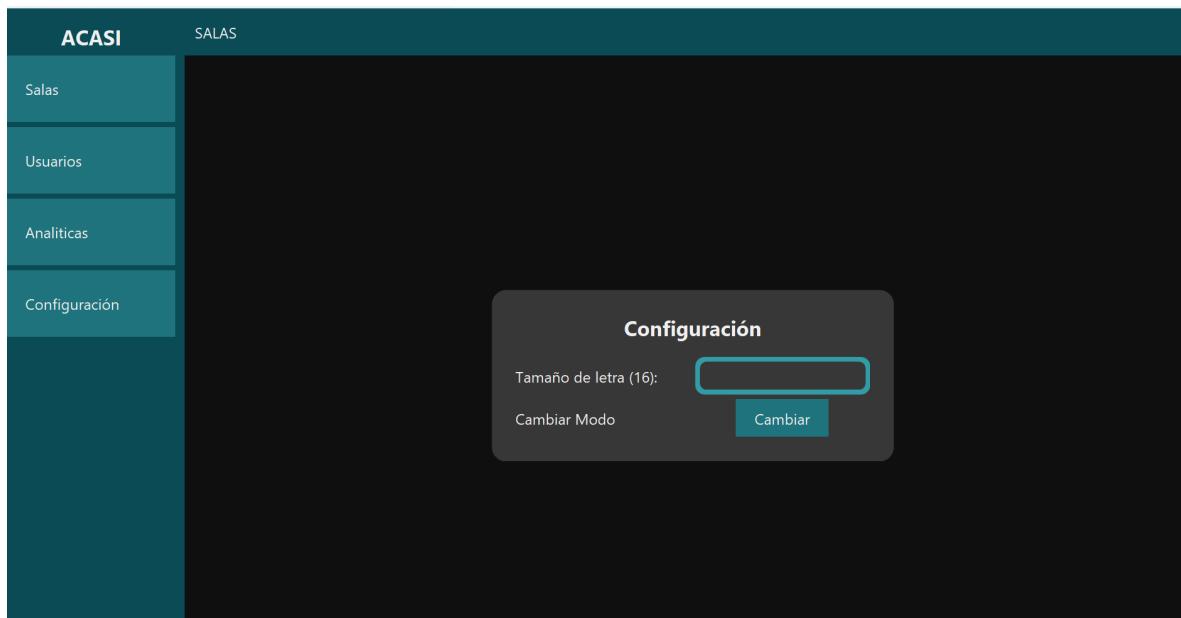
## Vista Create.Sala

En esta vista el docente puede generar una nueva sala a partir de una **malla base**, puede designar los espacios en los que se pondrán los computadores, el nombre que tendrá y adaptar el número de columnas y filas de la malla



## Vista Config

En esta visual el usuario puede modificar el color de la aplicación de modo claro a oscuro y viceversa, también permite modificar el tamaño del texto de la aplicación



## Package Vista.components

### Clase CustomButton

**Constructor CustomButton:** Este constructor por defecto crea un objeto **CustomButton** sin texto y ejecuta el método **configureButton** para configurar el botón con los estilos y comportamientos definidos.

```
public CustomButton() {  
    super();  
    configureButton();  
}
```

**Método configureButton:** Configura los aspectos visuales y de comportamiento del botón, tales como la interfaz de usuario (UI), el fondo, la fuente, el color del texto, el borde, el cursor y los eventos de ratón.

```
private void configureButton() {
    setUI(new BasicButtonUI() {
        @Override
        protected void installDefaults(javax.swing.AbstractButton b) {
            super.installDefaults(b);
            b.setFocusPainted(false);
            b.setContentAreaFilled(false);
            b.setOpaque(true);
        }
    });
}

setBackground(UIConfig.PRIMARY700);
setFont(UIConfig.getMediumFont());
setForeground(new java.awt.Color(255, 255, 255));
setBorder(javax.swing.BorderFactory.createEmptyBorder(10, 30, 10, 30));
setCursor(new Cursor(Cursor.HAND_CURSOR));
setMargin(new java.awt.Insets(5, 20, 5, 20));
setMaximumSize(new java.awt.Dimension(600, 150));

addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseEntered(java.awt.event.MouseEvent evt) {
        btnLoginMouseEntered(evt);
    }

    public void mouseExited(java.awt.event.MouseEvent evt) {
        btnLoginMouseExited(evt);
    }
});
}
```

**Método btnLoginMouseEntered:** Cambia el color de fondo del botón cuando el puntero del ratón entra en su área.

```
private void btnLoginMouseEntered(java.awt.event.MouseEvent evt) {
    Utils.changeBackgroundColor(this, UIConfig.PRIMARY900, 100);
}
```

**Método btnLoginMouseExited:** Cambia el color de fondo del botón cuando el puntero del ratón sale de su área.

```
private void btnLoginMouseExited(java.awt.event.MouseEvent evt) {
    Utils.changeBackgroundColor(this, UIConfig.PRIMARY500, 100);
}
```

**Método setLoadingState:** Cambia el estado del botón para mostrar un cursor de espera y deshabilitar el botón cuando se está cargando (**isLoading**), y vuelve al estado normal una vez completada la carga.

```
public void setLoadingState(boolean isLoading) {  
    if (isLoading) {  
        setCursor(new Cursor(Cursor.WAIT_CURSOR));  
        setEnabled(false);  
    } else {  
        setCursor(new Cursor(Cursor.HAND_CURSOR));  
        setEnabled(true);  
    }  
}
```

### Clase CustomInput

**Constructor CustomInput:** Este constructor por defecto crea un objeto CustomInput y ejecuta el método initComponents para configurar los paneles y el campo de texto.

```
public CustomInput() {  
    super();  
    initComponents();  
}
```

**Método initComponents:** Configura los paneles exteriores e interiores, así como el campo de texto dentro del **CustomInput**. Se define la apariencia de los paneles con bordes redondeados y se aplican colores según la configuración de la interfaz (**UIConfig**).

```
private void initComponents() {  
    // Outer panel configuration  
    setBackground(UIConfig.getBg(300));  
    outerPanel = new PanelRound();  
    outerPanel.setBackground(UIConfig.getPrimaryColor(900));  
    outerPanel.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));  
    outerPanel.setRoundBottomLeft(20);  
    outerPanel.setRoundBottomRight(20);  
    outerPanel.setRoundTopLeft(20);  
    outerPanel.setRoundTopRight(20);  
    outerPanel.setLayout(new BoxLayout(outerPanel, BoxLayout.LINE_AXIS));  
  
    // Inner panel configuration  
    innerPanel = new PanelRound();  
    innerPanel.setBackground(UIConfig.getBg(300));  
    innerPanel.setBorder(BorderFactory.createEmptyBorder(3, 5, 3, 5));  
    innerPanel.setPreferredSize(new java.awt.Dimension(164, 25));  
    innerPanel.setRoundBottomLeft(15);  
    innerPanel.setRoundBottomRight(15);  
    innerPanel.setRoundTopLeft(15);  
    innerPanel.setRoundTopRight(15);  
    innerPanel.setLayout(new BoxLayout(innerPanel, BoxLayout.LINE_AXIS));  
  
    // Username field configuration  
    field = new JTextField();  
    field.setBackground(UIConfig.getBg(300));  
    field.setColumns(15);  
    field.setFont(UIConfig.getDefaultFont());  
    field.setText("");  
    field.setBorder(null);  
    field.setForeground(UIConfig.getForeground());  
  
    // Add the username field to the inner panel  
    innerPanel.add(field);  
  
    // Add the inner panel to the outer panel  
    outerPanel.add(innerPanel);  
  
    // Add the outer panel to this component  
    this.setLayout(new BoxLayout(this, BoxLayout.LINE_AXIS));  
    this.add(outerPanel);  
}
```

