

**MAXIMIZANDO EL POTENCIAL ATLÉTICO: CIENCIA DE DATOS Y
LACTATO EN EL DEPORTE DE ALTO RENDIMIENTO**

Manual Técnico

AUTORES

**WILLIAM ENRIQUE MARTINEZ
CRISTIAN STIVEN GUERRERO**

Director: ing. Edgar Arturo Bustos Caldas

**TRABAJO PARA OBTENER TITULO (TECNOLOGO EN DESARROLLO DE
SOFTWARE)**

**Universidad de Cundinamarca
Facultad de ingeniería
Programa Tecnología en Desarrollo de Software
Soacha (Cundinamarca)
Agosto 2024**

Tabla de contenido

Introducción	6
Tecnologías del desarrollo	7
Android Studio.....	7
Git	10
Java.....	12
Xml.....	12
Gradle	13
SQLite en Android Studio	13
Instalación del software.....	13
1. Clonar el Repositorio desde Git:	14
2. Acceder a la Carpeta del Proyecto por Android Studio:	15
3. Compilar:	16
Estructura Del Proyecto.....	17
Directorio app	18
Subdirectorio manifests.....	18
Subdirectorio java	18
Subdirectorio Res.....	19
Directorio Gradle Scripts.....	19
Subdirectorio layout.....	19
Explicación de Funcionalidad de clases y Actividades	21
DatabaseHelper.java	21
Métodos Sobrescritos DatabaseHelper	21
Métodos Personalizados DatabaseHelper	22
SignupActivity.java.....	23
Métodos Sobrescritos SignupActivity	23
Manejo del Botón de Registro SignupActivity.....	24
Redirección a la Pantalla de Login SignupActivity	25
LoginActivity2.java	25
Métodos Sobrescritos LoginActivity2	26
Manejo del Botón de Inicio de Sesión LoginActivity2.....	27
Redirección a la Pantalla de Registro LoginActivity2.....	28

seccion_iniciada_student_mode__activity.java.....	28
Métodos Sobrescritos seccion_iniciada_student_mode__activity	29
Configuración de Listeners para Botones seccion_iniciada_student_mode__activity	30
definir_datos_deentrada_activity.java	31
Métodos Sobrescritos definir_datos_deentrada_activity	32
Configuración de Listeners para Botones definir_datos_deentrada_activity	32
Métodos Personalizados definir_datos_deentrada_activity	33
Código de pruebas.....	34
Métodos Sobrescritos Código de pruebas	34
Configuración de Listeners para Botones Código de pruebas.....	35
Métodos Personalizados Código de pruebas	36
ScatterPlot.java	37
Métodos Personalizados ScatterPlot	37

Tabla de ilustraciones

Ilustración 1	7
Ilustración 2	8
Ilustración 3	9
Ilustración 4	10
Ilustración 5	10
Ilustración 6	11
Ilustración 7	11
Ilustración 8	14
Ilustración 9	14
Ilustración 10.....	15
Ilustración 11.....	15
Ilustración 12.....	16
Ilustración 13	16
Ilustración 14	17
Ilustración 15	20
Ilustración 16	21
Ilustración 17	21
Ilustración 18	22
Ilustración 19	23
Ilustración 20	23
Ilustración 21	24
Ilustración 22	25
Ilustración 23	25
Ilustración 24	26
Ilustración 25	27
Ilustración 26	28
Ilustración 27	28
Ilustración 28	29
Ilustración 29	30
Ilustración 30	31

Ilustración 31	32
Ilustración 32	32
Ilustración 33	33
Ilustración 34	34
Ilustración 35	35
Ilustración 36	35
Ilustración 37	36
Ilustración 38	37
Ilustración 39	38

Introducción

En este manual técnico se registrará la información para los encargados del mantenimiento del software como para las personas que deseen información sobre la aplicación con el fin del uso correcto del sistema. Se encuentra información tanto sobre el diseño, código del software como una descripción de sus componentes

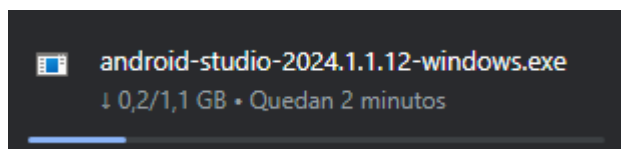
Tecnologías del desarrollo

Durante el desarrollo del proyecto, se emplearán diversas herramientas que contribuirán a la actualización del software. Por ello, es fundamental conocer qué son, cómo funcionan y de qué manera se aplicarán en el proyecto. Estas herramientas juegan un papel crucial en el desarrollo, ya que serán manipuladas, mostradas y utilizadas de manera efectiva.

Android Studio

Se ha decidido utilizar Android Studio como el entorno de desarrollo integrado (IDE) para este proyecto. Esta herramienta ofrece una flexibilidad notable para desarrollar aplicaciones móviles en Android, permitiendo trabajar con una amplia gama de bibliotecas y herramientas de desarrollo. Además, Android Studio facilita una gestión organizada de los directorios y archivos del proyecto, lo cual es crucial para mantener un flujo de trabajo eficiente y ordenado. Su capacidad para integrar diversas funcionalidades y su compatibilidad con múltiples lenguajes de programación hacen de Android Studio una opción ideal para llevar a cabo un desarrollo robusto y eficiente.

Ilustración 1



Fuente: Autoría propia.

En la (ilustración 1), es necesario resaltar que la versión utilizada de Android Studio para el desarrollo de la aplicación, con el fin de evitar errores con Gradle durante su mantenimiento, fue la versión: android-studio-2024.1.1.12-windows.

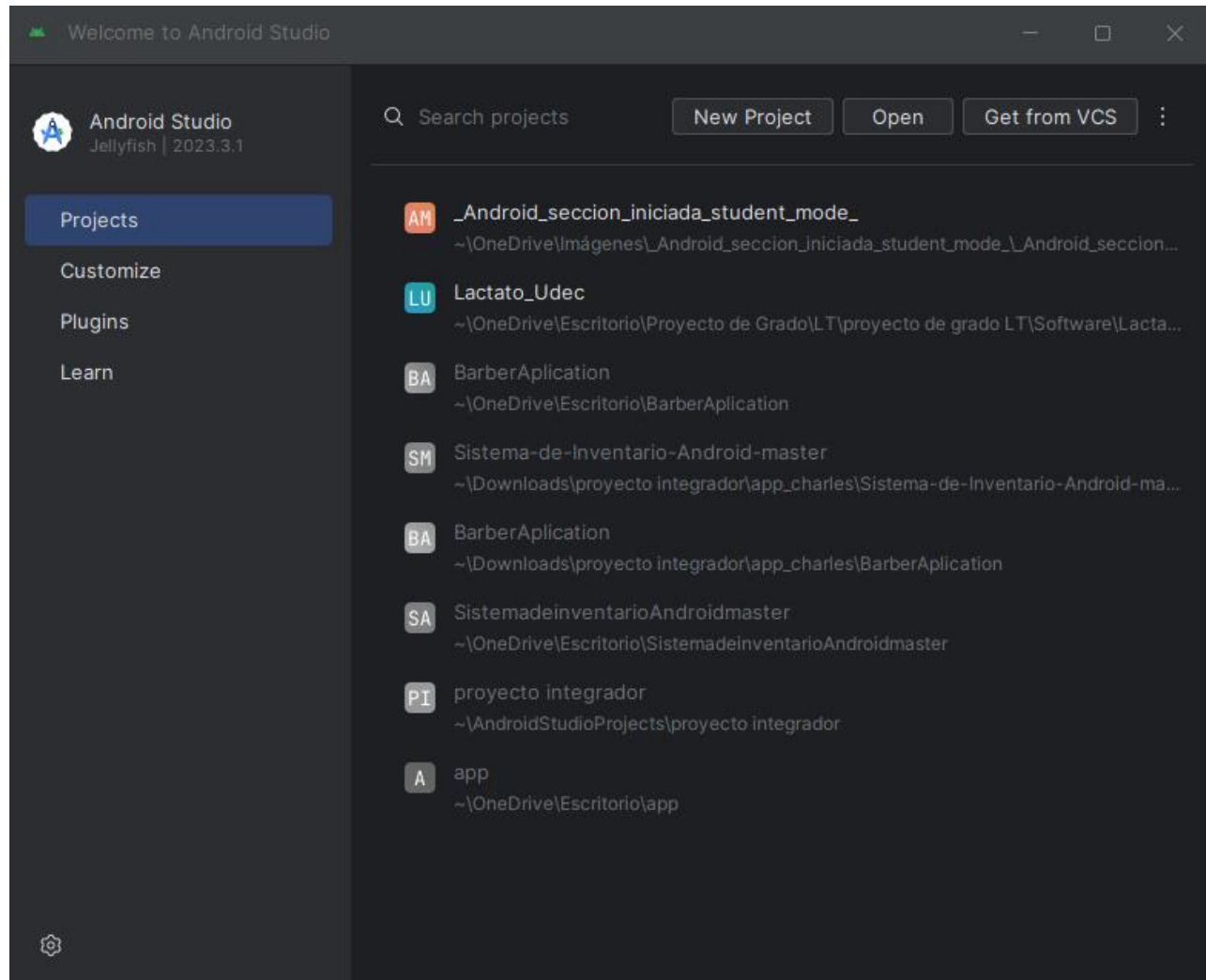
Ilustración 2



Fuente: Autoría propia.

