

Developing a management mobile application

Ferreira da Silva João Carlos



Author	
Ferreira da Silva João Carlos	
Degree programme	
Business IT	
Thesis title	Number of pages and appendix pages
Developing a management mobile application	57
<p>Web applications, mobile applications and progressive web applications are the three main types that can be used in smartphones today. These have multiple pros and cons and the decision about which type to choose must be made before the start of the development.</p> <p>When developing a mobile application, the developer should take into account at least two operating systems: Android and iOS. This can be achieved by either building two native applications or by using a framework or development kit that allows the creation of the two version with only one source code.</p> <p>The purpose of this thesis is to build an application that answers the management needs of a nail salon in Switzerland such as appointment management and storage for client information.</p> <p>As the app should be available in iOS and Android, it will be built using Flutter which is a development kit created by Google that uses Dart as its main language to build the application. The backend part will be created using Firebase which can contain the database and numerous functionalities that will be discussed later during this thesis.</p>	
Keywords	
Android, iOS, salon, management, mobile, Flutter	

Table of contents

Table of Figures.....	1
Abbreviations.....	2
1 Introduction	3
1.1 Sponsor company.....	3
1.2 Objective of the project	4
1.3 Out of scope	5
2 Use Case	6
2.1 Requirement analysis.....	7
3 Existing solutions	8
3.1 Fresha.com.....	8
3.2 Vello.....	10
3.3 Vagaro.com	12
4 Application types	13
4.1 Web app	13
4.2 Mobile app	14
4.3 Progressive web app.....	16
5 Comparison of possible solutions.....	19
6 Development tools	20
6.1 Android SDK	20
6.2 VS code	21
6.3 Flutter	22
7 Database and backend	25
7.1 Firebase.....	26
7.2 Realtime Database and Cloud Firestore	28
7.3 Data structure	29
7.4 Cloud functions	31
7.5 Cloud storage	34
7.6 Firebase security rules.....	34
8 Frontend.....	35
8.1 Mock-ups	35
9 Widgets.....	37
9.1 Layouts	39
9.2 inputs	43
9.3 calendar	44
10 Results.....	45

11 Conclusion and future development	48
11.1 Future development	48
11.2 Conclusion	49
12 References.....	51

Table of Figures

Figure 1 use case of Pretty Nicky app.....	6
Figure 2 Popularity of Flutter and react native on Google trends.....	16
Figure 3 concerns when building PWA's	17
Figure 4 comparison table of possible solutions.....	19
Figure 5 example of AppBar	22
Figure 6 Example of StreamBuilder	23
Figure 7 result of Flutter doctor confirming that VS Code is configured.....	24
Figure 8 commands to create and open a new Flutter project.....	25
Figure 9 List of Firebase's services within categories.....	26
Figure 10 Realtime database and Firestore logos.....	28
Figure 11 database structure for the application	30
Figure 12 cloud functions example	33
Figure 13 Firestore rules.....	34
Figure 14 client management screens	36
Figure 15 calendar and appointment management	36
Figure 16 home screen and finance management	37
Figure 17 schema of the main screens and how they are linked	38
Figure 18 Example of row and column and their axis (flutter, Layouts in Flutter, s.d.)	39
Figure 19 Row with Expanded children and flex property (flutter, Layouts in Flutter, s.d.)	40
Figure 20 Stack example from the application to position the FAB in the right bottom	42
Figure 21 Example of aligned Stack (left) and Stack with Positioned children (right).....	42
Figure 22 two Cards with one ListTile each to display two appointments	43
Figure 23 example of validator for FormField.....	43
Figure 24 example of form submission to trigger the validators.....	44
Figure 25 example of TableCalendar	44
Figure 26 screen capture of the application client management screens	45
Figure 27 screen capture of the application appointment management screens	46
Figure 28 screen capture of the application finance management screens	46
Figure 29 Screen capture of the email sent to the client.....	47

Abbreviations

B2C – Business to client

UML – Unified modelling language

PWA – Progressive web apps

HTML – Hypertext Markup Language

CSS – Cascading Style Sheets

URL – Uniform Resource Locator

UI – User Interface

API – Application program interface

HTTPS – Hypertext Transfer Protocol Secure

SDK – Software development kit

IDE – Integrated development environment

CLI – command line interface

FAB — Floating Action Button

1 Introduction

1.1 Sponsor company

Pretty Nicky is a nail salon founded in April 2019 and based in Vevey, Switzerland. The owner began her career as a nail artist before that date, going from house to house to do her work at her clients' homes.

During those humble beginnings, the owner, and currently the only worker of the company, got used to a paper agenda to make appointments and keep track of money transactions.

Since the salon exists in Vevey, the owner dedicates a 100% of her time to the business and the number of clients has grown, which makes it harder to keep track of appointments and finances.

1.1.1 Advantages of a paper agenda

After some suggestions to use a digital agenda, the owner prefers the traditional way of doing because none of the options presented correspond exactly to the way she's used to work. After a quick discussion, different advantages of a paper agenda were pointed out by her.

First reason was the freedom to take notes about the client and the appointments. With the current way of doing, she can easily take notes about how an appointment went or about reasons of a possible cancelation.

Other than that, with a non-digital solution, it's easier to take appointments whenever and wherever she is. As the company is still a start-up, the founder takes appointments even during her days off which could mean a limited access to internet or even electricity.

1.1.2 Advantages of a digital agenda

On the other hand, some stakeholders have shown their interest for a digital solution. Two stakeholders of the company are the marketer currently helping with marketing and online advertisement and the company managing the accounting and tax reports.

To well manage the marketing department, there is an increasing need for a digital solution that could help with statistics about which type of services are more popular or which type of clients are more loyal.

On the finance side, it would be easier and more reliable to keep track of transactions using an application. Client payments could automatically appear when an appointment is marked has done and other payments could be simply created in the same application.

Moreover, this type of solution could help the owner managing clients. Simply searching a client name, it would be possible to find information about past and cancelled appointments and more information about clients.

1.2 Objective of the project

The conclusion of this thesis will provide the company in question with a beta version of the application that solves or offers a possibility to solve the following problems:

- Difficult collaboration with accounting expert
- Difficult analysis of data to help define a marketing strategy
- Photos of client's nails are stored randomly
- Searching for client's information is difficult
- Online applications don't exactly replace a paper agenda

All concepts discussed with the client will be documented, either in text form or using UML diagrams, as well as any research on technologies that could be included to help develop the application.

Once this project is completed, the writer should be able to develop and structure a functional application with ease and have a broader knowledge of existing technologies that can be included in future projects.

1.3 Out of scope

During this thesis we will develop an application with a lot of features. During research about databases and user experience, features such as image storage and dashboard will be taken into account but will not be developed. The end of this project should provide the sponsor company with an management application in beta version that has its main features function such as client, appointment and transaction management.

Moreover, the possibility of a future connection to the website or to a future client-side application could be considered during development but are not the responsibility of the writer during the thesis.

Also, all future development, testing and maintenance will be held outside of the scope of the thesis. As the application requested is complex, a Beta version will be the result of this project.

2 Use Case

To better illustrate the needs of the company, we will use a use case diagram with the three roles for the nail artist and the two stakeholders mentioned above.

A use case is a UML diagram that simply represents possible users of an application and their interactions with it. Thanks to its simplicity, these diagrams are very helpful during communication between client/user and developers.

To go even further illustrating the application structure, other type of diagrams can be used, such as class, activity or sequence diagrams.

The management application (subject to this thesis) will not have a complicated logic because most of the functionalities are data insertion and visualization. A use case should be enough for all concerned parties to understand the structure of the application that will be developed.

The following image illustrates the interactions with the app for the following possible users: Marketing expert, Nail artist and an Accountant.

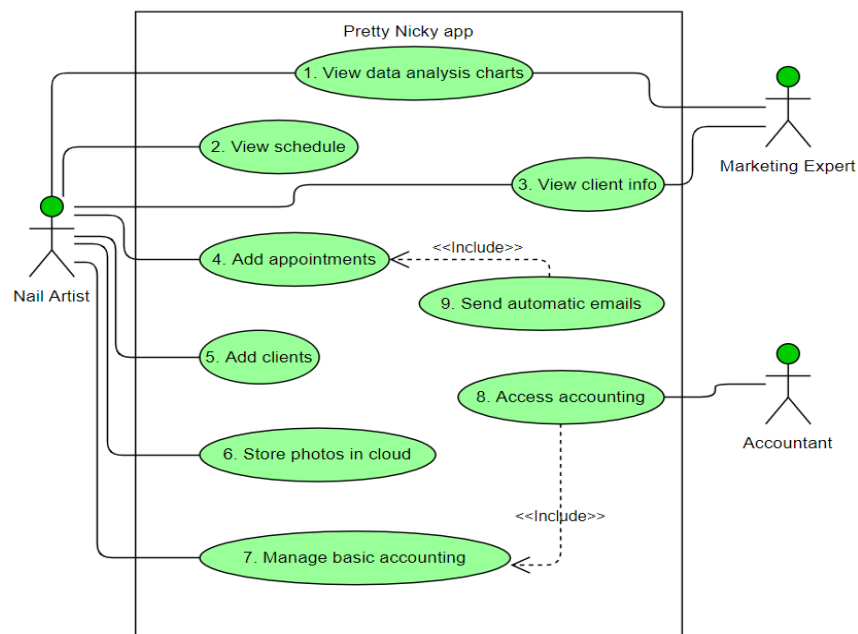


Figure 1 use case of Pretty Nicky app

2.1 Requirement analysis

1. View data analysis charts – A dashboard with some charts and filters should be available to help the marketing expert make some data analysis that will help him deciding on campaigns or price adaptation. This functionality will not be developed for the beta version but the database should be built considering this need.
2. View schedule – The nail artist should be able to view a schedule of appointments by day. Some information should be easily read, such as hour, client name, service type and price. Other information should be accessible by opening the appointment.
3. View client info – The marketing expert needs some contact and geographical information about the clients stored in a database. The nail artist needs to be able to find clients and have information about appointments and the sum of past unpaid appointments.
4. Add appointments – The nail artist needs to be able to add appointments for clients so they can be displayed in the schedule. These appointments should be stored with notes about it and a status to indicate if it's opened, cancelled or paid.
5. Add clients – If a new client wants an appointment, the nail artist should be able to add his information to then be able to add appointments.
6. Store photos in cloud – The nail artist wants to store photos in the cloud. The suggestion to have pictures related to each appointment was accepted by the sponsor company and should be integrated. Sadly this wont be part of the beta version but the design of the app should integrate this.
7. Manage basic accounting – The nail artist wants to stop using excel to store each transaction. A small company like Pretty Nicky only needs to separate incomes from expenditures so a functionality to manage finances would be easy to implement. This feature should also integrate all paid appointments as income.
8. Access accounting – Using the feature of point 7, we can store all transactions in our database. The accounting responsible should be able to see them easily to fill tax reports for example.
9. Send automatic emails – An automatic email should be sent to the clients when a new appointment is created for them. This email will be the confirmation of each booking and should be sent as an html document that displays all the company's design.

