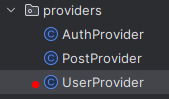
**ACTIVIDAD N° 7:**

**Clase UserProvider**



La clase UserProvider interactúa con una base de datos basada en Parse Server para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) relacionadas con un objeto de la clase User.

También podemos decir que la clase UserProvider actúa como un proveedor de datos que encapsula las operaciones necesarias para manejar usuarios en una base de datos Parse. Permite:

* Crear usuarios.
* Recuperar usuarios por correo electrónico.
* Actualizar información de usuarios.
* Eliminar usuarios por su ID.

Esto facilita la integración del backend con la lógica de negocio y la interfaz de usuario en una aplicación Android.

**MÉTODOS**

1. **createUser(User user)**

public LiveData<String> createUser(User user) {  
  
 MutableLiveData<String> result = new MutableLiveData<>();  
 ParseObject userObject = new ParseObject("User");  
 userObject.put("user\_id", user.getId());  
 userObject.put("email", user.getEmail());  
 userObject.put("password", user.getPassword());  
  
 userObject.saveInBackground(new SaveCallback() {  
 @Override  
 public void done(ParseException e) {  
 if (e == null) {  
 result.setValue("Usuario creado correctamente");  
 } else {  
 result.setValue("Error al crear usuario");  
 }  
 }  
 });  
  
 return result;  
}

1. Este método recibe como parámetro un objeto de la clase User, con este objeto crea un nuevo usuario en la base de datos de Parse.
2. La clase ParseObject es una clase genérica en la biblioteca de Parse que representa una fila en una tabla de la base de datos en nuestro caso permite manejar datos sin necesidad de definir previamente una estructura fija.
3. El MutableLiveData<String> result  almacena mensajes de éxito o error y notifica automáticamente a la interfaz o al resto de la aplicación cuando esos mensajes cambian.
4. ParseObject userObject = new ParseObject("User");

Crea un objeto userObject, indica que el objeto pertenece a la tabla o clase "User" en Parse. Si la clase "User" no existe en la base de datos, Parse la creará automáticamente al guardar este objeto.

1. userObject.put("user\_id", user.getId());  
   userObject.put("email", user.getEmail());  
   userObject.put("password", user.getPassword());

Este código agrega (también se utiliza para actualizar) datos en el objeto userObject para los campos user\_id, email y password con valores obtenidos del objeto user. Cada campo se almacena como un par clave-valor en el objeto Parse (userObject).

En resumen:

* El método put() es la forma de agregar o modificar datos en un objeto Parse antes de guardarlos en la base de datos.
* Los campos ("user\_id", "email", "password") representan las claves en la estructura de la tabla, y los valores asignados son los datos asociados.

1. userObject.saveInBackground userObject.saveInBackground():

Guarda el objeto userObject (que representa un registro de la clase/tabla User) en la base de datos de Parse de forma asíncrona.

El método saveInBackground() evita bloquear el hilo principal, permitiendo que la operación de guardar se ejecute en segundo plano.

1. new SaveCallback():

Define una callback (función de devolución de llamada) que se ejecuta cuando la operación de guardado termina, ya sea con éxito o con error.

1. done(ParseException e):

Este método se llama cuando finaliza el guardado. Recibe un objeto ParseException llamado “e”:

* + - Si e == null: La operación fue exitosa, le asigna string "Usuario creado correctamente" a la variable “result”.
    - Si e != null: Ocurrió un error durante el guardado, le asigna string " Error al crear usuario " a la variable “result”.

**2. getUser(String email)**

Este método recibe el parámetro “email” de tipo String, para obtener un usuario por su correo electrónico en Parse.

// Método para obtener un usuario por su correo electrónico en Parse  
public LiveData<User> getUser(String email) {  
 MutableLiveData<User> userData = new MutableLiveData<>();  
  
 // Crear una consulta para obtener el usuario por su correo electrónico  
 ParseQuery<ParseObject> query = ParseQuery.getQuery("User");  
 query.whereEqualTo("email", email);  
  
 query.findInBackground(new FindCallback<ParseObject>() {  
 @Override  
 public void done(List<ParseObject> users, ParseException e) {  
 if (e == null && users.size() > 0) {  
 ParseObject userObject = users.get(0);  
 User user = new User();  
 user.setId(userObject.getString("user\_id"));  
 user.setEmail(userObject.getString("email"));  
 user.setPassword(userObject.getString("password"));  
  
 userData.setValue(user);  
 } else {  
 userData.setValue(null);  
 }  
 }  
 });  
  
 return userData;  
}

1. MutableLiveData<User> userData = new MutableLiveData<>();

* Crea una instancia de MutableLiveData, el genérico <User> indica que esta instancia almacenará datos del tipo User.
* userData almacenará objetos de tipo User y notificar automáticamente a cualquier observador cuando el valor cambie.

