

Manual Técnico Camellap



Integrantes:

Pablo Esteban Pachón Vega

David Steven Pinzon Hernadez

Jaider Fernando Rodriguez Sanabria

Docente:

Nestor German Bolivar Pulgarin

Contenido:

Descripción	3
Modo de uso	3
Crear un nuevo evento	3
Visualizar y editar un evento	4
Crear y visualizar personal	4
Crear y visualizar a un contratante	5
Crear y visualizar a un material de inventario	6
Ejemplo de uso	6
Diseño y Arquitectura	7
Elección de colores, logos y mockups	7
Colores	7
Logos	7
Mockups	7
Arquitectura	8
Diagramas	9
Diagrama de clases	9
Diagrama de casos de uso	10
Diagramas de secuencia	11
Descripción del código	13
Lógica del programa	13
Interfaz gráfica	16
Persistencia de datos	19

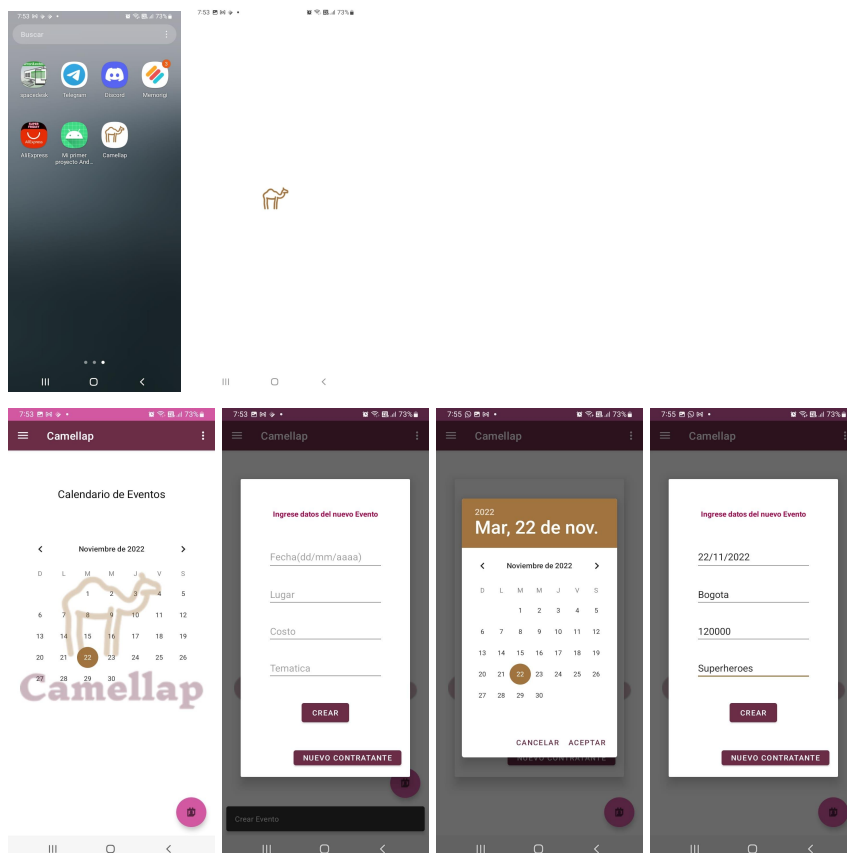
Descripción

El proyecto que a continuación se va a presentar consiste en una aplicación móvil, que tiene la funcionalidad de organizar la información en una empresa de eventos, como lo son el inventario, los datos de los eventos y del personal, y las finanzas. Esta aplicación se hace con el fin adicional de permitirle a esta empresa la utilización de las herramientas tecnológicas que en la actualidad se pueden emplear. Además esta aplicación busca tener una interfaz sencilla para el usuario, respetando el esquema de colores predefinido por la empresa a la que se le desarrolla este producto.

Modo de uso

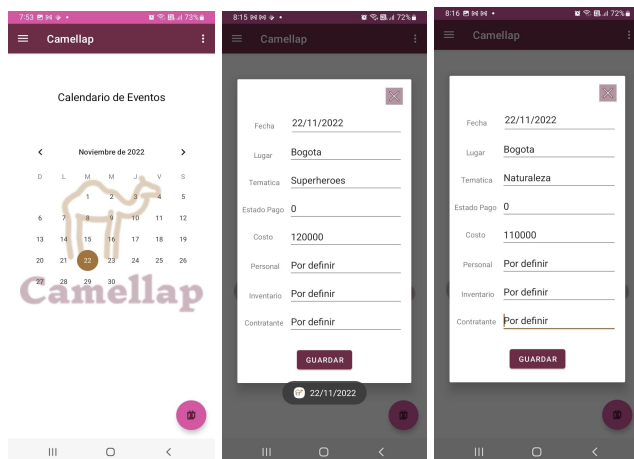
Crear un nuevo evento

Para crear un nuevo evento hay que realizar los siguientes pasos. En primer lugar abrir la aplicación, posteriormente en la pantalla que aparece inmediatamente se abre la aplicación se debe buscar un botón ubicado en la parte inferior derecha, y presionarlo. Al presionarlo se abrirá un cuadro de diálogo en el que hay unos espacios para llenar con los datos que desea guardar. Por último, rellenos los campos de información, debe presionar en el botón CREAR. (Nota: Para poner la fecha deseada debe de presionar sobre al campo de fecha dos veces para que salga un calendario donde se puede elegir la fecha.)