En la (ilustración 2), Se puede ver, una vez instalada la aplicación de Android Studio, que se utilizó la versión Koala // 2024.1.1. Esta versión específica fue elegida para asegurar la compatibilidad y estabilidad del desarrollo de la aplicación. La instalación completa del entorno de desarrollo en la máquina indica que el software está listo para su uso. Con esta versión, los desarrolladores pueden aprovechar las últimas mejoras y características integradas, garantizando así un flujo de trabajo eficiente y evitando posibles errores con Gradle durante el mantenimiento. Android Studio Koala // 2024.1.1 proporciona una plataforma robusta y confiable para trabajar con diversas herramientas y bibliotecas necesarias en el desarrollo de aplicaciones Android.

Ilustración 3



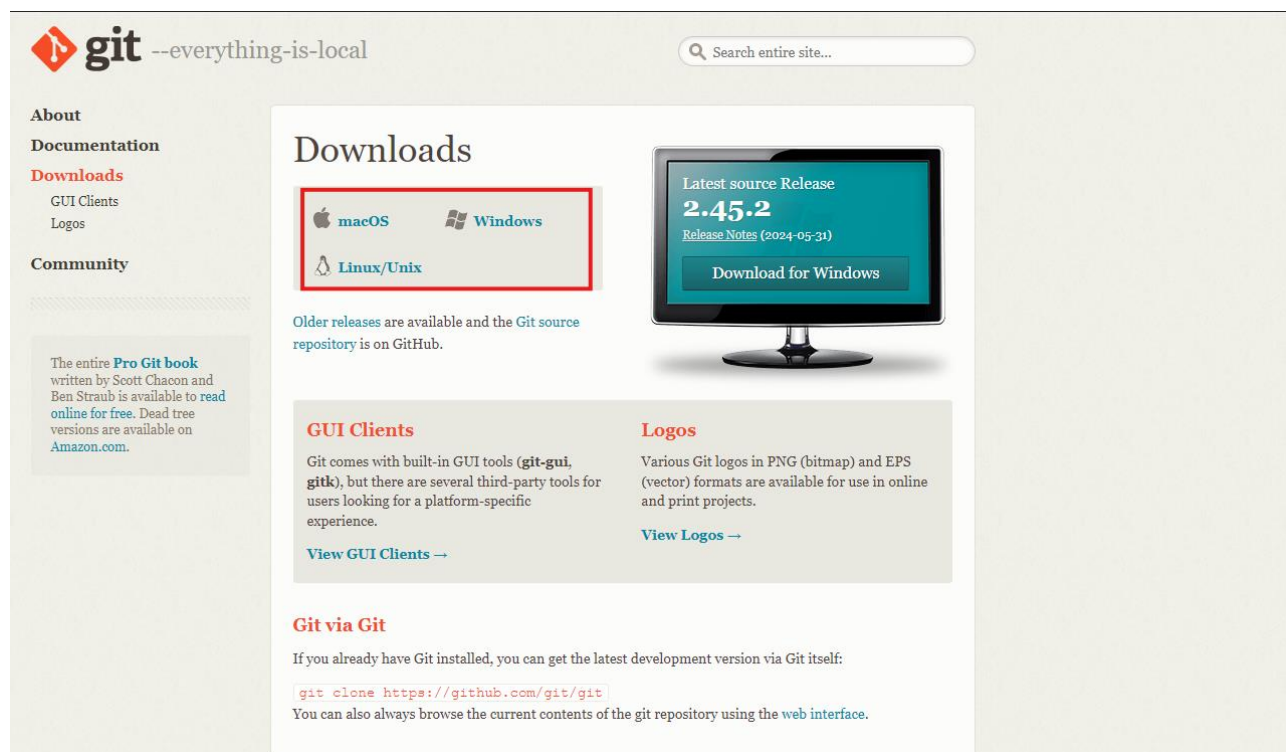
Fuente: Autoría propia.

En la (ilustración 3) se puede ver que ya se instaló perfectamente la aplicación de Android Studio. Antes de proceder a abrir el proyecto, necesitamos tener otra herramienta a la mano para poder descargar el proyecto y seguir sus cambios. Esta herramienta es fundamental para gestionar el código fuente y colaborar eficientemente con el equipo de desarrollo.

Git

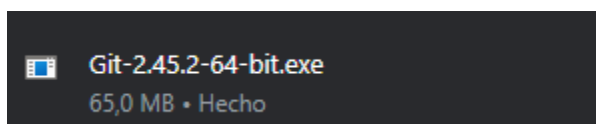
Para este proyecto, hemos optado por utilizar Git como nuestro sistema de control de versiones. Git proporciona una gran flexibilidad para la gestión colaborativa del código, permitiendo a los desarrolladores trabajar con diversos flujos de trabajo y métodos de desarrollo. Además, Git ayuda a mantener un control organizado de las ramas y las versiones del proyecto, lo que es fundamental para asegurar un flujo de trabajo ordenado y eficiente. Sus múltiples funcionalidades integradas y su compatibilidad con varias plataformas convierten a Git en una herramienta indispensable para un desarrollo sólido y efectivo.

Ilustración 4



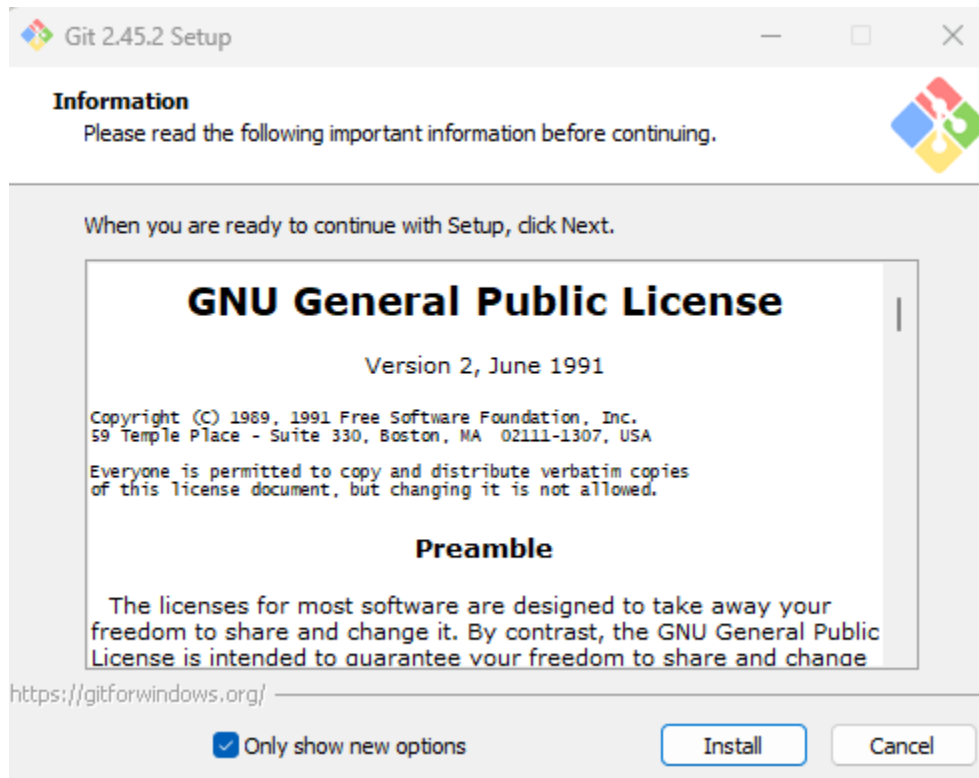
Fuente: Autoría propia.

Ilustración 5



Fuente: Autoría propia.

Ilustración 6



Fuente: Autoría propia.

Ilustración 7



Fuente: Autoría propia.

En la (ilustración 4) se muestra la página de descarga del gestor Git, donde procedemos a hacer clic en "Descargas". En la (ilustración 5) se puede ver la instalación del instalador del gestor y su tamaño de archivo. De la (ilustración 6) a la (ilustración 7) se documenta el proceso de instalación del gestor, destacando cada paso necesario para completar la configuración de Git en el sistema.

Java

Se utilizó Java como lenguaje principal en el proyecto para desarrollar la lógica y funcionalidad de la aplicación. Java es un lenguaje de programación versátil y robusto que se caracteriza por su portabilidad y capacidad para ejecutarse en múltiples plataformas. Este lenguaje se basa en la programación orientada a objetos, permitiendo la creación de clases y objetos que encapsulan datos y comportamientos. Java es ampliamente utilizado en el desarrollo de aplicaciones empresariales, móviles y web debido a su estabilidad, seguridad y amplia comunidad de desarrolladores.

Xml

Utilizamos XML (Extensible Markup Language) para definir la estructura y el contenido visual de la interfaz de usuario (UI). XML es un lenguaje de marcado que nos permite organizar y etiquetar datos de manera jerárquica utilizando etiquetas personalizadas. Con XML, podemos describir elementos visuales como botones, campos de texto, imágenes, listas y otros componentes de la interfaz de usuario de manera clara y estructurada. Esto facilita la separación de la estructura y el contenido, mejorando así la legibilidad del código y la facilidad para mantener y actualizar el proyecto en Android Studio.

Gradle

En el proyecto, se utilizó Gradle como sistema de automatización de compilación y gestión de dependencias. Gradle es una herramienta moderna y eficiente que permite definir de manera declarativa la estructura del proyecto, las tareas de compilación y las dependencias necesarias. Esto facilita la configuración y optimización del proceso de construcción del software de manera reproducible y escalable.

Gradle es ampliamente reconocido por su capacidad para manejar proyectos complejos con múltiples módulos y configuraciones personalizadas. Su integración nativa con Android Studio proporciona una experiencia de desarrollo fluida y eficiente para construir aplicaciones Android robustas y optimizada.

