Emulair

by Radu Bratan,
Eduard Cristea,
Cosmin Colceru,
Alexandru Petrescu,
Eduard Marin Florin,
Mihai Grigore

Table of contents:

Software purpose

Emulair is an Android exclusive open-source front-end for Libretro cores, based on Lemuroid. The primary goal of Emulair is to combine the ease of use and simple but accessible interface of Lemuroid with RetroArch's extensive customizability and features, while also adding support for standalone emulators.

At this point in time, Emulair is a robust emulation frontend, offering users save states, shaders, on-screen touch controls, automatic game finding, gamepad connectivity, access to the emulators' settings, local multiplayer, a modern and easy to navigate interface, as well as plenty customizable preferences that greatly enhance the ease of use of the app.

Until project fulfillment, we plan to offer users the ability to access all their save data, a prettier interface with fancy animations, as well as refined mechanic (such as better controls customization, access to more settings, improved support for external hardware and internal sensors, multiple layouts for differently sized screens, as well as even more customizable preferences that should allow users to take full control of how they want to use this app).

Guides

Installing Android Studio

This project is being worked on using Android Studio. Unfortunately, other IDEs cannot build this app, so you will have to download it. To install Android Studio, go to the Android Developers website, click "Download", and follow all the necessary steps.

Cloning the repo

To clone the repository on your PC, it is recommended to use Git, but you can also just click on "Download ZIP" directly from the web version of GitHub.

Don't forget to also clone the LibretroCores repository that contains all the emulators Emulair is using, if you wish to create a version of Emulair with built-in cores that don't require any in-app download. Place this repo directly in the root folder, where files such as .gitignore and README.md are located.

Building Emulair

Once the project has been opened, Android Studio will automatically download all necessary libraries and sync all the build.gradle files with the project. In case syncing fails or is interrupted, click on the "Sync project with Gradle files" button located at the right section of the top toolbar (the one that looks like a miniature elephant, the logo of

Gradle). To build the project, simply click on the green hammer shaped button, located on the same toolbar. If anything fails, check if all the files are in place and if you are using the latest version of Android Studio.

Choosing what version to run

Emulair has four flavours, two related to the cores (bundled or downloaded), and the other two related to what sources the app is downloaded from (Google Play or other sources). There is also a distinction to be made between "Debug" and "Release" - the "Debug" version has different colours and a different name ("Emulair D" instead of simply "Emulair"), just so it won't be confused with the "Release" version during development. For testing purposes, it is recommended to choose the debug version, along with the "bundled" and "gplay" flavours. To select a specific flavour for building, go to Build -> Select build variant..., click on :emulair-app's Active build variant, then select a desired build type from the dropdown menu.

Running Emulair on Android Studio's built-in emulator

To run Emulair directly from Android Studio, you need to set up a virtual Android device in the "Device Manager" section. Upon completing the setup, simply select your virtual device from the top toolbar's dropdown menu, then click the green arrow button on the right to run the app.

Running Emulair on your mobile device

To run Emulair directly on your Android device, you'll first have to enable USB Debugging from your phone's settings, then select your physical device from the same dropdown menu mentioned above. The last step is to click the green "Run" button.

Contributing

If you wish to contribute to this project, there are certain requirements for you to meet before any of your work can be accepted. Firstly, to maintain consistency across the project, it's crucial to follow the existing Kotlin code style. Secondly, any of the changes you make should use little to no programming hacks, because such code becomes unmaintainable in the long run.

It is advised that you focus on the main goals Emulair tries to accomplish, and not waste everyone’s time by creating features that would not fit in this project. Accurately pointing out what features fit and what don’t is a subjective task, which is why it is always encouraged to first open a discussion thread on GitHub to ask the code owners and principal mainteiners on feedback. Lastly, please maintain a professional level of respect towards everyone and do not break any of GitHub’s rules.

Application entry points

Data sources

As data sources, we use a static SQLite database containing indexed entries for almost every game any of the internal emulators can run. Those entries include the name of the game, the platform they were created for, the publisher, as well as other fields used for identification.

On top of our static database, we use Android's Shared Preferences to locally store user settings, as well as the Storarge Access Framework to read the game files stored on every user's machine.

For the purpose of retrieving cover art for every game, we use Retrofit and OkHttp to acces Libretro's thumbnails database.

