

Lucrarea 5 Aplicații Android

5.1 Introducere

Android este o platformă software care permite dezvoltarea aplicațiilor mobile pe smartphone-uri echipate cu versiune modificată a sistemului de operare Linux.

Tehnologia a fost propusă inițial de Google și ulterior dezvoltată de Open Handset Alliance, o corporație formată din actori cheie pe piața IT&C, care are drept obiectiv principal promovarea inovării în domeniul aplicațiilor mobile prin folosirea de tehnologii open-source [1]. Pentru dezvoltarea aplicațiilor Android, un mediu dedicat de programare (SDK- Software Development Kit) Android (Android SDK) [2] este utilizat uzual în conjuncție cu un mediu integrat de dezvoltare integrat (IDE- Integrated Development Environment). Exemple de IDE-uri : Eclipse, NetBeans, Android Studio.

Limbajul utilizat uzual pentru programarea pe platforma Android este Java (există posibilitatea de a utiliza însă și alte limbaje cum ar fi Kotlin sau C++) iar mașina virtuală Java poartă numele de ART (Android Runtime) (utilizată de versiuni Android mai noi de 5.0) sau Dalvik (versiuni mai vechi).

Aplicațiile dezvoltate pentru mașina virtuală Dalvik sunt compatibile cu cele dezvoltate pentru ART. Versiunea curentă de Android este Android 9 Pie.

5.2 Structura unei aplicații Android

Aplicațiile Google Android sunt dezvoltate utilizând programarea orientată pe obiecte (OOP). Orice aplicație include una sau mai multe componente de următoarele tipuri:

- **Activity** – clasă asociată cu o fereastră/interfață grafică care poate fi populată cu diferite tipuri de alte componente de tip *UI (User Interface)* cum ar fi meniuri, liste, casete text, spinnere etc. Utilizarea de ferestre multiple necesită mai multe instanțe ale acestei clase, o aplicație Android fiind uzual formată din una sau mai multe componente de tip Activity.
- **Service** – componentă care include secvențe de cod care rulează în background (posibil în alte thread-uri) și nu oferă o interfață grafică.
- **Content provider** – componentă Android care permite transferul și partajarea de date între diverse aplicații. Datele pot fi stocate în baze de date (tipic SQLite), citite de pe Internet sau stocate în format sistem de fișiere.
- **Broadcast receiver** – componentă ce permite interceptarea evenimentelor sistem și a mecanismelor de tip **Intent** ce permit transferul asincron de mesaje între elemente de tip Activity și/sau Service, sau instanțierea a altor obiecte de tip Activity. Toate clasele utilizate de o aplicație și proprietățile lor asociate trebuie să fie declarate în format XML într-un fișier numit **AndroidManifest.xml** inclus în proiect.

5.3 Dezvoltarea de aplicații Android folosind mediul de dezvoltare Android Studio

Conform celor menționate mai sus , medii de dezvoltare de tip IDE sunt de obicei utilizate în conjuncție cu un SDK Android. Secțiunea prezintă doar modalitatea de utilizare a mediului Android Studio pentru dezvoltarea de aplicații Android. Fig.1 ilustreaza etapele necesare pentru configurarea unei astfel de aplicații folosind versiunea de Android 5.0 ('Lollipop') care este instalată pe toate calculatoarele din laborator. Structura aplicației create automat este cea ilustrată în Fig.2.

