Free Books project

A Flutter technical challenge

David GONZALEZ - 2020 - david.gonzalez.freelance@gmail.com

Purpose of the document

On November 09, I discovered on Twitter that VeryGood Ventures company were hiring a Flutter Engineer. We can show us our Flutter skills by sending source code (even not finished application).

On my side, I have two released Flutter applications, but technically outdated from what I’m capable of.

Through the Free book project, started on November 10th, this document will explain you more about the **methodology** and **technical skills** involved the develop that application.

Who am I

My name is David, **I work since 2010 as a mobile developer** for various companies, creating native Android and iOS applications.

Even if I'm found of technical part of my job, I also focus my expertise on other topics related to the computer science, such as **dev ops** (a little), **architecture**, **Agile Scrum**, **methodologies** and **communication**.

Regarding Flutter, I'm started have a deeper look at it since **march 2020**. Before, I experienced that technology since November 2018 (launch of the first official release).

Objectives

Free Books main concept

My goal was to develop an application that uses an opened API. My searches lead me to https://github.com/public-apis/public-apis, a page listing all public api. There, I found something that interests me: books, through the Google Books API.

First of all, **I needed to ensure that I will be able to produce something to show on 22th November**, deadline to candidate on VeryGood Venture website. In other words, **develop an application under 12 days**, plus 1 day to write that document.

And I wanted to test some other features, **flavors**, to create an Adult and Child versions of the application.

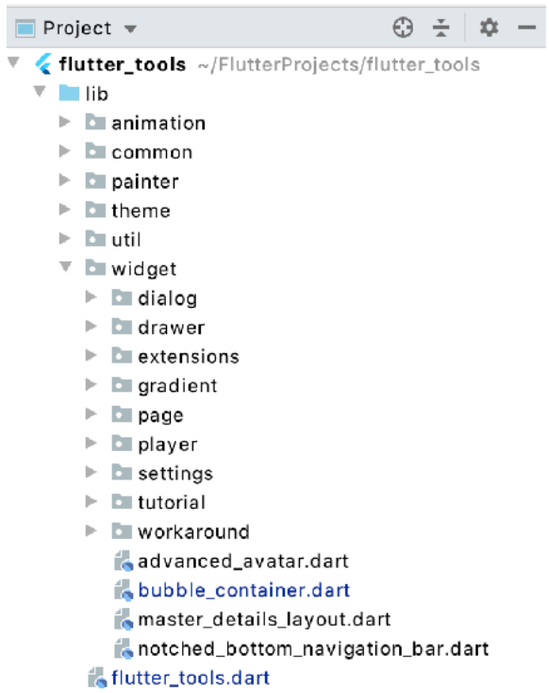
Technical goals

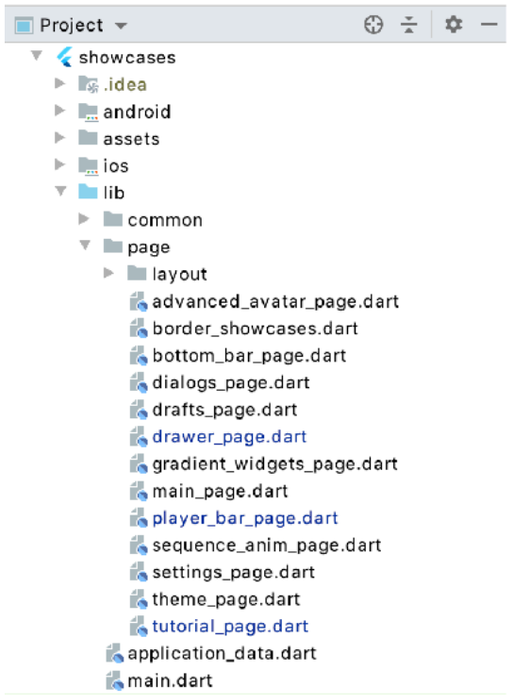
Added to the idea of retrieving and consulting easily books, I challenged myself with technical points to fulfil :

- Usage of **flavors** (also known **build variants**) to provide a sober design version for **adult**s, and a more coloured version for **child**ren. The last one will be more restrictive in searches, to target specific book categories (short novels, tales, bed time stories,…)
- Include **tutorials** in the application, thanks a solution I previously developed (but **still buggy**). I will have to fix it in that application.
- Provide an online/offline mode (implies save data into a local **sqlite database**)
- Provide **specific designs** (and navigation between pages) for android, iOS and Tablets.
- Provide **light** and **dark** modes
- Provide i18n (strings **localisation**)

