UNIVERSITY OF TRENTO

Department of Information Engineering and Computer Science

Bachelor's Degree in
Computer Science

Final Dissertation

Interactive Code Playgrounds on Android

Using mobile devices to make educational technologies more accessible in contexts of scarcity

Supervisors
Lorenzo Angeli
Maurizio Marchese

Student
Salvatore Gilles Cassarà

Academic year 2023/2024

Contents

1 Summary

1.1 Problem Statement

In the current times, digital skills are becoming essential for every person, but not everybody can easily acquire these skills: the digital divide still is a great problem, with 1/3 of the global population lacking internet access and universities struggling to provide computers to all their students. When promoting programming learning in these contexts of scarce internet and digital infrastructures, it is important to use tools that take into account the constraints of potentially lacking internet connection and the availability of less performant devices.

The Interactive Code Playgrounds (ICP) are a tool that provides an alternative slideshow system, with slides containing code editors, while taking into account many of the possible limitations coming from contexts of scarcity. The aim of this dissertation is to take ICP a step further, by enabling its distribution to Android mobile devices. This is needed because ICP slides have to be to be served through an HTTP server, which has to be made easy for students new to programming, and they can not be seen well on mobile screens. This distribution on mobile devices is an important step as there is quite a large percentage of mobile-only users in the Global South thanks to the affordability of such devices. If ICP were to be available on mobile devices, universities could supply more affordable devices other than computers to their students and many more people would already have their own device that could be used to follow programming courses.

1.2 Development

During the development process, I produced two versions of the app. The first version is mostly based on GeckoView, a WebView-like dependency that works like Android's default WebView but it is based on Firefox instead of Android's native browser. This choice is important because some of the programming languages used in ICP need the browser to support SharedWorkers, which on Android mobile devices can only be used on Firefox. This version is quite heavy both in terms of data storage used (around 400MB) and execution times.

The second and final architecture is a modular app that allows the user to set it up to work only for the needed programming languages and it takes advantage of the browsers already present on the device, greatly decreasing data storage used (around 30MB). In order to do so, the user manually imports the slides and the support of the languages needed for those slides, and everything is then hosted through a local HTTP server. Moreover, this version of the app is programmed to target old, low-end devices, working on devices down to Android 5.

Finally, part of the work also consisted on adapting ICP to mobile screens. In order to do so, I modified the Svelte files of the Editors and the main CSS files of the project, making all the buttons and text visible on smaller screens.

1.3 Key Findings

The app was tested on several Android devices, and particularly on the lowest-end one that was available to me: a Huawei Mediapad T3 7, running on Android 7 and with only 1GB RAM. Tests on this device revealed that C++ can not be run with less than 2GB RAM, as the device is not able to both keep the local HTTP server open and execute the code. At the same time these tests showed overall acceptable performance (the execution times will not exceed 20 seconds, meanwhile initialization times can take up to about 100 second). In particular, a use case that simulates the use of the app to follow a programming course showed that the app uses 17,5% of the internet data that would be used in the best scenario that was possible before this intervention, with the original ICP project.

In addition, these tests revealed two bugs. I was not able to identify the exact reason, but they only happen on Firefox mobile browser and are possibly caused by some incompatibilities between the dependencies of the original ICP project and Firefox mobile. Although I was not able to fix them, they only are minor nuisances that do not compromise the use of ICP slides.

1.4 Final considerations

What has been defined as scarcity is a complex situation: it is not only about the lack of powerful devices, but it can also affect the technological capabilities of the people that live in those situations or simply their interest towards technology. For this reason, future works should mainly focus on testing the use of ICP slides in real programming courses held in contexts of scarcity. Despite the described limitations, this project already shows good performances on low-end devices with no internet connection and with time it can become a useful pedagogical tool to help students from all around the world approach programming learning.

2 Introduction

New and better technologies are developed each year, in an endless chase to continuous upgrades and improvements. As a matter of fact, nowadays everything becomes outdated pretty fast and staying on the edge of technology becomes an impressive task, but what happens to those who do not have the means or the will to keep up? Usually, they are left behind with no effort from newer technologies to support older hardwares. In this dissertation the focus will be on developing a technology used in programming learning that is able to include as many people and devices as possible by following one of the key principles of "Computing Within Limits" [12]: to consider models of scarcity.

2.1 Background Information

The aforementioned "Computing Within Limits" is a community that approaches programming by taking into account planetary limits, as a continuous growth of the IT systems without any side effects can not be taken for granted. Conversely, resources are limited and we are getting closer to a point of failure. By following this approach to programming, not only it is possible to decrease the burden on our planet, but as a result we also get more accessible programs, even in contexts of scarcity.

Before introducing scarcity, let's take into account the definition of Digital Divide. UN-Habitat defines the Digital Divide as the gap between those who have access to ICTs and those who do not [19]. This gap can be in several fields, going from infrastructures and internet access to culture and economy. What has just been given is a definition coming from the policy-making world, but we can go more into details. There are several orders of Digital Divide: the **first-order** consists in those who have access to internet and digital technologies and those who do not, the **second-order** considers the social inequalities that cause such disparity, the **third-order** regards the application of digital technologies in the fields of education, culture and healthcare [15]. In this dissertation the focus will be on the lack of access to internet and digital technologies (first-order) and their use in education (third-order). We will define such contexts as contexts of scarcity, without failing to address the fact that such situations can not only be deemed typical of the Global South, but also happen in rural areas of the Global North and could actually become more widespread in the future instead of decreasing. If we were to try to quantify this gap, 1/3 of the global population lacks access to the internet, going as low as the 40% in Africa [21]. Unfortunately, this is not only limited to low-income families, as even universities struggle to provide "appropriate computer and connectivity infrastructure to the whole of the university community" [16]. At the same time, those same community members do not usually have a computer or internet connection at home, so are forced to rely on their university, resulting in a vicious cycle. This lack of access to digital devices also implies a lack of digital skills. For this reason, even if new devices were to simply be introduced, it is also important how and to what devices people who lack these skills are approached with. A perfect device would be a mobile phone without internet connection, being the one with the lowest level of digital skills demanded for its use [9].

As an example of how easy to use mobile devices are, even though the lack of computers and access to the internet, a significant percentage of mobile-only users is arising. For example in Thailand we have a 60% higher adoption rate of mobile phones compared to computers, and this happens both in urban and rural areas [10]. Similar data comes from Chile, where 21% of the population only accesses the internet through mobile phones [3]. From such information we can understand that mobile devices are much more widespread compared to computers, mostly thanks to their affordability, and could be one of the best means to include people from places that suffer from scarce internet and digital infrastructures, even in rural areas. As an example, such devices allowed the birth of "Internet Plus Agriculture" in rural China, helping farmers improve their efficiency [1]. Even without an internet connection, the possibility to learn how to use mobile devices can be exploited to make education much more accessible to all people [4].

Considering how essential electronic devices are in today's world, the development of digital skills, including programming skills, is of great importance. For this reason, there have been many initiatives

to improve those skills all over the world. For example we have the "Youth Coding Initiative"[1] from UNESCO to improve the programming skills of both teachers and students in Africa and Asia. In addition to this initiatives, programming courses are also held in universities. Several articles discussing the difficulties met by the students during these courses state their interest as a major problem [5, 14]. This is mostly due to the approach used by the teacher, who may focus on theory during class without any interactive, hands-on activity with the students. As a matter of fact, the most used educational technology still consists in slideshows that are explained to the students [17], causing poor interest from their part. An article that reviewed all the teaching approaches to programming in the Global South assessed in fact that the best approach is one that combines in-class theory and practice (defined "blended" in this article) [7]. Moreover, the same article also states that the most efficient and less expensive approach to have interactive lessons with hands-on activities is to incorporate learning with the use of mobile devices. Problems do not end here, as practice classes also have their difficulties: students new to programming have to use an IDE and, if their university does not provide a computer with an IDE already installed, they also have to download and set it up. All this leads to a number of issues, such as the difficult installation and use of an IDE and students having different configurations, possibly causing incompatibility problems [8, 2].

