República Bolivariana de Venezuela
Universidad de Los Andes
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas
Departamento de Computación
Materia: Ingeniería de Software

Sistema Médico Galeno

Integrantes:
Mudafar El Halabi
Juan Andrés Vivas

Descripción del Sistema

El Sistema Médico Galeno es un sistema modular dirigido a clínicas o consultorios medicos, el cual facilita la gestión de personal, gestión de cobros, historial de visita de pacientes, manejo de lista de espera de pacientes a ser atendidos e ínter-conexión entre los distintos entornos del espacio médico. Con el sistema se espera reducir el consumo de papel utilizado en la clínica, ya que serán almacenadas y manejadas en un entorno digital, teniendo así una mejor organización de las citas e historial medico entre otras cosas.

El sistema abarca cinco aspectos importantes de la clínica:

- Gestiona el historial del paciente, el cual cubre las visitas, exámenes realizados, notas del medico y un historial de pagos.
- Gestión básica de personal.
- Gestión de citas programadas y gestionadas por el sistema.
- Modulo de facturación para la gestión de cobros a los pacientes.

El sistema proporcionara una interfaz amigable y manejable para los usuarios.

Objetivos

El objetivo principal de este sistema es desarrollar un software que cubra las operaciones diarias de gestión de la clínica; ayudando a mejorar la calidad de servicio y atención al paciente, así como también llevar un control básico de cobros. Este sistema permite disponer de toda la información de forma rápida y precisa, reduciendo el tiempo de atención y costos de materiales.

Software usado en el proyecto

Frontend: Java
Backend: MySQL
Servidor de la aplicación: MySQL Server
Sistema Operativo: Windows, Linux, MacOSX

Requerimientos de Hardware para el proyecto

Procesador: Pentium4 o superior.
Velocidad de procesador: 2.00 Ghz o superior.
RAM: 1GB o mas.
Disco Duro: 20GB o mas.

Usuarios del Sistema

Los usuarios del sistema son los médicos, secretarias y administrador.

Médicos: Son los principales usuarios del sistema, ya que gestionan toda la parte medica. Su trabajo es la creación del historial medico, recipes y ordenes de exámenes para los pacientes.

Secretarias: Llevan el control de las citas, los cobros.

Administrador: Su trabajo es la asignación de roles dentro del sistema. Verifica también la integridad del sistema.

Requisitos del sistema

Funcionales

- Gestión de citas.
- Gestión de historial medico.
- Flexibilidad al momento de agregar nuevas funcionalidades.
- Gestión básica de cobros.
- Interfaz amigable para el usuario.
- Creación periódica de respaldos en caso de perdida de datos.

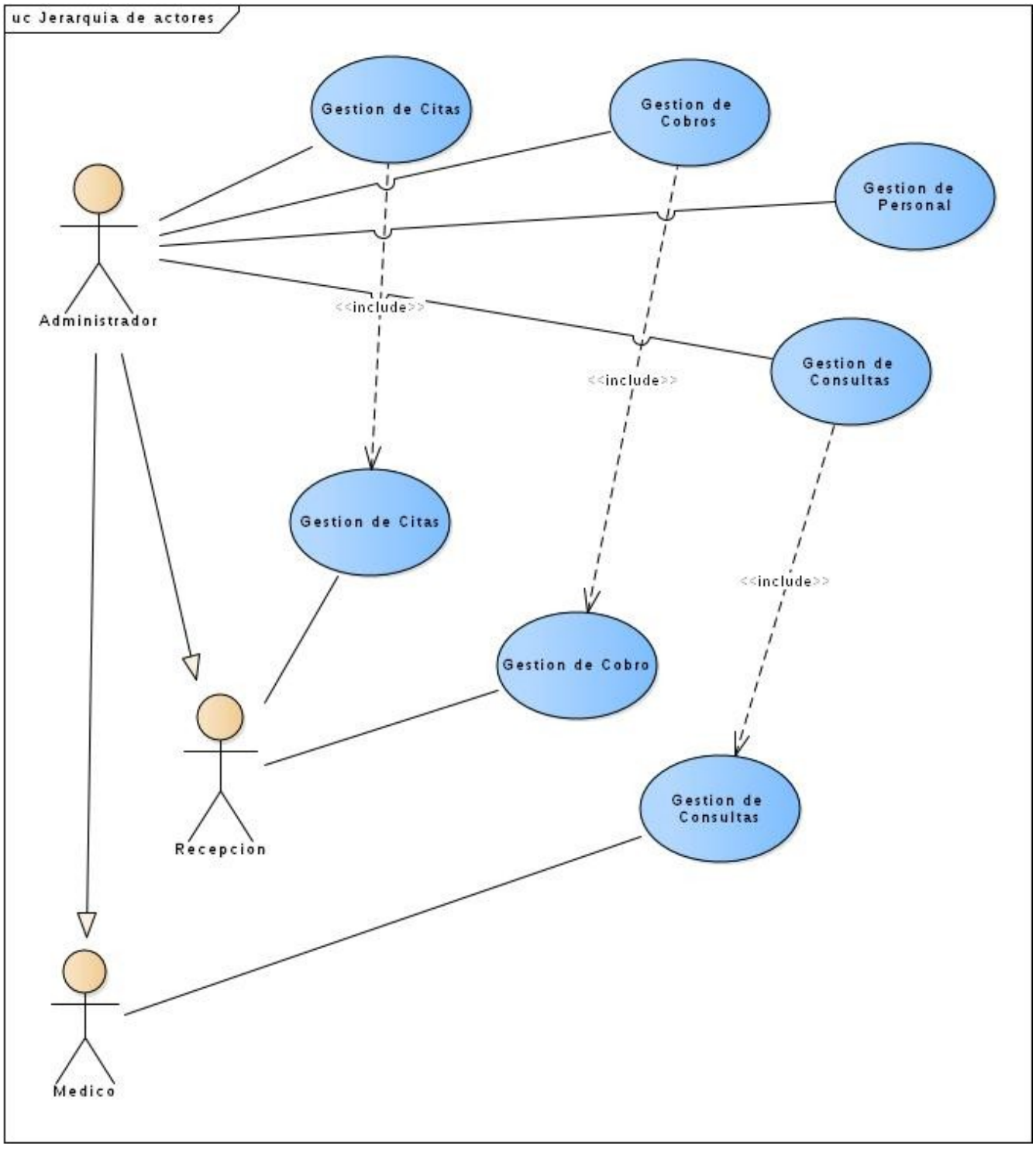
No funcionales

- Reducir el consumo de papel dentro del entorno de trabajo.
- Minimizar el tiempo del medico a la hora de crear o gestionar una historia medica.
- Subir la satisfacción de los pacientes.
- Facilita la corrección de errores en los datos cometidos por los médicos.
- Reducir el tiempo de espera de los pacientes.
- Minimizar los riesgos de perdida de información de pacientes.

Reglas de negocio

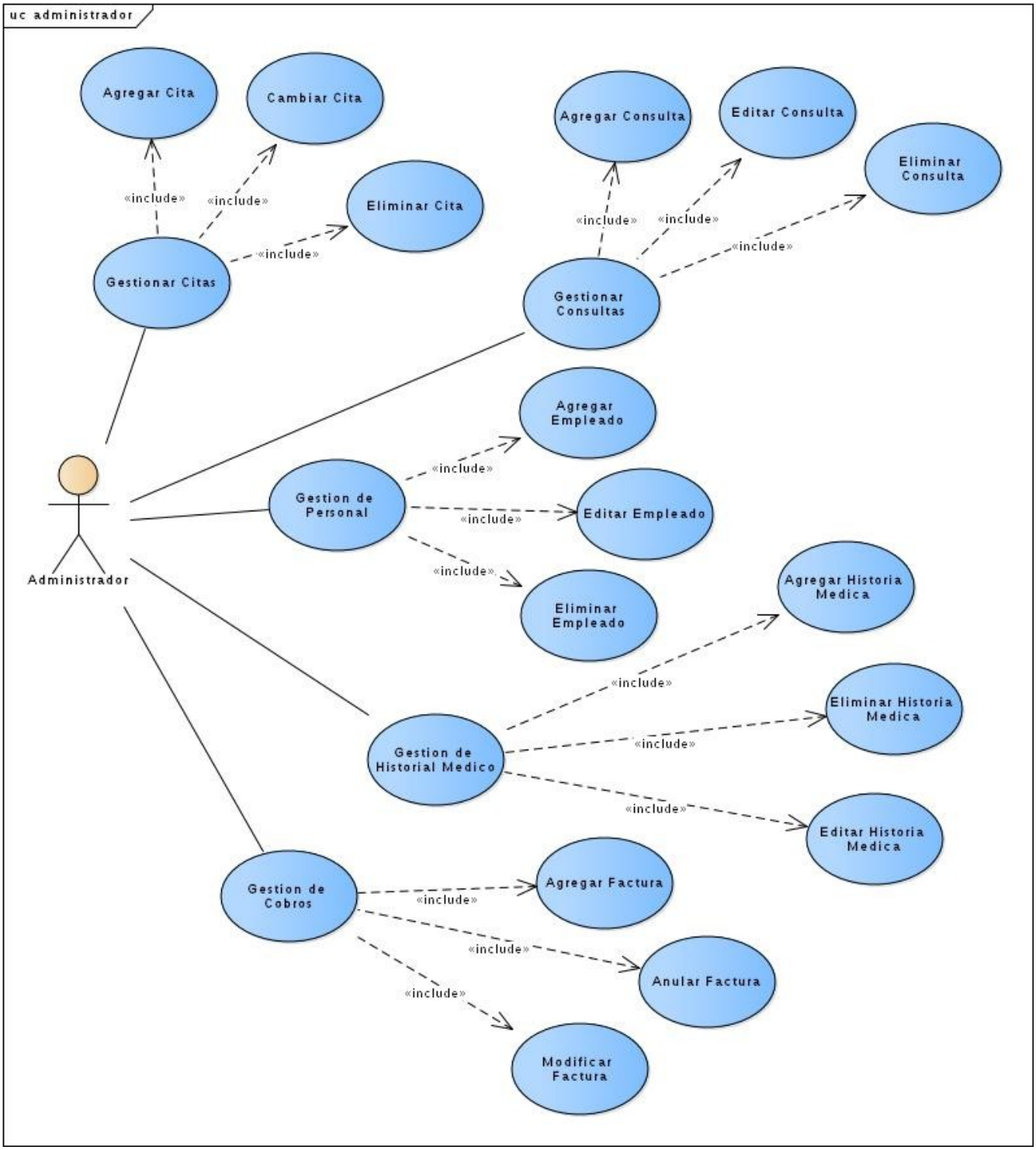
- La secretaria se limita a cobrar pagos en general.
- La secretaria puede crear nuevas citas, modificar citas y eliminarlas.
- El doctor puede crear nuevas historias y modificarlas.
- El doctor puede agregar ordenes medicas y recipes.
- El administrador es el único que puede crear empleados y asignar roles.
- El administrador es el único que puede eliminar y/o modificar una historia medica.
- El administrador tiene acceso a todo el sistema y puede desempeñar cualquier rol dentro del mismo.