**Método getField:** Retorna el objeto JTextField contenido en el CustomInput.

```
public JTextField getField() {  
    return field;  
}
```

**Método setText:** Establece el texto del campo **JTextField**.

```
// método para establecer el texto
public void setText(String text) {
    field.setText(text);
}
```

**Método getText:** Obtiene el texto actual del campo **JTextField**.

```
// método para obtener el texto actual
public String getText() {
    return field.getText();
}
```

## Clase CustomLabel

**Constructor CustomLabel:** Este constructor por defecto crea un objeto CustomLabel y aplica la fuente y el color de texto predeterminados definidos en UIConfig.

```
public class CustomLabel extends javax.swing.JLabel {

    public CustomLabel() {
        super();
        setFont(UIConfig.getDefaultFont());
        setForeground(UIConfig.getForeground());
    }

}
```

## Package Vista.utils

### Clase Utils

**Método changeBackgroundColor:** Este método estático se utiliza para cambiar gradualmente el color de fondo de un componente Component de Swing desde un color inicial al color final especificado (endColor) durante un período de tiempo determinado (duration).

### Clase UIConfig

**Método loadPreferences:** Carga las preferencias desde el archivo de propiedades config.properties. Si el archivo no se encuentra, utiliza los valores por defecto. Este método es clave para inicializar la configuración de la UI de acuerdo con las preferencias guardadas.

```

private static void loadPreferences() {
    Properties properties = new Properties();
    try (FileInputStream in = new FileInputStream(name: CONFIG_FILE)) {
        properties.load(inStream: in);
        darkMode = Boolean.parseBoolean(s: properties.getProperty(key: DARK_MODE_KEY, defaultValue: "false"));
    } catch (IOException e) {
        System.out.println(x: "No se pudo cargar el archivo de configuración. Usando valores por defecto.");
    }
}

```

**Método savePreferences:** Guarda las preferencias actuales en el archivo de propiedades config.properties. Es utilizado para persistir cambios en las configuraciones de la UI, como el modo oscuro.

```

private static void loadPreferences() {
    Properties properties = new Properties();
    try (FileInputStream in = new FileInputStream(name: CONFIG_FILE)) {
        properties.load(inStream: in);
        darkMode = Boolean.parseBoolean(s: properties.getProperty(key: DARK_MODE_KEY, defaultValue: "false"));
    } catch (IOException e) {
        System.out.println(x: "No se pudo cargar el archivo de configuración. Usando valores por defecto.");
    }
}

```

**Método updateSize:** Actualiza los tamaños de texto basándose en el tamaño por defecto proporcionado. Ajusta todos los tamaños de texto, en consecuencia, desde text\_sm hasta text\_4xl.

```

public static void updateSize(Integer defaultSize) {
    text_default = defaultSize;
    text_sm = text_default - 2;
    text_md = text_default;
    text_lg = text_default + 2;
    text_xl = text_default + 4;
    text_2xl = text_default + 6;
    text_3xl = text_default + 8;
    text_4xl = text_default + 10;
}

```

**Método generateFont:** Genera una fuente con el peso y tamaño especificado. Es un método útil para crear fuentes personalizadas según las necesidades de la UI.

```

public static Font generateFont(Integer fontWeight, Integer size) {
    return new Font(name: PRIMARYFONT, style: fontWeight, size);
}

```

**Método getDefaultFont:** Obtiene la fuente por defecto para el texto general. Este método es útil para asegurar la coherencia en el uso de fuentes en toda la aplicación.

```
public static Font getDefaultFont() {  
    return generateFont(fontWeight: Font.PLAIN, size:text_md);  
}
```

**Método getSubtitleFont:** Obtiene la fuente predeterminada para subtítulos, que generalmente es una fuente en negrita y de tamaño mayor que la del texto por defecto.

```
public static Font getSubtitleFont() {  
    return generateFont(fontWeight: Font.BOLD, size:text_xl);  
}
```

**Método getTitleFont:** Obtiene la fuente predeterminada para títulos, generalmente en negrita y de mayor tamaño que la fuente de subtítulos.

```
public static Font getSubtitleFont() {  
    return generateFont(fontWeight: Font.BOLD, size:text_xl);  
}
```

**Método getTitleXLFont:** Obtiene la fuente para títulos extra grandes, ideal para encabezados principales.

```
public static Font getSubtitleFont() {  
    return generateFont(fontWeight: Font.BOLD, size:text_xl);  
}
```

**Método getMediumFont:** Obtiene la fuente de tamaño medio, útil para textos que necesitan destacarse sin ser tan grandes como los títulos.

```
public static Font getSubtitleFont() {  
    return generateFont(fontWeight: Font.BOLD, size:text_xl);  
}
```

**Método generateFontBySize:** Genera una nueva fuente según el tamaño de texto especificado, proporcionando flexibilidad en la personalización del texto en la UI.

```
public static Font generateFontBySize(Integer size) {  
    return generateFont(fontWeight: Font.PLAIN, size);  
}
```

**Método getBgColor:** Devuelve el color de fondo según el modo actual (oscuro o claro). Este método permite ajustar la interfaz según la preferencia del usuario.

```
public static Color getBgColor() {  
    return darkMode ? BGDIRTCOLOR : BGWHITECOLOR;  
}
```

**Método getForeground:** Devuelve el color de primer plano (foreground) según el modo actual, asegurando la legibilidad del texto en diferentes configuraciones de fondo.

```
public static Color getForeground() {  
    return darkMode ? FOREGROUNDBLACK : FOREGROUNDWHITE;  
}
```

**Método getPrimaryColor:** Devuelve el color primario ajustado según el modo (oscuro o claro) y la tonalidad deseada. Este método es esencial para mantener la coherencia del diseño en toda la aplicación.

```
public static Color getPrimaryColor(int shade) {  
    if (darkMode) {  
        switch (shade) {  
            case 100:  
                return DARKPRIMARY100;  
            case 300:  
                return DARKPRIMARY300;  
            case 500:  
                return DARKPRIMARY500;  
            case 700:  
                return DARKPRIMARY700;  
            case 900:  
                return DARKPRIMARY900;  
            default:  
                return DARKPRIMARY;  
        }  
    } else {  
        switch (shade) {  
            case 100:  
                return PRIMARY100;  
            case 300:  
                return PRIMARY300;  
            case 500:  
                return PRIMARY500;  
            case 700:  
                return PRIMARY700;  
            case 900:  
                return PRIMARY900;  
            default:  
                return PRIMARY;  
        }  
    }  
}
```

**Método getBg:** Devuelve el color de fondo ajustado según la tonalidad deseada y el modo oscuro o claro. Proporciona flexibilidad para el diseño de la UI.

```
public static Color getBg(int shade) {  
    if (darkMode) {  
        switch (shade) {  
            case 100:  
                return BGDARKCOLOR;  
            case 300:  
                return BGDARK300;  
            default:  
                return BGDARKCOLOR;  
        }  
    } else {  
        switch (shade) {  
            case 100:  
                return BGWHITECOLOR;  
            case 300:  
                return BGWHITE300;  
            default:  
                return BGWHITECOLOR;  
        }  
    }  
}
```