Technical goals : tools at disposal

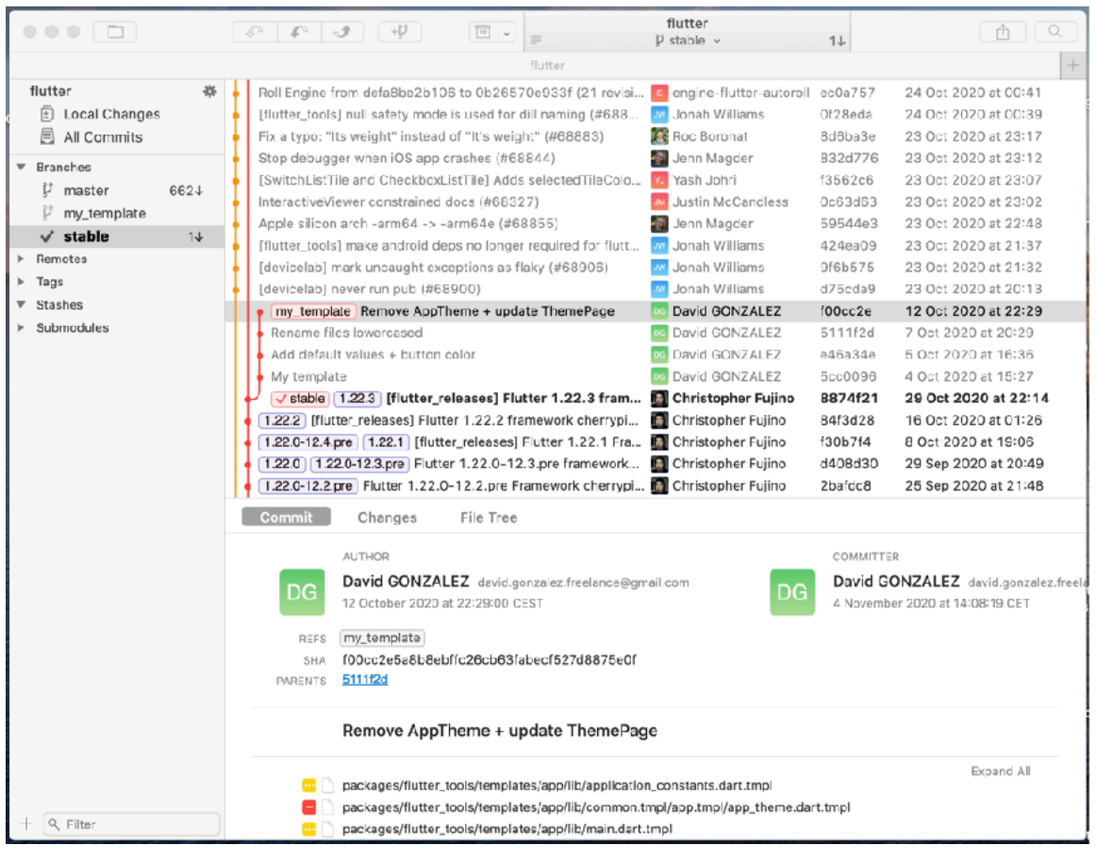
To perform that, I already made a toolbox, my own **private** widget library storing Flutter components involved in application development. I started this library on August 2020, named ***flutter_tools***, and containing another flutter projects "***showcases***", to show how these components are integrated.





Technical goals : tools at disposal

Added to that, **I modified my flutter sdk local version** to add some page templates and my core features in all application I create (through “flutter create” command). I also developed a little script to update properly the stable branch, where my custom code is.



Studies

Proof Of Concepts (POC)

During three days, I developed POC to test different flutter packages, and identify which ones I will use, but most importantly to **identify as quick as possible risks**.

- Usage of flavors, to create different build versions (adult, child)
- Read an epub file, and find a library displaying pages
- Understand and define a solution for using Google Books API

POC : Usages of Flavors

Goals

Ensure that I can do 2 flavors (*child* and *adult*), for both android and iOS. The second goal is to understand in which way flavours library can interact with *flutter_launcher_icons*, a development library generating all sizes of icons.

The most popular library is *flutter_flavorizr*, so I started testing that in my POC. Added to that, I found an example of coexistence between *flutter_flavorizr* and *flutter_launcher_icons*: https://github.com/fluttercommunity/flutter_launcher_icons/tree/master/example/flavors

Results

The library *flutter_flavorizr* allows me to define flavors **in yaml files** (instead of specific config files).

Adding icons using *flutter_launcher_icons* is possible, but I have to manually move generated images and remove some others to make it compile and works.

Interesting time saving, minimal risk.

POC : Handle epub files

Goals

Ensure that I can **read** and **display** an epub file, saved on the device (to let me manage storage).

There is 3 main packages on pub:

- epub: epub file parsing (low level component).
- epub_viewer : seems to be a high level component.
- epub_view : middle level component, having a basic display of epub files.

Results

After having tested the three libraries, my choice is to use *epub_view*, even if **ebook display doesn't reach my expectations** (I wanted an automatic pagination).

- epub_viewer seems great, but have some issues with my epub files. Maybe a version problem, as listed in git issues of that library.
- epub is easy to handle, but I have to develop entirely the display (and pagination) of each pages. **Too expensive for this challenge**.

Identified risk (medium): there is no integrated way to define a book location (bookmark).

POC : Usage of the Google Books API

Goals

Ensure that I can search a book and download it, using the google books api.

- Enable the google book api on my Google console
- Read api documentation and tests requests using Firefox RESTClient plugin (that allow me to execute https requests and check response header, data, …)
- Check that API is accessible using **https** (avoid unsecure calls that could be prevented by the device)

Results

Concerning searches, everything works as expected: I can filter on free ebooks, pagination in requests let me load data step by step, and results contains a link to cover image (thumbnail).

Concerning the download of a epub file, a redirection to a **reCaptcha** page before the download is always done, even by using the API with my developer API key.

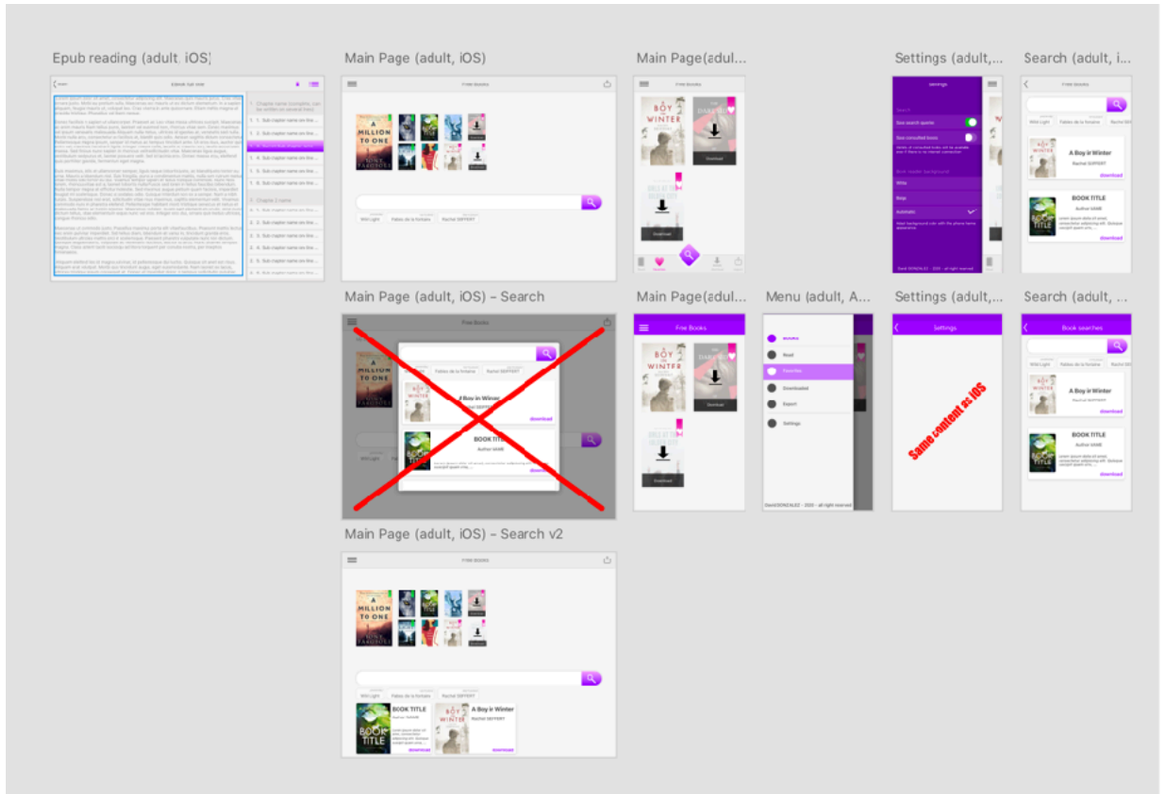
So I have to encapsulate that extra step into my application, which will degrade the user experience.