2.2 State of the Art

In the previous section, I described some of the problems met by the students who try to learn programming languages, which are: the difficulty to learn how to use an IDE, the lack of standardization and the approach used in-class by the teachers. Moreover, these problems increase in contexts of scarcity as neither universities nor students might be able to afford computers, or they could be older, less performant models. At the moment, there are three main approaches that solve part of these problems: the use of Virtual Machines, Docker Containers or In-browser Environments. What follows is a brief summary of what they offer and their problems.

Virtual Machines

A Virtual Machine (VM) is an isolated environment created from a pool of hardware resources of the host machine. Their use allows to offer students a perfectly standardized learning environment and it is even possible to grant students admin privileges on the VM, which can be helpful in some courses. At the same time, they have a complex installation, can be really slow and students new to programming have to learn how to use both a VM and an IDE.

Docker Containers

A Docker container offers a standardized environment just like VMs while still using the Operating System of the host machine, allowing for better performances. The same drawbacks are still there, as their installation can become even more complex than VMs and the difficulties of using an IDE aren't solved.

In-browser Environments

The diffusion of internet led to the birth of online code playgrounds which work just like an IDE, but are much more simple to use. In a single, standardized and platform-independent environment they provide access to different programming languages and they can simply be accessed through a web browser [18]. A notable example is Replit[2], an online platform for software development. The use of such a tool allows to solve all the mentioned problems, as it also allows to code using something simpler than an IDE, but it still has a major flaw: it needs an internet connection. Since this dissertation takes into consideration contexts of scarce internet and digital infrastructures, this is not a requirement that can be underestimated.

[1] Available at https://iite.unesco.org/news/unesco-and-codemao-united-to-strengthen-coding-skills-of-students-and-teachers-in-africa-and-asia/, last accessed on 01/07/2024

[2] Available at `https://replit.com/`, last accessed on 02/08/2024

Considering all the benefits and drawbacks of the previous solutions, the one that performs better is an In-browser Environment as not only it provides a standardized alternative to IDEs, but it's less resource-intensive compared to VMs. In the next section, I will describe the Interactive Code Playgrounds, an In-browser Environment that aims to be accessible and inclusive, representing the ideal tool for these situations.

2.3 Interactive Code Playgrounds

The Interactive Code Playgrounds [11] (ICP) are a HTML slideshow system that can be used in introductory courses to programming languages. ICP offers: **1.** a standardized and platform-independent environment, **2.** an easy access through web browsers, not requiring any additional downloads, **3.** code editors are put into slides, offering a familiar environment to both teachers and students, **4.** the possibility to be used entirely offline, **5.** several different languages to choose from. Such characteristics also make ICP an accessible and inclusive technology compared to the previous solutions.

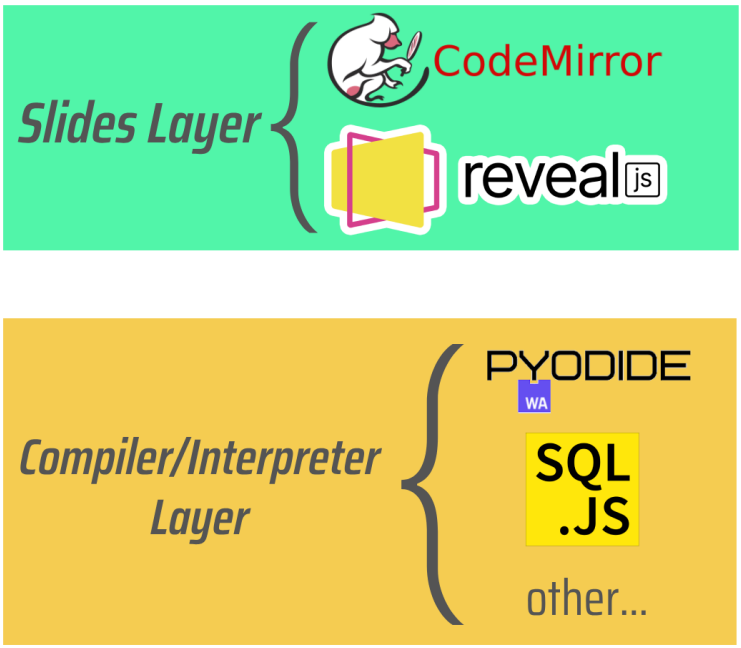

Figure 1: Architecture of the Interactive Code Playground

Going a bit into details, Figure 1 gives an idea of the architecture of ICP. The Slides Layer contains Reveal.js[3] (the presentation framework) and Codemirror[4] (the code editors). In this layer the user can view the slides and write code. Once the code is executed, it is taken as a string and passed to the next layer. Here there are all the various compilers and interpreters (from here onwards, they will be defined as "support for the languages") which are needed to convert the string of code into WebAssembly (WASM), which can then be directly executed by the browser. What is put into this layer can be chosen when creating a set of ICP slides. In fact, this layer has a modular structure that allows to put in it only the compiler and interpreters needed for the programming languages used in the slides. This structure allows to reduce both the data storage used by the slides and the data traffic needed to reproduce the slides if they are used online.

An important theme that has been discussed is the lack of digital skills in contexts of scarcity. The Interactive Code Playgrounds are a pedagogical technology which does not aim to completely reshape

[3]Available at `https://revealjs.com/`, last accessed on 02/08/2024
[4]Available at `https://codemirror.net/`, last accessed on 02/08/2024

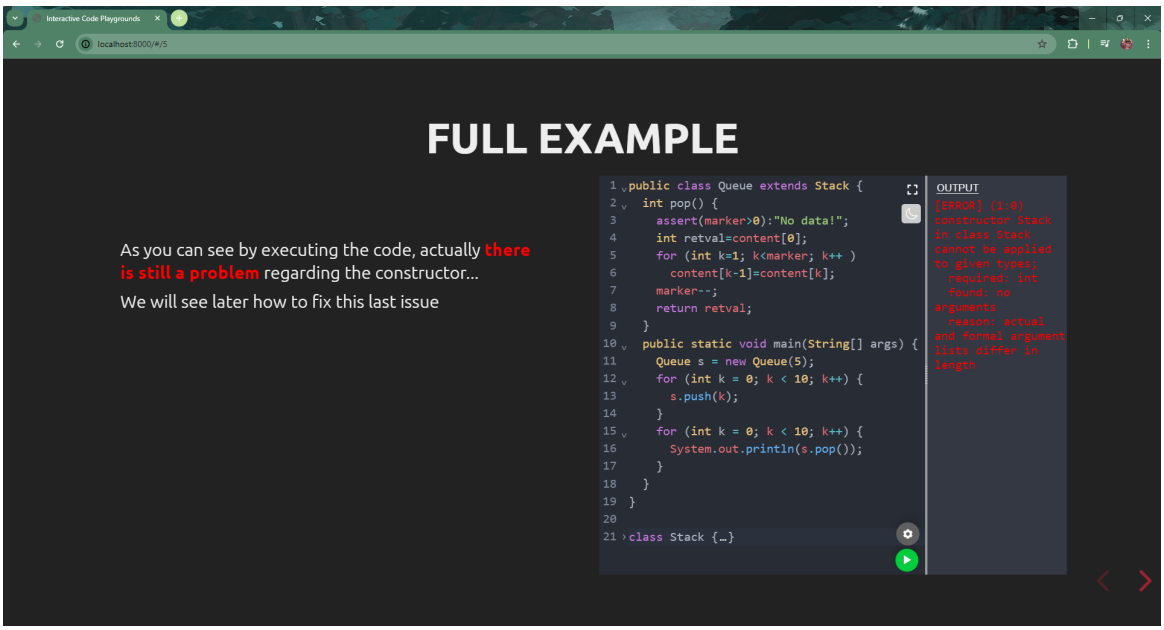

Figure 2: Example of ICP slides in a browser

teaching methodologies. Conversely, it looks and works just like the widely used slideshows which are already familiar to both students and teachers, while still adding some extra features. For this reason the original authors of ICP defined it an incremental technology. Such core ideas of ICP are what inspired the goal of this dissertation.

