Draft Report

Mobile and Social Computing Course

PubEvents

(An Android Application Prototype)

Professors:
Ing. Gianluigi Folino
Ing. Andrea Vinci

Students:
Davide Amato (ID 199670)

Department of Mathematics and Computer Science, UNICAL

UNIVERSITÀ DELLA CALABRIA DIPARTIMENTO DI MATEMATICA E INFORMATICA

Academic Year 2020-2021

Contents

Objectives

PubEvents is essentially an application born to collect every information about pubs in one place, so that for the user is easier to keep track of pubs and the relative events. With this application you can easily find a pub, add it in your favorite pub list, see the upcoming events of a pub, buy tickets about these events and much more.
The app offers many features including:

- a map where you can visually find pubs close to your location
- the possibility to follow pubs for which you want to receive notifications
- a list of "*followed pubs*" to manage the pubs you are interested in.
- the possibility to buy tickets for events.
- a list that collects all next events for which you bought tickets.
- the possibility to receive suggestions about new pubs to follow as well as new events to participate in.

To fully enjoy all of these features the user needs to have:

- an active internet connection
- a personal account registered into the system, in order to save all his/her preferences and relevant information.
- device's GPS active, to find pubs and events close to him/her.

Context-Aware

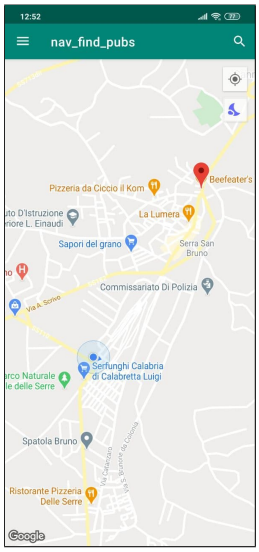
The **context-awareness**, specially today, is a crucial part in the development of an application. It allows software to change their behaviour and to adapt itself to a user to best fit his/her routines, behaviours, preferences. It is used to make an app "*context-aware*", i.e, aware, conscious of the surrounding "*context*" in which the user is *immersed*, so that the app can anticipate user's needs and the latter has less things to worry about, less actions to do. In this case, "context" can have many meanings. E.g, context could be the current "location" where the user currently is (the city, the store, the country, ...) with the relative weather conditions, time zone, etc. It can be also the user's calendar, schedule and plans.

The following paragraphs will give you an overview of the main "context-aware" features that *PubEvents* owns.

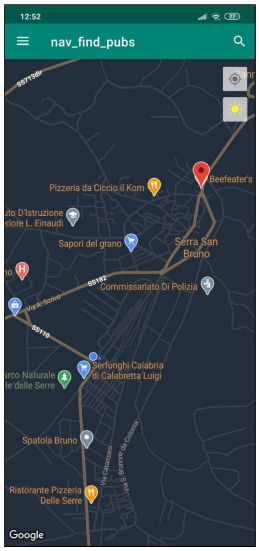
2.1 Geolocation

This app uses **user's current location** to show up the closest pubs and events by using a simple, visual map (figure on the side). By tapping on the red marks, user can access the relative pub/event page to see all the information. In order to make this feature work, user needs to allow the app to access device's position.

Figure 2.1: Pictures of the Map view



Day mode



Night mode

2.2 Local time

PubEvents uses the local time value to decide which style to give to the map view when user accesses it. There are two different styles available(figure on the side):

- **Day mode**: used when the local time is between 06:00h and 18:00h
- **Night mode**: conversely, it is used when the local time is between 18:00h and 06:00h

Moreover, user can manually change this setting by pressing the relative button.

2.3 User preferences

Some information about user preferences are stored and used in order to suggest him a list of pubs/events he could be interested in.
This suggestions are computed using parameters like:

- *location* and *overall rating** of the last pubs followed;
- *location*, *type* and *number of tickets owned** of the last events for which he bought tickets;

***Note**: The usage of these parameters has yet to be implemented.

Features & Implementation

3.1 Architecture

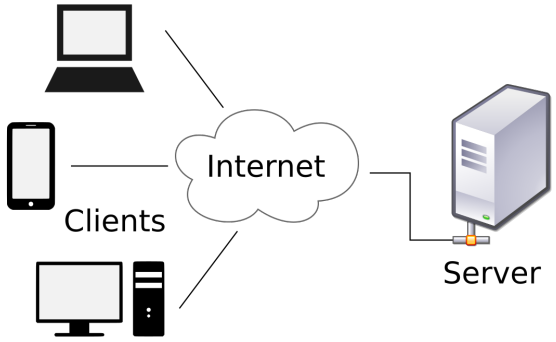


Figure 3.1: Client-server architecture model

The functioning of PubEvents is based on a **client-server architecture** (see figure above). In this kind of architecture the clients (user's android devices) communicate with the server through http requests, asking for/sending resources. In this project, **Firebase** has been used server side.