3 Existing solutions

Before starting development, we should take into consideration the solutions already existing in today's market. Even if the sponsor company asked for a proprietary solution, looking at the competitors could be beneficial to analyse their strengths and weaknesses. This information can then be considered when developing the application.

Doing some research about tools available for this type of management, we found three that could be useful for the business. These three applications are Fresha, Vello and Vagaaro.

3.1 Fresha.com

Before launching Fresha.com, the company was known for the application by the name Schedul.com. The company was founded in 2015 and, in April 2019, raised \$20 million dollars in series B investment, which is the second round of investment after the company has already reached some of the milestones established before. (O'Hear, 2019)

Fresha.com has since replaced the first app, having the same functionalities for the business site but also launching a new app for the client part, becoming a B2C solution.

Today, Fresha.com has 50,000+ partner businesses, 150,000+ stylists and professionals, is available in over 120 countries and has over 250 million appointments booked. (fresha, s.d.) It offers valuable functionalities such as appointment scheduling, analytics and staff management.

3.1.1 Pricing

For the moment, every new client will have the main features of the application without any subscription fee or monthly charges, but this is uncertain for the future. Currently, Fresha.com generates revenue by charging a small fee for each booking made via their application.

As the premium version is not available in all countries, the US prices are the following:

- 2.19%+\$0.20 for each transaction which includes transfers to the client bank and payments from clients booking online.
- 20% fee for new clients that discover the business through Fresha marketplace. (fresha, s.d.)

Companies need to move their payment processing to Fresha to get all the benefits from the premium version.

3.1.2 Functionalities

Based on personal experience, the most important features of this application are free and easy to implement. Some of these free features are:

Staff management – This feature is not valuable to the sponsor company in its current state but could be useful in the future. It allows the manager to add staff members to the company and manage their availabilities and appointments.

Client profiles – With Fresha.com, we can store client information and easily find them in a list using a search box. We can also easily see a list of all the bookings for a client and some of its statistics.

Appointment scheduling – The most interesting functionality of this app is the appointment scheduling. We can save bookings per client and per staff member for a specific service that takes a specific amount of time. There is the possibility to take notes for each appointment and identify them with labels such as new, cancelled or no show.

Reporting and analytics – In the analytics tab we have some insight about the company. We can here see how many appointments are booked online, total sales and appointments, client retention and more.

Inventory management – This feature allows the user to register products and suppliers. There is then the possibility to create orders for these products and send them automatically to the supplier by email.

Other functionalities are part of the Fresha plus mentioned before. These include payment processing, SMS reminders, marketing tools and presence in their marketplace with a possible connection to the social media accounts which allows online booking.

3.1.3 Disadvantages

The vast number of features can be considered an advantage, but it makes Fresha an app for bigger companies with bigger needs. As stated before, Pretty Nicky is a small company used to a paper agenda and the transition to this application can be complicated and overwhelming.

Also, the mobile application doesn't save any sort of cache, which means that it won't work properly without an internet connection. This functionality could be useful for this business as it was mentioned before.

3.1.4 Conclusion

Fresha is a very complete solution and the company behind it is still developing new features that could also be valuable for the company in the future.

There are no guaranties that this application will remain free but for the time being it can be a powerful tool for companies with employees and a lot of inventory to manage. All of it without subscription fees or monthly payments.

Unfortunately, it is not a modular application and even if it is user-friendly, the number of available features can be overwhelming for a company like Pretty Nicky where the staff is not used to a digital solution.

3.2 Vello

Vello is an application made by the Finnish company, Vello Solutions Oy. It is a simple application where users can easily set up an account and it's used today by more than 11800 entrepreneurs and organizations worldwide. (Vello, s.d.)

The company was founded in 2015 and sales started rising since 2016. New features for this app are still being developed and we can expect even more in the future.

3.2.1 Pricing

The free version is great to start exploring the application, but it doesn't allow companies to save a lot of information. Users can choose to pay a monthly, 6 months or a year subscription. Saving up to 35% by selecting the longer option. (Vello, s.d.)

Three types of subscription are then offered to the users:

- The Free version, described as “great for start-ups” allows the user to register one employee, up to 50 customers, 5 types of services and enable email support.
- With the Basic version, users can have unlimited customers and services and in addition to email, have phone support. This version costs 14€ per month and per employee / resource for the monthly subscription (10€ per month for the 6 months subscription and 9€ for the annual one).
- Finally, with the Premium version users can book group appointments, have personal support, reminders, services available only for employees and more. This version costs 20€ per month and per employee / resource for the monthly subscription (15€ per month for the 6 months subscription and 13€ for the annual one).

3.2.2 Functionalities

The most important functionality is appointment scheduling. As seen with the previous app, with Vello the user can create appointments for clients for a specific service that takes a specific amount of time.

Vello also registers a list of customers to the database. With this application, just by creating an appointment and filling out the name field of the new client, the app automatically registers the new client in the database. This makes it faster to make appointments to first time customers.

Moreover, the Vello application allows its users to see reports of their bookings and customers per year or month.

Finally, with the paid version, users can create a customizable online booking page, send automatic reminders to the customers and so on.

3.2.3 Disadvantages

In this precise case, the price is a disadvantage because a proprietary solution will be developed for free. In a normal situation, the monthly cost of this application is still very low.

Also, Vello doesn't offer the possibility to identify appointments by labels. They are either past, future or cancelled which leaves the user without any information about payment status.

Finally, this application doesn't allow the user to upload files for an appointment. One of the current problems in Pretty Nicky, even if not urgent and not relevant for the beta version, is to store data such as photos for each appointment.

3.2.4 Conclusion

This application, even if more simplified than Fresha, is still not a complete solution for the company's needs.

The version of this app that will correspond the most to the company's needs would be the basic one which will cost 108€ per year for the annual subscription. In this precise case, a proprietary app will only have the maintenance costs of the database which are lower than the ones offered by Vello for the same data stored.

3.3 Vagaro.com

Vagaro is another an app that let's businesses manage schedules, invoices, customers and so on.

This app exists since 2009 and since, it has been exceeded by its competitors because it has some flaws that were never corrected. (Vagaro Reviews & Product Details, s.d.) Simple things like having a client without an email address or sharing one with other clients are not possible. Even if this application offers a lot of interesting features like a website builder, email marketing or the possibility to integrate online booking in social media, it lacks in quality with the most basic and important ones. We can also find some complaints online about minor bugs in the mobile app and about servers running slow.

The price for this app starts at 25\$ per month and increases 10\$ for each extra employee that the company wants to register. (Vagaro Reviews & Product Details, s.d.) This makes the cost of this app too high for its value and not a good solution for the sponsor. Another disadvantage of this app is that users need to create an account with their credit card information and cancelation is only done by phone.

4 Application types

After taking a look at the existing apps that could solve the current problems of the sponsor company, none was the correct fit for the nail artist. Because of this, the company still desires a proprietary solution.

There are three major types of applications that can be built for this purpose and they are: web apps, mobile apps and the more recent type, PWA's or progressive web apps. These different types present some differences not only for developers but also for the users.

To decide which type best suits our company, let's look at these three different types.

4.1 Web app

A web app is usually built with HTML, CSS, and JavaScript and don't need to be downloaded to be used. The user can simply get access to them by typing the right URL in any browser.

To develop these apps, a developer only needs one codebase regardless if the app is going to be used in a desktop, Android phone or iOS, which makes these apps easy to maintain and to build. Also, because they are operated in a browser, these apps don't need to be published in any application store. The publisher only needs to own a domain and have a server available to publish his app.

On average, the maintenance of a web application could be 150\$ per month if we take into account the price of the online servers to host the content and the database, the cost of the domain name and also the price a IT expert that would take care of all security updates of the web application. (Fowler, 2019)

This type of application also has some downsides (Chheda, 2016):

It is impossible to reach the app server without an internet connection and so, without saving any data in the phone, it is impossible to open the app while offline. This also means that depending on the bandwidth, the application may run slower than any other type.

Also, because these apps are not built with a native code, it's difficult or even impossible to access some functionalities of a smartphone, such as the camera, notifications, or file system. The application is restricted to the functionalities that can be accessed by the browser.

4.2 Mobile app

Mobile apps are designed only to be used in smartphones and tablets and in contrast with the other types of applications they need to be downloaded. With all the codebase stored in the phone, this type of application is more secure against hackers, can easily access more functionalities of the device and even work offline. They are although, more expensive to build than web apps because they are usually more complex than a simple web app and require more specific knowledge from the developers.

To build mobile apps we can use frameworks, development kits or build it with native code which most of the time means developing multiple codebases for multiple platforms.

4.2.1 Native development

Native development refers to the creation of one app using the native language of the operating system where they will run. For example, native applications on Android can be built with Java or Kotlin, on iOS using Swift and on Windows phone using C#. (Siripathi, 2017)

The market share for Android phones is today 76% and 13% for iOS (Milijic, 2019). Because of this, when building a mobile app, developers usually will do it for those two operating systems.

However, building an app with native code doesn't mean that we shouldn't be careful about its usage in multiple platforms. Almost every operating system will be available in multiple types of devices, such as tablets, smartphones and more recently, smartwatches. Moreover, Android is an operating system used by multiple smartphone brands which differ in screen size and ratio.

The application requested by the sponsor company should be available in iOS smartphones and Android tablets, which would mean that two codebases should be developed for this application.

Even if native development is the best way to have access to all the features of a smartphone without any restrictions, to be available in multiple different platforms the developers need to use different languages. That's where Frameworks and development kits come into play.

4.2.2 Compiled app

Compiled apps can be created using frameworks or development kits such as React Native or Flutter and those also have their differences on how they work.

Flutter is a development kit built by Google that uses Dart, a language developed by the same company. It compiles code to C++ which is close to the native code. (Academind, 2018) This means that apps built with Flutter will be a bit faster than react native and will be able to access more features of the device. Developers can build an app by putting together numerous widgets available that have properties and can be personalized.

