

15-1-2023

Autoescuela IsCar

App de apoyo a autoescuelas



Contenido

1. INTRODUCCIÓN.....	3
1.1. MOTIVACION.....	3
1.2. ESTRUCTURACIÓN DE LA MEMORIA.....	3
1.3. OBJETIVOS GENERALES.....	3
1.4. PUBLICO OBJETIVO.....	3
2. DESARROLLO.....	4
2.1. TECNOLOGÍAS UTILIZADAS.....	4
o Lenguajes de programación:.....	4
o Programas de desarrollo:.....	4
o Librería Volley:.....	4
o MPAndroidChart:.....	4
o Arquitectura.....	5
2.2. SEGUIMIENTO.....	6
o Base de datps.....	6
o Primeros pasos cargando datos desde consulta PHP.....	6
o Recycler_Test. Realización del cuestionario.....	7
2.3. UML.....	8
2.3.1. DIAGRAMA DE PACKAGES.....	8
2.3.2. DIAGRAMA DE FLUJO E IMPLEMENTACION DE CLASES PADRE.....	9
2.4. DIAGRAMAS DE CLASES.....	10
2.4.1. CLASES RELACIONADAS CON TEMARIOS (PAGINAS WEB).....	10
2.4.2. CLASES RELACIONADAS CON CUESTIONARIO (TEST).....	10
2.4.3. CLASES RELACIONADAS CON LAS GRAFICAS.....	11
2.5. BASE DE DATOS.....	12
2.6. HTML:.....	13
2.7. WIREFRAMES.....	14
o Login.....	14
o Activity_temario_Test.....	15
o Wireframe común a todas las que tiene un elemento Recycler view de Seccion. (Respectivamente WF_Recycler_Secciones, WF_Recicler_Temas).....	16
o ActivityDirectorioSeccionesTemario (WF_Recycler_Secciones).....	17
o ActivityTemasTemario (WF_Recycler_Temas).....	18
o	18
o ActivityTemarioWeb.....	19

○	19
○ ActivityDirectorioSeccionesCuestionarios (WF_Recycler_Secciones).....	20
○	21
○ ActivityTemarioTemas_Cuestionario (WF_Recycler_Temas).....	21
○	21
○ Activity_Test:	22
○ Activity_Test_2.....	23
○ Activity_Secciones_Graficos (WF_Recycler_Secciones).....	24
○ Activity_GridLayout_graficos_secciones.....	25
○ Activity_Temas_Graficos (WF_Recycler_Temas).	26
○ Activity_GridLayout_graficos_temas.	27
2.8. NAVEGACIÓN.....	28
2.9. PATRONES DE CONSULTA CON VOLLEY.	29
○ Patrón de consulta de validación.....	29
○ Patrón de consulta de recuperación de datos.	31
○ Consultas PHP.	32
2.10. CREACIÓN DE GRÁFICAS CON MPANDROIDCHART.....	33
○ Conceptos básicos.	33
○ Representación de el proceso de configuración de los gráficos:.....	33
3. CONCLUSIONES.....	35
3.1. OBJETIVOS CUMPLIDOS.	35
3.2. OBJETIVOS CUMPLIDOS PARCIALMENTE.....	35
3.3. OBJETIVO NO CUMPLIDOS.....	36
3.4. PROPUESTAS DE MEJORA.	36
4. WEBGRAFÍA.	37

1. INTRODUCCIÓN.

1.1. MOTIVACION.

Elegí este proyecto, ya que me parece que es muy completo, entendiendo por completo que se han utilizado dos lenguajes de programación (Java y PHP), así como la implementación de dos API's (Volley y MPAndroidChart).

1.2. ESTRUCTURACIÓN DE LA MEMORIA.

- Incide
- Introducción
- Desarrollo
- Conclusiones
- Webgrafía

1.3. OBJETIVOS GENERALES

Mis objetivos son realizar una Aplicación de fácil utilización que permita tanto el estudio del material lectivo como la realización de pruebas para medir las aptitudes del usuario.

En cuanto a los objetivos de forma más precisas:

- Crear una base de datos de fácil utilización
- Crear una aplicación de navegación sencilla.
- Gestión de usuarios
- Hacer listados más dinámicos y visualmente más personalizados.
- Cuestionarios fáciles de rellenar.
- Gestión correcta de resultados.
- Visualización de los resultados en gráficos.

1.4. PUBLICO OBJETIVO.

Mi público objetivo principalmente es la gente entre 16-22 años, ya que es la edad media en la que se suele sacar el carnet de conducir.

2. DESARROLLO.

2.1. TECNOLOGÍAS UTILIZADAS.

○ Lenguajes de programación:

- Java
- PHP

○ Programas de desarrollo:

- Android Studio
- Dreamweaver (PHP)
- XAMMP.
- Photoshop.


○ Librería Volley:

Volley es una biblioteca HTTP que hace que la creación de redes para aplicaciones de Android sea más fácil y, lo que es más importante, más rápida. Volley está disponible en [GitHub](#).

Cita Webgrafía

Esta librería básicamente nos ayuda a conectar los datos en la arquitectura cliente servidor, permitiéndonos pasar parámetros a la consulta php, que en mi caso están guardadas en el directorio del programa XAMMP.

○ MPAndroidChart:

Un poderoso  Vista de gráfico de Android / biblioteca de vista de gráfico, que admite gráficos de barras lineales, circulares, de radar, de burbujas y de velas japonesas, así como escalas, panorámicas y animaciones.

Cita Webgrafía

Esta API es digamos el “Inflator” de mis datos de recogidos por consulta PHP y los pinta en modo de gráfico. Con la misma configuramos con Java que tipo de grafico queremos, como lo queremos mostrar y posicionar y añadimos nuestra información.

○ Arquitectura

Utilizo una arquitectura cliente servidor, en este caso el servidor se presenta en modo local para realizar el proyecto con el programa XAMMP.

Con la imagen de abajo explico un poco el flujo que seguirá mi aplicación y como se implementaría.

- **Navegador:** En este caso representa la Aplicación móvil.
- **Servidor web:** En este caso utilizo el programa XAMMP que es el que hace las veces de servidor web en el que se guardaran la base de datos las consultas PHP...
- **Página PHP:** Al enlazar con la consulta PHP guardada en el servidor en realidad lo que estoy haciendo es crear una visualización web de esa consulta, que en vez de mostrar por navegador, mandare a la aplicación como un json.
- **Resultado HTML:** En mi caso esto equivale a ese elemento json generado en la consulta y que se manda a la aplicación.



Cita Webgrafía

2.2. SEGUIMIENTO.

A nivel persona ha significado un gran reto, ya que no sabía nada de las consultas PHP, por lo cual me tuve que dedicar a aprender este lenguaje de programación.

○ Base de datos

Sin lugar a duda el trabajo más tedioso fue crear la base de datos y los HTML de los temarios, ya que la base de datos se ha creado manualmente añadiendo todas las preguntas a mano. Probablemente habrá una forma mucho más fácil de realizar esta tarea pero al inicio del proyecto no encontré una forma correcta de hacerlo. Muchas veces la falta de conocimiento se suple con trabajo duro, lo que llevo a la creación de las 110 tablas que contienen los test.

○ Primeros pasos cargando datos desde consulta PHP

El proceso de crear las activities y RecyclerView a nivel visual no ha supuesto mucho problema, no tanto así ha sido la implementación de los datos recogidos de la php. Un ejemplo muy claro y que he decidido dejar así para mostrar la evolución del proyecto es que al principio no sabía cómo hacer bien las consulta a pesar de contar con la API Volley de la que hay muchos tutoriales y documentación.

El siguiente ejemplo refleja este hecho:

Los primer RecyclerView que cree introduje los datos a mano con String sin hacer la búsqueda de los datos.

```
public void init()
{
    elementos = new ArrayList<RecyclerViewElemento>();
    //Cargar numero de secciones

    //Anyadimos elementos ejemplo a la lista TODO
    elementos.add((new RecyclerViewElemento( seccion: "Sección 1 ", estado: "nuevo")));
    elementos.add((new RecyclerViewElemento( seccion: "Sección 2 ", estado: "nuevo")));
    elementos.add((new RecyclerViewElemento( seccion: "Sección 3 ", estado: "nuevo")));
    elementos.add((new RecyclerViewElemento( seccion: "Sección 4 ", estado: "nuevo")));
    elementos.add((new RecyclerViewElemento( seccion: "Sección 5 ", estado: "nuevo")));

    //Adaptador que recibe la lista, context e interface OnClickListener de AdapterDirectorio
    AdapterDirectorioSecciones elementAdapter = new AdapterDirectorioSecciones(elementos, ctxt: this, new Adap

    @Override
    public void onItemClick(RecyclerViewElemento item) { goToTest(item); }
});
//Enlace con el recyclerView de ActivityDirectorio
RecyclerView rv = findViewById(R.id.reciclerTemarioTest);
//Evita invalidar el diseño cuando cambie el contenido del adaptador.
rv.setHasFixedSize(true);
//Hace el listado lineal (arriba-abajo)
rv.setLayoutManager(new LinearLayoutManager( context: this));
//Adaptador que utilizamos
rv.setAdapter(elementAdapter);
}
```

Al final del proyecto mi evolución es notable al cargar esto de manera automática y permitiendo que se carguen solo los elementos que tiene dato.

```
private void inflateViewRecycler() {
    //Adaptador que recibe la lista, context e interface OnClickListener de AdapterDirectorio
    AdapterSeccionesCarpetasGraficas elementAdapter = new AdapterSeccionesCarpetasGraficas(elementos, ctxt: this, new

    @Override
    public void onItemClick(String item) { goToTest(item); }
});
//Enlace con el recyclerView de ActivityDirectorio
RecyclerView rv = findViewById(R.id.reciclerTemarioTest);
//Evita invalidar el diseño cuando cambie el contenido del adaptador.
rv.setHasFixedSize(true);
//Hace el listado lineal (arriba-abajo)
rv.setLayoutManager(new LinearLayoutManager( context: this));
//Adaptador que utilizamos
rv.setAdapter(elementAdapter);
}
```

```
public void obtenerSecciones(String responseBody, String stringTabla) {
    StringRequest stringRequest = new StringRequest(POST, responseBody, new Response.Listener<String>() {

    @Override
    public void onResponse(String response) {
        try {
            JSONObject jsonObject = new JSONObject(response);
            JSONArray result = jsonObject.getJSONArray( name: "secciones");
            for (int i = 0; i < result.length(); i++) {
                JSONObject data = result.getJSONObject(i);
                String miElemento = data.optString( name: "seccion");
                elementos.add(miElemento);
            }
            inflateViewRecycler();
        }
    }
    });
}
```

○ Recycler_Test. Realización del cuestionario.