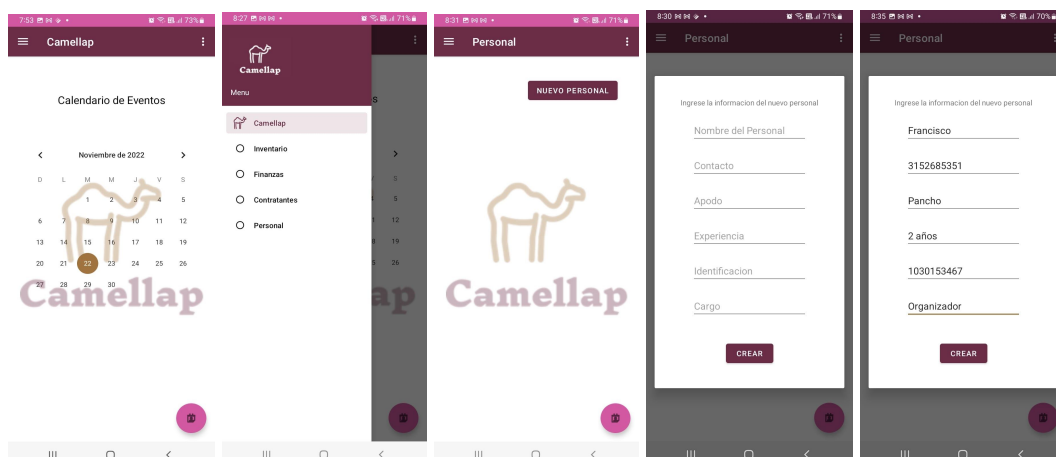
Visualizar y editar un evento

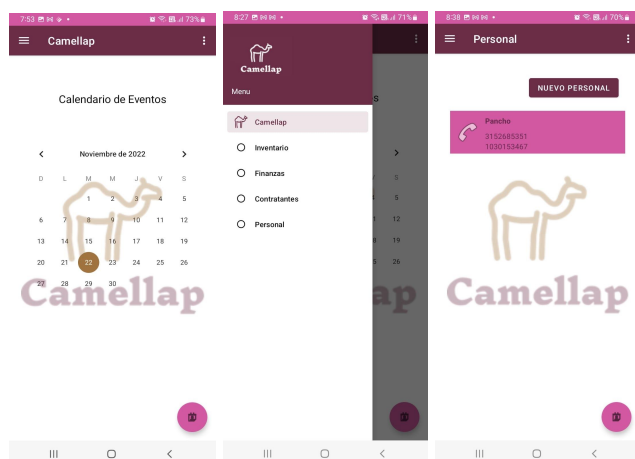
Una vez abierta la aplicación se debe presionar encima de una fecha deseada en el calendario que aparece inicialmente, si en la fecha seleccionada hay un evento entonces se abrirá un cuadro de diálogo en el que se mostrará toda la información del evento. Si desea solo observar la información, una vez termine de hacerlo debe darle en la x que se encuentra en la esquina superior derecha, por otro lado si desea editar la información de evento debe cambiarla directamente en los campos donde aparece la información actual del evento. Una vez editado debe darle en el botón GUARDAR. Si desea comprobar el cambio deberá volver a realizar el proceso de elegir una fecha en el calendario.



Crear y visualizar personal

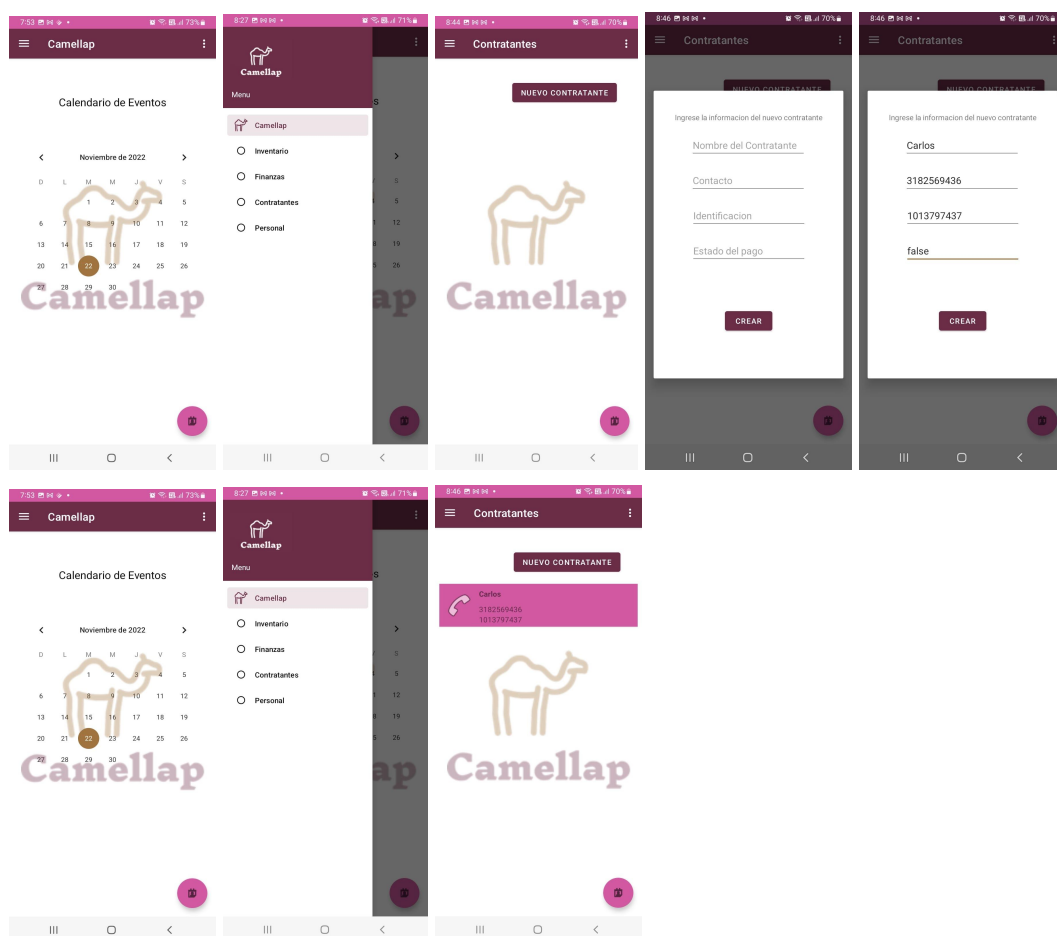
Para crear un nuevo personal, y iniciada la aplicación, y estando en la pantalla principal, es necesario presionar sobre las tres rayas blancas ubicadas en la parte superior izquierda. Se abrirá una lista de navegación entre fragmentos, seleccione personal. Una vez en la pantalla de personal, presione sobre el botón NUEVO PERSONAL. Al presionarlo se abrirá un cuadro de diálogo en el que se encontrarán unos campos en los que debe llenar la información. Ya con la información en los campos debe presionar en el botón CREAR. Para visualizar la información del personal existente, en cuanto a apodo, identificación y número de contacto, solo basta con abrir la pantalla personal, y se visualiza debajo del botón de NUEVO PERSONAL.