1. ParseQuery<ParseObject> query = ParseQuery.getQuery("User");

* ParseQuery es una clase que se utiliza para construir y ejecutar consultas (queries) sobre los datos almacenados en el backend de Parse. En este caso, se está creando una consulta para obtener objetos de tipo ParseObject.
* El tipo genérico <ParseObject> indica que la consulta devolverá instancias de ParseObject, y los asignará a la variable “query”.
* ParseObject es la clase base para representar cualquier tipo de objeto almacenado en Parse, por lo que se pueden almacenar datos de cualquier tipo con esta clase.
* ParseQuery.getQuery("User") es un método que crea una nueva consulta para buscar objetos de la clase User en el backend de Parse.
* "User" es el nombre de la clase en Parse (tabla) sobre la que realizaremos la consulta. En este caso, está buscando datos en la clase User que contiene la información de los usuarios.

1. query.whereEqualTo("email", email);

* query.whereEqualTo(...):

**whereEqualTo** es un método de la clase ParseQuery que se utiliza para agregar una condición de filtro a la consulta. Este método filtra los resultados de la consulta para que solo se devuelvan los objetos que cumplen con una determinada condición.

* En este caso busca los objetos cuyo campo especificado tenga un valor igual al valor que se le pase como argumento, el email o sea que está filtrando los usuarios por el campo email.
* "email": es el nombre del campo en los objetos de la clase User (en la base de datos de Parse) que contiene la dirección de correo electrónico del usuario.

1. query.findInBackground(new FindCallback<ParseObject>() {...})

* findInBackground ejecuta la consulta en segundo plano y obtiene los resultados de la base de datos sin bloquear el hilo principal. El resultado se maneja a través del FindCallback.

1. public void done(List<ParseObject> users, ParseException e)

* El método done dentro de FindCallback recibe dos parámetros: una lista de objetos ParseObject (users) y un posible error (e).

1. if (e == null && users.size() > 0)

* Se comprueba si e == null (lo que significa que no hubo errores) y si la lista de usuarios (users) tiene elementos (es decir, al menos un usuario fue encontrado).
* De lo contrario (por la rama del else ) le asigna string "Usuario no encontrado" a la variable “result”.

1. ParseObject userObject = users.get(0);

* Si ambas condiciones son verdaderas, el código procederá a manejar los datos del primer usuario encontrado.

1. User user = new User();  
   user.setId(userObject.getString("user\_id"));  
   user.setEmail(userObject.getString("email"));  
   user.setPassword(userObject.getString("password"));

* Estos métodos obtienen los valores de los campos "user\_id", "email", y "password" del objeto ParseObject, que corresponde al usuario en la base de datos de Parse y los asignan al objeto User creado previamente.

1. userData.setValue(user);

* userData.setValue(user): establece el objeto User dentro del LiveData llamado userData, para que cualquier observer de LiveData reciba el nuevo usuario.

1. else {  
    userData.setValue(null);  
   }

* Si no encuentra el usuario o se produce un error, se establece null en el LiveData

**3. updateUser(User user)**

Este método recibe el parámetro un objeto “user” de tipo User, para luego acceder a la base y modificar el registro del usuario.

// Método para actualizar un usuario en Parse  
public LiveData<String> updateUser(User user) {  
 MutableLiveData<String> result = new MutableLiveData<>();  
  
 // Crear una consulta para obtener el usuario por su ID  
 ParseQuery<ParseObject> query = ParseQuery.getQuery("User");  
 query.whereEqualTo("user\_id", user.getId());  
  
 query.findInBackground(new FindCallback<ParseObject>() {  
 @Override  
 public void done(List<ParseObject> users, ParseException e) {  
 if (e == null && users.size() > 0) {  
 ParseObject userObject = users.get(0);  
 userObject.put("email", user.getEmail());  
 userObject.put("password", user.getPassword());  
  
 // Guardar el usuario actualizado en Parse  
 userObject.saveInBackground(new SaveCallback() {  
 @Override  
 public void done(ParseException e) {  
 if (e == null) {  
 result.setValue("Usuario actualizado correctamente");  
 } else {  
 result.setValue("Error al actualizar usuario");  
 }  
 }  
 });  
 } else {  
 result.setValue("Usuario no encontrado");  
 }  
 }  
 });  
  
 return result;  
}

1. MutableLiveData<String> result = new MutableLiveData<>();

* almacena mensajes de éxito o error y notifica automáticamente a la interfaz o al resto de la aplicación cuando esos mensajes cambian.

