Object-Oriented Architecture and Portable Computing
-
Assignment Report

Athanasios Argyropoulos

E22012

Table of Contents

Assumptions

The following project has a few assumptions built in to it. More specifically:

1) Only the UI's language needs to change. Contents of the database don't.

2) The providers have already submitted their information, as well as their services to the app.

3) The summary created in the appointment history adjusts depending on the filtered appointments, and can still show statistics, even if the time range selected is less than a month.

Introduction

The following application is a university assignment meant to introduce the student to the technologies and techniques of android development, such as android emulators, Android Studio IDE and languages targeted towards android development. More specifically, the purpose of this assignment is to develop an appointment application for users to manage, view and book various kinds of appointments using either the Kotlin or the Dart(Flutter) language in Android Studio IDE.

Aside from the familiarization with the android technologies, this assignment also requires the use of a BaaS (Backend as a Service), such as Firebase and Appwrite, in order to get a better look into the serverless architecture that is getting more and more popular.

App Description

The application will allow the user to organize their appointments, track their history, and have a comprehensive overview of the time and cost of the services they utilize. Specifically, the application will include the following:

1. User Login: Through an authentication mechanism, users must be able to register and log in to the system.

2. Service Catalog: Display of services organized by category. For each service, the following will be displayed: service name, category (health, wellness, technical services, education, etc.), a brief description, estimated duration, and indicative cost.

3. Service Details: Detailed view of elements such as a full description, list of providers (e.g., specific doctor or establishment), appointment duration (minimum-maximum duration), and cost.

4. Appointment Management: The following must be supported:

a. Creation of a new appointment for a specific service, with a specific provider selected by the user from the provider list, at a date and time of their choice (selected from a list of available days and times). Here, the user will be able to add notes if desired.

b. Editing / cancellation of appointments.

c. View appointments by day, week, and month.

5. Appointment History: View past appointments with filtering by service category and time period. A summary must also be provided to the user, including the total number of appointments, total appointment time, and total cost per time unit (month, year).

6. User Profile: Management of user details such as first name, last name, date of birth, etc.

7. Multilingual Support (Localization): Support for two languages (Greek and English).

App Development

For this assignment, I chose to use Dart(Flutter) in combination with Firebase. In the following sub-chapters, we will go over the Firebase setup, as well as the core logic behind the application.

Firebase Setup

In order to start the application, we have to setup the Firebase project.

- Login in to our Firebase account and enter console

- Create a new Project

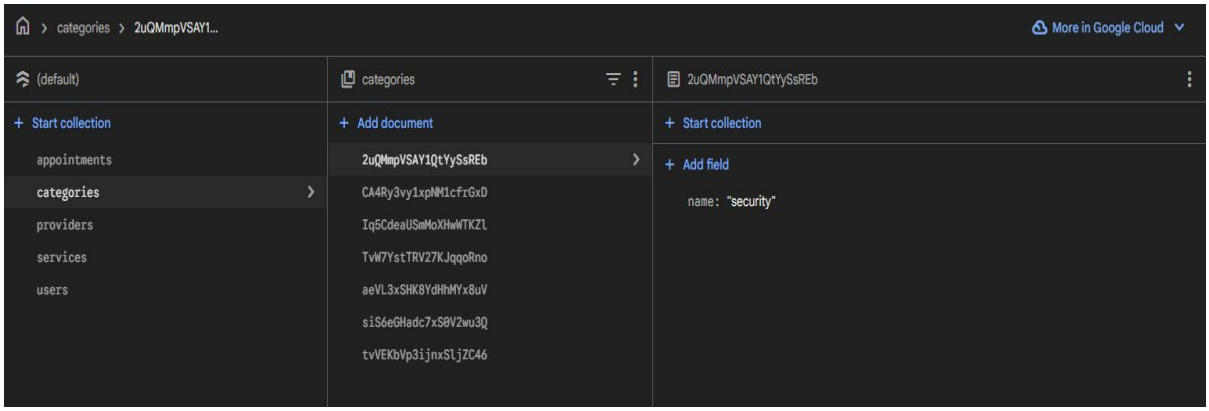
- Go to `Authentication` tab and select the email/password method.

- Go to `Firestore Database` tab and create a new database.

- Create the following empty Collections with an id the same as their respective names:

I) users: Where we store user data,
II) services: Where we store service data,
III) providers: Where we store provider data,
IV) categories: Where we store category data,
V) appointments: Where we store appointment data.

- Add some documents to the category Collection, where each one only has a `name` field. Example:



- Add 1-2 documents to the providers Collection with the following structure:

Field	Type
name	String
phone	String
email	String
bio	String
address	String
Schedule	Map<Array<Map<String>>>*

* The Schedule is a Map of arrays. Every array represents a day of the week. Also, every array contains 1 or more {End: *String*, Start: *String*} Map.

- Add 1-2 documents to the services Collection with the following structure:

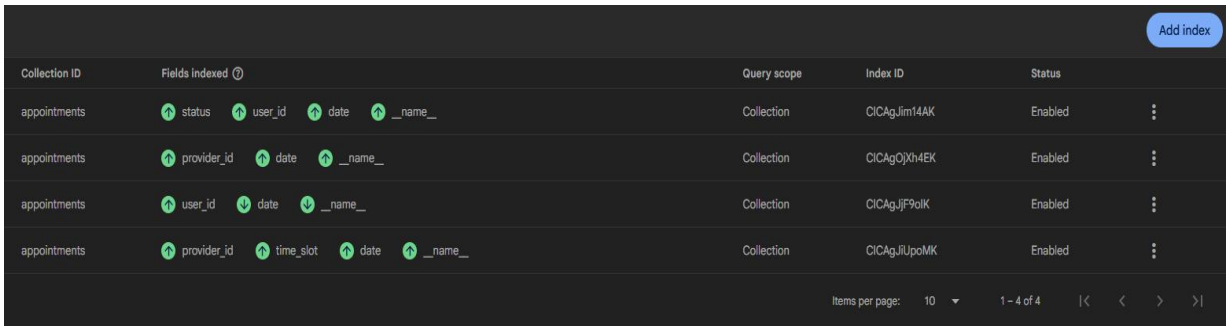
Field	Type
Cost	Number
Duration	Number
Full Description	String
Short Description	String
category	String
name	String
providers	Array<Map<String>>*

* The providers is an Array of Map<String, String>. Every block of the array represents a provider with their details.

- Go to the application's documents: `test_app/lib/firebase_options.dart` and configure all keys and IDs according to your Firebase project.

- In order for the app to run smoothly, we also need to install the following indexes:

Collection ID	Fields Indexed	Query scope
appointments	**status** (Ascending) **user_id** (Ascending) **date** (Ascending) **__name__** (Ascending)	Collection
appointments	**provider_id** (Ascending) **date** (Ascending) **__name__** (Ascending)	Collection
appointments	**user_id** (Ascending) **date** (Descending) **__name__** (Descending)	Collection
appointments	**provider_id** (Ascending) **time_slot** (Ascending) **date** (Ascending) **__name__** (Ascending)	Collection



- Lastly, go to the `Rules` tab of the Database section and place the following Rules:

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {

    // User Profiles
    match /users/{userId} {
      allow read, write: if request.auth != null &&
request.auth.uid == userId;
    }

    // Services
    match /services/{document=**} {
      allow read: if request.auth != null;
    }

    // Provider
    match /providers/{document=**} {
      allow read: if request.auth != null;
    }

    // Appointments
    match /appointments/{document=**} {
      allow read: if request.auth != null;
      allow write: if request.auth != null;
    }