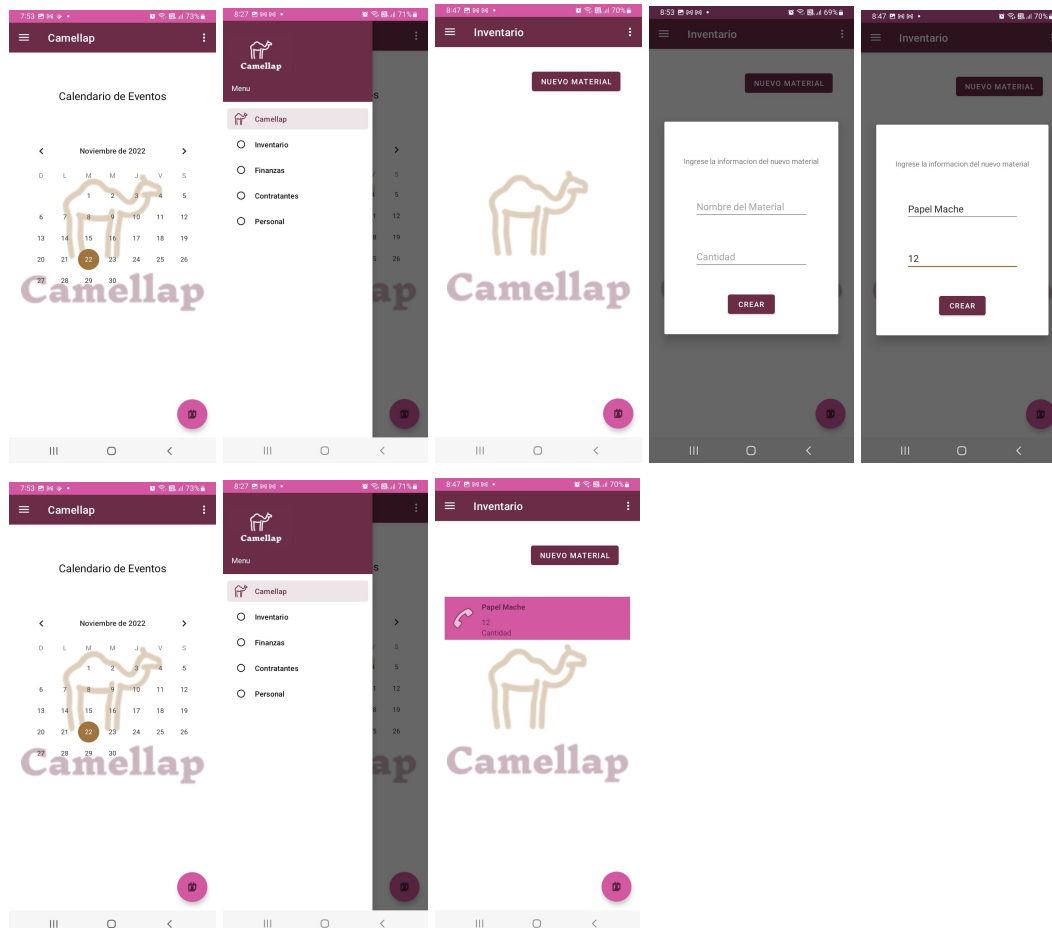
Crear y visualizar a un contratante

Para crear el contratante y visualizarlo se sigue el mismo procedimiento que en la sección anterior, con la diferencia de que se selecciona la pantalla contratantes, y se presiona el botón que dice NUEVO CONTRATANTE, lo que nos muestra un cuadro de diálogo específico para crear contratantes.



Crear y visualizar a un material de inventario

Para crear el contratante y visualizarlo se sigue el mismo procedimiento que en la sección anterior, con la diferencia de que se selecciona la pantalla inventario, y se presiona el botón que dice NUEVO MATERIAL, lo que nos muestra un cuadro de diálogo específico para crear inventario.



Ejemplo de uso

Para comprobar su funcionalidad se le pidió al cliente que revisara una versión incompleta de la aplicación. El cliente realizó el siguiente comentario:

En el aspecto estético la aplicación está tal cual me la mostraron y eso me gusto.

Veo que puede ser algo confusa pero es porque no le tengo la maña aun a usarla.

Me gustaría que pudiera guardar los whatsapp de la gente para poderles escribir desde ahí pero me imagino que eso es como difícil.

Me gusta que puedo guardar mejor la información en el mismo lugar y ojalá la aplicación mejore con el tiempo.

Diseño y Arquitectura

Elección de colores, logos y mockups

Colores

Los colores se eligieron a partir del esquema de colores de la empresa del cliente, como requerimiento del proyecto. Además se ha asociado cada color a los elementos de la interfaz gráfica de la aplicación.

#000000, Negro - Texto, sombras.

#ffffff, blanco - Fondos o base.

#A1743F, Dorado - Camello.

#6B2C45, Morado/ vino tinto - Nombre.

#D259A1 Rosa/magenta - Eslogan.

#B595A2 Rosa pálido - Fondos o bases

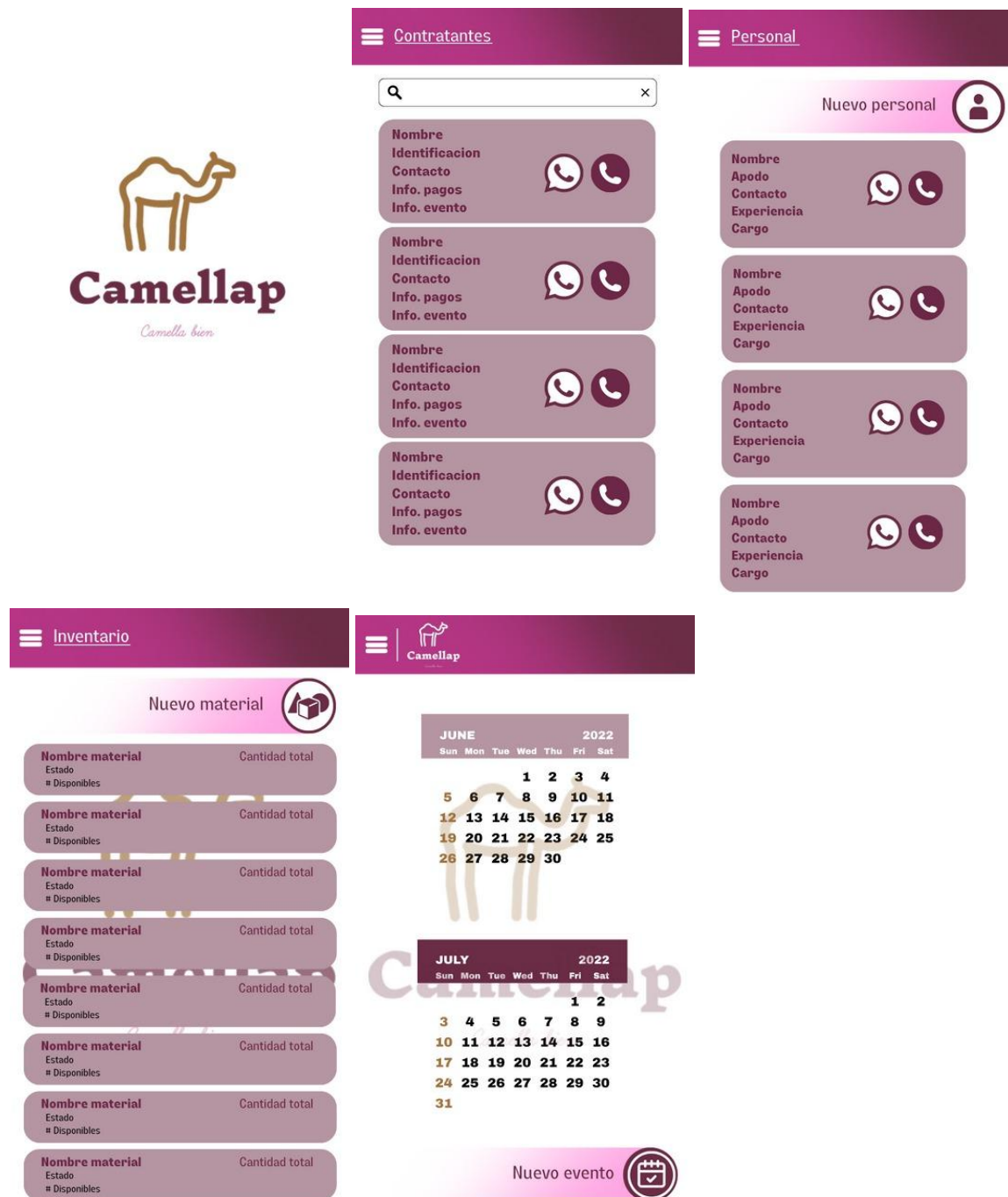
Logos

Respetando el nombre que se le ha dado a la aplicación y los colores anteriormente mostrados. Además el logo se creó a partir del nombre.



Mockups

Posteriormente se desarrollaron los mockups para tener una idea inicial de cómo debería quedar la interfaz de usuario, y las diferentes pantallas a visualizar. Por lo tanto se crearon pantallas de inicio de la aplicación, de visualización de contratantes, de personal, de eventos y de finanzas.



Arquitectura

Para el desarrollo del programa se empleó la arquitectura de datos Model-View-Model-View. Esto se observa con la imagen:

- ▼ com.example.camellap
 - > Model
 - > ui
 - > ViewModel

Más adelante en el documento se explica más detalladamente cada uno de los componentes.