1. ParseQuery<ParseObject> query = ParseQuery.getQuery("User");

* ParseQuery es una clase que se utiliza para construir y ejecutar consultas (queries) sobre los datos almacenados en el backend de Parse. En este caso, se está creando una consulta para obtener objetos de tipo ParseObject.
* El tipo genérico <ParseObject> indica que la consulta devolverá instancias de ParseObject y los asignará a la variable “query”.
* ParseObject es la clase base para representar cualquier tipo de objeto almacenado en Parse, por lo que se pueden almacenar datos de cualquier tipo con esta clase.
* ParseQuery.getQuery("User") es un método que crea una nueva consulta para buscar objetos de la clase User en el backend de Parse.
* "User" es el nombre de la clase en Parse (tabla) sobre la que realizaremos la consulta. En este caso, está buscando datos en la clase User que contiene la información de los usuarios.

1. query.whereEqualTo ("user\_id", user.getId());

* query.whereEqualTo(...):

**whereEqualTo** es un método de la clase ParseQuery que se utiliza para agregar una condición de filtro a la consulta. Este método filtra los resultados de la consulta para que solo se devuelvan los objetos que cumplen con una determinada condición.

* En este caso busca los objetos cuyo campo especificado tenga un valor igual al valor que se le pase como argumento, el “user\_id” o sea que está filtrando los usuarios por el campo “user\_id”.
* "user.getId()": obtiene el id, del objeto “user” que el método updateUser recibió como pará metro.

1. query.findInBackground(new FindCallback<ParseObject>() {...})

* findInBackground ejecuta la consulta en segundo plano y obtiene los resultados de la base de datos sin bloquear el hilo principal. El resultado se maneja a través del FindCallback.

1. public void done(List<ParseObject> users, ParseException e)

* El método done dentro de FindCallback recibe dos parámetros: una lista de objetos ParseObject (users) y un posible error (e).

1. if (e == null && users.size() > 0)

* Se comprueba si e == null (lo que significa que no hubo errores) y si la lista de usuarios (users) tiene elementos (es decir, al menos un usuario fue encontrado).
* De lo contrario (por la rama del else ) le asigna string "Usuario no encontrado" a la variable “result”.

1. ParseObject userObject = users.get(0);

* Si ambas condiciones son verdaderas, el código procederá a manejar los datos del primer usuario encontrado.

1. userObject.put("email", user.getEmail());  
   userObject.put("password", user.getPassword());

* Estos métodos obtienen los valores de los campos "email", y "password" del objeto ParseObject, que corresponde al usuario en la base de datos de Parse y los asignan al objeto user de la clase User.

1. userObject.saveInBackground(new SaveCallback() {…})

* Guarda el objeto userObject (que representa un registro de la clase/tabla User) en la base de datos de Parse de forma asíncrona.
* El método saveInBackground() evita bloquear el hilo principal, permitiendo que la operación de guardar se ejecute en segundo plano.
* new SaveCallback() define una callback (función de devolución de llamada) que se ejecuta cuando la operación de guardado termina, ya sea con éxito o con error.

1. done(ParseException e):

Este método se llama cuando finaliza el guardado. Recibe un objeto ParseException llamado “e”:

* + - Si e == null: La operación fue exitosa, le asigna string "Usuario actualizado correctamente" a la variable “result”.

Si e != null: Ocurrió un error durante el guardado, le asigna string " Error al actualizar usuario " a la variable “result”.

**4. deleteUser(String userId)**

Este método recibe el parámetro “userId” de tipo String, para obtener un usuario por su “user\_id” en Parse.

// Método para eliminar un usuario en Parse  
public LiveData<String> deleteUser(String userId) {  
 MutableLiveData<String> result = new MutableLiveData<>();  
  
 // Crear una consulta para obtener el usuario por su ID  
 ParseQuery<ParseObject> query = ParseQuery.getQuery("User");  
 query.whereEqualTo("user\_id", userId);  
  
 query.findInBackground(new FindCallback<ParseObject>() {  
 @Override  
 public void done(List<ParseObject> users, ParseException e) {  
 if (e == null && users.size() > 0) {  
 ParseObject userObject = users.get(0);  
  
 // Eliminar el usuario de Parse  
 userObject.deleteInBackground(new DeleteCallback() {  
 @Override  
 public void done(ParseException e) {  
 if (e == null) {  
 result.setValue("Usuario eliminado correctamente");  
 } else {  
 result.setValue("Error al eliminar usuario");  
 }  
 }  
 });  
 } else {  
 result.setValue("Usuario no encontrado");  
 }  
 }  
 });  
  
 return result;  
}

1. MutableLiveData<String> result = new MutableLiveData<>();

* almacena mensajes de éxito o error y notifica automáticamente a la interfaz o al resto de la aplicación cuando esos mensajes cambian.