Jerarquía de actores

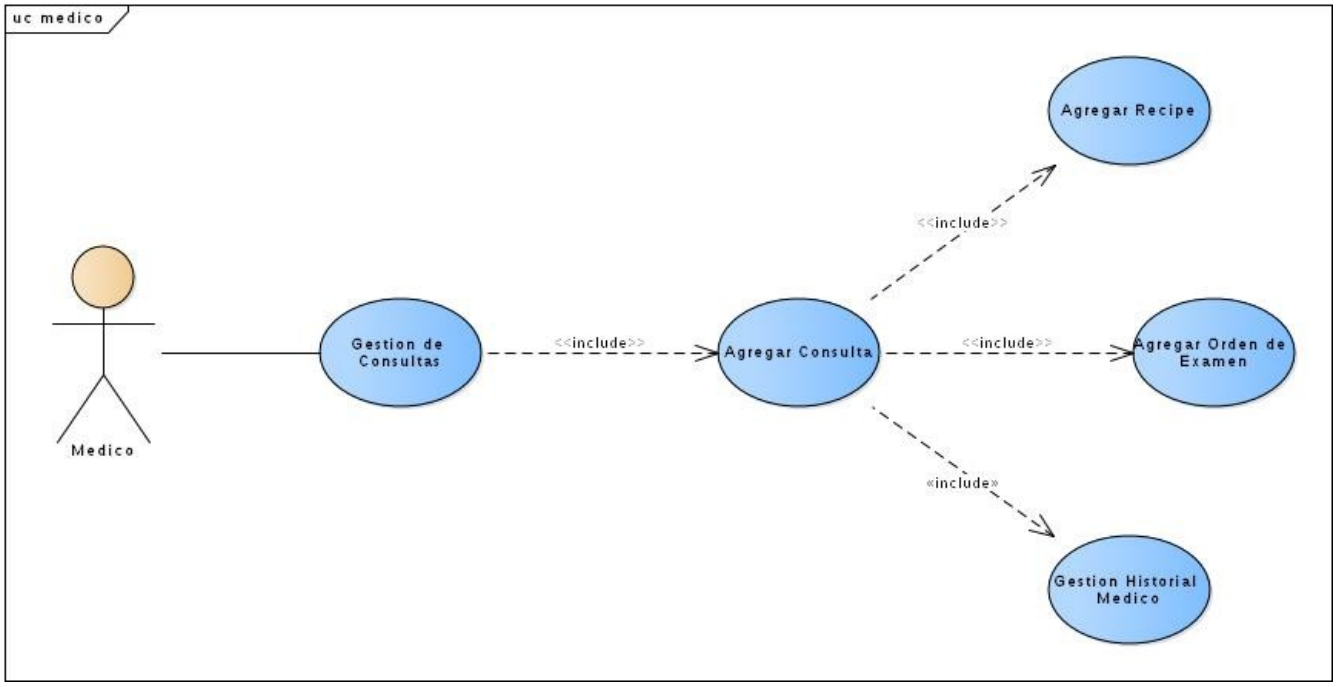


Casos de uso

Administrador



Médico


uc medico
Agregar Recipe
<<include>>
Medico
Gestion de Consultas
<<include>>
Agregar Consulta
<<include>>
Agregar Orden de Examen
«include»
Gestion Historial Medico

Recepción

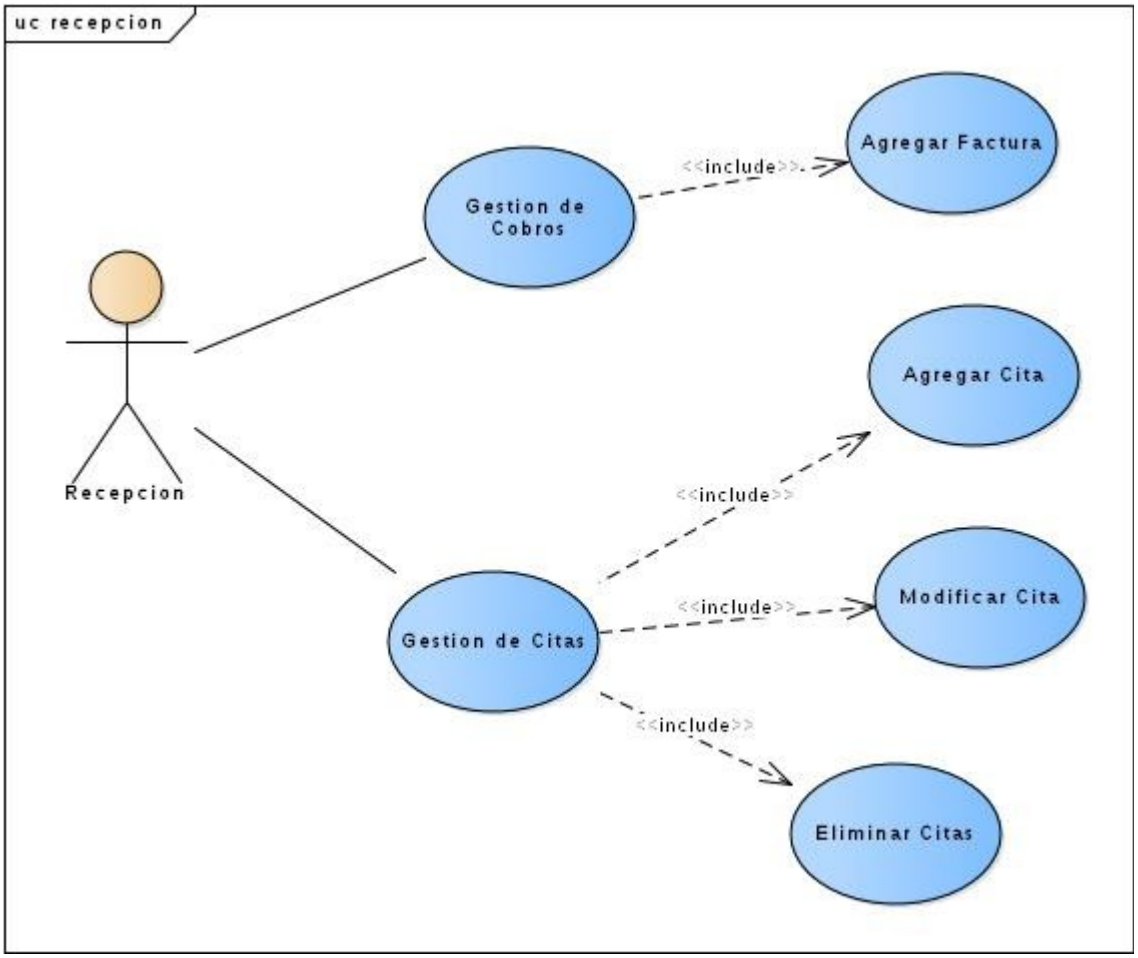

uc recepcion
Agregar Factura
<<include>>
Gestion de Cobros
Agregar Cita
<<include>>
Recepcion
Modificar Cita
<<include>>
Gestion de Citas
<<include>>
Eliminar Citas