Design

Application design

All the design has been made through **Adobe XD**.

Due to the lack of time, **I focused on adult application variant**, and especially iOS phone.



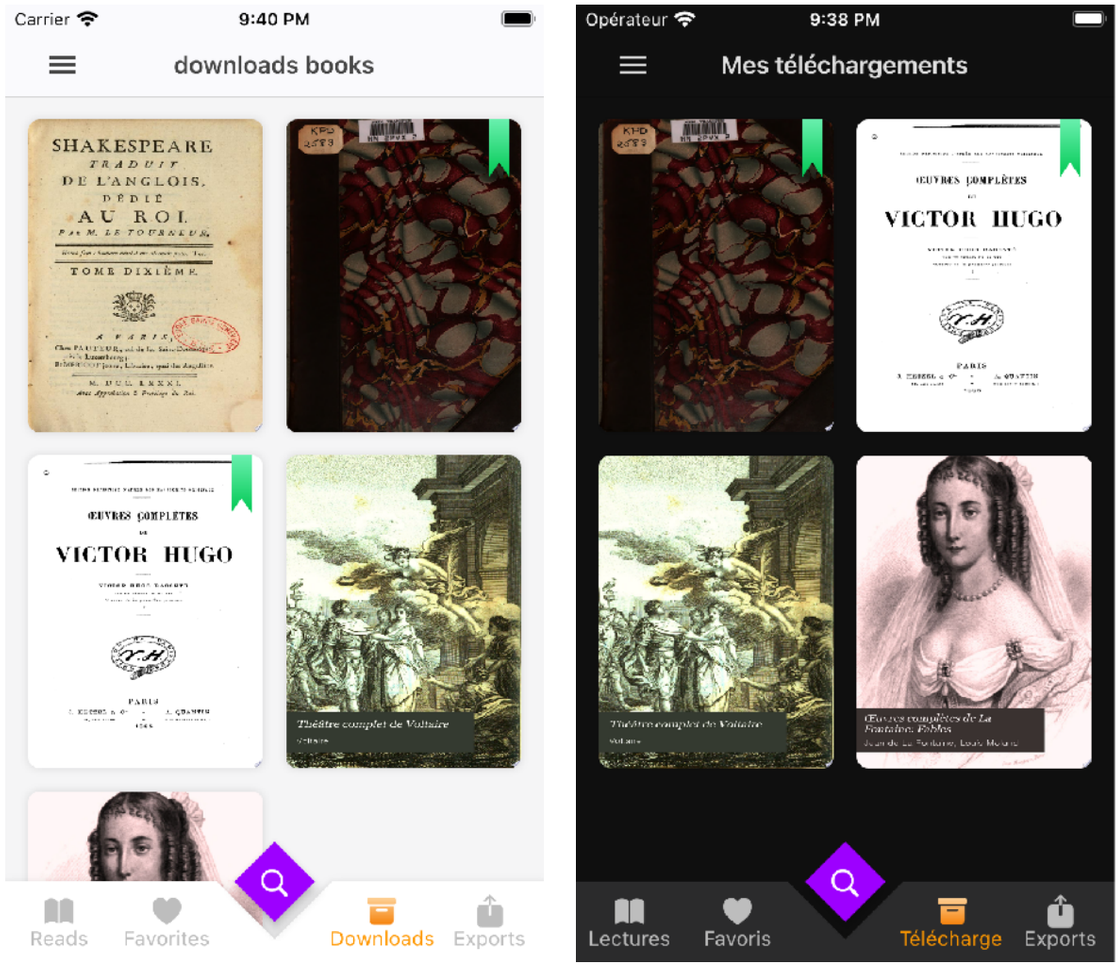
Applications icons have been made using Adobe XD too.

Code

Application development

general features

The application, for each support, handle automatically **light** and **dark** mode, and text **localisation** (in French and English)



Left: iOS, light mode in English
Right: iOS, dark mode in French

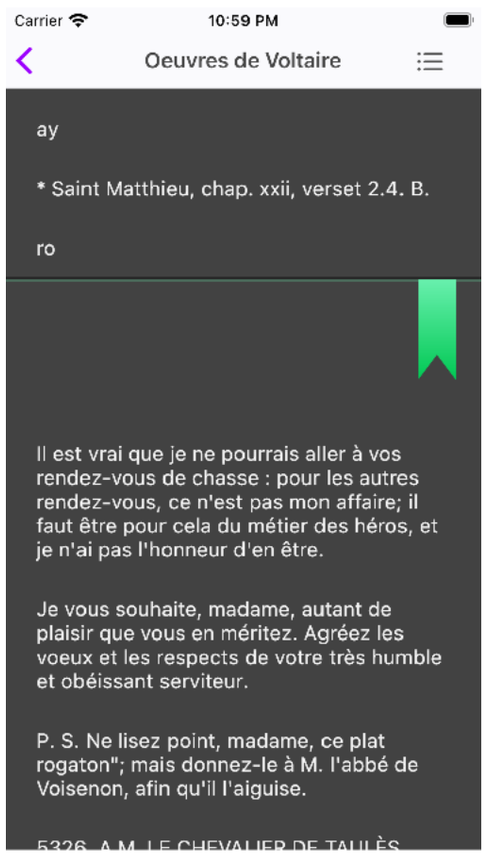
Application presentation

general feature : epub reader page

All version share the **same epub reader page**. *This feature were not intended like this*.