3.1.1 Firebase

Firebase is a platform developed by Google that offers many services that can be used via its API. The services used by PubEvents are:

- **Authentication**: it provides easy-to-use backend services to authenticate users to the app. Beside the standard email and password authentication, it supports authentication many different providers among which Google, Facebook and Twitter.
- **Cloud Firestore**: it offers an online, no-sql, document-oriented database, very flexible and scalable.
- **Cloud Storage**: this service is responsible for the upload/download of files.

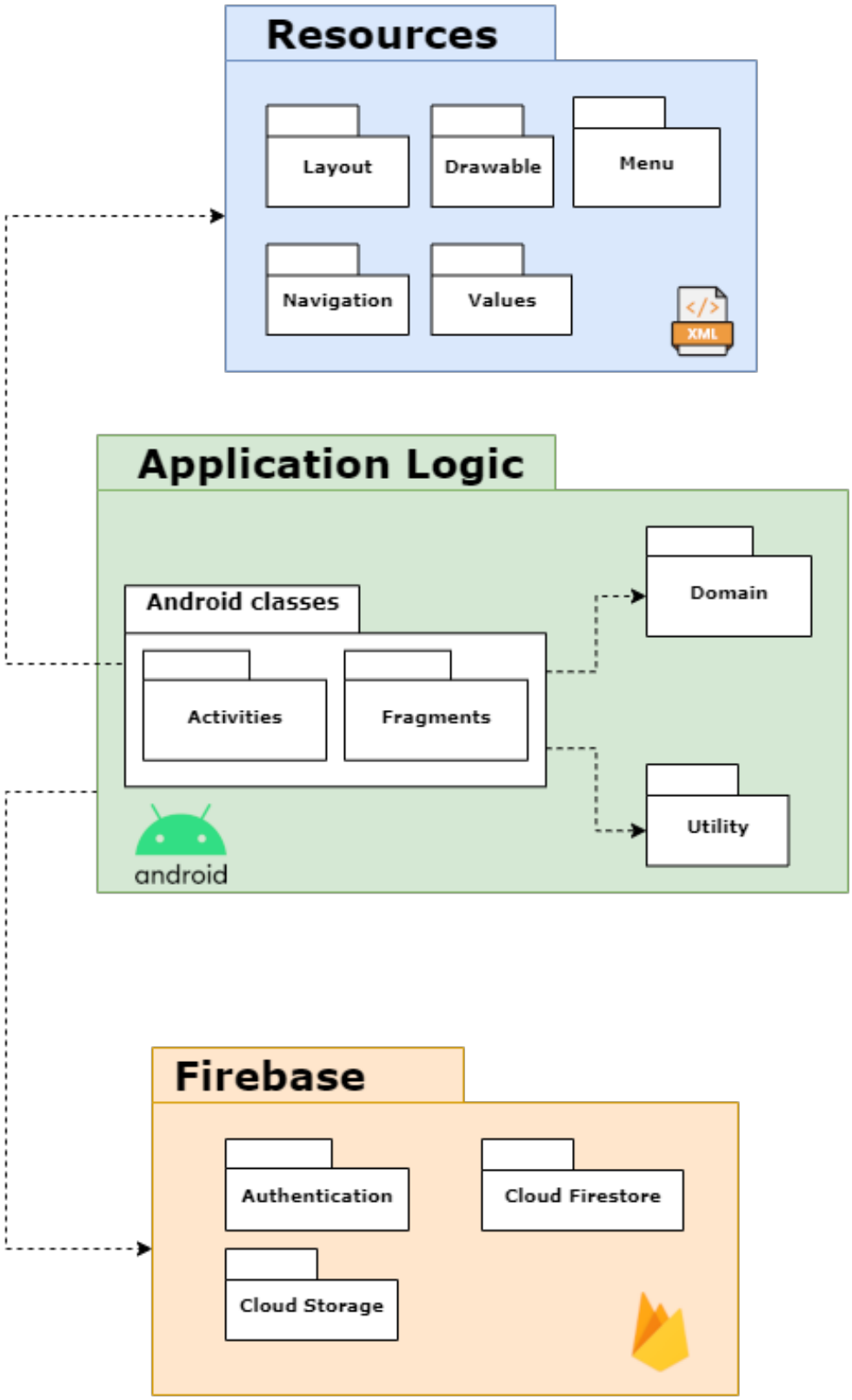


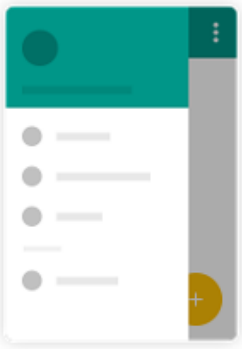
Figure 3.2: Software architecture diagram

3.2 Application structure

This application makes use of one main activity, the **Main activity**, and other two activities used to manage login and registration. In particular, the activity model chosen to implement this activity is the **Navigation Drawer Activity**.

3.2.1 Navigation Drawer Activity

This activity model is based on the **Navigation component**, part of the *Android Jetpack*'s libraries. The *Navigation component* helps to implement navigation (understood as the set of interactions users make in order to navigate across, into, and back out from the different pieces of content within the app). This component is composed by Three main parts:



Pictures of the *Navigation Drawer Activity* template

- **Navigation graph**: An XML resource that contains all navigation-related information in one centralized location. This includes all of the individual content areas within your app, called destinations, as well as the possible paths that a user can take through your app.
