



L-Università ta' Malta
Faculty of Information &
Communication Technology

Department of
Computer Information
Systems

Mobile Computing Report

50905L

*B.Sc. (Hons) Software Development.

Study-unit: **Mobile Computing**
Code: **CIS2208**
Lecturer: Conrad Attard

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I / We*, the undersigned, declare that the assignment submitted is my / our* work, except where acknowledged and referenced.

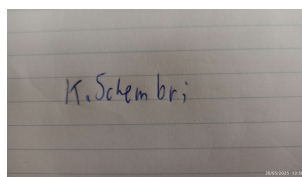
I / We* understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected and will be given zero marks.

* Delete as appropriate.

(N. B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Keith Schembri



Student Name

Signature

Student Name

Signature

Student Name

Signature

Student Name

Signature

Course Code

Title of work submitted

Date

Contents

1. Introduction.....	1
2. Technologies Deployed	1
3. Introduction.....	2
4. MainActivity	2
4.1. UI.....	2
4.1.1. Edge to Edge.....	Error! Bookmark not defined.
4.2. Spinners	2
4.3. Buttons.....	3
5. PadsPage	3
5.1. Core Functionality and Architecture.....	Error! Bookmark not defined.
5.1.1. Musical Chord Generation	4
5.1.2. Audio Playback	4
5.1.3. MusicBrainz API Integration	5
6. ChordPadFragment	5
6.1. Overview	Error! Bookmark not defined.
6.2. Components and Fields:	6
6.3. Functionality	6
6.3.1. Fragment Instantiation.....	6
6.3.2. Lifecycle Methods	6
6.3.3. Touch Gesture Handling(OnTouchListener)	Error! Bookmark not defined.
6.3.4. Helper Funcitons	7
6.4. Usage.....	Error! Bookmark not defined.
6.4.1. Hosting	7
6.4.2. Instantiation:	8
6.4.3. Layout.....	8

7. Artist Class	8
7.1. Fields	8
7.2. Getters	8
8. ArtistResponse Class.....	10
9. MusicBrainzApiService Interface.....	10
10. MusicBrainzClient.....	10
10.1. Overview	10
10.2. Key Components and Fields.....	10
10.3. Core Functionality.....	11
Process Flow	11
11. Tag	11
12. UserAgentInterceptor Class	12
12.1. Functionalities.....	12
12.2. Key Components	12

1. Introduction

This application is designed in order to help beginner musicians and producers to explore different chord progressions and spark creativity during music composition. Once the user sets a few initial preferences, the app shows four playable chord pads on the screen. Each pad plays a different chord based on the user's chosen settings. As a bonus, the app displays facts about musical artists atop the pads, adding a little inspiration while the user plays.

2. Technologies Deployed

- AudioManager[1]: it is a component used for Audio stream control.
- SoundPool[2]: it is a component used for the playback of sounds
- Activities[3]: A core component that represents a single screen with a user interface in android.
- Fragments[4]: Fragments are reusable items in the app. This helps prevent any duplication from being written.
- Intents[5]: They are a way on how to transfer data between different activities.
- ViewCompat[6] and WindowsInsetsCompat[7] which allows new APIs to run on older versions on android.
- Toolbar [8]
- Drawer Layout[9]: A way on how one can navigate throughout the app.
- Navigation view[10] for side-menu navigation.
- Edge to edge[11]: A way on how UI elements can take up the whole screen by drawing behind the system bars.
- Retrofit[12]: HTTP client for android, lets the app connect to the internet.
- Gson[13]: A package for Json parsing
- MusicBrainzAPI[14]: an open-source encyclopaedia of music information. Its data can be used directly with its REST API.
- Music Theory Logic: Uses Chord Generation based on scales
- Material Design Paradigm[15]: A design language characterised by cards, flat designs and responsive UI.

3. Introduction

Music composition and improvisation are essential skills for musicians but beginners often struggle with music theory concepts like chord progressions and harmonic structure. To bridge this gap, this report presents “Chordz 4 Dayz”, an Android application designed to simplify music creation by providing an interactive interface for playing chord progressions.