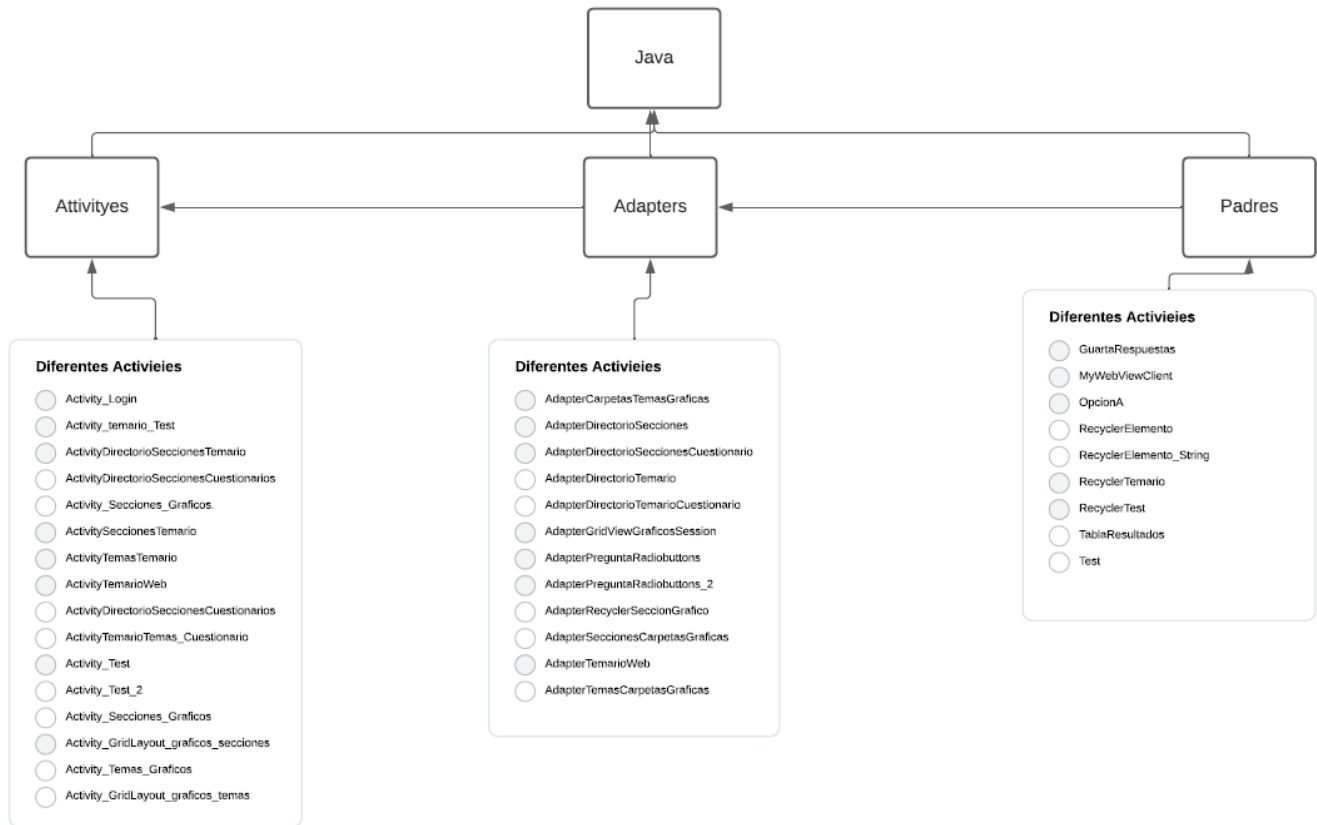
Este ha sido sin duda una de mis bestias negras. Ya que me encontré con el error de que al realizar el test había veces que el radio Button que pulso no es capturado como el Radio Button del elemento que veo, si no el del que se ha empezado a cargar de nuevo en el RecyclerView.

Para poder solventar este problema he separado más los elementos visuales para que tarde más en pasar al siguiente elemento del recycler y he añadido el método **“notifyDataSetChanged();”** con más frecuencia, esto permite que aunque a veces esto pase no se refleje en la vista y permita revisar de que preguntas de verdad se ha revisado una respuesta.

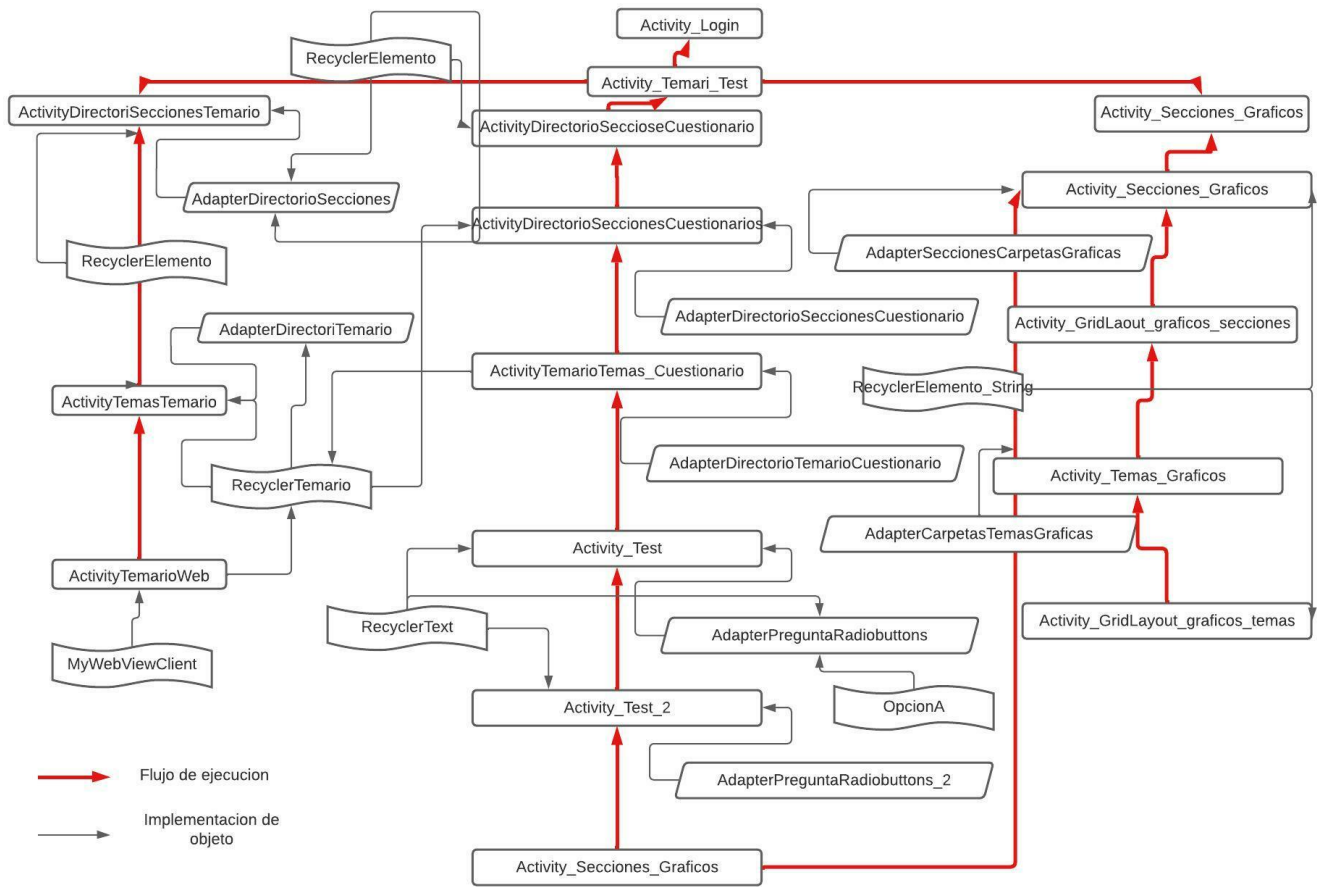
En definitiva ha sido un proceso largo en el que he dedicado muchas horas en las que resumiendo los mayores problemas que he encontrado es mi falta de formación, la que gracias a dios gracias a la realización del proyecto se ha visto muy mejorada.

2.3. UML.

2.3.1. DIAGRAMA DE PACKAGES.

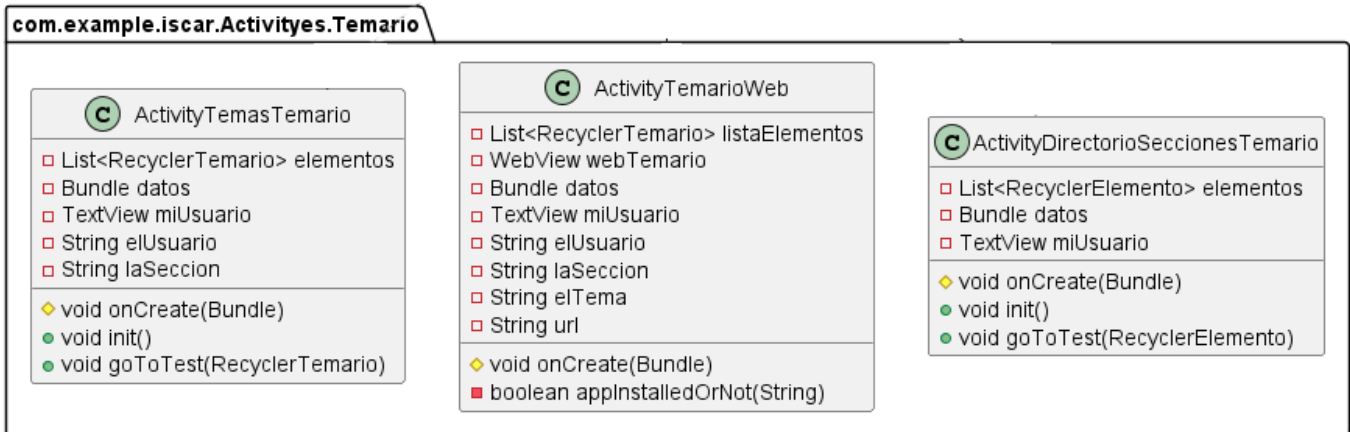


2.3.2. DIAGRAMA DE FLUJO E IMPLEMENTACION DE CLASES PADRE.



2.4. DIAGRAMAS DE CLASES.

2.4.1. CLASES RELACIONADAS CON TEMARIOS (PAGINAS WEB).



2.4.2. CLASES RELACIONADAS CON CUESTIONARIO (TEST).



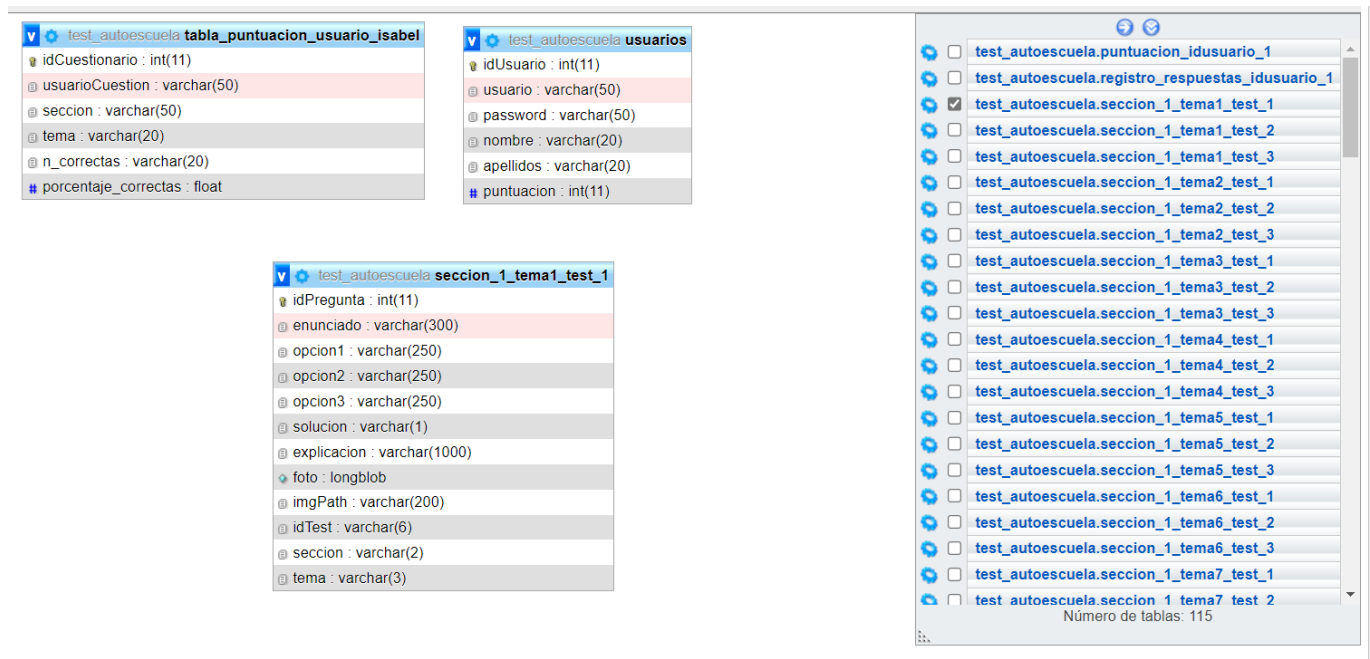
2.4.3. CLASES RELACIONADAS CON LAS GRAFICAS.