SQLite en Android Studio

En el desarrollo del proyecto, se utilizó SQLite como sistema de gestión de bases de datos integrado en la aplicación Android. SQLite es una biblioteca de software que proporciona una base de datos relacional, compatible con SQL, y que está integrada en el sistema operativo Android. Esta elección permite una gestión eficiente y local de los datos, sin la necesidad de una conexión a internet.

Instalación del software

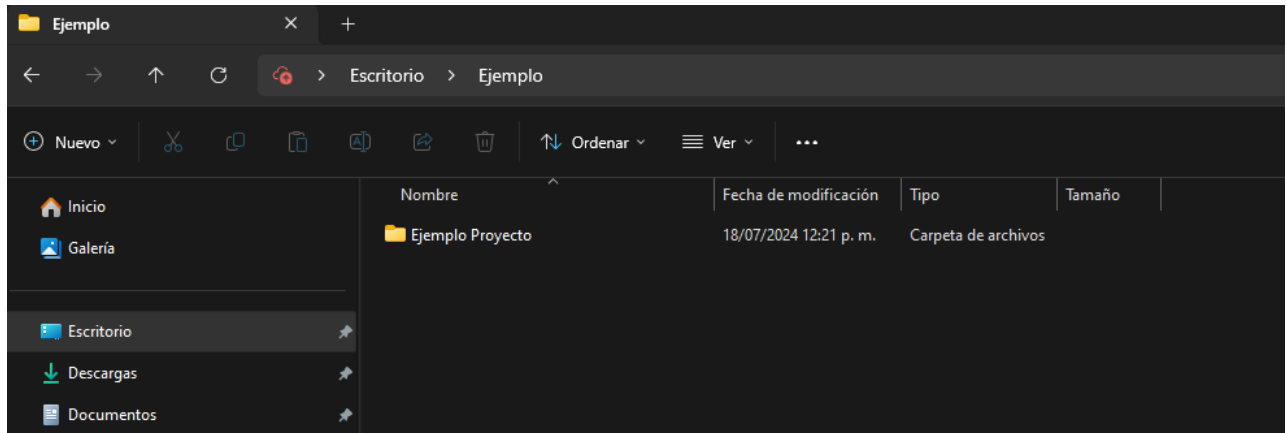
Para llevar a cabo el desarrollo del software, es fundamental seguir un procedimiento específico para traer el proyecto desde Git hacia la máquina local del sistema. Este proceso implica una serie de pasos organizados que garantizan la correcta obtención y configuración inicial del código fuente desde el repositorio remoto de Git al entorno de desarrollo local.

Para iniciar el proceso de traer un proyecto desde Git hacia la máquina local, el primer paso es posicionarse en la carpeta del sistema donde se desea almacenar y trabajar con el código

del proyecto. Este paso es crucial porque determina el directorio principal donde se establecerá la conexión con el repositorio remoto y se descargará el código fuente.

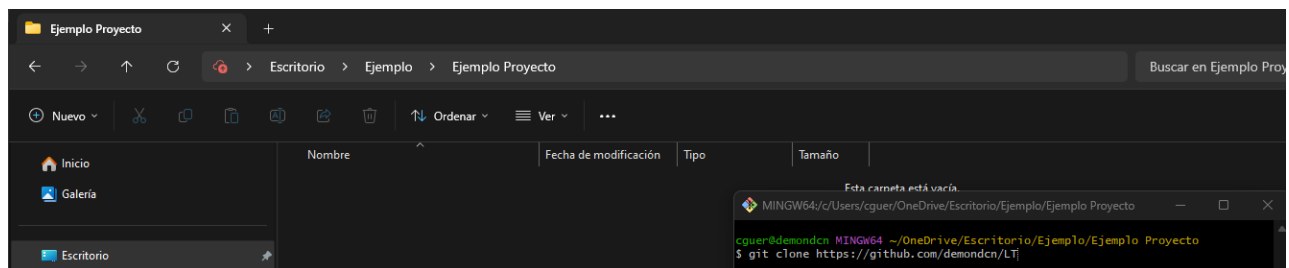
1. Clonar el Repositorio desde Git:

Ilustración 8



Fuente: Autoría propia.

Ilustración 9

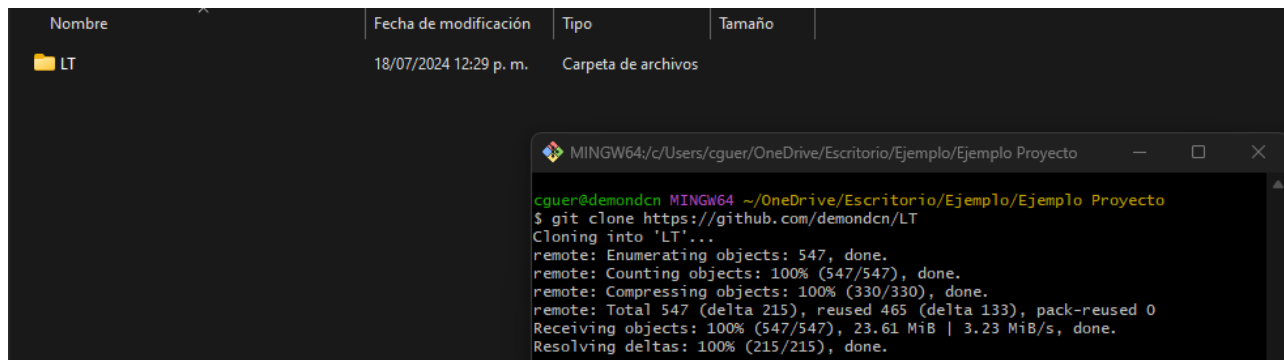


Fuente: Autoría propia.

En la (ilustración 8) y (ilustración 9), se ubica en la carpeta y se accede al repositorio en Git utilizando el comando `git clone https://github.com/demondcn/LT`. Esto descarga todos los archivos del repositorio remoto a la máquina local

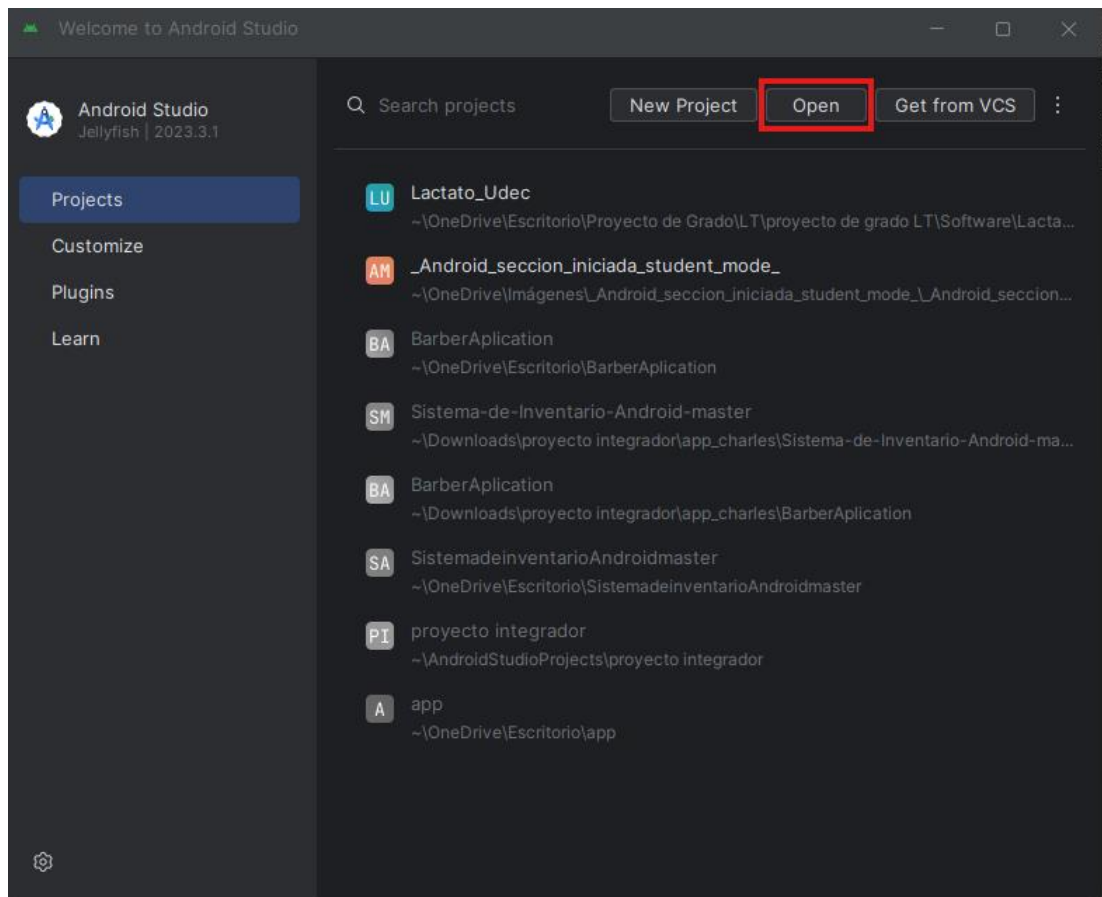
2. Acceder a la Carpeta del Proyecto por Android Studio:

Ilustración 10



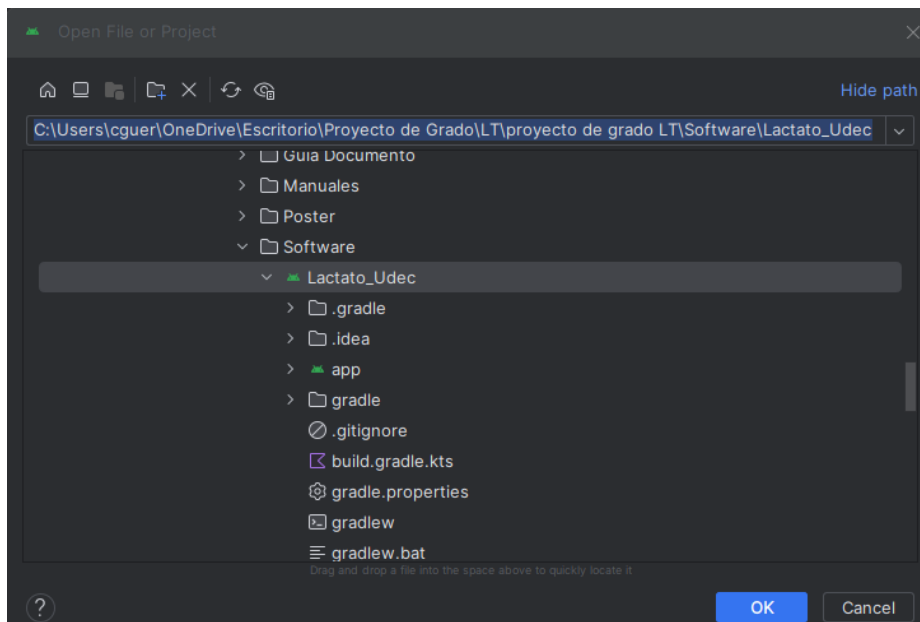
Fuente: Autoría propia.

Ilustración 11



Fuente: Autoría propia.

Ilustración 12

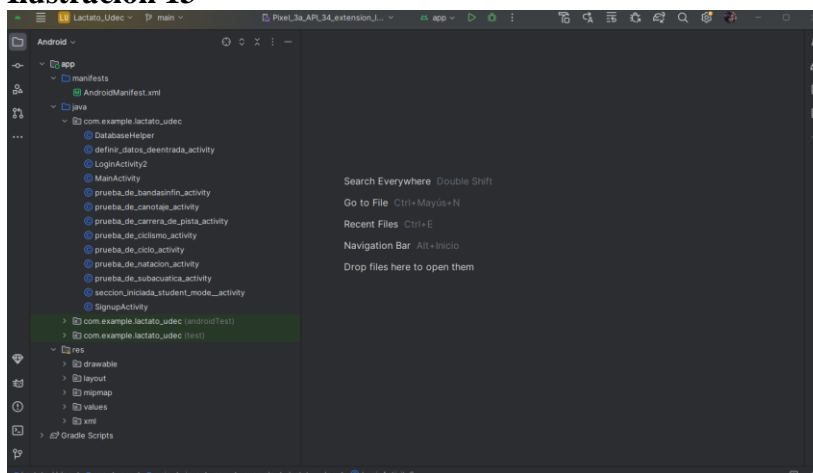


Fuente: Autoría propia.

En la (ilustración 10), verificamos que la carpeta LT haya sido descargada correctamente. En la (ilustración 11), hacemos clic en el botón "Open" en Android Studio e navegamos hasta la carpeta LT. Dentro de ella, buscamos la ruta: \LT\proyecto de grado LT\Software\ Lactato_Udec. La (ilustración 12) muestra detalladamente cómo se lleva a cabo este proceso.

3. Compilar:

Ilustración 13



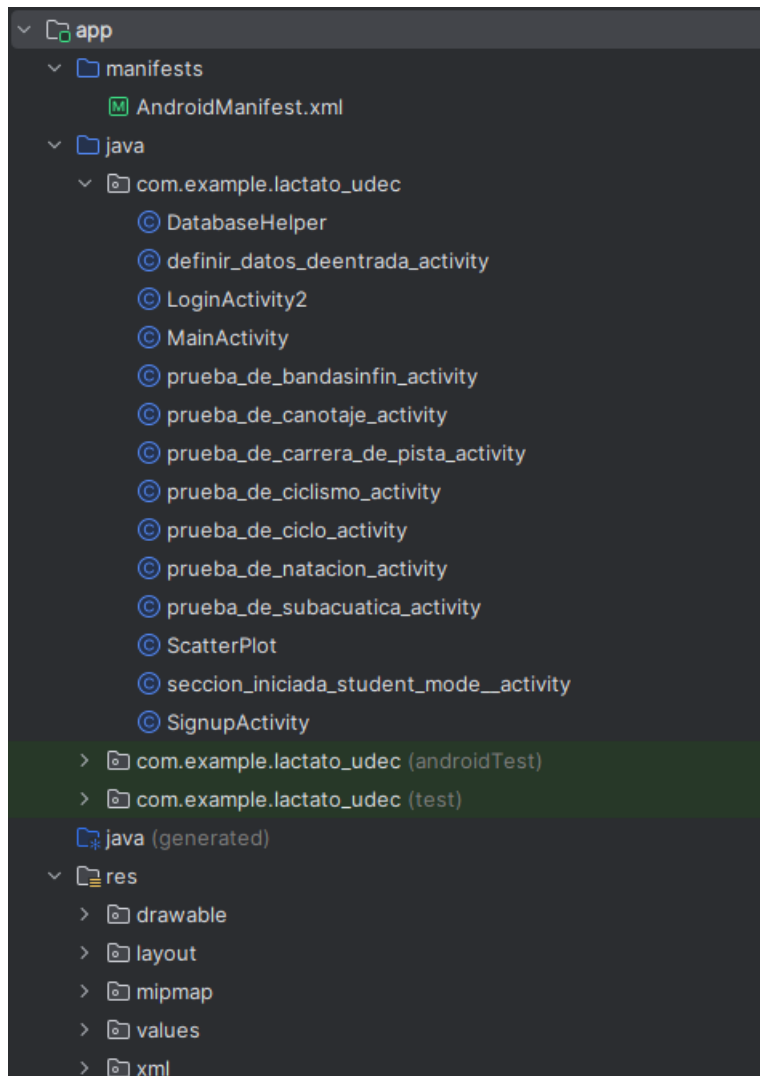
Fuente: Autoría propia.

En la (ilustración 13), verificamos que haya sido completado correctamente la compilación del gradle. En la (ilustración 13),Podemos ver que se ve todo el proyecto dentro de la carpeta app.

Estructura Del Proyecto

Para el desarrollo del software, se identificó una arquitectura modular y organizada, facilitando el desarrollo, mantenimiento y escalabilidad del software.

Ilustración 14



Fuente: Autoría propia.

En la (ilustración 14) se puede visualizar la estructura del proyecto. A continuación, detallaremos cada una de sus partes de manera específica.

Directorio app

Este es el directorio principal del módulo de la aplicación. Contiene todos los archivos y recursos necesarios para construir tu aplicación.

Subdirectorio manifests

AndroidManifest.xml: Este archivo es fundamental para cualquier aplicación Android.

Define la estructura básica y los componentes de la aplicación, como actividades, servicios, receptores de difusión, y proveedores de contenido. También especifica los permisos necesarios y otras características de la aplicación.

Subdirectorio java

Este directorio contiene el código fuente Java de la aplicación.

- **com.example.lactato_udec:** Este es el paquete principal de la aplicación. Contiene las siguientes clases y actividades:
 - **DatabaseHelper:** Clase para la gestión de la base de datos SQLite.
 - **definir_datos_deentrada_activity:** Actividad relacionada con la definición de datos de entrada del deportista.
 - **LoginActivity2:** Actividad para el inicio de sesión del deportista.
 - **prueba_de_bandasinfinito_activity:** Actividad relacionada con una prueba específica de bandas sin fin.
 - **prueba_de_canoaje_activity:** Actividad relacionada con la prueba de canotaje.
 - **prueba_de_carrera_de_pista_activity:** Actividad relacionada con la prueba de carrera de pista.
 - **prueba_de_ciclismo_activity:** Actividad relacionada con la prueba de ciclismo.
 - **prueba_de_ciclo_activity:** Actividad relacionada con la prueba de ciclo.
 - **prueba_de_natacion_activity:** Actividad relacionada con la prueba de natación.
 - **prueba_de_subacuatica_activity:** Actividad relacionada con la prueba subacuática.

- **ScatterPlot**: Actividad encargada de crear graficas de dispersión.
- **seccion_iniciada_student_mode_activity**: Actividad para la sección iniciada en modo estudiante.
- **SignupActivity**: Actividad de registro de nuevos usuarios.
- **com.example.lactato_udac (androidTest)**: Este paquete contiene pruebas instrumentadas, es decir, pruebas que se ejecutan en un dispositivo Android real o emulador.
- **com.example.lactato_udac (test)**: Este paquete contiene pruebas unitarias, es decir, pruebas que se ejecutan en la JVM local.

Subdirectorio Res

Este directorio contiene todos los recursos de la aplicación, como imágenes, diseños de UI, cadenas de texto, etc.

- **drawable**: Carpeta para recursos gráficos como imágenes.
- **layout**: Carpeta para archivos de diseño XML que describen las interfaces de usuario.
- **mipmap**: Carpeta para iconos de la aplicación en diferentes resoluciones.
- **values**: Carpeta para recursos de valores como cadenas de texto, estilos y temas.
- **xml**: Carpeta para otros archivos XML que no encajan en las otras categorías (como configuraciones adicionales).

Directorio Gradle Scripts

Este directorio contiene los scripts y archivos de configuración de Gradle, que es la herramienta de automatización de compilación utilizada por Android Studio.