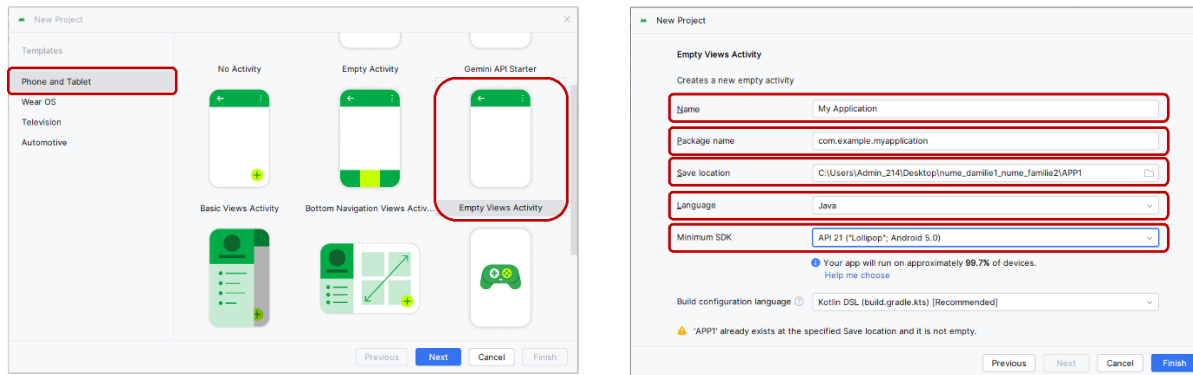


Fig. 1 Setări aferente pentru crearea unei aplicații Android

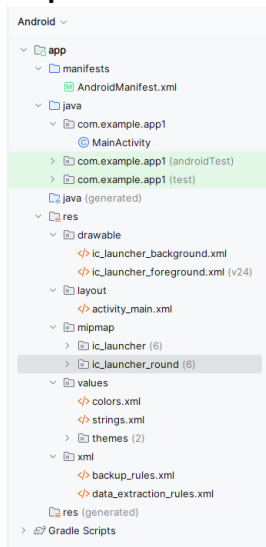


Fig. 2 Structura unei aplicații Android

Orice proiect Android trebuie să includă un fișier cu numele **AndroidManifest.xml** plasat în directorul rădăcină aferent proiectului. Pentru aplicația anterioară structura acesteia este:

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
```

```

        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.APP1"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

Fig. 3 Fișierul *AndroidManifest.xml* pentru aplicația creată

Acest fișier este o parte integrală a aplicației Android, deoarece definește structura și metadatele aplicației, componentele și limitările acestora. Acest fișier include noduri pentru fiecare dintre activitățile ("activity") ce compun orice aplicație, alături de intenție și permisiunile ce determină modul în care elementele sunt conectate unele cu celelalte. Aceste proprietăți ale aplicației sunt specificate în format eXtensible Markup Language (XML).

Fișierul ***MainActivity.java*** include codul Java propriu-zis; pentru aplicația creată acesta are conținutul:

```

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        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;
            });
    }
}

```

Fig. 4 Fișierul *MainActivity.java*

Cuvântul cheie ***extends***, similar aplicațiilor JME, JSE sau JME, este utilizat pentru a implementa mecanisme de moștenire Java (preluarea structurii și a comportamentului unei alte clase cu adăugarea de elemente noi, specifice).

Cuvântul cheie ***@Override*** permite supradefinirea metodelor clasei părinte. Pentru exemplul din Fig. 4, metoda ***OnCreate*** este prima metodă apelată la lansarea aplicației

(echivalent JME StartApp()). Obiectele de tip **Bundle** sunt folosite în Android pentru gestiunea situațiilor în care aplicațiile sunt eliberate din memorie.

Fișierul **activity_main.xml**, plasat în directorul **“res/layout”**, ilustrat în Fig.2, permite definirea interfeței cu utilizatorul în format XML sau, pentru anumite IDE-uri, și în format grafic.

Pentru aplicația creată anterior acesta are structura următoare:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Fig. 5 Fișierul **activity_main.xml**

Fișierul XML poate fi modificat fără a fi necesară o recompilare a codului sursă; acest mod de concepere al unei aplicații este asimilabil modelului arhitectural MVC (Model View Controller) de dezvoltare a aplicațiilor software. Acesta divide o aplicație în trei componente Model, View, Controller; componenta Model gestionează datele aplicației, componenta View permite afișarea modelului și interacțiunea cu utilizatorul, iar componenta Controller interceptează acțiunile utilizatorului inițiate prin intermediul (tipic) touchscreen-ului.