2.4 Problem Statement and Goal

As previously stated, ICP slides are accessed through a browser. This can happen in several ways on computers:

1. The slides are hosted on the internet and accessed through a link, requiring an internet connection
2. The slides are locally hosted on the machine and accessed offline
3. The slides are exported as a redbean file[5], a single executable file that takes care of both containing the slides and its dependencies and locally hosting it, allowing to access them offline

However, Section 2.1 expressed how much mobile devices also are an important tool for social inclusion. If we were to use the slides on a mobile device, we would meet some difficulties because:

1. Slides hosted online can be accessed through a mobile browser, but some contents might not be visible due to the different size of mobile screens
2. It is possible to locally host the slides on a mobile device, but it has the same problem of hidden contents
3. A redbean file does not work on mobile devices

The difficulties to properly reproduce a set of ICP slides on mobile devices led me to propose the development of an Android app in order to solve these problems. Due to time constraints, it was only possible to produce an Android app or an iOS one. Since Android devices are much more widespread, I chose to focus on them. This led to the development of an app which works as a kind of media player: it allows to simply import the set of slides and the zip containing the support for the languages used in the slides from local storage and then open them in a browser. Following the principles of

[5]Available at: `https://redbean.dev/`, last accessed on 02/07/2024

ICP, the app allows to reproduce the slides without an internet connection and the storage space it takes is minimal. It also targets older devices, having been programmed to support down to Android 5 (released at the end of 2014).

To sum up, the goal of the app is to better cover a part of the devices that can reproduce ICP slides, the mobile devices, while also taking into account the limitations of the often ignored older, less performant ones. This kind of approach allows to take advantage of the affordability and availability of such devices when compared to computers, thus allowing students to approach programming much more easily even in contexts of scarce internet and digital infrastructures.

3 Development

This section will describe the different stages of development of the app, the difficulties faced and the intervention on the original ICP project.

3.1 Preliminary analysis

The first step of my work consisted in understanding how to make the ICP slides work on mobile devices. In order to do so, I made a series of tests aiming to verify its correct functioning using different approaches.

All these preliminary tests were performed using a OnePlus Nord 2T and consisted in trying to correctly show the *example.html* slides that can be found in the icp-bundle[6] after building a language. In order to properly show the slides, the *.html* file was placed inside a folder consisting in a copy of the *dist* folder of the original ICP project, which contains all the needed imports (specifically, the style files, Reveal.js[7] script and the support for the languages used in the slides). Results can be summarized in the following table:

	Website	Text	Reveal.js	Images	Editors	WebWorkers	SharedWorkers
1. Computer + WebServer	OK	OK	OK	OK	OK	OK	OK
2. Android - HTML opened through Chrome	NO	NO	NO	NO	NO	NO	NO
3. Android - HTML opened through HTML Viewer	NO	NO	NO	NO	NO	NO	NO
4. Android - HttpServer + Chrome	OK	OK	OK	OK	OK	OK	NO
5. Android - HttpServer + Firefox	OK	OK	OK	OK	OK	OK	OK
6. Android - Cordova App	OK	OK	OK	OK	OK	OK	NO
7. Android - HttpServer + GeckoView App	OK	OK	OK	OK	OK	OK	OK
8. Android - GeckoView App	OK	OK	OK	OK	OK	OK	OK

Figure 3: Table summarizing results of the tests

Going into details:

- Point 1: The original ICP project working on computer.
- Points 2, 3: The *.html* file opened using Chrome or an HTML Viewer.
- Points 4, 5: The app "Simple HTTP Server"[8] was used to locally host a folder containing the *.html* file of the slides and all the needed imports, the local address was then opened using different browsers.
- Point 6: Apache Cordova[9] was used to build the previous folder into an app which shows the website in an embedded browser (WebView).
- Points 7, 8: An app using GeckoView[10] as a WebView instead of the system's default one. In point 7 the embedded browser opened the local address hosted by another app, while in point 8 the app hosted it itself.

The first thing to notice is that it is not possible to simply open the *.html* slides, as mobile browsers can't directly access the local storage of the device in order to use the components imported by the *.html*, not even layout files like css. The slides were correctly displayed only if served through a local http server.

Another important discovery is that SharedWorkers did not seem to work on every mobile browser, but they are needed because languages such as C++, Java and Python have a large environment that has to be initialized in every editor that uses them. SharedWorkers allow to use the same environment

[6] Available at: `https://github.com/lucademenego99/icp-bundle`, last accessed on 22/05/2024
[7] Available at: `https://revealjs.com/`, last accessed on 22/05/2024
[8] Available at: `https://play.google.com/store/apps/details?id=com.phlox.simpleserver`, last accessed on 22/05/2024
[9] Available at: `https://cordova.apache.org/`, last accessed on 22/05/2024
[10] Available at: `https://wiki.mozilla.org/Mobile/GeckoView`, last accessed on 22/05/2024

for all of them, greatly reducing initialization times. Upon further research, I found out that on mobile only Safari and Firefox support SharedWorkers. Given these results, I decided to further develop the app used for testing in point 8.

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
	✓	✓	✓	✓	✓	✗	✓	✗	✓	✗	✗
`SharedWorker`	5	79	29	10.6	16 …	No *	33	11–12.1	16 …	4.0–4.2	No *

Figure 4: Table describing browser support for SharedWorkers. Image Source: mdn web docs. Accessed via `https://developer.mozilla.org/en-US/docs/Web/API/SharedWorker`, last accessed on 22/05/2024

3.2 First version

The first version of the app took care of both hosting the slides and showing them in an embedded browser. Every android device natively has a system WebView, a view which can be called inside an app's activity in order to show a web page. The main problem is that this WebView is based on Chrome and using it would have meant indefinitely locking languages that need SharedWorkers out of the app. In fact, although requests for Chrome Mobile to support SharedWorkers date back to 2012, they still have not been satisfied and deemed not of immediate importance. To solve the problem, I decided to use GeckoView, a WebView-like dependency that works like the system's default one but it is based on Firefox.

In order to work, the ICP need 3 components: the **slides**, the support for the **languages** and all the files used for the **style** (going from css to the script of reveal.js). The app contained in its assets folder both last 2 components, style and languages. Once the app was launched, it requested for permissions to manage the external storage and when granted it extracted the contents of the assets folder in the local storage of the device. The user then had to manually copy the slides inside that same folder and rename the file as *index.html*.

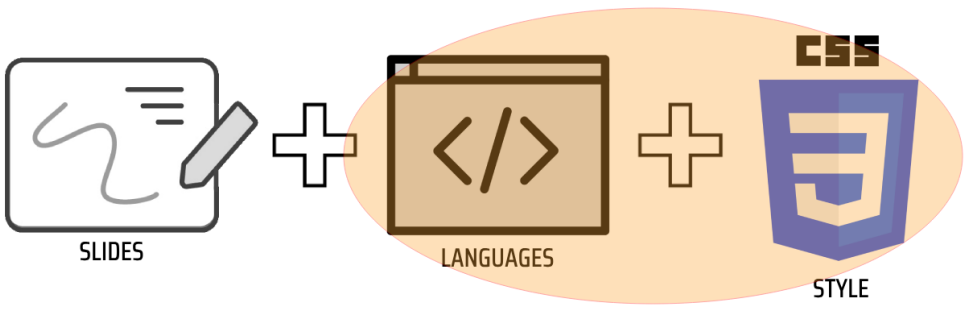

Figure 5: Contents provided by the first version

The original ICP project allows to only build the support for specific languages instead of building the whole package. Since the app had to work with all of them, I put the full bundle inside the assets

folder. Once everything was extracted into local storage, the target folder had a size of approximately 90MB. Moreover, since the app did not use the native WebView of the system, the app itself reached the size of circa 250MB. Although fully functional, the app did not comply with one of the goals of the project: it had to be as slim as possible, meanwhile the result was well over three hundreds of megabytes.

