Universität Regensburg

Home Connect Android Application

Project Report

Fehemi Lecini,
Kuldeep Bundela

Project under the supervision
of Professor Bernd Ludwig

Department of Informatics
Universität Regensburg
Germany
March 2021

1 Introduction

It was 1969 when the first message was sent online through ARPANET from one computer to another computer. From then on, things changed a lot and internet became a place where to show information to users, so in other terms all the pages were static. Nowadays, we are entering a new era of communication, the era of Internet of Things, also known as IoT, where devices, embedded with sensors and software, are connected to the Internet and exchange information and communicate with each other in order to reach what is wanted from the user.
This report describes a project between Bosch and Siemens, that together want to create the future of home automation. Basically, the project consists in having all the home devices like oven, coffee machine, dryer, fridge , etc., connected to Internet. The end purpose is to monitor and control these devices, also called home appliances, through an app. This app will then allow users, for instance, to prepare a coffee from the app without the need to physically go to the coffee machine. This report is a short description of our winter semester's internship project accomplished as a mandatory and free choice components of the Master program. The internship was carried out within the University campus of Regensburg from November 2020 to March 2021. Our project is focused on an android application which controls and monitors a smart coffee machine using Home Connect.

Home Connect [1][2] provides a Restful API to manage enabled home appliances. This API allows access to home appliances connected to Home Connect and in our case access to enabled coffee machine. In our case, for instance, you would have the possibility to turn on your coffee machine and make your favorite cup of coffee.

Figure 1: Home Connect

Home connects provides home connect API as well as simulator for the development phase of applications [1]. They provide the virtual environment for simulated appliances. But to use the home connect simulator, user should log in his/her account. The application should be authorized by the API for the interaction with coffee machine. The authorization process includes OAuth 2.0

Authorization Code Grant Flow process which results in a JSON token. To monitor, user should make request to choose the appliance and then to retrieve the operation state. And to control, user must turn on the appliance and then start the program.

The API of Home Connect is based on HTTPS protocol with the JSON data encoding and it supports many languages, including english. To request a specific language user should add the Accept-Language header during GET request. Moreover, Home Connect API supports amount-based and time-based rate limits in case of too many requests are sent. For every request, API returns a proper HTTP status code.

2 System Architecture

Home Connect API provides remote access via Home Connect to all the appliances, so a direct access to home appliances is actually not possible. The graphic in the Figure 2 shows a high-level overview of the connections between the technical components.

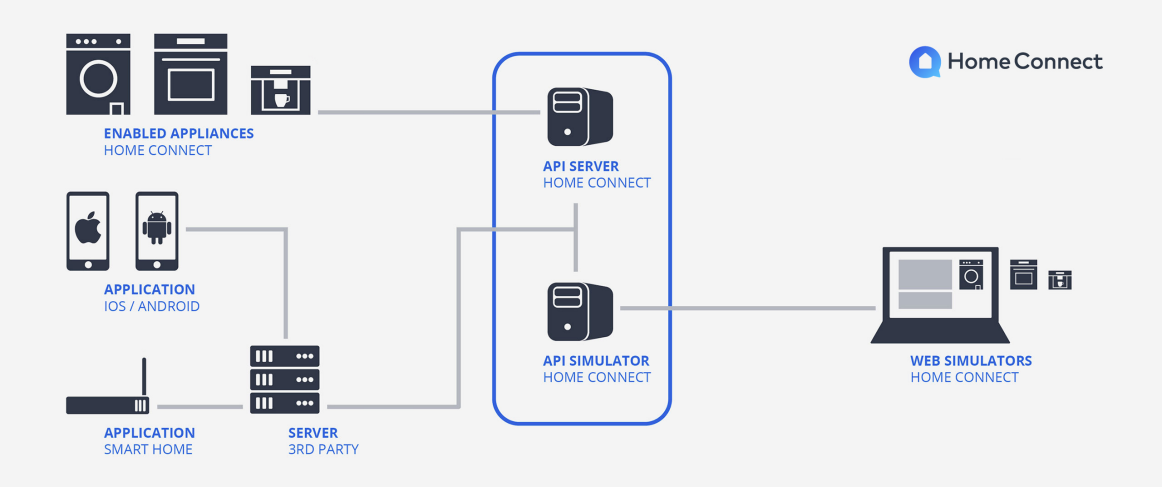

Figure 2: System Architecture

As we can see from Figure 2, the different home appliances, on the top-left corner, are connected to an API Server, which is connected to the API Simulator that we use with our app in order to make a request. Note that it is possible to make requests both from the Application with an IOS/ANDROID Phone and from the Web Simulator on the Home Connect page.

