# **Summary**

Skill Barter App project was carried out under the NSSL Lab of Electrical Engineering Faculty of Technion.

The essence of the project is design and implementation of an android app that will help students to get assistance from their peers in various fields without the financial obligation.

The app contains many features, where the major ones are:

* Authentication and registration of users for personalization of the app.
* Usage of a real time database to store and retrieve constantly changing data.
* Filtered searches for required services and scheduling appointments.
* Intuitive and easy management of public user data.

The app’s logic is implemented with JAVA (logic) and XML (design).

The IDE used for this project is Android Studio, using Android SDK-version 28.

# **Project Goals**

As students ourselves, studying in a very demanding environment with very little or non-existent income, we often must give up on things such as:

* developing hobbies, especially new ones
* getting proper assistance in courses
* getting help with chores

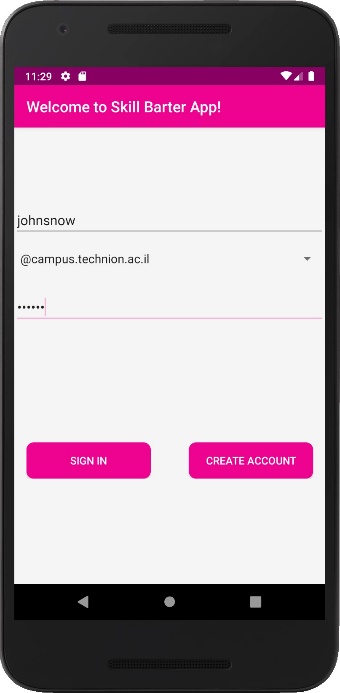
We wanted to allow students to share and exchange their skills and knowledge with their peers, on their own terms with no money involved. Each student that uses this app, can offer his/her skill set, on his/her free time without any strings attached. By supplying the services, you can earn points that will allow you to become a client. We believe in free market, so we give the users the freedom to decide what is their level of expertise and the points value they think they deserve, having in mind that other users can rate them afterwards.

We believe that this platform is a great way to develop and improve your skills (both as a “client” and as a “provider”), get assistance where you are struggling and last but not least – meet new friend based on your interests.

# **Implementation**

## **Authentication and Registrations of users**

On first entrance to the application, the user must create an account for future personal usage. As we decided to restrict the app for Technion students and personnel only, we require e-mail verification with Technion domain only.



1.1 - Authentication

On screen 1.1 the user must enter Technion username, choose the domain from a dropdown options and a password.

### **Create Account (new users)**

When “CREATE ACCOUNT” button is clicked, “onCreateAccountClicked” method is called. We validate the form (username and password must be filled correctly) and then we call to FirebaseAuth’s method “createUserWithEmailAndPassword(email, password)”. This triggers the following actions:

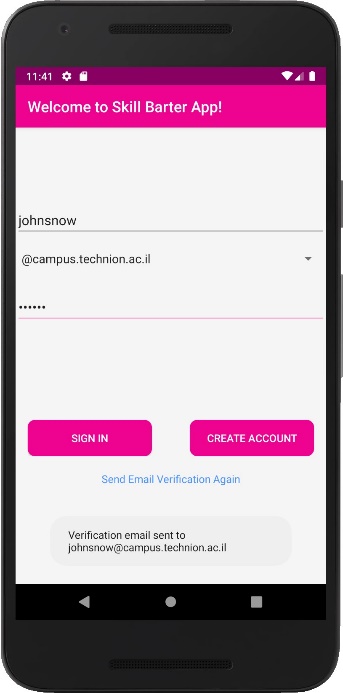
1. Verification that the e-mail address is not used by another user.
2. Validation of e-mail address syntax.
3. The user is added to users list (1.2) in Firebase Authentication service and the password is encrypted. Password encryption logic and configurations can be altered to increase the security level (using more complicated hash logic / fingerprint authentication etc.).

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

1.2 - Authentication service users list

1. The user is given a unique ID (User UID), that allows us to retrieve and edit user information from the DB for app’s different features.
2. Verification e-mail is sent to the e-mail address as show on screen 1.3.



1.3 - Verification e-mail sent

New users must now click the link in the verification e-mail as shown in pictures 1.4-1.5.

