CookATool

Final report – Project for mobile application

Susana Rebolledo Ruiz, 2021

Programming for Mobile Devices, IE2

Submission date: 13/05/2021

Supervisor: Liviu Octavian Mafteiu-Scai

Index

1. Abstract

This report includes a model overview of the proposed recipes mobile app, as well as comprehensive details about the app's features, target users and objectives, a comparison of similar apps, new proposed concepts, an original development plan with system functionality diagrams, and used links references.

2. Goal and users

This app is aimed at home-cooks who want to improve their cooking skills and enjoy learning new original recipes. Although CookATool has no age restrictions, the target demographic are people old enough to cook. The age of 11 could be taken as a reference for this application.

3. Introduction

One of the most common cooking issues is deciding what to do with leftovers and how to make the most of them without wasting food or money. This app assists the user in managing their leftovers while also teaching them fresh and interesting recipes.

Not only does this app enrich the user’s cooking knowledge but it also prevents them from wasting usable food and money, making it a very useful tool.

4. Original contribution of the author

CookATool introduces the concept of ingredient-filtered search so to provide the option to search from a user’s updatable database instead of different external websites. This feature sets a different approach to a recipe’s app than usual since the database can actually be updated by each member or logged in user.

Therefore, the app will have a log in option after which each user can upload new recipes to the platform. This new feature gives the app a more social dimension since the experience is no longer limited to a single user but shared among other users.

In addition, every member or logged in user has the possibility to consult a list of the recipes that they themselves have uploaded, because there is a new extra feature available to filter them from the whole database and show them on screen.

CookATool is more than a recipes app, it creates a community of home-cooks in which to learn every day. The advantage of the log in-uploading feature is that the database will never be redundant or boring, since every day new fresh recipes will be added from different users, providing a never-ending experience.

5. Functionality

The final functionality of the application, considering the previously described features, is reflected in the following Use Case Diagram (see Figure 1), created through the UML tool Visual Paradigm.