**Método changeMode:** Cambia el modo oscuro y guarda la preferencia en el archivo de configuración. Este método permite al usuario alternar entre modos y persistir su elección.

```

public static Color getBg(int shade) {
    if (darkMode) {
        switch (shade) {
            case 100:
                return BGDARKCOLOR;
            case 300:
                return BGDARK300;
            default:
                return BGDARKCOLOR;
        }
    } else {
        switch (shade) {
            case 100:
                return BGWHITECOLOR;
            case 300:
                return BGWHITE300;
            default:
                return BGWHITECOLOR;
        }
    }
}

```

## Clase PanelRound

**Método PanelRound:** Constructor de la clase PanelRound. Inicializa un panel con esquinas redondeadas y lo configura como no opaco. Este panel personalizado permite ajustar individualmente el radio de redondeo de cada esquina (superior izquierda, superior derecha, inferior izquierda, e inferior derecha).

```

public PanelRound() {
    setOpaque(isOpaque: false);
}

```

**Método getRoundTopLeft:** Devuelve el valor del radio de redondeo de la esquina superior izquierda. Este método permite obtener el radio actual configurado para la esquina superior izquierda del panel.

```
public int getRoundTopLeft() {  
    return roundTopLeft;  
}
```

**Método setRoundTopLeft:** Establece el valor del radio de redondeo de la esquina superior izquierda y repinta el panel. Es útil para personalizar la esquina superior izquierda del panel.

```
public void setRoundTopLeft(int roundTopLeft) {  
    this.roundTopLeft = roundTopLeft;  
    repaint();  
}
```

**Método getRoundTopRight:** Devuelve el valor del radio de redondeo de la esquina superior derecha. Permite obtener el radio actual configurado para la esquina superior derecha del panel.

```
public int getRoundTopRight() {  
    return roundTopRight;  
}
```

**Método setRoundTopRight:** Establece el valor del radio de redondeo de la esquina superior derecha y repinta el panel. Sirve para personalizar la esquina superior derecha del panel.

```
public int getRoundTopRight() {  
    return roundTopRight;  
}
```

**Método getRoundBottomLeft:** Devuelve el valor del radio de redondeo de la esquina inferior izquierda. Este método permite obtener el radio actual configurado para la esquina inferior izquierda del panel.

```
public int getRoundBottomLeft() {  
    return roundBottomLeft;  
}
```

**Método setRoundBottomLeft:** Establece el valor del radio de redondeo de la esquina inferior izquierda y repinta el panel. Es útil para personalizar la esquina inferior izquierda del panel.

```
public void setRoundBottomLeft(int roundBottomLeft) {  
    this.roundBottomLeft = roundBottomLeft;  
    repaint();  
}
```

**Método getRoundBottomRight:** Devuelve el valor del radio de redondeo de la esquina inferior derecha. Permite obtener el radio actual configurado para la esquina inferior derecha del panel.

```
public int getRoundBottomRight() {  
    return roundBottomRight;  
}
```

**Método setRoundBottomRight:** Establece el valor del radio de redondeo de la esquina inferior derecha y repinta el panel. Sirve para personalizar la esquina inferior derecha del panel.

```
public void setRoundBottomRight(int roundBottomRight) {  
    this.roundBottomRight = roundBottomRight;  
    repaint();  
}
```

**Método paintComponent:** Sobrescribe el método paintComponent de JPanel. Este método se encarga de dibujar el panel con las esquinas redondeadas según los valores actuales de los radios de redondeo. Utiliza antialiasing para suavizar los bordes y asegura que el panel se pinte correctamente. El método crea áreas de forma redondeada para cada esquina según las configuraciones, y las combina para pintar el panel.

```

protected void paintComponent(Graphics grphcs) {
    Graphics2D g2 = (Graphics2D) grphcs.create();
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
    g2.setColor(c.getBackground());
    Area area = new Area(s.createRoundTopLeft());
    if (roundTopRight > 0) {
        area.intersect(new Area(s.createRoundTopRight()));
    }
    if (roundBottomLeft > 0) {
        area.intersect(new Area(s.createRoundBottomLeft()));
    }
    if (roundBottomRight > 0) {
        area.intersect(new Area(s.createRoundBottomRight()));
    }
    g2.fill(s.area);
    g2.dispose();
    super.paintComponent(g: grphcs);
}

```

**Método createRoundTopLeft:** Crea y devuelve una forma que representa el área redondeada de la esquina superior izquierda. Utiliza la clase Area para combinar un rectángulo redondeado con rectángulos adicionales que completan la forma.

```

private Shape createRoundTopLeft() {
    int width = getWidth();
    int height = getHeight();
    int roundX = Math.min(a: width, b: roundTopLeft);
    int roundY = Math.min(a: height, b: roundTopLeft);
    Area area = new Area(new RoundRectangle2D.Double(x: 0, y: 0, w: width, h: height, arcw: roundX, arch: roundY));
    area.add(new Area(new Rectangle2D.Double(roundX / 2, y: 0, width - roundX / 2, h: height)));
    area.add(new Area(new Rectangle2D.Double(x: 0, roundY / 2, w: width, height - roundY / 2)));
    return area;
}