React Native is a framework that uses JavaScript, which is a very popular language that most developers know how to use already for other purposes such as web development. Contrary to Flutter, it does not compile the app to native code but only compiles the UI elements. It also has some pre-built components such as buttons or inputs that developers can use but this library is smaller than Flutter. (Academind, 2018)

React native differentiates between platforms and implements the design chosen in each mobile platform while Flutter uses material design in both Android and iOS. (Nader, 2020) This means that an application built using react native may look more familiar to a user

because it resembles some of the UI components of the operating system but some of the components will need to be developed twice.

If we go even further with our analysis, we can discover that the first alpha release of Flutter was in May 2017 and it has 89.9k stars on GitHub while react native was released in 2015 and only has 86.1k stars. This proves a much bigger growth in popularity for Flutter which makes it an interesting platform to use when building our mobile app. This popularity could be also proven with Google trends. The graphs bellow represents which of the two topics is more researched on Google for each country: Flutter (in red) or react native (in blue).

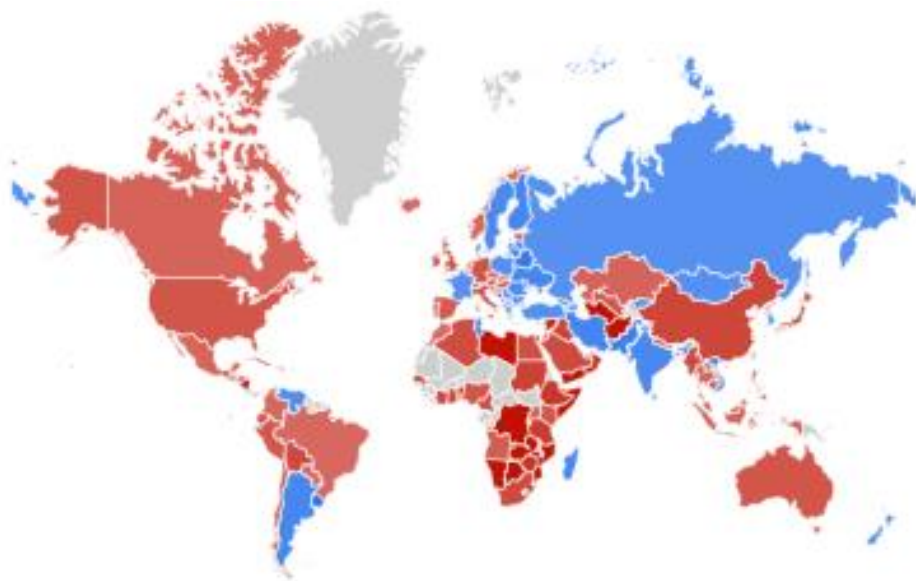


Figure 2 Popularity of Flutter and react native on Google trends

4.3 Progressive web app

Nowadays, people use more smartphones than computers and when using them, spend more time in applications than in websites. We could then conclude that it's more beneficial for a company to have an application rather than a website but that's not completely true. Smartphone users tend to use only some applications. Of course, as this application will be used by a restrict group of users, we don't need to be concerned about attracting

users, but this is the reason why this type of application was created, and it also has other advantages.

To better understand what a PWA is, let's look at Google's advice to build this type of application that focusses on four main concerns:



Figure 3 concerns when building PWA's

Firstly, developers should ensure that the app loads fast. More than half of the users will quit a webpage if it takes more than 3 seconds to load.

Furthermore, the apps should feel integrated in the device. A quick access from the home screen of a smartphone is now possible without downloading an app from the store. Also, the PWA should use some of the functionalities like payment API to take advantage of payment apps on the device (Google pay, Apple pay, PayPal) or media session API to add the media functionalities on the locked screen. This could improve usability for the nail artist to ensure that the app is easy to reach and to use

Also, developers must ensure that the app is reliable. Internet users are no longer used to "see the dinosaur in the browser" which would mean they are offline. Connectivity to the internet is nowadays something taken for granted but it's not always the case on mobile. Mobile users can experience low bandwidth in some geographical regions or might need to turn on airplane mode for some reason, but apps should be able to open even during those situations. A possible solution could be to save cache data on the device so, if the user has no connection, he can still see a not up to date version of the app.

Finally, it's important to keep users engaged and smartphones are a great way to make that happen with one of the best functionalities being push notifications. With modern browsers, web developers are now able to send notifications even when the browser is


closed. This concern is not important for our application but the fact that PWA's are able to send push notifications could be useful to our sponsor company.

Apart from these advices, delivering features in a secure manner is also very important. Not only to ensure integrity and confidentiality, that are the focus of SSL/TLS but also because some of the API's that can be used in PWA's only work if everything uses https.

All these capabilities are now possible thanks to service workers in browsers which can store cache versions of a web app and send push notifications when notified by the online server but also to app manifests that allow an application to be downloaded and appear in the home screen.

PWA's can be the future of mobile development but for our use case, a simple mobile app will be enough to deal with the problems of the company. Although, this type of application could be a great idea for a future client-side application that can be built using the same database.

5 Comparison of possible solutions



	Existing App	Web App	Mobile App
Data analysis charts for marketing and finance	Integrated with multiple options	High personalization for needs	High personalization for needs
Schedule with client appointments	Integrated and simple to use	Multiple options available	Some options available
Camera API and photo galery in the App	No	Not available through all browsers	Possible
user interface easy to use for the nail artist	Simple but many useless features	Tailored for the user	Tailored for the user
project lentgh and maintenance costs	Short and quite cheap	150€/month on average	Usually the most expensive type

Figure 4 comparison table of possible solutions

We can determine after this small study that the best solution for this company is indeed the development of a mobile application. Even if the project can take a longer period before being deployed in the company's devices, the only storage needed will be for the data and not for the app itself.

With a mobile app, there is also a possibility to have a system that allows some changes and queries without internet connection and the app will be developed with the specific user needs always in mind.

Some functionalities are easier to implement in a mobile application, such as the usage of the device's camera, and with enough workload, all the features necessary will be available.

Using an existing app would perhaps be the cheaper solution but the sponsor company is not interested in this type of software.

6 Development tools

After deciding which type of application to build we need to decide how to develop it. As stated before, there are multiple ways to develop a mobile application and for this project, we'll need it to be available for Android and iOS. This makes native development much longer to implement and the features needed don't require it.

Another way to develop a mobile application is with Flutter or React Native. Based on the short analysis made before about these two platforms, we saw that Flutter is becoming more popular and because its code is compiled to native, the application will run faster than one built using React Native. Also, Flutter has numerous widgets available to help with database connections which we will discuss later. We will then, for the development of this app, use Flutter.

For this we need to prepare our file system and work environment.

To develop this app on a windows computer we have the following requirements:

- Windows 7 or higher
- 4 GB of RAM
- 2,4 GB of available disk space (minimum)
- Windows PowerShell 5.0 or newer

6.1 Android SDK

To develop an Android application in Flutter we need a software development kit that in our case will be the one to develop Android applications. This SDK is installed at the same

time as Android studio, which is the official IDE advised by Google to build Android apps. (flutter, Windows install, s.d.)

An SDK is a set of tools in this case used to build Android applications. It includes libraries that are “premade” code that can be used by the developers instead of coding everything from scratch. This SDK also includes a debugger, an emulator and sample source code.

6.2 VS code

To develop an application, developers need an integrated development environment. An IDE is usually composed of a code source editor, automation tools and a debugger.

As stated before, Android studio is the advised IDE to develop Android applications. Although, during this project we will use Visual Studio Code, which is a code editor. A code editor is a software that allows developers to edit almost any type of files. VS code offers a lot of plugins which can add functionalities to it such as compilation, debugging, code auto-completion and many other functionalities which can make it behave like an IDE. Because all this extensions and plugins aren't integrated, we need to call it a source code editor. (Gregg, 2019)

The plugins that must be used in VS code to develop a Flutter application are Flutter (3.9.1 in our case) that contains a debugger and support for editing, refactoring, running and reloading Flutter applications and Dart (3.9.1 in our case) that also contains a debugger and support for this language that is used to build Flutter applications.

Image preview by Kiss Tamás is also a useful plugin that allows developers to see an image or icon by simply hovering their path. Bracket Pair Colorizer 3 by CoenraadS is also useful for Flutter development as it colour-codes brackets. This is helpful because a Flutter application is built by nesting widgets and with this plugin it is easier for developers to spot bracket pairs in their code.

6.3 Flutter

Flutter is a UI toolkit that lets us develop front-end applications using prebuilt widgets. Because those widgets will then be compiled into native code, they will work in all types of devices and operating systems.

To use this toolkit, we need to know Dart which is not as popular as other programming languages such as JavaScript for example. Gladly, it is a language developed by Google which means that it's very easy to find good documentation about it and because its logic is very similar to JavaScript, it's usually very easy to learn it for developers.

To add an Application bar for example, we can simply declare an AppBar widget with attributes to define its colour and title like in the example in Figure 5.

```
appBar: AppBar(  
  backgroundColor: Colors.blue,  
  title: Text("Pretty Nicky"),  
), // AppBar
```

Figure 5 example of AppBar

Being developed by Google not only gives us good documentation but also other advantages. Flutter uses Material Design out-of-the-box which is used by a multitude of applications and so is a familiar design to most of the possible users. Also, it has a lot of prebuilt widgets to connect the app with Firebase as a backend. (Developers, 2018) This allows us to quickly have an application that will automatically update when there is a change on the database with widgets such as StreamBuilder.

The code in figure 6 is an example of this Widget. In the property "stream" we can add our query to the database, in this case from the collection "clients" ordering it in ascending order. The builder property is where we can return a Widget to be displayed with the content we fetched. Here we have a list of ListTile inside a column that have information about each client.

```

StreamBuilder<QuerySnapshot>(
  stream: db
    .collection('clients')
    .orderBy('name', descending: false)
    .snapshots(),
  builder: (context, snapshot) {
    if (snapshot.hasData) {
      return Expanded(
        child: SingleChildScrollView(
          child: Column(
            children: snapshot.data.documents.map((doc) {
              if (filter.length == 0 ||
                (doc.data["name"]).contains(filter)) {
                return ListTile(
                  leading: Icon(
                    Icons.account_circle,
                    size: 40,
                  ), // Icon
                  title: Text(doc.data['name']),
                  subtitle: Text(doc.data['phone']),
                  onTap: () {
                    setState(() {
                      userId = doc.documentID;
                    });
                  },
                ); // ListTile
              } else {
                return SizedBox();
              }
            }).toList(),
          ), // Column
        ), // SingleChildScrollView
      ); // Expanded
    }
  }, // StreamBuilder
);

```

Figure 6 Example of StreamBuilder

6.3.1 Getting started with Flutter

To begin with Flutter, we need to have the Android software development kit installed on the computer. By installing Android studio, this SDK is immediately installed, and we have the possibility to download Android emulators directly from the IDE.