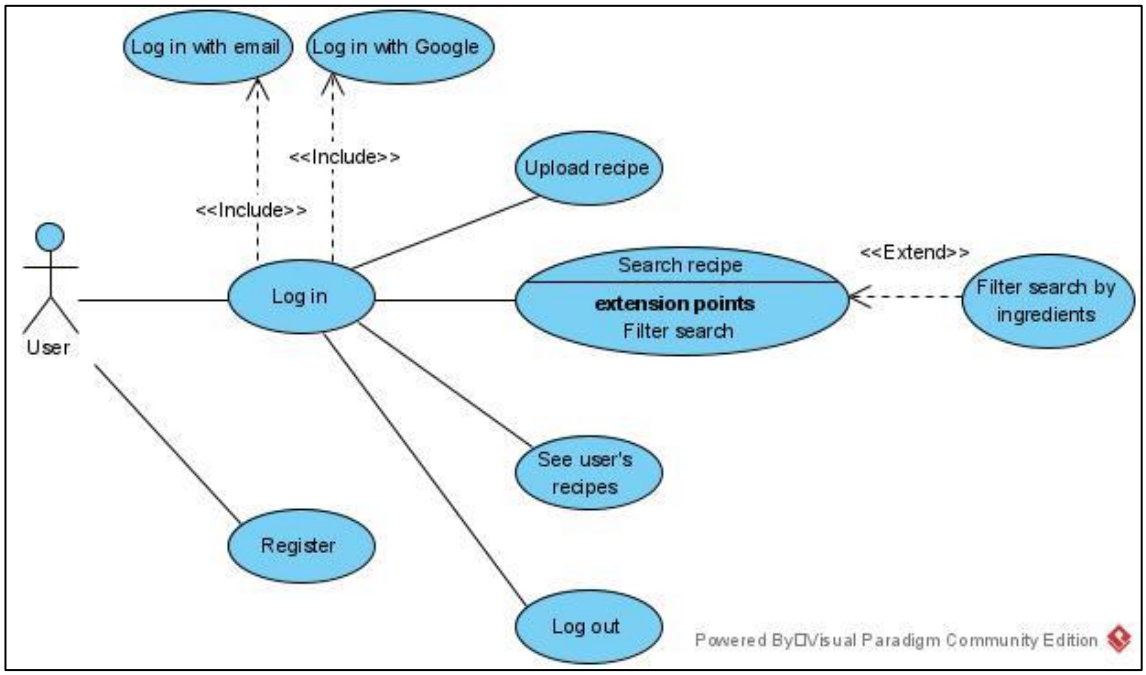

Figure 1. CookATool's Use Case Diagram.

CookATool app is developed in Android Studio using Kotlin as programming language and Firestore from Firebase as database.

6. User interface (UI)

The following diagram (see Figure 2) shows a prototype of the User Interface, created by the GUI prototyping tool Balsamiq. It reflects the home menu, the screen with the application options that shows after logging in.

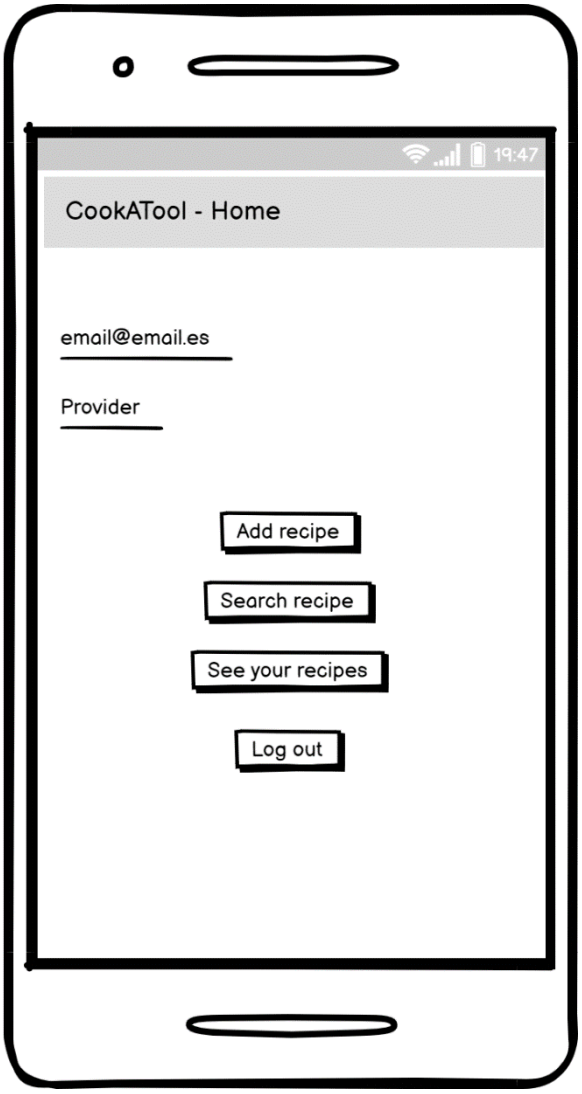

Figure 2. CookATools's GUI prototype of the home screen.

7. Running the app (user's manual)

The app starts, when clicking on the CookATool's icon, after the logo splash.

The first screen is the main screen (see Figure 3) with the options to Log in or Register in the app. After filling in the gaps for email and password, the user can register to become a member or log in if they have previously registered. An error message will appear after trying to register with an email that is already registered on the app's database or after trying to log in with incorrect credentials. There is also a possibility of logging in trough Google in the lowermost button with Google's icon, to provide an easier and quicker accessibility.

After logging in, the application will show the home screen (see Figure 4). This screen will also be shown after the splash in future accesses if the session is already

started, which means that the app was closed without logging out. At the home screen, the user can see the email and the provider with through they logged in. On the options menu, there are four buttons, each of whom implements a different functionality of the app.