In a future version, chapters dialog design and tablet navigation will changes.

At this time, user experience is bad in that page. **A rework have to be done**.



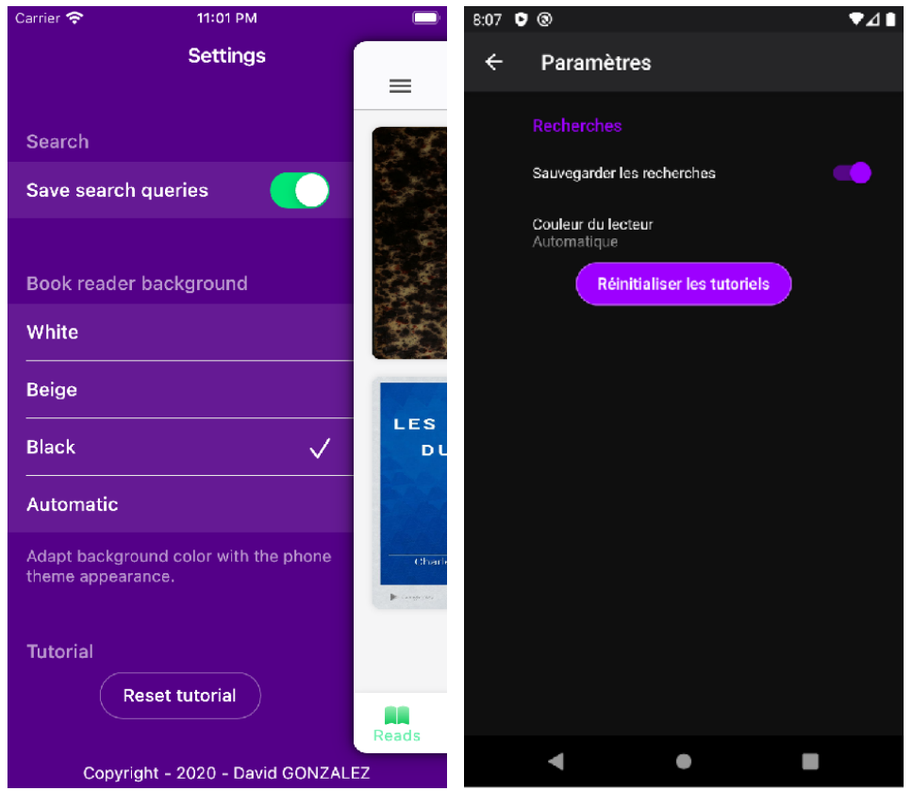
Epub reader page, also displaying one bookmark

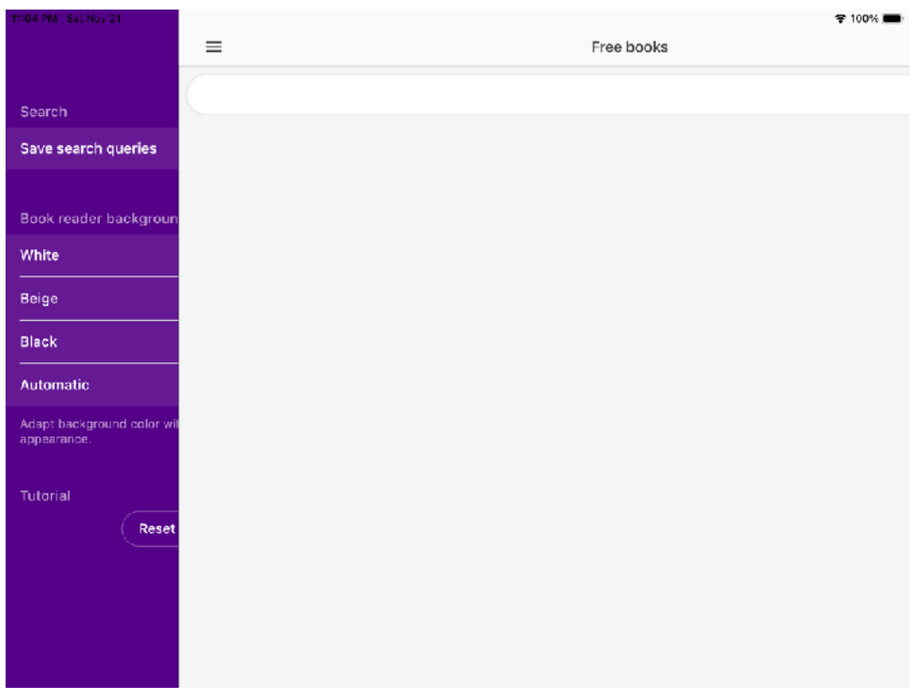
Application presentation

general feature : settings

Even if setting page have different display (layouts), they all **uses the same set of data** to describe all settings parameters.

The rendering of these parameters have been developed to (approximatively) **looks like native platform UI/UX** (except the background color).