Diagramas

Diagrama de clases

Para la lógica del programa se plantean 6 clases, dos de las cuales heredan de una clase padre. A continuación se describirán los atributos y métodos de cada una de ellas.

La primera clase tiene como nombre Gerente, y será tomada como la clase principal, en ella se tiene como atributo el nombre del gerente. Esta clase también posee métodos, que en total suman 8, en los que se encuentran los métodos para la creación de los demás objetos, la modificación de los mismos, y la eliminación de un evento.

Por otro lado, la clase Persona tiene dos atributos que son nombre, identificación y contacto, además de poseer el método entregar Info, el cual tiene el fin de mandar la información de estos atributos. De esta clase heredan Contratante y Personal. Contratante tiene adicionado el atributo estado Pago, es importante aclarar que estado pago nos permite saber si el contratante debe dinero o no, además tenemos 4 métodos que nos permiten modificar la información del contratante y mostrarla. Con respecto a Personal esta clase añade los atributos apodo, experiencia y cargo, además de tener métodos para modificar y mostrar la información.

La clase Evento conserva toda la información relacionada a un evento, por lo tanto tiene los atributos inventario, personal, contratante, costó estado Pago, el cual nos informa si el evento ya ha sido pagado o aun no, lugar, temática y fecha. Como métodos posee los necesarios para modificar y mostrar la información, de la misma forma que las demás clases.

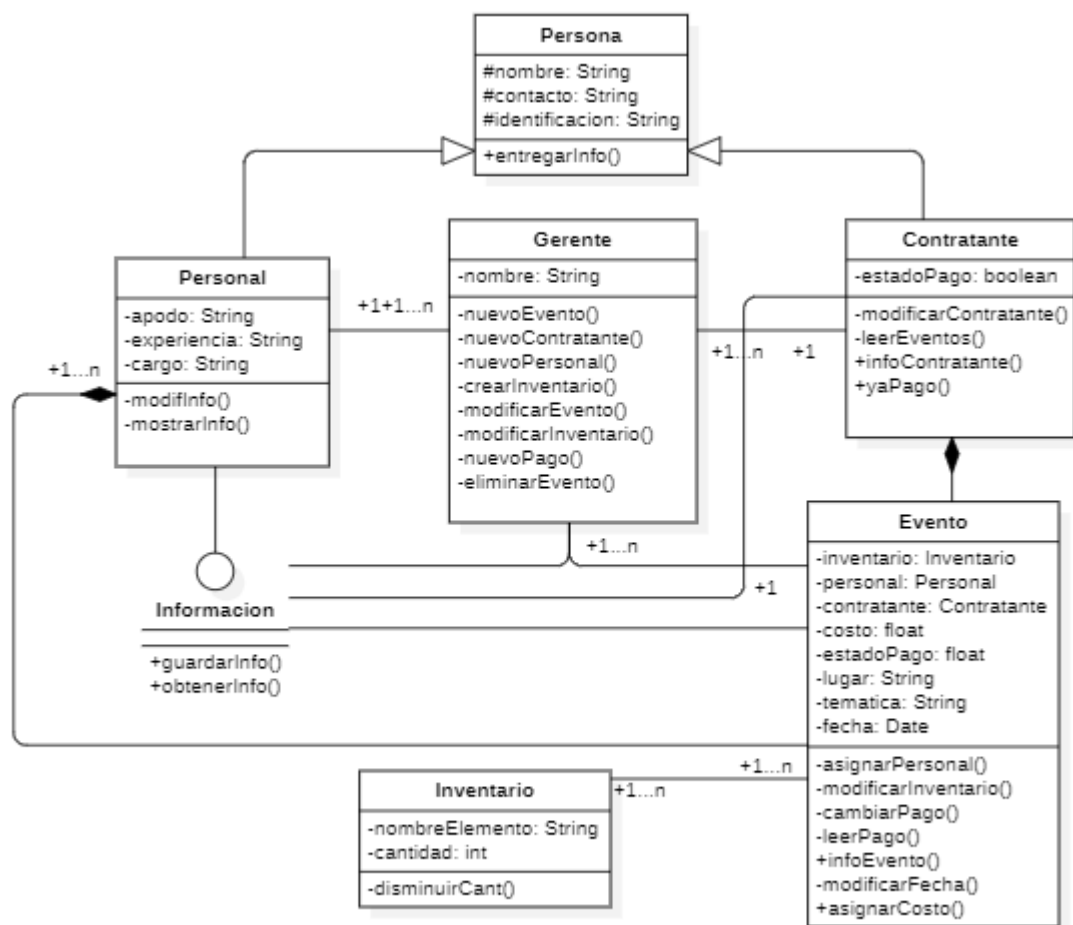
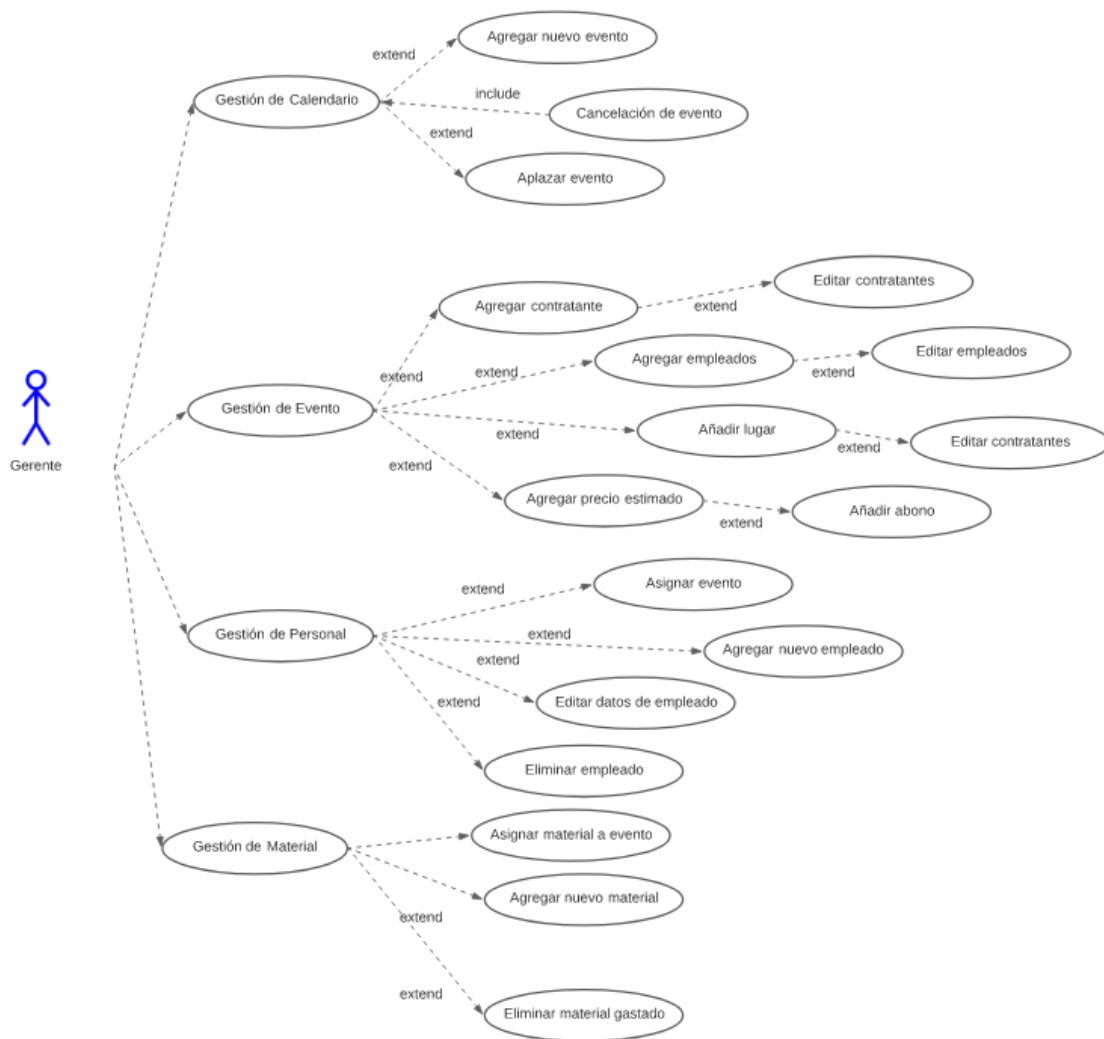