    // Categories
    match /categories/{document=**} {
      allow read: if request.auth != null;
    }
  }
}
```

App Code

Main.dart ----------

This is the start of the program and its base.

1. *Imports*: Here, we have all of the necessary imports for the current file to run smoothly.

2. *Main:* Which is the function that starts Flutter, connects to the Firebase based on the platform used and renders the root widget `RootApp()`.

3. *RootApp*: This is the widget that wraps the whole app `MyApp()` with 2 providers:
i) *LocaleProvider*, responsible for connecting to the Localization files, which can be found in the `lib/l10n` folder and `providers` folder, through Flutter's innate localization function.
ii) *User Stream*, a firebase stream used to constantly "watch" the Auth User state. If a User logs in or out, the stream will notice.
With this structure, any screen in the app can have access to localization and the current Auth status(if someone is logged in or not).

4. *MyApp:* This is the widget responsible for setting the actual app and its Navigator.
i)First, it gets access to the localeProvider and initializes all necessary settings for the localization to be achieved inside the app.
ii) Then, it gains access to the Auth User Stream and listens for when a user is logged in/out.
iii) Inside the `builder` method, if there is no logged in user, it returns the Navigator as is, otherwise, it wraps it inside a 2nd stream that listens for the logged in user's data in the database. The stream retrieves the user's data and formats them to follow the userModel that can be found in the `lib/models/` folder. This way, all screens in the app, have access to the user's data in the same format.
iv) Lastly, if there is no logged in user, the widget proceeds to render the Login screen, otherwise, it renders the Home screen.

Key Logic:

main
|--> RootApp
|--> Providers
|--> MyApp
|--> User Data Stream (Omitted, if no logged in user)
|--> Navigator
|--> Home Screen (or Login, if no logged in user)

login_screen.dart ----------

This is the Login screen, this is the first screen that a user sees when opening the app, unless they are logged in from before, then they are transported to Home screen.

1. *Imports:* Here, we have all of the necessary imports for the current file to run smoothly.

2. *LoginScreen:* The main stateful widget being rendered, it creates the state `_LoginScreenState()` in order to display information.

3. *_LoginScreenState*: After initializing all necessary states, it starts building the widget.

i) It gains access to localization through the global stream, which can be used to switch between English and Greek using the appbar button.
ii) It displays a form with the `email` and `password` fields. Each field corresponds to a state.
iii) When the data are submitted, the login function will be called and the login button will turn to a spinner, indicating that the process has started.