Diagrama de actividades

Recepción

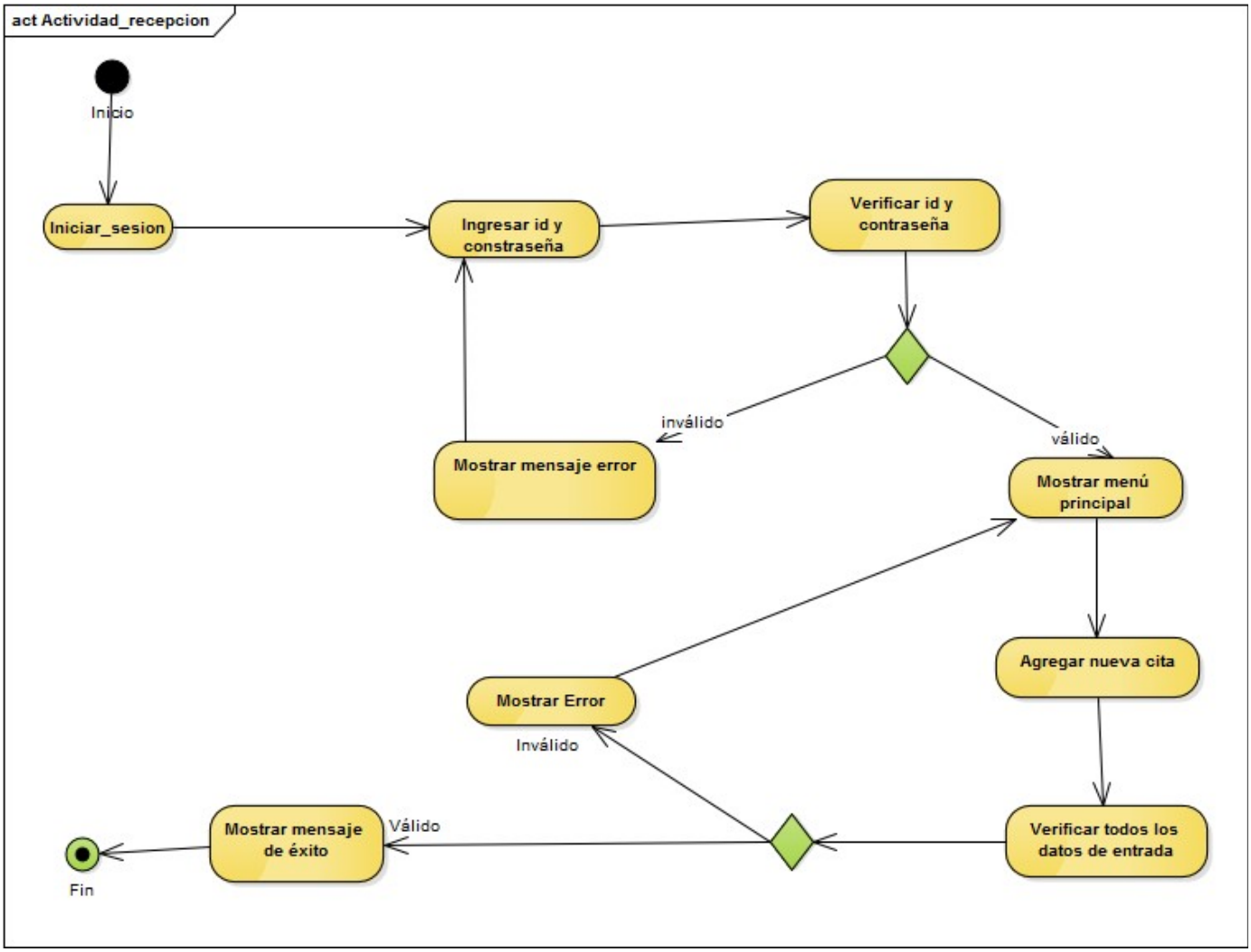


Diagrama de Clases

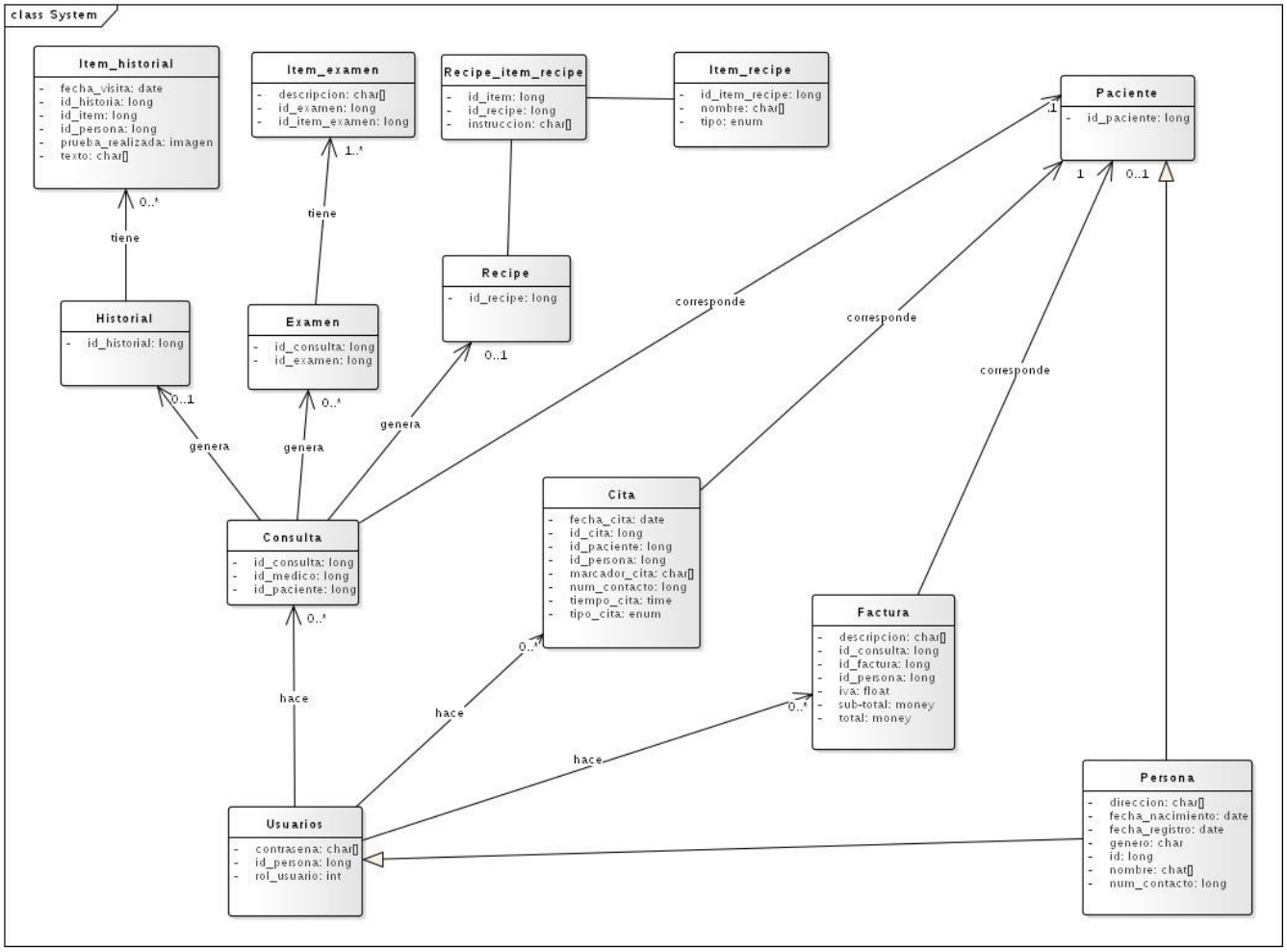


Tabla descriptiva de campos del Diagrama de Clases

Atributo	Descripción	Tipo	Entidad
id_persona	Id de la persona	Entero grande	Usuario
rol_usuario	Rol del usuario	Enumerador	Usuario
contraseña	Hash de la contraseña del usuario	Cadena caracteres	Usuario
id_consulta	Id de la consulta	Entero grande	Consulta
id_medico	Id del medico que hace la consulta	Entero grande	Consulta
id_paciente	Id del paciente atendido en la consulta	Entero grande	Consulta
id_cola	Id de la cola	Entero grande	Cola
id_paciente	Id del paciente en cola	Entero grande	Cola
id_medico	Id del medico que le corresponde la cita	Entero grande	Cola
fecha_cola	Fecha actual de la cola	Fecha	Cola
id_persona	Id del usuario quien gestiona la cola	Entero grande	Cola
posición	Posición correspondiente al paciente en la cola.	Entero sin signo	Cola
id_historial	Id del historial médico de un paciente	Entero grande	Historial
id_item	Id del item (genérico) del historial médico	Entero grande	item_historial
id_historia	Id del historial médico	Entero grande	item_historial
id_usuario	Id del usuario quien agrega el elemento	Entero grande	item_historial
fecha_visita	Fecha de la visita del paciente	Fecha	item_historial
texto	Descripción textual del item del historial	Cadena de caracteres	item_historial
prueba_realizada	Archivos y/o imágenes de pruebas realizadas al paciente	Binario/imagen	item_historial
id_examen	Id del examen que corresponde a un	Entero grande	Examen

