MARMARA
UNIVERSITY

4086 – Mobile Device Programming Project

Application Name: *Radio Station*

Created By:

150116842 – CANER YEŞİLDAĞ

150116839 – YASİN EMRE ÖZBARUT

150113075 – ZAFER EMRE OCAK

Course Lecturer:

Assoc. Prof. Ali Fuat ALKAYA

1. Aim of the Project

Radios are not famous in earlier years but provides us different music types and music cultures. We tried to implement an application that contains a lot of radio channels. For making our application more user friendly we implemented voice control option and recording radio stream on real-time. Increasing the usage of radio channels with our application is our main aim.

2. Scope of the Project

Creating a useful application for users that has

- Radio Streaming
- Voice Control
- Recording Radio Stream Real-Time
- Favourite List

3. Main Screen of the Application

We implemented a user friendly application that has a custom listview on the main screen. User can reach any radio with touching on the radio's name.

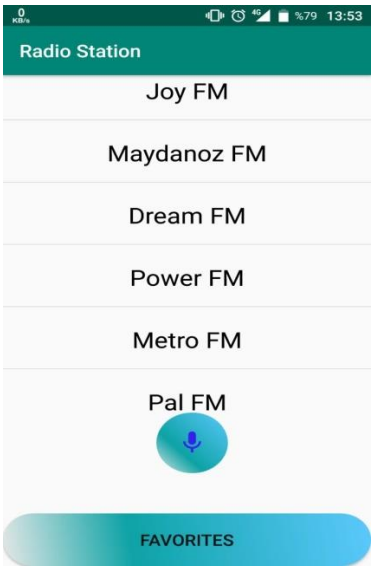


Figure 1: Main Screen

On the main screen we have 3 main options:

- Voice Control Button
- Favorites Button
- Touchable Custom ListView of Radios

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main_screen);
    lv =  findViewById(R.id.lv);
    btnnext =  findViewById(R.id.next);
    radioArrayList = getRadioList();
    customAdapter = new CustomAdapter( context: this);
    customAdapter.notifyDataSetChanged();
    lv.setAdapter(customAdapter);
    txtSpeechInput =  findViewById(R.id.txtSpeechInput);
    btnSpeak =  findViewById(R.id.btnSpeak);


    lv.setOnItemClickListener((parent, view, position, id) -> {
            sendMessage(view,(Radio)parent.getItemAtPosition(position));
    });

    btnnext.setOnClickListener((v) -> { nextActivity(v); });
    btnSpeak.setOnClickListener((v) -> { promptSpeechInput(); });

}
```

Figure 2: Main Screen's onCreate Method

We had a custom adapter for our custom listview. In this code with using our custom adapter which name is customAdapter, we created out touchable custom listview.

For proceeding the favourites activity we implemented a button that name is btnnext.

Another button is speak button. With this button user can reach the radio with saying its name.

3.1. Custom Adapter

Textview on our mainscreen is set with customAdapter.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    final ViewHolder holder;

    if (convertView == null) {
        holder = new ViewHolder();
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        convertView = inflater.inflate(R.layout.lv_item,  root: null,  attachToRoot: true);
        holder.tvRadio = (TextView) convertView.findViewById(R.id.temp);
        //holder.tvFavs=(TextView)  convertView.findViewById(R.id.isfav);
        convertView.setTag(holder);
    }else {
        // the getTag returns the viewHolder object set as a tag to the view
        holder = (ViewHolder)convertView.getTag();
    }

    holder.tvRadio.setText(MainScreen.radioArrayList.get(position).getChannel());
    //holder.tvFavs.setText(String.valueOf(MainScreen.radioArrayList.get(position).isFaved()));
    return convertView;
}
```

Figure 3: customAdapter getView Method

3.2. Like Button

We implemented a like button for adding or removing radios from the favourites list. With onClick method we control the buttons state and add current radio if its state is false on the other option we are removing current radio from the favourites list.

```
//LIKE BUTTON
likeButton=findViewById(R.id.heart_button);
likeButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(likeButton.isLiked()){
            likeButton.setLiked(false);
            MainScreen.radioArrayList.remove(stream.getId());
            stream.setFaved(false);
            MainScreen.radioArrayList.add(stream.getId(),stream);
        }else{
            likeButton.setLiked(true);
            MainScreen.radioArrayList.remove(stream.getId());
            stream.setFaved(true);
            MainScreen.radioArrayList.add(stream.getId(),stream);
        }
    }
});
```

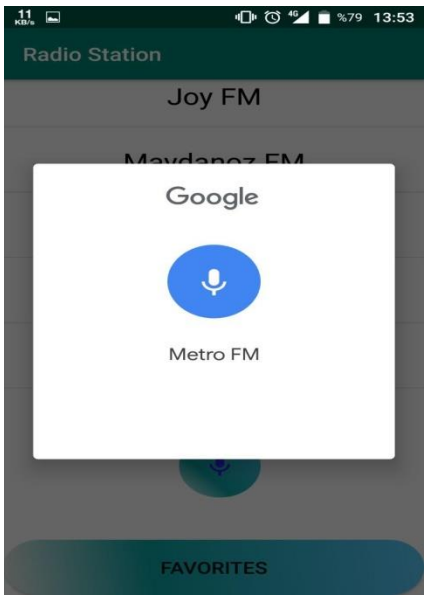
Figure 4: likeButton Implementatio

4. Speech to Text and Voice Control

In this app, we decided to create a voice command control system. Because there are hundreds of radio channels for listening in today. We have a long radio channel list for our user in this app. When a user would like to listen to radio channel he or she desire, he or she does not have to search this channel in long radio list by hand.