2.5. BASE DE DATOS.

En este caso mis tablas son muy simples no la he hecho con ningún tipo de relación entre ellas, ya que las relaciones necesarias las hare a nivel aplicación.

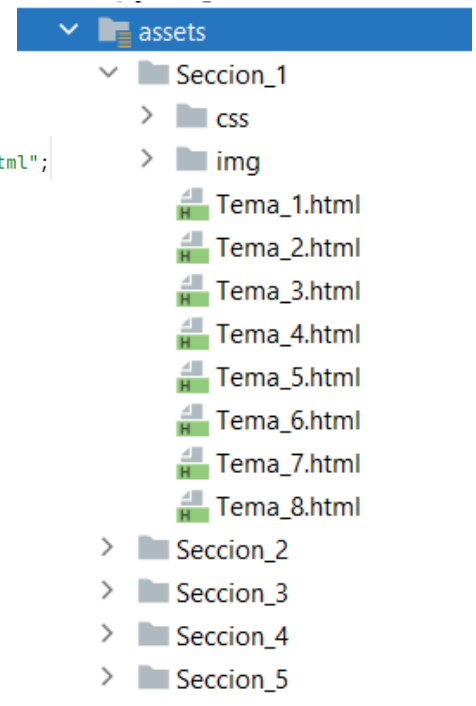
- **Usuarios:** Tabla creada para almacenar la gestión de usuarios.
- **Sección_x_temay_test_o:** Esta estructura de tablas representa cada uno de los cuestionarios. Cada uno de estos cuestionarios tiene 30 preguntas. En total de 5 Secciones de las cuales tienen entre 7 u 8 Temas los cuales cada uno tiene 3 test posibles que se cargaran de forma “Random”.
- **Tabla_puntuación_usuario_isabel:** Tabla que se creara para cada usuario en la que se registraran los resultados de sus cuestionarios.



2.6. HTML:

Los HTML son almacenados en la memoria del dispositivo en una carpeta especial para este tipo de recursos llamada assets, a los que se accederá a través del Path de la misma.

```
url = "file:///android_asset/" + "Seccion_" + countSeccion + "/" + "Tema_" + countTema + ".html";
```



Los HTML los cree con el editor del mismo de IDE Eclipse copiando la información de una página web.

Índice del contenido

1. Conceptos referidos al factor humano
2. Conceptos referidos al vehículo
 1. Clasificación de los vehículos según si tienen motor o no
 2. Vehículos especiales agrícolas
 3. Vehículos especiales para obras y servicios
 4. Conceptos básicos complementarios
3. Conceptos referidos a la vía
 1. Partes de la vía
 2. Otros conceptos referidos a la vía
4. Tipos de vías
 1. Vías urbanas
 2. Travesías
 3. Vías interurbanas

1. Conceptos referidos al factor humano

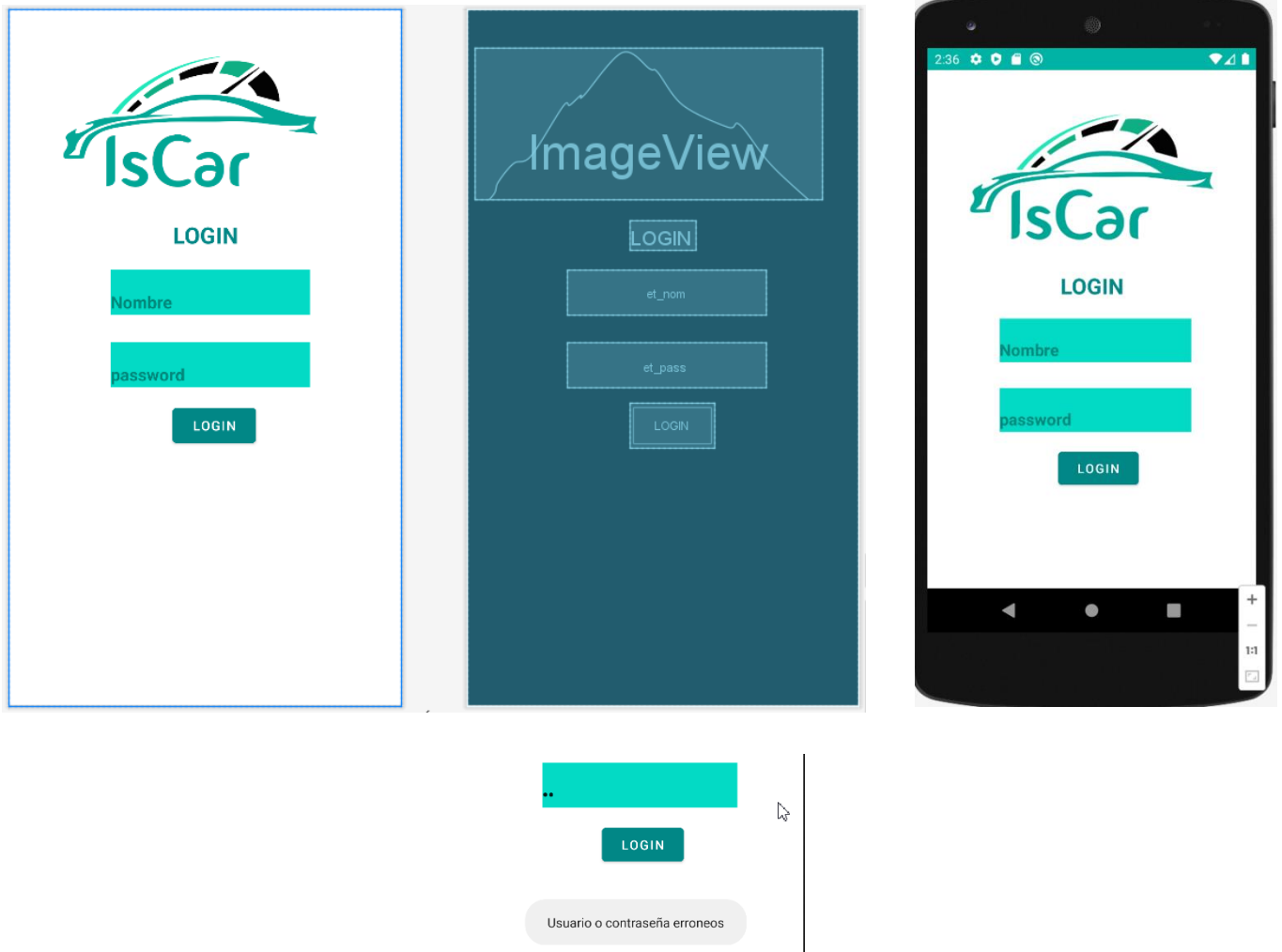
- **Conductor:** Es la persona que maneja la dirección o va al mando de un vehículo, o a cuyo cargo está un animal/es. En los vehículos de aprendizaje se considera conductor al formador que está a cargo de los mandos adicionales.
- **Conductor profesional:** Es aquel que tiene como actividad laboral la conducción de vehículos dedicados al transporte de mercancías y personas.
- **Conductor novel:** Es aquel conductor con un permiso de conducción de menos de un año de antigüedad.



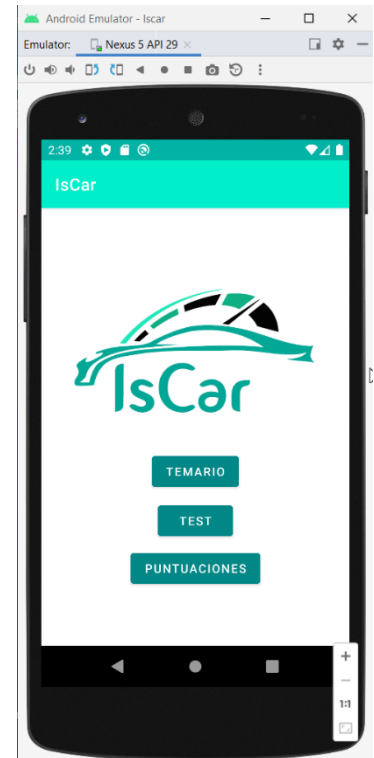
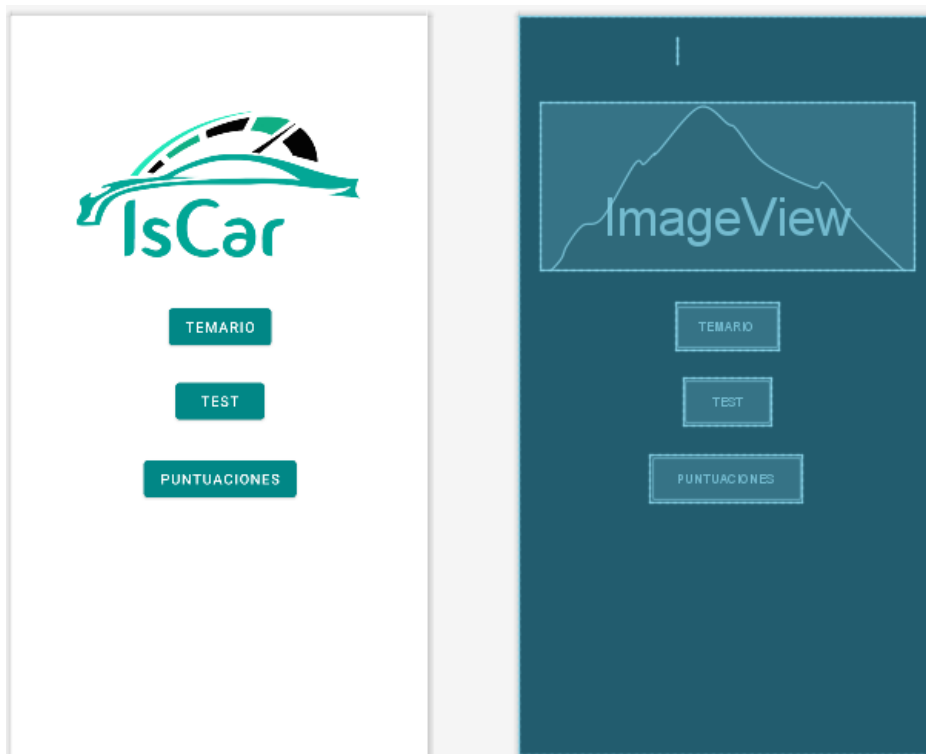
2.7. WIREFRAMES.