3 Authorization

Home Connect API supports the OAuth2 Authorization Code Grant Flow, but first, before starting with the authorization of the application, it is necessary to register the application in the developer portal. Once this step is completed, we will get a client ID and client secret, which will be used while making requests to the appliance. All the process of authorization is shown in the Figure 3 below.

From Figure 3 what emerges is that there are three different actors that interact with each other, namely the Application, the Authorization Server and the User. The process starts basically, as soon as the user opens the app and clicks on the login button. When this happens, the app automatically will send client-id and redirect-uri. At this point the user will be prompted to a page to login and once he gives the input correctly, the Authorization Server sends back to the Application an authorization code. Now, thanks to this code the Application can now ask to the Authorization Server for the access-token, by sending client-id, client-secret and the previously

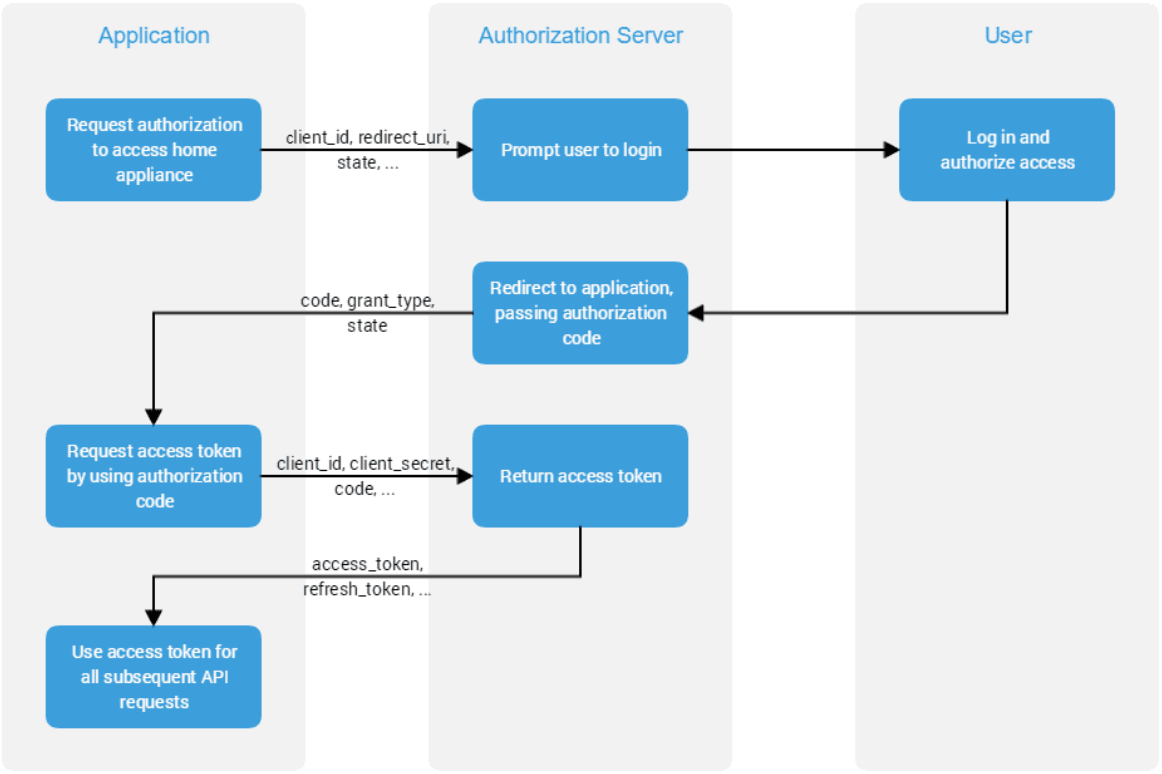

Figure 3: Authorization Code Grant Flow

received code. Last step will be for the Authorization Server to send back to the Application refresh and access-token. The access- token is what is needed to make requests, basically to control and monitor the home appliance through the API.

4 Programming IDE and Activity Flow

In order to realize the App we opted for Android Studio as it is designed specifically for android development and in our case we were asked to develop an android application. As we should know, Android Studio comes with Activities, but what is an Activity and why are we supposed to use them? In Android Studio an Activity class takes care of creating a window for you in which you can place your UI (user interface) with setContentView(View). In this project we used three different activities, namely *MainActivity*,*LoadActivity* and *RequestActivity*. MainActivity class is used to display the login screen where the user can press login button and proceed. The Figure 4 below offers a clear view of the MainActivity class.