Diagrama de casos de uso

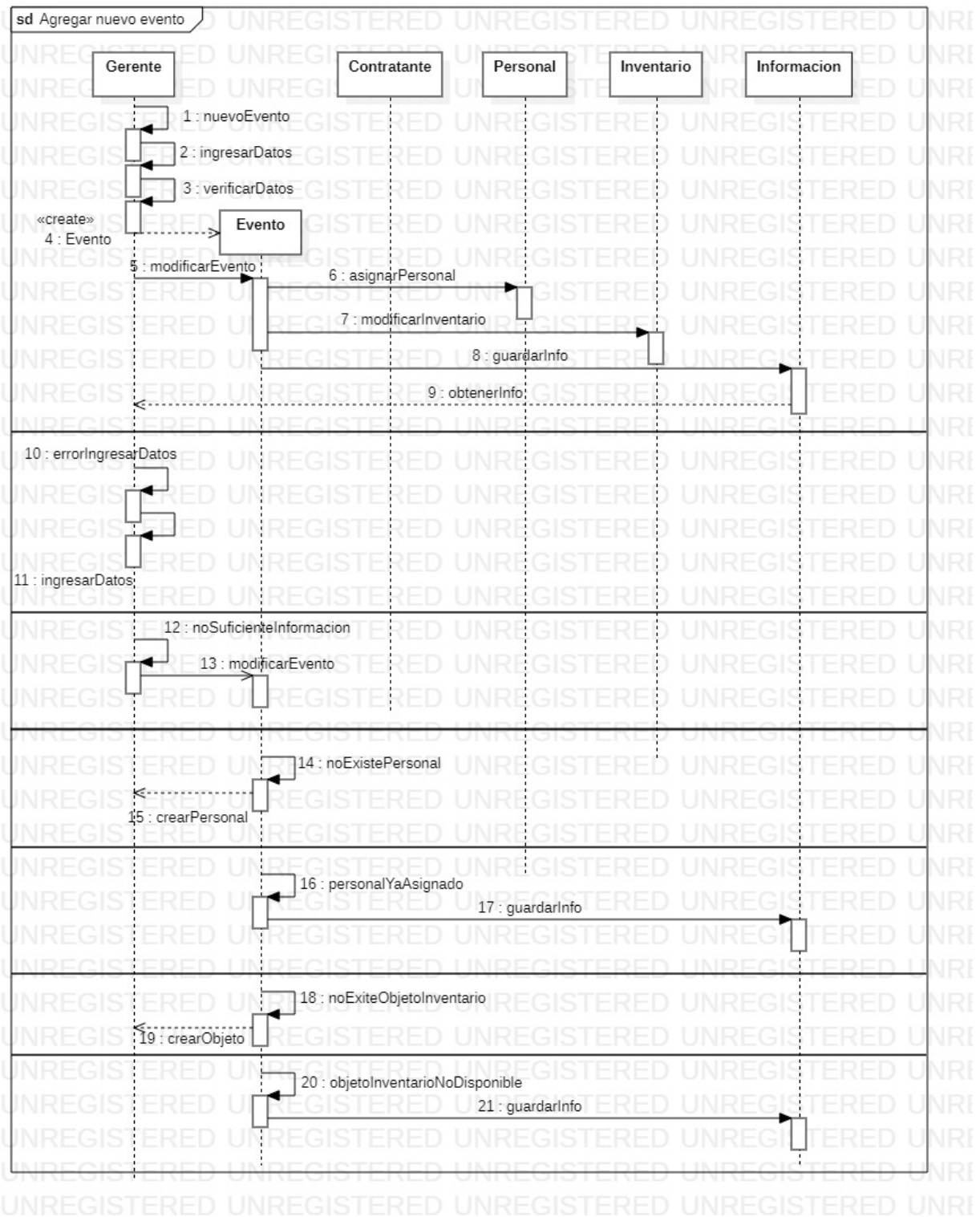
El diagrama mostrado a continuación representa los casos de uso de la aplicación, tanto generales como específicos, los cuales se describen en un documento de excel, al cual se le ha hecho un hipervínculo.