3.3 Final architecture

The second and final version of the ICP app aims to slim down the project, now working as a kind of media player for ICP slides. The app has a fully modular structure, only providing the style content inside its assets folder and not using GeckoView anymore. This version takes instead advantage of browsers already available on a mobile device, allowing the users to freely choose which one to use and to only import the support for the languages actually needed for the slides.

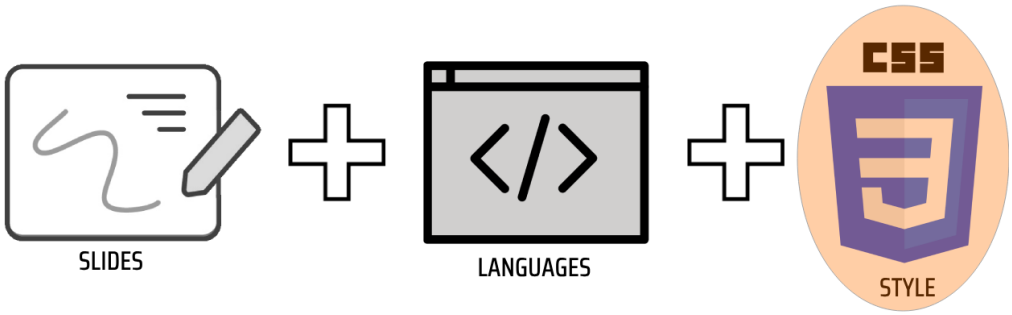

Figure 6: Contents provided by the final version

The app is structured into three main activities:

- **Import Slides**: used for importing slides from local storage.
- **Import Languages**: used for importing the support for the languages from local storage.
- **Show Slides**: used for starting the local http server and open the browser.

Similar to the previous version, once opened the app requests for permissions to manage the external storage and to send notifications. Upon receiving the needed permissions, it unpacks the contents of the assets folder into the storage of the device. Since the folder only contains the style component, it has the size of only 200KB while the app of circa 20MB. Following the creation of the folder in the storage of the device, it is possible to import the slides and languages components. In order to do so, the *ActivityResultContract.GetContent()* contract is used, which allows to visually select a file from the storage of the device and return its path. Since another goal of the app is to be accessible from as many devices as possible, it targets down to Android 5, but when this contract is used with Android versions lower than 8 paths are returned in a different way. Therefore, the two cases are handled differently.

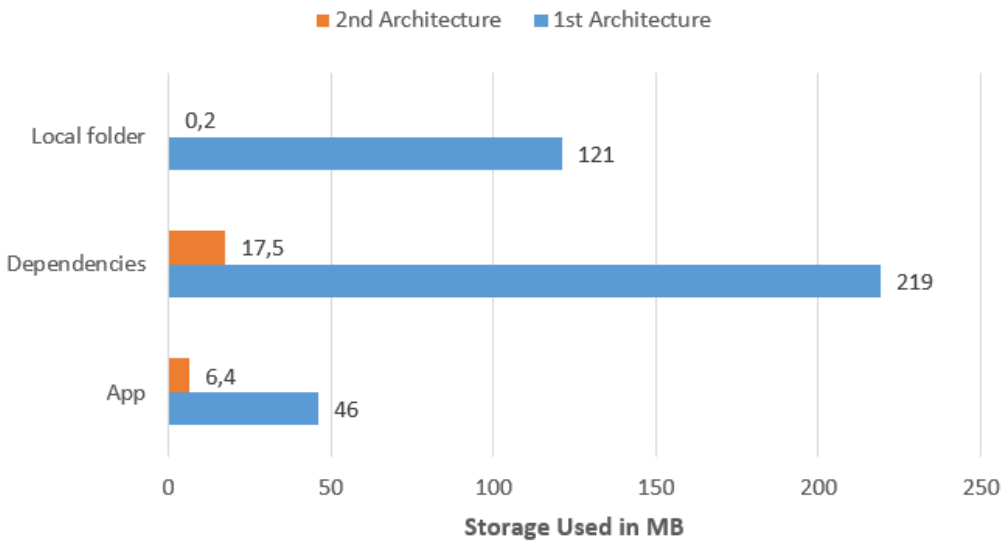

Figure 7: Storage used in detail by the different parts of the two versions of the app

In order to avoid problems when serving the contents through the local http server, the app allows to move from *Import Slides* to *Import Languages* only once an *.html* file has been imported. The next activity works just like the previous one, but is instead used to import the languages component. This activity also allows to move to the next one only when all the languages needed in the set of slides previously chosen have been imported.

Finally, the *Show Slides* activity allows to start the local http server. This is done inside a Service, an activity which runs in the foreground and does not terminate when leaving the app. This allows to keep serving the folder once in the browser. When the Service is running, a notification is shown allowing the user to stop the http server. Moreover, the user also has the possibility to choose between opening the local address on the default browser of the system or paste and copy it in the desired browser.

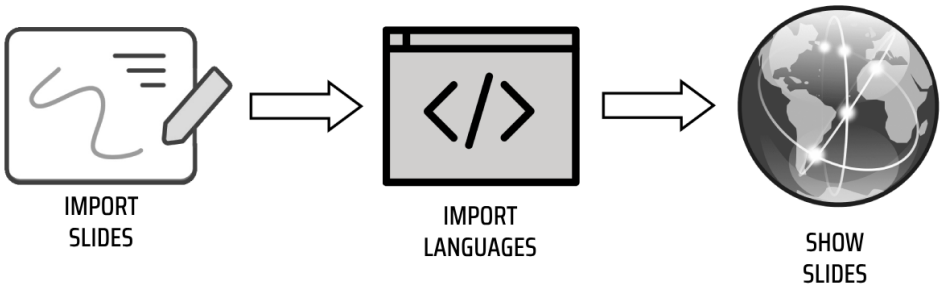

Figure 8: The flow of the app

3.4 UI

The main focus of the UI of the app is to guide the user through its use in a simple way. The interface is minimal, only showing the essential information. In the top bar there are the info and settings buttons. The info section didactically explains how to use the app, while the settings allow to manage the permissions and the folder in the storage of the device, change the theme of the app and the text size of the editors in the slides when they are shown in the browser.

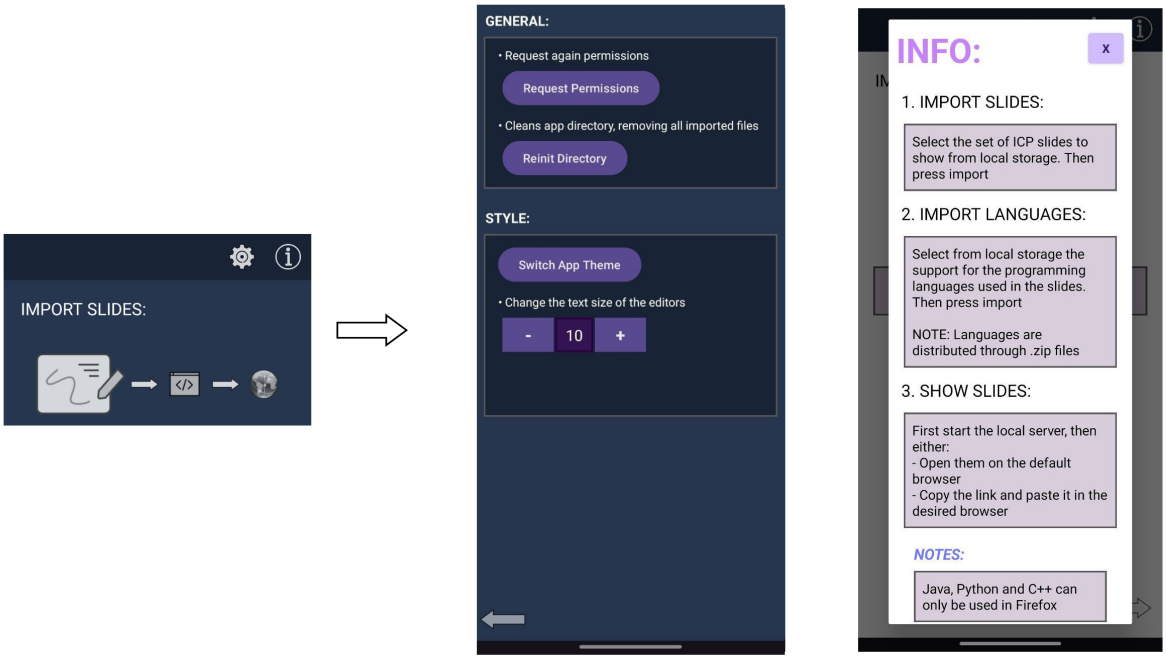

Figure 9: On the left the top bar and the visual guide of the steps, on the right the settings menu and the info pop-up (using light theme)

The first activity, *Import Slides*, keeps track of the last imported slides in order to help the user remember which slides would currently be shown and has a greyed out *Import* button which becomes purple once a set of slides has been selected to be imported, visually informing the user that the button can now be pressed.