After this we need to download the Flutter SDK. We can have a fixed version directly from the website or get the source code from GIT which will help us with future updates. After downloading GIT, we can open the windows command prompt in the destination folder and run the following command:

- `git clone https://github.com/Flutter /Flutter.git -b stable`

In the same folder we can now run a command that will check if the SDK is installed, if an Android device is available (emulator or real device) and if the required plugins are installed in VS code.

- C:\src\Flutter> Flutter doctor

The result should look like this:

```
[✓] Android toolchain - develop for Android devices (Android SDK version
    29.0.3)
[!] Android Studio (version 3.5)
    X Flutter plugin not installed; this adds Flutter specific functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
[✓] VS Code (version 1.44.1)
[!] Connected device
    ! No devices available
```

Figure 7 result of Flutter doctor confirming that VS Code is configured

We can see in figure 7 that Android Studio is not configured and that's because we are not using it for the development of our application. Also, during this test, no Android device or emulator were connected.

The last thing we need to do is to add Flutter to our PATH variables. This allows us to use Flutter commands without being in the folder where Flutter is downloaded. To do this we can "Edit environment variables" from the control panel. Under "Environment variables..." edit the Path variable and add the path to the bin folder in the Flutter files downloaded.

Now that everything is installed, we can create an app, open it in VS code and run it in an available device or emulator using the following commands in the prompt:

```
> flutter create pretty_nicky_app  
> cd appname  
> code .  
> flutter run
```

Figure 8 commands to create and open a new Flutter project

An app will now open in our Android emulator. This is the sample app for Flutter which contains a counter that increments each time the user presses the floating button on the bottom right corner.

After making changes to the source code we can either reload the app with the command “R” in the same control prompt as we did the “Flutter run” or a hot reload with “r” which reloads the layout of the app without restarting it. Full reloads might be needed when the application stops or freezes for some reason but if we are only changing the interface, a hot reload should be enough to refresh the widget with its new properties.

7 Database and backend

Flutter is a UI toolkit, which means that we can use it to deal with the frontend side of the application, but an app also needs a backend that will connect the app with a database and add some functionalities that are not linked with the interface like the usage of external API’s for example.

As our application needs to be available in multiple devices with the same shared data, we can’t only store it in the device’s storage. We need a cloud database that will save all the changeable data.

Multiple providers offer cloud storage for our database such as Microsoft, Amazon and Google. For this application we will use Firebase which is an easy to setup backend and gives all the functionalities required for our project, which are offline support for data, content and data storage and the possibility to integrate functions to act has triggers on the database.

7.1 Firebase

We discovered before that Flutter has prebuilt widgets compatible with Firebase which is a platform created by Firebase Inc. and acquired by Google in 2014.

Before creating Firebase, the same company had created Envolv in 2011. It was a platform for real-time and embeddable chat services for websites and only later the company realized that most of the users weren't using the platform for chats but were instead using it for the real-time and synched data capabilities. They then created Firebase in 2012 focusing on this type of features. (Melendaz, 2014)

After being acquired by Google, this platform became a backend-as-a-Service (BaaS) offering many functionalities such as cloud storage, authentication, Realtime database, hosting, cloud functions, crash reports and integration with other Google services such as Google Cloud, Google ads or analytics.

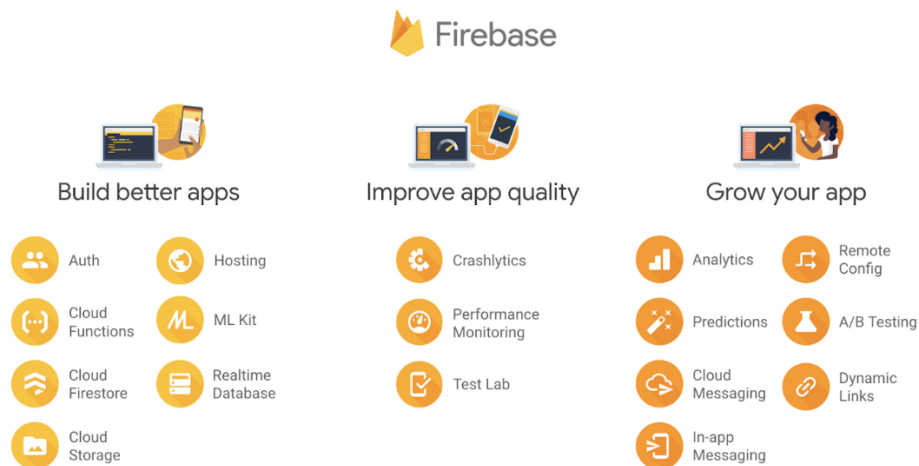


Figure 9 List of Firebase's services within categories

Firebase's services are divided in three categories that are presented as goals:

- Building better apps
- Improve app quality
- Growing the app

The first category helps developers create robust, secure and scalable applications. The platform offers cloud Firestore and Realtime database for data storage. It also offers authentication, cloud storage to store images or videos for example and cloud functions that allows users to create server-side functionalities and database triggers using node.js. Our app will use mainly features from this category such as Firestore, cloud functions and cloud storage.

The second category of functionalities is about improving the quality of the apps. Crash analytics and performance monitoring gives insights to the developers about the app's performance. Also, Test lab or even the beta version of App Distribution help the app makers to test new functionalities with groups of physical or virtual testers. During the development of the Beta version of the application we won't use features from this category but once the app is stable, crashlytics is a great feature to implement in case any errors occur.

The third and last category of features helps the app grow with tools beneficial for user engagement, in this group we can find Google Analytics, cloud messaging, or predictions to help developers know how users navigate through the app. Analytics is installed and configured during the first connection to the database so it will also be part of our application. It can help us detect how the user interacts with the app and then make changes before releasing an alpha version.

Some of these features can be beneficial for our company's needs. Cloud storage can solve the problem about image storage for each appointment, Firestore can be used for data storage and with cloud functions it would be easy to integrate an automatic mailing system to contact clients before their appointments.

7.2 Realtime Database and Cloud Firestore

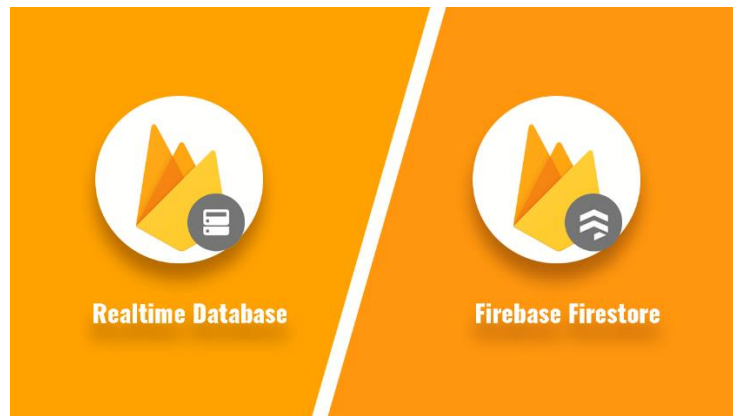


Figure 10 Realtime database and Firestore logos

Two databases are offered in Firebase. Realtime Database is the older and original type of database offered in Firebase and Cloud Firestore is the newer type, with better scalability, faster queries and richer features.

In both types data is stored in a NoSQL database but in the older one it is stored as a JSON tree while in Firestore it is stored in collections and documents. This is very similar to JSON but with some differences. (firebase, s.d.)

NoSQL is an alternative to relational databases where data is placed in tables and a data structure is carefully designed. This type of databases gives developers more flexibility on how they want to use data and are usually faster than relational databases.

In the newer type of database, it is possible to create indexes and do queries with multiple conditions. This allows developers to do more specific queries and retrieving only wanted data. In the older model, queries return an entire JSON subtree as they cannot handle filtering and sorting at the same time. (Kerpelman, 2017)

Also, autogenerated keys on Realtime database are based on timestamp which means that it can support a limited number of writes per second. (Firebase, 2018) Cloud Firestore is more scalable and able to withstand more connections at the same time as it doesn't use the same type of autogenerated keys. A timestamp can still be added to do queries

ordered or filtered by date. Of course, this wouldn't be a problem for Pretty Nicky, but it is a

One of the reasons why Firebase has two types of databases.

At last, they both offer offline support for iOS and Android but Firestore offers it also to web applications which can be beneficial for a future web app or PWA for a possible client-side application. This functionality helps developers to build an app that saves data in cache and use it when data is not available.

Following the analysis made during this thesis, we will use Cloud Firestore as a backend for our project.

7.3 Data structure

Before creating a Firestore database we need to decide how the data will be structured. We need to consider how data will be accessed by the app in order to decide which structure would be more efficient. Firestore will only save data for the application and all other content is saved in Cloud Storage.

Data is stored in documents that are in nodes. For our application we will have four nodes that are appointments, clients, finance and services. Each document in those nodes will have an Id and properties where the data is saved.

This database will mostly use Strings to save data and it doesn't enforce any integrity for foreign keys so, when saving data in external nodes we need to make sure that the app will save the right values. Also, when deleting documents, we need to choose if we want to delete all the other documents related to it or not.

The schema below represents the structure of our database with four collections with documents identified by their id. Each document then has different properties that describe it.