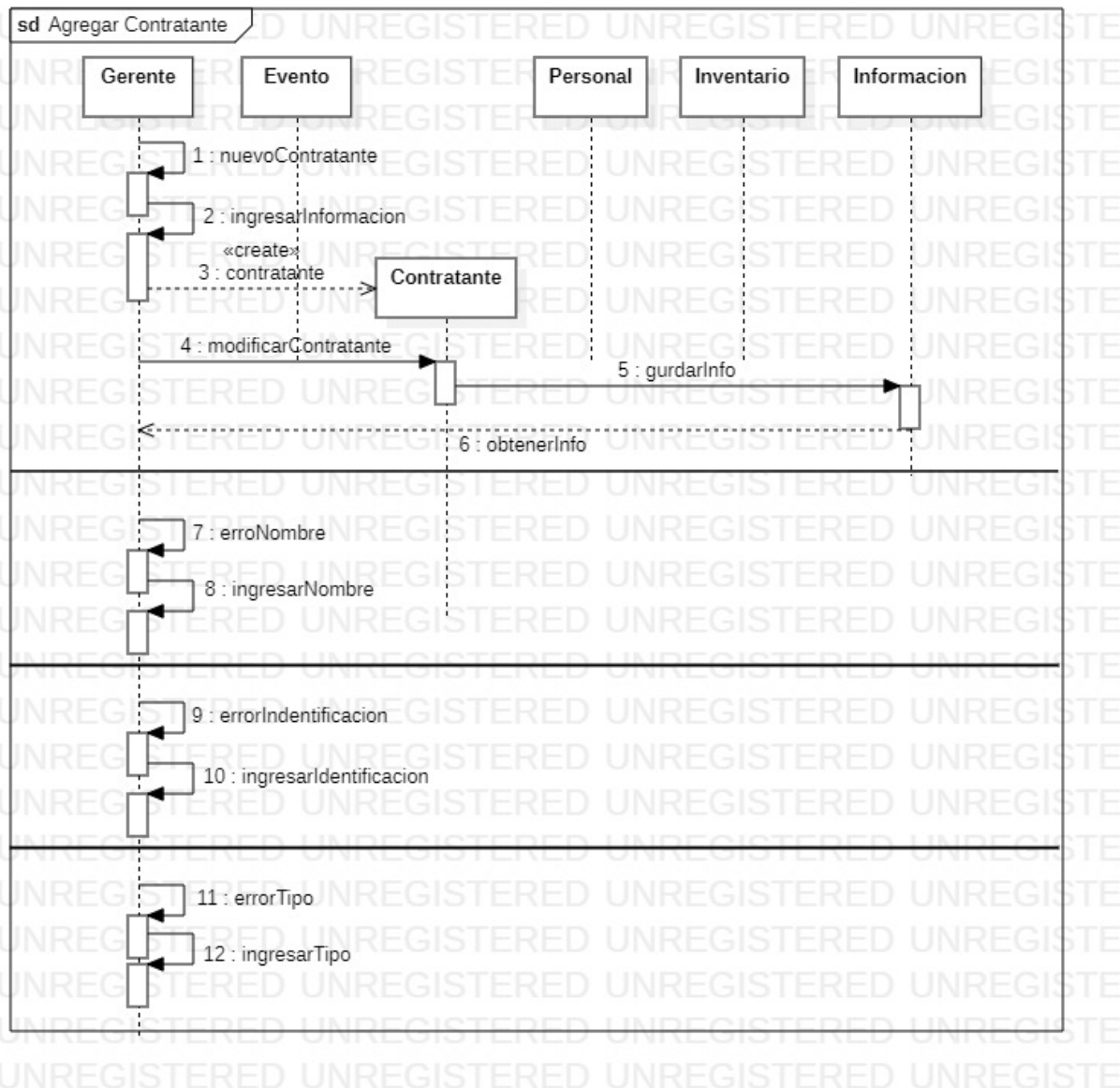


+ Casos de uso detallados Camellap .

Diagramas de secuencia

A continuación se muestran unos diagramas de secuencia de creación de un nuevo elemento. Como se evidencia se crearon diagramas de secuencia de cada caso de uso específico, relacionando las clases de la parte lógica que intervienen en el proceso, teniendo en cuenta los métodos que se emplean. Si desea consultar la totalidad de los diagramas de secuencia presione aquí [☰ Diagramas de Secuencia Camellap](#) .





Descripción del código

Lógica del programa

Clases

A continuación se llevará a cabo una descripción de las clases que se desarrollaron y el criterio de selección para llevarlas a cabo, mediante el análisis de los requerimientos del usuario y las historias de usuarios se decidió que las clases necesarias para llevar a cabo el desarrollo de la app y cumplir con los requerimientos del usuario son las siguientes:

- Persona
- Personal
- Evento
- Contratante

- Inventario
- Gerente

Estas clases se definieron teniendo en cuenta los siguientes requisitos

1. La necesidad del cliente, descartamos con este requisito cualquiera que no fuera de primera necesidad para el cliente.
2. La facilidad de desarrollo, descartamos las opciones de clases y métodos que pudieran complicar el desarrollo de la app..
3. Optimización y minimización de líneas de código.

A continuación se llevará a cabo la descripción de cada clase de forma individual

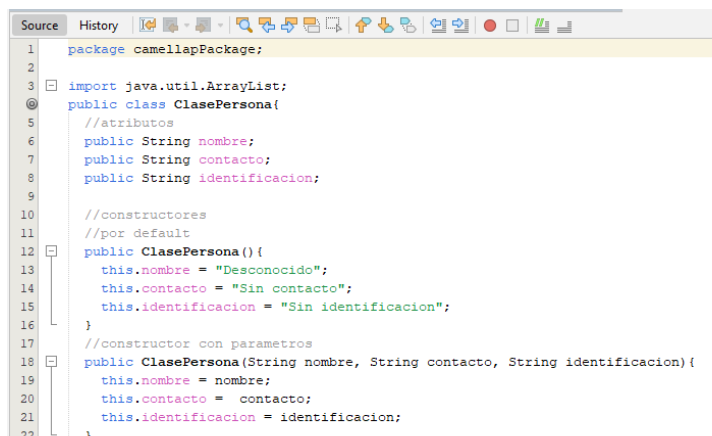
Persona.

La clase persona es una clase madre que tiene como función reunir los atributos y métodos comunes que tendrían las clases ClaseContratante, ClasePersonal y ClaseGerente, para reducir las líneas de código al final del desarrollo, los atributos de la clase son los siguientes, **nombre**, de tipo **String**, que cumple la función de captar el nombre del contacto a crear sin importar el rol que vaya a fungir durante el programa, **contacto**, de tipo **String**, este cumple la función de almacenar el contacto de la persona en cuestión, es de tipo string debido a que no se operará de forma aritmética con él, y, al ser un número compuesto de 10 cifras resulta más útil manejarlas como una cadena de caracteres, finalmente **identificacion**, tipo **String**, se encarga de almacenar la información de la cédula de identificación del contacto a crear. A continuación una lista de los métodos de la clase y su función.

Métodos

Esta clase únicamente cuenta con un método que es el de entregar información.

- **entregar Info()**, es un método público que retorna un ArrayList con toda la información que se encuentra dentro de la clase al momento de crear el objeto, lo que significa que si se crea un objeto de tipo persona sin atributos, retorna la información que se encuentra en el constructor sin parámetros con valores por defecto.
- **modificar Info()**, es un método público que no retorna valor y únicamente cambia la información asociada a un evento.



```

1 package camellapPackage;
2
3 import java.util.ArrayList;
4
5 public class ClasePersona{
6     //atributos
7     public String nombre;
8     public String contacto;
9     public String identificacion;
10
11     //constructores
12     //por default
13     public ClasePersona() {
14         this.nombre = "Desconocido";
15         this.contacto = "Sin contacto";
16         this.identificacion = "Sin identificacion";
17     }
18     //constructor con parametros
19     public ClasePersona(String nombre, String contacto, String identificacion){
20         this.nombre = nombre;
21         this.contacto = contacto;
22         this.identificacion = identificacion;
23     }
24 }

```

```

Source History
25 //getters y setters
26 public String getNombre(){
27     return nombre;
28 }
29 public String getContacto(){
30     return contacto;
31 }
32 public String getIdentificacion(){
33     return identificacion;
34 }
35 public void setNombre( String nombre){
36     this.nombre = nombre;
37 }
38 public void setContacto( String contacto){
39     this.contacto = contacto;
40 }
41 public void setIdentificacion(String identificacion){
42     this.identificacion = identificacion;
43 }
44 // metodos
45 public ArrayList entregarInfo(){
46     String nombre = this.nombre;
47     String contacto = this.contacto;
48     String identificacion = this.identificacion;
49
50     ArrayList<String> info = new ArrayList<String>();
51     info.add(nombre);
52     info.add(contacto);
53     info.add(identificacion);
54     return info;
55 }
56
57
58 public void modificarInfo(String nombre,String contacto, String identificacion){
59     this.nombre = nombre;
60     this.contacto = contacto;
61     this.identificacion = identificacion;
62 }

```

Gerente

La clase gerente es una clase sencilla en cuanto atributos, pero más compleja en cuanto a métodos ya que esta clase funge como un administrador y modo de control de funciones de las demás clases de modo que es capaz de gestionar muchas actividades y llevar a cabo otras, la clase gerente solo consta del atributo **nombre** esto con el objetivo de que el cliente se sienta más cercano a la aplicación y poder personalizar más la experiencia de la app. A continuación una lista detallada de los métodos y la función que cumplen dentro del programa

Métodos

Contratante.