- Wireframes comunes a las tres remas de ejecución.

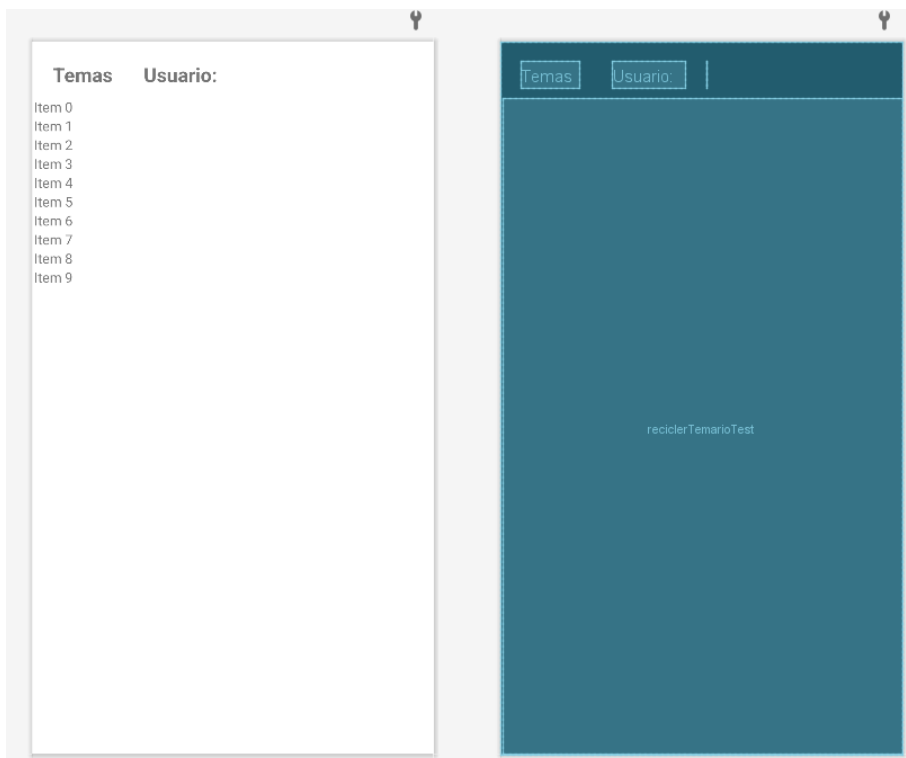
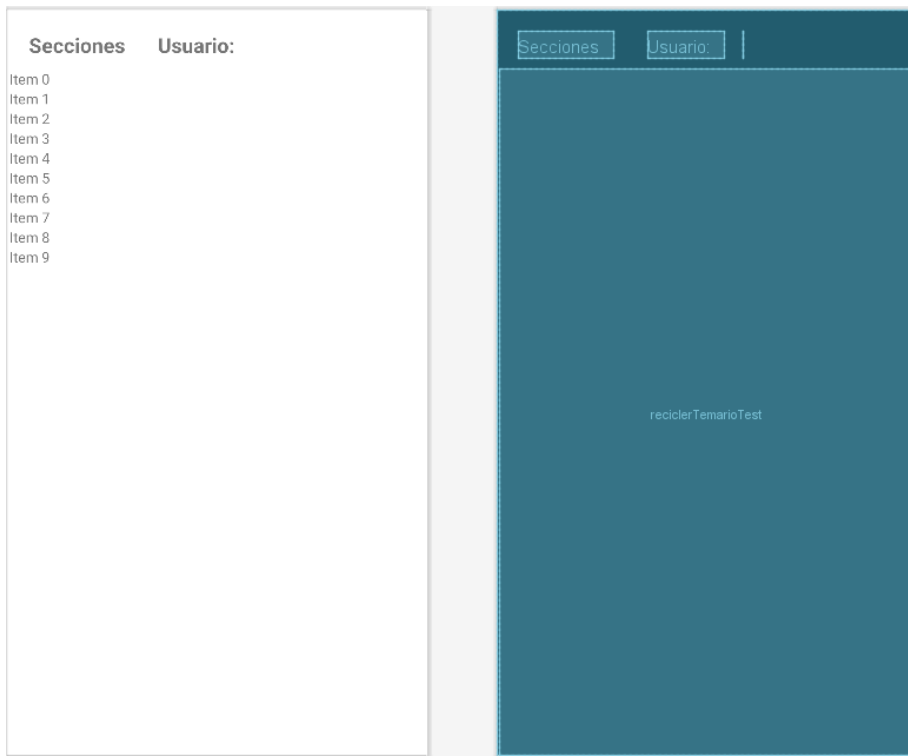
○ Login



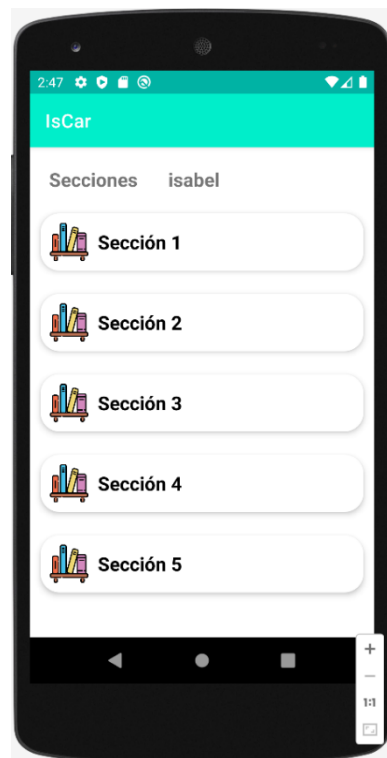
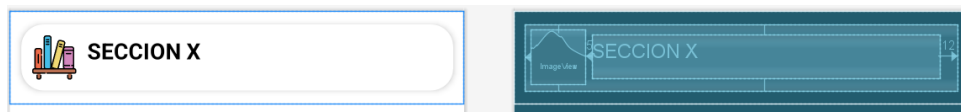
○ Activity_temario_Test.



- Wireframe común a todas las que tiene un elemento Recycler view de Sección. (Respectivamente WF_Recycler_ Secciones, WF_Recycler_Temas)

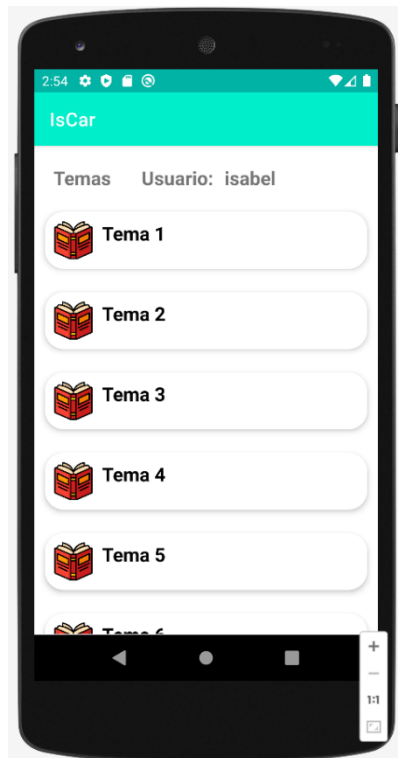


- Wireframes de la rama de ejecución del Temario.
 - ActivityDirectorioSeccionesTemario (WF_Recycler_Seciones).

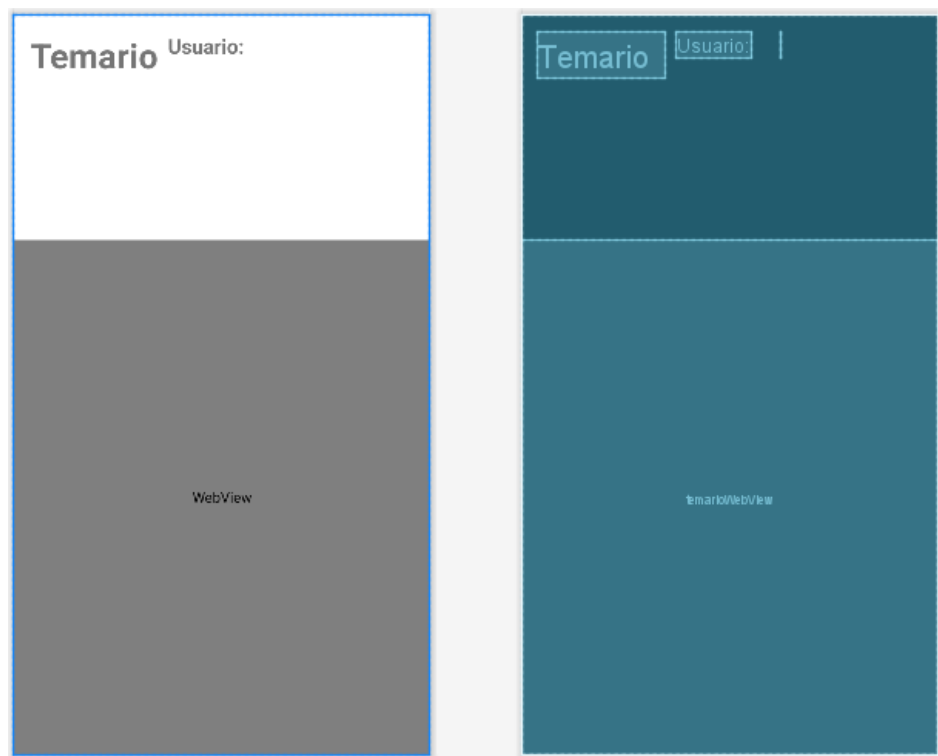


○ ActivityTemasTemario (WF_Recycler_Temas).

○



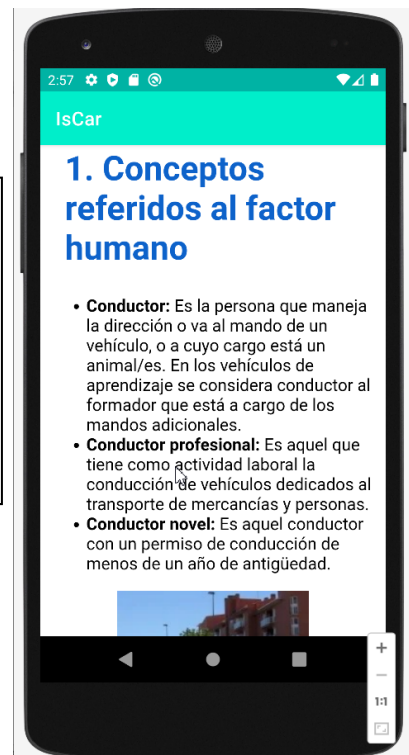
○ ActivityTemarioWeb.