```

**Método createRoundTopRight:** Crea y devuelve una forma que representa el área redondeada de la esquina superior derecha. Utiliza la clase Area para combinar un rectángulo redondeado con rectángulos adicionales que completan la forma.

```

private Shape createRoundTopRight() {
    int width = getWidth();
    int height = getHeight();
    int roundX = Math.min(a: width, b: roundTopRight);
    int roundY = Math.min(a: height, b: roundTopRight);
    Area area = new Area(new RoundRectangle2D.Double(x: 0, y: 0, w: width, h: height, arcw: roundX, arch: roundY));
    area.add(new Area(new Rectangle2D.Double(x: 0, y: 0, width - roundX / 2, h: height)));
    area.add(new Area(new Rectangle2D.Double(x: 0, roundY / 2, w: width, height - roundY / 2)));
    return area;
}

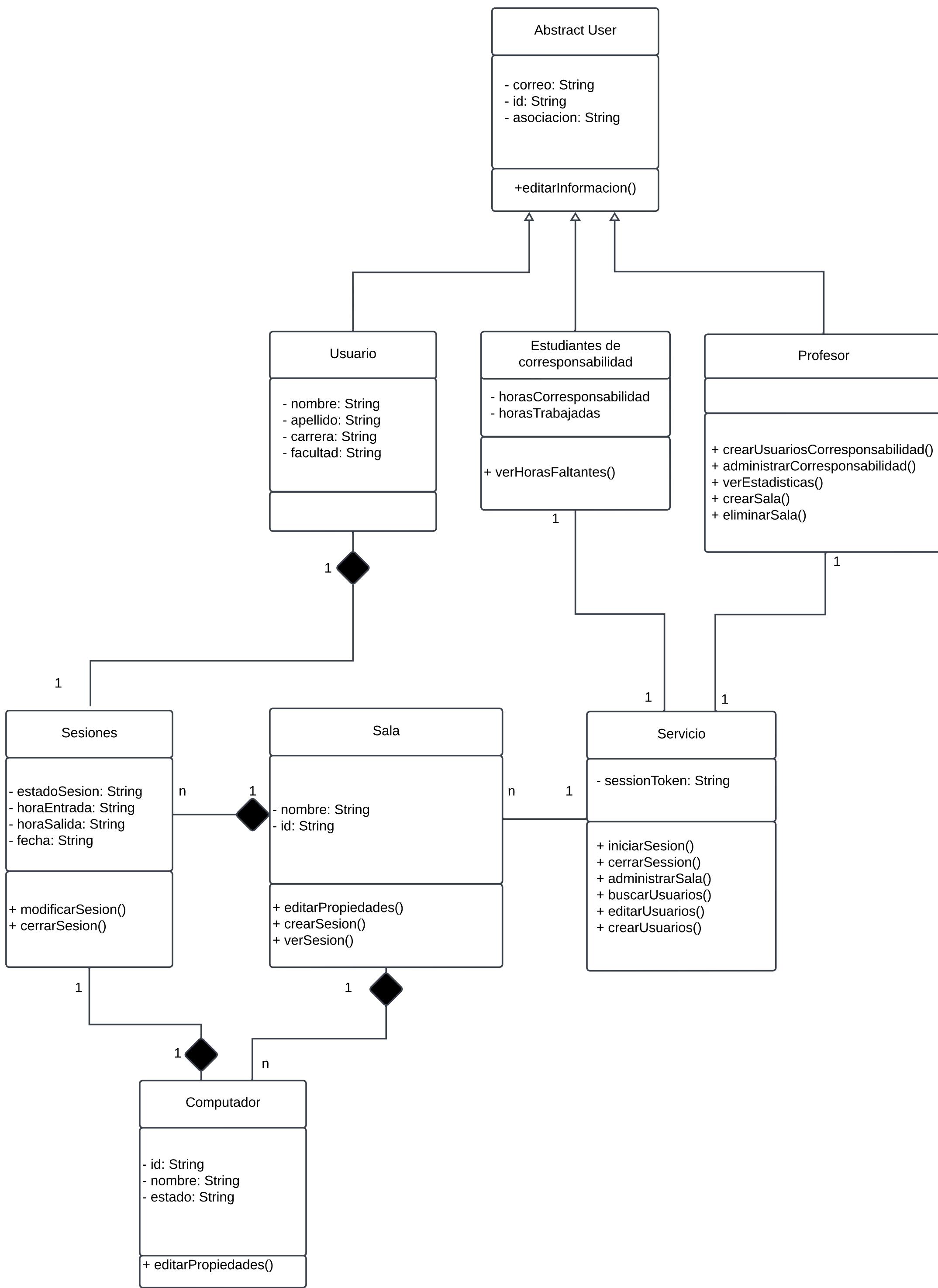
```

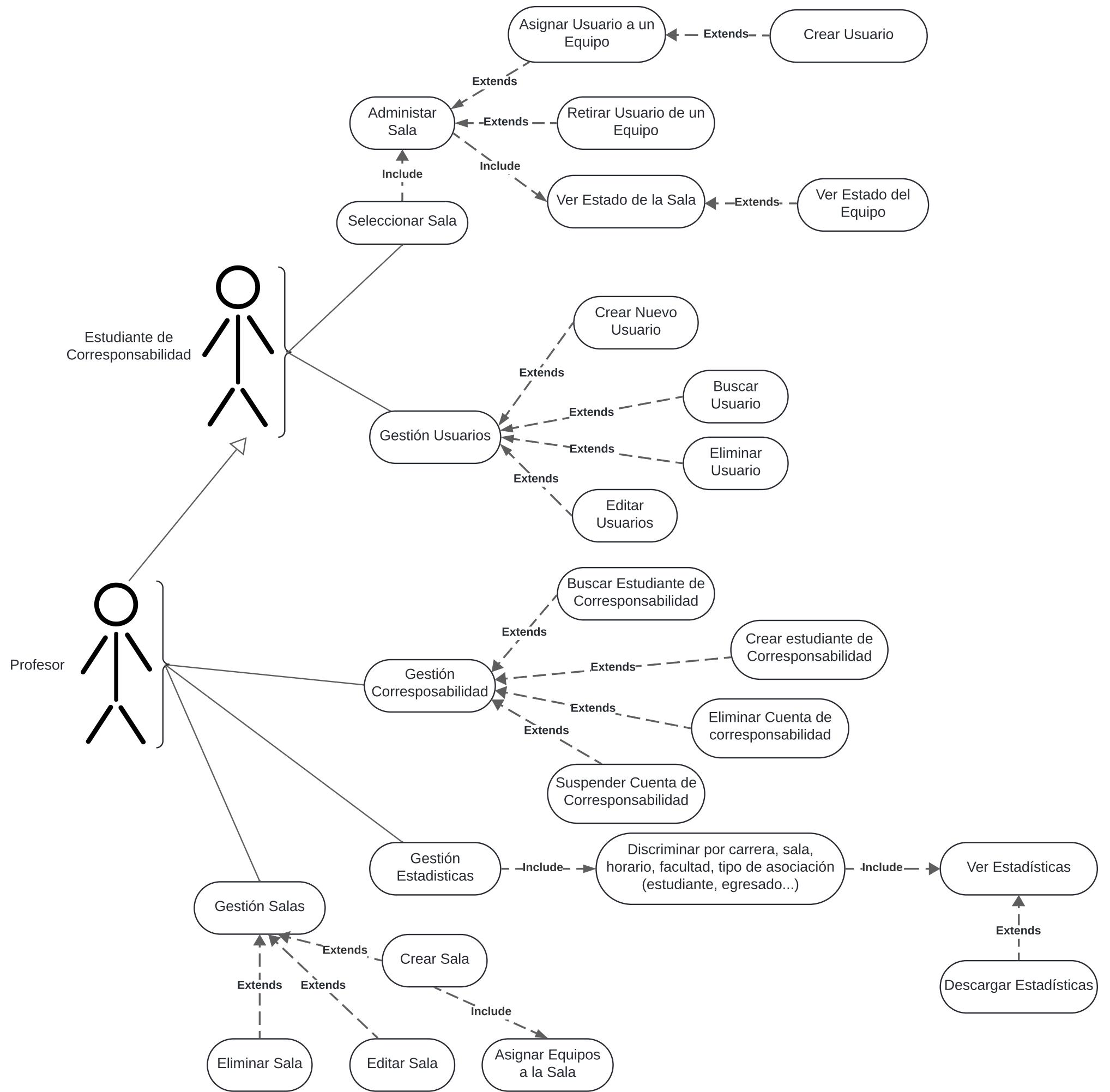
**Método createRoundBottomLeft:** Crea y devuelve una forma que representa el área redondeada de la esquina inferior izquierda. Utiliza la clase Area para combinar un rectángulo redondeado con rectángulos adicionales que completan la forma.

```
private Shape createRoundBottomLeft() {
    int width = getWidth();
    int height = getHeight();
    int roundX = Math.min(a: width, b: roundBottomLeft);
    int roundY = Math.min(a: height, b: roundBottomLeft);
    Area area = new Area(new RoundRectangle2D.Double(x: 0, y: 0, w: width, h: height, arcw: roundX, arch: roundY));
    area.add(new Area(new Rectangle2D.Double(roundX / 2, y: 0, width - roundX / 2, h: height)));
    area.add(new Area(new Rectangle2D.Double(x: 0, y: 0, w: width, height - roundY / 2)));
    return area;
}
```