- **NavHost**: An empty container that displays destinations from your navigation graph. **Fragments** have been used as destinations.
- **NavController**: An object that manages app navigation within a NavHost. It guides the swapping of destination (Fragment) content in the NavHost as users move throughout your app.

3.2.2 Android classes

For the imlementation of PubEvents, both activities and fragments have been used.

Activities

There are three activities:

- **Main Activity:** activity that handles much of the application.
- **Login Activity:** it has control over the login process.
- **Register Activity:** it has control over the registration process.

Fragments

Here is the list of fragments created to manage the various features provided by PubEvents:

- **Home Fragment:** fragment that handles the application's home view.
- **Find Pubs Fragment:** fragment that encapsulates the Map view, used to search pubs visually.
- **Pubs List Fragment:** it is used to show up a list of pubs.
- **Events List Fragment:** it is used to show up a list of events.
- **Pub Fragment:** fragment that contains all the information relative to a pub
- **Event Fragment:** fragment that contains all the information relative to an event.

RecyclerView

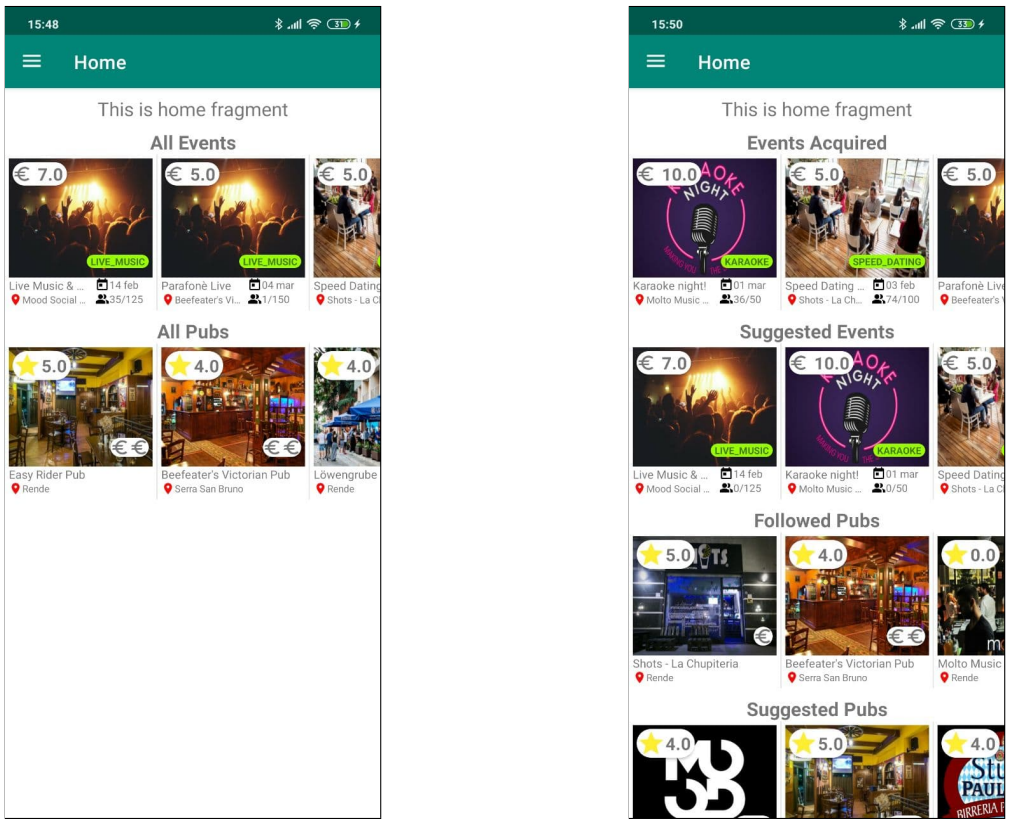
In this application a huge use of dynamic lists has been made and, to implement them, Android Jetpack's **RecyclerView** component has been used, so it deserves a bit more attentions. RecyclerViews are used to display large sets of data in dynamic lists in an easy, extendable and efficient way. This component is based on the co-ordination of four main classes:

- **RecyclerView:** it is the ViewGroup that contains the views corresponding the data. It's a view itself, so it can be added into a layout the way any other UI element can.
- **ViewHolder:** each individual element in the list is defined by a ViewHolder object. When the view holder is created, it doesn't have any data associated with it. After the view holder is created, the RecyclerView binds it to its data. The definition of a view holder can be done by extending **RecyclerView.ViewHolder** class.
- **Adapter:** the RecyclerView requests those views, and binds the views to their data, by calling methods in the adapter. The adapter can be defined by extending **RecyclerView.Adapter** class.
- **LayoutManager:** the layout manager arranges the individual elements in the list. One of the layout managers provided by the RecyclerView library can be used, or a custom one can be defined. Layout managers are all based on the library's LayoutManager abstract class.

3.3 App Flow

Home

First of all, a distinction can be made if the user **is** or **isn't** logged into the system. Based on the fact that the user **is/isn't** logged into the system when the application is launched, the home view may differs, showing one of the views in figure.



(a) Guest user

(b) Logged in user

Figure 3.3: "Home" view comparison between *Guest* and *Logged in user*

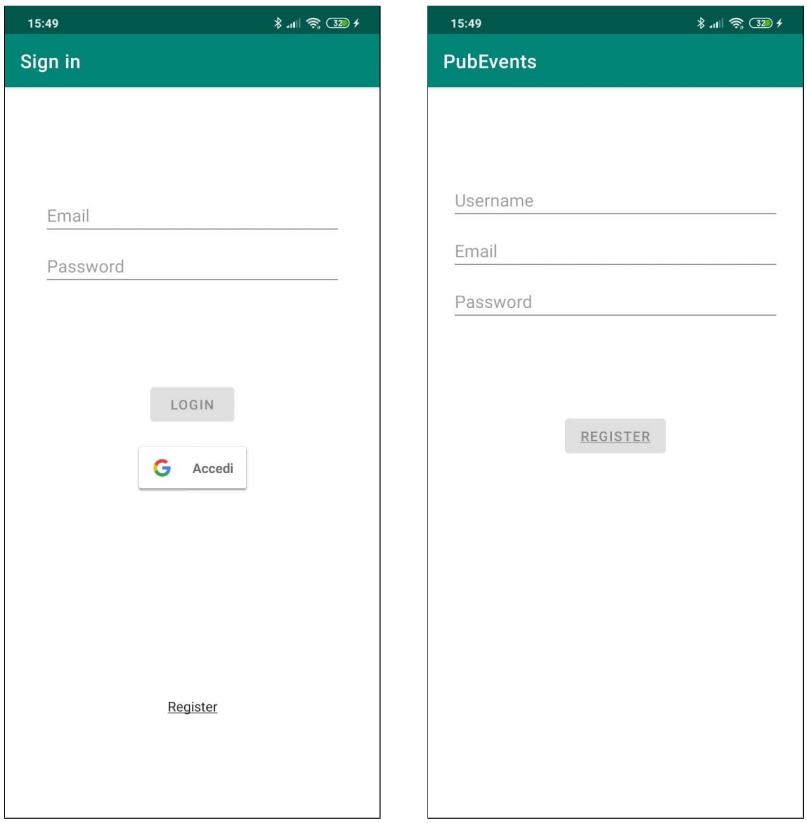
Home view owns four different containers for four different lists:

- **Events Acquired container:** this container is filled with the list of all events for which the user bought some tickets. This container remains empty if the user is a guest user.
- **All/Suggested Events:** this container is filled with the list of all events if the user is a guest, otherwise it is filled with the "suggested" events for that user. The suggestion process of the events will be explained later.
- **Followed Pubs container:** this container is filled with the list of all the pubs the user is following. This container remains empty if the user is a guest user.

- **All/Suggested Pubs container:** this container is filled with the list of all pubs if the user is a guest, otherwise it is filled with the "suggested" pubs for that user. The suggestion process of the pubs will be explained later.

Login and Registration

To fully exploit all the features provided by this application, user need to be logged into the system, and therefore he/she need to have an account registered. For this reason, PubEvents provide a **Login** and a **Registration page**. The app gives the possibility also to perform the login/registration using his/her own google account.



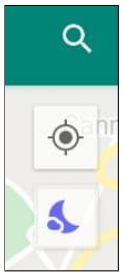
(a) Login page. (b) Registration page

Figure 3.4: Picture representing Login (left) and Registration (right) pages

Find pub

This fragment is accessible both for guest and logged in users. The expected view is the one showed in the figure below.

(a) Picture of the "Find Pub" view.



(b) Buttons to trigger actions on the "Find Pub view"

Figure 3.5: Picture of "Find Pub" view and the relative buttons

This fragment contains a map (implemented using the Google API) where the user can easily find pubs close to him. From this view, by pressing the buttons in the top right corner of the view (figure above), the user can also perform the following actions:

- **Localize himself:** user can localize his/her position in the map by pressing the "my location" button (the button in the middle of the image (b) above).
- **Change map style:** user can switch between *Day* and *Night map mode* by pressing the relative button located below the "my location" button.
- **Open the Pubs List Fragment:** user can perform this action by pressing on the search icon in the right corner of the app bar, right above the "my location" button.

When the user accesses this fragment and he/she has not already granted GPS permission for this app, the dialog shown in the figure below will show up, giving the possibility to grant the permissions.

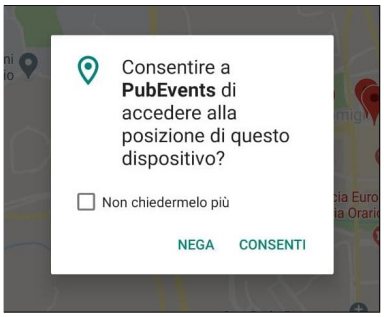


Figure 3.6: Figure of the GPS Permission dialog.

Find Pubs List

This fragment can be reached by navigating in the Find Pub fragment and pressing the search icon in the top-right corner. The fragment will be presented as shown in the figure below.

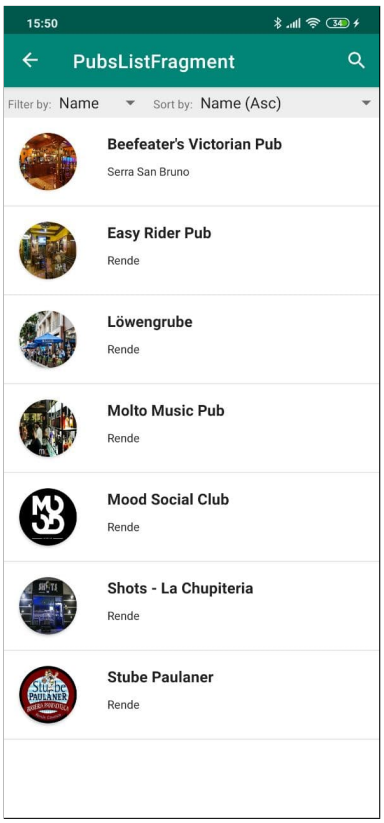


Figure 3.7: Representation of the "Find Pub List" view.

This view allow user to search pubs typing the name in the search bar. Moreover, user can also filter and sort the result with different options.

Filter: there are two filter options available:

- **Name:** the list will be filled with pubs whose names match the string typed in the search box.
- **City:** the list will be filled with pubs whose city's names match the string typed in the search box.