You can tap to the mic and say your radio channel name that you desire listening. Our app will find this channel immediately.

We can see how it works.



In this part, we created a new intent and it provides starting new activity to listen user speech command.

```
private void promptSpeechInput() {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT, getString(R.string.speech_prompt));
    try {
        startActivityForResult(intent, REQ_CODE_SPEECH_INPUT);
    } catch (ActivityNotFoundException a) {

    }
}
```

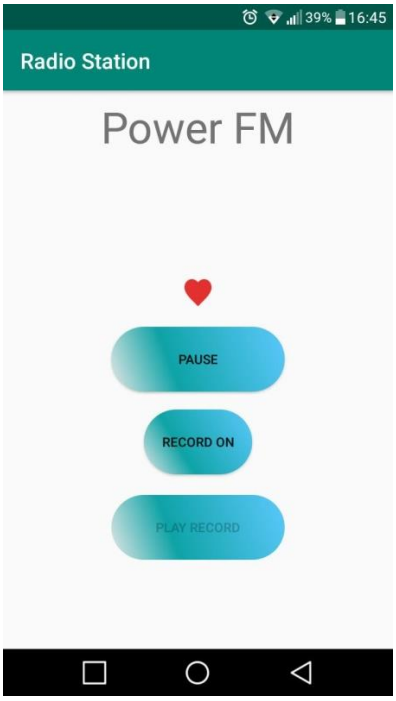
In this part, we get user's speech text data. If result code is OK and speech text data is not null, we store this text data in result value with string type. After this operation, if result value is located in our radio channel list, it will find and return desired radio channel.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    switch (requestCode) {
        case REQ_CODE_SPEECH_INPUT: {
            if (resultCode == RESULT_OK && null != data) {

                result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
                txtSpeechInput.setText(result.get(0));

                for (Radio radios : radioArrayList){
                    if (radios.getChannel().equals(result.get(0))){

                        sendMessage(view,radios);
                    }
                }
            }
            break;
        }

    }
}
```

5. Play – Record – Replay Radio Stream

First of all, this part of our application takes place in a different activity. Initially the user selects a radio channel and then the program guides the user to this screen.

At this part, there are 3 buttons available but each action triggers other buttons depending on user actions. For instance, without recording, the replay button will not be available to the user. Because, if there is no record file then you cannot listen to it.



Another example is that, if the connection is not established the “PLAY” button is going to set false. Finally, if everything is OK then user can start listening to radio that he/she picked at the main screen.

There are 5 important features we are providing to users on this activity:

- The name of the radio channel is displayed at top of the screen. So this way, user will know which channel he/she listens to at the moment.
- There is a like button which users can press and record into a favourite list. Later on, users can remember which radio channels he/she did like.
- Basically, user can play the stream by pressing play button. There are actually a couple of different methods have been used. First and foremost, the play button actually utilizes AsyncTask. Whenever this activity is showed up, the activity will receive a stream URL and tries to establish a connection via AsyncTask.

  AsyncTask has an important feature which is to execute the connection process at the background side of the program. Through this, users will not interrupted by this task. After the task is done then the final methods will be called and everything is done.

AsyncTasks are invoked by creating a subclass. The existing subclass is shown below:

```
class PlayerTask extends AsyncTask<String, Void, Boolean> {

    @Override
    protected Boolean doInBackground(String... strings) {

        try {
            mediaPlayer.setDataSource(strings[0]);
            mediaPlayer.prepare();
            prepared = true;
        } catch (IOException e) {
            e.printStackTrace();
        }

        return prepared;
    }

    @Override
    protected void onPostExecute(Boolean aBoolean) {
        super.onPostExecute(aBoolean);
        playButton.setEnabled(true);
        playButton.setText(getString(R.string.play));

    }
}
```

- Record On/Off Button starts and stops saving stream. It is created whenever the Record Button is pressed. It takes two arguments to start the thread. First argument is the file and the second argument is the streaming link that the worker thread will receive and overwrite into a record file. Whenever user presses the “Record Off” button, the stream connection, the FileOutputStream object stop the process and close.

The “Record Button” Listener method is shown below:

```
// RECORD BUTTON LISTENER
recordButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(recordButton.getText().toString().equals("RECORD OFF")){

            try {
                outputStream = new FileOutputStream(
                        new File(getFilesDir(), "My_Record.mp3"), false);
                //newRecord = getFilesDir(), "My_Record" + i + ".mp3");
                //i++;
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
```

```
            recordButton.setText(getString(R.string.recordOn));
            recorder = new Recorder(outputStream, stream);
            new Thread(recorder).start();
            playRecordButton.setEnabled(false);

        }else{
            try {
                recorder.terminate();
                Thread.currentThread().join(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            recordButton.setText(getString(R.string.recordOff));
            playRecordButton.setEnabled(true);
        }
    }
});
```

- Finally, “Play Record” Button is redirect the newly created record file to the already created MediaPlayer object and this object starts to playing from this file as reading byte content.