**Método createRoundBottomRight:** Crea y devuelve una forma que representa el área redondeada de la esquina inferior derecha. Utiliza la clase Area para combinar un rectángulo redondeado con rectángulos adicionales que completan la forma.

```
private Shape createRoundBottomRight() {
    int width = getWidth();
    int height = getHeight();
    int roundX = Math.min(a: width, b: roundBottomRight);
    int roundY = Math.min(a: height, b: roundBottomRight);
    Area area = new Area(new RoundRectangle2D.Double(x: 0, y: 0, w: width, h: height, arcw: roundX, arch: roundY));
    area.add(new Area(new Rectangle2D.Double(x: 0, y: 0, width - roundX / 2, h: height)));
    area.add(new Area(new Rectangle2D.Double(x: 0, y: 0, w: width, height - roundY / 2)));
    return area;
}
```





<b>Nombre</b>	<b>GU-1 Buscar Usuario</b>
Descripción:	En este proceso se muestra las diversas formas de buscar usuarios al administrador o estudiante, acontinuacion el programa busca los datos ingresados para encontrar un usuario que coincida con los datos ingresados.
Flujo Principal:	1.El sistema autentica muestra las diferentes opciones de busqueda del usuario <b>[A1]</b> 2.El programa busca en la base de datos un usuario que coincide con los datos <b>[A2] [A3]</b> . 3.El sistema muestra en un nuevo visual InfoUser toda la informacion del usuario encontrado
Flujo Alterno:	<b>A1.</b> Muestra las opciones de busqueda por Nombre, Apellido o Documento 1. No se muestra las opciones gestionar corresponsabilidad. 2. Se continua con el flujo principal. <b>A2.</b> No se encuentra un usuario que coincida con los datos ingresados. 1. Se muestar un errorMessage. 2. Se retorna el flujo principal. <b>A3.</b> Se encontraron dos o mas usuarios con los mismos datos (Nombre o Apellido) 1. Se muestra la informacion del primer usuario. 2. Se continua con el flujo principal del proceso seleccionado.

Nombre	GU-2 Crear Usuario	
Descripción:	En este proceso se muestra la visual para crear usuarios, junto a todos los datos que este debe tener, estos datos se capturan para crear un nuevo estudiante en la base de datos.	
Flujo Principal:	<p>1.Muestra todos los campos para ingresar los datos del nuevo usuario: Nombre, Apellido, Documento, Correo, Vinculacion, Carrera, Facultad<b>[A1]</b>.</p> <p>2.El programa crea un usuario en la base de datos con los datos obtenido.</p> <p>3.El sistema muestra en un nuevo mensaje de usuario creado exitosamente.</p>	
Flujo Alterno:	<p><b>A1.</b> Si no se llenaron todos los espacio</p> <p>1. Se muestra un errorMessage.</p> <p>2. Se realiza un return.</p>	

Nombre	GU-3 Editar Usuario	
Descripción:	En este proceso se muestra la visual para modificar usuarios, en este se buscara el usuario que desea modificar junto a que dato desea cambiar	
Flujo Principal:	1. Realiza la buscar usuarios[Gu-1] 2. El programa muestra la visual con diferentes opciones para escoger el dato que desea modificar. 3. El administrador o EstudianteCorresponsabilidad ingresa el nuevo dato <b>[A1]</b> 4. El programa cambia el dato modificado en la base de datos. 5. El programa muestra un mensaje de cambio exitoso	
Flujo Alterno:	<b>A1.</b> Se ingresa un dato no valido 1. Se muestra un errorMessage. 2. Se realiza un return.	

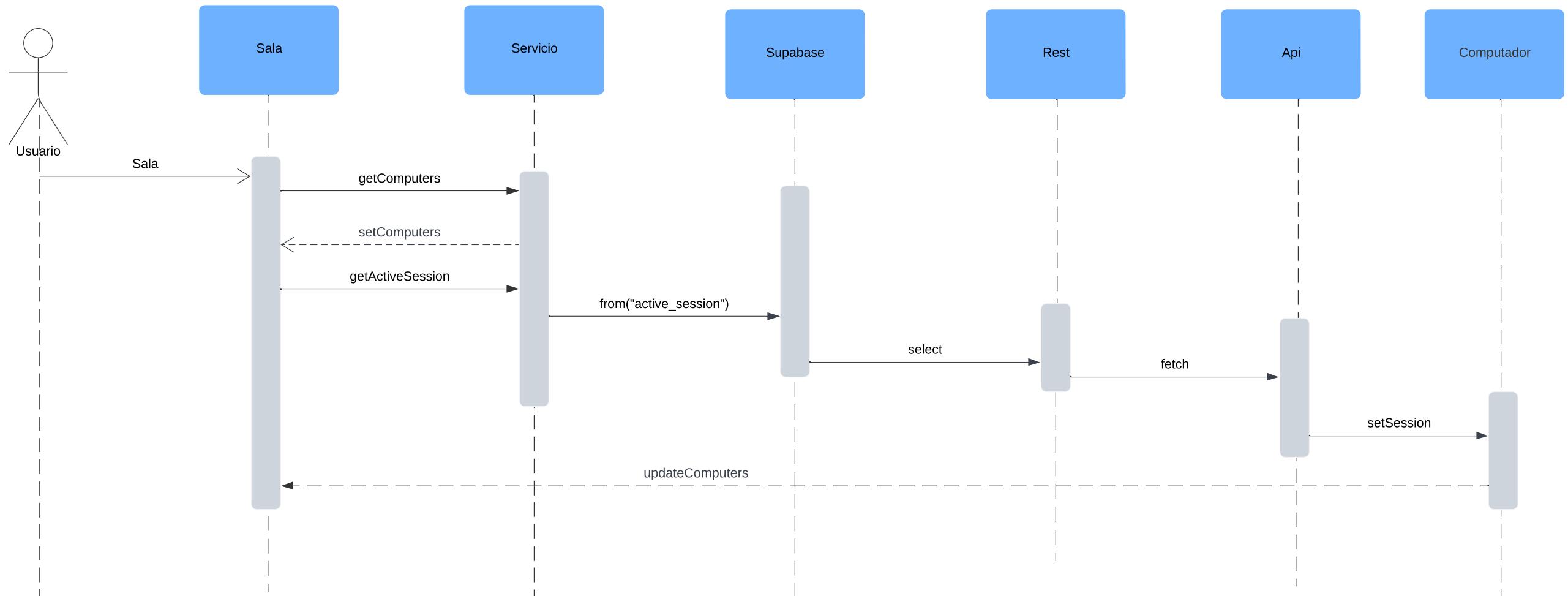
<b>Nombre</b>	<b>GU-4 Eliminar Usuario</b>	
Descripción:	En este proceso se muestra la visual para Eliminar Usuarios, en este se hara la busqueda del usuario a eliminar y se pedira una confirmacion.	
Flujo Principal:	1.Realiza la buscar usuarios[Gu-1] 2.El programa pide un confirmation mediante un boton 3.El programa elimina el usuario en la base de datos. 4.El programa muestra un mensaje de eliminado exitosamente exitoso	
Flujo Alterno:		
<b>Nombre</b>	<b>GC-1 Buscar Estudiantes Corresponsabilidad</b>	
Descripción:	En este proceso se muestra las diversas formas de buscar usuarios al administrador, acontinuacion el programa busca los datos ingresados para encontrar un usuarioCorresponsabilidad que coincida con los datos ingresados.	
Flujo Principal:	1.El sistema autentica muestra las diferentes opciones de busqueda del usuario <b>[A1]</b> 2.El programa busca en la base de datos un usuario que coincide con los datos <b>[A2] [A3]</b> . 3.El sistema muestra en un nuevo visual InfoUser toda la informacion del usuario encontrado	
Flujo Alterno:	<p><b>A1.</b> Muestra las opciones de busqueda por Nombre, Apellido o Documento</p> <ol style="list-style-type: none"> <li>1. No se muestra las opciones gestionar corresponsabilidad.</li> <li>2. Se continua con el flujo principal.</li> </ol> <p><b>A2.</b> No se encuentra un usuario que coincida con los datos ingresados.</p> <ol style="list-style-type: none"> <li>1. Se muestra un errorMessage.</li> <li>2. Se retorna el flujo principal.</li> </ol> <p><b>A3.</b> Se encontraron dos o mas usuarios con los mismos datos (Nombre o Apellido)</p> <ol style="list-style-type: none"> <li>1. Se muestra la informacion del primer usuarioCorresponsabilidad.</li> <li>2. Se continua con el flujo principal del proceso seleccionado.</li> </ol>	
<b>Nombre</b>	<b>GC-2 Crear Estudiantes Corresponsabilidad</b>	
Descripción:	En este proceso se muestra la visual para crear usuarios, junto a todos los datos que este debe tener, estos datos se capturan para crear un nuevo estudianteCorresponsabilidad en la base de datos.	
Flujo Principal:	1.Muestra todos los campos para ingresar los datos del nuevo usuarioCorresponsabilidad Nombre, Apellido, Documento, Correo, Vinculacion, Carrera, Facultad, TimeCorresponsabilidad <b>[A1]</b> . 2.El programa crea un usuario en la base de datos con los datos obtenido. 3.El sistema muestra en un nuevo mensaje de usuario creado exitosamente.	
Flujo Alterno:	<p><b>A1.</b> Si no se llenaron todos los espacio</p> <ol style="list-style-type: none"> <li>1. Se muestra un errorMessage.</li> <li>2. Se realiza un return.</li> </ol>	