iv) The login function will attempt to login the user with the provided credentials in the states using the innate Auth function of Firebase. If the login is successful, the Auth stream wrapped around the app will hear it, and will fetch the Auth user uid. Based on that, the 2nd stream that listens for a user's Data in the DB is now initialized and wrapped around the app. If the login failed, the user is informed.
v) Since now we have a logged in user, the root app renders the Home screen.

registration_screen.dart ----------

This is the Registration screen, it can only be seen through the link in the login form.

1. *Imports:* Here, we have all of the necessary imports for the current file to run smoothly.

2. *RegistrationScreen:* The main stateful widget being rendered, it creates the state `_RegisterScreenState()` in order to display information.

3. *_RegistrationScreenState*: Similarly to Login, after initializing states, as well as a controller for the birthday, it starts building the widget.

i) It gains access to localization, which can be used to switch between English and Greek using the appbar button.
ii) It displays a form with the `Name`, `Surname`, `Birthday`, `Phone`, `Email`, `Password` and `Confirm Password` fields. Each field corresponds to a state.

iii) When tapping on the birthday field, the calendar function will be triggered, displaying a calendar that awaits the user to select a date.That date is then shown to the field through a controller.
iv) When the user taps on the Register button, the program checks for whether the 2 passwords match and if all fields are filled. If any of those conditions are false, an error message pops through a snackbar, otherwise the register function is called.

4. *Register:* Attempts to create an Auth user in the Firebase project. If the creation was successful, it creates a document in the users collection and the user is informed for the outcome. If the operation was successful, the User is transported to the Home screen.

home_screen.dart ----------

This the Home screen, it can only be seen if the user is logged in. All other screens for the logged in User have to follow after Home screen.

1. *Imports:* Here, we have all of the necessary imports for the current file to run smoothly.

2. *HomeScreen:* The main stateful widget being rendered, it creates the state `_HomeScreenState()` in order to display information.

3. *_HomeScreenState:* After initializing the states, we use initState() in order to fetch the categories once upon entry to the screen. The data were formatted into a dropDownMenuItem list, so that it can be used later. Now it starts building the widget.
i) It gains access to localization, which can be used to switch between English and Greek through a button in the drawer.
ii) It retrieves the user's data through the 2nd stream in order to display their name in the drawer.
iii) The drawer generally contains buttons in order to push new screens in front for functions that a logged in user can use, such as see their Profile, their appointments, their appointment history, change language and logout.