	paciente		
id_consulta	Id de la consulta que corresponde al paciente	Entero grande	Examen
id_item_examen	Id del item del examen	Entero grande	item_examen
id_examen	Id del examen al cual corresponde el item	Entero grande	item_examen
descripcion	Descripción textual del examen	Cadena de caracteres	item_examen
id_recipe	Id del recipe asignado a un paciente	Entero grande	Recipe
id_item	Id del item del recipe	Entero grande	recipe_item_recipe
id_recipe	Id del recipe que corresponde dicho item	Entero grande	recipe_item_recipe
instruccion	Descripción textual de las instrucciones	Cadena de caracteres	recipe_item_recipe
id_item_recipe	Id del item que corresponde a un recipe	Entero grande	item_recipe
nombre	Nombre del item del recipe	Cadena de caracteres	item_recipe
tipo	Tipo del item:	Enumerador	item_recipe
id_cita	Id de la cita	Entero grande	Cita
id_paciente	Id del paciente que le corresponde la cita	Entero grande	Cita
id_persona	Id del usuario quien genera la cita	Entero grande	Cita
num_contacto	Numero de contacto del paciente	Entero grande	Cita
fecha_cita	Fecha de la cita	Fecha	Cita
tiempo_cita	Hora de la cita	Hora	Cita
marcador_cita	Descripción textual de la cita	Cadena de caracteres	Cita
tipo_cita	Tipo de la cita: control / primera vez etc	Enumerador	Cita
id_factura	Id de la factura asignada a un paciente	Entero grande	Factura
id_consulta	Id de la consulta que corresponde a la factura	Entero grande	Factura
id_persona	Id del usuario que genera la factura	Entero grande	Factura

descripción	Descripción textual de la factura	Cadena de caracteres	Factura
iva	Impuesto sobre el valor agregado	Real	Factura
total	Cantidad total a pagarse	Dinero	Factura
sub-total	Sub-total a pagar	Dinero	Factura
id_paciente	Id del paciente	Entero grande	Paciente
Nombre	Nombre de la persona	Cadena de caracteres	Persona
id	Id de la persona	Entero grande	Persona
genero	Genero de la persona	Carácter	Persona
fecha_nacimiento	Fecha de nacimiento de la persona	Fecha	Persona
direccion	Dirección de domicilio de la persona	Cadena de caracteres	Persona
fecha_registro	Fecha del registro de la persona	Fecha	Persona
num_contacto	Numero telefónico de contacto de la persona	Entero grande	Persona

Arquitectura del sistema
3 capas.

Capa de presentación

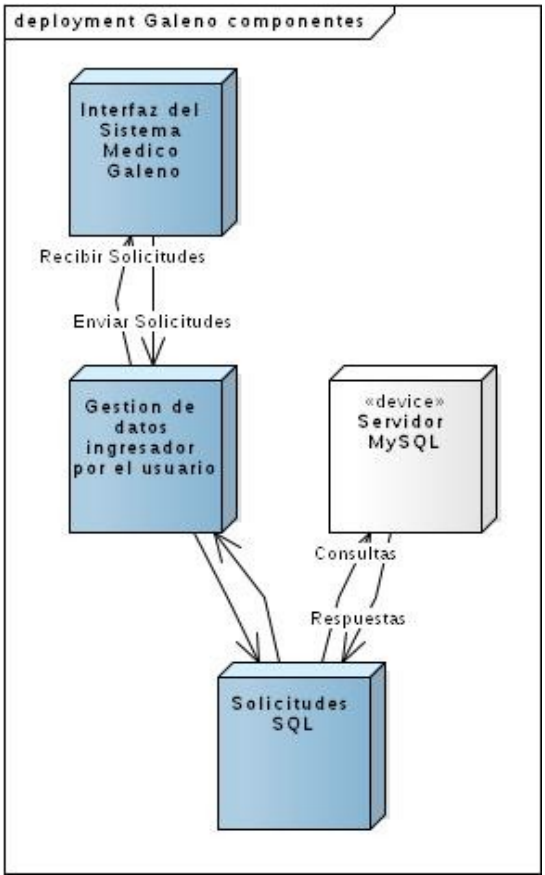
Se refiere a la presentación del programa frente al usuario, la cual debe cumplir su propósito con el usuario final, siendo una presentación fácil de usar y amigable. Las interfaces deben ser consistentes con la información dentro del software (Por ejemplo; en los formularios no debe haber más que lo necesario), donde deben ser tomados en cuenta los requerimientos del usuario; la capa de presentación se comunica solamente con la capa de la lógica de negocio.

Capa de lógica de negocio

En esta capa se encuentran los programas que son ejecutados, recibe las peticiones del usuario y posteriormente envía las respuestas tras el proceso. Esta capa es muy importante, pues es donde se establecen todas aquellas reglas que se tendrán que cumplir. Como se indicaba anteriormente, la capa de presentación tiene comunicación con la capa de lógica de negocio, ya que estas tienen que comunicarse para recibir las solicitudes y presentar los resultados.

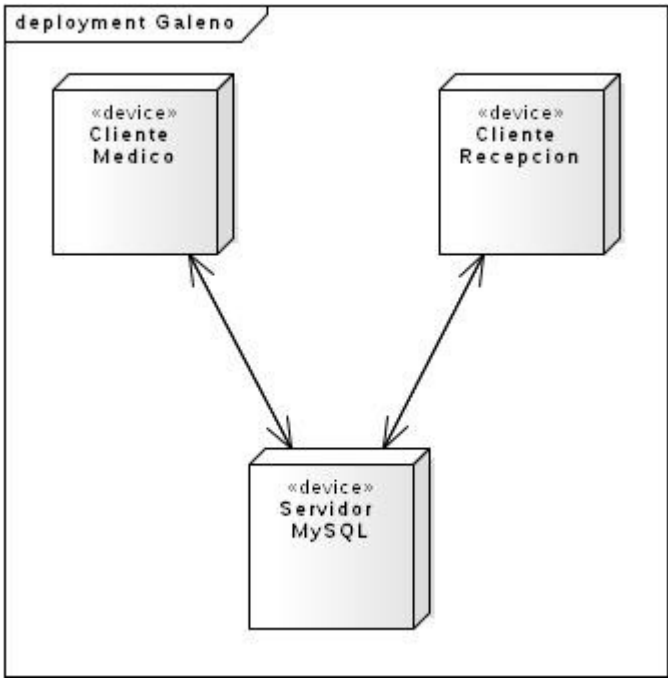
Capa de datos

Se encarga de realizar las transacciones con la base de datos y otros servicios o servidores para descargar o insertar información al sistema. La consistencia en los datos es sumamente importante, es decir, los datos que se ingresan o insertan deben ser precisos y consientes. Aquí se definen las consultas que vamos a realizar en la base de datos, o consultas para los reportes. La comunicación que tiene la capa de datos con la capa de lógica de negocio se refiere a que esta capa es la que enviá información a la capa de negocios para que sea procesada e ingresada en objetos según sea necesario (encapsulamiento).



Arquitectura Cliente/servidor
2 niveles.

Clientes: son los equipos que interactúan con la interfaz para utilizar los servicios.
Servidor: Provee el servicio de manejar la base de datos.