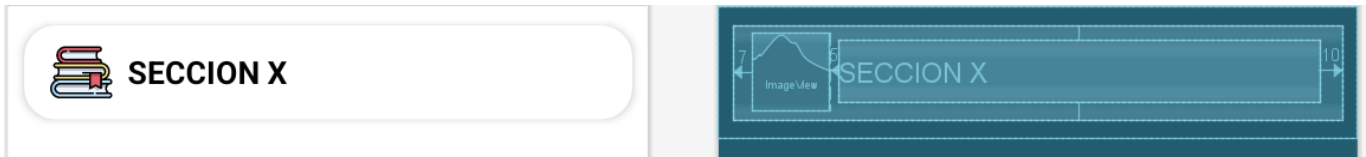
○



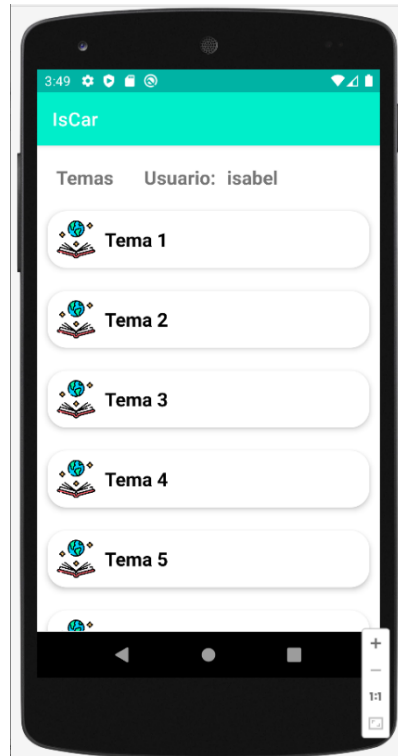
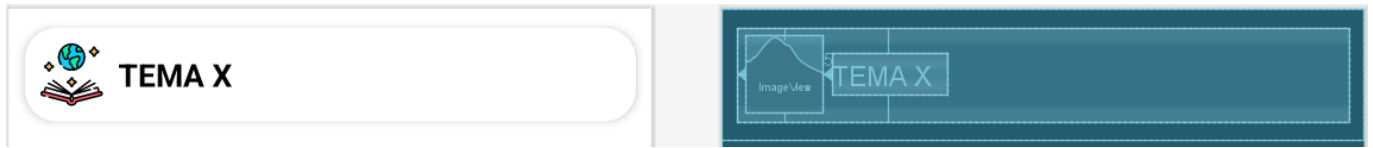
La visualización se realiza a través de la carga de una página html (Compartiré esta información más adelante).
Además los índices son dinámicos al pincharlos nos llevarán al título deseado.



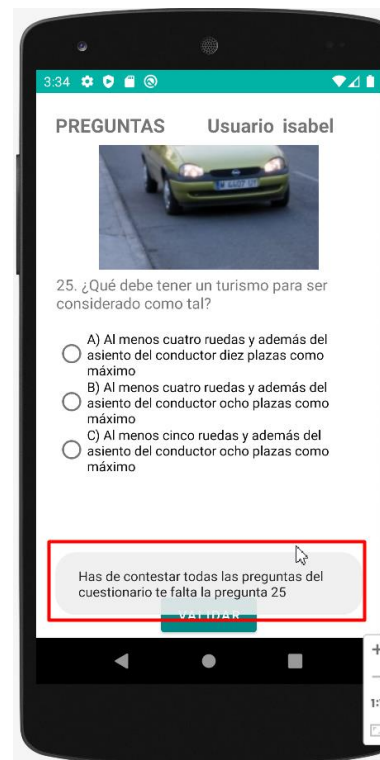
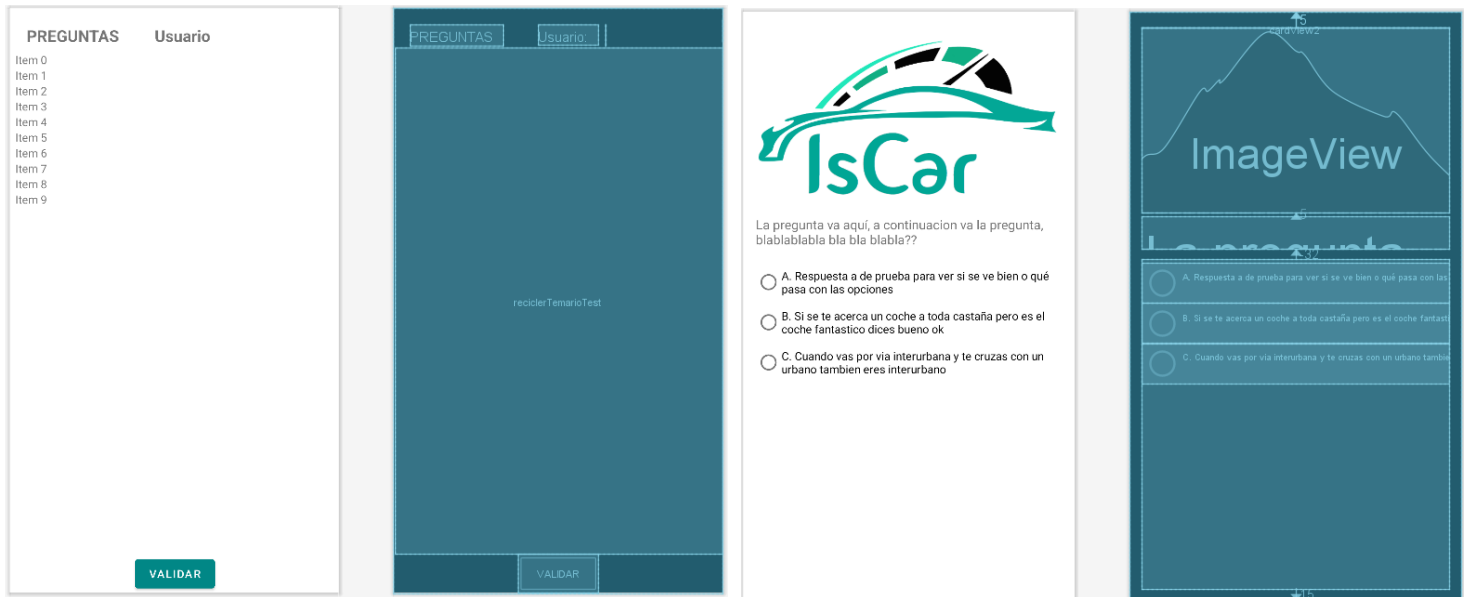
- Wireframes de la rama de ejecución de los Cuestionarios.
 - ActivityDirectorioSeccionesCuestionarios (WF_Recycler_Seciones).



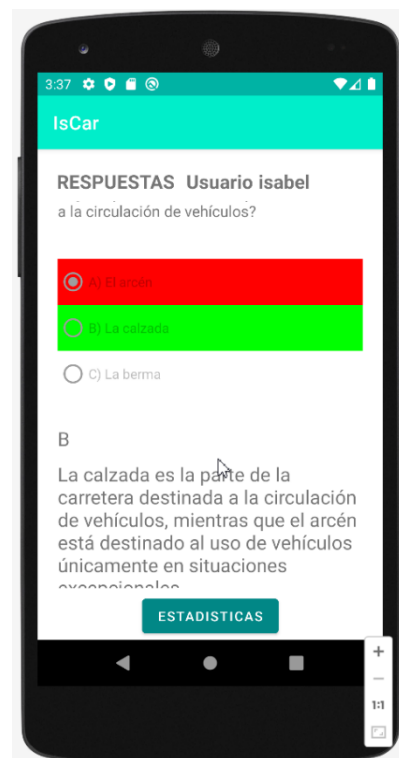
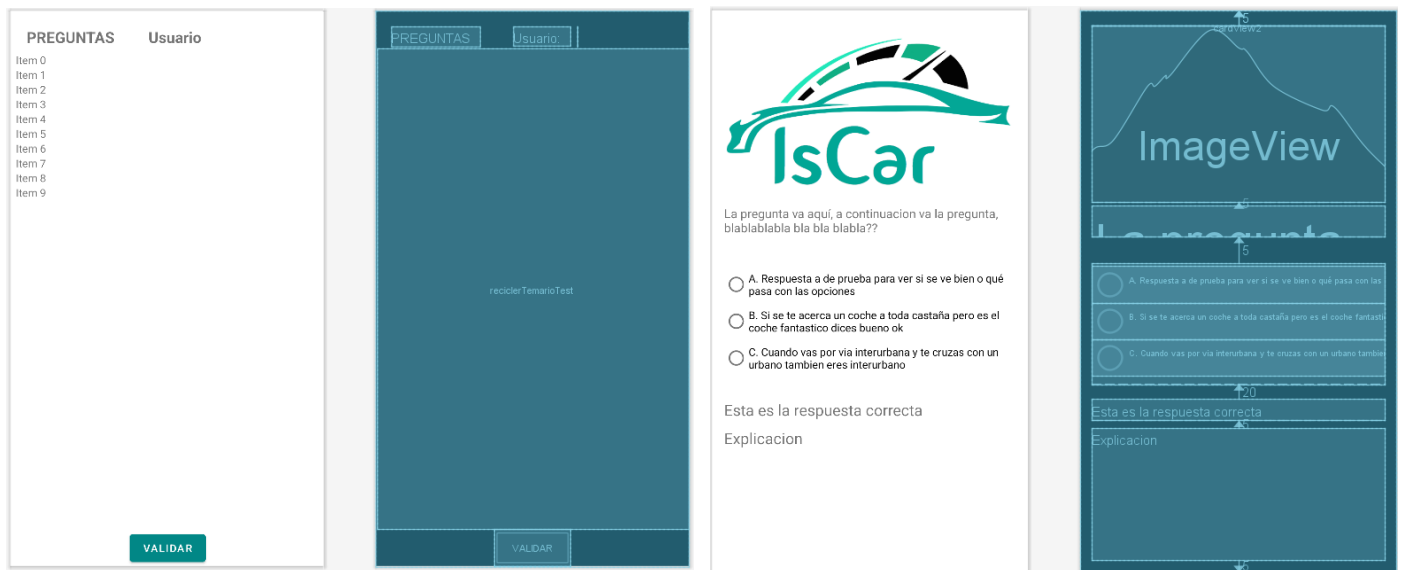
-
- ActivityTemarioTemas_Cuestionario (WF_Recycler_Temas).