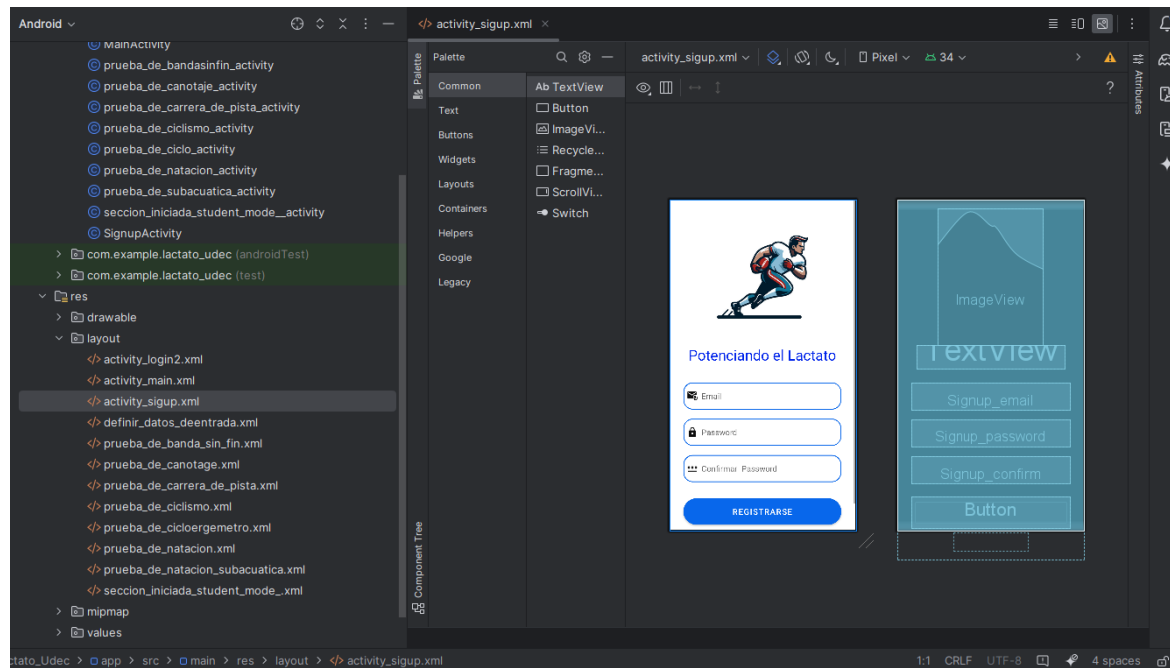
Subdirectorio layout

Este subdirectorio contiene los archivos de diseño en formato XML que definen la interfaz de usuario (UI) de las actividades y fragmentos en la aplicación.

- **activity_login2.xml**: Archivo de diseño para la actividad de inicio de sesión.
- **activity_signup.xml**: Archivo de diseño para la actividad de registro de nuevos usuarios. Aquí se define la interfaz con campos para el email, la contraseña, la confirmación de la contraseña y un botón de registro, como se muestra en la ilustración.

- **definir_datos_deentrada.xml:** Archivo de diseño para la actividad de definición de datos de entrada.
- **prueba_de_banda_sin_fin.xml:** Archivo de diseño para la actividad de prueba de banda sin fin.
- **prueba_de_canotaje.xml:** Archivo de diseño para la actividad de prueba de canotaje.
- **prueba_de_carrera_de_pista.xml:** Archivo de diseño para la actividad de prueba de carrera de pista.
- **prueba_de_ciclismo.xml:** Archivo de diseño para la actividad de prueba de ciclismo.
- **prueba_de_ciclogerometro.xml:** Archivo de diseño para la actividad de prueba de ciclogerometro.
- **prueba_de_natacion.xml:** Archivo de diseño para la actividad de prueba de natación.
- **prueba_de_natacion_subacuatica.xml:** Archivo de diseño para la actividad de prueba de natación subacuática.
- **seccion_iniciada_student_mode.xml:** Archivo de diseño para la actividad de la sección iniciada en modo estudiante.

Ilustración 15



Fuente: Autoría propia.

Como se puede ver en la (ilustración 15) Estos archivos definen la disposición de los elementos visuales en cada actividad, como botones, campos de texto, imágenes, etc. En la

interfaz de diseño de Android Studio, se pueden ver y editar visualmente estos elementos para construir la UI de la aplicación.

Explicación de Funcionalidad de clases y Actividades

DatabaseHelper.java

La clase DatabaseHelper facilita la gestión de una base de datos SQLite para almacenar y recuperar información de usuarios y sus datos relacionados en la aplicación. Cada método está diseñado para manejar operaciones específicas sobre las tablas allusers y userdata

Ilustración 16

```
no usages
public static final String databaseName = "Signup.db";

wikiky35
public DatabaseHelper(@Nullable Context context) {
    super(context, name: "Signup.db", factory: null, version: 1);
}
```

Fuente: Autoría propia.

En la (Ilustración 16) se pueden visualizar los constructores y atributos. La base de datos se llamará "Signup.db" y el constructor DatabaseHelper(Context context) la inicializa con el nombre "Signup.db" y la versión 1.

Métodos Sobrescritos DatabaseHelper

Ilustración 17

```
@Override 1 demandon
public void onCreate(SQLiteDatabase MyDB) {
    MyDB.execSQL("CREATE TABLE allusers (id INTEGER PRIMARY KEY AUTOINCREMENT, email TEXT UNIQUE, password TEXT);");
    MyDB.execSQL("CREATE TABLE userdata (id INTEGER PRIMARY KEY AUTOINCREMENT, user_id INTEGER, email TEXT, f);");
    MyDB.execSQL("CREATE TABLE scatterdata (id INTEGER PRIMARY KEY AUTOINCREMENT, user_id INTEGER, xData TEXT);");
}

@Override 10 usages 1 demandon
public void onUpgrade(SQLiteDatabase MyDB, int oldVersion, int newVersion) {
    MyDB.execSQL("drop Table if exists allusers");
    MyDB.execSQL("drop Table if exists userdata");
    MyDB.execSQL("drop Table if exists scatterdata");
    onCreate(MyDB);
}
```

Fuente: Autoría propia.

En la (Ilustración 17) se pueden visualizar los métodos sobrescritos, los cuales son:

- **onCreate(SQLiteDatabase MyDB):** Crea tres tablas, allusers, userdata y scatterdata, al inicializar la base de datos.
- **onUpgrade(SQLiteDatabase MyDB, int oldVersion, int newVersion):** Elimina las tablas existentes y las vuelve a crear en caso de una actualización de la base de datos.

Métodos Personalizados DatabaseHelper

Ilustración 18

```
33      >      public long insertScatterData(int user_id, String xData, String yData) {...}
43      >      public Cursor getAllScatterData(int user_id) {...}
47      >      public long insertUserData(int user_id, String fecha, String name, int age, int w
63      >      public long insertDate(String email, String password){...}
84      >      public int getUserIdByEmail(String email) {...}
97      >      public Boolean CheckUser(String email) {...}
105     >      public Boolean CheckUserPassword(String email, String password) {...}
113     >      public String getUserByNameById(int userId) {...}
126     >      public String getDatos(int userId, int n) {...}
173     >      public boolean areDatosDefinidos(int userId) {...}
183     >      public void setDatosDefinidos(int userId, boolean definidos) {...}
189
```

Fuente: Autoría propia.

En la (Ilustración 18) se pueden visualizar los métodos creados para la clase

DatabaseHelper, los cuales se explican a continuación:

- **insertScatterData:** Inserta datos para guardar los datos de creación de una gráfica.
- **getAllScatterData:** interactúa con la base de datos para verificar graficas ya creadas.
- **insertUserData:** Inserta datos del usuario en la tabla userdata.
- **insertDate:** Inserta un nuevo usuario en la tabla allusers y retorna el ID del nuevo usuario.
- **getUserIdByEmail:** Retorna el ID de un usuario basado en su correo electrónico.
- **CheckUser:** Verifica si un usuario con un correo electrónico específico existe en la base de datos.
- **CheckUserPassword:** Verifica si un usuario con un correo electrónico y contraseña específicos existe en la base de datos.
- **getUserByNameById:** Obtiene el nombre de usuario basado en su ID.
- **getDatos:** Obtiene un dato específico del usuario basado en su ID y un índice que determina qué campo se desea obtener.
- **areDatosDefinidos:** es responsable de verificar si ya se definieron datos.
- **setDatosDefinidos:** manda la confirmación de definición de datos.

SignupActivity.java

La clase SignupActivity gestiona el registro de nuevos usuarios en la aplicación. Se asegura de que los campos requeridos estén completos, valida la coincidencia de contraseñas, verifica la existencia del correo electrónico en la base de datos, inserta los datos del usuario y redirige al usuario a la pantalla de inicio de sesión en caso de éxito. Además, proporciona una forma fácil de redirigir a los usuarios existentes a la pantalla de inicio de sesión.

Ilustración 19

```
public class SignupActivity extends AppCompatActivity {  
  
    7 usages  
    ActivitySigupBinding binding;  
  
    3 usages  
    DatabaseHelper db;  
}
```

Fuente: Autoría propia.

En la (Ilustración 19) se pueden visualizar los constructores y atributos:

- **binding:** Variable para manejar el enlace de vistas (View Binding) para la actividad de registro.
- **db:** Instancia de DatabaseHelper para interactuar con la base de datos.