Interfaz de la Aplicación

Interfaz Recepción

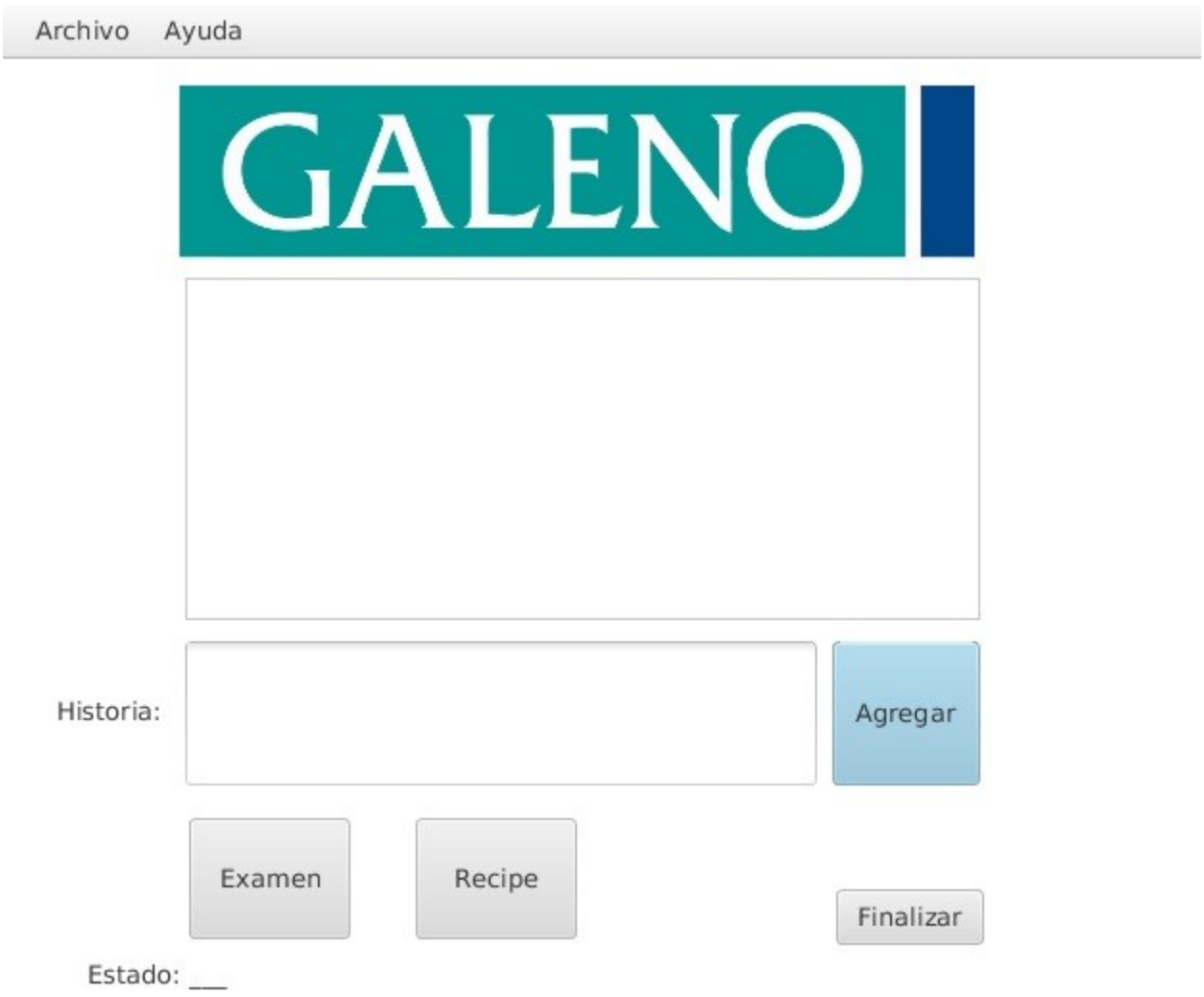
Interfaz Médico

Interfaz Citas

Interfaz Recipe

Interfaz Exámenes

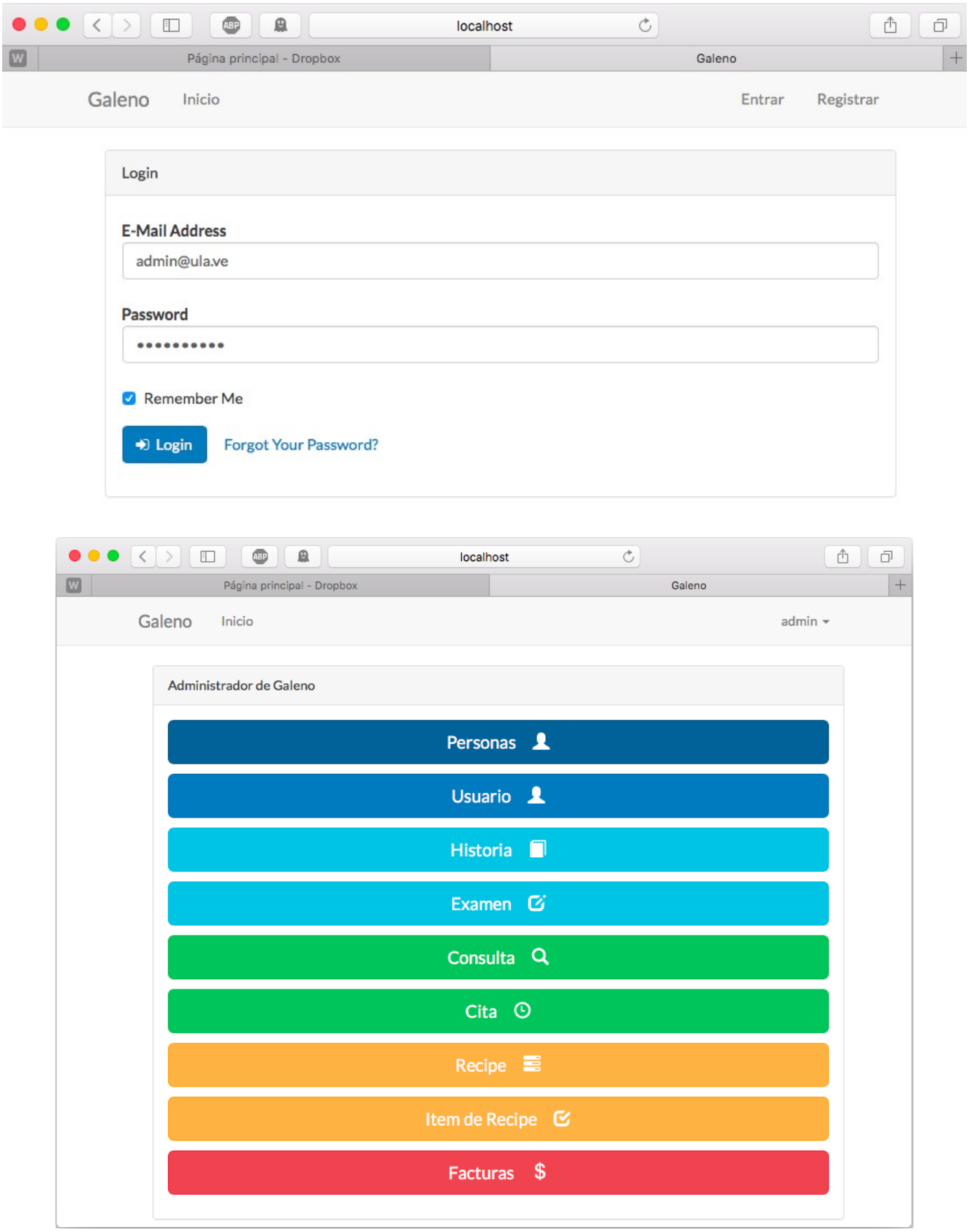
Interfaz Consulta

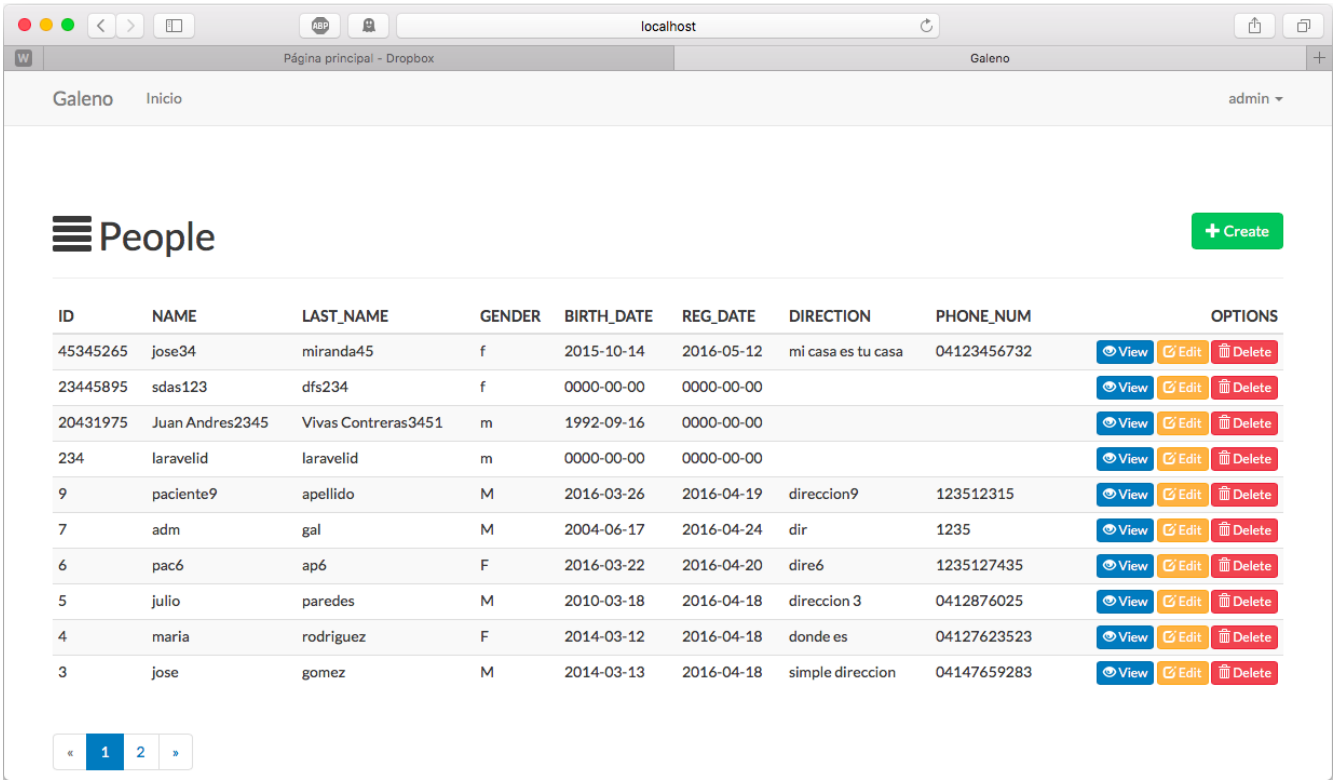


Interfaz Factura

Interfaz de Ingreso

Interfaz Administrador




localhost
Página principal - Dropbox
Galeno
Galeno
Inicio
admin
People
+ Create
ID NAME LAST_NAME GENDER BIRTH_DATE REG_DATE DIRECTION PHONE_NUM OPTIONS
45345265 jose34 miranda45 f 2015-10-14 2016-05-12 mi casa es tu casa 04123456732 View Edit Delete
23445895 sdas123 dfs234 f 0000-00-00 0000-00-00 View Edit Delete
20431975 Juan Andres2345 Vivas Contreras3451 m 1992-09-16 0000-00-00 View Edit Delete
234 laravelid laravelid m 0000-00-00 0000-00-00 View Edit Delete
9 paciente9 apellido M 2016-03-26 2016-04-19 direccion9 123512315 View Edit Delete
7 adm gal M 2004-06-17 2016-04-24 dir 1235 View Edit Delete
6 pac6 ap6 F 2016-03-22 2016-04-20 dire6 1235127435 View Edit Delete
5 julio paredes M 2010-03-18 2016-04-18 direccion 3 0412876025 View Edit Delete
4 maria rodriguez F 2014-03-12 2016-04-18 donde es 04127623523 View Edit Delete
3 jose gomez M 2014-03-13 2016-04-18 simple direccion 04147659283 View Edit Delete
« 1 2 »

Implementación

Prueba de unidad

Prueba de la clase Citas: Para la prueba de unidad de esta clase, validamos que todos los setter cumplieran y validaran los datos de forma correcta.