Data inputs

We make use of UI inputs such as a spinner that appears while a game is loading and a progress bar to show that the app is scanning for games in real time. The spinner can be triggered only if the user selects a game and opens it, while the progress bar appears at launch time, after the app is resumed and after a game session was closed.

Intents are used in Emulair in order to pass relevant data to the game loader, before a game can be run or while a game is being run by an emulator. Such relevant data consists of user preferences and emulator settings. User preferences can be set only in the settings screen and are passed via an intent before the game is loaded. Emulator settings can be changed while a game is running from a pause menu and are passed back to the game activity after the menus is closed.

Services in Emulair take care of every action, input, setting and preference of the user. An example of a service is the thumbnail loading service which handles the logic of connecting to the Libretro database to obtain relevant images or, as fallback if there is no internet connection of if no thumbnail appears in their database, generate an image consisting of a color and the initials of the game's name.

Configuration files

The Android Manifest contains all of our essential configurations for the app to work, including the structure of the activities, the requested permissions and the theme. The build.gradle contains all of our essential configurations for building the app, including dependencies, flavors, build types and targeted Android versions. The res/values folder includes all of our XML files, including strings, colors, themes, preferences and dimensions. The proguard-rules file is used to obfuscate and optimize the code.

High level diagrams

User journey

On app launch, the user lands on the games page. The bottom bar allows the user to navigate to the games page, the systems page and the settings page. The games page has a search icon that allows the user to search for games. The games, systems and settings pages have a "more information" button on the top left corner that allows the user to acces a page containg frequently asked questions about the app, while the top right corner has a "profile" button that leads the user to an empty profile page that is still in development.

When clicking on a system from the systems page, a new page containing all the games of that system appears. When clicking on one of the sections in the settings page, a new page with more settings appears. When clicking on a game in the games page, in the systems's games page or in the searched games list, the user is sent to a game activity that loads their game. During the game, the user can long press a menu button to pause the game and open a pause menu, which allows them to save their game state, load one of their previous states, change emulator settings, close the pause menu or quit the game and return to the games page.

Most valuable output

Our MVO is defined as “increasing the conversion rate of users from dedicated emulators to our application with integrated emulators”.

MVO Analysis

- metrics: the conversion rate of users from dedicated emulators to our application with integrated emulators
- impact factors: quality and diversity of features, app accessibility, efficiency of promoting the app on the Internet
- optimization strategies:
    - offering more features: compared to dedicated emulator apps, Emulair should offer users a modern UI and more settings to customize their app and gameplay experience
    - improving the UI: the UI is essential to how an app is percieved, which is why an accessible interface with plenty of ways to interact should make Emulair more accessible compared to other apps which are usually barebones and focus solely on the most basic features

- promoting the app: after successfully implementing every planned feature, spreading the word on social media (such as Reddit) about how good our app is should help boost popularity

- measure success: take a look at what people say on the Internet about the app and how they describe it, then make use of the user's feedback (from emails, direct messages, social media, GitHub or Play Store comments) to understand what works and what doesn't

Deployment plan

Our web team who created the presentation site are ready to deploy the web application. The plan is to deploy the app using azure database and hosting service. Our web Team Lead already purchased the domain emulair.com and emulair.ro and soon the application will go live.

QA process

Our test suite is comprised of test cases and test data. Usually, our test data includes dummy text (such as lorem ipsum) and/or dummy images (any random image already present in our app) for when we want to create a new component, a shortened version of our SQLite database for when we want to test functionalities related to it, or even manually generated data (such as game files or preferences) when we want to test data processing mechanics.

Test cases vary. In the context of providing the users acces to their save files, we first generated save files using the built-in state mechanism, then we pressed the button that should allow the transfer of the files from the internal storage to the emulated storage. If the files were not transferred, if the app crashed or if the app took too much RAM to transfer the files (above a specified hard limit in GB), then the test case failed.

External dependencies

APIs

We use Libretro's thumbnail database to allow users to view cover art for their games. In older versions of our app, we used the Firebase API to store data and retrieve it from their servers, as well as the Google Drive API to fulfill the same purpose.

Libraries

We are using a variety of libraries across different categories such as AndroidX, Google Dagger, Kotlin, Epoxy and Google Play Services.

Considering most of our dependencies are outdated, our project could be vulnerable to dependency attacks. However, this is mitigated by the fact that all of our libraries come from either official or highly trusted sources, as well as the fact that there are very few transitive dependencies present.