iv) The Home screen itself is the place that allows a user to book an appointment. More specifically, it places all fetched categories from the start inside a dropdown list. Below the dropdown, is a stream that listens for services in the database with a category == selected category from the user. If the user clicks a category that has services, the stream will immediately bring them to the program.
v) When the program has services to display based on a category, it will display another dropdown list with their names, otherwise it prints a "No services found" message.
vi) When the user selects a service from the new dropdown, the program will search for the name of the service in the stream output, store it's data and display it's name, short analysis and full analysis under the dropdowns. Furthermore, the providers for that service, which are kept in that service's data, will also be listed in a dropdown list with their names.

vii) When a provider is selected, their relevant data will be displayed from the selected service's data, such as their name, specific cost and duration that they have listed.
viii) After that, and only after that, will the `Book Appointment` button be displayed, which will transfer them to the Booking screen.
ix) The states are only nullified when the Booking screen returns true, meaning, the booking was successful.

booking_screen.dart ----------

This is the Booking Screen, it can only be seen after the `Book Appointment` button has been tapped in the Home Screen.

1. *Imports:* Here, we have all of the necessary imports for the current file to run smoothly.

2. *HomeScreen:* The main stateful widget being rendered, it creates the state `_BookingScreenState()` in order to display information.

3. *_BookingScreenState:* After initializing all states and controllers, the app displays the Service's and Provider's details, which it acquires through it's parameters when the widget was called. On top of the details the screen already had about the provider, it will also initiate a fetch to the `providers` collection at the start of the body, in order to get more details on the provider. If the provider's data was not found, the whole page will show an error.
i) Additionally, another fetch will occur in order to get all future appointments of the provider.
ii) The user can then write some notes for the appointment.
iii) The user can also can tap on the date field in order to select a date. Similarly to the registration page, a calendar will show up awaiting the selection of a date.
iv) When a date has been selected, if the provider's schedule contains a time range on that day, meaning they are available for that time range (ex. "14:00 - 18:00"), the function _generateTimeSlots will run.
v) The function will convert the time range string into DateTime with the selected date as the base, get the service's max duration and start counting the number of appointments the provider can fit into that time range(ex. 14:00 - 15:00 and duration=40min: only one appointment 14:00 - 14:40).