The first button, *Add recipe*, leads to a form in which the user can fill a new recipe's details. The details consist of the recipe's name, a list of its ingredients, steps and tags, the estimated time in the preparation, and the servings the recipe is specified for. After filling all the data, the user can press the *Upload recipe* button and the recipe will be added into the app's database.

The second button, *Search recipe*, leads to a searching screen in which the user can filter the search by ingredients by typing the required ones on the input box. When pressing the button *Search*, a new screen will appear with a list of the matching found recipes. If there is no matching, an error message will appear. If there is no filter provided, a list of all recipes from the database will appear. In the list, the recipes have all the details which where specified at the uploading screen and an email field of the user who uploaded them.

The third button, *See your recipes*, leads to a list of all the recipes the user uploaded. If the user did not upload any recipe, an error will appear.

The last button on the menu, *Log out*, closes the current member's session and leads back to the main screen.

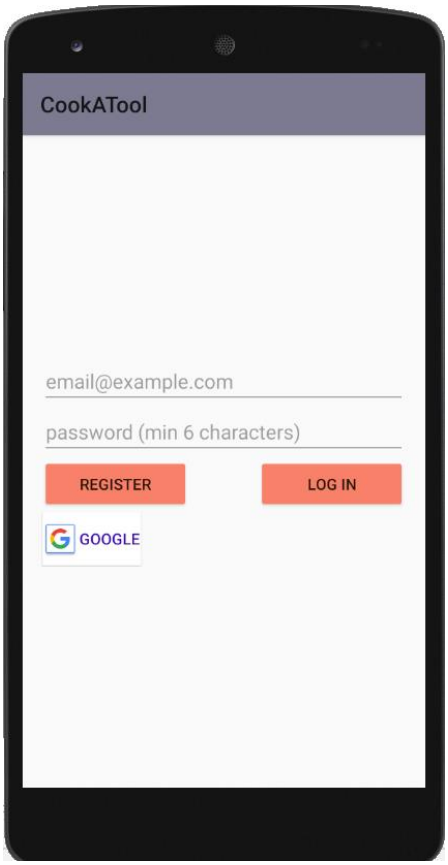

Figure 3. CookATool's main screen.

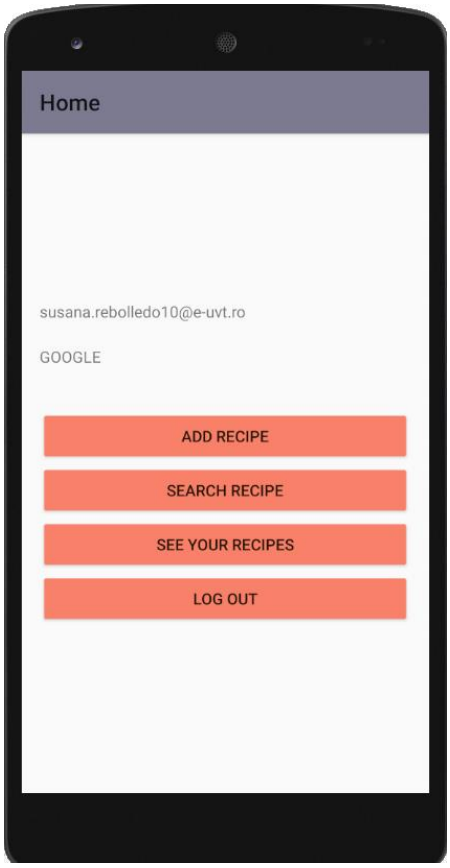

Figure 4. CookATool's home screen.

8. App's structure (technical manual)

This section focusses on the main screen (see Figure 3 from previous section) code, considering both activity and layout from Android Studio.

The main activity is called *MainActivity.kt* and its layout, *activity_main.xml*.

8.1. Layout

In the *activity_main.xml*, there is a vertical layout. In it, there are two *EditText* whose ids are *emailEditText* and *passwordEditText* (see Figure 5), in which the user can introduce their email and password, respectively, to register or log in. They both have a hint example of what the user should introduce, specifying, for instance, that the password should consist of at least 6 characters.

```
<EditText
    android:id="@+id/passwordEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="password (min 6 characters)"
    android:inputType="textPassword" />
```

Figure 5. EditText for the password from activity_main.xml layout.