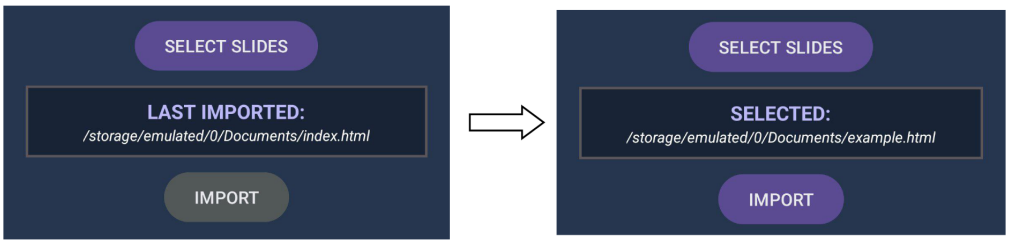

Figure 10: On the left, no HTML has been selected and the last imported one is shown. On the right, an HTML has been selected and it can be imported

The following activity is *Import Languages*. This one works just like the previous, but it also shows the languages that are used in the slides and if they are already available.

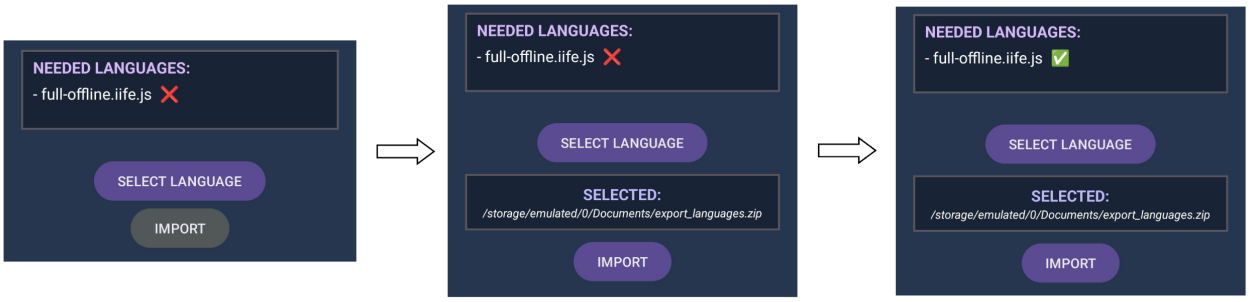

Figure 11: From left to right: needed languages not available and no language has been selected, a language has been selected, the selected language has been imported and it corresponds to the needed one

Finally, the *Show Slides* activity only has a button to start the local http server. Once pressed, two more buttons appear: one to open the default browser and one to copy the link to the local address.

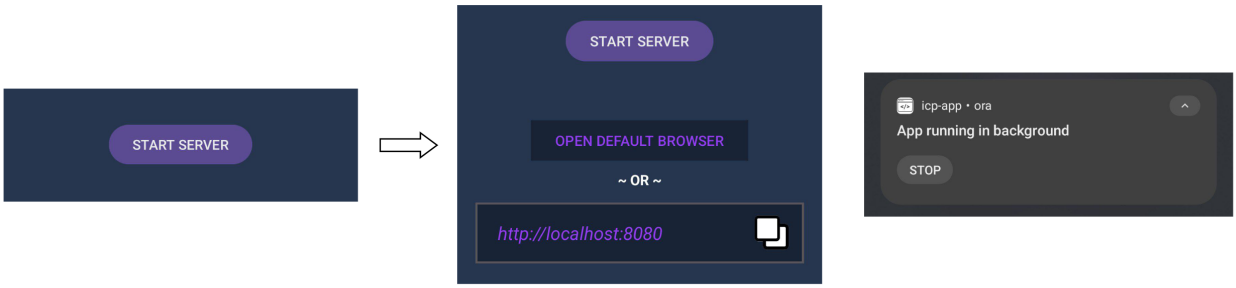

Figure 12: On the left the *Start Server* button before being pressed, on the right the buttons and the notification that appear once it is pressed

3.5 Interventions on original project

My interventions on the original ICP project are of two kinds: on the style and on the bundling. Starting from the style, I modified the Svelte files of the Editor and its buttons and the main css files. They now recognize when the screen showing the slides is a small screen (thus including both phones and tablets) and scale text and buttons accordingly. A margin below the editors was added because vertical editors couldn't fit in a mobile browser, preventing the run and settings buttons from being pressed.

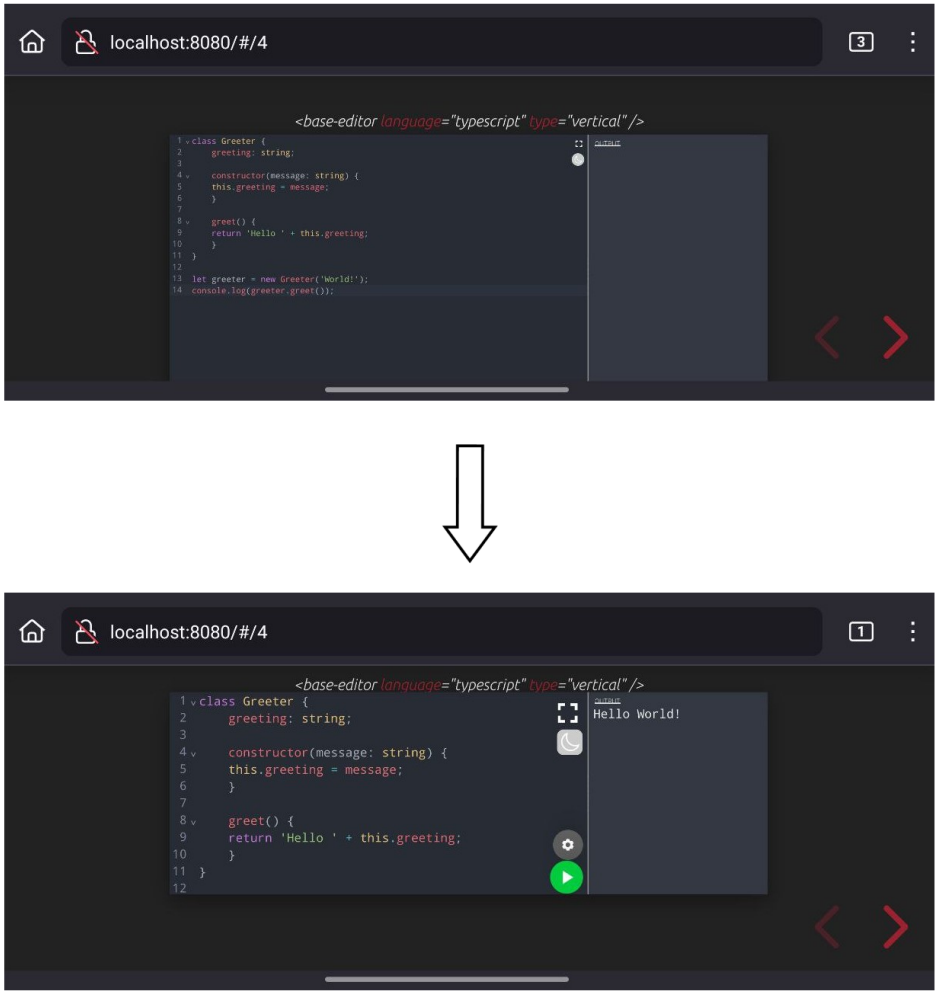

Figure 13: On the top, how ICP slides originally looked on mobile. On the bottom the final version

As for the bundling, the original project incrementally built the supports for the languages. This can make the size of the project become bigger than necessary. It is now possible to choose if building for a language erases the previously built languages and it's also possible to add the flag *–mobile* to create a zip folder containing the built language component which can easily be transferred to mobile devices.