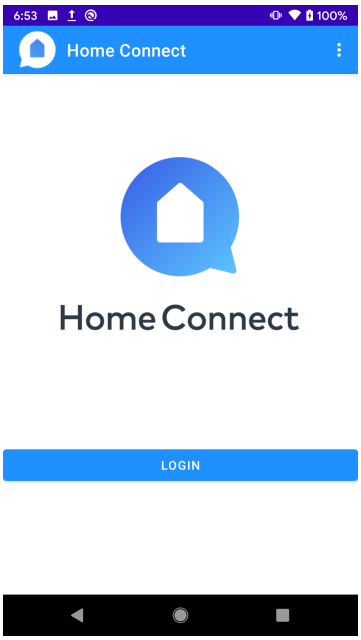

Figure 4: Login Activity

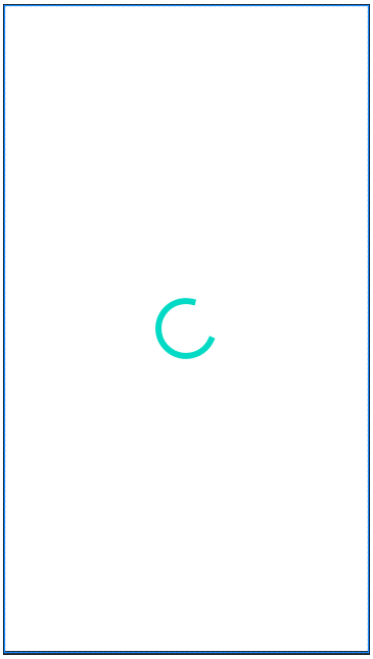

Figure 5: Load Activity

Now here is where the user starts to interact with the system by pressing the Login button. This action will direct the user to a browser in order to input the correct username and password. However, what is obscure to the user is what is happening in the background: the app gets a code, that needs to be sent again to the API Server and finally the app receives a token, that will be used to make requests to the Coffee Machine home appliance. So, basically, during all this time that information is exchanged, to the user is displayed the LoadActivity class. This activity just shows a circular progress bar, which can be seen in the Figure 5 above o the right.

Now this process will take few seconds and once the app get the token from the API Server, it finally can move to the last activity, namely RequestActivity. Now, this is actually what the user cares about, because from here he finally can communicate with the coffee machine.

The activity itself, as shown in Figure 6, consists, from bottom to top, of a microphone image button, a picture of a coffee machine, a progress bar and an edittext. When the user finds himself on this activity, he is able to ask to the coffee machine to prepare a determinate coffee (with some characteristics) as well as other information, by simply pressing the mic button and talking. If the user hits the mic button it will change color, signaling that it is actually listening for input and at the same time on the edittext field appears the word 'Listening'. What the users says by voice is then displayed on the edittext field. At this point, if what the user is asking for, complies with the required and accepted parameters, the simulator will do what the user asked, for instance prepare a macchiato of 50ml, and when this process starts, the yellow progress bar in the coffee machine picture starts to fill, simulating the time that a macchiato needs to get done.

Figure 6: Request Activity

4.1 Menu

The app has a toolbar that includes an icon, a text and also a menu, as we can see from Figure 7. In this subsection we will see the items that a menu contains. As the picture shows, there are two options available: *Info* and *Exit*. The Info option is shown in Figure 8. When the user clicks on Info a window, that includes all the instructions to use the app, will open and to close it the user just has to click on "OK". On the other side there is also an option *Exit*, which if pressed, will stop and exit the application.