In addition, there is a horizontal layout inside the vertical one in which buttons are allocated. There are two ordinary buttons, separated with a space, one with the text 'Register' on it, whose id is *registerButton* (see Figure 6), and one with the text 'Log in', with the id *logInButton*. They have both defined their background tint to the colour *#F98069*, which corresponds to the orange used in CookATool's logo.

```
<Button
    android:id="@+id/registerButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    app:backgroundTint="#F98069"
    android:text="Register" />
```

Figure 6. Button to register from activity_main.xml.

Below these buttons, there is a more complex button (see Figure 7) to provide the log in though the Google platform. This button's id is *googleButton* and it has the

background tint set to the colour white and a drawable left defined to the icon *common_google_signin_btn_icon_dark* found in the *drawable* folder.

```
<Button
    android:id="@+id/googleButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="1dp"
    android:layout_weight="1"
    android:text="google"
    android:textColor="?attr/colorPrimaryDark"
    app:backgroundTint="@color/white"
    app:iconGravity="top|textStart"
    android:drawableLeft="@drawable/common_google_signin_btn_icon_dark" />
```

Figure 7. Button for Google log in from activity_main.xml.

8.2. Activity

In the *MainActivity.kt* there is the Kotlin code for the functions related to this activity.

There is an overridden function *onStart* which guarantees the on-start visibility of the layout by setting its visibility to *VISIBLE*.

There is another overridden function *onCreate* which, apart from having the saved instance *Bundle* as a parameter and setting the content view to the *activity_main* layout, obtains the database instance, in this case, the FirebaseAnalytics instance for Analytics Events. This function also calls two other functions: *setUp* and *session*.

In the *setUp* function, the behaviour after clicking each button from the layout is defined. For the log in and the register button (see Figure 8), there is defined a *SetOnClickListener*, which uses the introduced credentials, email and password, if not empty, to call the *FirebaseAuth* instance with the *createUser* or *signIn* functions respectively. To check the successfulness or not, there is set an *addOnCompleteListener* to call the private functions *showHome*, which receives the used credentials as parameters, if success or *showAlert* if error.

The *showHome* (see Figure 9) function starts the next activity, *LogIn.kt*, after creating its *Intent* with the email and password credentials as extra parameters.

The *showAlert* (see Figure 9) function creates an *AlertDialog* which will pop up to advert of an authentication error.

```
private fun setUp() {
    // Register button
    registerButton.setOnClickListener {  it: View!
        if (emailEditText.text.isNotEmpty() && passwordEditText.text.isNotEmpty()) {
            FirebaseAuth.getInstance().createUserWithEmailAndPassword(emailEditText.text.toString(),
                passwordEditText.text.toString()).addOnCompleteListener {  it: Task<AuthResult!>
                if (it.isSuccessful) {
                    showHome( email: it.result?.user?.email ?: "", ProviderType.BASIC)
                } else {
                    showAlert()
                }
            }
        }
    }
```

Figure 8. Function for the registerButton click from the setUp function from MainActivity.kt.

```
// Shows authentication alert
private fun showAlert() {
    val builder : AlertDialog.Builder = AlertDialog.Builder( context: this)
    builder.setTitle("Error")
    builder.setMessage("Authentication failed")
    builder.setPositiveButton( text: "Accept",  listener: null)
    val dialog: AlertDialog = builder.create()
    dialog.show()
}

// Shows home when logged in
private fun showHome(email: String, provider: ProviderType) {
    val homeIntent = Intent( packageContext: this, LogIn::class.java).apply {  this: Intent
        putExtra( name: "email", email)
        putExtra( name: "provider", provider.name)
    }
    startActivity(homeIntent)
}
```

Figure 9. Functions showAlert and showHome from MainActivity.kt.