vi) After that, the function will exclude all that are reserved by other appointments of the provider (fetched at the start). If the appointment is for today, then time slots focused for earlier as now are also removed (ex. right now it's 15:00, so appointments before 15:00 must not appear).
vii) Lastly, the time slots are shown to the user in order for them to pick a time range for their appointment. Lastly, the Confirmation button is shown.
viii) However, it is possible, that 2 users select the same service and provider at the same time, thus they see the exact same appointment time slots. Right before the program creates a new appointment in the database, it has to fetch again all appointments for the provider, which refer to the selected date, and check if the desired time slot was taken. If it wasn't, the appointment in created, otherwise, a snackbar error is shown.

my_profile_screen.dart ----------

This is the screen that displays the user's profile. It can only be accessed through the Home screen's Drawer.

1. *Imports:* Here, we have all of the necessary imports for the current file to run smoothly.

2. *MyProfileScreen:* The main stateful widget being rendered, it creates the state `_MyProfileScreenState()` in order to display information.

3. *_MyProfileScreenState:* After initializing all states and controllers (it is important to use controllers here, so that the fields can display pre-installed and easily-adaptable information), it displays a container with all of the given user's information in textfields. All fields, except the email, can be updated.

i) The app connects to the localeProvider for localization.

ii) Firstly, the program fetches the user's data from the 2nd stream wrapped around the app and initializes the controllers with that information inside the `didChangeDependencies()`. We use `didChangeDependencies()` because initState() doesn't have access to the context yet, which is needed in order to get the user data.

iii) The birthday field calls the calendar function when clicked in order to get a new date.

iv) When the user taps on the `Save Changes` button, the saveProfile function will be triggered, which updates the user data in the database through a call. The 2nd stream wrapped around the app will listen to that change and automatically update the userModel (app's formatted information about the user).

my_appointments_screen.dart ----------

This is the screen that displays all of the user's future appointments. It can only be accessed through the Home screen's Drawer.

1. *Imports:* Here, we have all of the necessary imports for the current file to run smoothly.

2. *MyAppointmentsScreen:* The main stateful widget being rendered, it creates the state `_MyProfileScreenState()` in order to display information.