Set id Código

```
  public void set_id(Long _id)
    {
        if(_id < new Long(0))
            this._id = new Long(0);

        else
            this._id = _id;
    }
```

Prueba

```
  @org.junit.Test
    public void set_id() throws Exception
    {
        app.set_id(new Long(-1));
        assertEquals(new Long(0), app.get_id());

    }
```

Set patient_id código:

```
  public void set_patient_id(Long _patient_id)
    {
        if(_patient_id < new Long(0))
            this._patient_id = new Long(0);
        else
            this._patient_id = _patient_id;
    }
```

Prueba:

```
  @org.junit.Test
    public void set_patient_id() throws Exception
    {
        app.set_patient_id(new Long(-1));
        assertEquals(new Long(0), app.get_patient_id());
    }
```

Set user_id código:

```
public void set_user_id(Long _user_id)
   {
       if(_user_id < new Long(0))
           this._user_id = new Long(0);
       else
           this._user_id = _user_id;
}
```

Prueba:

```
   @org.junit.Test
   public void set_user_id() throws Exception
   {
       app.set_user_id(new Long(-1));
       assertEquals(new Long(0), app.get_user_id());
   }
```

Set doctor_id código

```
  public void set_doctor_id(Long _doctor_id)
   {
       if(_doctor_id < new Long(0))
           this._doctor_id = new Long(0);
       else
           this._doctor_id = _doctor_id;
   }
```

Prueba

```
  @org.junit.Test
   public void set_doctor_id() throws Exception
   {
       app.set_doctor_id(new Long(-1));
       assertEquals(new Long(0), app.get_doctor_id());
   }
```

Set date código

```
  public void set_date(Date _date)
   {
       Date d = Date.valueOf(LocalDate.now());

       if(d.after(_date))
```

```
            this._date = d;
        else
            this._date = _date;
  }
```

Prueba

```
   @org.junit.Test
    public void set_date() throws Exception
    {
        app.set_date(Date.valueOf("2016-03-26"));
        assertEquals(Date.valueOf(LocalDate.now()), app.get_date());

//        app.set_date(Date.valueOf("2016-03-26"));
//        assertEquals(Date.valueOf("2016-05-14"), app.get_date());
    }
```

Set Time codigo

```
  public void set_time(Time _time)
    {
                        if(_time.before(Time.valueOf("20:00:00"))   &&
_time.after(Time.valueOf("08:00:00")))
            this._time = _time;
        else
            this._time = Time.valueOf("08:00:00");
    }
```

Prueba

```
  @org.junit.Test
    public void set_time() throws Exception
    {
        app.set_time(Time.valueOf("07:00:00"));
        assertEquals(Time.valueOf("08:00:00"),app.get_time());

//        app.set_time(Time.valueOf("07:00:00"));
//        assertEquals(Time.valueOf("09:00:00"),app.get_time());

    }
```

Prueba de caja blanca

Para la prueba de caja blanca, partimos de conocer todo el código de la clase Person de nuestra aplicación y probamos los setter de la clase y su correcto funcionamiento. Para los id, verificamos que si son negativos, se les asigne el id 0 y para las cadenas de caracteres como nombres, dirección, teléfono si están vaciás, se le agrega la cadena “empty”.

Setter de la clase Person

```
public void set_id(Long _id)
    {
        if(_id < new Long(0))
            this._id = new Long(0);
        else
            this._id = _id;
    }
public void set_name(String _name)
    {
        if(_name.isEmpty())
            this._name = new String("empty");

        else if(_name == null)
            this._name = new String("null");

        else
            this._name = _name;

    }

    public void set_last_name(String _last_name)
    {
        if(_last_name.isEmpty())
            this._last_name = new String("empty");
        else
            this._last_name = _last_name;
    }

    public void set_gender(String _gender)
    {
        if(_gender.isEmpty())
            this._gender = new String("M");
        else
            this._gender = _gender;
    }


    public void set_direction(String _direction)
    {
        if(_direction.isEmpty())
            this._direction = new String("empty");
        else
            this._direction = _direction;
    }
```

```
    public void set_phone_num(String _phone_num)
    {
        if(_phone_num.isEmpty())
            this._phone_num = new String("empty");
        else
            this._phone_num = _phone_num;
    }
```

Batería de pruebas:

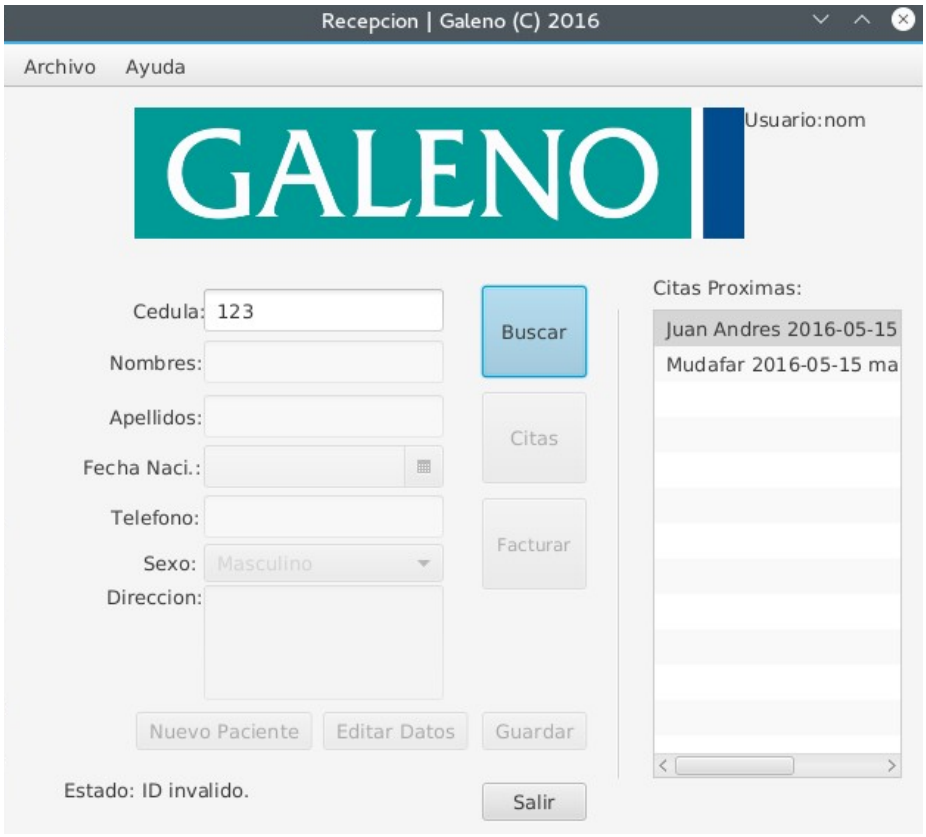
```
public class PersonTest
{

    Person per = new Person();

    @Test
    public void set_id() throws Exception
    {
        per.set_id(new Long(-1));
        assertEquals(new Long(0), per.get_id());
    }

    @Test
    public void set_name() throws Exception
    {
        per.set_name("");
        assertEquals(new String("empty"), per.get_name());

//        per.set_name(null);
//        assertEquals(new String("null"),per.get_name());

    }

    @Test
    public void set_last_name() throws Exception
    {
        per.set_last_name("");
        assertEquals(new String("empty"),per.get_last_name());
    }

    @Test
    public void set_gender() throws Exception
    {
        per.set_gender("");
        assertEquals(new String("M"),per.get_gender());
    }
```

```
    @Test
    public void set_direction() throws Exception
    {
        per.set_direction("");
        assertEquals(new String("empty"),per.get_direction());
    }

    @Test
  public void set_phone_num() throws Exception
    {
        per.set_phone_num("");
        assertEquals(new String("empty"),per.get_phone_num());
    }

}
```