Métodos Sobrescritos SignupActivity

Ilustración 20

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);  
    setContentView(R.layout.activity_sigup);  
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {  
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());  
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);  
        return insets;  
    });  
}
```

Fuente: Autoría propia.

En la (Ilustración 20) se puede visualizar el método sobrescrito onCreate(Bundle

savedInstanceState), el cual inicializa la interfaz de usuario y configura los escuchadores de eventos para los elementos de la vista.

Manejo del Botón de Registro SignupActivity

Ilustración 21

```
37      binding.SignupButton.setOnClickListener(new View.OnClickListener(){
38          @Override
39          public void onClick(View view){
40              String email = binding.SignupEmail.getText().toString();
41              String password = binding.SignupPassword.getText().toString();
42              String confirmPassword = binding.SignupConfirm.getText().toString();
43              if(email.equals("") || password.equals("") || confirmPassword.equals(""))
44                  Toast.makeText(context: SignupActivity.this, text: "Todos los campos son requeridos", Toast.LENGTH_SHORT);
45              else {
46                  if(password.equals(confirmPassword)){
47                      Boolean checkUserEmail = db.CheckUser(email);
48                      if (!checkUserEmail){
49                          long userId = db.insertDate(email, password);
50                          if (userId != -1){
51                              Toast.makeText(context: SignupActivity.this, text: "Los datos han sido guardados correctamente", Toast.LENGTH_SHORT);
52                              Intent intent = new Intent(getApplicationContext(), LoginActivity2.class);
53                              intent.putExtra(name: "userId", userId);
54                              intent.putExtra(name: "email", email);
55                              startActivity(intent);
56                          }
57                      } else {
58                          Toast.makeText(context: SignupActivity.this, text: "Ha ocurrido un error al guardar los datos", Toast.LENGTH_SHORT);
59                      }
60                  } else {
61                      Toast.makeText(context: SignupActivity.this, text: "El email ya existe, intenta con otro", Toast.LENGTH_SHORT);
62                  }
63              }
64              } else {
65                  Toast.makeText(context: SignupActivity.this, text: "Las contraseñas no coinciden", Toast.LENGTH_SHORT);
66              }
67          }
68      });
```

Fuente: Autoría propia.

En la (Ilustración 21) se puede visualizar el código del botón de registro, cuyas funciones son las siguientes:

- **Recopilación de Datos:** Recoge el correo electrónico, la contraseña y la confirmación de contraseña ingresados por el usuario.
- **Validación de Campos Vacíos:** Verifica que ninguno de los campos esté vacío.
- **Validación de Contraseñas:** Comprueba que la contraseña y la confirmación de contraseña coincidan.
- **Verificación de Existencia de Usuario:** Usa db.CheckUser(email) para verificar si el correo electrónico ya está registrado.
- **Inserción de Datos:** Si el correo electrónico no existe, inserta los datos del usuario en la base de datos usando db.insertDate(email, password).
- **Navegación a Login:** Si la inserción es exitosa, muestra un mensaje de éxito y redirige al usuario a la actividad de inicio de sesión (LoginActivity2), pasando el ID de usuario y el correo electrónico como extras en el Intent.

Redirección a la Pantalla de Login SignupActivity

Ilustración 22

```
71      wikiky35
      binding.loginRedirectText.setOnClickListener(new View.OnClickListener() {
72          wikiky35
          @Override
73      public void onClick(View v) {
74          Intent intent = new Intent(getApplicationContext(), LoginActivity2.class);
75          startActivity(intent);
76      }
77  });
```

Fuente: Autoría propia.

En la (Ilustración 22) se puede visualizar el código del botón de redirección a la pantalla de inicio de sesión, en el cual `binding.loginRedirectText.setOnClickListener` configura un escuchador de clics para redirigir al usuario a la pantalla de inicio de sesión (`LoginActivity2`) cuando se hace clic en el texto correspondiente.

LoginActivity2.java

La clase `LoginActivity2` gestiona el inicio de sesión de los usuarios en la aplicación. Se asegura de que los campos requeridos estén completos, verifica las credenciales del usuario, obtiene el ID del usuario, lo guarda en `SharedPreferences` y redirige al usuario a la actividad principal en caso de éxito. Además, proporciona una forma fácil de redirigir a los usuarios a la pantalla de registro si aún no tienen una cuenta.

Ilustración 23

```
16  public class LoginActivity2 extends AppCompatActivity {
17      ActivityLogin2Binding binding;
18      DatabaseHelper databaseHelper;
```

Fuente: Autoría propia.

En la (Ilustración 23) se pueden visualizar los constructores y atributos:

- **binding**: Variable para manejar el enlace de vistas (View Binding) para la actividad de registro.
- **databaseHelper**: Instancia de DatabaseHelper para interactuar con la base de datos.

Métodos Sobrescritos LoginActivity2

Ilustración 24

```
protected void onCreate(Bundle savedInstanceState) {  
  
    super.onCreate(savedInstanceState);  
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);  
    setContentView(R.layout.activity_login2);  
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {  
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());  
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);  
        return insets;  
    });  
  
    binding = ActivityLogin2Binding.inflate(getLayoutInflater());  
    setContentView(binding.getRoot());  
  
    databaseHelper = new DatabaseHelper( context: this);  
}
```

Fuente: Autoría propia.

En la (Ilustración 24) se puede visualizar el método sobrescrito onCreate(Bundle savedInstanceState), el cual inicializa la interfaz de usuario y configura los escuchadores de eventos para los elementos de la vista.

Manejo del Botón de Inicio de Sesión LoginActivity2

Ilustración 25

```
37 binding.LoginButton.setOnClickListener(new View.OnClickListener() {
38     + demondcn +1
39     @Override
40     public void onClick(View v) {
41         String email = binding.LoginEmail.getText().toString();
42         String password = binding.LoginPassword.getText().toString();
43
44         if (email.equals("") || password.equals("")) {
45             Toast.makeText(context: LoginActivity2.this, text: "Todos los campos son requeridos", Toast.LENGTH_SHORT);
46         } else {
47             Boolean checkCredentials = databaseHelper.CheckUserPassword(email, password);
48             if (checkCredentials) {
49                 // Obtener userId después de la autenticación exitosa
50                 int userId = databaseHelper.getUserIdByEmail(email);
51
52                 // Guardar el ID del usuario en SharedPreferences
53                 SharedPreferences sharedPreferences = getSharedPreferences(name: "user_prefs", MODE_PRIVATE);
54                 SharedPreferences.Editor editor = sharedPreferences.edit();
55                 editor.putInt("user_id", userId);
56                 editor.apply();
57
58                 // Iniciar la actividad seccion_iniciada_student_mode__activity y pasar el ID del usuario
59                 Intent intent = new Intent(getApplicationContext(), seccion_iniciada_student_mode__activity.class);
60                 intent.putExtra(name: "user_id", userId);
61                 startActivity(intent);
62                 finish(); // Finalizar LoginActivity2 para que no se pueda volver atrás con el botón de retroceso
63             } else {
64                 Toast.makeText(context: LoginActivity2.this, text: "Credenciales incorrectas", Toast.LENGTH_SHORT);
65             }
66         }
67     });
```

Fuente: Autoría propia.

En la (Ilustración 25) se puede visualizar el código del botón de Inicio de Seccion, cuyas funciones son las siguientes:

- **Recopilación de Datos:** Recoge el correo electrónico y la contraseña ingresados por el usuario.
- **Validación de Campos Vacíos:** Verifica que ninguno de los campos esté vacío.
- **Verificación de Credenciales:** Usa `databaseHelper.CheckUserPassword(email, password)` para verificar si las credenciales son correctas.
- **Obtención del ID de Usuario:** Si las credenciales son correctas, obtiene el ID del usuario usando `databaseHelper.getUserIdByEmail(email)`.
- **Guardado en SharedPreferences:** Guarda el ID del usuario en `SharedPreferences` para persistencia.
- **Navegación a la Actividad Principal:** Inicia la actividad `seccion_iniciada_student_mode__activity` pasando el ID del usuario como extra en el `Intent` y finaliza `LoginActivity2`.

Redirección a la Pantalla de Registro LoginActivity2

Ilustración 26

```
68      binding.SignUpRedirectText.setOnClickListener(new View.OnClickListener() {  
69          wikiky35 +1  
69          @Override  
70          public void onClick(View v) {  
71              Intent intent = new Intent( packageContext: LoginActivity2.this, SignupActivity.class);  
72              startActivity(intent);  
73          }  
74      });
```

Fuente: Autoría propia.

En la (Ilustración 26) se puede visualizar el código del botón de redirección a la pantalla de inicio de sesión, en el cual `binding.SignUpRedirectText.setOnClickListener` configura un escuchador de clics para redirigir al usuario a la pantalla de registro (`SignupActivity`) cuando se hace clic en el texto correspondiente.

seccion_iniciada_student_mode__activity.java

La clase `seccion_iniciada_student_mode__activity` gestiona la interfaz donde los estudiantes pueden seleccionar diferentes pruebas para realizar y definir sus datos. Esta clase recibe el ID del usuario mediante un `Intent`, inicializa las vistas correspondientes y configura los escuchadores de eventos para los elementos interactivos.

Ilustración 27

```
31      private TextView carrera_en_pista;  
31      1 usage  
32      private TextView banda_sin_fin;  
32      1 usage  
33      private View rectangle_1;  
33      2 usages  
34      private TextView __definir_datos__;  
34      10 usages  
35      private int userId;
```

Fuente: Autoría propia.

En la (Ilustración 27) se pueden visualizar los constructores y atributos:

- **userId:** Variable que almacena el ID del usuario recibido desde el `Intent`.

- **Vistas (TextViews, ImageViews, y Views):** Variables que representan los elementos de la interfaz definidos en el archivo XML correspondiente.

Métodos Sobrescritos seccion_iniciada_student_mode__activity

Ilustración 28