4 Results

This section consists in a discussion about how the final app behaves on the target devices and the limitations of the project.

4.1 Performances

The final app has been tested on three devices:

- OnePlus Nord 2t, Android 14, 8GB RAM
- Huawei Nova Young, Android 6, 2GB RAM
- Huawei Mediapad T3 7, Android 7, 1GB RAM

Since one of the objectives of the app is to able to reproduce the ICP slides on as many devices as possible, I will now take into consideration how the different languages work on the lowest-end device of the three: the Huawei Mediapad T3 7. The heavier languages (Java, Python and C++) were tested using as language component both the support for that specific language and the full bundle which supports all languages. The browser used for the tests is Firefox (126.0.1, as of 05/06/2024), being the only one supporting SharedWorkers.

Programming Language	**Initialization (s)**	**Code Execution (s)**
Java & java-offline	100	18.5
Java & full-offline	105	19
Python & python-offline	41	Immediate
Python & full-offline	46.5	Immediate
C++ & cpp-offline	[Hangs]	[Hangs]
C++ & full-offline	[Hangs]	[Hangs]
Standard ML	Immediate	9
Typescript	Immediate	1.7
SQL	Immediate	8
Processing, P5, Javascript	Immediate	Immediate

Table 1: Performance of ICP on a Huawei Mediapad T3 7

From Table 1 we can immediately notice that using the support for only a specific language or the full bundle doesn't have notable effects on the performances, but only on the storage space as installing the whole bundle needs more space than the support for specific languages. All languages worked on the low-end device except for C++, which worked instead on the Huawei Nova Young. This difference suggests that 2GB RAM are a reasonable constraint to be able to run C++ on a mobile device.

Programming Language	**Initialization (s)**	**Code Execution (s)**
Java	59	15.5
Python	59	Immediate
C++	209	0.5 to 8
Processing	Immediate	Immediate
P5	Immediate	Immediate
Standard ML	Immediate	4.6
Javascript	Immediate	Immediate
Typescript	Immediate	0.6
SQL	Immediate	0.5

Table 2: Performance of ICP on a PC with 4GB DDR3 RAM and a dual-core AMD E-300 1.3GHz CPU. Table taken fro a previous thesis (namely [11])

Comparing the performances of ICP on a low-end mobile (Table 1) with those on a low-end computer (Table 2), it's possible to see as expected that both the initialization and the execution times of all languages are slower on mobile but still acceptable. Surprisingly, Python is the exception.

4.2 Use Case

In order to test the validity of the app with respect to what could already be done before my intervention, I tried to simulate an use case. I took into account four programming courses from the University of Trento: two Computer Programming 1 courses and two Computer Programming 2 courses. These all are 12 EC introductory courses to programming, akin to the ideal use of ICP.

As an average, these courses have 25 sets of slides, each containing 40 slides. I recreated a slideshow about Inheritance in Java from a Computer Programming 2 course as a set of ICP slides and then used it in three different contexts:

1. Hosting the slides on another machine and opening them on a mobile browser, using cloud-hosting deployment
2. Opening a locally stored HTML file that imports its dependencies through the internet, being the best that could be done on mobile with the original project
3. A single HTML file that uses imports locally served by the app, which is the result of my intervention

To estimate internet data consumption while following one of these courses, I supposed that each set of slides is opened three times: during the lesson, at home to revise the lesson and to prepare for the exam.

For case 1, I used "HFS"[11] to serve the slides and its dependencies over my network. Accessing them from Firefox after clearing its cache and executing every snippet of code used 12,3MB of internet data, which with our assumption gets to a total of circa 950MB of internet data needed to follow the whole course. It is not possible to know if some files will still be in the cache when the slides are accessed the following time, but if they were only 0,81MB of internet data would be used to reproduce them. We could assume that for each set of slides, they are still in the cache when they are reopened at home after the lesson, thus reducing the total data consumption to 635MB.

Case 2 consisted in a single HTML file opened from Firefox after clearing its cache. This method used 7,73MB of internet data plus the 2MB needed to download the slides. This leads to a total data consumption of 630MB. If we were to consider cache, reproducing the slides uses 0,7MB. Using the same assumption as above about cache, the total data consumption can be reduced to 400MB.

Finally in case 3, the app with the offline version of the slides was tested. For this case the app (apk file of 8MB) and the support for the Java language (12MB) need to be downloaded only once. We then have to download all the sets of slides, each one of 2MB. Putting everything together, we get a total data consumption of 70MB.

From the results summarized in Figure 14 it's possible to notice that having someone host the slides (whether a professor or an institution) is both the most convenient and the most expensive approach. Although less data-hungry than hosting, the online version of the slides still needs a significant amount of internet data in order to work. On the other hand, the offline version of the slides that uses the app not only has, in the worst case scenario, an enormously lower data consumption, but it can also be reduced to zero if all the files are physically transferred to the mobile devices.

Naturally, the following question would be if the internet data requirements of the online approaches are sustainable for places with scarce internet and digital infrastructures. The main answers come from a 2021 report produced by the International Telecommunication Union [20]. From this report we understand that "while 76 per cent of the population of LDCs is covered by a mobile broadband signal, only 25 per cent are using the Internet" and for these people who are actually using internet, the median internet data usage was of 2.1GB per month in 2020 (for the countries where this statistic

[11] Available at `https://rejetto.com/hfs/`, last accessed on 07/06/2024

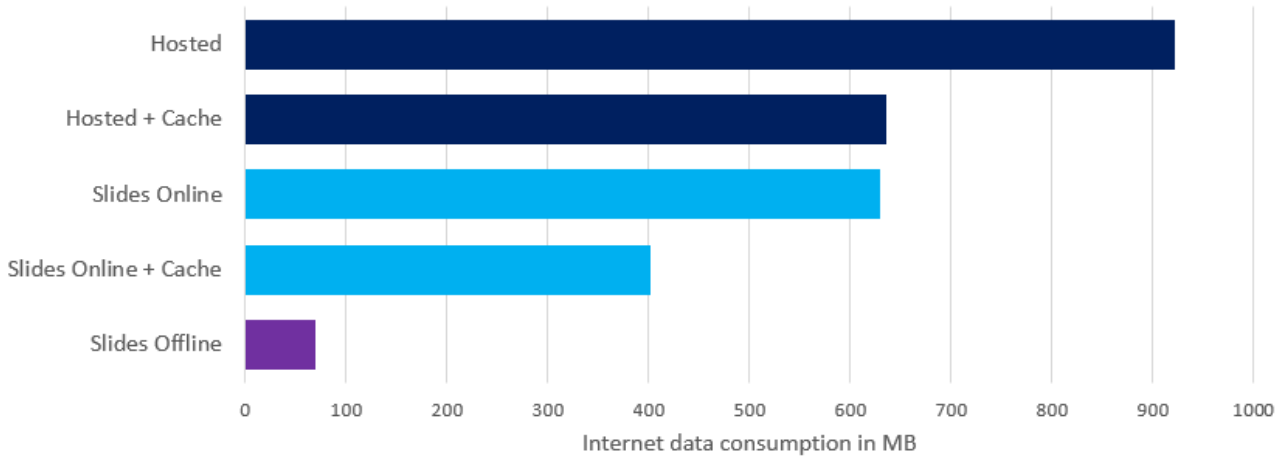

Figure 14: Internet data consumption of ICP slides used in different ways

is available). Although quite vague, this data suggests that the people who are using the internet could be able to use the online or hosted version of the slides, as a 12 EC course usually lasts 4 months and in that span of time they would have circa 8.4GB of internet traffic available. Yet, this would still take up to the 11% of their traffic in that span of time. Moreover, an extremely large percentage of the population of these countries would not be able to use neither of the online versions of the slides (as an interesting example, 40% of nursing students from the Menoufia University, Egypt say they lack internet connection although 90% of them have devices for e-learning activities [6]). The problem would be even greater for institutions, as they would have to provide the slides to many students. From this data, the possibility to reproduce the slides without an internet connection appears to be an impacting feature in the process of social inclusion.