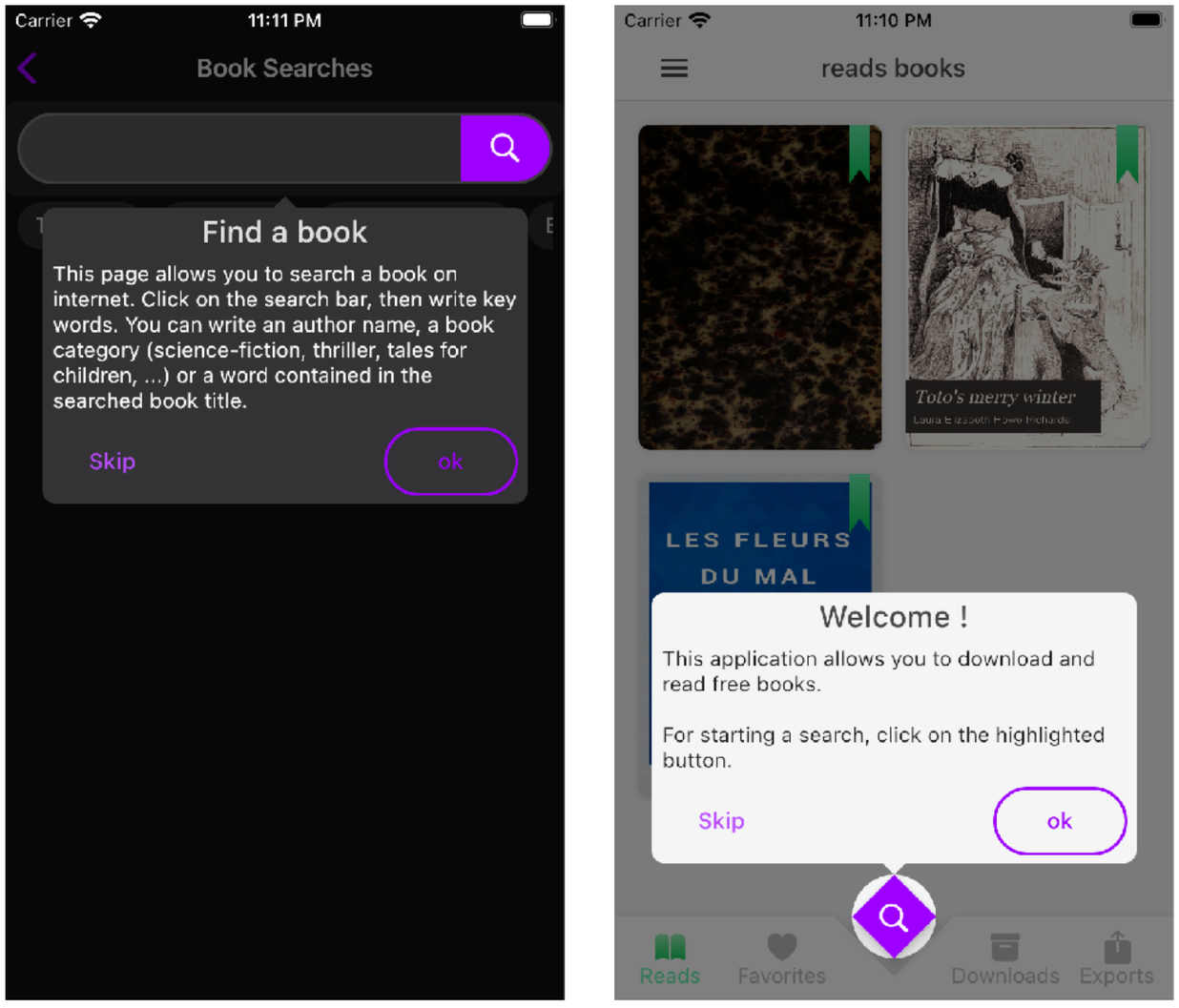


Application presentation

general feature : tutorial

Another useful tool used in that application, allowed me to integrate quickly **tutorials**, as **popup pointing to the element to describe**.

This component is from my homemade private library.



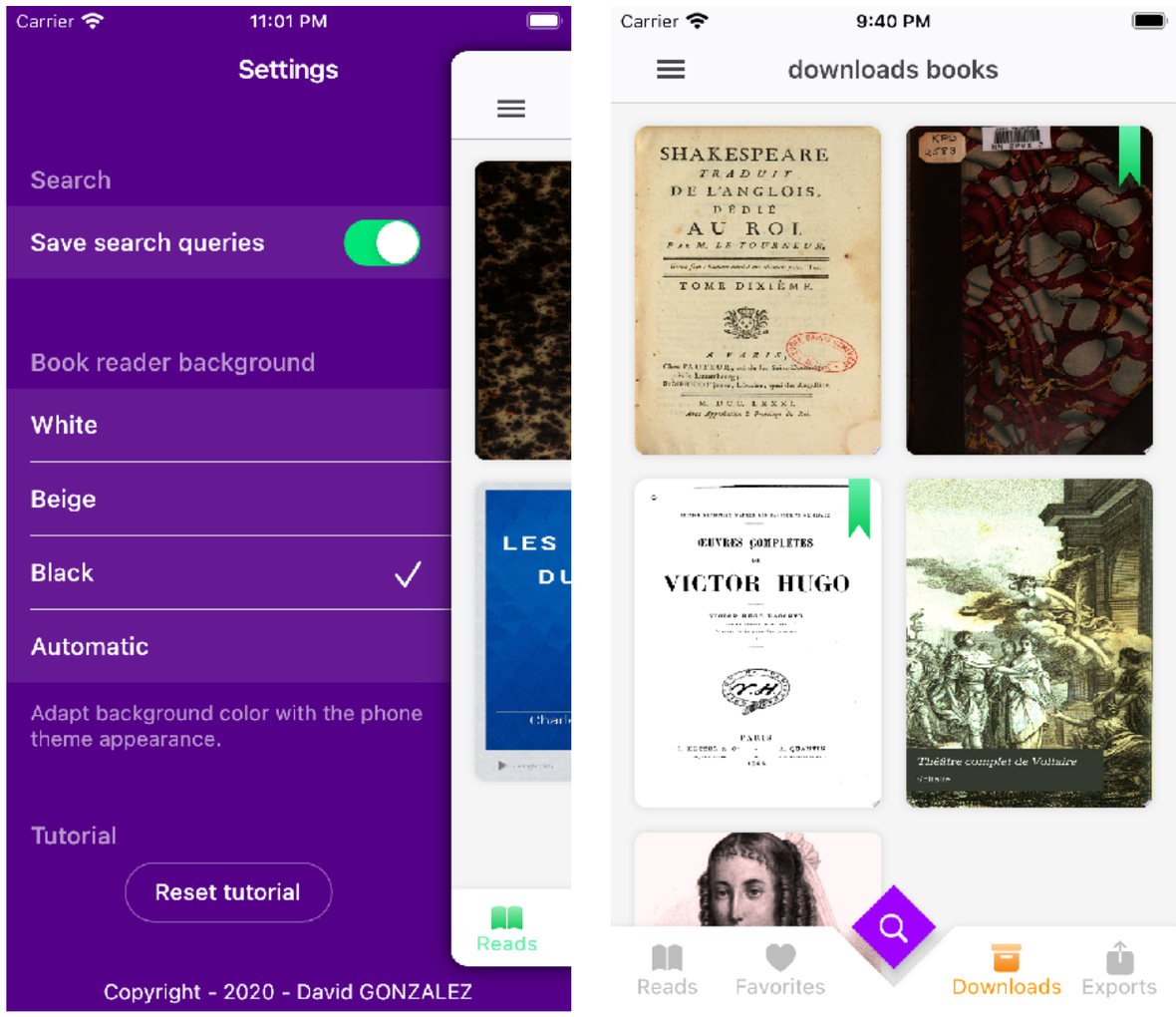
Tutorials display, directly integrated into pages