The app features a 2 by 2 grid of pads, each triggering a chord from a random selection from a set of chord progressions. A chord is based on a selected key, with intuitive dropdown menus that let the person select their key, complexity (triads, 7th chords or extended chords). By integrating audio synthesis via SoundPool and adhering to music theory principles.

Furthermore, the app also offers facts about a random musical artist on top of the pads from the “MusicBrainz” API, an open source musical encyclopaedia.

4. MainActivity

4.1. UI Functionality

According to the Android Documentation, EdgeToEdge[11] makes sure UI components take up all of the screens by drawing behind the system bars (Status Bar, Navigation Bar). This contributes to a seamless experience that avoids borders. Furthermore, the status bar and navigation bar is hidden to further show this Edge-To-Edge technology.

4.2. Spinners

The “Spinner” Function offers dropdown lists for easy selection of scales.

The list of variables that the user can select are the:

- Letter Name(A,B,C,D,E,F,G): This selects the base note of the scale
- Accidental : Also changes the root note slightly
 - Flat(\flat)- Lowers the sound by one semitone (offset -1) (e.g. from E becomes E \flat)
 - Natural(\natural) – Doesn’t change the sound (e.g. F stays F).

- Sharp(#) – Raises the sound by one semitone (offset +1) (e.g. from C becomes C#)
- Mode: Defines whether the progression is major or minor. (Major is brighter while Minor is darker).
- Complexity: Defines the complexity of the chord.
 - Triads consist of only three notes (root, third and fifth).(e.g. a C major triad is C, E, G)
 - Seventh chords add the seventh note on top of a triad. Adding colour or tension. (e.g. a C major 7th chord is C,E,G,B)
 - Extended chords have a randomised chance to add a seventh, ninth and thirteenth note in the chord. (e.g. a C major 9th is C,E,G,B,D).
- Instrument: Chooses the sample between Piano, Rhodes or Guitar.

I used two dropdown menus to choose the root note because that's how musicians normally think about notes. Hence, it is more natural to select the letter name of the note and an accidental afterwards.

4.3. Buttons

When the open Button is Clicked, the user proceeds to the PadsPage activity. First, it creates an Intent object that stores the variables declared by the spinners as extras. It then starts the PadsPage Activity with the generated intent. This will help carry the selected scale and instrument data to the PadsPage screen.

5. PadsPage

The PadsPage is an activity where the user plays the 4 chords of a chord progression on buttons(called pads) in a 2 by 2 grid. Above the pads there is a piece of text displaying a fact on about a music artist from the MusicBrainz API.

The PadsPage Activity extends from the AppCompatActivity and implements the ChordPadFragment.OnChordPlayListener. This suggests that, as well as its role as a UI element, is an event-receiver from its dependent chordpad UI elements. Its design is based on several essential functional areas.

5.1. Musical Chord Generations

The Core Function of the PadsPage is chord creation

This is done via:

- **Musical Scales:** The scale array, dynamically set to either major or minor, defines the intervals between notes. This array serves as the building blocks for creating chords within a specific musical scale.
- **Chord Progressions:** Predefined majorProgressions and minorProgressions arrays provide chord sequences. When the PadsPage initialises, a chordProgression is selected at random from the appropriate set of chord progressions. This then creates the four interactive chord pads.
- The getChord method is essential in creating individual chords. It takes a rootIndex and calculates the root third and fifth of the chord based on the chosen scale.
- The getChord also implements logic for varying chord complexity based on user preferences (passed using Intent extras as "COMPL"):
 - 7ths complexity adds the seventh degree to the basic triad.
 - Exts (short for extensions) complexity randomly adds seventh, ninth and thirteenth degrees to the chord.
 - For chords marked as altered, the getChord method introduces randomly altered extensions such as flat 9th, sharp 9th or flat 13th, adding harmonic complexity and tension.
- The offset value(which is a result of the letter name and accidental from the Intent extras) allow the entire musical scale to be transposed, by a certain number of semitones, enabling playback in different keys.