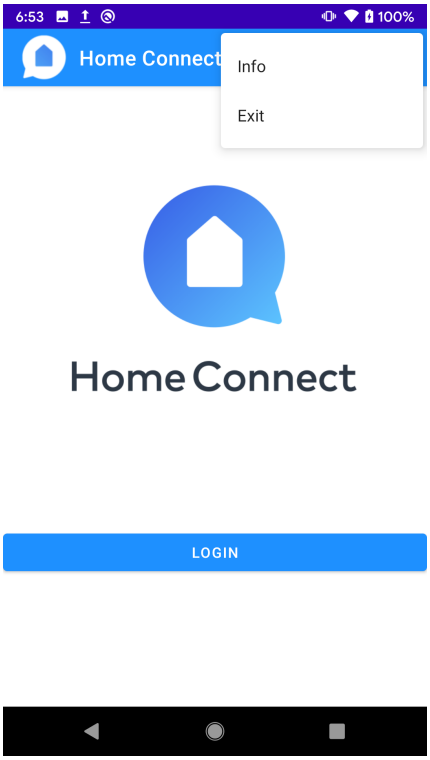

Figure 7: Menu

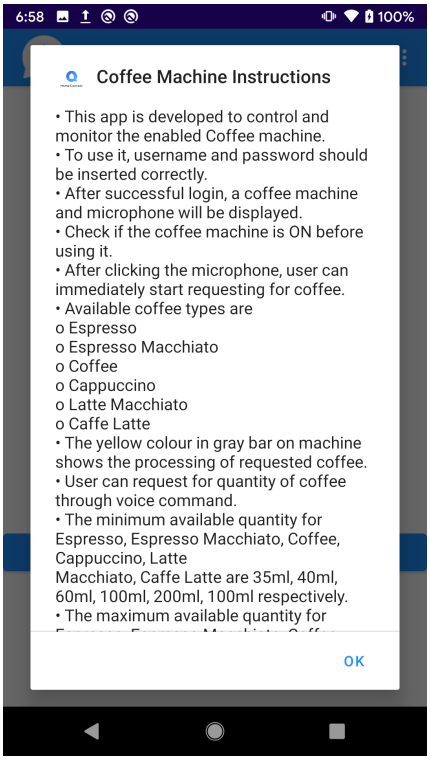

Figure 8: App Instruction

5 Login

In this section we will have a deeper insight onto Login process from the point of view of code implementation. Well, as we saw in the authorization section, as soon as the user presses the login button the process of authorization start, but here we will go more into detail, so let us see how it works internally. From the Figure 9 below, we can see that the authorization process starts at line 40, where the application requests authorization to access home appliance. We have to let know the Authorization Server who we are and we do this by attaching to the url the client-id, which identifies us.

Figure 9: Authorization Code Request

On the other side we also have to tell what are we looking for, so in our case the response-type, as shown in the line 41, will be *code*. We will get the code back in the url and we will catch it within the OnResume method. The next step of the procedure will be to ask for the access token and we can see its implementation in Figure 10 below.

In these lines of code we send a post request to the Authorization Server by passing also *clientid*, *clientsecret*, the *authorization code*, that we just received from the previous step, and grant-type set to *authorization code*. Once we make the call, we also have to wait for response from the server, which is implemented in the lines from 45 to 50. On line 47 we finally get what we were seeking, the token, that we will use to make further requests to the home appliance. But how does actually this request to the server work? Well, for that we need a client, in this case *Retrofit* and this will be explained more in detail in the section HTTP Requests.

```
//send code and get token

LoginService client = ApiManager.getClient().create(LoginService.class);
Call<AccessToken> accessTokenCall = client.getAccessToken(
        ApiManager.clientid,
        ApiManager.clientsecret,
        code,
        grant_type: "authorization_code"
);


accessTokenCall.enqueue(new Callback<AccessToken>() {
    @Override
    public void onResponse(Call<AccessToken> call, Response<AccessToken> response) {

        token = response.body().getAccessToken().toString();

        openRequestActivity(token);
    }
```

Figure 10: Token Request

6 HTTP Requests

The Hypertext Transfer Protocol, aka HTTP, is the protocol on which the World Wide Web is based on, thus the protocol that machines use to communicate with each other. In order to make these requests, however, we do need a client on our side and in the login part this is implemented with the *Retrofit* client. This is not the only possible client that we can use. Actually, the requests to the home appliances are done with a library called *Volley*. All this will be explained more precisely with example in the next subsections. In the previous section we briefly touched the topic of http requests and what there is more to say about this is that there are different type of requests based on your needs, among them *Get Request*, *Put Request* and *Delete Request*. Let us have a look at each of them separately and at the way they have been implemented.

6.1 Get Request

GET request is one of the most common HTTP methods and it is used to request data from a server. So what are some situations in which we would use a Get Request? Well, within this project some of the possible Get Requests are the following:

- Get all programs of a given home appliance
- Get all programs which are currently available on the given home appliance
- Get a list of available settings, eg. to check if machine is on

Let us now go more into the details of implementation of a Get request, with the example in the Figure 11 below.