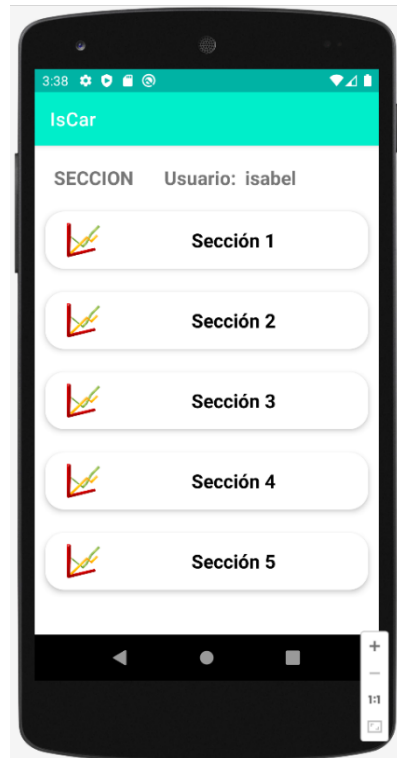
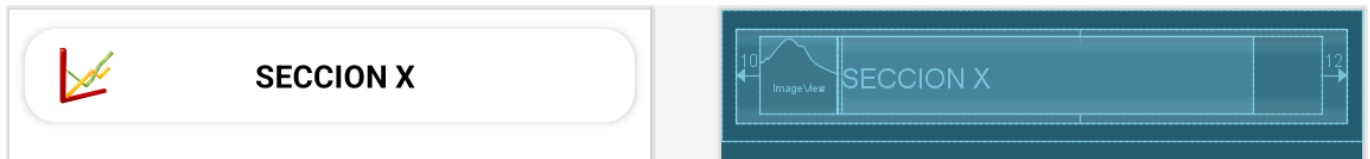
○ Activity_Test:



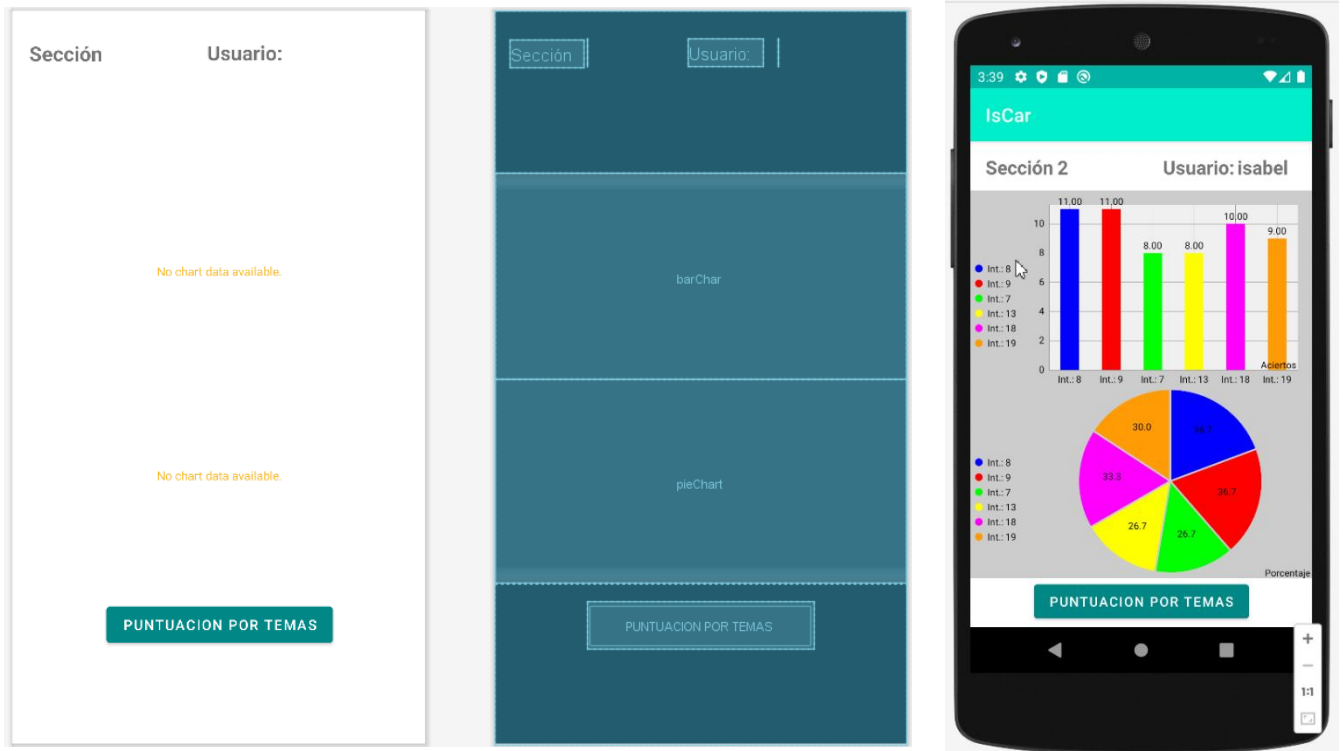
○ Activity_Test_2.



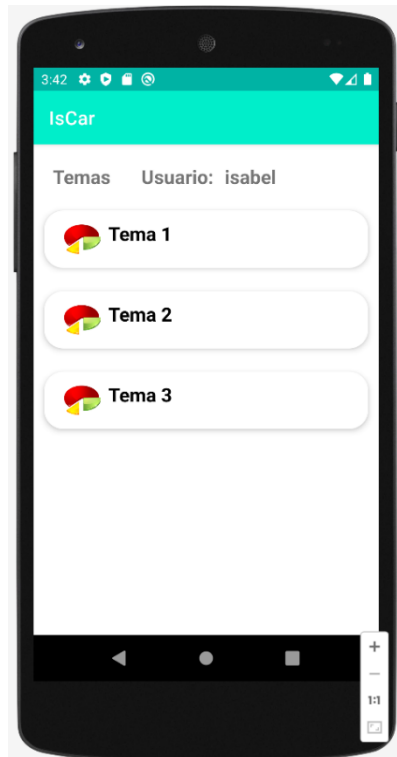
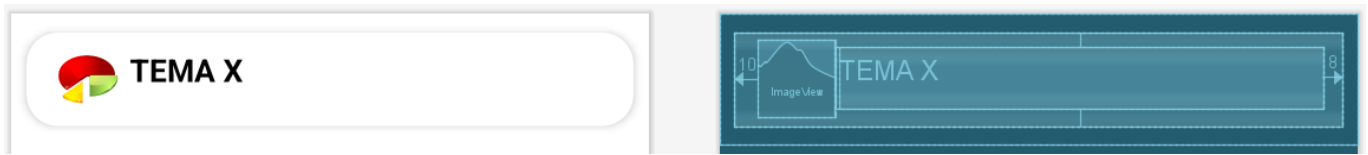
- Wireframes de la rama de ejecución de los Gráficos.
 - Activity_Secciones_Graficos (WF_Recycler_Secciones).



○ Activity_GridLayout_graficos_secciones.

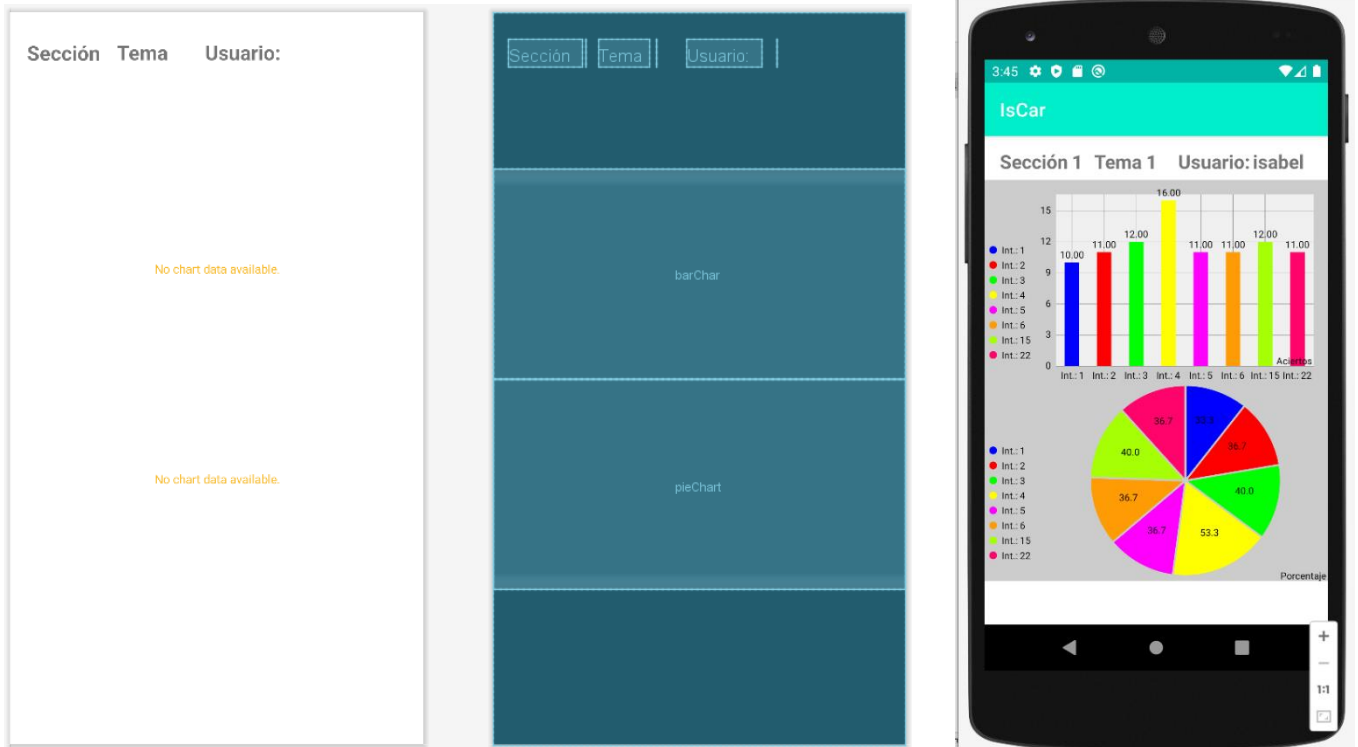


○ Activity_Temas_Graficos (WF_Recycler_Temas).

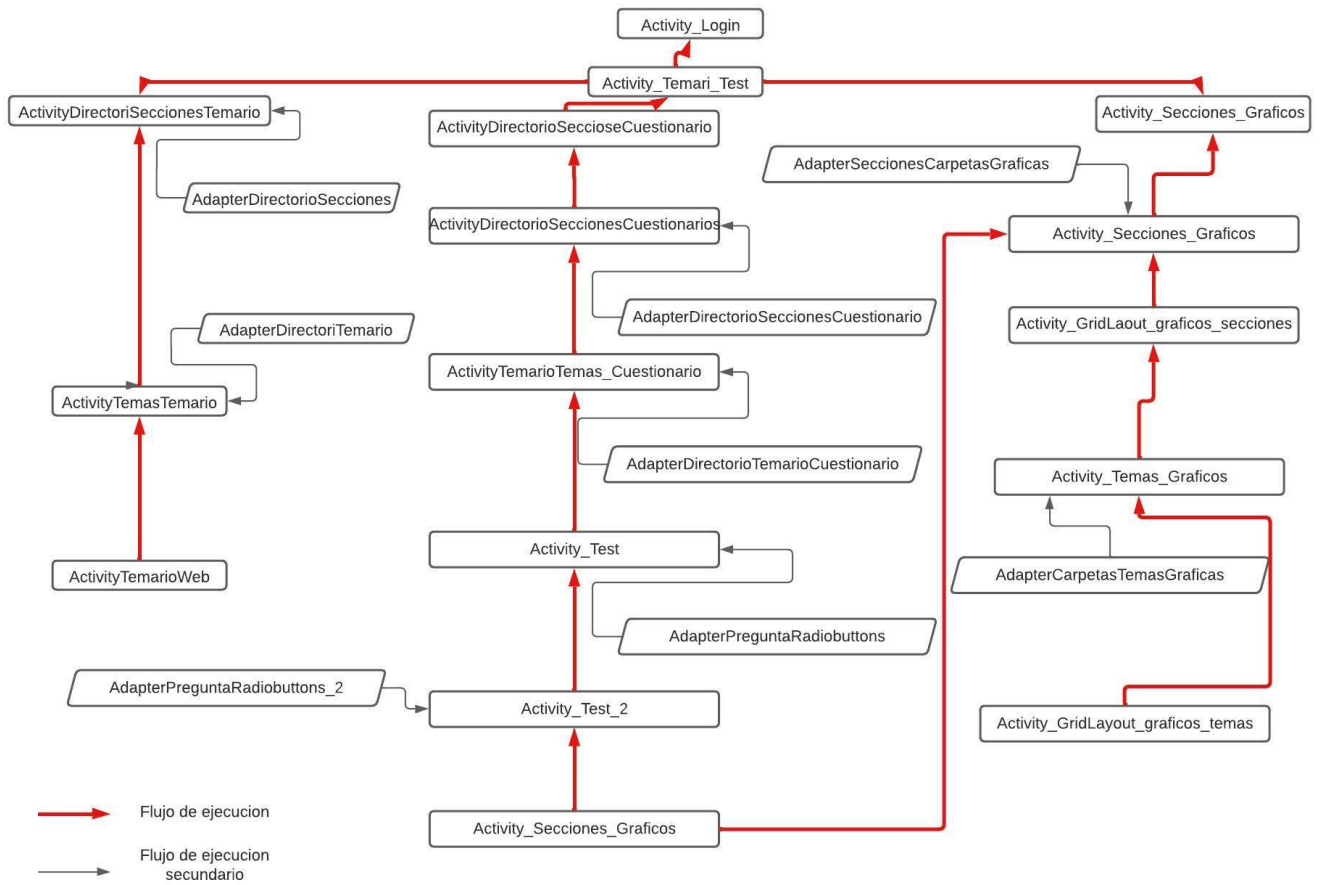


En esta imagen vemos que solo hay temas del 1 al 3, esto lo he programado adrede para muestra solo los datos de los que hay registro, por lo que en este caso los temas del 4 al 8 no tienen Cuestionario realizados.