5.2. Audio Playback

Sound playback is managed efficiently using the SoundPool class, which is a simple audio playback class.

- **Instrument Loading:** Modified according to the "INSTR" Intent extra, the appropriate instrument sound file is loaded from the raw file in the project during onCreate. A isLoading flag ensures that playback attempts only occur after sound data is fully loaded.
- **Chord Playback (chord method):** This method iterates through the semitone values from the getChord from earlier and for each note in the chord, it calls the

soundPool.play() [2] method, dynamically adjusting the rate using $\text{Math.pow}(2, (\text{note} + \text{offset}) / 12)$. This formula transposes the single loaded instrument sound to produce the desired musical pitch for each note, this is what enables the chord to finally be played.

- Resource Management: In the onDestroy method, the SoundPool[2] resources get released via the release() function in order to avoid memory leaks and endure proper system resource management.

5.3. MusicBrainz API Integration

In order to enhance the user experience with engaging content, the PadsPage fetches random artist facts:

- Retrofit[12]: The retrofit library makes asynchronous network requests to the MusicBrainz API[14]. Automatic JSON parsing is enabled with the GsonConverterFactory [13].
- FetchRandomArtistsFact: this method initiates a search for random artists. The enqueue method ensures the API call runs in the background.
- onResponse and onFailure: These callback methods handle the results of the API call, where the program either displays a formatted fact about a random artist or showing a failure to load. The factText view shows the fact or error message.

6. ChordPadFragment

6.1. Summary

The ChordPadFragment is a modular user interface element that represents a single interactive button, or “pad”.

- Chord Playback: Using a listener interface, the Fragment produces a chord sound in response to a single tap.
- Background Customisation: By using a two-tap gesture that opens an image picker, users can alter the pad’s visual background.
- State Management: Touch gesture recognition is managed by each pad, which also stores the related chord information.

6.2. Components and Fields:

Included are the following important fields and components:

- `ActivityResultLauncher<String>`: A function to start the image selection activity and manage its result.
- `currentPadToChange`: This View is used to store the reference to the pad whose background the user is currently updating.
- `listener`: This is an object of type `OnChordPlayListener` which interfaces between the chord being played and `Fragment`.
- `selectedChords`: A 2D array representing the musical notes (semitone values) for the chords loaded into all pads.
- `Fab`: A Floating Action Button that enables the user to change the background of the pad.

6.3. Functionality

6.3.1. Fragment Instantiation

The `newInstance` static factory simplifies the creation of `ChordPadFragment` objects.

It accepts two parameters:

- `chordIndex`: an integer which determines which chord is to be rendered.
- `chords`: a two-dimensional array of integers that represents the chords in the chord progression.

6.3.2. Lifecycle Methods

- `onAttach(@NonNull Context context)`: Ensures that the host context implements the `OnChordPlayListener` interface. If not, a `RuntimeException` is thrown to prevent runtime errors.
- `onCreate(@Nullable Bundle savedInstanceState)`:
 - Initialises `pickImageLauncher` using `registerForActivityResult`, setting up a contract to launch an image picker.
 - Defines a lambda callback (`uri->{}`) to handle the selected image. If an image is selected, it attempts to@+:
 - Open an input stream from the URI.
 - Convert the stream into a `Drawable`.

- Set the drawable as the background for the currently selected pad
 - The callback tries to open an Input Stream from the selected URI, create a Drawable from, it and set the drawable as the background of the currentPadToChange. Error logging is included for cancellation.
 - A GestureDetector is initialised to specifically listen for double-tap events. When a double-tap event is detected, it calls changePadBackground().
- onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):
 - Inflates the layout found in R.layout.fragment_chord_button in order to create the fragment's UI.
 - Assigns the inflated rootView to padButtonView.
 - Sets up an OnTouchListener on padButtonView which is integral to multi-touch gesture detection.
- onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState)
 - Retrieves the chordIndex and selectedChords from the fragment's arguments.
 - Gets a reference to the chord_button in the layout.
 - Assigns a chordIndex to the button's text.
 - Assigns an OnClickListener to the chord_button. Upon clicking, it calls the onPlayChord method of the listener to play the chord of the pressed pad.