Elementele UI care pot fi utilizate într-o aplicație Android de tip Activity sunt definite complet în [3], printre acestea fiind:

- TextView – afișarea de informații de tip text;
- EditText – editarea informației de tip text;
- Button – elemente UI de tip command-button pentru interacțiunea cu utilizatorul.

După compilarea aplicației elementele definite în fișierul **xml** sunt reprezentate prin componente individuale Android de tip **View** ce ocupă o anumită zonă de pe ecran. Apelarea metodei **onCreate** (Fig. 4) cu un parametru de tip **R.layout.layout_file_name**, permite afișarea acestora:

setContentView (R.layout.file_name);

Toate obiectele Android de tip **View** pot fi referențiate în mod unic dacă li se asociază identificatori sub formă de numere întregi. Următoarea secvență **xml** permite asocierea de identificatori unici elementelor de tip UI:

android: id="@+id/elem_UI;

Subșirul @+id indică compilatorului să creeze o nouă resursă de tip **View**; pentru exemplul de mai sus numele acestuia este **elem_UI**. Toate aceste componente de tip **xml** declarate sunt incluse într-un fișier cu numele **R.java** generat automat în etapa de compilare. Acest fișier permite mai apoi asocierea resurselor grafice codului Java. Elementele UI astfel declarate pot fi referite în cod prin folosirea de instrucțiuni de tipul (Fig. 4):

Class object_UI = (Class) findViewById(R.id.elem_UI);

Elementele de tip UI asociate tastelor programabile (meniuri) sau obiecte Android de tip **Spinner** pot fi declarate de asemenea în fișiere **xml**. În cazul definirii unui meniu de tip **popup** fișierul **xml** trebuie să fie distinct. Aplicațiile pot fi testate folosind emulatoare definite în prealabil, cu diferite caracteristici (rezoluție, memorie etc). În cazul aplicației create anterior, rezultate posibile sunt afișate în Fig. 6.