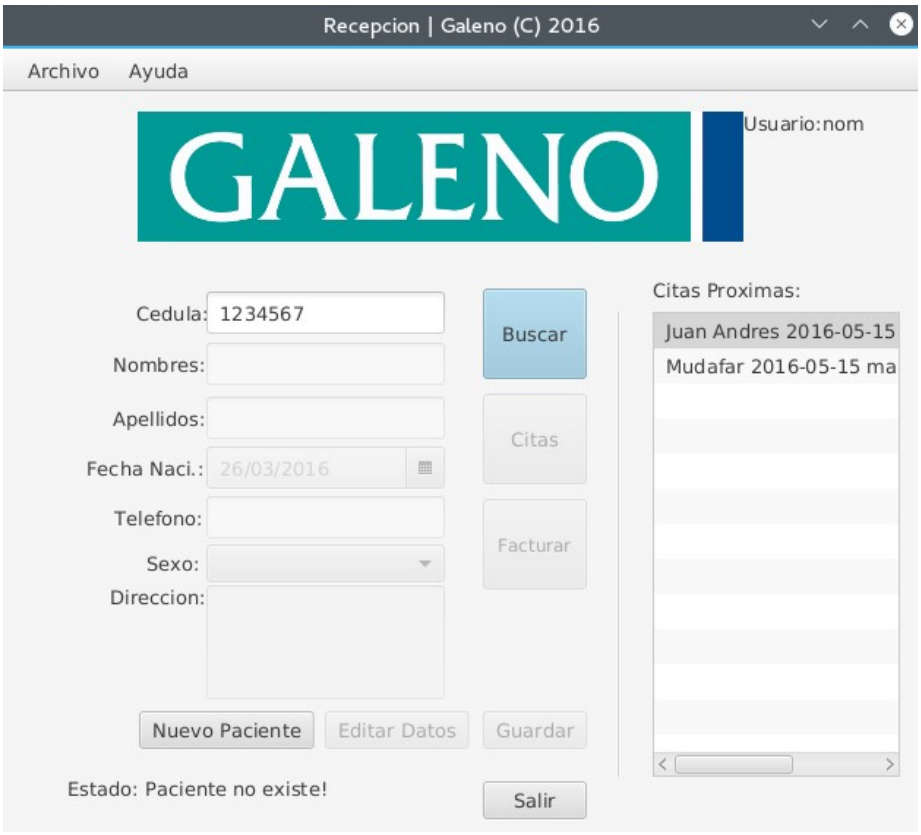
Prueba de caja negra

Para la prueba de caja negra, probaremos las validaciones de los campos de la interfaz de recepción, conociendo previamente que las cedulas están comprendidas en un rango de 500.000 a 99.999.999, nombres y apellidos no permiten caracteres numéricos ni especiales, teléfono no permite caracteres alfabéticos.

Validación de cedula en interfaz

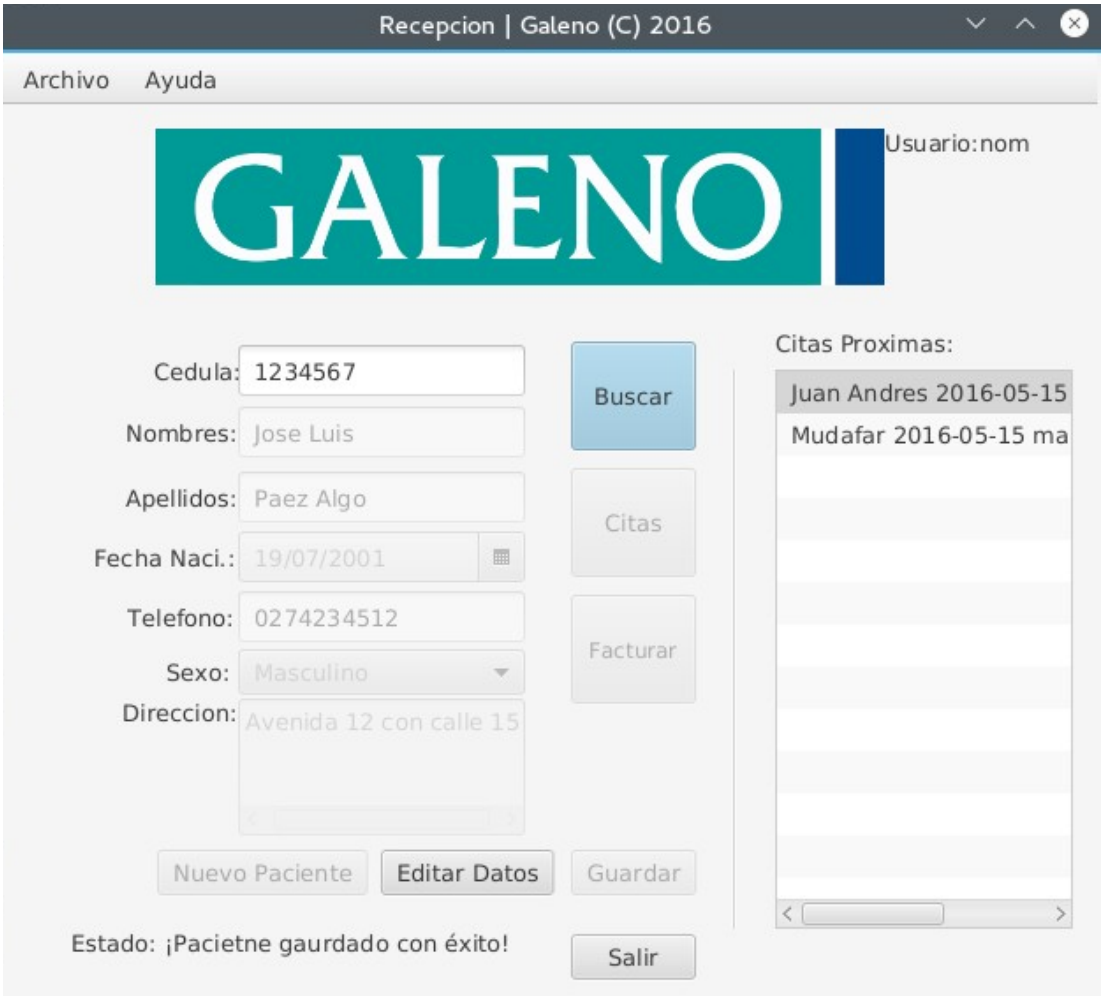


Se verifica que al intentar buscar cedulas menores o mayores dentro del rango validado, el Estado notifica que el ID es invalido.



Al poner una cedula valida, el Estado del programa cambia y te permite crear un nuevo paciente.

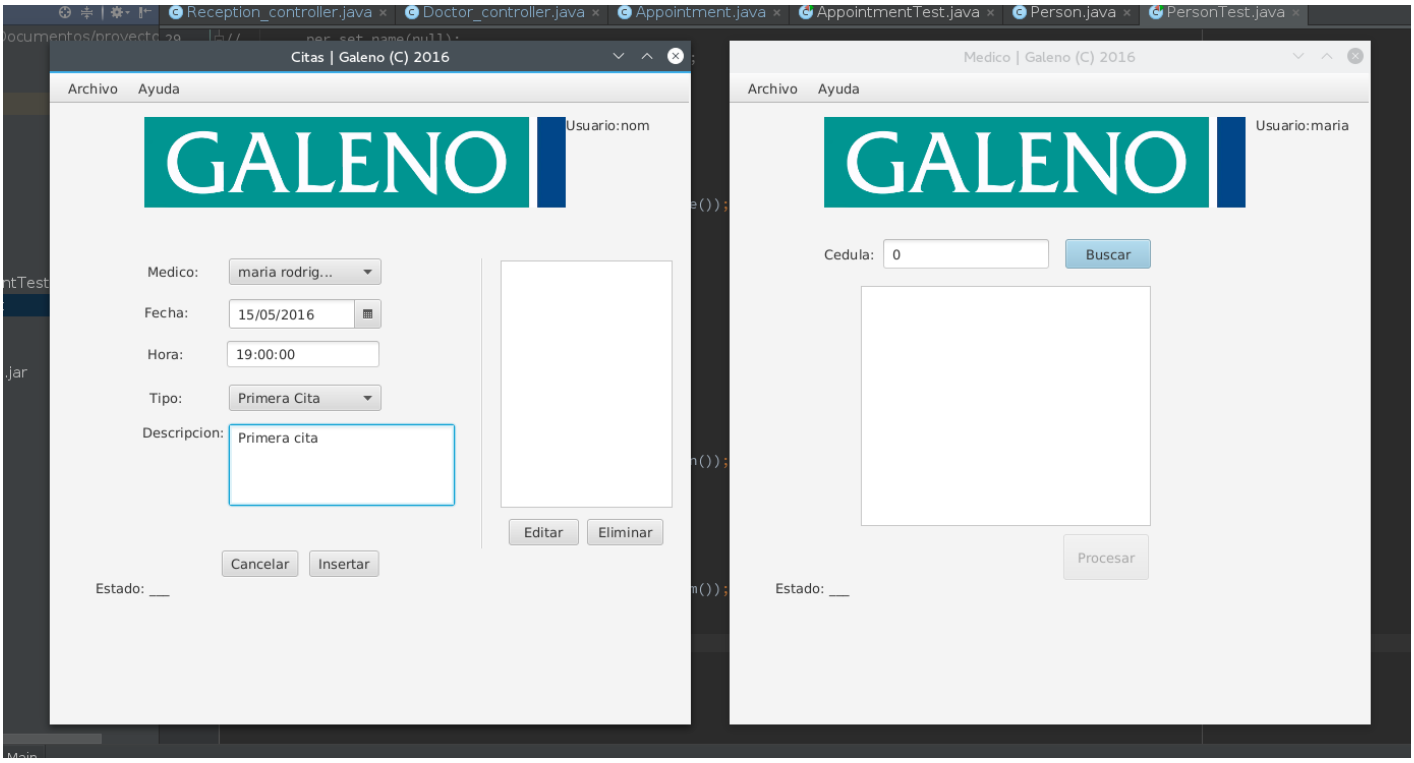
Para la siguiente prueba se agrego el paciente y se valido que en los campos de Nombres y Apellidos no permitió agregar caracteres especiales ni numéricos, para el campo Teléfono se verifico también que no permite el ingreso de caracteres alfabéticos ni especiales. Al finalizar se guarda el paciente con éxito.