○ Activity_GridLayout_graficos_temas.



2.8. NAVEGACIÓN.



2.9. PATRONES DE CONSULTA CON VOLLEY.

○ Patrón de consulta de validación.

Este patrón lo utilizo para validar los datos introducidos por el usuario, como por ejemplo el acceso de usuario o comprobar si la creación de una tabla o la inserción de datos en la misma se ha realizado.

Bajo estas líneas en la creación del método incluimos la url de la consulta guardada en el servidor.

```
private void validarUsuario(String URL){
```

Con la siguiente línea creamos el elemento que ejecutara la consulta al servidor

```
    StringRequest = new StringRequest(Request.Method.POST, URL, new
    Response.Listener<String>() {
        @Override
        //Recoge la respuesta de la consulta y podemos decidir qué hacer con ello
        En este caso si la respuesta es 1 abrirá la siguiente activity
        public void onResponse(String response) {
            if (response.equals("1")){
                Intent intent= new
                Intent(getApplicationContext(),Activity_temario_Test.class);
                intent.putExtra("usuario",usu);
                startActivity(intent);
            }else{
                Toast.makeText(Activity_Login.this, "Usuario o contraseña
                erroneos", Toast.LENGTH_LONG).show();
            }
        }
    }, new Response.ErrorListener() {
        //En caso de que ocurra un error
        @Override
        public void onErrorResponse(VolleyError error){
            Toast.makeText(Activity_Login.this, error.toString(),
            Toast.LENGTH_SHORT).show();
            Toast.makeText(Activity_Login.this, "Error de conexion",
            Toast.LENGTH_SHORT).show();
        }
    }) {
        //Elemento en el que guardamos los datos introducidos por el usuario y que
        se pasara por parámetro a la consulta
        @Override
        protected Map<String, String> getParams() throws AuthFailureError {
            Map<String,String> parametros = new HashMap<String,String>();
            parametros.put("usuario",et_nom.getText().toString());
            parametros.put("password",et_pass.getText().toString());
            return parametros;
        }
    };
    //Intancia de la conexion con los datos
    RequestQueue = Volley.newRequestQueue(this);
    requestQueue.add(stringRequest);
```

En cuanto a los archivos .php para las consulta contamos con el archivo “base de todas las consultas en este caso “conexión.php”, en los que tenemos todos los datos de la conexión. Esto podría hacerse de otra manera en la que el usuario tuviera que ingresar el usuario y contraseñas para acceder a la base de datos. En mi caso decidí no hacerlo así, entendiendo que todo usuario tiene acceso a la misma solo desde la aplicación.

```

1 <?php
2 $hostname='localhost';
3 $database='test_autoescuela';
4 $username='root';
5 $password='12345';
6
7 $conexion=new mysqli($hostname,$username,$password,$database);
8 if($conexion->connect_errno){
9     die("El sitio web está experimentado problemas" . mysqli_connect_error()) ;
10 }else{
11     // echo "Conexion realizada con exito <br>";
12 }
13 ?>

```

Todas las demás consultan tendrán el código requer(“conexión.php”), que le dará acceso a la base de datos.

```

1 <?php
2 require( "conexion.php" );
3
4 $usu_usuario = !empty( ( $_POST[ 'usuario' ] ) ) ? $_POST[ 'usuario' ] : NULL;
5 $usu_password = !empty( ( $_POST[ 'password' ] ) ) ? $_POST[ 'password' ] : NULL;
6
7 //$usu_usuario = 'isabel';
8 //$usu_password = 'isa';
9
10 $consulta = "SELECT COUNT(*)
11 FROM usuarios
12 WHERE `usuario` = '$usu_usuario' AND
13     password = '$usu_password'";
14 $resultado=mysqli_query($conexion,$consulta);
15
16 while ($row = mysqli_fetch_row($resultado))
17 {
18     echo $row[0];
19 }
20
21 mysqli_free_result($resultado);
22 ?>

```

- **Línea 4 y 5:** En estas líneas recogeremos los datos introducidos por el usuario convirtiéndolos en variables utilizadas en la consulta sql (Línea 10).
- **Línea 21:** este método es el que se asegura de que la consulta no se sigue ejecutando y se manda de regreso a la aplicación.

○ Patrón de consulta de recuperación de datos.

En este tipo de consulta queremos almacenar los datos devueltos de la consulta como puede ser las preguntas del cuestionario o las puntuaciones de un usuario.

```
public List<RecyclerTest> obtenerPreguntas(String responseBody, String stringTabla) {
    StringRequest = new StringRequest(POST, responseBody, new Response.Listener<String>() {
En este caso en el response creamos un objeto JSONObject que nos ayuda a que la
aplicación interprete el json enviado por la consulta y así crear un objeto java que
permita tratar por separado cada elemento de la consulta (En este caso cada elemento de
las preguntas):
        @Override
        public void onResponse(String response) {
            try {
                JSONObject jsonObject = new JSONObject(response);
                JSONArray result = jsonObject.getJSONArray("preguntas");
                for (int i = 0; i < result.length(); i++) {
                    JSONObject data = result.getJSONObject(i);
                    RecyclerTest recycler = new RecyclerTest();
                    recycler.setImagen(data.optString("foto"));
                    recycler.setId(data.optInt("idPregunta"));
                    recycler.setPregunta(data.optString("enunciado"));
                    recycler.setOpcA(data.optString("opcion1"));
                    recycler.setOpcB(data.optString("opcion2"));
                    recycler.setOpcC(data.optString("opcion3"));
                    recycler.setSolucion(data.optString("solucion"));
                    recycler.setExplicacion(data.optString("explicacion"));
                    arrayListTest.add(recycler);

                }
                for (RecyclerTest : arrayListTest) {
                    arrayListTestRespuestas.add(recyclerTest);
                }
                inflateViewRecycler();
            } catch (JSONException e) {
                Toast.makeText(Activity_Test.this, e.toString(), Toast.LENGTH_SHORT).show();
                Toast.makeText(Activity_Test.this, "Error de conexion",
                Toast.LENGTH_SHORT).show();
                e.printStackTrace();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Toast.makeText(Activity_Test.this, error.toString(), Toast.LENGTH_SHORT).show();
            Toast.makeText(Activity_Test.this, "Error response", Toast.LENGTH_SHORT).show();
            error.printStackTrace();
        }
    }) {
        @Override
        protected Map<String, String> getParams() {
            Map<String, String> params = new HashMap<String, String>();
            params.put("tabla", stringTabla);
            return params;
        }
    };
    //Intancia de la conexion con los datos
    RequestQueue = Volley.newRequestQueue(this);
    requestQueue.add(stringRequest);
    return arrayListTest;
}
```



```

1 <?php
2 require( "conexion.php" );
3 require_once 'RecyclerTest.php';
4 require_once 'Test.php';
5 require_once 'OpcionA.php';
6
7 $tabla = !empty( ( $_POST[ 'tabla' ] ) ) ? $_POST[ 'tabla' ] : NULL;
8
9 $consulta = "SELECT `foto`, `idPregunta`, `enunciado`, `opcion1`, `opcion2`, `opcion3`, `solucion`, `explicacion`, `idTest` FROM $tabla ";
10 $objects = array();
11 $resultado = $conexion->query( $consulta );
12
13 $i = 0;
14 while ( $row = mysqli_fetch_array( $resultado ) ) {
15     $objects[ $i ] = array(
16         "foto" => "data:image/jpeg; base64," . base64_encode($row[ 'foto' ]),
17         "idPregunta" => $row[ 'idPregunta' ],
18         "enunciado" => $row[ 'enunciado' ],
19         "opcion1" => $row[ "opcion1" ],
20         "opcion2" => $row[ "opcion2" ],
21         "opcion3" => $row[ "opcion3" ],
22         "solucion" => $row[ 'solucion' ],
23         "explicacion" => $row[ 'explicacion' ],
24         "idTest" => $row[ 'idTest' ]
25     );
26     $i++;
27 }
28 echo json_encode(array("preguntas" => $objects));
29
30 $resultado->free();
31 $resultado->close();
32 $conexion->close();
33
34 ?>

```

- **Línea 14 a 26:** En estas líneas después de recoger el valor de la tabla a consultar (el string se crea a nivel java según la sección y tema seleccionados por el usuario) crea un iterador (elemento que pasa por cada una de las digamos “líneas” devueltas por la consulta) en el que se crea un elemento que contenga todos los puntos de la consulta separándolos por las columnas de la tabla.
- **Línea 28:** Esta línea convierte la información guardada en el array y lo convierte en un elemento json indicando que el nombre del conjunto de datos es preguntas y le pasa el objeto que contiene los datos.

○ Consultas PHP.

- **conexion.php:** Consulta por la que pasamos la base de datos, usuario y contraseña que se usaran para hacer las consultas.
- **Validar_usuario.php:** Consulta para ver si existe un usuario a través de su usuario y contraseña.
- **devuelve_array_preguntas.php:** consulta que devuelve las preguntas de una tabla en concreto que es pasada como parámetro, lo que hace que la consulta sea reutilizable para todas las tablas.
- **existe_tabla.php:** Consulta que comprueba si una tabla ya existe en la base de datos y que su creación no produzca un error en la aplicación.
- **crea_tabla_cuestionarios_puntuacion.php:** Consulta que crea una tabla de puntuaciones para un usuario concreto (no incluye comprobación de si existe la tabla, este paso se hace con java a través de la aplicación.)
- **insertar_puntuacion_cuestionarios.php:** Consulta a la que pasamos el usuario, sección, tema y preguntas correctas para que inserte los puntos en la tabla personal del usuario creada don crea_tabla_cuestionarios_puntuacion.php.