Application presentation

iOS Phone navigation

The iOS phone version has been designed to have some classic components such as the navigation bar.

Added to that, some trends UI elements such as the **underneath menu** and a **notched bottom bar.**



Left: underneath menu
Right: bottom notched bar with floating action button

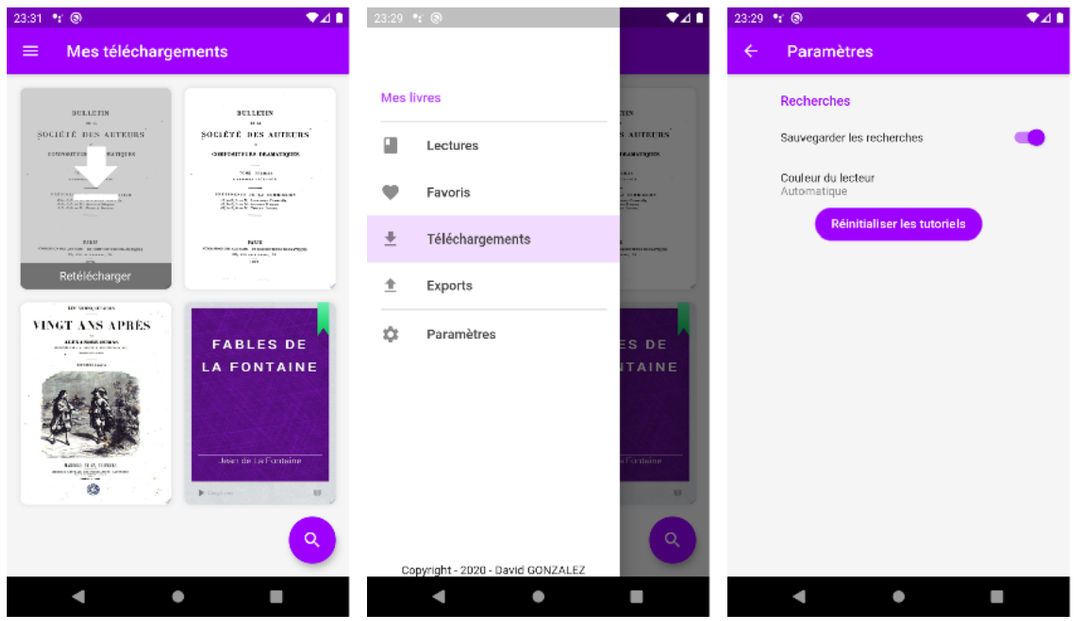
Application presentation

Android phone navigation

Navigation in the Android phone is slightly different, better corresponding to Android world.

From the home page, we can open the “**drawer**” menu displaying book filters and settings.

The settings are displayed in a dedicated page.