3. *_MyAppointmentsScreenState:* After initializing all states, initSate() initializes fetches all appointments from the appointments Collection based on the user's ID. The app displays segmented button at the top, which can be used to filter the calendar's view based on days, weeks, months or just see all of the appointments together. A button on the appbar can be used to reset the selected day to today.

i) The app connects to the localeProvider for localization.

ii) At the start, the app calls the filteredAppointments getter, which returns filteredAppointments based on the selected filter. Because the default filter is `All`, it returns all of them.

iii) When the user selects a filter, like `day` or `week` or `month`, the selected filter changes and because it is a state, it triggers the recalculation of the filtered Appointments.
iv) If the filter is `day`, then the getter returns all fetched appointments it has that match the selected specific calendar day. If the filter is a week, the it finds all appointments between the current day and 7 days from it. If the filter is a month, then it finds all appointments that match the specific calendar month (yyyy/mm).
v) The user can also navigate through the calendar through the arrows below the segmented button. When the left arrow is clicked, it will set a step of -1 and call navigateDate() function. If the right one is clicked, it will set a step of 1. Depending on the filter and step, navigateDate() will add or subtract duration from the current day (-1 day, +1 * 7days, -1 * 7days).
vi) Additionally, appointments shown can be changed or canceled at least 12 hours before the appointment's time. If the user taps on the change button, I twill transfer them to the booking screen in order to book a new appointment with the selected service and provider of the old appointment. If the user selects a date, a time slot and completes the procedure, the old appointment will be removed from the database.
vii) If the user taps the cancel button, it will pop a small alert window to confirm cancellation. If the user consents, the program will delete the appointment from the database.

appointments_history.dart ----------

This is the screen that displayes the user's appointment history. It can only be accessed through the Home screen's Drawer.

1. *Imports:* Here, we have all of the necessary imports for the current file to run smoothly.

2. *AppointmentsHistoryScreen:* The main stateful widget being rendered, it creates the state `_*Appointments*ScreenState()` in order to display information.

3. *_AppointmentsHistoryScreenState:* After initializing all states, it displays a container with textfields for filters based on category and date, as well as the appointments for those filters. The filtered appointments also get a small summary.
i) The app connects to the localeProvider for localization.
ii) initState() fetches all service categories from the database, as well as all of the past appointments of the user from the respective Collections.