6.3.3. Helper Functions

- changePadBackground(View padView): stores the padView into currentPadToChange and then launches the pickImageLauncher to open the device's image gallery.
- setSelectedChords(int[][] selectedChords): A public setter to update the selectedChords array for the pad. This allows the hosting PadPage activity to dynamically change the chord associated with the pad without recreating the fragment.

6.3.4. Hosting

The Activity must implement the OnChordPlayListener interface in order to receive callbacks when a chord pad is tapped.

6.3.5. Instantiation:

New instances must be created using the `ChordPadFragment.newInstance(int chordIndex, int[][] chords)` factory method, passing the relevant chord data.

6.3.6. Layout

The fragment expects a `Button` with the ID `chord_button` in its layout (`fragment_chord_button.xml`) for single-tap.

7. Artist Class

The artist serves as an object to model artist data retrieved from the MusicBrainzAPI. It encapsulates crucial information about an artist, making it suitable for data transfer and deserialization of JSON responses into Java objects.

7.1. Fields

- `private String id`, holds the unique MusicBrainz ID for the artist within the MusicBrainz database.
- `private String name`, This field stores the name of the artist.
- `private String disambiguation`, holds additional textual information that aid in telling the specific artist and other artists apart.
- `Private List<Tag> tags`: Stores the list of tag objects that are associated with the artist. Each Tag object is expected to contain details such as the tag's name and its relevance count.

7.2. Getters

These public methods provide controlled read-only access to the private fields:

Whenever the programmer wants to get an attribute, the getters provide a safe way to access them they include:

- `Public String getId()`
- `Public String getName()`

- `Public String getDisambiguation()`
- `Public List<Tag> getTags()`

These getters return the id, name, disambiguation and list of tags respectively

8. ArtistResponse Class

The ArtistResponse class acts as a container for a collection of Artist objects, this mirrors the structure of a JSON response from the MusicBrainzAPI when an API call returns a list of artists. This enables the automatic parsing of JSON data into a usable Java object structure. It only has one field, that being the artists which is a list of artists that was fetched by the API. It only has a getter that retrieves the list of artists. The ArtistResponse Class has no setters since it is meant to be read only.

9. MusicBrainzApiService Interface

The MusicBrainzApiService interface acts as a contract for interacting with the MusicBrainzAPI. It defines the structure of the API endpoints the application communicate searchArtist

It sends a GET request to the MusicBrainz API via the endpoint “ws/2/artist”

It appends:

- Query which filters to only include people or events or anything else.
- Format the format in which the API response should be sent.
- Limit which maps a limit parameter to the URL

10. MusicBrainzClient

10.1. Overview

This class consolidates the configuration and creation of the MusicBrainzApiService, which defines the API endpoints. The singleton pattern is added to prevent redundant initialisation of network components, which can be taxing for performance and memory.

10.2. Key Components and Fields

The class contains a constant BASE_URL = <https://musicbrainz.org/> in which all requests are based are defined relative to this URL.

It has a static retrofit instance that is reused for all subsequent API interactions, embodying the singleton pattern

10.3. Core Functionality

There is only one method which is called `getClient()`. It guarantees that a fully configured Retrofit client is provided by implementing the singleton logic. It returns a `MusicBrainzApiService` Interface which is used to make API calls.