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

1.4 - Verification e-mail

תמונה שמכילה צילום מסך

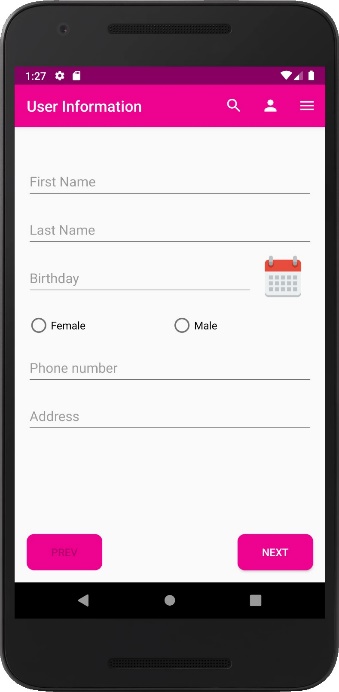
התיאור נוצר באופן אוטומטי

1.5 – Successful verification confirmation

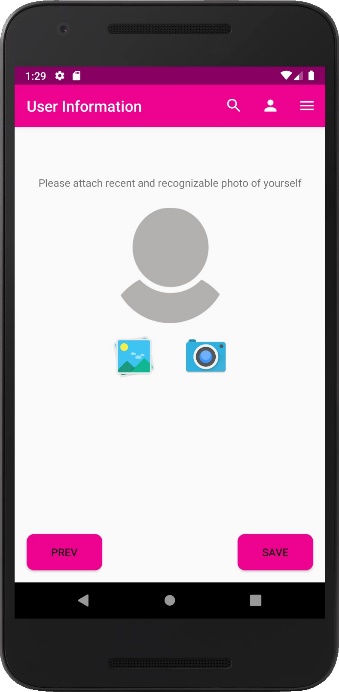
### **Sign-In**

Already registered users and those that verified the e-mail via verification link must click “SIGN IN” button. When the button clicked, the form is validated (as in “create account” case). To distinguish between these two states of user, the app checks whether “User Data” collection has a document with key that equals to current user’s UID. This check is not required for the authentication process of Firebase Authentication Services per se, but it is necessary for the app to be able to manage user’s data properly.

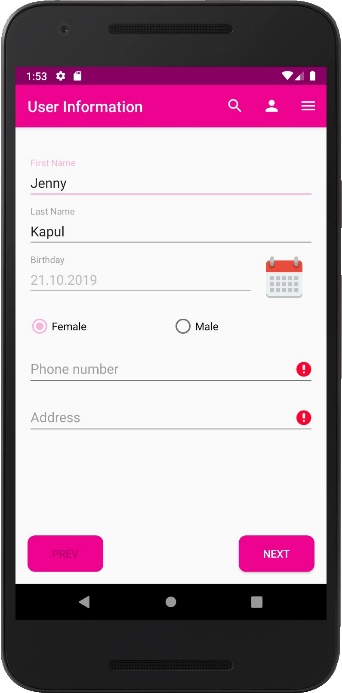
If “User Data” collection does not have current user’s document, the user must complete the registration by filling all the required information as shown on screens 1.6-1.7. If the field is left empty or filled with invalid data, the user won’t be allowed to proceed, and an error indicator will be shown next to the relevant field shown in screen 1.8. At any point, the user can stop the registration and sign out of the app (using menu bar option “sign out”), the progress won’t be saved, and all the data must be re-entered on the next registration attempt.



1.6 - User information I



1.7 - User information II



1.8 - Required field missing

Only after all mandatory fields are filled correctly, UserData object is created and added to “User Data” collection in the Firestore DB under the key that is equal to “user UID” of the current user. The picture is stored in Firebase Storage, and the document itself in the DB contains a URL to its’ location in the storage.

## **User Data Display and Management**

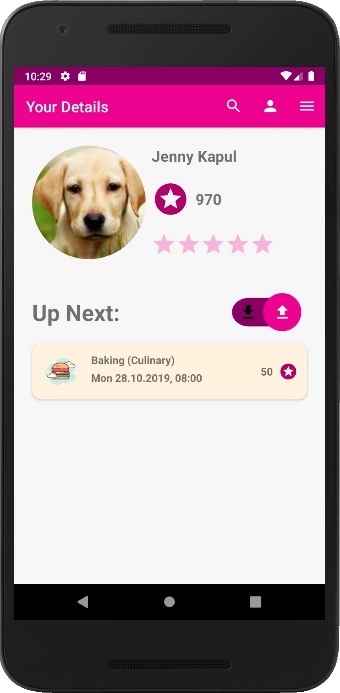
On signing in, successfully registered users are redirected to user home screen. If a signed in user did not actively signed out, he will remain in “online” state and will be automatically redirected to user home screen on app entrance.

### **User Home Screen**

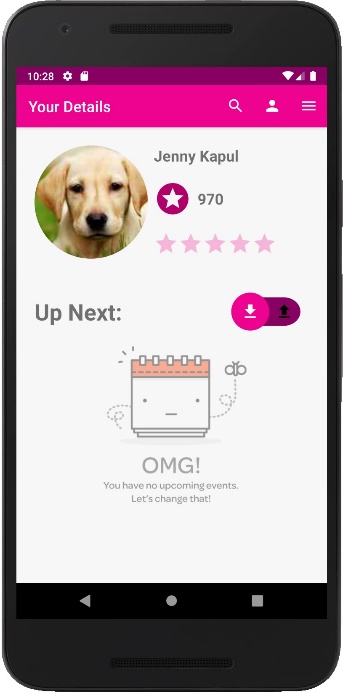
At the top of user home screen (2.1-2.2), all basic user profile information is displayed:

* Full name
* Current point balance
* Current average ranking
* Profile picture

This data is retrieved from the “User Data” collection in DB with query on current user UID. The data from the query is retrieved as “DocumentSnapshot” object and a listener is attached to it so that any change in the DB will be automatically reflected in user home screen.