iii) The app calls the getter applyFilters which returns appointments based on category and date. If an appointment fits the category and/or date range selected, it returns it.
iv) With the filtered appointments, it calculates the amount, total duration and cost per time unit of them. And displays it in a small container below the filters.
v) All containers are in a scroll view for easier management of space.

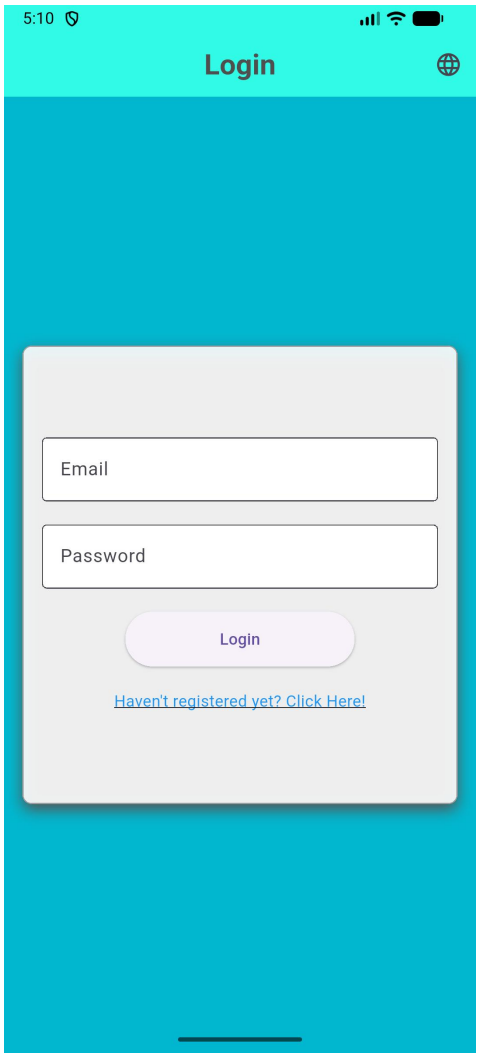
User Manual

In this chapter, we will go over the utilities of the app, as well as how to navigate through all the different screens in it for the ordinary user.

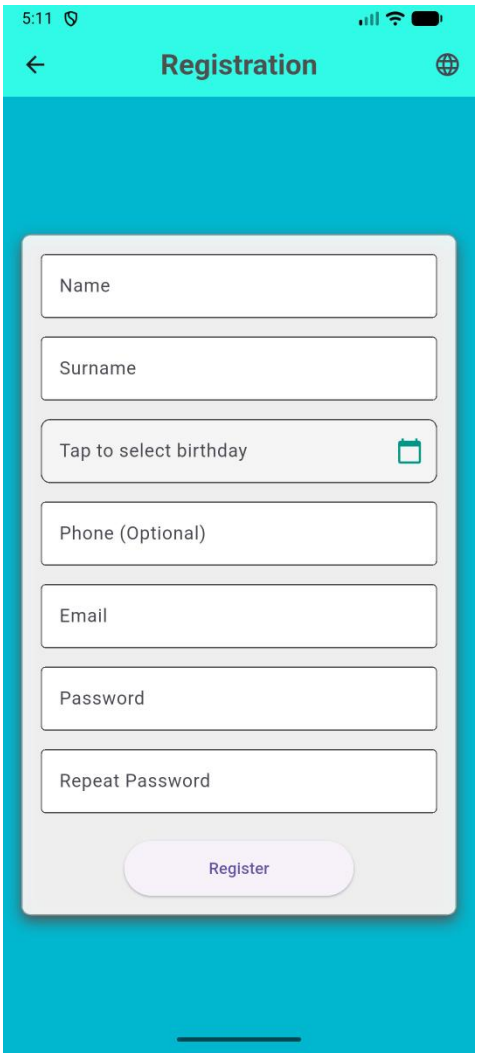
Login / Registration

When the User first opens the app, they will encounter the Login screen, in which they can type their credentials to enter the Home page of the app. Since they do not yet have an account on the app they will need to create one by clicking on the ***"Haven't registered yet? Click Here!"*** link, which will transport them to the Register page.

In case they need to change language, both pages have a small globe button at the top right that will switch between English and Greek.

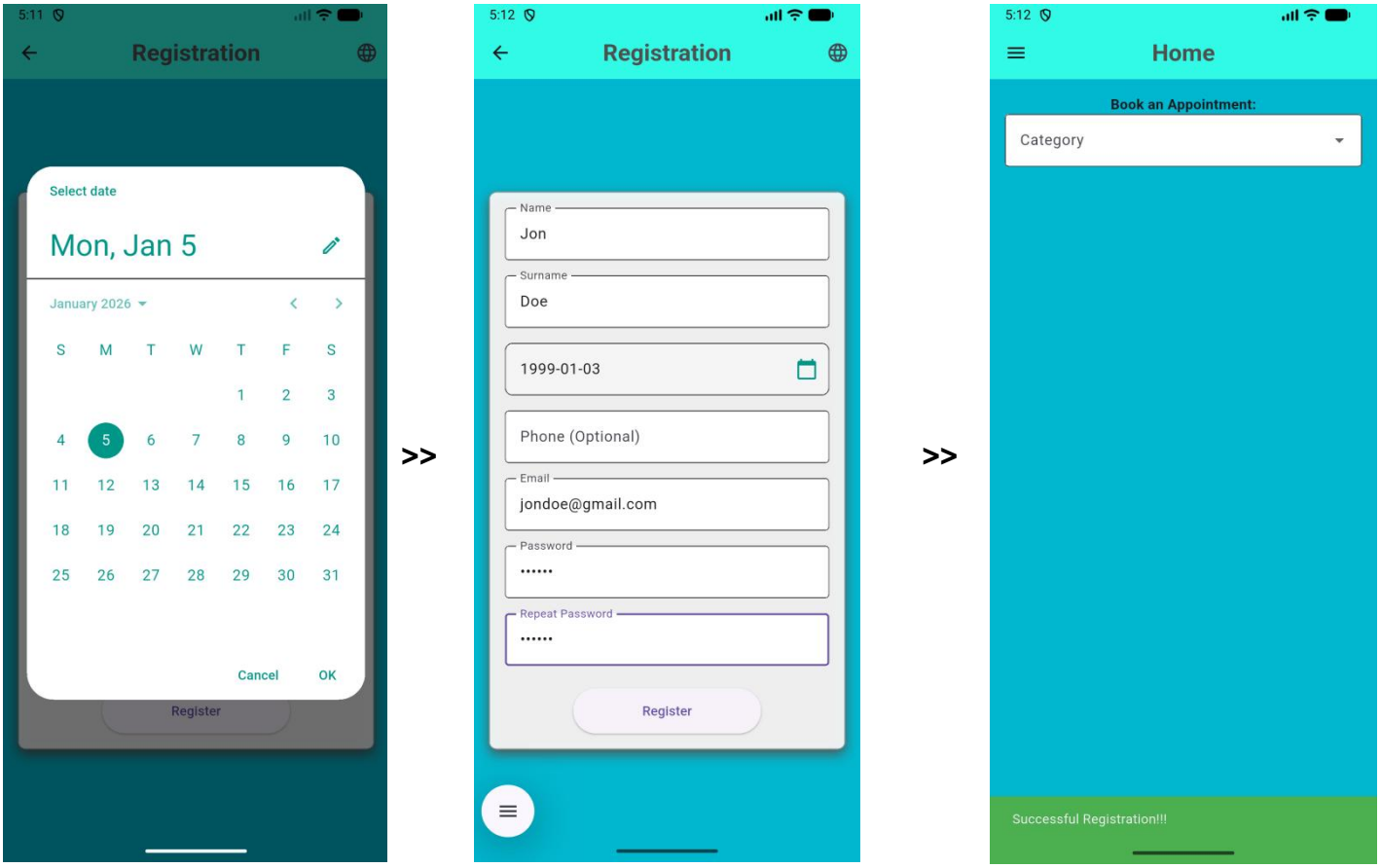


>>



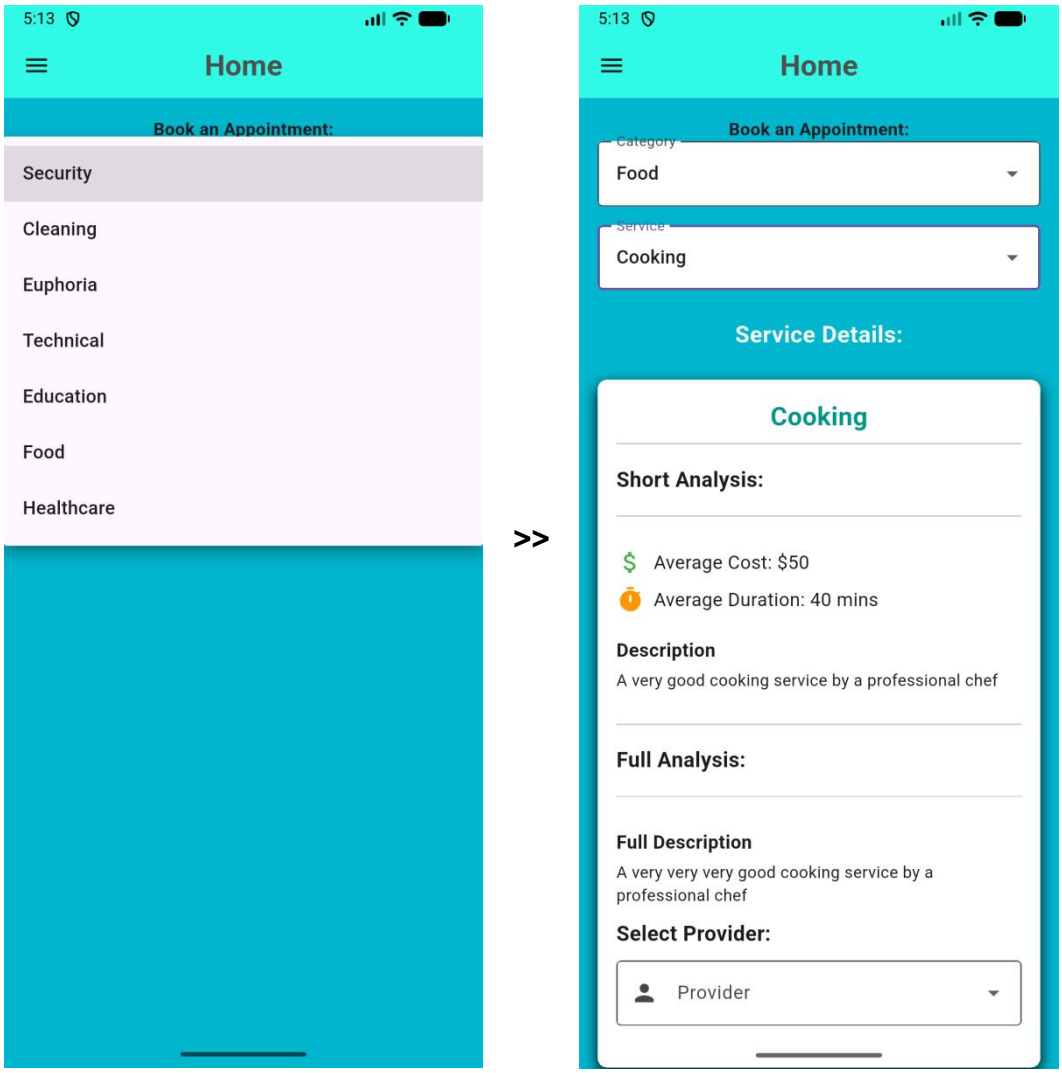
Once there, the User will have to fill out a form with their information. All fields are mandatory, except from the *phone* one. The birthday field has to be filled by clicking on it and selecting the desired calendar day from the calendar that will appear. The 2 passwords given must also match, otherwise, an error will be shown.

After the User clicks the Register button, if all conditions are satisfied, they will be transported to the Home page of the app.

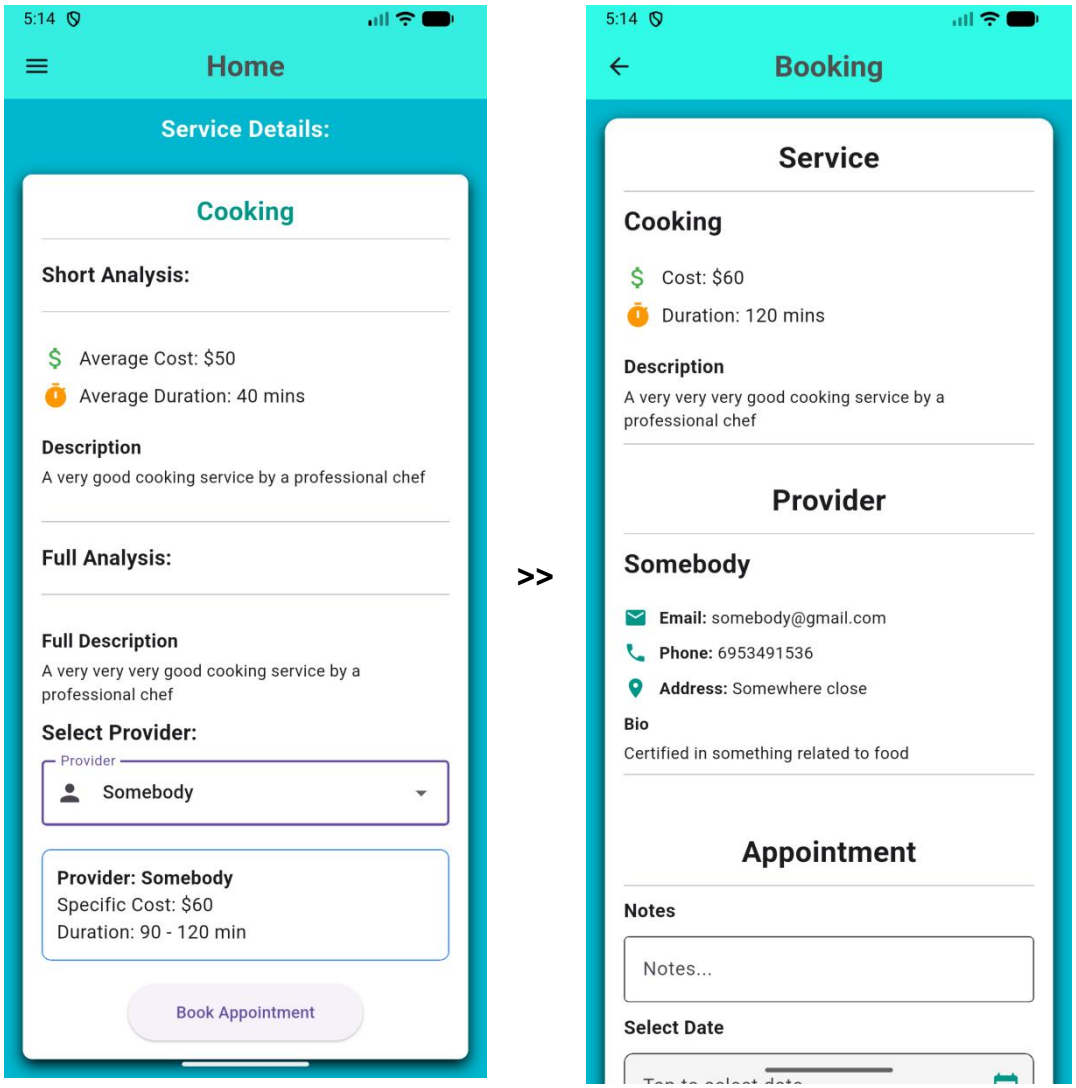


Appointment Booking

In the Home screen of the app, the user is able to select a category from the *Category Dropdown.* After selecting one, if there are services for that category, a new Dropdown will appear containing all of the services in that category. The User can select one of them and see their details, like the usual cost and duration for that service, a short description, as well as a full analysis that contains the full description and the list of providers. One category can have many services, and one services multiple providers.

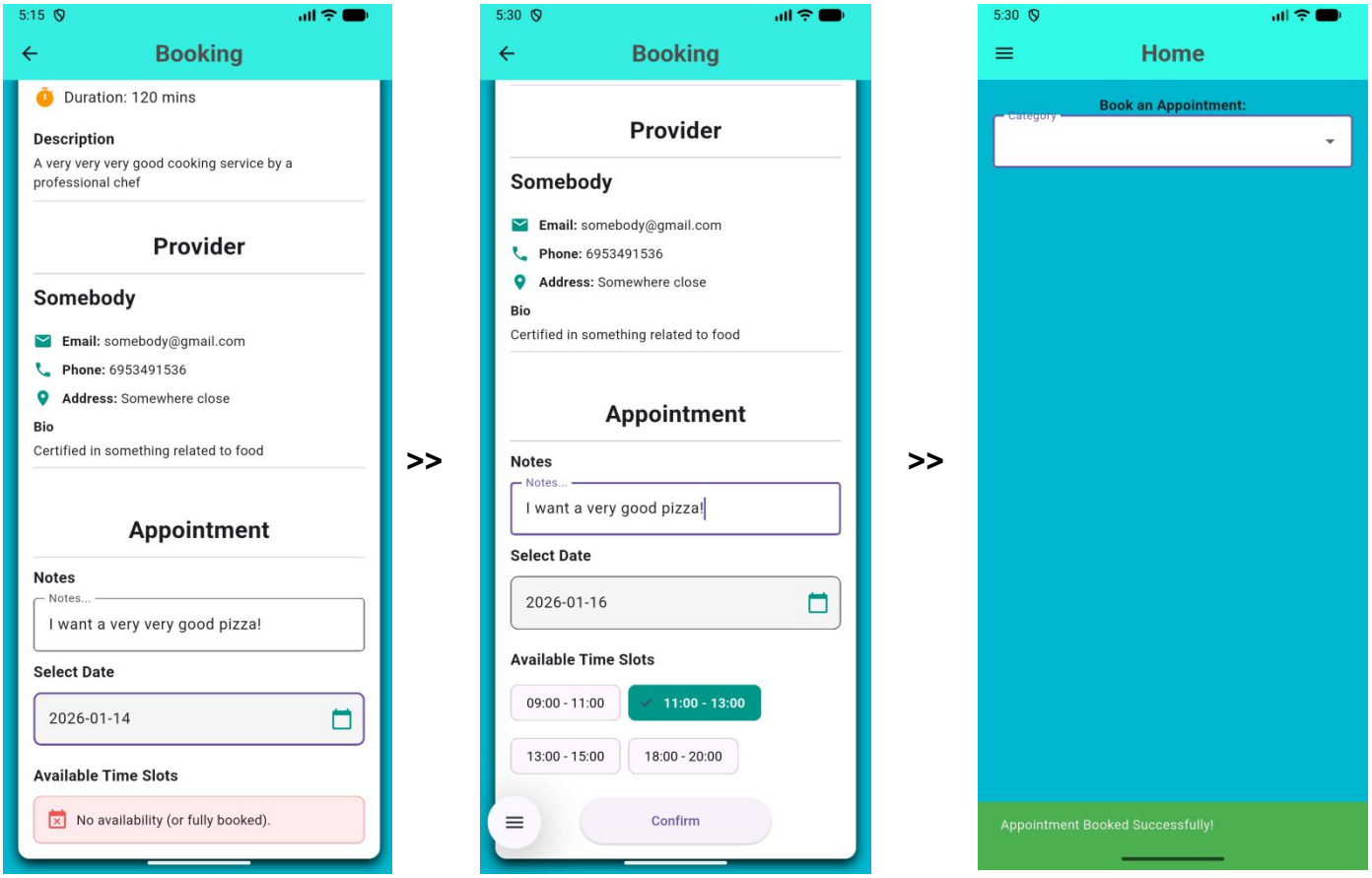


After the User has confirmed that they want the specific service, they must first decide on the provider. There can be many providers for the service, each with a different price and duration. When the User selects a provider, they can tap on the Book Appointment button, to be move on to the next step: selecting a date and time.



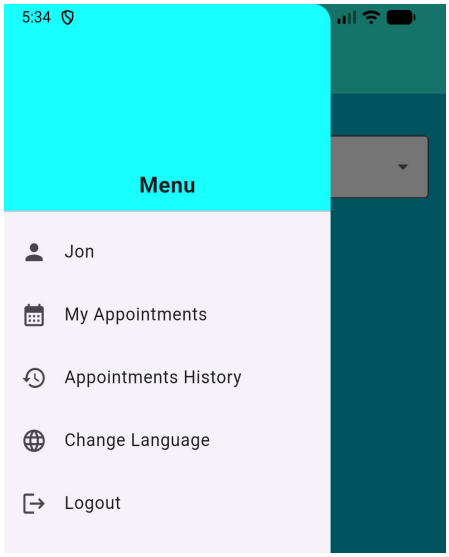
Here, the User can see more of the provider's details as well as write some notes for the appointment. When choosing a date, the user must again choose a date from the calendar. If the provider is available at that date, the user will be shown all of the possible appointment times for that day, otherwise, an error will be shown.