1. ParseQuery<ParseObject> query = ParseQuery.getQuery("User");

* ParseQuery es una clase que se utiliza para construir y ejecutar consultas (queries) sobre los datos almacenados en el backend de Parse. En este caso, se está creando una consulta para obtener objetos de tipo ParseObject.
* El tipo genérico <ParseObject> indica que la consulta devolverá instancias de ParseObject, y los asignará a la variable “query”.
* ParseObject es la clase base para representar cualquier tipo de objeto almacenado en Parse, por lo que se pueden almacenar datos de cualquier tipo con esta clase.
* ParseQuery.getQuery("User") es un método que crea una nueva consulta para buscar objetos de la clase User en el backend de Parse.
* "User" es el nombre de la clase en Parse (tabla) sobre la que realizaremos la consulta. En este caso, está buscando datos en la clase User que contiene la información de los usuarios.

1. query.whereEqualTo("user\_id", userId);

* query.whereEqualTo(...):

**whereEqualTo** es un método de la clase ParseQuery que se utiliza para agregar una condición de filtro a la consulta. Este método filtra los resultados de la consulta para que solo se devuelvan los objetos que cumplen con una determinada condición.

* En este caso busca los objetos cuyo campo especificado tenga un valor igual al valor que se le pase como argumento, el “user\_id” o sea que está filtrando los usuarios por el campo “user\_id”.
* "user.getId()": obtiene el id, del objeto “user” que el método updateUser recibió como pará metro.

1. query.findInBackground(new FindCallback<ParseObject>() {...})

* findInBackground ejecuta la consulta en segundo plano y obtiene los resultados de la base de datos sin bloquear el hilo principal. El resultado se maneja a través del FindCallback.

1. public void done(List<ParseObject> users, ParseException e)

* El método done dentro de FindCallback recibe dos parámetros: una lista de objetos ParseObject (users) y un posible error (e).

1. if (e == null && users.size() > 0)

* Se comprueba si e == null (lo que significa que no hubo errores) y si la lista de usuarios (users) tiene elementos (es decir, al menos un usuario fue encontrado).
* De lo contrario (por la rama del else ) le asigna string "Usuario no encontrado" a la variable “result”.

1. ParseObject userObject = users.get(0);

* Si ambas condiciones son verdaderas, el código procederá a manejar los datos del primer usuario encontrado.

1. userObject.deleteInBackground(new DeleteCallback()

* El método deleteInBackground (inicia la operación para eliminar el objeto userObject) con un DeleteCallback (este callback permite manejar el resultado de la operación de eliminación) se utiliza para eliminar un objeto de Parse de manera asíncrona.
* Este enfoque asegura que la operación de eliminación no bloquee el hilo principal (UI), permitiendo que la aplicación siga funcionando normalmente mientras se completa la operación.

1. done(ParseException e):

Este método se llama cuando finaliza el guardado. Recibe un objeto ParseException llamado “e”:

* + - Si e == null: La operación fue exitosa, le asigna string "Usuario actualizado correctamente" a la variable “result”.

Si e != null: Ocurrió un error durante el guardado, le asigna string " Error al actualizar usuario " a la variable “result”.

**CÓDIGO**

mport androidx.lifecycle.LiveData;  
import androidx.lifecycle.MutableLiveData;  
import com.example.appchat.model.User;  
  
  
import com.parse.DeleteCallback;  
import com.parse.ParseException;  
import com.parse.ParseObject;  
import com.parse.ParseQuery;  
import com.parse.SaveCallback;  
import com.parse.FindCallback;  
  
import java.util.List;  
  
  
public class UserProvider {

// Método para obtener crear usuario en Parse punto 1)

public LiveData<String> createUser(User user){…}

// Método para obtener un usuario por su correo electrónico punto 2)

public LiveData<User> getUser(String email){…}

// Método para actualizar un usuario en Parse punto 3)

public LiveData<String> updateUser(User user){…}

// Método para eliminar un usuario en Parse punto 4)

public LiveData<String> deleteeUser String email){…}

}