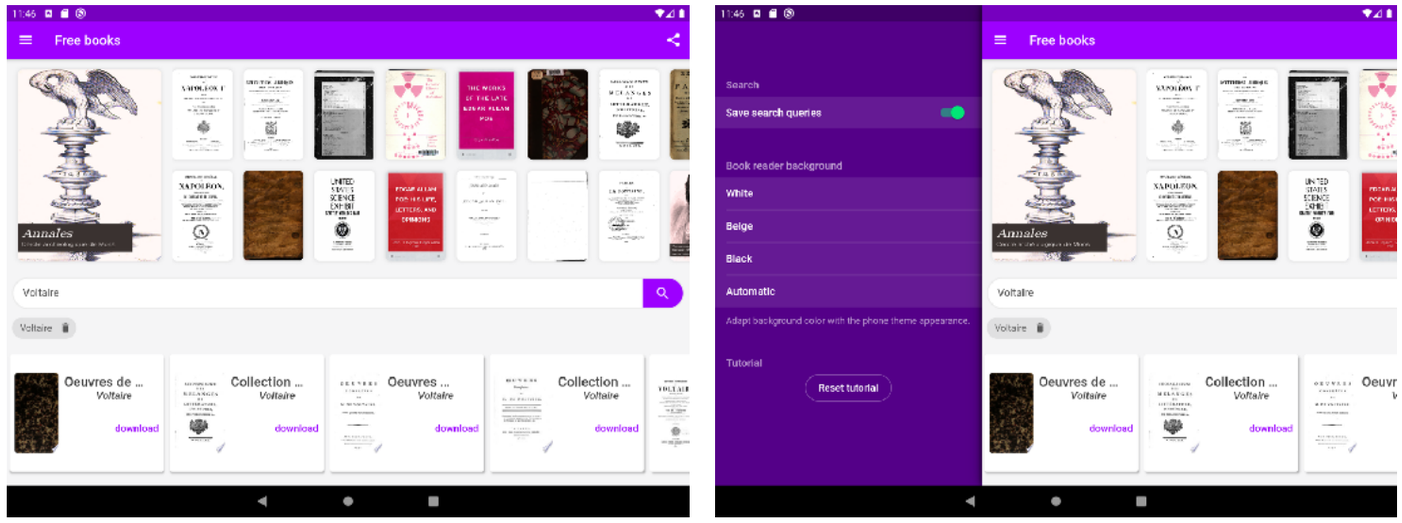


Left: home page, **middle**: menu opened, **right**: setting page

Application presentation

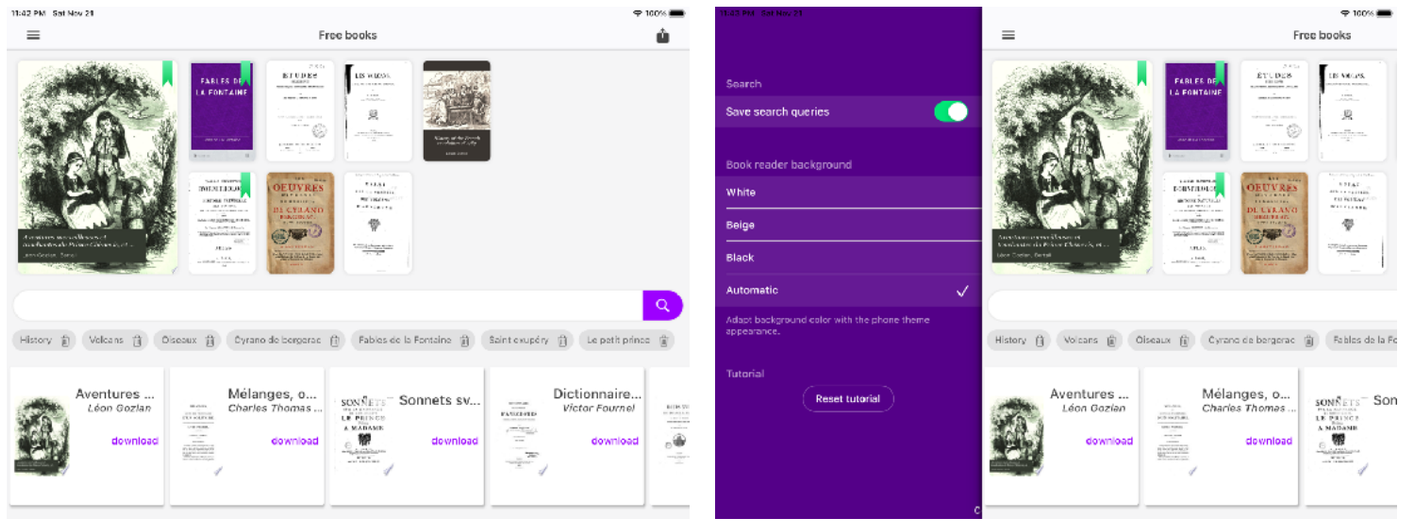
Tablets navigation

Design on tablets is **radically different** from phones, leading to a **unique page**.



Android Tablets design

Thanks to the extra space, we separate the screen in two spaces: on top the list of books, on the bottom the book search module.



iOS Tablets design

Child Variant

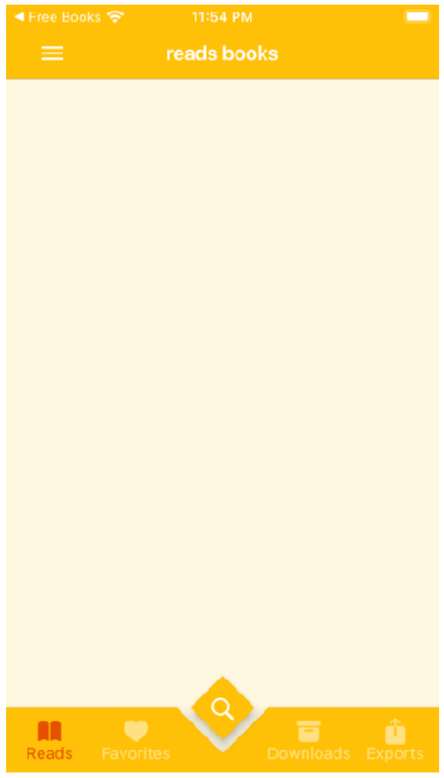
Application presentation

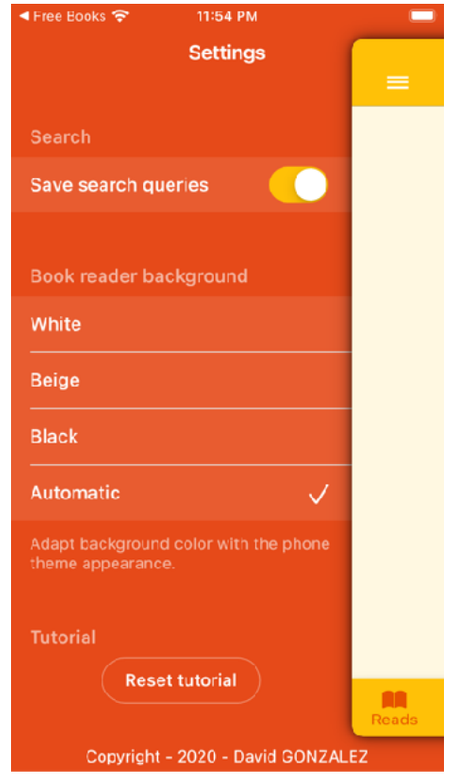
Child variant

For now, the application child variant contains exactly the same features. Only design (**theme**) changes.

On next version, a control policy will be added to **filter searches**.

One idea is also to get books from **another web service** (focus on child books) for the child variant.





Application presentation

Variant architecture

Differentiate application variants is done at the beginning, in the main function. I save all **Flavors** data into a static variable, accessible anywhere in the code.

Moreover, to add custom behaviour, the flavour data will be able to **contain involved objects in the creation of the provider**. Like this, I will be able to develop a new HttpBooksApi for child application, pointing to a dedicated child books API.

For design, I mainly focused on **theme customisation**. Like this, **I let widgets build themselves** without explicitly defining their background color, size, behaviour, and so on…

Conclusion

Conclusion about the development

Pros

- I was tough, but I provided on time applications that demonstrate, for now, what I'm capable of. **That's far from a good application**, but I'm satisfied that the application actually let users find, download and read books. All with a database, for an offline readings.