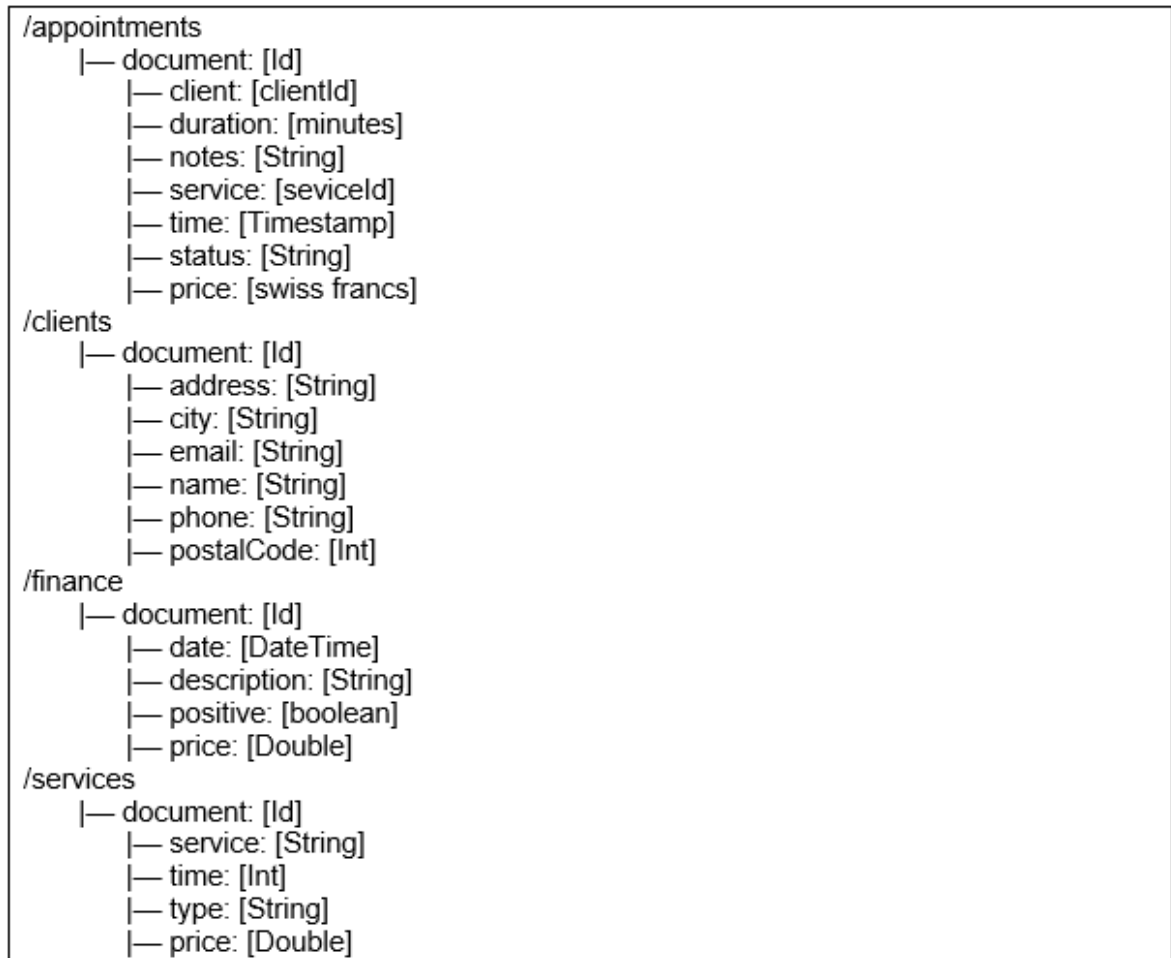


Figure 11 database structure for the application

The first collection will be for appointments. These could have been held in a sub-collection in clients but when accessing the full schedule, we will need to access all the appointments for all the clients which means we would need to regroup the results of the query. Each appointment document will have an autogenerated id, a client id, a service id, notes about the appointment, the time of the appointment, a price, a duration and a status that will allow us to know if the appointment is cancelled, open, paid or due.

The second collection is where client documents will be saved, each client will have an autogenerated id, an address, a city, an email, a name, a phone number and a Postal Code. All this information are currently used by the company to save information about the clients and can also be very important for statistics.

The finance collection will hold all non-appointment related transactions. They will have a date, a description, an amount and a Boolean that will be true if it's an income or false if it is an expenditure. We could include appointment transactions when they are paid but this will result in duplicated data. Because we are using Firestore we can simply do a query filtering only paid appointments and quickly have all the transactions related to client payments.

The last collection will be used to store the types of services offered by the company. Each document will have an id, the service title, the time in minutes that the service takes and the type of service (in this case either hands or feet). This collection will be used mostly to autocomplete forms, when the user selects a service, the price and duration of the appointment will automatically be filled but can still be changed.

7.4 Cloud functions

One great service of Firebase is Cloud Functions which allows us to deploy some server-side functionalities. We can deploy triggers for the database or simply create some functions that reply to HTTP methods.

The Cloud Functions feature can have multiple use cases such as database maintenance which can be used to automate a daily cleaning process for example, heavy task execution outside the database like resizing images and creating thumbnails each time an image is uploaded, communication with external API's or notification to users sending a push notification when data is modified.

For our application, we need to create a trigger on our Firestore database that will send an email to the client each time an appointment is added. Unfortunately, we cannot send an email directly via Firebase, but we can use an external API called Sendgrid.

7.4.1 Usage of cloud functions

To create cloud functions and use them we need to download node.js and the Firebase CLI. Also, the package manager npm should be installed with the version 5 or higher to handle the download of all the necessary packages.

After downloading everything, we can use “Firebase login” to login into our Firebase account and “Firebase init” to create the filesystem for our future function. During the execution of this last command users will be asked if they want to add functions to an existing project or create a new one and if they want to use TypeScript to JavaScript to create the functions.

For our application we will choose JavaScript and we can now run the following commands to install the necessary packages for this project:

- `npm install Firebase-admin@latest Firebase-functions@latest`
- `npm install @sendgrid/mail --save`

7.4.2 Sending automatic emails

As stated before, to send emails we need to use an external API called SendGrid. For this we need to create an account in app.sendgrid.com and ask for an API key. We will also for this application create a template on SendGrid that will be used to ease the creation of new emails.

After retrieving the new API key, we can save it with the following command:

- `Firebase functions:config:set sendgrid.key=[your key]`

Now we can use the following code in the `index.js` created during the “Firebase init” before:

```

exports.firestoreEmail = functions.firestore
  .document('appointments/{appointmentId}')
  .onCreate((snap, context) => {
    const db = admin.firestore()
    const appointmentId = context.params.appointmentId;

    return db.collection('appointments').doc(appointmentId)
      .get()
      .then(doc => {
        const mail = snap.data()
        const msg = {
          to: mail.address,
          from: 'contact@prettynicky.ch',
          templateId: 'd-dff2254a8f474a4cb9f8fb41c9b47fda',
          substitutionWrappers: ['{{', '}}'],
          dynamic_template_data: {
            message: mail.message
          }
        };
        return sgmail.send(msg)
      })
      .then(() => console.log('email sent!'))
      .catch(err => console.log(err.response.body))
  });

```

Figure 12 cloud functions example

The code above is a trigger that will execute each time a new document is added under appointments/ and retrieves the new id. With this information, it will retrieve the appointment information, write an email to the address of the client with the corresponding id and replace the date and time information on the template with the actual data. The email will be sent from the email contact@prettynicky.ch, which is the corporate address of the company and a registered address on the SendGrid account.

7.5 Cloud storage

With cloud storage we can save files in a “bucket”. These files are stored and available in Firebase but also via Google cloud.

All transfers of content are made over a secure connection to ensure that our data is private and, because files in cloud storage are usually more robust than in Firestore, the transfer of data will pause and resume in case of bad internet connection.

This functionality is also scalable and will allow the company to upload an unlimited number of photos to the cloud when this feature is integrated for the alpha version.

7.6 Firebase security rules

In the Firebase console, we can change the security rules of our Firestore. Without these rules, any user could make any kind of request to our database and the solution would be to have another server analysing requests before sending them to the Firebase server.

One of the most important thoughts to have in mind when working on the security of our application and database is that the users may not always do what we expect.

Each request will involve a document, either to modify, read, add or delete them. Firebase rules will, for each type of document, run a set of tests to determine if the request is allowed.