4.3 Limitations

The project faced two kinds of limitations: some incompatibilities with mobile browsers and the uncertainty about the characteristics of mobile devices in places with scarce internet and digital infrastructures.

Starting from the incompatibility problems, the first and most evident one is the lack of support for SharedWorkers: if users want to use some specific languages, they are forced to use Firefox. Regrettably, there seems to be no plans in the near future to implement the support for SharedWorkers in other mobile browsers. For example, there is an issue[12] open on Chromium since 2012 asking for their implementation, but the last answer from 2023 states that this addition is a feature of less impact than others that are currently being worked on, and as such it is not in the plans for the moment.

Since Firefox is needed to have the complete functionality of ICP, I focused my tests on Firefox (126.0.1, as of 05/06/2024). This led to the discovery of two bugs that I wasn't able to fix and which can affect the quality of ICP slides on mobile devices. They are the following:

- Only on Firefox, when trying to delete text from editors whose languages need WebWorkers using Gboard as a keyboard, text is not deleted.
- Only on Firefox, pressing the settings button inside an editor opens the keyboard. This is the only button that causes the issue.

Going into the details of the first bug, I tried to recreate it in a minimal example that resambles ICP: a Svelte file containing a Codemirror 6 editor and a run button that takes the Javascript code inside the editor and spawns a WebWorker which executes it and prints the result through a *console.log()*. Although this minimal example reflects how ICP works, moving it to mobile, locally hosting it through

[12] Available at `https://issues.chromium.org/issues/40290702`, last accessed on 07/06/2024

an http server, opening it on Firefox and trying to delete text in the editor using Gboard worked without a problem. This probably means it comes from some incompatibility problem with Firefox due to some other of the many dependencies used by ICP that I haven't been able to identify. Although I wasn't able to find a solution, I found a workaround: I added inside the distribution folder of the Github repository of the app the *.apk* folder of another keyboard called Codeboard[13]. This keyboard not only allows easier access to symbols used in programming languages, but also allows to succesfully delete text. Moreover, it is lightweight: it only takes 13MB of storage space.

As for the second problem, my idea was that the click went through the button and also pressed the underlying editor, causing the keyboard to appear. The solution for such problems is to stop the propagation of the event inside the function of the button or fix the z-indexes of the editor and the button. Unfortunately, none of these worked. An online research did not provide any more possible solutions and I wasn't able to fix it in any way.

As all these problems only take place in Firefox, they could be caused in some way by the browser. The easier solution would be that they get fixed in the future by Mozilla itself.

The next limitation depends on the characteristics of mobile devices. As discussed in Section 4.1, I realized that mobile devices need to have more than 1GB of RAM in order to be able to use all the supported languages, but what are the average characteristics of mobile devices in the Global South? The main source of information is DeviceAtlas[14], which produced Intelligence Reports from 2017 to 2019. These reports were analyzed to understand the average RAM availability of mobile devices in the following countries: Argentina, Brazil, Colombia, Egypt, India, Malaysia, Nigeria, South Africa.

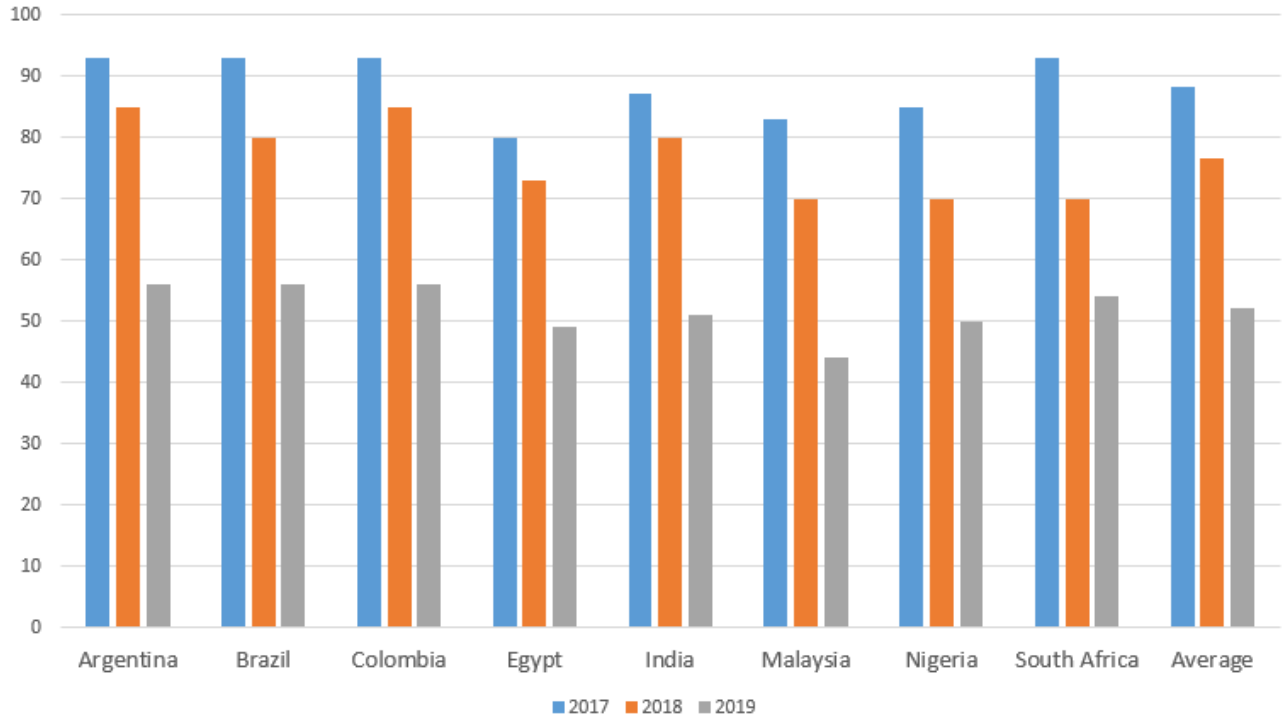

Figure 15: Percentages of devices with RAM $\leq$ 1GB, data taken from DeviceAtlas

From Figure 15 it can be understood that the average percentage of devices with RAM less or equal than 1GB went from 88% in 2017 to 52% in 2019. This trend is corroborated by a 2023 article [13] which states that people from countries in the Global South prefer to buy smartphones with the combination of 3GB RAM / 32GB ROM as they have the most convenient price, so the percentage is expected to have become even lower in the past years.

Although there is no data that explicitly focuses on the current characteristics of mobile devices in

[13] Available at: `https://github.com/codeboardio/codeboard`, last accessed on 25/05/2024
[14] Available at `https://deviceatlas.com/`, last accessed on 06/06/2024

the Global South, we can understand that the situation is improving and at least half of the devices are expected to have more than 1GB of RAM. Nevertheless, waiting for these countries to only use higher-end devices is far from the ideal solution. In fact, this is a problem that affects every country: a person who doesn't have the technical needs or the means to buy a high-end mobile device might buy a cheap smartphone with low RAM even if better devices are available.

For this reason, the best solution would be that Firefox is optimized in order to require less resources to be run or SharedWorkers are implemented in a browser that demands less resources compared to Firefox, thus allowing to decrease the minimal constraint in terms of RAM. On a project level it could be worth researching for better technologies that allow to simplify the translation of code written in other languages to WebAssembly and reduce the number of dependencies used by the main project. Even more broadly, on a systemic level it would be important to restart focusing more on efficient programming.