```
@Override
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.seccion_iniciada_student_mode_);
    Intent intent = getIntent();
    if (intent != null && intent.hasExtra( name: "user_id")) {
        userId = intent.getIntExtra( name: "user_id", defaultValue: -1); // Asigna el userId globalmente
    } else {
        // Manejar el caso cuando no se proporciona userId
        userId = -1; // Asigna un valor por defecto o maneja el caso según tu lógica
    }

    databaseHelper = new DatabaseHelper( context: this);
    datosDefinidos = databaseHelper.areDatosDefinidos(userId);
    _bg__seccion_iniciada_student_mode__ek2 = (View) findViewById(R.id._bg__seccion_iniciada_student_mode__e
    selecciona_la_prueba_a_realizar = (TextView) findViewById(R.id.selecciona_la_prueba_a_realizar);
    __ya_definiste_tus_datos_ = (TextView) findViewById(R.id.__ya_definiste_tus_datos_);
    image_5 = (ImageView) findViewById(R.id.image_5);
    image_6 = (ImageView) findViewById(R.id.image_6);
    image_7 = (ImageView) findViewById(R.id.image_7);
}
```

Fuente: Autoría propia.

En la (Ilustración 28) se puede visualizar el método sobrescrito onCreate(Bundle savedInstanceState), el cual inicializa la interfaz de usuario y configura la conexión a la base de datos y los escuchadores de eventos para los elementos de la vista.

Configuración de Listeners para Botones seccion_iniciada_student_mode__activity

Ilustración 29

```
78 View.OnClickListener pruebasClickListener = new View.OnClickListener() {
79     @Override
80     public void onClick(View v) {
81         if (!datosDefinidos) {
82             Toast.makeText(context, seccion_iniciada_student_mode__activity.this, text: "Debes definir tus datos antes de continuar", Toast.LENGTH_SHORT).show();
83             return;
84         }
85         Intent intent = null;
86         int id = v.getId();
87         if (id == R.id.image_6) {
88             intent = new Intent(context, prueba_de_natacion_activity.class);
89         } else if (id == R.id.image_7) {
90             intent = new Intent(context, prueba_de_subacuatica_activity.class);
91         } else if (id == R.id.image_8) {
92             intent = new Intent(context, prueba_de_ciclismo_activity.class);
93         } else if (id == R.id.image_9) {
94             intent = new Intent(context, prueba_de_canoaje_activity.class);
95         } else if (id == R.id.image_10) {
96             intent = new Intent(context, prueba_de_bandasinfinitas_activity.class);
97         } else if (id == R.id.image_11) {
98             intent = new Intent(context, prueba_de_ciclo_actividad.class);
99         } else if (id == R.id.image_12) {
100             intent = new Intent(context, prueba_de_carrera_de_100m.class);
101         }
102         if (intent != null) {
103             intent.putExtra("userId", userId);
104             startActivity(intent);
105         }
106     }
107 }

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 1 && resultCode == RESULT_OK) {
        datosDefinidos = databaseHelper.areDatosDefinidos(userId); // Verifica el estado en la base de datos
        Toast.makeText(this, "Datos definidos exitosamente", Toast.LENGTH_SHORT).show();
    }
}
```

Fuente: Autoría propia.

En la (Ilustración 29) se puede visualizar la configuración de los escuchadores de eventos para los botones de la interfaz y el método onActivityResult. Los escuchadores gestionan la navegación a diferentes actividades dependiendo de la selección del usuario y el método se encarga de verificar de que ya se definieron datos.

- **definir_datos**: Redirige a la actividad definir_datos_deentrada_activity.
- **image_6**: Redirige a la actividad prueba_de_natacion_activity.
- **image_7**: Redirige a la actividad prueba_de_subacuatica_activity.
- **image_8**: Redirige a la actividad prueba_de_ciclismo_activity.
- **image_9**: Redirige a la actividad prueba_de_canoaje_activity.

- **image_10:** Redirige a la actividad prueba_de_bandasinfinito_activity.
- **image_11:** Redirige a la actividad prueba_de_ciclo_activity.
- **image_12:** Redirige a la actividad prueba_de_carrera_de_pista_activity.

definir_datos_deentrada_activity.java

La clase definir_datos_deentrada_activity maneja la interfaz donde los usuarios pueden ingresar sus datos personales y seleccionar opciones relacionadas con su género, etapa de entrenamiento y deporte o evento. Esta clase recibe el ID del usuario mediante un Intent, inicializa las vistas correspondientes y configura los escuchadores de eventos para los elementos interactivos.

Ilustración 30

```

67     private View mitad1v;
        2 usages
68     private View mitad2v;
        2 usages
69     DatabaseHelper databaseHelper;
        3 usages
70     private int userId;
        1 usage
71     private String email;

```

Fuente: Autoría propia.

En la (Ilustración 30) se pueden visualizar los constructores y atributos:

- **userId:** Variable que almacena el ID del usuario recibido desde el Intent.
- **email:** Variable que almacena el correo electrónico del usuario.
- **DatabaseHelper databaseHelper:** Objeto que maneja la conexión con la base de datos.
- **EditText, TextView, ImageView, View:** Variables que representan los elementos de la interfaz definidos en el archivo XML correspondiente..

Métodos Sobrescritos definir_datos_deentrada_activity

Ilustración 31

```
79      @Override
80      public void onCreate(Bundle savedInstanceState) {
81
82          super.onCreate(savedInstanceState);
83          setContentView(R.layout.definir_datos_deentrada);
84          databaseHelper = new DatabaseHelper( context: this);
85          // Obtener el userId y el email del intent
86          Intent intent = getIntent();
87          if (intent != null && intent.hasExtra( name: "userId")) {
88              userId = intent.getIntExtra( name: "userId", defaultValue: -1);
89          } else {
90              // Manejar el caso cuando no se proporciona userId
91              Toast.makeText( context: this, text: "No se proporcionó el ID de usuario.", Toast.LENGTH_SHORT).show();
92              finish(); // Finalizar la actividad si no hay userId válido
93          }
94          email = intent.getStringExtra( name: "email");
95          _bg__definir_datos_deentrada_ek2 = (View) findViewById(R.id._bg__definir_datos_deentrada_ek2);
96          llena_tus_datos_correspondientes = (TextView) findViewById(R.id.llena_tus_datos_correspondientes);
97          __ya_definiste_tus_datos_ = (TextView) findViewById(R.id.__ya_definiste_tus_datos_);
98          tu_genero = (TextView) findViewById(R.id.tu_genero);
99          tu_edad_ = (TextView) findViewById(R.id.tu_edad_);
```

Fuente: Autoría propia.

En la (Ilustración 31) se puede visualizar el método sobrescrito onCreate(Bundle savedInstanceState), el cual inicializa la interfaz de usuario y configura los escuchadores de eventos para los elementos de la vista.

Configuración de Listeners para Botones definir_datos_deentrada_activity

Ilustración 32

```
142      >      mitad1v.setOnClickListener(new View.OnClickListener() {...});
143      >
144      >      mitad2v.setOnClickListener(new View.OnClickListener() {...});
145      >
146      >      // Asignación de listeners a los botones de grupo Seleccion Etapa
147      >
148      >      findViewById(R.id.rectangle_1_ek5).setOnClickListener(new View.OnClickListener() {...});
149      >
150      >      findViewById(R.id.rectangle_1_ek1).setOnClickListener(new View.OnClickListener() {...});
151      >
152      >      findViewById(R.id.rectangle_1_ek9).setOnClickListener(new View.OnClickListener() {...});
153      >
154      >      // Asignación de listeners a los botones del grupo de Seleccion Deporte o Evento
155      >
156      >      findViewById(R.id.rectangle_1_ek8).setOnClickListener(new View.OnClickListener() {...});
157      >
158      >      findViewById(R.id.rectangle_1_ek2).setOnClickListener(new View.OnClickListener() {...});
```

Fuente: Autoría propia.

En la (Ilustración 32) se puede visualizar la configuración de los escuchadores de eventos para los botones de la interfaz. Estos escuchadores gestionan la selección de género, etapa de entrenamiento y deporte o evento dependiendo de la selección del usuario.

- **mitad1v:** Listener para la selección del género masculino. Cambia la imagen y actualiza las variables de estado seleccionHombre y seleccionMujer.
- **mitad2v:** Listener para la selección del género femenino. Cambia la imagen y actualiza las variables de estado seleccionHombre y seleccionMujer.
- **rectangle_1_ek5, rectangle_1_ek1, rectangle_1_ek9:** Listeners para seleccionar la etapa de entrenamiento (preparación, precompetencia, competencia). Llamam al método manageButtonSelection para gestionar la selección de botones.
- **rectangle_1_ek8, rectangle_1_ek2, rectangle_1_ek3, rectangle_1_ek4:** Listeners para seleccionar el deporte o evento (resistencia, pelotas, combate, velocidad/fuerza rápida/anaeróbico). Llamam al método manageButtonSelection para gestionar la selección de botones.
- **guardar:** Listener que llama al método saveUserData para guardar los datos ingresados por el usuario en la base de datos

Métodos Personalizados definir_datos_deentrada_activity

Ilustración 33

```

1 usage  👤 demondcn *
257      > private void saveUserData() {...}
7 usages  👤 demondcn
291      > private void manageButtonSelection(View clickedView, View lastSelecte
300
2 usages  👤 demondcn
301      > public void FuncionBotonesSeleccion(View view, boolean colorOriginal,

```

Fuente: Autoría propia.

En la (Ilustración 33) se pueden visualizar los siguientes métodos:

- **Método saveUserData:** Obtiene la fecha actual y los datos ingresados por el usuario. Verifica que todos los campos estén completos. Guarda los datos en la base de datos usando el objeto databaseHelper. Muestra un mensaje de confirmación y finaliza la actividad si los datos se guardan correctamente.
- **Método manageButtonSelection:** Gestiona la selección y desección de botones, cambiando su apariencia según el estado actual.
- **Método FuncionBotonesSeleccion:** Cambia el fondo del botón según el estado de selección.

Código de pruebas

Las clases de pruebas manejan la interfaz donde los usuarios pueden ingresar datos relacionados con la prueba respectiva y generar un informe en formato PDF. Esta clase recibe el ID del usuario mediante un Intent, inicializa las vistas correspondientes y configura los escuchadores de eventos para los elementos interactivos.

Ilustración 34