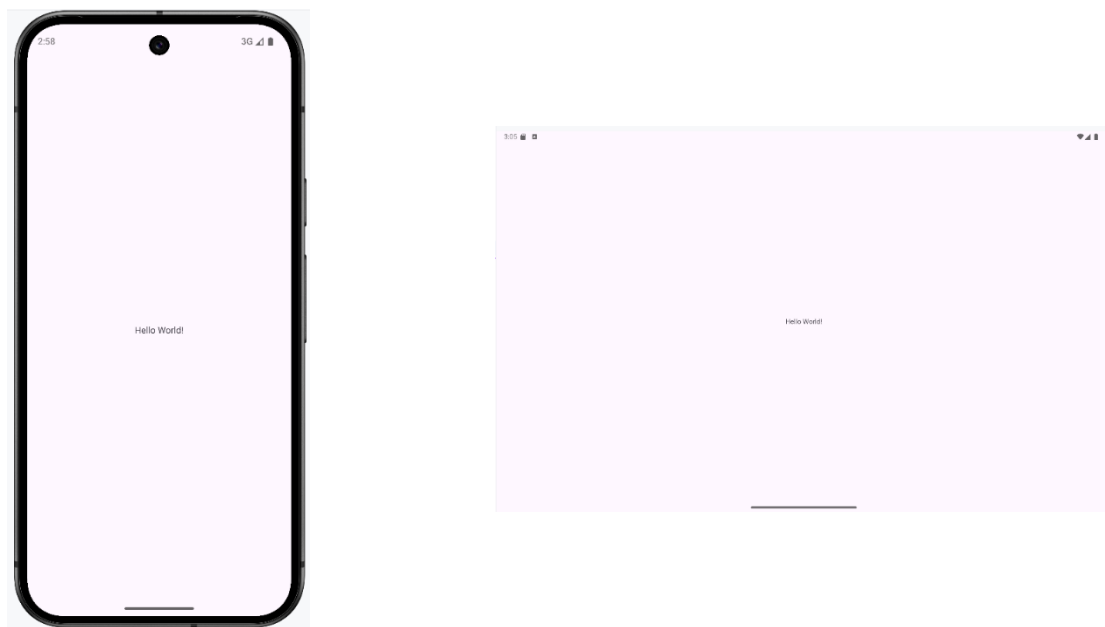


Fig. 6 Emulatoare Android (AVD) – rezultate ale rulării aplicației de tip Hello world

Un dispozitiv virtual Android (AVD) este o configurație care definește caracteristicile unui telefon Android, tabletă, dispozitiv Wear OS, Android TV sau dispozitiv Automotive OS pe care dorim să simulăm în emulatorul Android. Managerul de dispozitiv este un instrument pe care îl putem lansa din Android Studio, ilustrat în Fig. 8.



Fig 7. Manager dispozitiv virtual (AVD)

5.4 Exerciții

- 1) Repetați pașii indicați mai sus pentru a crea o aplicație Android folosind mediul de dezvoltare Android Studio.
- 2) Folosind referința bibliografică [3] inserați comentarii în codul Java și xml (**MainActivity.java** și **activity_main.xml**) ilustrând rolul fiecărei instrucțiuni.
- 3) Modificați aplicația astfel încât textul "Hello World" să fie înlocuit de "Lab 5: Android Applications".
- 4) Testați aplicația folosind mai multe emulatoare Android (Android Virtual Device).
- 5) Creați un fișier de tip .doc în care să inserați capturi de ecran care să ilustreze funcționarea aplicației pentru textul definit la pct. 3).