- **listar_secciones.php:** Consulta para consultar las secciones existentes en nuestro caso en la tabla de puntuación del usuario, así solo se mostrarán en el RecyclerView las secciones de las que se haya realizado algún cuestionario.
- **obtenerTemasPorSeccion.php:** Consulta para consultar los temas existentes en nuestro caso en la tabla de puntuación del usuario, así solo se mostrarán en el RecyclerView los temas de las que se haya realizado algún cuestionario.
- **obtenerPuntuacionPorSeccion.php:** Consulta que nos devuelve los datos de puntuación de un usuario concreto de los cuestionarios realizados por sección.
- **obtenerPuntuacionPorTema.php:** Consulta que nos devuelve los datos de puntuación de un usuario concreto de los cuestionarios realizados por tema.

2.10. CREACIÓN DE GRÁFICAS CON MPANDROIDCHART.

○ Conceptos básicos.

Para trabajar con graficas de barras los conceptos básicos a tener en cuenta son:

- **AxixY:** Este es el eje vertical que se suele mostrar a la izquierda en el que se suele indicar los valores representados en nuestro caso el número de aciertos.
- **AxixX:** En este eje se representarán los diferentes valores en nuestro casos los distintos intentos de cuestionarios.
- **Leyenda:** Son la representación gráfica de los elementos del AxisX, nos muestra que indica esa barra o porción del gráfico.

En nuestro caso estos elementos los pasamos como un array list por cada uno de los elementos. Además también tenemos el número máximo de aciertos que se usara como valor máximo del AxixY.

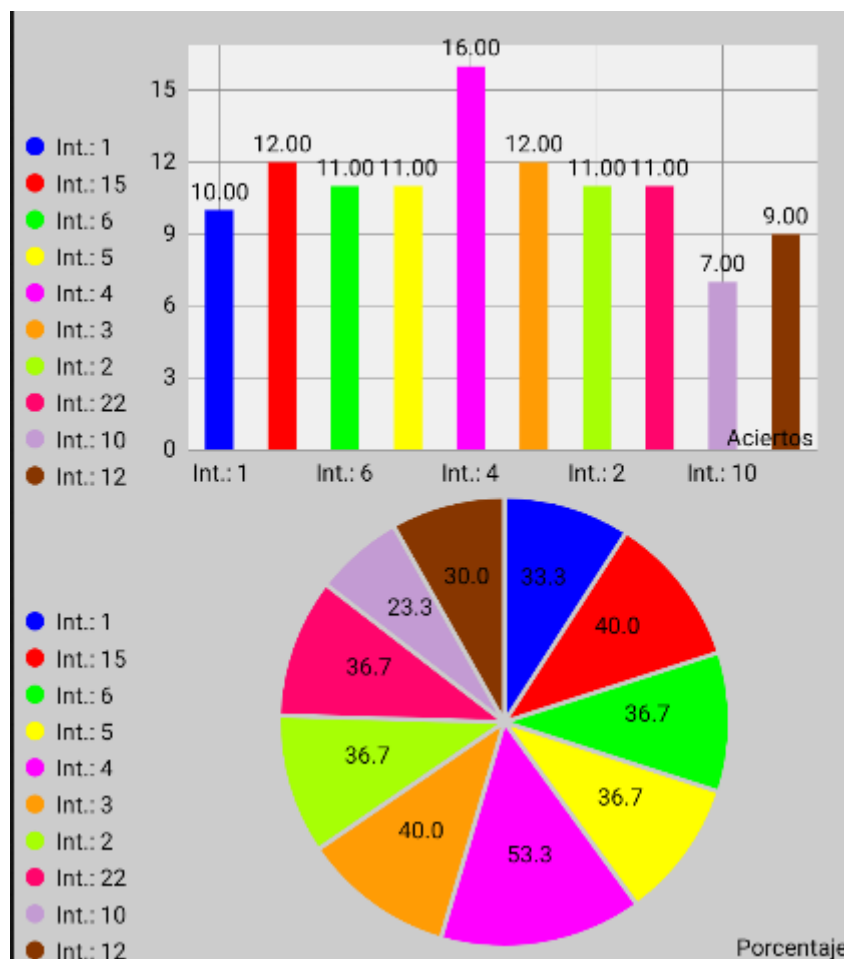
○ Representación del proceso de configuración de los gráficos:

- **getSameChart:** método con el que creamos los elementos BarChart y PieChart, en él se le pasa el propio elemento a crear que enlaza con el de la vista de Android, la leyenda de lo que representa, el color de la fuente, el color de fondo y el tiempo de la animación.

```
barChar = (BarChart) getSameChart(barChar, descripcion: "Aciertos", Color.BLACK, Color.LTGRAY, animateY: 3000);
```

- **setDrawGridBackground:** Este elemento indicado como true nos muestra las líneas que marcan los diferentes puntajes del AxixY en este caso la escala de aciertos.
- **setData(getBarData()):** Método por el cual cargamos la información en el gráfico.

- **getBarData():** Método que unifica los datos de los tipos nombrados anteriormente a través del método BarDataSet con el método getBarEntries(). En este método se establece el fondo del propio gráfico y el tamaño de las letras.
- **getBarEntries():** Método en el que cargamos los datos con los números de aciertos.
- **Invalidate():** este método indica que hemos terminado de configurar el Chart.
- **axisX:** Este método implementa donde se posicionará el eje x y se le direccionara al array que contiene las leyendas.
- **axisLeft:** con este método introducimos el alto en este caso del AxisY.
- **axisRight:** este método configura el AxisY de la derecha en nuestro caso queremos que no se represente nada.
- **setTransparentcorcñeRadois: / setDrawHoleEnable:** Con estos Métodos se configura el círculo interior de los “quesitos” del grafico tipo Pie o tarta , en nuestro caso al disponer de poco espacio lo configuramos pequeño y sin reborde.



3. CONCLUSIONES.

Los objetivos que debían de ser cumplidos son:

1. **Crear una base de datos de fácil utilización**
2. **Crear una aplicación de navegación sencilla.**
3. **Gestión de usuarios**
4. **Hacer listados más dinámicos y visualmente más personalizados.**
5. **Cuestionarios fáciles de rellenar.**
6. **Gestión correcta de resultados.**
7. **Visualización de los resultados en gráficos.**

La numeración me servirá para enlazar el objetivo en las tres secciones siguientes.

3.1. OBJETIVOS CUMPLIDOS.

2. **Crear una aplicación de navegación sencilla:**

La navegación de la aplicación es muy visual y lo suficiente sencilla para navegar entre los diferentes activitys. He utilizado elementos simples de android sin añadir menús laterales ni elementos ocultos que puedan llevar a confusión. Lo hice así para que en el supuesto de que la aplicación se ha utilizada por personas mayores o con capacidades limitadas sea más accesibles.

6. **Gestión correcta de resultados:**

En este apartado estoy conforme con mis resultados, ya que creo que los datos se graban y recuperan de forma limpia y dinámica, no se crean tablas que no sean necesarias y se muestra la información necesaria y deseada. Además su tratamiento es fácil.

3.2. OBJETIVOS CUMPLIDOS PARCIALMENTE.

1. **Crear una base de datos de fácil utilización:**

La base de datos creada es fácil de utilizar pero no es una base de datos optimizada, ya que para su creación lo hice a través de múltiples tablas cuando hay formas más dinámicas y fáciles de hacer que habrían requerido de la utilización de menos recursos para hacer lo mismo.

3. **Gestión de usuarios:**

La aplicación empieza pidiendo un nombre de usuario y contraseña que serán necesarios para seguir con la utilización de la aplicación. Es completamente funcional y tiene su debido sistema de comprobación.

En este caso pongo que se ha cumplido parcialmente, porque me faltó añadir una pantalla por la cual los usuarios puedan registrarse en esa base de datos. En el apartado de propuestas de mejora desarrollare más a fondo este tema.

7. Visualización de los resultados en gráficos:

Si bien mis resultados se muestran correctamente en los gráficos correspondientes, tengo que mejorar la dimensión de los gráficos, así como la visualización de leyendas y como se visualizaría una gran cantidad de datos.

3.3. OBJETIVO NO CUMPLIDOS.**4. Hacer listados más dinámicos y visualmente más personalizados:**

En ese punto mi intención no era utilizar los RecyclerView para listar las secciones y temas disponibles. Tenía pensado utilizar una vista GridView.

Cuando empecé a programarlo me encontré con el problema de que en según qué cosas necesitaba que se pasara más información de la que permite GridView, como el estado del elemento. (que se marque como nuevo si no se ha entrado todavía).

Intente implementar un estado que se marcara como nuevo si el usuario no había pasado por la sección correspondiente, en la implementación de este punto vi que no sabía muy bien como empezar y fui aplazando su implementación hasta el final y finalmente por falta de tiempo no lo implante (en la sección de Propuestas de mejora indicare más sobre este tema).

5. Cuestionarios fáciles de rellenar:

En este punto si bien la aplicación funciona no pude solucionar el incorrecto funcionamiento de los RadioButtons, que al ir pasando de elementos si te pasas un poco más de la cuenta del elemento actual el RadioButton marcado será el de el siguiente elemento. El problema quedo medianamente resulte refrescando la página en cada cambio mostrando como marcados los elementos que así lo estén en la lista que guarda dicha información en ejecución, por lo cual solo se mostraran marcados los elementos cullo dato así lo esté permitiendo que no se quede ninguno sin marcar (en la sección de Propuestas de mejora indicare más sobre este tema).

3.4. PROPUESTAS DE MEJORA.**1. Crear una base de datos de fácil utilización.**

Mi propuesta de mejora para la base de datos sería crear una única tabla en la que se guardara toda la información y que al cargar las preguntas se consultara la tabla por sección y tema y que devolviera aleatoriamente 30 preguntas de entre todas las coincidentes. Esto haría que los test fueran más aleatorios y no habría que crear tantas tablas.

2. Crear una aplicación de navegación sencilla.

Hacer un estudio para que la navegación por la aplicación fuera más optima sin la necesidad de complicarla visualmente.

3. Gestión de usuarios:

Añadir la funcionalidad de crear usuarios, estimo que en un día o dos más de trabajo este punto se podría implementar sin más contratiempos, dado que la tecnología e implementación son similares a otras que ya tenemos y son fáciles de reutilizar.

4. Hacer listados más dinámicos y visualmente más personalizados.

Se realizaría la implementación del elemento nuevo, que marcaría si el usuario ya ha visto dicha sección. No he podido ver mucho este apartado pero considero que con un par de días sería suficiente implementar y probar este recurso.

5. Cuestionarios fáciles de rellenar.

El problema generado por los RadioButtons podría solucionarse con la implementación de una Interfaz que ayudara a conectar mejor el elemento del test con la activity en la que se carga y así poder eliminar este inconveniente. Considero que con mis conocimientos actuales necesitaría al menos 4 días para estudiar y arreglar este problema.

6. Gestión correcta de resultados.

En este punto tiene mucho que ver el anterior, ya que no hay una buena comunicación entre el elemento del RecyclerView y la Activity tratar que respuesta ha sido marcada actualmente se hace recogiendo “la lista” de una clase a otra (duplicando la información en ejecución). Solucionar esto con la Interfaz ahorraría recursos al sistema.

7. Visualización de los resultados en gráficos.

Aunque la librería MPAndroidChart, cumple su función, es poco configurable y sus gráficos son muy simples.

4. WEBGRAFÍA.

- <https://google.github.io/volley/>
- <https://cdsphp.wordpress.com/cliente-servidor/#:~:text=La%20arquitectura%20inform%C3%A1tica%20de%20Cliente,que%20por%20otro%20lado%20est%C3%A1n>