Process Flow

1. It first checks if the retrofit instance is null. If so then it initialised the client.
2. An `HttpLoggingInterceptor` is instantiated. This interceptor is used for printing HTTP request and response data to the Console(Logcat).
3. An `OkHttpClient.Builder` is used to construct a custom `OkHttpClient` instance. `OkHttp` is the HTTP client that is used to make network requests.
4. Two interceptors are added. The `HttpLogging Interceptor` from earlier and a custom `UserAgentInterceptor` which is important for including a User-Agent header for every outgoing request.
5. A `Retrofit Builder` is used in order to construct a retrofit instance.
6. The `BASE_URL` is set, defining the root for all API endpoints.
7. The custom `OkHTTIClient` is set as the client for Retrofit.
8. The `GsonConverterFactory.create()` method is added as a converter factory, in order to automate serialisation and deserialization of Java objects to/from JSON.
9. The `retrofit.create(MusicBrainzApiSercive.class)` is called. Retrofit dynamically generates an implementation fo the `MusicBrainzApiService` interface based on the `@GET,@Query,@Path`
10. The configured `MusicBrainzApiService` instance is returned.

11. Tag

A tag is the blueprint for a tag that a MusicBrainz entry has. It has only two fields; a name of type string and a count of type int. The name variable represents the name of the tag, such as, rock, jazz, political, etc. which give descriptors of the artists genres, cause of death, etc. The count variable represents how many times the tag is “upvoted” by the community. Getters are set to return the name and count as well.

12. UserAgentInterceptor Class

The `UserAgentInterceptor` is a specialised component within the `OkHttp` library. This class is used for automatically adding a specific “User-Agent” header to every HTTP request your app sends. This header identifies the application when it talks to services like `MusicBrainz`.

12.1. Functionalities

- App Identification: When the app makes an API call, this interceptor announces itself with a custom name. Many web services use the “User-Agent” string to understand who is making the requests.
- API Requirements: The User-Agent header is required in order to be accepted by the API.
- Tracking and Analytics: It helps the API provider track usage patterns, this is useful for traffic management.

12.2. Key Components

- Private final `String userAgent`;

This field stores the text that identifies the app in the User-Agent header. It's set when you create an instance of this interceptor.

- Public `UserAgentInterceptor(String userAgent)`

This is the constructor. This method is invoked when a new instance needs to be made.

- Public `Response intercept(Chain chain)` throws `IOException`

`OkHttp` calls this method for every single HTTP request the app makes.

- `Chain.request()`: This gets the original request that is going to be sent.
- `.newBuilder().header("User-Agent", userAgent).build()`: This part of the code and creates a modified version of the original request and adds the “User-Agent” header with the custom string that is provided.
- `Chain.proceed(request)`: After the request is modified, `OkHttp` continues with the process, by sending the modified request to the network. A response is then returned from `OkHttp`.