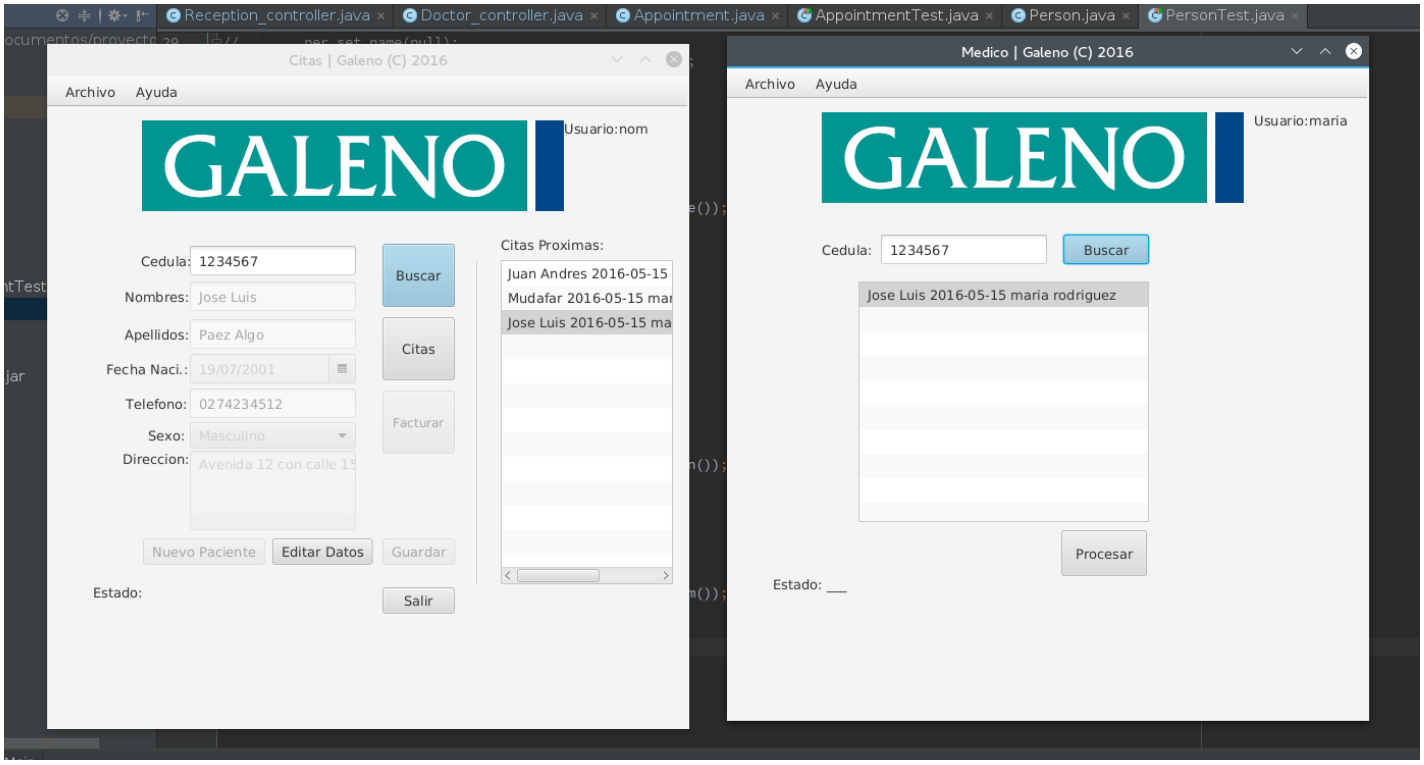
Prueba de integración de componentes

Nuestra prueba de integración consiste en una vez agregado un paciente, se procede a crearle una cita y asignarla a un doctor. Verificaremos que este proceso se lleve a cabo correctamente.

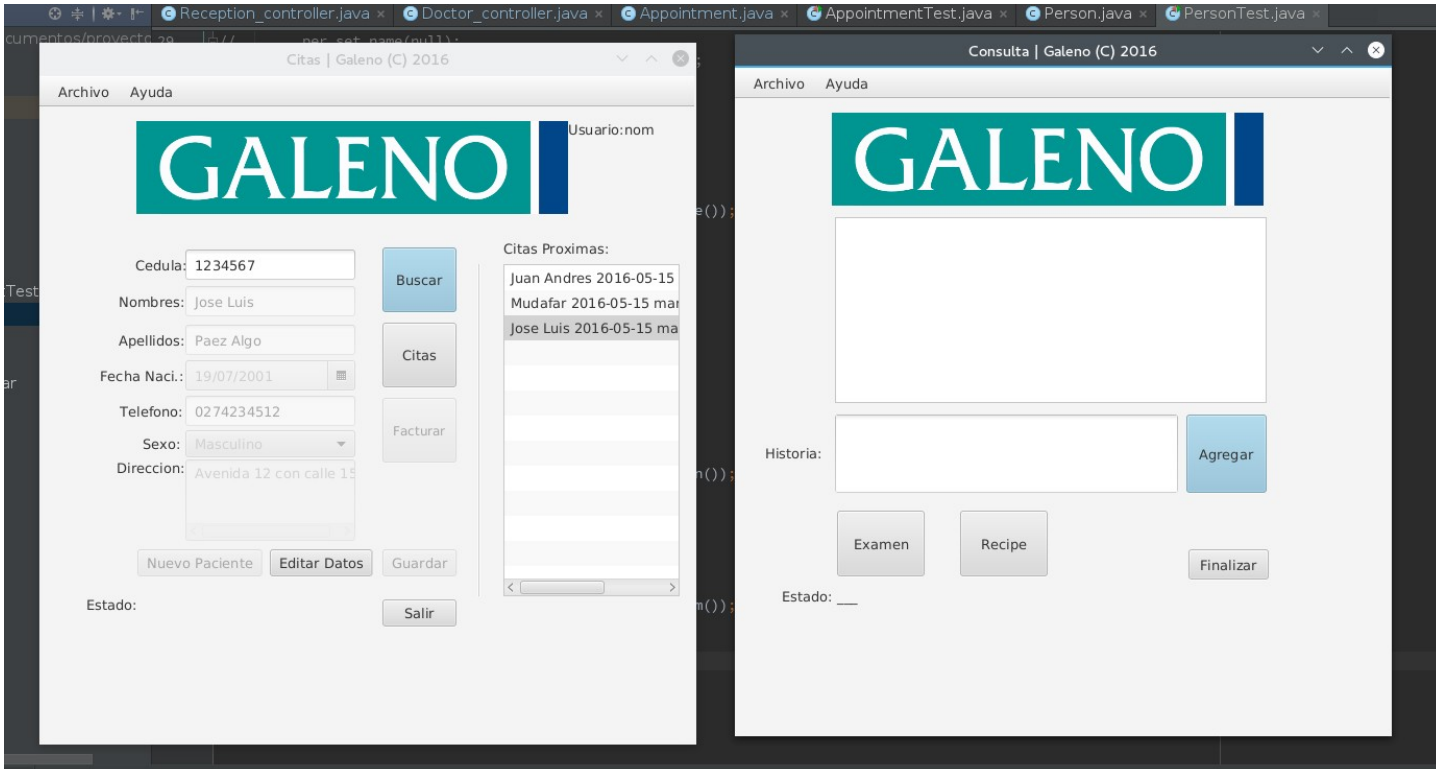
Luego de crear el paciente en la prueba de caja negra, procedemos a asignarle la cita mientras se monitorea la interfaz del doctor.



Se procede a insertar la cita y luego verificamos que la misma aparezca en la interfaz del doctor.



Como ultimo paso se procede a crear la consulta del paciente.



Notas importantes sobre la implementación del Sistema Medico Galeno:

Reutilizar Componentes

- Nivel de abstracción: Patrones de la arquitectura de tres capas con cliente servidor.
- Nivel de objeto: Uso de objetos de las bibliotecas de Java.SQL, Junit, JavaFX, Java.Util.
- Nivel de Componentes: Uso del IDE para el desarrollo de la aplicación IntelliJ IDEA, componente de la interfaz JavaFX y el “FXML Scene Editor”, el componente para conectar con la base de datos JBCD.

Licenciamiento

- MySql, Free Open Source Software (“FOSS”) → (GPL V2).
- Java, JavaFX → BCL for JAVA SE.
- Galeno → LGPL (Lessly Restricted GPU License)