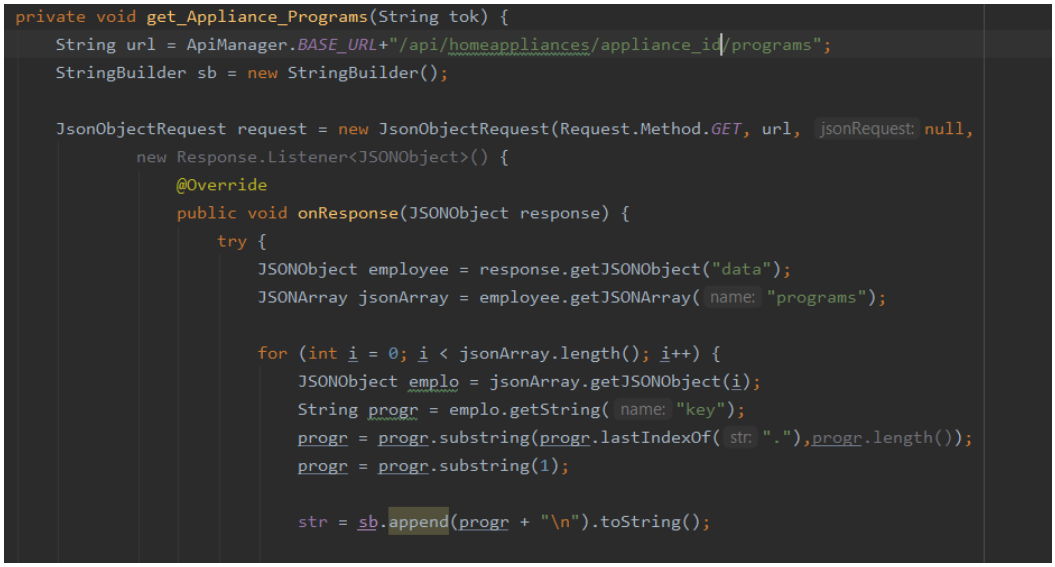

Figure 11: GET Request with Volley

The method *get-Appliance-Programs* in the Figure 11 implements a Get Request to the home appliance with a determinate id and here we are asking for all the available programs that it can offer to us. In the case of the coffee machine, the programs would be *Espresso*, *Cappuccino* so on and so forth. The request is made through an HTTP library called Volley. It was also possible to use Retrofit, like we did in the login part, but Volley is easier in treating the data. All the communication between client and server happens with a format called Json, so we send and receive data in Json format. Once we send the request, we can see in the *onResponse* method that we receive the data and we have to work with it to get every specific JSON object.

Figure 12: GET Request Header with Volley

To complete the request, however, like we have seen in the previous sections, we need to include the token. This part is shown in Figure 12, where we add some parameters to the header, namely the type of information we are passing, JSON in our case, and also the authorization token.

6.2 Put Request

The Put Request method is used to send data to a server to create and update a resource and we would use this in case we would like to have a Cappuccino. You would need to tell to the server to start to produce it and we can see its implementation in the Figure 13. Here we can see that differently from the Get Request, we first have to prepare the data that we want to send to the server and only then send the final request. Just like in the Get Request, also here we create the Json objects and structure, and, moreover, every Jsonobject has a key for being identified and also a value.

```
631    private void start_coffee(String tok, String program, int amount,String strength) {
632
633        String url = ApiManager.BASE_URL+"/api/homeappliances/appliance_id/programs/active";
634
635        final JSONObject jsonObject = new JSONObject();
636
637            JSONObject dataobj = new JSONObject();
638            //Create json array for options
639            JSONArray arrayoptions = new JSONArray();
640
641            //Create json objects for two options
642            JSONObject option1 = new JSONObject();
643            JSONObject option2 = new JSONObject();
644
645            String cof = "ConsumerProducts.CoffeeMaker.Program.Beverage.";
646            String bean= "ConsumerProducts.CoffeeMaker.EnumType.BeanAmount.";
647        try {
648            dataobj.put( name: "key",  value: cof+program);
649            option1.put( name: "key", value: "ConsumerProducts.CoffeeMaker.Option.BeanAmount");
650            option1.put( name: "value", value: bean+strength);
651
652            option2.put( name: "key", value: "ConsumerProducts.CoffeeMaker.Option.FillQuantity");
653            option2.put( name: "value",amount);
654            option2.put( name: "unit", value: "ml");
655
656            arrayoptions.put(option1);
657            arrayoptions.put(option2);
658
659            dataobj.put( name: "options",arrayoptions);
660
661            jsonObject.put( name: "data",dataobj);
662        } catch (JSONException e) {
663            e.printStackTrace();
664        }
665
```

Figure 13: Put Request with Volley

Moreover, just like in the Get Request, also here we need to add the header, which is shown in Figure 12, but we are not going to repeat it again.