2.1 User Home Screen Appointments as “provider”



2.2 User Home Screen Appointments as “client”

#### Points balance update logic

OnCreate of current activity, user balance is updated – and all points from past appointments in which he was the “provider” are summed to current points balance. Each appointment has a boolean field “providerPaid” which indicates if the points was already transferred to the provider. To minimize DB access, we update the provider balance only when the user opens the app and enters home screen by quarrying in documents in “Appointments” collection which uphold:

* providerUID == current user UID
* date < current date
* providerPaid == false

Thus, the points from an appointment where user was a “provider” will be added to the user’s balance on the first time he opens the app after the appointment has passed. After the points are transferred, the boolean flag is updated to the value “true”.

#### User Avg Rating

The rating bar (stars) display current user average rating. The rating is calculated every time new rating is submitted by other users. User Data document stores two fields related to this calculation:

* Avg rating
* numOfRatings

#### Upcoming appointments

Below user information, upcoming user’s appointments are displayed. The toggle is used to display separately “incoming” appointments in which the user is the “client” (toggle switch on the left) and “outgoing” appointments in which the user is the “provider” of the service (toggle switch on the left).

Appointments data is retrieved from “Appointments” collection in the DB and displayed inside a RecyclerView. Since we are using Firebase Firestore DB, it was only natural to use an adapter that is native to this DB – FirestoreRecyclerAdapter. This adapter monitors in real-time the items that should be displayed according to a supplied query. The query retrieves all appointments that uphold:

* date > current date
* if toggle selection is “outgoing” (right):
  + providerUID == current user UID

else (toggle selection is “incoming” (left)):

* + clientUID == current user UID

Since the placement of the toggle affects the query, a listener is attached to it, so that when the toggle selection is changed, the query is updated and the RecyclerView is re-initialized. This implementation ensures that any change in the DB will be immediately reflected on the screen.

#### Menu Bar

The menu bar at the top of the screen allows to navigate quickly and easily through the app’s features. The menu implemented on class named “ActionBarMenuActivity” which extends “BaseActivity” class. This allows us to avoid code duplication, so that any activity that should have menu bar will simply extend “ActionBarMenuActivity” class. Features that has access to menu bar:

* [User home screen](#_User_Home_Screen)
* Search
* [Skills manager](#_Skills_Manager_2)
* [Available Dates Manager](#_Available_Dates_Manager)
* History
* User Details Editor

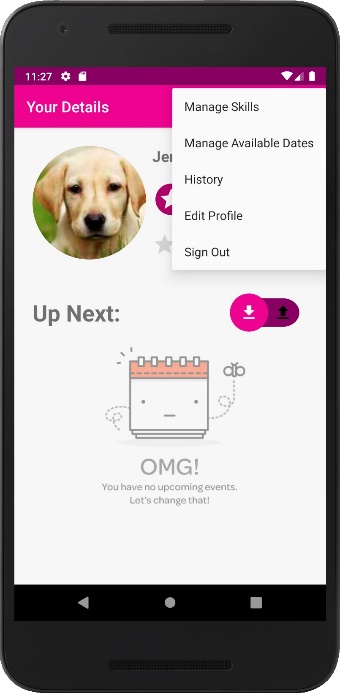
Features that can be accessed from menu bar are:

* תמונה שמכילה צילום מסך, אלקטרוניקה

  התיאור נוצר באופן אוטומטי - Search
* תמונה שמכילה צילום מסך, אלקטרוניקה

  התיאור נוצר באופן אוטומטי - [User Home Screen](#_User_Home_Screen)
* תמונה שמכילה צילום מסך, אלקטרוניקה

  התיאור נוצר באופן אוטומטי - drop down menu with options as shown in screen 2.3:



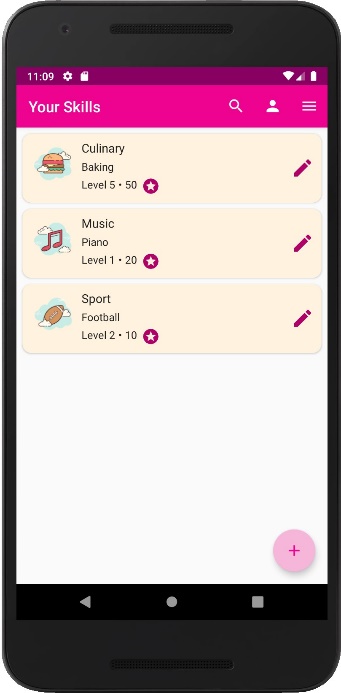
2.3 - Drop Down Menu

### **Skills Manager**

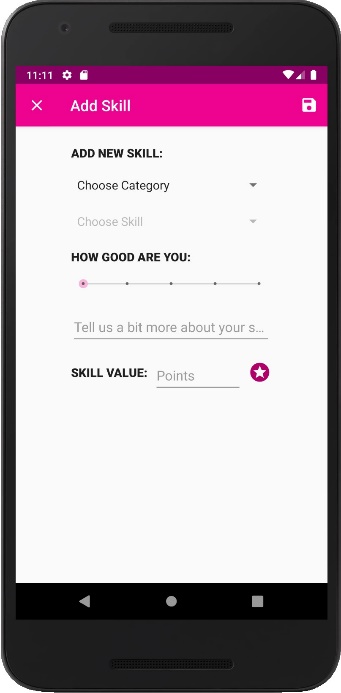
This screen is accessible from the drop-down menu of the menu bar. Skill manager allows the user to add and edit user skills that can be found by other users via search. User skills are stored in the DB under the collection “User Skills” and each document in this collection includes the following fields:

* Category
* Skill
* userID
* Level
* Points Value
* Is Enabled?

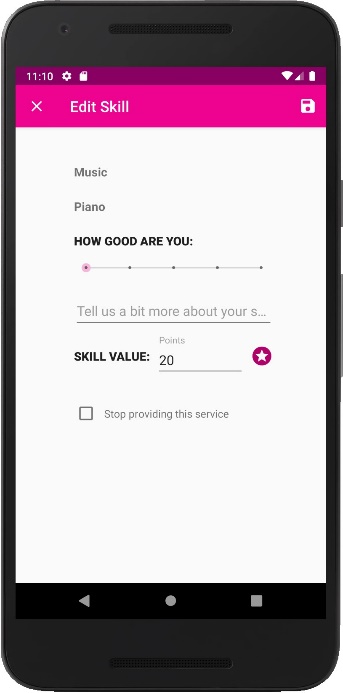
The key of the document determined by the unique combination of userID, category and skill. Unlike other fields, this fields are read-only after the skill is saved for the first time and cannot be edited later.



2.4 - Skill manager screen



2.5 - Add new skill screen



2.6 - Edit Skill Screen

OnCreate, the Recycler view is initialized and with documents retrieved from a query on “User Skills” collection that uphold:

* userId == current user UID

As in upcoming events RecyclerView – we use the native FirestoreRecyclerAdapter to implement this view and ensure that the view is up to date with the DB.

#### Add new Skill

In order to add new skill, user must click ‘+’ button at the bottom right screen of Skill Manager (2.4). The user will be redirected to 2.5 screen to add a new skill. All fields except “details” are mandatory and must be filled to properly add this skill. The “level” field is mandatory, but has a default value of “1”, so technically the user doesn’t have to change it. To prevent from users entering similar categories and skills by different names (or even spelling errors), and thus making the search more accurate and efficient, the category and the skills are chosen from spinner options. Since the list of skills is directly affected by the chosen category, we force the user to choose the category first (before category is chosen, skill spinner is disabled). After the category was chosen, the listener that is attached to this spinner, triggers a callback on changed choice, which enables the second spinner with list of skills that is relevant to the chosen category only. You can see that skills (second) spinner is disabled prior choice of category in screen 2.5, and it’s activation with the relevant list of skills is shown in screen 2.7:

תמונה שמכילה אלקטרוניקה

התיאור נוצר באופן אוטומטי

2.7 - Skill Spinner Activation

To save the skill, save icon at the top right corner of 2.5 screen must be pressed. To cancel the operation at any point (prior to saving), ‘x’ button at the top left corner of screen 2.5 must be clicked.

When ‘save’ is clicked, UserSkill object is created and added to “User Skills” collection with a key generated using the format “<userID>.<category>.<skill>”.

#### Edit Skill

In order to edit an existing skill, edit icon must be clicked on the relevant skill item from the RecyclerView. The user will be redirected to screen 2.6, and all the data related to that skill will be loaded from the DB. As mentioned before the category and the skill cannot be edited. The user can update details, his level in that skill and points value. If for some reason the user wishes to suspend his future not yet booked services of that skill, “Stop providing this service” checkbox must be checked. NOTE! Future already booked appointments for that skill will not be cancelled and the user is obligated to supply the service as agreed.

On save icon clicked, the document in “User Skills” collection will be updated. The edit can be canceled by clicking ‘x’ button.

### **Available Dates Manager**

This screen is accessible from