There currently is a long-standing trend that sees both websites and apps getting heavier and heavier, relying on new and stronger devices to be able to run even less optimized programs[15]. In fact, the current approach can not be sustained in the long run as Moore's Law is flattening and much of the bloat comes from Software Engineering practices that rely on hardware improvements for better program performances [22]. Considering this, a change of the established practices could be extremely beneficial for both the long time health of programs and their ability to work on (sometimes not so) old devices.

[15]Interesting example for websites available at: `https://danluu.com/slow-device/`, last accessed on 14/06/2024

5 Conclusion

This dissertation presented the distribution of Interactive Code Playgrounds on Android mobile devices through a lightweight app that aims to work in contexts of scarce digital and internet infrastructures. In particular, this distribution targets low-end devices without an internet connection and the work consisted in the adaptation of the Interactive Code Playgrounds to the screens of mobile devices, the development of an Android app, and the evaluation of its performances on low-end devices.

This mobile distribution allows Interactive Code Playgrounds to now cover a substantial part of the electronic devices even in contexts of scarce internet and digital infrastructures, further boosting its core idea of social inclusion. Furthermore, there is a consistent percentage of mobile-only users that will now have the possibility to take advantage of ICPs. Mobile devices also are generally more affordable and available than computers, allowing institutions that struggle to provide computers to all their students to use these kind of devices too.

Following the previous work on the original project, the performances of the app have been assessed in section 4 by simulating a context of scarcity through the use of low-end devices. The tests revealed more than acceptable results in terms of initialization/execution times (considering a Huawei Mediapad T3 7, the execution times will not exceed 20 seconds, meanwhile initialization times can take up to about 100 seconds), a great reduction of internet data usage compared to what was possible before this intervention (considering the previous best scenario, the app uses 17,5% of the internet data of the online slides with cache) and it allowed to draw a minimal constraint in terms of needed RAM.

This leads to the first limitation of the project, because there is no explicit data on the characteristics of mobile devices in the target contexts. The only data available suggests that at least half of the devices reach the minimal constraint, but at the moment it is not clear if it can be considered acceptable or too high. At the same time, I found some bugs that impact the usability of the Interactive Code Playgrounds, and which I was not able to fix. It is still unclear if these bugs are caused by the app or an incompatibility of the dependencies of the original project between them, with mobile browsers or a mix of these two, but surprisingly they only seem to manifest on Firefox mobile browser.

Future works should focus on improving the app and on further testing. Possible improvements are about the accessibility, compatibility and minimal constraints of the app, and potentially reducing the amount of dependencies in use. It is also worth noting that Apple mobile devices are not part of the mobile distribution at the moment and should be addressed in the future. As for testing, what has been done up until now is the weakest part of the project because what has been defined as scarcity is a complex situation: it is not only about the lack of powerful devices, but it can also affect the technological capabilities of the people that live in those situations or simply their interest towards technology. From this point on it would be important to test the app in real contexts of scarcity, ranging from local students from low-income families to students from the Global South. Only this step can allow to properly assess the technical performances of the app with the devices available in those real situations, but it will also allow to see those contexts from a human and pedagogic point of view: the interest towards learning programming, the usability of the app and its pedagogical benefits and drawbacks.

In conclusion, this dissertation follows and expands Luca De Menego's Interactive Code Playgrounds [11] by providing an inclusive and resource-aware app for programming education. The results already show good performances on low-end devices with no internet connection, but with future work on optimization, accessibility and testing in real situations I believe the app could constitute a useful tool in allowing people from different backgrounds and countries to approach programming. Finally, this is only a small step towards a more sustainable approach to programming, but it comes with the hope that future works on this and other projects will help redefine the current Software Engineering paradigms by taking into account a resource-constrained world.

References

[1] Asian Development Bank. Internet plus agriculture: A new engine for rural economic growth in the people's republic of china. Technical report, 2018.

[2] Christopher Boroni, Frances Goosey, Michael Grinder, and Rockford Ross. A paradigm shift! the internet, the web, browsers, java and the future of computer science education. *ACM SIGCSE Bulletin*, 30, 08 2002.

[3] Teresa Correa, Isabel Pavez, and Javier Contreras. Digital inclusion through mobile phones?: A comparison between mobile-only and computer users in internet access, skills and use. *Information, Communication & Society*, 23(7):1074–1091, 2020.

[4] Santiago Criollo-C, Sergio Luján-Mora, and Angel Jaramillo-Alcázar. Advantages and disadvantages of m-learning in current education. In *2018 IEEE World Engineering Education Conference (EDUNINE)*, pages 1–6, 2018.

[5] Salihu Dasuki and Ago Quaye. Undergraduate students' failure in programming courses in institutions of higher education in developing countries: A nigerian perspective. *The Electronic Journal of Information Systems in Developing Countries*, 76(1):1–18, 2016.

[6] Gehan Mohamed Abd El-Hamed Diab and Nahid Fouad Elgahsh. E-learning during covid-19 pandemic: Obstacles faced nursing students and its effect on their attitudes while applying it. *American Journal of Nursing Science*, 9(4):295–309, 2020.

[7] Idongesit Eteng, Sylvia Akpotuzor, Solomon O. Akinola, and Iwinosa Agbonlahor. A review on effective approach to teaching computer programming to undergraduates in developing countries. *Scientific African*, 16:e01240, 2022.

[8] Thomas F. Griffin and Zack Jourdan. Educational use cases for virtual machines. In *Proceedings of the 50th Annual Southeast Regional Conference*, ACM-SE '12, page 365–366, New York, NY, USA, 2012. Association for Computing Machinery.

[9] Jeffrey James. Confronting the scarcity of digital skills among the poor in developing countries. *Development Policy Review*, 39(2):324–339, 2021.

[10] Weerachart T. Kilenthong and Patarapan Odton. Access to ict in rural and urban thailand. *Telecommunications Policy*, 38(11):1146–1159, 2014.

[11] Luca De Menego. Interactive code playgrounds: A slides system to make university programming classes more active and inclusive, 2022/2023.

[12] Bonnie Nardi, Bill Tomlinson, Donald J. Patterson, Jay Chen, Daniel Pargman, Barath Raghavan, and Birgit Penzenstadler. Computing within limits. *Commun. ACM*, 61(10):86–93, sep 2018.

[13] Karamoko N'da, Jiaoju Ge, Steven Ji-Fan Ren, and Jia Wang. What matters for international consumers' choice preferences for smartphones: Evidence from a cross-border ecommerce platform. *Plos one*, 18(5):e0285551, 2023.

[14] J Oroma, Herbert Wanga, and Fredrick Ngumbuke. Challenges of teaching and learning computer programming in a developing country: Lessons from tanzania. In *INTED2012 Proceedings*, pages 3820–3826. IATED, 2012.

[15] Zhuangzhuang Peng and Ting Dan. Digital dividend or digital divide? digital economy and urban-rural income inequality in china. *Telecommunications Policy*, 47(9):102616, 2023.

[16] Guillermo Rodríguez-Abitia, Sandra Martínez-Pérez, Maria Soledad Ramirez-Montoya, and Edgar Lopez-Caudana. Digital gap in universities and challenges for quality education: A diagnostic study in mexico and spain. *Sustainability*, 12(21), 2020.

[17] N. Selwyn. The use of computer technology in university teaching and learning: a critical perspective. *Journal of Computer Assisted Learning*, 23(2):83–94, 2007.

[18] Aaron Daniel Snowberger, Semin Kim, and Sunghee Woo. Analysis and application of front-end code playground tools for web programming education. 16(1):11–19, 2024.

[19] UN-Habitat. Assessing the digital divide. Technical report, 2021.

[20] Internetional Telecommunication Union. Connectivity in the least developed countries. Technical report, 2020.

[21] Internetional Telecommunication Union. Measuring digital development: Facts and figures 2022. Technical report, 2022.

[22] Guoqing Xu, Nick Mitchell, Matthew Arnold, Atanas Rountev, and Gary Sevitsky. Software bloat analysis: finding, removing, and preventing performance problems in modern large-scale object-oriented applications. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, page 421–426, New York, NY, USA, 2010. Association for Computing Machinery.