Sort: user can choose one among the sort options listed below:

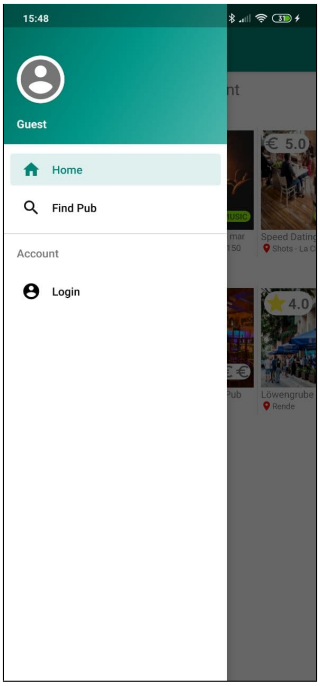
- **Name (Asc)/(Desc):** the list will be sorted in ascending/descending order based on pubs' name.
- **City (Asc)/(Desc):** the list will be sorted in ascending/descending order based on the cities' name of the pubs.
- **Proximity (Nearest)/(Furthest):** the list will be sorted from Nearest to Furthest/Furtest to Nearest based on the distance between pubs and user location. In order to use this filter, user must have granted the GPS permissions for this app.
- **Overall Rate (Asc)/(Desc):** the list will be sorted in ascending/descending order based on pubs' overall rate.
- **Price (Asc)/(Desc):** the list will be sorted in ascending/descending order based on pubs' price value.
- **Avg. Age (Asc)/(Desc):** the list will be sorted in ascending/descending order based on pubs' average age.

Navigation Drawer

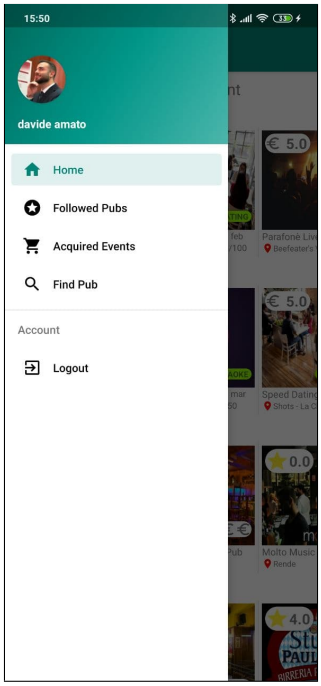
The navigation drawer displays different options based on user status (*guest* or *logged in*).

Followed Pubs

This fragment contains the list of all pubs followed by the user. This view can be reached only be logged in users. This list can be filtered and sorted in the same way as for the "Find Pubs" list.



(a) Guest user



(b) Logged in user

Figure 3.8: "Navigation Menu" view comparison between *Guest* and *Logged in user*

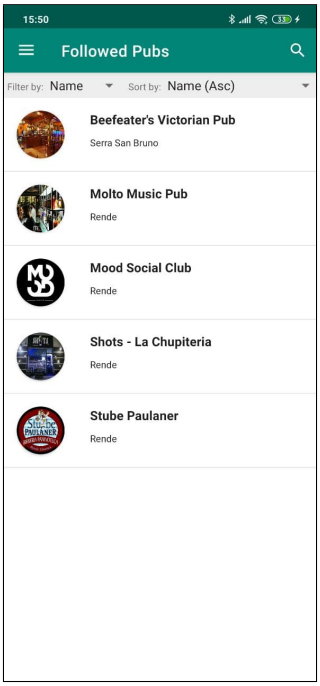


Figure 3.9: Representation of the "Followed Pubs List" view.

Acquired Events

The fragment will be presented as shown in the figure below.

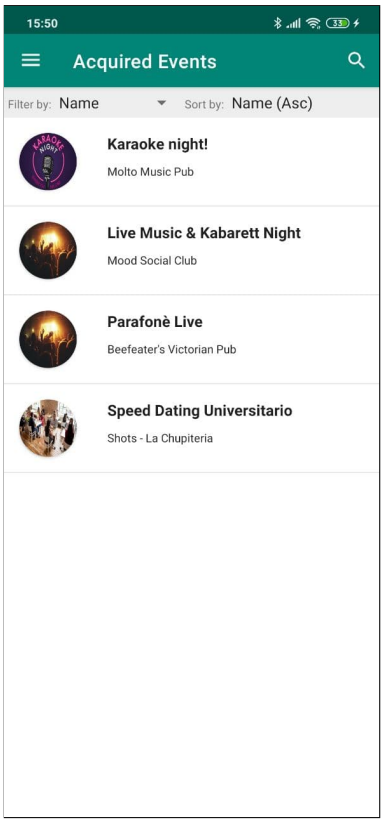


Figure 3.10: Representation of the "Acquired Events List" view.