5.5 Aplicație client-server folosind Android (client) și tehnologia Microsoft IIS (server)

5.5.1 Arhitectura aplicației

Arhitectura aplicației este cea indicată în Fig.8

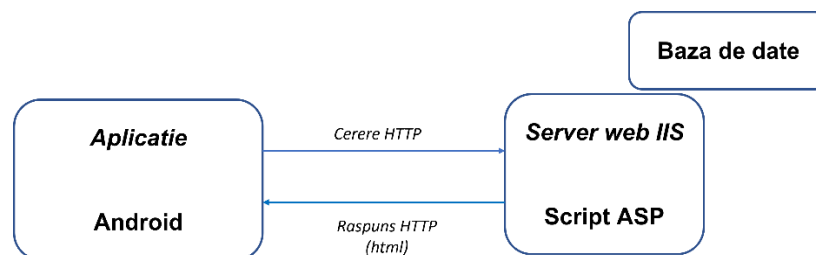


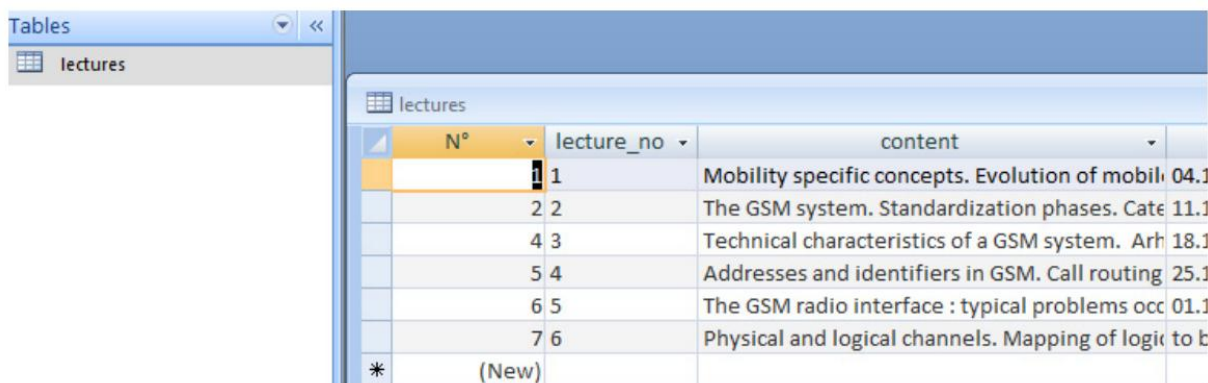
Fig 8. Arhitectura aplicației

Aplicația client Android permite setarea parametrilor de configurare **HTTP** pentru selectarea datelor ce urmează să fie vizualizate pe un dispozitiv virtual, dintr-o bază de date stocată pe un server, prin interogarea acesteia de la distanță. Dialogul dintre aplicația-client și

serverul web se realizeaza folosind protocolul **HTTP** prin intermediul unei solicitari de tip **GET** pentru plasarea unei interogări interpretată de un fișier **ASP**. Răspunsul trimis de server va fi afișat de echipamentul virtual (AVD). Scenariul pentru care a fost concepută aplicația permite accesul la conținutul cursului de **Comunicații mobile** (Fig. 9) care este stocat într-o bază de date **wapdb.mdb**, plasată pe un server web IIS.

5.5.2 Structura și conținutul bazei de date

Informațiile stocate sunt indicate în Fig. 9



N°	lecture_no	content
1	1	Mobility specific concepts. Evolution of mobile 04.1
2	2	The GSM system. Standardization phases. Cate 11.1
4	3	Technical characteristics of a GSM system. Arh 18.1
5	4	Addresses and identifiers in GSM. Call routing 25.1
6	5	The GSM radio interface : typical problems occ 01.1
7	6	Physical and logical channels. Mapping of logic to b
*	(New)	

Fig. 9 Structura bazei de date/informațiile stocate

5.5.3 Aplicația client

1. Creați o aplicație Android procedând într-un mod similar cu cel indicat în secțiunea 5.3
2. Modificați fișierul **activity_main.xml** după cum urmează:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/widget29"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:textColor="#ffffff"
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <TextView android:id="@+id/text"
        android:textStyle="bold"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#ffff00"
        android:textSize="32sp"
        android:text="Select URL" >
    </TextView>
    <EditText
        android:id="@+id/setari"
        android:layout_width="fill_parent"
        android:layout_height="60dip"
        android:capitalize="sentences" >
```