By default, all requests inside `/databases/{database}/documents` (which represents the entirety of the nodes) are blocked so to start using Firestore in our application we need to change the rules and change the read and write false to true like the following:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if true;
    }
  }
}
```

Figure 13 Firestore rules

Of course, this isn't the best solution for the security of an application but in our case, the app will only be used by one user that needs administrator level control of the database to read and write in every node of the Firestore.

If the company needs, in the future, an application for their clients, the security rules will need to be more strict, only allowing the second application to write on the appointments and clients nodes and allow users to only read their own data.

8 Frontend

After deciding the app's structure, its functionalities and the backend, we should start looking at the design of the application. As the app will be built using Flutter, we should use Material design. Of course, the easy implementation is a great advantage but also, by using a design created by Google, developers can use an already tested and recognizable layout in their applications that will make the user feel more comfortable but also subconsciously more secure. (Woodhead, 2018)

8.1 Mock-ups

These mock-ups represent the design decided for the app. The text inputs will display error messages, will behave like Material design outlined text fields and everything will follow a colour code of mostly green and grey, red for error messages and other colours to describe different appointment statuses.

The three first screens in figure 14 represent how the client management will work. In the first screen we have a list of clients in alphabetical order that can be easily filtered by name via the search bar. When clicking on a client we will be able to discover their appointments and other valuable information. When clicking the floating action button in the client list we will see a form to add new clients to the database represented in the third screen.

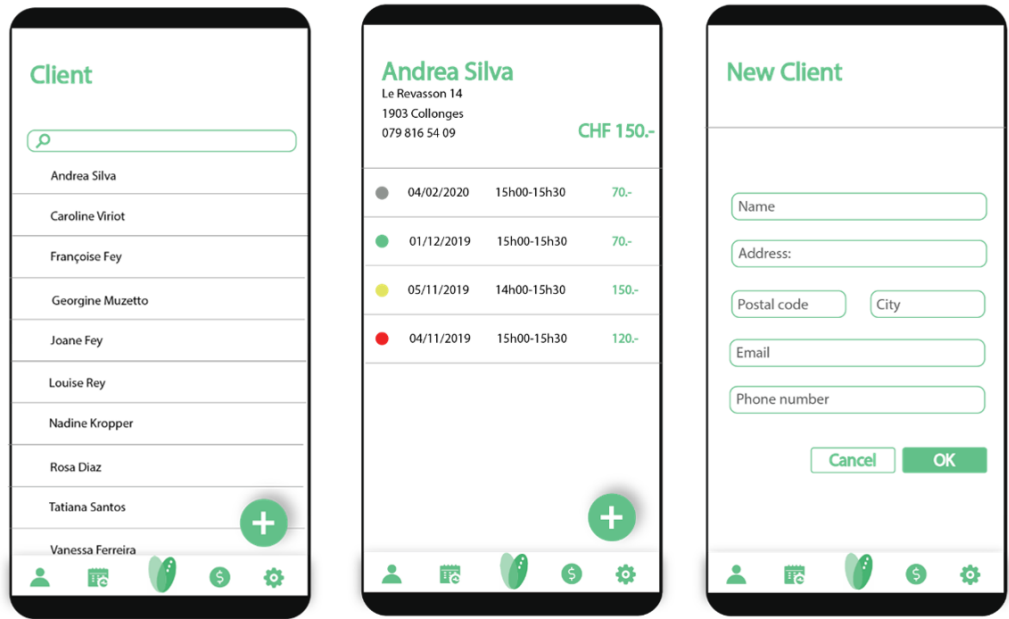


Figure 14 client management screens

Figure 15 displays the Calendar with appointments. User can click on a specific date and appointments for that day will be displayed in cards. In the next picture we can see how new appointments can be added and modified. After choosing the type of service, on the new appointment screen, the time, price and status will be automatically added.

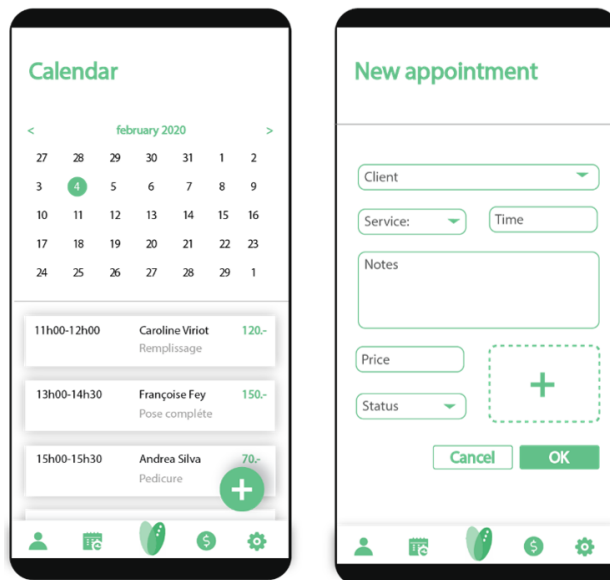


Figure 15 calendar and appointment management

Figure 16 represents the screens used by the stakeholders. The home screen is the dashboard where charts to analyse data will be presented. The two following screens represent how finance data will be presented and added. Paid appointments will also be present in the list of transactions automatically.

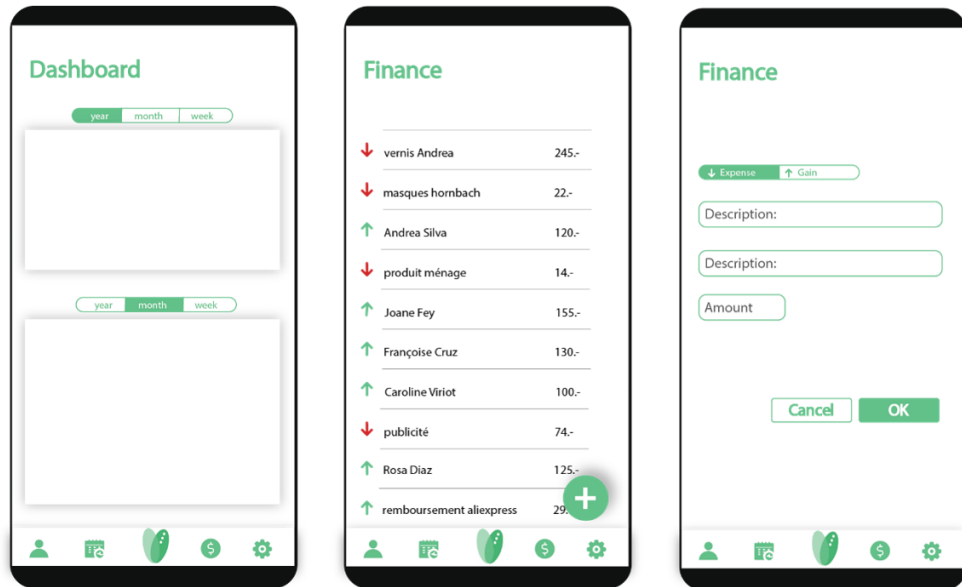


Figure 16 home screen and finance management

9 Widgets

Flutter offers multiple widgets that will be compiled into native code. To build an application we will need to learn more about some of the widgets, what kind of properties they have and how to personalize them.

Two types of widgets exist in Flutter, Stateless and Stateful. A Stateless widget doesn't hold a state and therefore never changes. Some examples of stateless widgets are Icon, IconButton or Text. (flutter, Adding interactivity to your Flutter app, s.d.)

In contrast, a Stateful widget can change on user input like Checkbox, TextField, Slider and so many others. The Stateful widget holds a state which contains all the changeable values the widget has (the text in the TextField for example). When there is a change on

the widget, the `setState()` function is called telling the framework to reload the widget with its new properties.

The following schema defines how users can navigate through the different screens and which of those screens are Stateless and Stateful widgets.

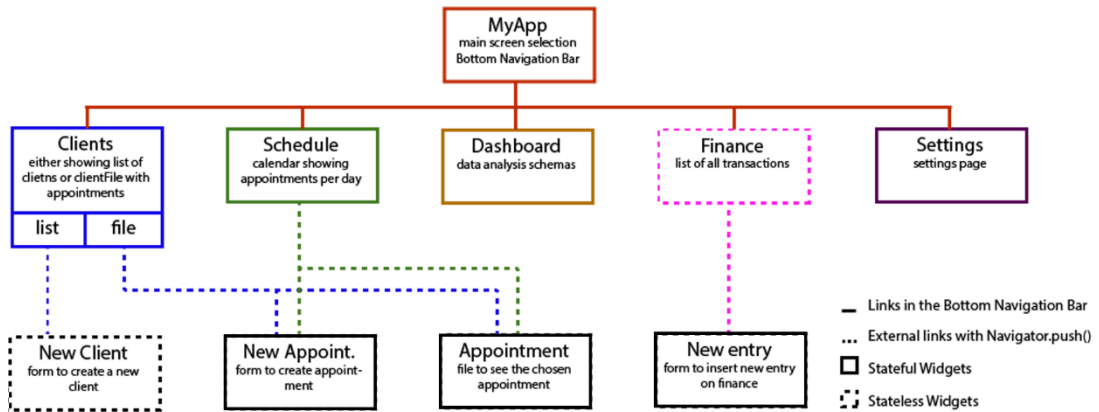


Figure 17 schema of the main screens and how they are linked

For our application we will have a main.Dart file that will contain a Stateful widget called MyApp. We will here define multiple possible screens for our application and a Bottom-NavigationBar() with five icons that will allow us to change which one of the screens is visible.

Each screen will then hold a custom Widget created in a separated Dart file. Those widgets can then inherit the class StatefulWidget or StatelessWidget. We could build an application with only StatefulWidget and change State when needed but this would mean that on every `setState()` the whole screen would be reloaded which would require more resources. The goal when developing a Flutter application is to use StatefulWidget only when needed.

The MyApp widget will be stateful because the all screen needs to be reloaded when the user clicks an icon of the navigation bar. Other screens also require stateful widget inheritance because they will change on user input. In contrast, the finance screen, New Client and Finance will be stateless. Those screens will be either forms or data visualisation without user interaction and to change from one screen to another we can simply use `Navigator.push()`. The appointment form will need to hold a state because of the automatic

inputs. When the user chooses a service, the duration and price of the appointment should be automatically filled.

9.1 Layouts

Developing a native app in Java using Android studio, developers have the possibility to create screens with a simple drag and drop of items inside the screen. In Flutter, laying out all the items can be a bit trickier sometimes. Widgets such as Textfields or Buttons need to be nested inside Containers and different types of containers have different properties.

9.1.1 Row and Column

As the name of the widget indicates, items nested in a Row or Column will be laid out either vertically or horizontally.

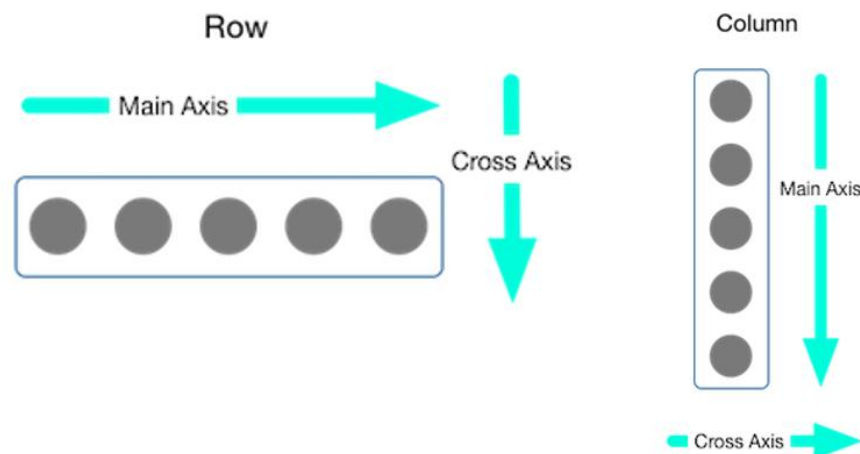


Figure 18 Example of row and column and their axis (flutter, Layouts in Flutter, s.d.)

We can go further in controlling how these widgets work with their properties. `MainAxisSize` accepts `min` or `max` as values and defines how much space the widget will occupy on its main axis (horizontally for rows and vertically for columns).

The next property is `mainAxisAlignment` and it can be used only if the previous one is set to `max`. This will allow the developer to choose how the children of the row or column will be displayed which of course won't work if they are restricted to the minimum size. The

choices are “start” to put every widget in the beginning of the parent (left or top), “end” to put everything on the end (right or bottom), “center” to put everything in the middle, “space between” to distribute the space available between each children, “space around” and “space evenly” to distribute the space available around each children.

The last property is `crossAxisAlignment` and it refers to how the widgets are placed in the secondary axis (vertical for row and horizontal for column). The possibilities for this property are start, center, end or stretch that will expand the row or column to the available space of its parent and align the widgets in the center.

Another possibility of layout with rows and columns is to fill these widgets with content that is itself nested in Expanded widgets. This will change the size of the children in a manner that fits the size of the row or column. These Expanded widgets can then have a “flex” property that allows developers to have some children bigger than the others.



Figure 19 Row with Expanded children and flex property (flutter, Layouts in Flutter, s.d.)

9.1.2 Container

Containers can only have one child widget. They are used to nest other layout widgets such as columns or rows because they have multiple properties that can be used for personalization of our application. With this widget, it's easy to add a background colour, padding, margin or borders.

9.1.3 GridView

GridView is a good Layout for a photo gallery for example, which can be useful if we create a gallery for each client. This layout will automatically create a scroll functionality if the content is bigger than the screen size.

Two types of grids are available, we can either create a GridView.count to specify the number of columns with the property crossAxisCount and automatically size the elements to fit the screen. The other option is to create a GridView.extent to specify the maximum size of each element with the property maxCrossAxisExtent and have them displayed in an automatic number of columns.

9.1.4 ListView

This widget can be used when creating list of customers, appointments or transactions. It will lay out items as a column or row and will automatically provide scrolling if necessary. This Widget doesn't allow as much personalization as a Column but is easier to use for the developers. A possibility to work with Columns would be to nest it inside a SingleChildScrollView that will add the scrolling functionality to it.

We can add ListTiles and Dividers to the ListView to add and organize our content.

9.1.5 Stack

With rows we align widgets in the y axis and with columns in the x axis. When we use the Widget Stack, we align our children in the z axis as they will be stacked together. When stacking elements, we will usually place the bigger Widget in the background and the smaller one in the foreground because as the phone screen is two dimensional, elements placed in the z axis will cover part of the ones placed before them. This could be useful when placing text in front of an image as we can see in figure 19.

Developers can use the alignment property to either align the children of the stack in the top start, top center, top end, center start, center, center end, bottom start, bottom center, bottom end or even choose ranges of value between -1 and 1 to align our widgets precisely where we need (Alignment(0,0) will be equivalent to Alignment.center. (Andrea, 2018)