6.3 Delete Request

The DELETE method requests is another type of HTTP request used in order to tell to the server to delete a resource, which is identified by the Request-URI. As we can see from the Figure 14, the structure is just the same as the one of GET and PUT request. In this case we do not need to send any other type of json data to the server, because we are not in a PUT request, however, we do need to include again the header with the available authorization token.

```
private void stop_program_executing(String tok) {

    String url = ApiManager.BASE_URL+"/api/homeappliances/BOSCH-HCS06COM1-0631A8BD1A43/programs/active";
    JsonObjectRequest delete_request = new JsonObjectRequest(Request.Method.DELETE, url, jsonRequest: null,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {

                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            error.printStackTrace();
        }
    })
    {
        @Override
        public Map getHeaders() throws AuthFailureError {
            HashMap headers = new HashMap();
            headers.put("Content-Type", "application/json");
            headers.put("Authorization", tok);
            return headers;
        }
    };
    mQueue.add(delete_request);
```

Figure 14: Delete Request with Volley

7 App Operating

Show that app is working with some pictures...it really does what the user says So far we have seen the context and we also had a look at the implementation in the previous section. In this current section we want to bring a real example of the app working. As we know the purpose of this App is to monitor and control a Coffee Machine. The user can choose a product of his/her desire, each with its own quantity limits, from the following table list in Figure 15.

Program	Size (ml)		Strength
	min	max	
Espresso	35	60	• Very Mild • Mild • Normal • Strong • Very Strong • Double Shot • Double Shot+ • Double Shot++
Espresso Macchiato	40	60	
Coffee	60	250	
Cappuccino	100	300	
Latte Macchiato	200	400	
Caffee Latte	100	400	

Figure 15: List of all programs with quantity limits

Moreover, if you choose your favourite coffee type, usually at vending machines you also have the opportunity to choose the strength and this is also the case of this app. So, besides the size, user can also choose the strength parameter, which is reported in the column number four of the above figure 15. For instance, the user could order, by voice, an espresso of size 60ml and a strength of type Strong. The product preparation would then be simulated on the web simulator of Home Connect. We are going a real example with some pictures right below.