La clase contratante es una clase que es subordinada a la clase persona, es una clase hija y por lo tanto toma los atributos y algunos de sus métodos, la clase contratante es necesaria para almacenar y crear información correspondiente a los contactos de los contratantes y permitiéndole al usuario asociar este contacto a un evento creado, facilitando para el usuario la tarea de asignar y llevar gestión de las personas con las que tiene contratos para eventos que se realizará, La clase contratante (ClaseContratante) consta de los siguientes atributos propios, estadoPago que es de tipo booleano, esta se encarga de almacenar la información relacionada al estado del pago como un true o un false dependiendo de si este se encuentra pagado o no, e infoEvento que es de tipo string, esta variable reúne toda la información adyacente relacionada con el evento pero que no es de primera necesidad como texto de forma que la reúne toda en un solo espacio, además de eso hereda todos los atributos de la clase madre o supraordinada ClasePersona. Por otro lado los métodos y su función se encuentran listados y descritos a continuación.

Métodos

Personal.

La clase personal es una clase encargada de la creación de personal para su posterior asignación a un evento como trabajador, esta es una clase subordinada o hija de la clase ClasePersona, sus atributos exclusivos son los siguientes, **apodo**, tipo **String**, este atributo se encarga de almacenar el apodo con el que se conoce al personal, es un detalle de personalización del cliente más que un detalle de utilidad práctica, **experiencia**, tipo **String**, este atributo almacena la experiencia con la que cuenta el trabajador y se almacena como texto de modo que es más fácil acceder a ella y entenderla, **cargo**, tipo **String**, se encarga de almacenar el rol que llevará a cabo el trabajador durante el evento, es de tipo texto porque puede variar entre varias opciones para las cuales el cliente le gustaría tener total libertad de alterar. A continuación se encuentran enlistados los métodos y las funciones que llevan a cabo cada uno.

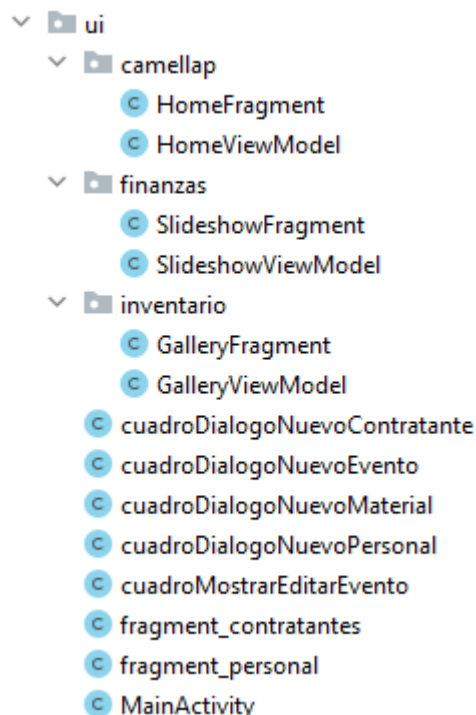
Métodos

Evento.

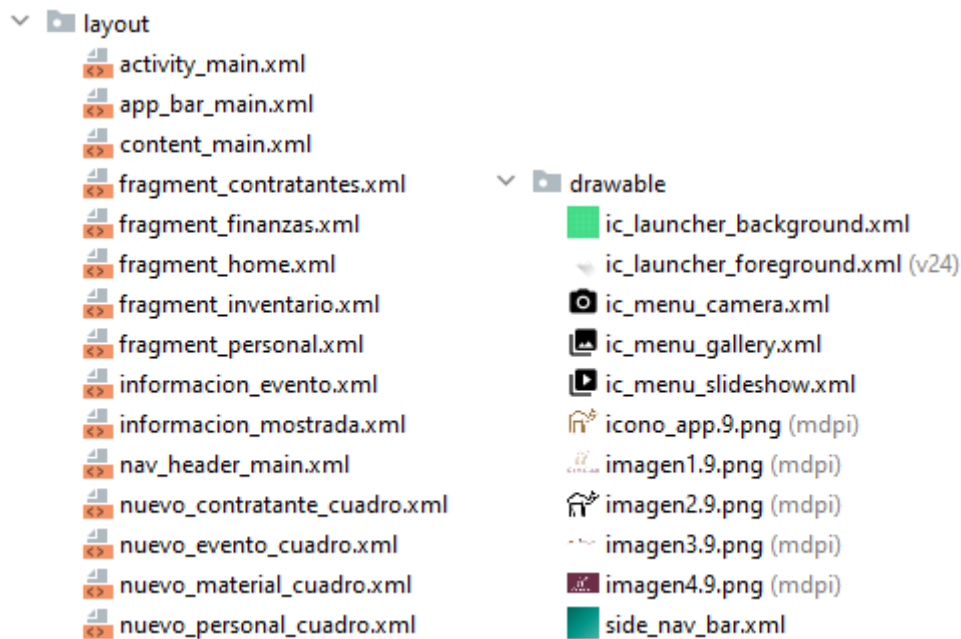
La clase evento es una clase encargada de reunir la información de todas las demás clases para compilar en un solo lugar toda la información que es útil para el cliente

Interfaz gráfica

Para la interfaz gráfica se tuvieron que crear clases que se encuentran distribuidas en el proyecto de la siguiente manera:

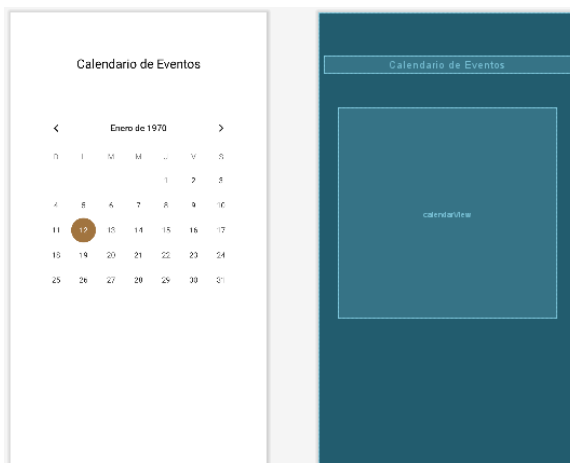


Además se tuvieron que hacer diferentes archivos xml y añadir algunos recursos para la parte visual necesaria:

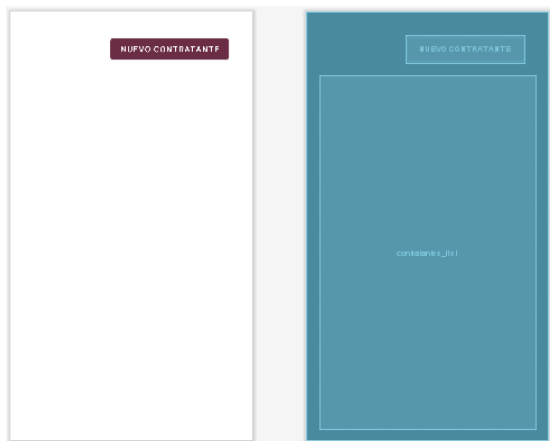


También se tuvo que añadir un estilo para definir el esquema de colores.