```
Stack(
  children: <Widget>[
    ...
    Positioned(
      bottom: 10,
      right: 10,
      child: FloatingActionButton(
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (BuildContext context) => NewClient(),
            ),
          );
        },
      ),
      child: Text(
        "+",
        style: TextStyle(
          color: Colors.white,
          fontSize: 35,
        ),
      ),
    ],
)
```

Figure 20 Stack example from the application to position the FAB in the right bottom

Stacks can also be used with children nested inside Positioned Widgets. This will allow us to stack items which are not aligned to each other. Positioned widgets can have properties such as left, right, top or bottom to define how far from that side they will be placed. This can be useful when placing a button on the bottom right corner for example and gives more freedom to developers to place their Widgets where they want inside the Stack.



Figure 21 Example of aligned Stack (left) and Stack with Positioned children (right)

9.1.6 Material Widgets (ListTile and Card)

Card is a widget from the Material library. This widget has one single child, but it can of a Row or Column for example. It is presented as a container with slightly rounded corners and a drop shadow which can be sued to replicate the design of the appointment and charts cards in the previous mock-ups.

The card is often used with ListTiles, also from the same library. With the ListTile we can easily create a Row containing a leading, a trailing, and up to 3 lines of text.



Figure 22 two Cards with one ListTile each to display two appointments

9.2 inputs

To build form screens such as we will need to add a new appointment, a new transaction or a new client, we can use the Form widget. We could simply place TextFields and Buttons inside rows and columns but by using the Form widget, we will be able to have functionalities such as save, reset or input validation integrated with ease.

Inside a Form widget, every child should be wrapped inside a FormField or should be a widget like TextFormField which is already prepared to be integrated in it. We can then analyse the inputs and display error messages with the property “validator” and save the content inserted with the property “onSaved”.

```
validator: (input) => !input.contains('@') ? 'Not a valid Email' : null,  
onSaved: (input) => _email = input,  
)
```

Figure 23 example of validator for FormField

To validate all inputs at once, when we click the submit button for example, we can use the GlobalKey registered as a key for the form (called formKey in the example bellow) with the function “currentState.validate()” which will use all the validators of the FormFields nested inside the Form and return false if any of the validators returns false. This function can also be used to display an error message in the field that is causing the error.

```
void _submit(){
  if(formKey.currentState.validate()){
    formKey.currentState.save();
  }
}
```

Figure 24 example of form submission to trigger the validators

9.3 calendar

In the website pub.dev we can find a lot of dependencies for our Flutter application. With a quick search we can find two main types of calendar: table_calendar 2.2.3 and Flutter_calendar_carousel 1.4.12 but for our application we will use the first one.

After importing the package 'package:table_calendar/table_calendar.Dart' we can use the widget TableCalendar. This widget has multiple properties to personalize it such as calendarStyle but also accepts a calendarController and a onDaySelected to declare a function that in our case will change which events are displayed on the screen.