In the *setUp* there is also defined the behaviour after clicking the Google button, with a *setOnClickListener* (see figure 10), as with the previous buttons. In it, *GoogleSignInOptions* is called with the token id associated to the app, requesting its email, and building it to configure the Google log in. Then, the function *startActivityFromResult* is called to show the Google Authentication screen with the identification parameter set as *GOOGLE_SET_IN* constant so as to collect the authentication response. Since there is an activity from which a response is expected, the function *onActivityResult* must be overridden. In that function, the request code is checked to be equal as the identification constant, and if so, the Google account is recovered. If this account is not null, the *signInWithCredentials* function is called, in the same way as it was with the ordinary log in button, but with this account's credentials

obtained from *GoogleAuthProvider*. The following *setOnCompleteListener* works in the same way as with the previous buttons but setting the provider to *GOOGLE* instead of *BASIC*.

```
// Google log in button
googleButton.setOnClickListener{  it: View!
    val googleConf = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken(getString(R.string.default_web_client_id))
            .requestEmail()
            .build()

    val googleClient = GoogleSignIn.getClient(this, googleConf)
    // Sign out in previous google account
    googleClient.signOut()
    // Sign in on google
    startActivityForResult(googleClient.signInIntent, GOOGLE_SIGN_IN)
}
```

Figure 10. Function for the googleButton click from the setUp function from MainActivity.kt.

The last function is the *session* function (see Figure 11), in which the saved email and password are recovered and if they are not null, as they are as default, *showHome* will be called so as to navigate to the next screen corresponding to the next activity, since that means the session is already started in the application. In addition, the function layout will be set as *INVISIBLE* so as to go directly to the next screen in case of the session being already started. This allows the user to access the application without having to log in each time, because it keeps record of the started session as long as they do not log out.

```
// Function to keep initiated session
private fun session() {
    val prefs = getSharedPreferences("com.example.cookatool.PREFERENCE_FILE_KEY", Context.MODE_PRIVATE)
    val email = prefs.getString( key: "email",  defValue: null)
    val provider = prefs.getString( key: "provider",  defValue: null)

    // In initiated session case
    if (email != null && provider != null) {
        // Layout invisibility to start home directly
        authLayout.visibility = View.INVISIBLE
        showHome(email, ProviderType.valueOf(provider))
    }

}
```

Figure 21. Function session from MainActivity.kt.

The logged in users are stored in the Firebase Authentication table for CookATool with its correspondent provider, id, register date and last accessed date.

9. Conclusions and future work

From the start, this project has taken a lot of time and effort. Due to my lack of expertise in programming for mobile devices and the exceptional characteristics of this course with a completely online education, this application creation posed a significant challenge for me. I am proud of finally being able to develop a functional application and of the great new knowledge I acquired by learning it.

Despite the fact that this app appears to be very basic in comparison to other available recipe apps, the huge progress I went through cannot be overlooked and makes me satisfied with my work for this course.

To conclude, I would like to state I had to renounce to some of the features I thought of initially, since it took me more time than the expected to solve the problems I encountered in the process.

10. References

The references consulted during the project development are the followings.

[1] CookATool. (2021). Retrieved 13 May 2021, from https://console.firebase.google.com/u/0/project/cookatool/authentication/users?hl=es

[2] GameAppStudio. (2021). Retrieved 13 May 2021, from https://www.youtube.com/watch?v=8KOXtOzTRTE&list=LL&index=2

[3] Kotlin Tutorial for Beginners | Learn Kotlin. (2021). Retrieved 13 May 2021, from https://beginnersbook.com/2017/12/kotlin-tutorial/

[4] Learn with Deeksha. (2021). Retrieved 13 May 2021, from https://www.youtube.com/watch?v=CpQH9TMmBMs&list=LL&index=1&t=463s

[5] Moure, B. (2021). MoureDev. Retrieved 13 May 2021, from https://www.youtube.com/channel/UCxPD7bsocoAMq8Dj18kmGyQ