```
double[][] EtapasIniciales = new double[100][100]; 50 usages
private int selectedEtapa = 0; 6 usages
private View lastSelectedButton; 6 usages
private int userId; 11 usages
private DatabaseHelper db; 9 usages
private String userName; 3 usages
```

Fuente: Autoría propia.

En la (Ilustración 34) se pueden visualizar los constructores y atributos:

- **userId:** Variable que almacena el ID del usuario recibido desde el Intent.
- **userName, userFecha, userEdad, userPeso, userTemperatura, userGenero, userPeriodo, userEvent:** Variables que almacenan datos personales del usuario recuperados de la base de datos.
- **DatabaseHelper db:** Objeto que maneja la conexión con la base de datos.
- **EditText, TextView, ImageView, View:** Variables que representan los elementos de la interfaz definidos en el archivo XML correspondiente.
- **EtapasIniciales:** Matriz que almacena los datos de las etapas iniciales de la prueba.
- **selectedEtapa, lastSelectedButton, n, m:** Variables de estado utilizadas para gestionar la selección y el flujo de las etapas de la prueba.

Métodos Sobrescritos Código de pruebas

Ilustración 35

```
@Override
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.prueba_de_carrera_de_pista);
    scatterPlot = new ScatterPlot(context: this);
    db = new DatabaseHelper(context: this);
    Intent intent = getIntent();
    if (intent != null && intent.hasExtra(name: "userId")) {
        userId = intent.getIntExtra(name: "userId", defaultValue: -1);
        userName = db.getDatos(userId, n: 1);
        userFecha = db.getDatos(userId, n: 2);
        userEdad = db.getDatos(userId, n: 3);
        userPeso = db.getDatos(userId, n: 4);
        userTemperatura = db.getDatos(userId, n: 5);
        userGenero = db.getDatos(userId, n: 6);
        userPeriodo = db.getDatos(userId, n: 7);
        userEvent = db.getDatos(userId, n: 8);
    } else {
        userId = -1; // Asigna un valor por defecto o maneja el caso según tu lógica
        userName = "Usuario desconocido";
    }

    textotipodesuperficie = (TextView) findViewById(R.id.textotipodesuperficie);
    _bg__prueba_de_carrera_de_pista_ek2 = (View) findViewById(R.id._bg__prueba_de_carrera_de_pista_ek2);
    image_6 = (ImageView) findViewById(R.id.image_6);
}
```

Fuente: Autoría propia.

En la (Ilustración 35) se puede visualizar el método sobrescrito onCreate(Bundle savedInstanceState), el cual inicializa la interfaz de usuario, recupera datos del Intent y configura los escuchadores de eventos para los elementos de la vista.

Configuración de Listeners para Botones Código de pruebas

Ilustración 36

```
111 //custom code goes here
112 > findViewById(R.id.tartan).setOnClickListener(new View.OnClickListener() {...});
121 > findViewById(R.id.ARCILLA).setOnClickListener(new View.OnClickListener() {...});
130 > findViewById(R.id.ASFALTO).setOnClickListener(new View.OnClickListener() {...});
139 > guardar.setOnClickListener(new View.OnClickListener() {...});
146 }
```

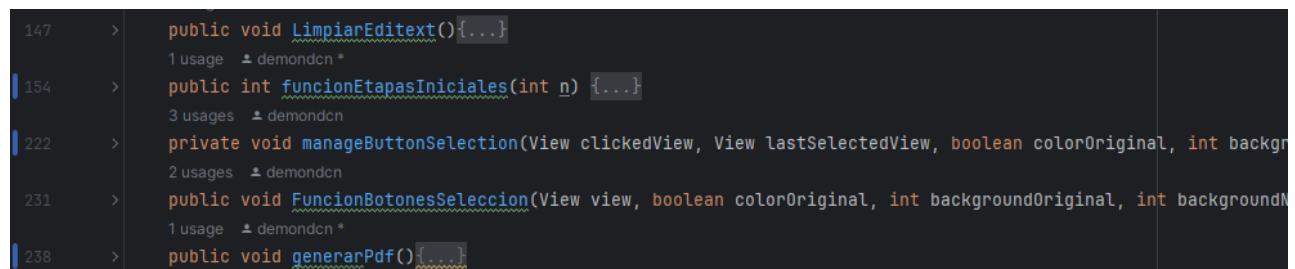
Fuente: Autoría propia.

En la (Ilustración 36) se puede visualizar la configuración de los escuchadores de

eventos para los botones de la interfaz. Estos escuchadores gestionan la selección de la etapa de entrenamiento y la acción de guardar los datos dependiendo de la selección del usuario tener en cuenta que estos botones cambian dependiendo la prueba.

Métodos Personalizados Código de pruebas

Ilustración 37



```
147 > public void limpiarEditText(){...}
154 > public int funcionEtapasIniciales(int n) {...}
222 > private void manageButtonSelection(View clickedView, View lastSelectedView, boolean colorOriginal, int backgr
231 > public void funcionBotonesSeleccion(View view, boolean colorOriginal, int backgroundOriginal, int backgroundN
238 > public void generarPdf(){...}
```

Fuente: Autoría propia.

En la (Ilustración 37) se pueden visualizar los siguientes métodos:

- **Método funcionEtapasIniciales:** Obtiene los datos ingresados por el usuario y los almacena en la matriz EtapasIniciales. Valida que todos los campos estén completos. Actualiza la interfaz de usuario según la etapa actual y muestra el mensaje adecuado en el botón de guardar. Genera el PDF al completar todas las etapas y finaliza la actividad.
- **Método manageButtonSelection:** Gestiona la selección y desección de botones, cambiando su apariencia según el estado actual.
- **Método FuncionBotonesSeleccion:** Cambia el fondo del botón según el estado de selección.
- **Método generarPdf:** Crea un documento PDF con los resultados de la prueba, incluyendo los datos personales del usuario y los resultados de las etapas de entrenamiento. Muestra un mensaje de confirmación y guarda el PDF en el dispositivo.

ScatterPlot.java

La clase ScatterPlot gestiona la creación de gráficos de dispersión (scatter plots) en un Bitmap para visualizar datos. Utiliza un objeto DatabaseHelper para guardar y recuperar datos de gráficos. Proporciona métodos para generar gráficos para un conjunto de datos o múltiples conjuntos en un solo gráfico.

Métodos Personalizados ScatterPlot

Ilustración 38

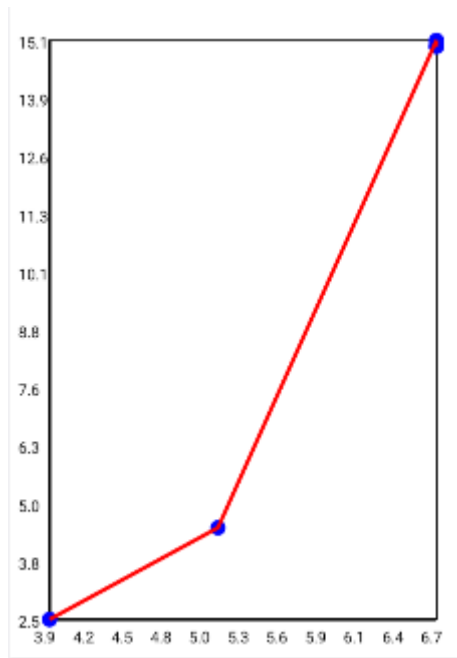
```
public class ScatterPlot { 2 usages 1 demandcn *  
  
    private DatabaseHelper dbHelper; 3 usages  
    public ScatterPlot(Context context) { dbHelper = new DatabaseHelper(context); }  
    public Bitmap createScatterPlotUniti(int width, int height, float[] xData, float[] yData) {...}  
    public Bitmap createScatterPlot(int width, int height, int user_id, float[] xData, float[] yData) {...}  
    private void saveScatterData(int user_id, float[] xData, float[] yData) {...}  
    private String floatArrayToString(float[] array) {...}  
    private float[] stringToFloatArray(String s) {...}  
    private float scale(float value, float min, float max, float newMin, float newMax) {...}  
    private float getMin(float[] data) {...}  
    private float getMax(float[] data) {...}  
}
```

Fuente: Autoría propia.

En la (Ilustración 38) se pueden visualizar los siguientes métodos:

- **Método createScatterPlotUniti:** Crea un gráfico de dispersión para un solo conjunto de datos.
- **Método createScatterPlot:** Crea un gráfico de dispersión con múltiples conjuntos de datos.
- **Método saveScatterData:** Guarda los datos del gráfico en la base de datos.
- **Método floatArrayToString:** Convierte un arreglo de flotantes en una cadena separada por comas.
- **Método stringToFloatArray:** Convierte una cadena separada por comas en un arreglo de flotantes.
- **Método scale:** Ajusta un valor dentro de un rango nuevo basado en un rango original.
- **Métodos getMin y getMax:** Encuentran el valor mínimo y máximo en un arreglo de datos, respectivamente.

Ilustración 39



Fuente: Autoría propia.

En la (ilustración 39) se puede ver la gráfica creada por la función.