```
TableCalendar(
  calendarStyle: CalendarStyle(
    ...
  ),
  daysOfWeekStyle: DaysOfWeekStyle(
    ...
  ),
  calendarController: _controller,
  onDaySelected: (date, events) {
    setState(() {
      _selectedDate = DateTime(date.year, date.month, date.day);
    });
  },
),
```

Figure 25 example of TableCalendar

10 Results

During development, the design was integrated with some small changes to correspond to the Flutter widgets. To answer the needs “add clients” and “view client info” presented in the use case of figure 1, all the features to add and view clients were successfully developed and can be seen below in figure 26 with some test data.

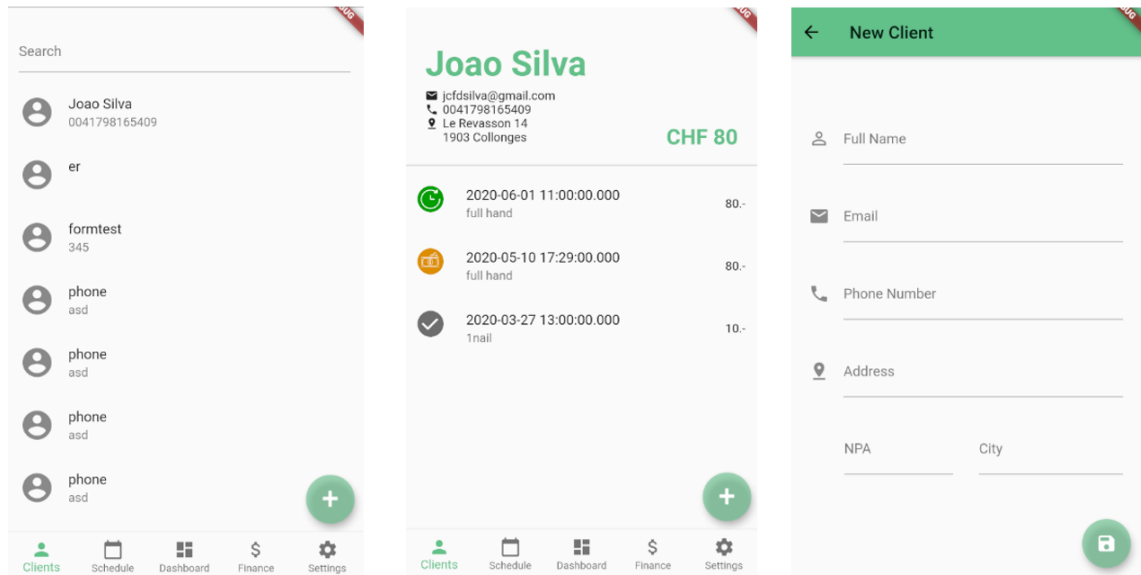


Figure 26 screen capture of the application client management screens

The schedule functionality was also successfully integrated using the table_calendar 2.2.3 mentioned before and all the screens to add or update appointments are functional. This answers to the need to “view schedule” and “add appointments” of the use case. When clicking on an appointment, a form opens with the data inserted on the fields and allows the user to change and update the information. This form can also be accessed by clicking the ListTile in the client screen. We can also here see that there is already a placeholder for the image storage that will not be part of this thesis but will be functional in future development.

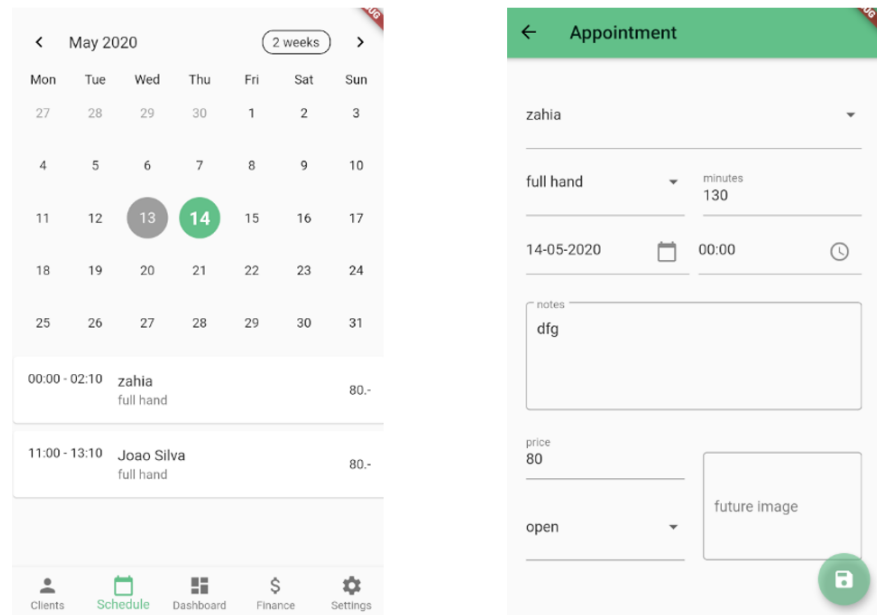


Figure 27 screen capture of the application appointment management screens

The screens related to the “manage basic accounting” needs of the use case can be seen in figure 28. When adding a transaction, the user can simply choose if it’s an expenditure or an income with the arrows. Also, paid appointments are automatically added to this screen as an income and can be accessed by clicking on the ListTile.

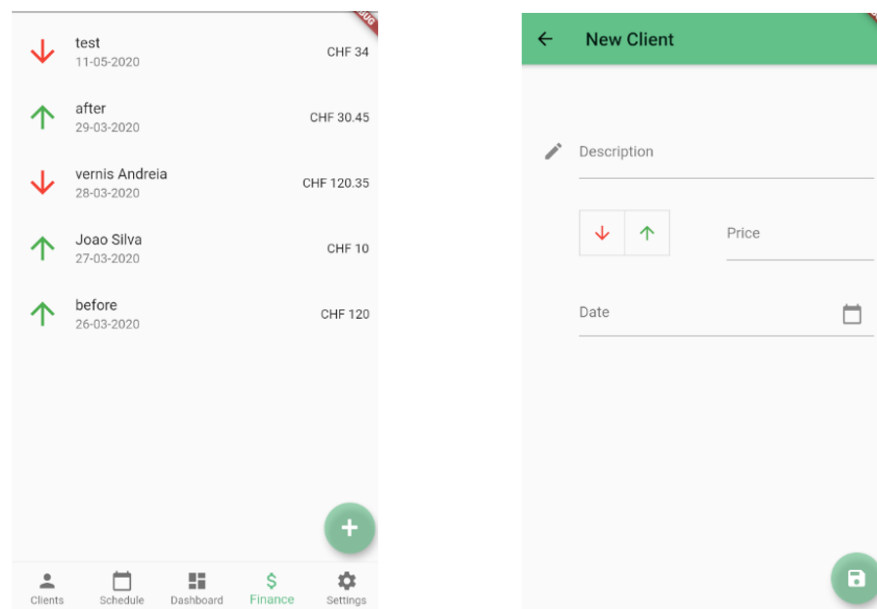


Figure 28 screen capture of the application finance management screens

In the figure 29 we can see the email sent to the client when a new appointment is added in the database. This email is sent by a cloud function that acts as a trigger and uses the API sendgrid to integrate the template and send the email.

The logo for 'Pretty Nicky' features the brand name in a black, elegant cursive font. Above the letter 'y' in 'Nicky' is a green heart-shaped icon with a white outline.



We are waiting for you!

zahia, nous vous attendons le 14 mai à 00h00
au studio Pretty Nicky à Vevey

Figure 29 Screen capture of the email sent to the client

During the development of this application some features of the use case were completely developed. The features “2. View schedule”, “3. View client info”, “4. Add appointments”, “5. Add clients”, “7. Manage basic accounting” and “9. Send automatic emails” were considered more important and are now available to test in the beta version.

Because of some management problems in the company during the conception of this thesis, the features that needed more input from the sponsor were postponed. The features “1. View data analysis charts”, “6. Store photos in cloud” and “8. Access accounting” were partially developed. Although, the frontend and backend of the beta version were build to easily implement those three features in the next version.

11 Conclusion and future development

In this section we will discuss about some future features to be implemented in the developed application. As the final product is a Beta version, some ideas are still in mind for the future development and not all issues were completely solved.

11.1 Future development

As stated, the app developed for this thesis is a beta version. This app already includes most of the functionalities needed but still needs to include some more.

The user of this app can now easily add and retrieve data from the database, but the deletion and update of data is not possible for all nodes. This can be done simply by adding a delete button after a long click on a ListTile or an update button on the screen. When deleting documents in “services” or “clients” nodes we should take into consideration that when retrieving appointments there could be a problem with data integrity and the best option would be to delete also the appointments related to them.

Also, the issue about image storage is not yet resolved but the Firebase cloud storage is already configured and a place holder for this functionality is already present in the appointment form. Still some discussion with the company is needed to discover if one picture per appointment is enough or if a general photo gallery or one per client should be added to the app. The dashboard functionality with charts for data analysis will also be developed during a future iteration, discussing with the company to better understand their needs.

Moreover, the app will need a phase of testing to ensure it is reliable. This is important for any application but especially for one that involves sensitive data that cannot be lost or misplaced. Flutter gives us the possibility to create unit tests to test functions and methods and widget test to test the widgets behaviour after some types of input. Firebase also has a lot of testing functionalities such as Crashlytics that collects and organizes problems in the application and helps understand which issues are more important. All these tests will be important to implement but especially the ones in the database.

Furthermore, a possible feature for the future of this application could be the possibility for others to connect and retrieve information needed. Not only could the stakeholders retrieve insights of the business but also a client-side application could be developed and will use the same Firebase database.

To finish, even if the app is tailored for only one user, a small demonstration will be needed to ensure the app will be used in the correct way. Past data can also be included in the database if the sponsor company desires, either by the developer or by some external person.

11.2 Conclusion

The topic of this thesis was to develop an application to improve the quality and productivity around the management tasks of a nail salon based in Switzerland. During this project we discovered more about new technologies such as Flutter and Firebase which compared to its competitors are very recent in this domain.

During the development we discovered a lot of Flutter widgets and functionalities of Firebase. Having developed mobile apps in React Native and Flutter, I have the feeling that the development process for the second is a bit faster and its very easy to find precise information about specific widgets. Developing applications in the future I will probably use the same technologies because they are very recent and already very powerful which could mean that they will improve even more during the next years.

Also, now that I am more used to Flutter, I would maybe divide my application into more small widgets that can be placed inside a bigger one instead of developing a whole screen per Dart file. This way, the code could be more structured, and widgets could easily be reused in the future.

The first version of the application has still some features that need to be added and improved but the sponsor company is satisfied with the result as of today. Building an application in Flutter can be faster than building it natively but other parts need also to be developed such as the use of external API's to send emails or the use of Firebase Firestore, cloud functions and cloud storage. Not only all these tasks take some time but also the study of which functionality to develop, how to develop it and which tools we should use,

consume some time that can postpone the deadline. With some effort from both parties, the deadline of the 10th of May was respected, and the app can be installed on the company's phones and tablets in June.

Even if the final result is not an Alpha version, which was difficult to achieve due to the complexity of a management app that manipulates sensitive data, the Beta version is finished and can already be tested by the company to ensure no errors are discovered in the already made available features.

Using an Agile development approach, we can now continue the development by adding features one by one until the app is finished and the client is completely satisfied. At this moment, the testing of this version as not yet started but the design was completely accepted by the sponsor company. Some minor bugs were already corrected and probably some others will appear as well during the following weeks.

As mentioned before, a client-side application is also an interest for this company in the future to automate all management processes. This side of the application will use the same database, with more restrictions to security, and will probably be developed as a PWA. That will make it easy to find for the clients but also will provide notifications for future appointments which are more interactive than the emails sent by the version developed during this project.

The conclusion of this thesis also provided more experience and knowledge, not only with some recent technologies that might be useful to the future, but also with having a permanent contact with a company during the development of an application. During the conception of this software the sponsor company, Pretty Nicky, was always informed of what was happening, either with contact with the owner Nadia Silva or the main contact person and marketing expert Rui Gonçalves.

12 References

Academind. (2018, July 2). *React Native vs Flutter*. Retrieved from Youtube:

<https://www.youtube.com/watch?v=bnYJRYFsrSw>

Andrea, C. W. (2018, May 26). *Flutter Layouts Walkthrough: Row, Column, Stack, Expanded, Padding*. Retrieved from youtube:

<https://www.youtube.com/watch?v=RJEnTRBxaSg>

Chheda, D. (2016, September 7). *What are the difference between web app and mobile app?* Retrieved from Quora: <https://www.quora.com/What-are-the-difference-between-web-app-and-mobile-app>

cloud_firestore 0.13.5. (2020, April 14). Retrieved from pub.dev:

https://pub.dev/packages/cloud_firestore

Developers, G. (2018, June 5). *Using Firestore as a backend to your Flutter app*.

Retrieved from youtube: https://www.youtube.com/watch?v=DqJ_KjFzL9I

Execution failed for task ':app:mergeDexDebug'. Firestore | Flutter. (2020, Mars).

Retrieved from stack overflow:

<https://stackoverflow.com/questions/60310873/execution-failed-for-task-appmergedexdebug-firestore-flutter>

facebook. (n.d.). *react-native*. Retrieved from github: <https://github.com/facebook/react-native>

Firebase. (2018, January 22). *What's the Difference Between Cloud Firestore & Firebase Realtime Database?* Retrieved from youtube:

<https://www.youtube.com/watch?v=Kelx-mArUck>

firebase. (n.d.). *Choose a Database: Cloud Firestore or Realtime Database*. Retrieved from firebase: <https://firebase.google.com/docs/database/rtdb-vs-firestore>

- firebase_auth 0.16.0*. (2020, April 13). Retrieved from pub.dev:
https://pub.dev/packages/firebase_auth#-installing-tab-
- flutter. (n.d.). *Adding interactivity to your Flutter app*. Retrieved from flutter:
<https://flutter.dev/docs/development/ui/interactive>
- flutter. (n.d.). *flutter*. Retrieved from github: <https://github.com/flutter/flutter>
- flutter. (n.d.). *Layouts in Flutter*. Retrieved from flutter:
<https://flutter.dev/docs/development/ui/layout>
- flutter. (n.d.). *Windows install*. Retrieved from flutter: <https://flutter.dev/docs/get-started/install/windows>
- Fowler, K. (2019, May 01). *Average Website Maintenance Costs in 2019*. Retrieved from volusion: <https://www.volusion.com/blog/website-maintenance-costs/>
- fresha. (n.d.). *Fresh - Onstantly book salons and spas nearby*. Retrieved from <https://www.fresha.com/>
- Gregg, K. (2019, July 15). *Why isn't visual studio an IDE*. Retrieved from Quora: <https://www.quora.com/Why-isnt-Visual-Studio-Code-an-IDE>
- inducesmile. (n.d.). *HOW TO ADD DATA TO FIRESTORE DATABASE IN FLUTTER*. Retrieved from Induce smile: <https://inducesmile.com/google-flutter/how-to-add-data-to-firestore-database-in-flutter/>
- Kerpelman, T. (2017, October 3). *Cloud Firestore vs the Realtime Database: Which one do I use?* Retrieved from Firebase: <https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html>
- Melendaz, S. (2014, may 05). *Sometimes You're Just One Hop From Something Huge*. Retrieved from fastcompany: <https://www.fastcompany.com/3031109/sometimes-youre-just-one-hop-from-something-huge>

Milijic, M. (2019, november 15). *CRAZY Android vs iOS Market Share Discoveries in 2020*. Retrieved from lefronic: <https://lefronic.com/android-vs-ios-market-share/>

Nader, Y. (2020, April 28). *React Native vs Flutter*. Retrieved from hackr.io: <https://hackr.io/blog/react-native-vs-flutter>

Ninja, T. N. (2019, October 21). *Flutter & Firebase App Tutorial #2 - Firebase Setup*. Retrieved from youtube: <https://www.youtube.com/watch?v=Wa0rdbb53I8>

O'Hear, S. (2019, April 11). *Shedul, the booking platform for salons and spas, raises \$20M Series B at a \$105M valuation*. Retrieved from techcruch: <https://techcrunch.com/2019/04/11/shedul-b-round/?guccounter=1>

Siripathi, S. (2017, july 24). *Mobile Development Languages*. Retrieved from tutstplus: <https://code.tutstplus.com/articles/mobile-development-languages--cms-29138>

Vagaro Reviews & Product Details. (n.d.). Retrieved from g2: <https://www.g2.com/products/vagaro/reviews>

Vello. (n.d.). *Vello - Online Appointment Schedulling Calendar*. Retrieved from <https://vello.fi/en>

Woodhead, W. (2018, April 11). *Should you use Material Design?* Retrieved from medium: <https://medium.com/pilcro/should-you-use-material-design-bfb596a04bae>