- Time was the main opponent, but I allowed myself to **refactor** some part of the application to always have a **good overview in the code**: splitting classes, redisign features as mixins, merging similar methods, …

Cons

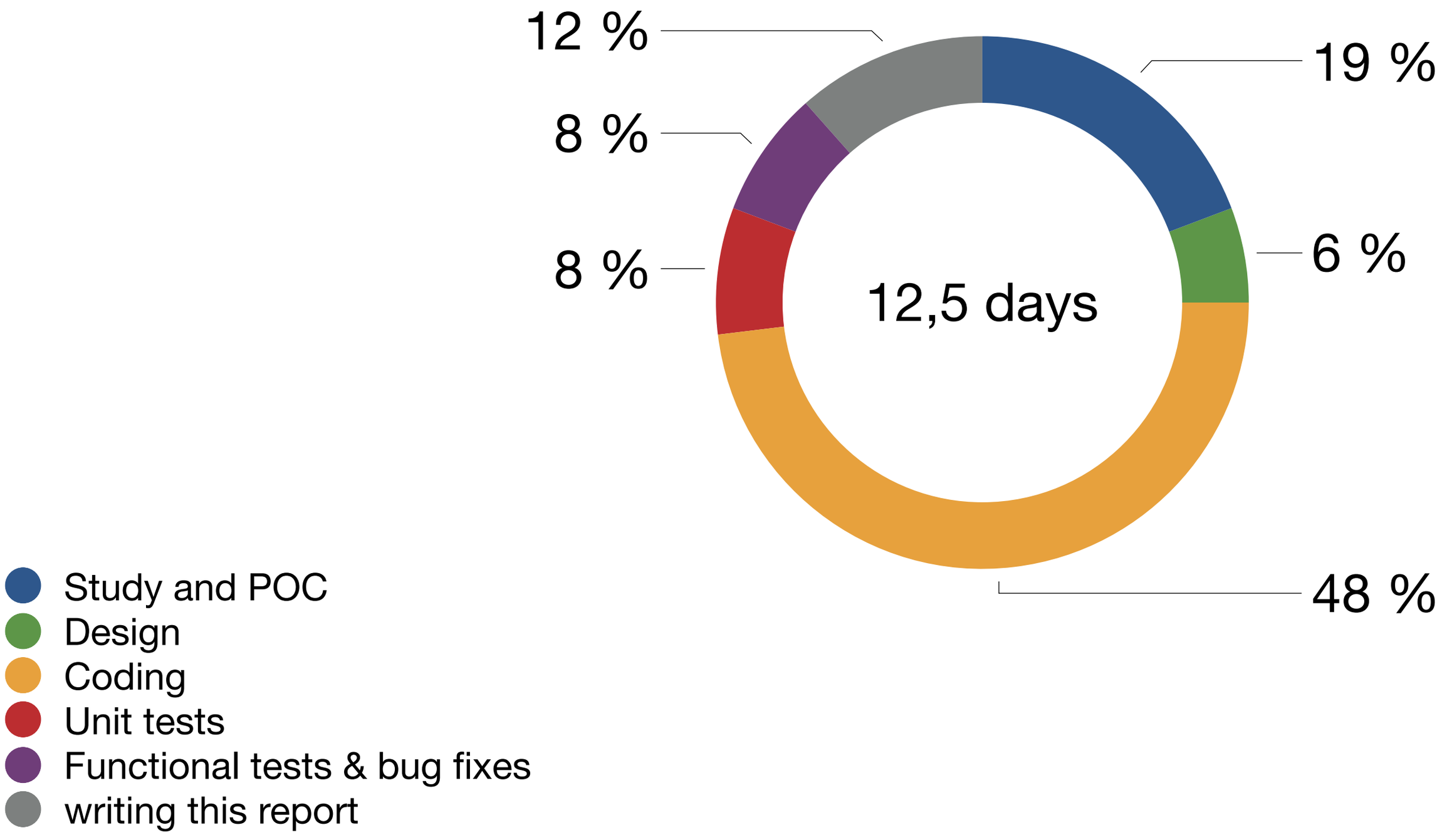
- Lack in **unit tests**: I would loved to deeper self training on UI testing

- Lack in the **design**: Child variant hasn't been designed actually. Theme changes only **demonstrate** that I can radically change the look.

- Lack of **animations**: animating UI can be **time expensive** (rather than technically difficult), that's why I avoid spending time on that development part.

- A lot of **instabilities in epub reader**, and a lot of bugs to fix

Time allocation



Time allocation

The methodology I developed focus on knowledge first, in order to estimate as soon as possible all **big risks**. At the end of the third day, I was able **establish a plan** to deliver an application on 21st of November (this report included).

I focus development on coding, avoiding testing all the UI through Flutter. I only **made unit tests for technical parts**, to be able to test critical components, such as the provider and the solution to use the google books api.

Testing database api through unit tests was not worth it, because I used "*DB browser for sqlite*" to tests data insertions, sql requests, and so on …

Concerning design, I focused on iOS phone design first, then on tablets. I avoided designing the "child" variant of the application, in order to **keep a little extra time for an unknown risk** that may occurs.

Lack of time during the development

The application is **very basic in its conception**: no animation, no image placeholder when downloaded, missing informative dialog or snackbar messages (such as download status in the search page).

My goal is to be able to **show my technical skills and methodology** more than aesthetic terms. But as you will be able to see, an effort in the design has been made, to have a UI and UX design close to the targeted platform, such as **navigation**.

Concerning the *child* variant of the application, it exists only to let me explain how I will handle differences.

Conclusion of this presentation

As you understood, the purpose of that document was double: on first hand, show you that I can handle an application development using Flutter sdk.

Technically, that may not be very impressive, but keep in mind that **I developed it alone** in **12 days**!

On the other hand, as you may understood, **the real purpose was to demonstrate that a good methodology is far more important that Flutter skills**. It let me adapt my development to ensure feature delivery, without a compromising **maintainability** and (a certain) **code quality**.

Regarding this last aspect, you may not find a class exceeding 300 lines of code, and commonly methods are not exceeding 30 lines. That's one of my **measurements** for code maintainability and quality. *Who likes to debug a class of 2000+ lines of code, having a method of 800+ line (and with nearly infinite widget indentations) ?*