<b>Nombre</b>	<b>GC-3 Eliminar Estudiantes Corresponsabilidad</b>	
Descripción:	En este proceso se muestra la visual para Eliminar UsuariosCorresponsabilidad, en este se hara la busqueda del usuario a eliminar y se pedira una confirmacion..	
Flujo Principal:	1.Realiza la buscar usuarios[Gc-1] 2.El programa pide un confirmation mediante un boton 3.El programa elimina el usuario en la base de datos. 4.El programa muestra un mesaje de eliminado exitosamente exitoso	
Flujo Alterno:		
<b>Nombre</b>	<b>GC-3 Suspender Estudiantes Corresponsabilidad</b>	
Descripción:	En este proceso se muestra la visual para Eliminar UsuariosCorresponsabilidad, en este se hara la busqueda del usuario a eliminar y se pedira una confirmation..	
Flujo Principal:	1.Realiza la buscar usuarios[Gc-1] 2.El programa pide un confirmation mediante un boton 3.El programa elimina el usuario en la base de datos. 4.El programa muestra un mesaje de eliminado exitosamente exitoso	
Flujo Alterno:		
<b>Nombre</b>	<b>Ver Estadisticas</b>	
Descripción:	En este proceso se nos permite ver las estadisticas con las limitaciones dadas.	
Flujo Principal:	1. Se filtra en la base de datos las sesiones que encajan con el tiempo especifico y la sala a analizar. 2. Se organiza la informacion en arreglos con el formato de la libreria "FreeChart" 3. Se recorre el arreglo anterior para añadirlo al elemento seleccionado de FreeChart. 4. Se activa la visualización del componente para que se pueda ver facilmente.	
Flujo Alterno:		
<b>Nombre</b>	<b>Exportar Estadisticas</b>	
Descripción:	En este proceso se nos permite exportar el grafico de estadisticas seleccionado que actualmente se estan viendo.	
Flujo Principal:	1. <u>Se</u> selecciona el grafico a exportar.[A1] 2. Se convierten las graficas que estan en la visual en imagenes.	
Flujo Alterno:	A1. Se oprime el boton exportar del grafico correspondiente.	

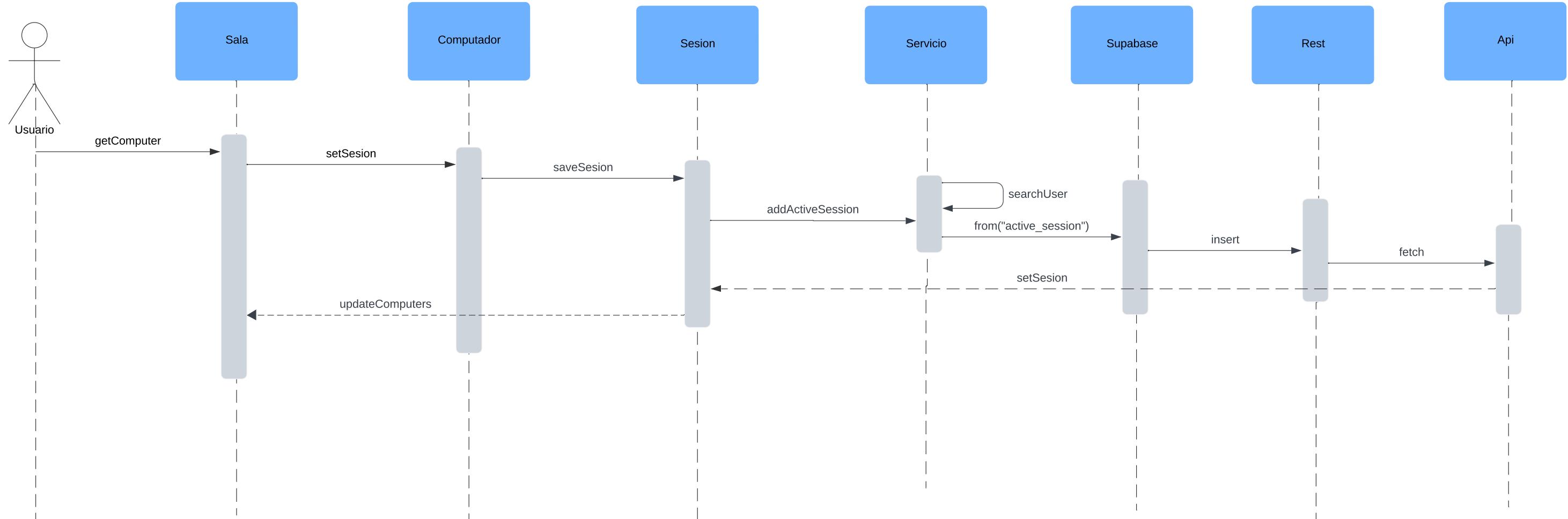
<b>Nombre</b>	<b>GS-1 Asignar Usuario a un equipo</b>
Descripción:	En este proceso se encuentran las actividades que deben realizar tanto los profesores como los estudiantes de corresponsabilidad para asignar un usuario a un equipo de una sala.
Flujo Principal:	<ol style="list-style-type: none"> <li>1. El administrador ingresa el número de identificación del usuario. <b>[A1]</b></li> <li>2. El administrador crea una sesión en esa sala.</li> <li>3. La aplicación envía el número de identificación del usuario a la base de datos.</li> <li>4. La base de datos retorna toda la información del usuario. <b>[A2]</b></li> <li>5. El administrador abre la sesión en el equipo asignado al usuario.</li> </ol>
Flujo Alterno:	<p><b>A1.</b> El administrador no está registrado en el sistema.</p> <ol style="list-style-type: none"> <li>1. El sistema no le permite ingresar.</li> <li>2. Se continua con el flujo principal.</li> </ol> <p><b>A2.</b> El usuario ingresado no existe.</p> <ol style="list-style-type: none"> <li>1. El administrador solicita al usuario su información y crea el usuario.</li> <li>2. Se continua con el flujo principal.</li> </ol>
<b>Nombre</b>	<b>GS-2 Cerrar Sesión Usuario de Equipo</b>
Descripción:	En este proceso se encuentran las actividades que deben realizar para eliminar a un usuario de los equipos de una sala
Flujo Principal:	<ol style="list-style-type: none"> <li>1. El administrador ingresa el número de identificación del usuario.</li> <li>2. La aplicación envía el número de identificación del usuario a la base de datos.</li> <li>3. La base de datos retorna toda la información del usuario.</li> <li>5. El usuario sale de la sala.</li> <li>6. El administrador cierra la sesión en el equipo asignado al usuario.</li> </ol>
Flujo Alterno:	<p><b>A1.</b> El administrador no está registrado en el sistema.</p> <ol style="list-style-type: none"> <li>1. El sistema no le permite ingresar.</li> <li>2. Se continua con el flujo principal.</li> </ol> <p><b>A2.</b> El usuario ingresado no existe.</p> <ol style="list-style-type: none"> <li>1. El administrador solicita al usuario su información y crea el usuario.</li> <li>2. Se continua con el flujo principal.</li> </ol>