The User has to choose one appointment time and then tap the Confirm button, in order to save the appointment.



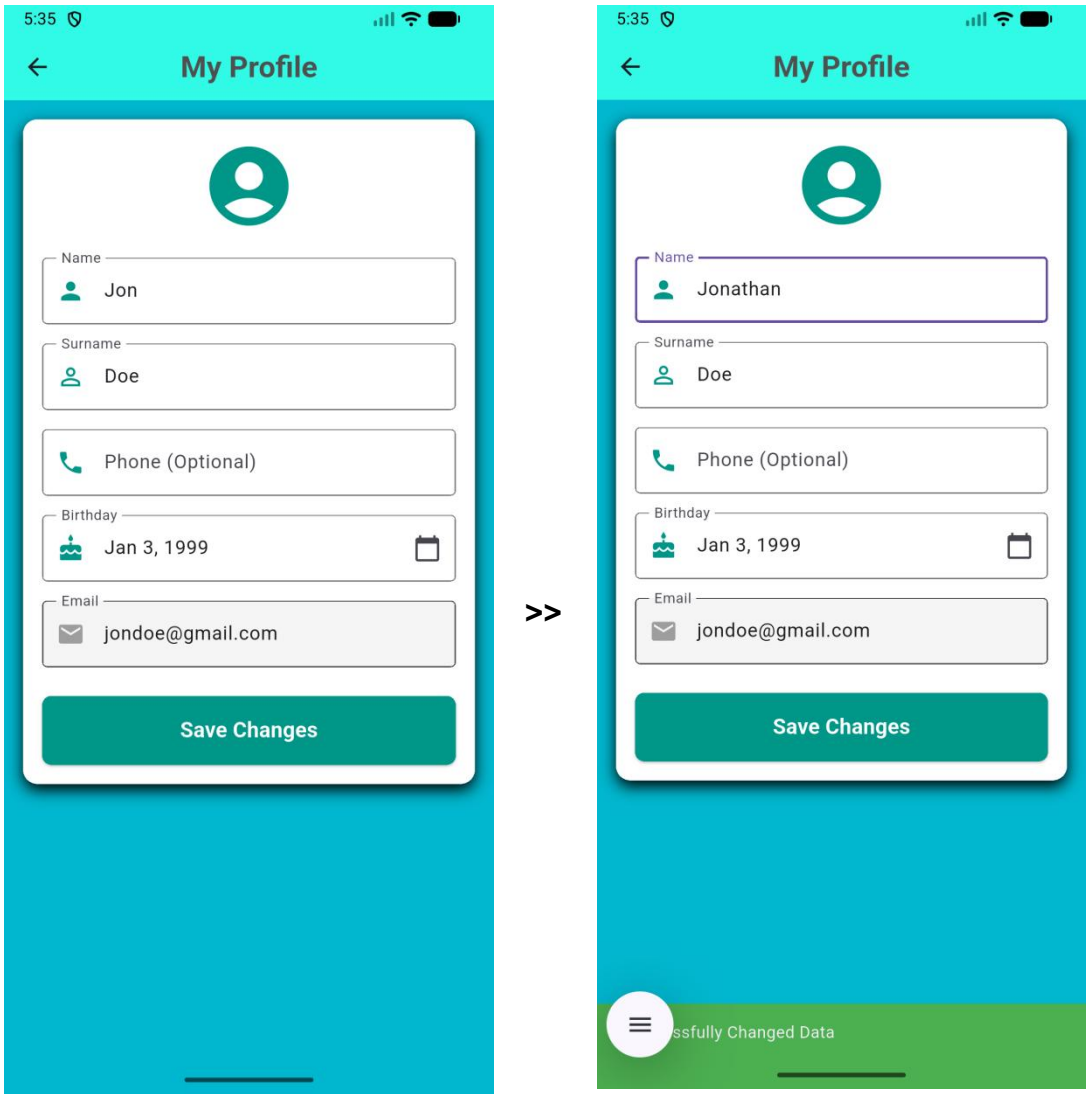
Side Menu

After the User has entered the Home screen, they can tap the hamburger button at the top, in order to open the side menu. There they can find multiple functions to use.



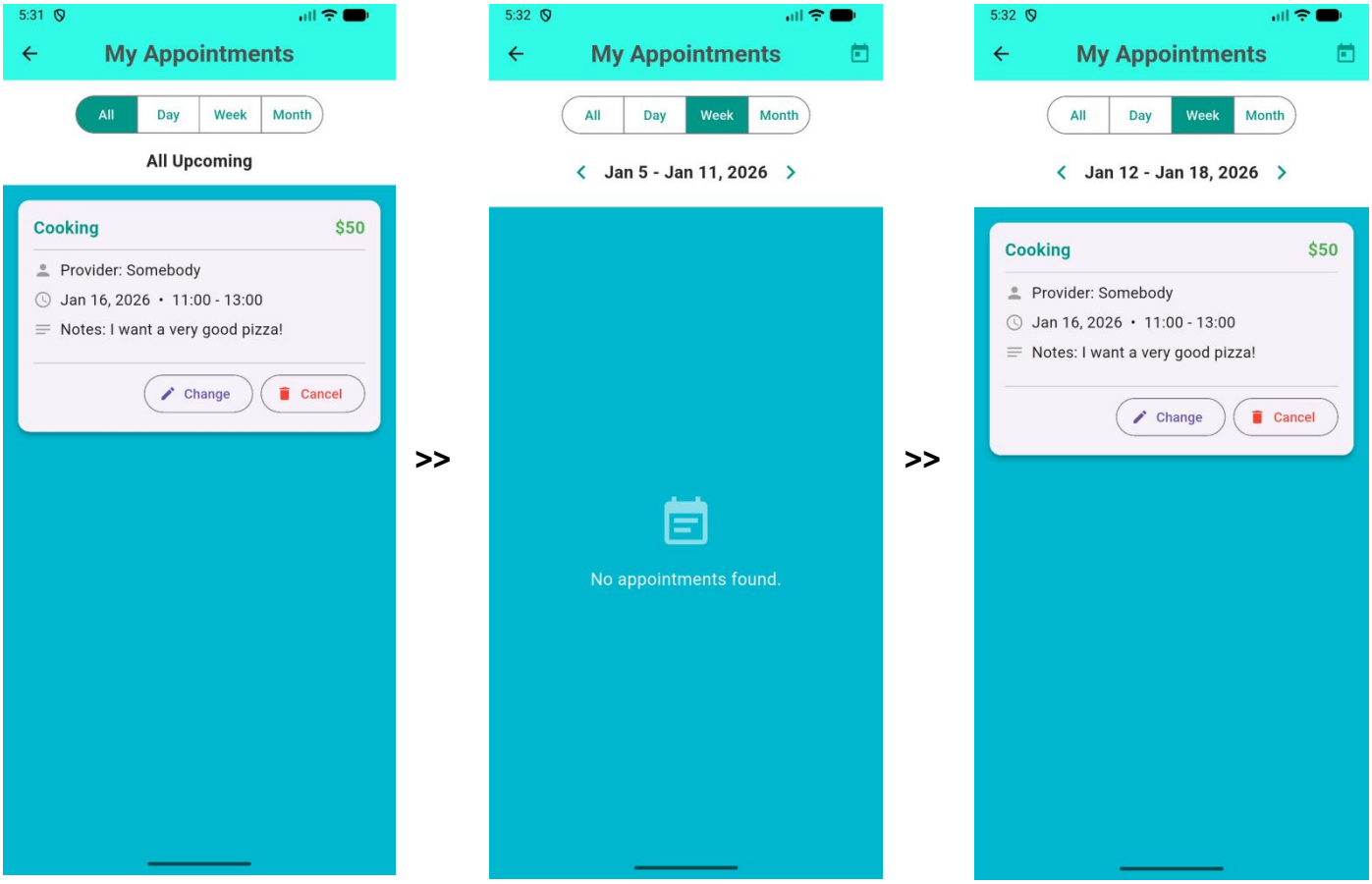
My Profile

The User can click the button with their name on it, in order to ope the User profile. There, they will find all of the data they have given to the app and they are also able to adjust them, minus the email. After the User taps the Save Changes button, they will be shown a notification to be informed.



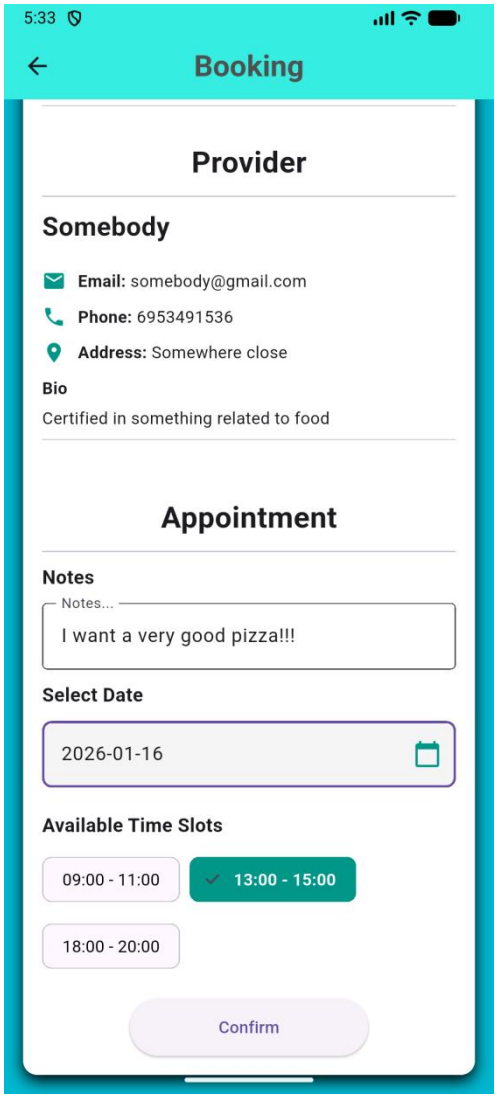
My Appointments

The User can enter the *My Appointments* page through the side menu and there, find all of the future appointments. By default, the User sees all of them in a list, however, they can choose to parse through the calendar days with a step of either a day, a week or a month.

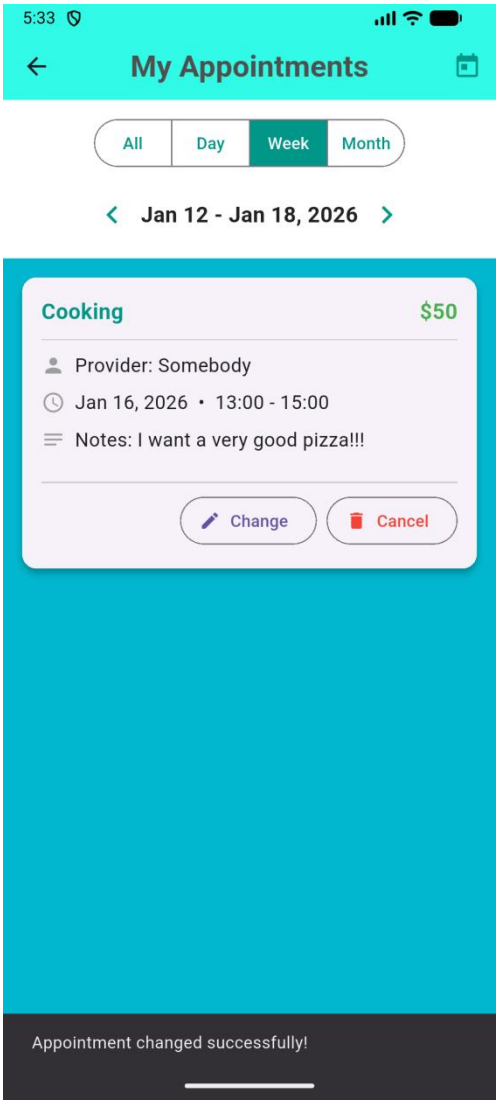


In addition to viewing them, if the appointment is at least 12 hours later, then the User is able to change or delete it.

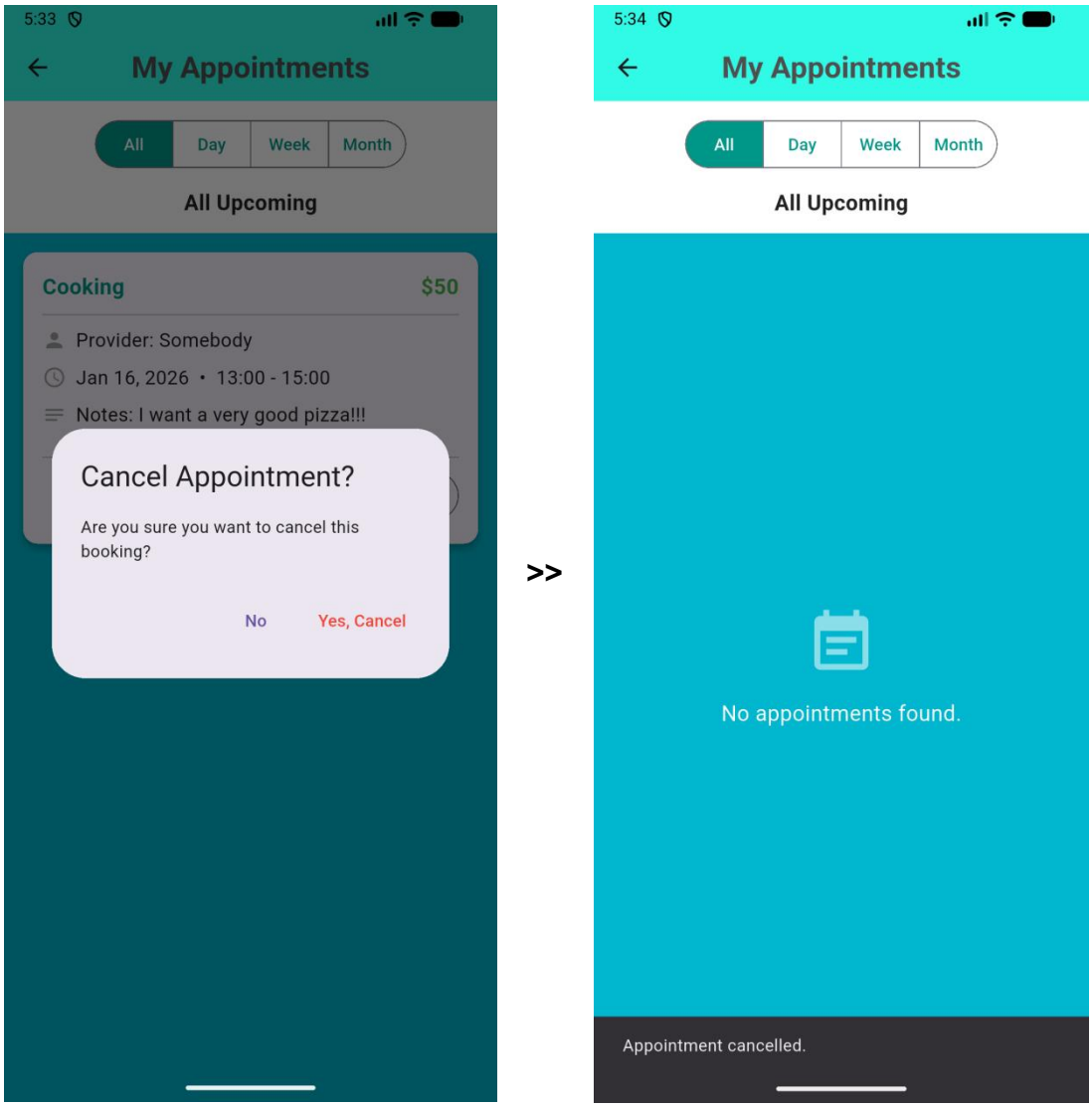
If the User wants to change an appointment, then they have to click the *Change* button of the corresponding appointment, which will take them to the Booking screen again to change the date and/or time, for the specific service and provider.



>>



If the User wants to cancel an appointment, then they have to click the *Cancel* button of the corresponding appointment, which will display an Confirmation window.

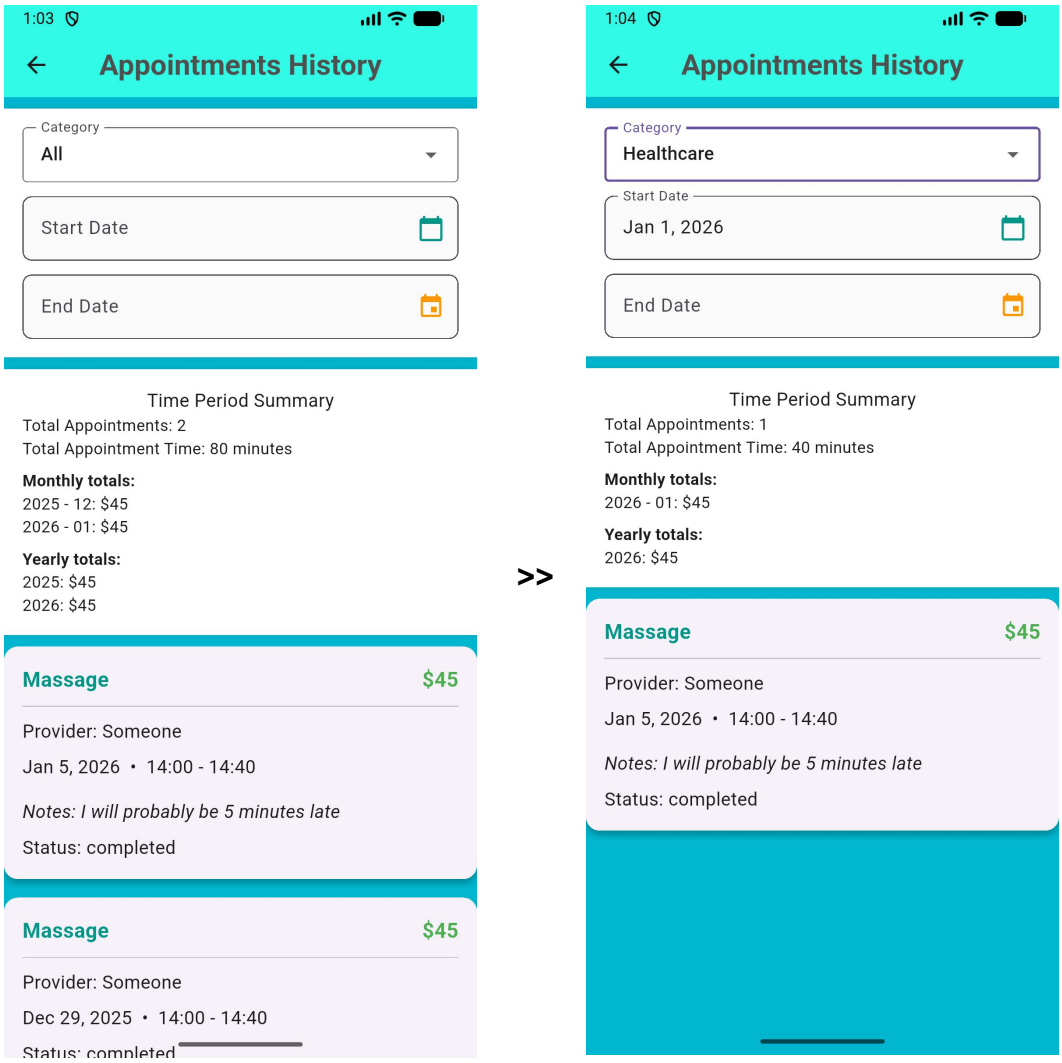


Appointments History

The User can enter the *Appointments History* page through the corresponding button of the side menu. By default, the User sees all of their past appointments. Every appointment has some details to it, as well as if the appointment was completed or not (status: upcoming, meaning it was never completed by the provider, or completed).

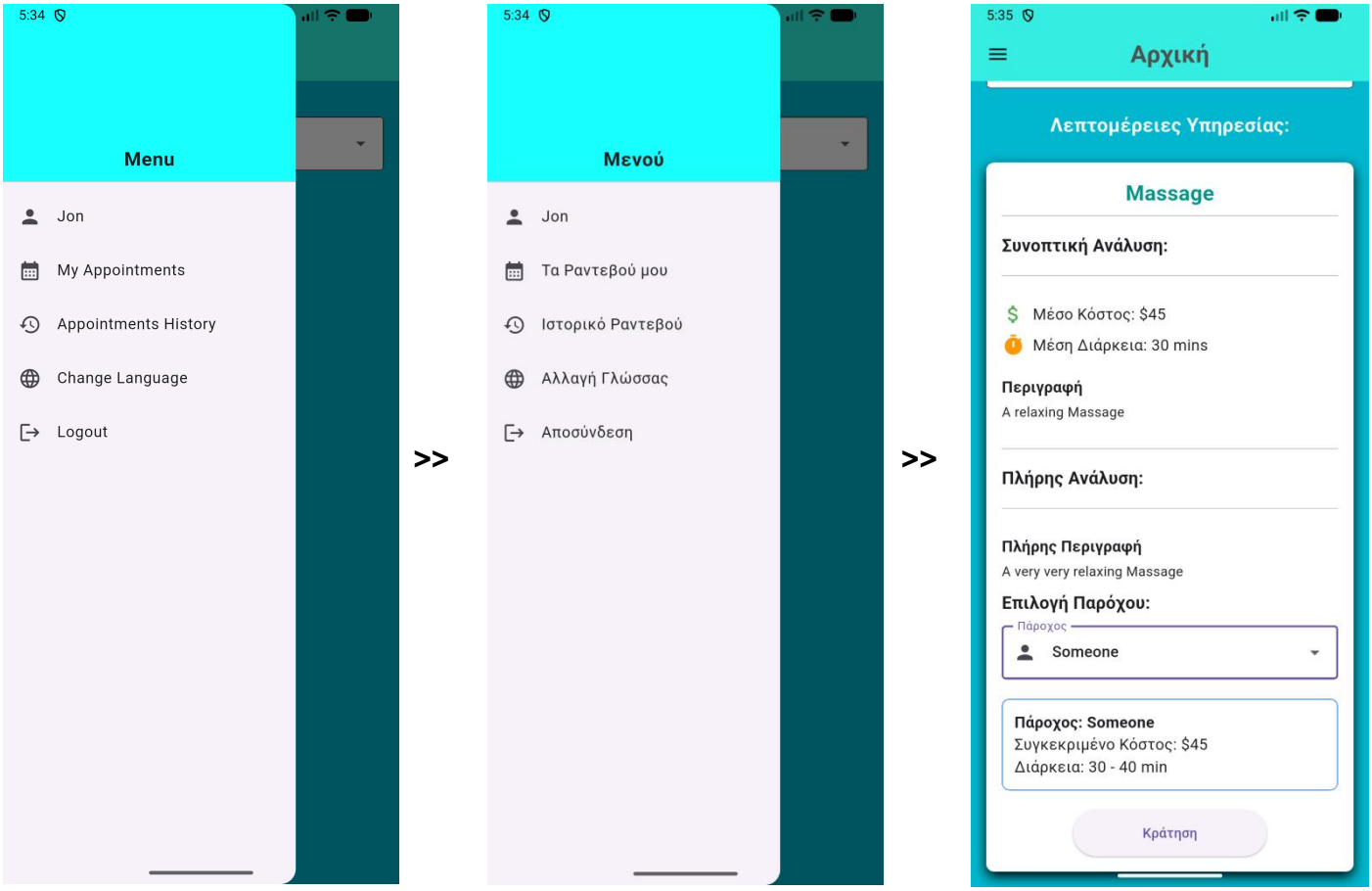
The User can choose to filter the appointments based on category of the service, or by date a date period. If no end date is specified, then it will display all appointments from the specified starting date up to today.

There is also a small segment dedicated to a small summary of the User's appointments. It contains the total number of appointments, their total duration, as well as total cost per month or year. The summary will change depending on the filters as well.



Change Language

A logged in User can change their app language through the *Change Language* button. It will switch between English and Greek.



Logout

The User is able to logout of the app by tapping the L*ogout* button on the side menu. This will pop a confirmation window, awaiting the users response. If the User confirms the logout, they will be transported to the login screen.