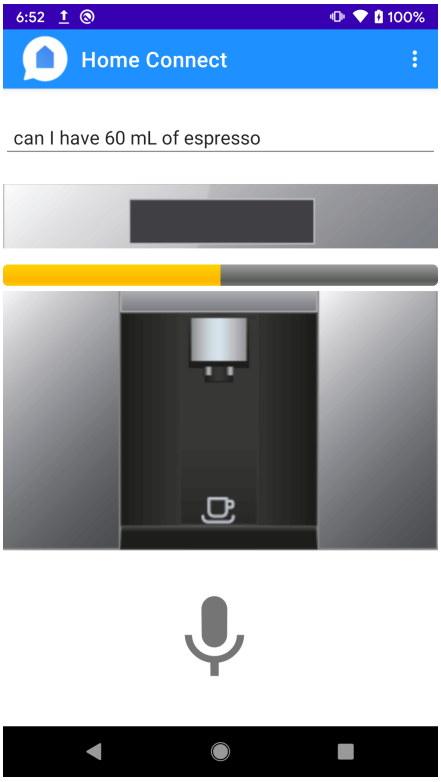

Figure 16: User request from app

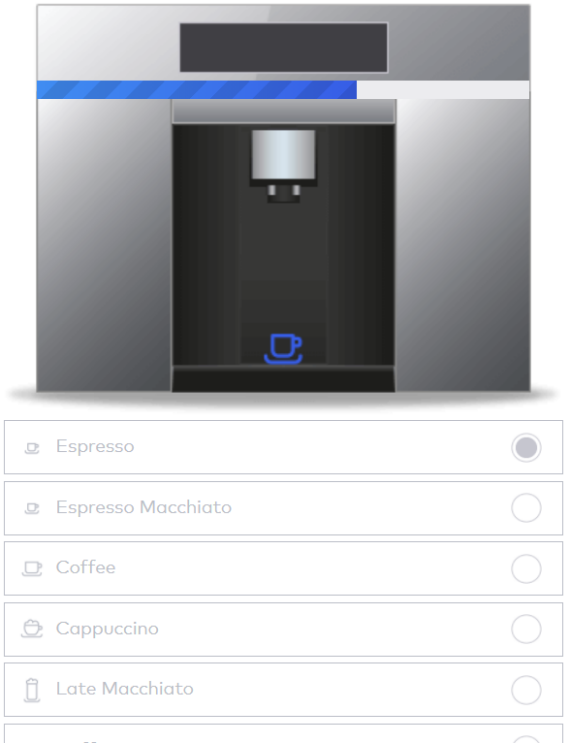

Figure 17: Web simulator working

The Figure 16 shows the process of the user requesting 60ml of espresso. The app captured user´s voice and it displayed it as a text on the screen. Basically, the user asked for 60ml of espresso. The yellow bar, on the other side, shows the request is processing, therefore the espresso is getting prepared. We can observe the Figure 17 to show that user´s request has been obtained correctly and is processing on the Web simulator side. The picture depicts that the correct product chosen by the user, namely the Espresso, is being prepared and the blue filling bar conveys the amount of time needed to get the product prepared.

So far we only showed that the Web simulator started the correct program (Espresso) precisely, but we did not illustrate the other parameters yet. Therefore, we are going to see this in the Figure 18 below.

In the Figure 18 below, we can first see the bar which describes the quantity of coffee type and below this bar there is displayed the number 60ml, which is exactly the quantity of Espresso the user asked for. Below the box describing the quantity, there is represented the box containing the other parameter, namely the coffee strength. Note that, the user in this example did not specify a certain strength value, so by default the app has been programmed to choose a *Normal* strength.

Figure 18: Espresso parameters

So, as we have seen, each coffee type has size and strength parameters and if the user does not specify them, in that case, the strength will be *Normal* as default. On the other side, if size is not specified as well, then the system is programmed to take a default size, which is different for each coffee, usually the average size.

7.1 Get available programs functionality

Besides getting a hot cup of coffee, the user also can ask for the programs available on that coffee machine. So, basically, ask for the type of coffee that they can get on that machine. The Figure 19 below describes the situation where the user asks for all the available programs. The app makes a request to the home appliance and after getting the results, displays them on a popup window visible to the user. The user then, by pressing *OK* button can close this window and proceed with the usage of the app.

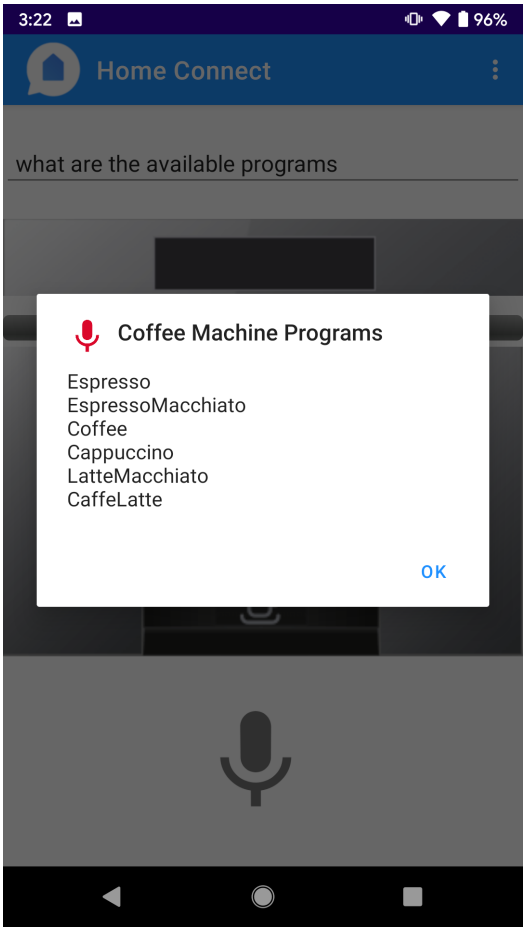

Figure 19: Available Coffee programs request

7.2 Warnings

In this subsection we will deal with some warnings displayed to the user to inform him/her of any incorrect functionality. As we saw previously, each coffee type has a range of quantity, i.e a cup of espresso only exists between 35 and 60ml. So what happens if the user asks or for 100ml of espresso. This situation is represented in the Figure 20 below. As the picture shows, the user requested 100ml of espresso, which is not available for this type of coffee, so the system displays a toast saying *You can choose Espresso size between 35 and 60 ml.*