	<p><b>Flujo Principal:</b></p> <ol style="list-style-type: none"> <li>1. El administrador ingresa el numero de identificacion del usuario.</li> <li>2. La aplicación envía el número de identificación del usuario a la base de datos.</li> <li>3. La base de datos retorna toda la información del usuario.</li> <li>5. El usuario sale de la sala.</li> <li>6. El administrador cierra la sesión en el equipo asignado al usuario.</li> </ol>									
	<p><b>Flujo Alterno:</b></p> <p>A1. El administrador no está registrado en el sistema.            1. El sistema no le permite ingresar.            2. Se continua con el flujo principal.            A2. El usuario ingresado no existe.            1. El administrador solicita al usuario su información y crea el usuario.            2. Se continua con el flujo principal.</p>									
	<table border="1"> <thead> <tr> <th><b>Nombre</b></th><th><b>GS-2 Ver Estado de la Sala</b></th></tr> </thead> <tbody> <tr> <td><b>Descripción:</b></td><td>En este proceso se encuentran las actividades que se deben realizar para poder mostrarle al usuario el estado de la sala que esta administrando.</td></tr> <tr> <td><b>Flujo Principal:</b></td><td>           1. Buscar en la base de datos las sesiones activas de la sala.            2. Recorrer cada una de las sesiones activas, para luego a su computador actualizarle el estado.            3. Actualizar el estado del computador, añadiendo el usuario de la session actual.            5. Mostrar graficamente el estado de cada computador.         </td></tr> <tr> <td><b>Flujo Alterno:</b></td><td>           A1. El administrador no está registrado en el sistema.            1. El sistema no le permite ingresar.            2. Se continua con el flujo principal.            A2. El usuario ingresado no existe.            1. El administrador solicita al usuario su información y crea el usuario.            2. Se continua con el flujo principal.         </td></tr> </tbody> </table>	<b>Nombre</b>	<b>GS-2 Ver Estado de la Sala</b>	<b>Descripción:</b>	En este proceso se encuentran las actividades que se deben realizar para poder mostrarle al usuario el estado de la sala que esta administrando.	<b>Flujo Principal:</b>	1. Buscar en la base de datos las sesiones activas de la sala. 2. Recorrer cada una de las sesiones activas, para luego a su computador actualizarle el estado. 3. Actualizar el estado del computador, añadiendo el usuario de la session actual. 5. Mostrar graficamente el estado de cada computador.	<b>Flujo Alterno:</b>	A1. El administrador no está registrado en el sistema. 1. El sistema no le permite ingresar. 2. Se continua con el flujo principal. A2. El usuario ingresado no existe. 1. El administrador solicita al usuario su información y crea el usuario. 2. Se continua con el flujo principal.	
<b>Nombre</b>	<b>GS-2 Ver Estado de la Sala</b>									
<b>Descripción:</b>	En este proceso se encuentran las actividades que se deben realizar para poder mostrarle al usuario el estado de la sala que esta administrando.									
<b>Flujo Principal:</b>	1. Buscar en la base de datos las sesiones activas de la sala. 2. Recorrer cada una de las sesiones activas, para luego a su computador actualizarle el estado. 3. Actualizar el estado del computador, añadiendo el usuario de la session actual. 5. Mostrar graficamente el estado de cada computador.									
<b>Flujo Alterno:</b>	A1. El administrador no está registrado en el sistema. 1. El sistema no le permite ingresar. 2. Se continua con el flujo principal. A2. El usuario ingresado no existe. 1. El administrador solicita al usuario su información y crea el usuario. 2. Se continua con el flujo principal.									
	<table border="1"> <thead> <tr> <th><b>Nombre</b></th><th><b>GS-2 Ver Estado del Equipo</b></th></tr> </thead> <tbody> <tr> <td><b>Descripción:</b></td><td>En este proceso se encuentran las actividades que se deben realizar para poder mostrar el estado específico de un equipo.</td></tr> <tr> <td><b>Flujo Principal:</b></td><td>           1. Se selecciona el computador para ver su estado.            2. Se busca la información del usuario, si el computador tiene una session activa.            3. Se muestra en el UI la información del computador y del usuario en caso de tener una session activa.         </td></tr> <tr> <td><b>Flujo Alterno:</b></td><td>           A1. El administrador no está registrado en el sistema.            1. El sistema no le permite ingresar.            2. Se continua con el flujo principal.            A2. El usuario ingresado no existe.            1. El administrador solicita al usuario su información y crea el usuario.            2. Se continua con el flujo principal.         </td></tr> </tbody> </table>	<b>Nombre</b>	<b>GS-2 Ver Estado del Equipo</b>	<b>Descripción:</b>	En este proceso se encuentran las actividades que se deben realizar para poder mostrar el estado específico de un equipo.	<b>Flujo Principal:</b>	1. Se selecciona el computador para ver su estado. 2. Se busca la información del usuario, si el computador tiene una session activa. 3. Se muestra en el UI la información del computador y del usuario en caso de tener una session activa.	<b>Flujo Alterno:</b>	A1. El administrador no está registrado en el sistema. 1. El sistema no le permite ingresar. 2. Se continua con el flujo principal. A2. El usuario ingresado no existe. 1. El administrador solicita al usuario su información y crea el usuario. 2. Se continua con el flujo principal.	
<b>Nombre</b>	<b>GS-2 Ver Estado del Equipo</b>									
<b>Descripción:</b>	En este proceso se encuentran las actividades que se deben realizar para poder mostrar el estado específico de un equipo.									
<b>Flujo Principal:</b>	1. Se selecciona el computador para ver su estado. 2. Se busca la información del usuario, si el computador tiene una session activa. 3. Se muestra en el UI la información del computador y del usuario en caso de tener una session activa.									
<b>Flujo Alterno:</b>	A1. El administrador no está registrado en el sistema. 1. El sistema no le permite ingresar. 2. Se continua con el flujo principal. A2. El usuario ingresado no existe. 1. El administrador solicita al usuario su información y crea el usuario. 2. Se continua con el flujo principal.									