```

</EditText>
<Button
    android:id="@+id/ok"
    android:layout_width="fill_parent"
    android:layout_height="60dip"
    android:text="OK" />
</LinearLayout>

```

Elementele UI folosite pentru interacțiunea cu utilizatorul sunt:

- o componentă de tip `TextView` pentru afișarea unui text prestabilit (ID Android text)
- o componentă de tip `EditText` care permite afișarea unei căsuțe de dialog pentru setarea adresei IP a serverului (ID Android settings)
- un buton de comandă (ID Android OK)

Modul de afișare a elementelor UI corespunde unui dispuneri liniare, având ca origine colțul din stânga sus al telefonului ecranului. Pentru toate elementele UI, lățimea a fost setată la dimensiunea ecranului (atributul `FILL_PARENT` pentru lățime) iar înălțimea lor este fie variabilă (cuvânt cheie `WRAP_CONTENT`) fie setată la o valoare implicită.

3) Aplicația va necesita acces **Internet**, aceasta presupune modificarea fișierului ***AndroidManifest.xml*** după cum urmează:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:usesCleartextTraffic="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.AppCompat"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

4) Selectarea informației care va fi afișate se efectuează prin utilizarea unei tastei programabile. Acestea îi va corespunde un fișier **my_menu.xml**. Creați mai întâi un subdirector cu numele **menu** în directorul **res**. Etapele necesare creării fișierului my_menu.xml sunt ilustrate în Fig. 11.

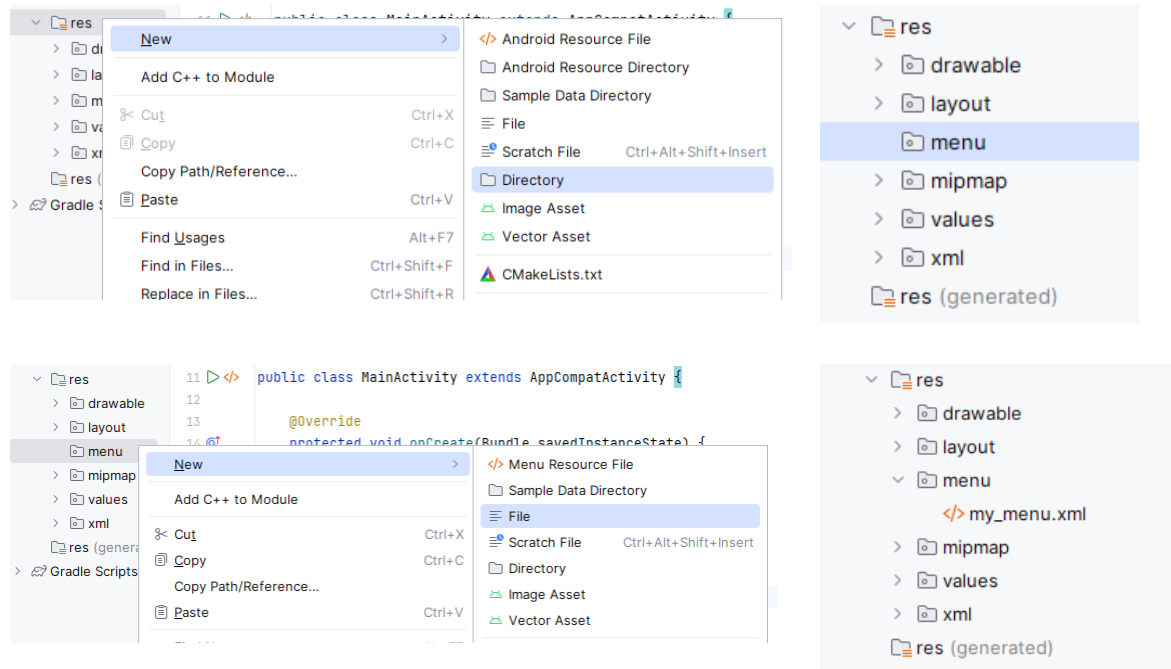


Fig. 10 Crearea fișierului **my_menu.xml**

Conținutul fișierului **my_menu.xml** trebuie să conțină următoarele:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/i1" android:title="Lecture 1"></item>
    <item android:id="@+id/i2" android:title="Lecture 2"></item>
    <item android:id="@+id/i3" android:title="Lecture 3"></item>
    <item android:id="@+id/i4" android:title="Lecture 4"></item>
    <item android:id="@+id/i5" android:title="Lecture 5"></item>
</menu>
```

5) Afișarea meniului de tip pop-up presupune utilizarea următoarei secvențe de cod în **MainActivity.java**:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        this.setTitle("Course content ...");
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater=getMenuInflater();
```

```
inflater.inflate(R.menu.my_menu, menu);  
return true;  
}  
}
```

Observație: pentru utilizarea în aplicație de obiecte de tip Menu următoarele clase trebuie importate în proiect:

```
import android.view.Menu;  
import android.view.MenuInflater;  
import android.view.MenuItem;
```

5) În cazul în care toate elementele anterioare au fost setate corect, rezultatul este cel din Fig. 11.

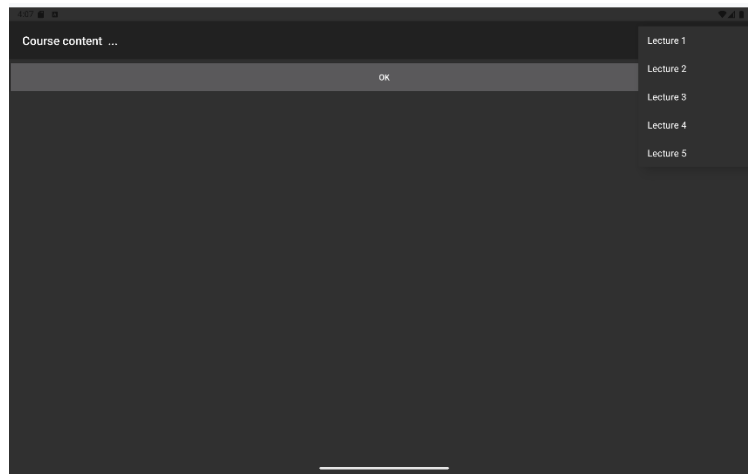


Fig. 11 Intefată grafică (UI) și tastă programabilă configurată

Cod Java

Aplicația utilizează o componentă Android de tip **Activity**. Mai multe variabile și metode sunt necesare pentru a face aplicația pe deplin funcțională.

Conținutul fișierului **MainActivity.java** este cel indicat în anexa 1.

5.6 Exerciții

- 1) Modificați conținutul fișierului **MainActivity.java** cu conținutul inclus în **anexa 1**.
- 2) Rulați aplicația și observați rezultatele obținute
- 3) Insezați comentarii în cod care explică rolul fiecărei metode.
- 4) Creați un fișier de tip **.doc**; insezați capturi de ecran care să ilustreze funcționarea aplicației pentru diverse cursuri selectate (AVD).
- 5) Creați o nouă aplicație. Folosiți clase de tip **WebView** și **Spinner** care să permită obținerea unui rezultat similar cu cel obținut prin aplicația propusă în secțiunea 5.5.3. Folosiți documentația online Android pentru a comenta codul.


```

        }
    });
}

@Override
public boolean onCreateOptionsMenu(Menu menu){ MenuInflater
    inflater=getMenuInflater(); inflater.inflate(R.menu.my_menu, menu);
    return true;
}

private String OpenHTTPConnection(String urlString) throws IOException
{
    int response = -1;
    String result="";
    URL url = new URL(urlString);
    URLConnection conn = url.openConnection();
    try{
        HttpURLConnection httpConn = (HttpURLConnection) conn;
        HttpURLConnection urlConn = (HttpURLConnection) url.openConnection();
        BufferedReader in = new BufferedReader(new InputStreamReader( urlConn.getInputStream()));
        String inputLine;
        int lineCount = 0;
        while ((lineCount < 10) && ((inputLine = in.readLine()) != null)) {
            lineCount++;
            result += "\n" + inputLine;
        } in.close();
        urlConn.disconnect(
        );
    }
    catch (Exception ex)
    {
        throw new IOException("Eroare conexiune");
    }
    return result;
}

private String dbInterrogation(String URL)
{
    int BUFFER_SIZE =
        2000; InputStream in =
        null; String result=null;
    try {
        result= OpenHTTPConnection(URL);
    } catch (IOException e1) {
// TODO Auto-generated catch block
        e1.printStackTrace(); return "";
    }
    return result;
}

public boolean onOptionsItemSelected (MenuItem item){
    String alias_localhost="193.226.17.162";
    String url="http://"+address+"/interog.asp"; String lecture_no;
    lecture_no=txtAfis((String)item.getTitle()

```

```
); url=url+"?"+"lecture="+lecture_no;
content=dbInterrogation(url);
txtShow.setText(content); return true;
}
public String txtAfis(String str) {
    if (str == null) {
        return null;
    }
    StringBuffer buffer = new StringBuffer();
    char c;
    for (int i = 0; i < str.length() ; i++)
    {
        c = str.charAt(i);
        if (Character.isDigit(c))
        {
            buffer.append(c);
        }
    }
    return buffer.toString();}
}
```