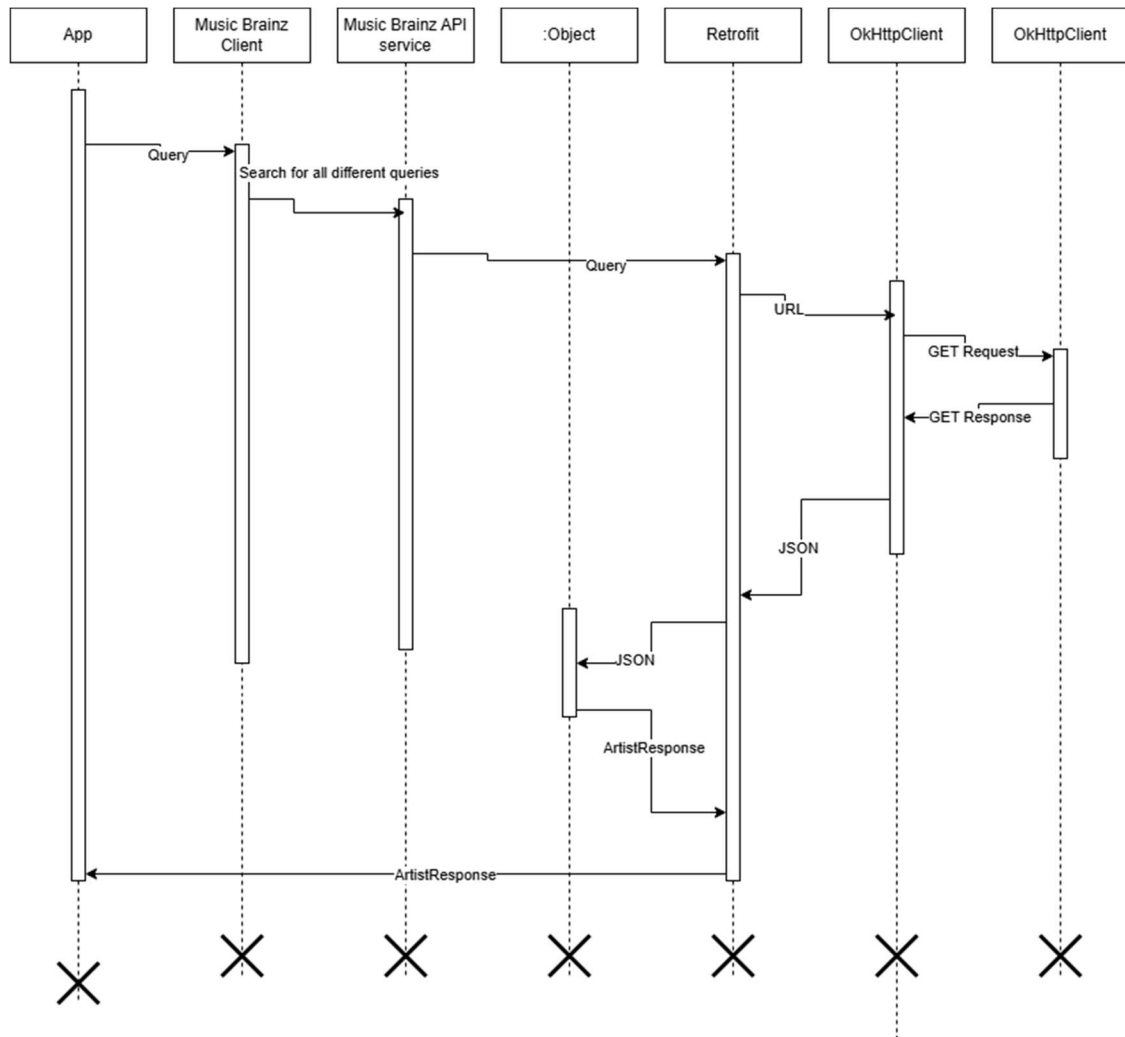


Figure 1: The UML Sequence diagram of the how the app retrieves the ArtistFacts

13. Layouts

13.1. activity_main.xml

The activity_main.xml is the first page the user sees when he/she enters the app. The user is greeted with the title "Chordz 4 Dayz" displaying prominently. In the app there is also a toolbar, with the hamburger menu icon, five drop-down menus that lets the user choose what root note, mode, chord complexity(the amount of distinct notes in each chord) and a button that guides the user into the pads_page.xml page. This is displayed as a quite minimalist, material art[15] style.