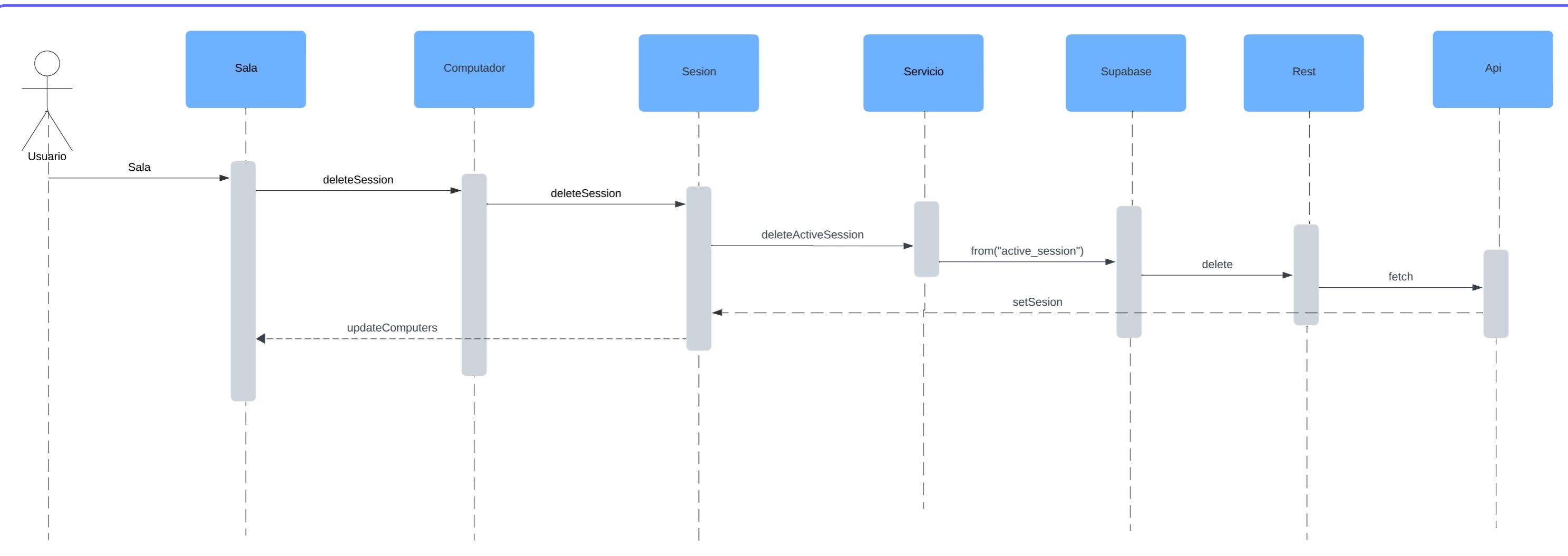
## Obtener Información de la Sala



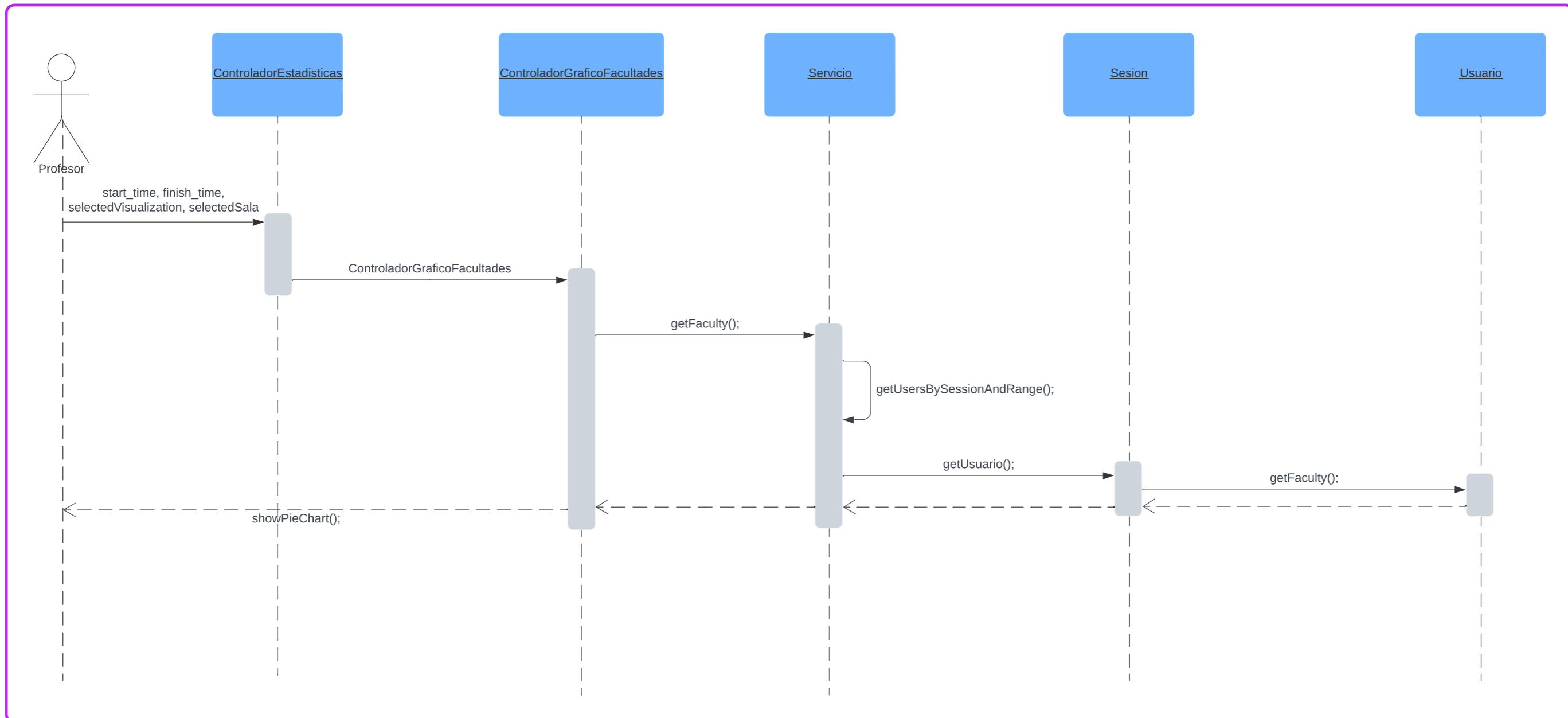
## Crear Nueva Sesión



## Eliminar Sesión Activa



## Crear Gráfico Facultades



## Crear Gráfico Meses