Figure 20: Espresso size exceeding limit

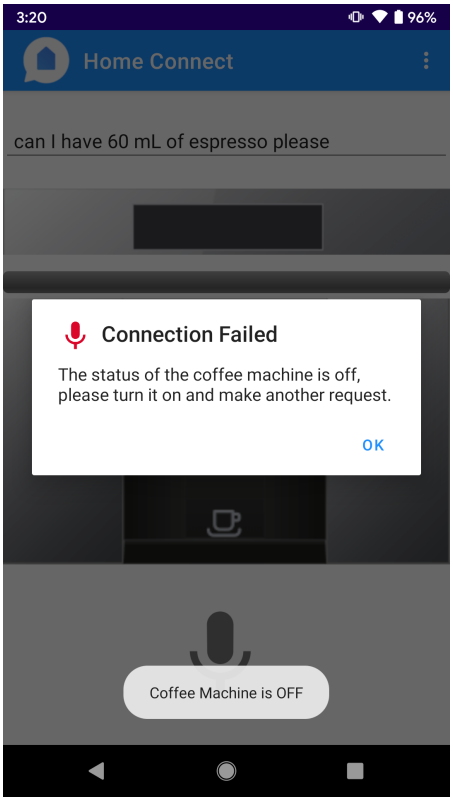

Figure 21: Coffee Machine Off

What we should question to ourselves also, when we want a product, is if the machine is online or not. The problem is that if the machine is offline, the first of all the coffee will not be prepared and second the user should be informed of this. Well, this situation is described in the Figure 21 below. In this case, we can see in the first line, that the user asked for 60ml of espresso. However, at the moment the coffee machine was offline and a popup window, with title *Connection Failed* opened to inform the user that the machine is offline. Also in the bottom part of the screen we can see a toast saying that the machine is OFF.

Another possibility could be that, in the moment that the user makes a request, the machine might be not connected to the network, so it would be unreachable. Also is this case we implemented a functionality that warns the user that the machine is not connected to internet. This situation is visible in the Figure 21 below. Here the user asked for the programs that the machine offers and a popup window informing about a problem reaching the machine was displayed.

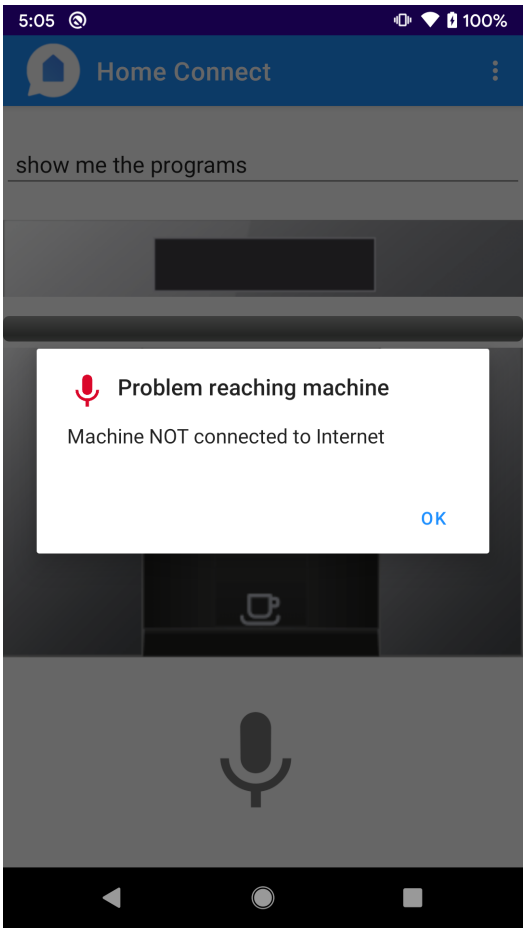

Figure 22: Coffee machine not connected to Internet

8 Conclusion

The project concludes an android applications which is used to control and monitor a coffee machine connected with Home connect API. Users can request different types of coffee, amount/quantity of coffee, strength of coffee through embedded voice recognition system only after providing correct credentials to log in and can check the types of coffee available through voice command or by checking the instructions by clicking on the info button in menu. If the coffee machine is off and

a user requests for a coffee then a pop-up message will be displayed indicating that the coffee machine is OFF. There exist a minimum and maximum quantity value for each coffee type and if user requests for a quantity which is lower than the minimum value or exceeds the maximum value of that particular coffee then a pop-up message will be displayed indicating that the requested quantity is not valid. In case, if no quantity and no strength will be requested/specified then default strength and default quantity will be selected i.e. normal strength and average quantity (different for each coffee type). The request activity page includes the text field showing the user's voice request or voice command, and the yellow filling colour in gray bar showing the processing of requested coffee.

References

[1] Home connect general information. https://api-docs.home-connect.com/general.

[2] Home connect quickstart guide. https://api-docs.home-connect.com/quickstart.