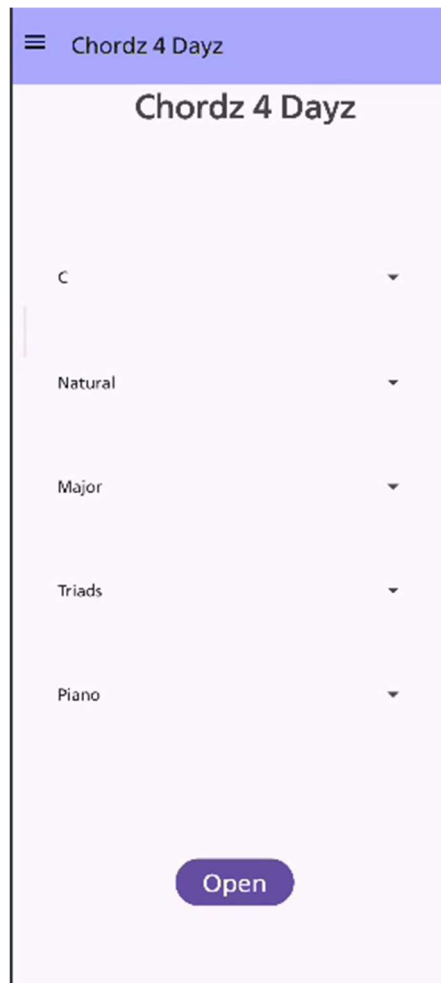


Figure 2: A screenshot containing the home screen of the app

13.2. pads_page.xml

The pads page is the page that the user goes to upon pressing the Open Button. The user is greeted with four large buttons called “pads” with floating action buttons on the bottom right hand side of each of them. On top of the pads the user finds a chunk of text, showing facts about a musical artist. At the very top of the screen we have the same toolbar that the user can use to navigate back to the main activity. This incorporates a material art style[15] and a flat UI design.

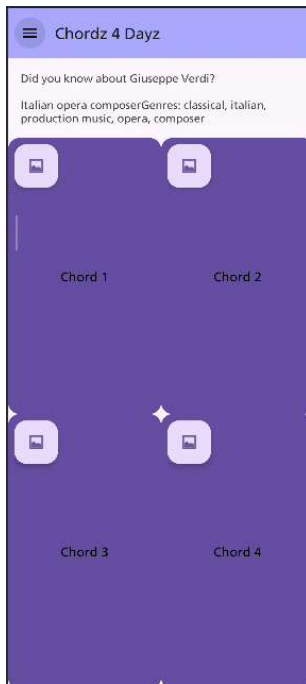


Figure 3: The pads page of the application

13.3. fragment_chord_button.xml

This xml file contains the fragment that represents every pad in the PadsPage. The fragment_chord_button.xml fragment consists of a responsive button that plays the chord and a floating action button that is used to change the background of the button.

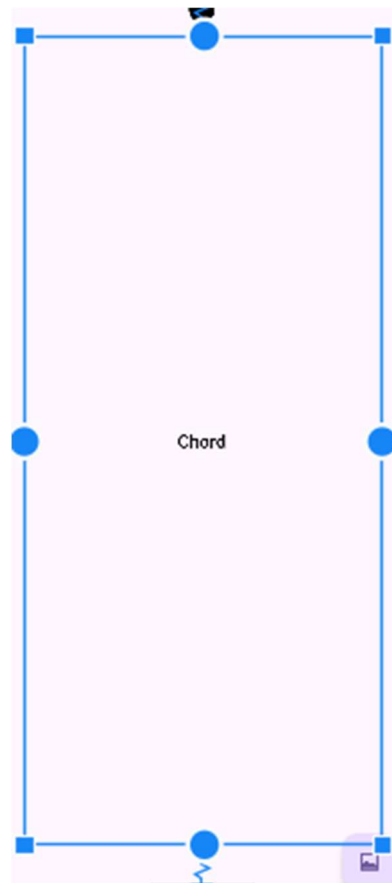
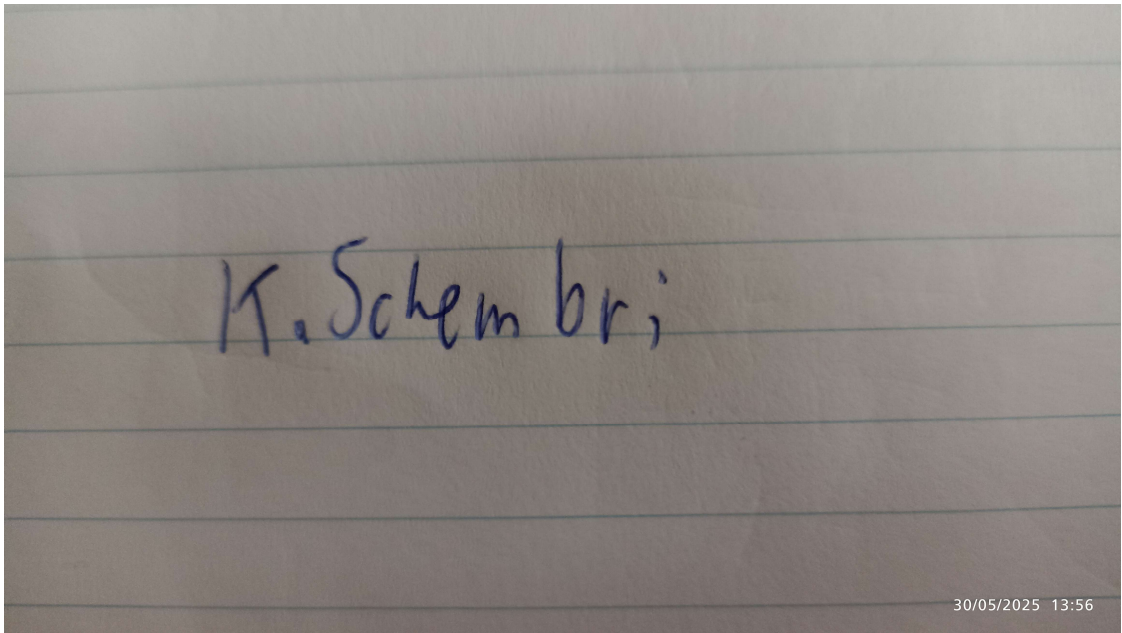