Ahora bien, para entender mejor el comportamiento de la parte gráfica, describiremos a continuación las clases más importantes involucradas. En primer lugar partimos de la clase HomeFragment, la cual utiliza archivo xml fragment home. Lo mismo sucede con las clases Slideshow Fragment, GalleryFragment, fragment contratantes y fragment personal, los cuales llaman a los archivos xml fragment finanzas, fragment inventario, fragment contratantes y fragment personal, respectivamente. Un ejemplo de un archivo xml visualizado que se creó se muestra en la imagen:

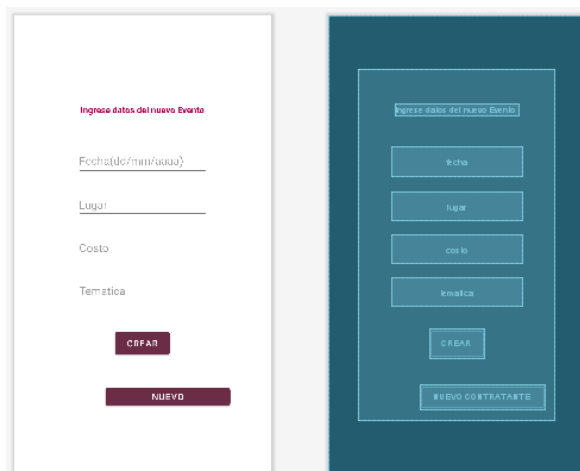


Es importante considerar que en cada uno de los fragments, exceptuando Slideshow Fragment, del proyecto se ha creado un listView en el que se muestra la lista de los objetos creados según corresponda a cada fragmento, además de un botón que permite llamar a cada cuadro de diálogo necesario para crear el objeto al que se haga mención en cada fragment. Esto se puede evidenciar con la siguiente imagen:



En el caso de Slideshow Fragment se busca mostrar los eventos pendientes de pago, y el botón es para marcar algún evento como pagado.

Por otro lado, se tuvieron que utilizar varios cuadros de diálogo para poder capturar información necesaria para la creación de los diferentes objetos. Estos cuadros de diálogo se crean a partir de las clases `cuadroDialogoNuevoContratante`, `cuadroDialogoNuevoEvento`, `cuadroDialogoNuevoMaterial`, `cuadroDialogoNuevoPersonal`, los cuales utilizan los archivos xml `nuevo_contratante_cuadro`, `nuevo_evento_cuadro`, `nuevo_material_cuadro`, `nuevo_personal_cuadro`. Como ejemplo de un archivo xml utilizado para los cuadros se tiene la siguiente imagen:

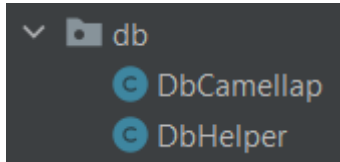


También se creó un cuadro de diálogo, para mostrar la información de los eventos ya creados. Este cuadro de diálogo se muestra una vez que en el calendario colocado en HomeFragment se seleccione la fecha que coincida con el evento creado.

Por último es importante mencionar la participación de la clase `MainActivity` la cual permite la navegación entre los diferentes fragments, además de que aquí se emplean tanto como la barra de tareas como el botón flotante, el cual permite la creación de un nuevo evento.

Persistencia de datos

Para este proyecto se decide usar una persistencia de datos local, en el mismo dispositivo móvil, para este fin se decidió utilizar SQLite, para esto se crean dos clases.



En estas clases tenemos DbHelper en la cuál se inicializa creando una base de datos con las diferentes tablas necesarias para el guardado de cada uno de los datos, luego de que se haya creado la base de datos esta se quedará en el almacenamiento de nuestro dispositivo sin que tengamos que inicializar alguna nuevamente o tener un método de autenticación para poder usar la base de datos.

```
public class DbHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DATABASE_NOMBRE = "Camellap.db";
    public static final String TABLE_INVENTARIO = "t_inventario";
    public static final String TABLE_CONTRATANTE = "t_contratante";
    public static final String TABLE_FINANZAS = "t_finanzas";
    public static final String TABLE_PERSONAL = "t_personal";
    public static final String TABLE_EVENTO = "t_evento";

    public DbHelper(@Nullable Context context) {
        super(context, DATABASE_NOMBRE, factory: null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {

        sqLiteDatabase.execSQL("CREATE TABLE " + TABLE_INVENTARIO + "(" +
            "id INTEGER PRIMARY KEY AUTOINCREMENT," +
            "nombre_material TEXT NOT NULL," +
            "cantidad_material INTEGER NOT NULL)");
        sqLiteDatabase.execSQL("CREATE TABLE " + TABLE_CONTRATANTE + "(" +
            "id INTEGER PRIMARY KEY AUTOINCREMENT," +
            "nombrecontratante TEXT NOT NULL," +
            "contatocontratante TEXT NOT NULL," +
            "identificacioncontratante INTEGER NOT NULL," +
            "estadopagocontratante TEXT NOT NULL)");
    }
}
```

En esta captura se puede observar que en principio se inicializan; la versión que se usará de la base de datos, la base de datos con su respectivo nombre y las tablas que se le agregan a esta base de datos.

Luego tenemos un onCreate en el cual vamos a decir las diferentes variables que se van a insertar en las diferentes tablas de datos.

Ahora respecto a la clase DbCamellap, se tienen un par de métodos, que son: métodos para insertar datos en las tablas de la base de datos, para el cual se hace un extends de nuestra clase DbHeler y así poder modificar, insertar y leer nuestra base de datos de una forma más eficaz. En nuestro método de insertar podemos observar que tenemos un try catch al rededor, el cual nos ayudará en caso de que al insertar nuestros datos en la tabla se produzca un error

nuestra aplicación no se cierre sino que nos muestre un aviso con el cuál nos indique el error provocado.

```
public class DbCamellap extends DbHelper {

    Context context;

    public DbCamellap(@Nullable Context context) {
        super(context);
        this.context = context;
    }

    public long insertarMaterial(String nombre_material, Integer cantidad_material) {

        long id = 0;

        try {
            DbHelper dbHelper = new DbHelper(context);
            SQLiteDatabase db = dbHelper.getWritableDatabase();

            ContentValues values = new ContentValues();
            values.put("nombre_material", nombre_material);
            values.put("cantidad_material", cantidad_material);

            id = db.insert(TABLE_INVENTARIO, nullColumnHack: null, values);
        } catch (Exception ex) {
            ex.toString();
        }
    }
}
```

y un método el cual nos sirve para leer y obtener la información introducida en las tablas para luego poder mostrarlas en una lista para que el cliente pueda tener toda la información a la mano. Llamamos las clases que contienen los getter y setter de cada dato, definimos los cursores los cuales nos ayudarán a leer la información de la tabla en la base de datos y luego poder pasarla a un arraylist en la cuál se facilitará el poder mostrarlas en la lista de la interfaz grafica.

```
public void obtenerInfo() {

    DbHelper dbHelper = new DbHelper(context);
    SQLiteDatabase db = dbHelper.getWritableDatabase();

    ClaseInventario inventarios;
    ClasePersonal personal;
    ClaseContratante contratantes;
    ClaseEvento eventos;
    Cursor cursorInventario;
    Cursor cursorContratante;
    Cursor cursorEvento;
    Cursor cursorPersonal;

    cursorInventario = db.rawQuery( sql: "SELECT * FROM " + TABLE_INVENTARIO, selectionArgs: null);

    if (cursorInventario.moveToFirst()) {
        do {
            inventarios = new ClaseInventario((cursorInventario.getString( columnIndex: 1)),
                cursorInventario.getInt( columnIndex: 2));

            gerente.inventario.add(inventarios);
        } while (cursorInventario.moveToNext());
    }
}
```