Figure 4: A single chord pad

14. Drawables

14.1. Button_translucent.xml

This xml file contains a simple rectangle. That is transparent when unclicked but becomes translucent when clicked. This adds a more responsive feel to the app. This xml file is located in the drawable folder.

14.2. Hamburger_menu.xml

This is the standard hamburger menu icon used to open the hamburger menu.

15. nav_menu.xml

This xml file contains the menu items in the navigation menu. This interface facilitates the navigation between the two pages.

16. Strings.xml

The strings.xml file contains the important string values that are used within the application including the official name of the application and the values of the drop down menus. This file is located in the values folder.

17. Raw files

The Files in the raw folder contain the samples of the piano, Rhodes and guitar, stored in wav form. These files are loaded when the pads page is loaded.

18. Android Manifest

The changes made to the androidmanifest.xml file in are the permissions, where it uses the permission to access the Internet in order to fetch the artist fact from the MusicBrainzAPI[14] and the permission to read media images which is used to replace the chord buttons with your own images.

References

- [1] Google LLC and the Open Handset Alliance, "AudioManager," 2025.
Available: <https://developer.android.com/reference/android/media/AudioManager>.
- [2] --- "Soundpool," 2025.
Available: <https://developer.android.com/reference/android/media/SoundPool>.
- [3] ---"Activity," 2025.
Available: <https://developer.android.com/reference/android/app/Activity>.
- [4] --- "Fragments," 2025. Available: <https://developer.android.com/guide/fragments>.
- [5] ---"Intent," 2025.
Available: <https://developer.android.com/reference/android/content/Intent>.
- [6] ---"ViewCompat," 2025.
Available: <https://developer.android.com/reference/androidx/core/view/ViewCompat>.
- [7] ---"WindowInsetsCompat," 2025.
Available: <https://developer.android.com/reference/androidx/core/view/WindowInsetsCompat>.
- [8] ---"Toolbar," 2025.
Available: <https://developer.android.com/reference/android/widget/Toolbar>.
- [9] ---"DrawerLayout," 2025.
Available: <https://developer.android.com/reference/androidx/drawerlayout/widget/DrawerLayout>.
- [10] ---"NavigationView," 2025.
Available: <https://developer.android.com/reference/com/google/android/material/navigation/NavigationView>.
- [11] --- "Display content edge-to-edge in views," 2025.
- [12] Square Open Source, "Introduction," 2025. Available: <https://square.github.io/retrofit/>.
- [13] Google LLC, "Gson," 2025. Available: <https://google.github.io/gson/>.
- [14] the MetaBrainz Foundation, "MusicBrainz API," 2025.
Available: https://musicbrainz.org/doc/MusicBrainz_API.
- [15] Google LLC, "Material Design," 2025. Available: <https://m3.material.io/>.