This view allows user to visualize all the events for which he/she bought some tickets and allows also to search among them typing the name in the search bar. Moreover, user can filter and sort the result with different options.

Filter: there are two filter options available:

- **Name:** the list will be filled with events whose name match the string typed in the search box.
- **Pub name:** the list will be filled with pubs whose pub's name match the string typed in the search box.

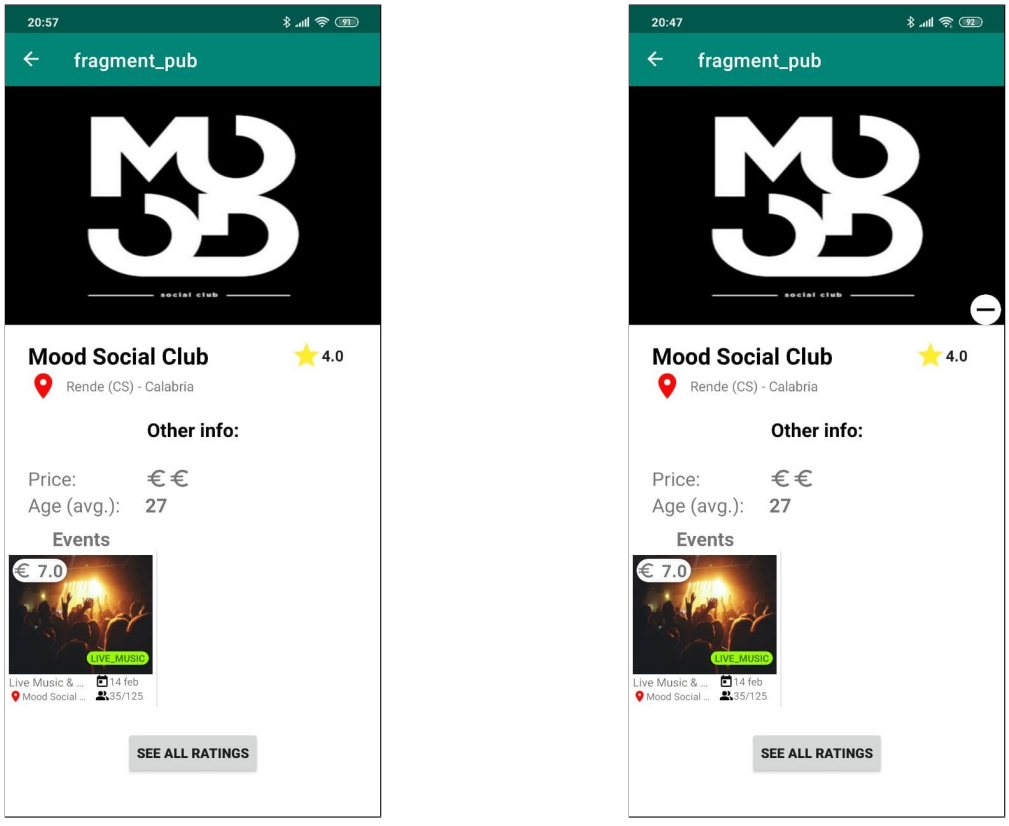
Sort: user can choose one among the sort options listed below:

- **Name (Asc)/(Desc):** the list will be sorted in ascending/descending order based on pubs' name.
- **Date (Nearest)/(Furthest):** the list will be sorted from Nearest to Furthest/Furtest to Nearest (in time) based on the distance (in days) between events and current date.

- **Pub name (Asc)/(Desc):** the list will be sorted in ascending/descending order based on the pubs' name of the events.
- **Proximity (Nearest)/(Furthest):** the list will be sorted from Nearest to Furthest/Furtest to Nearest based on the distance between events and user location. In order to use this filter, user must have granted the GPS permissions for this app.
- **Price (Asc)/(Desc):** the list will be sorted in ascending/descending order based on events' price.

Pub page

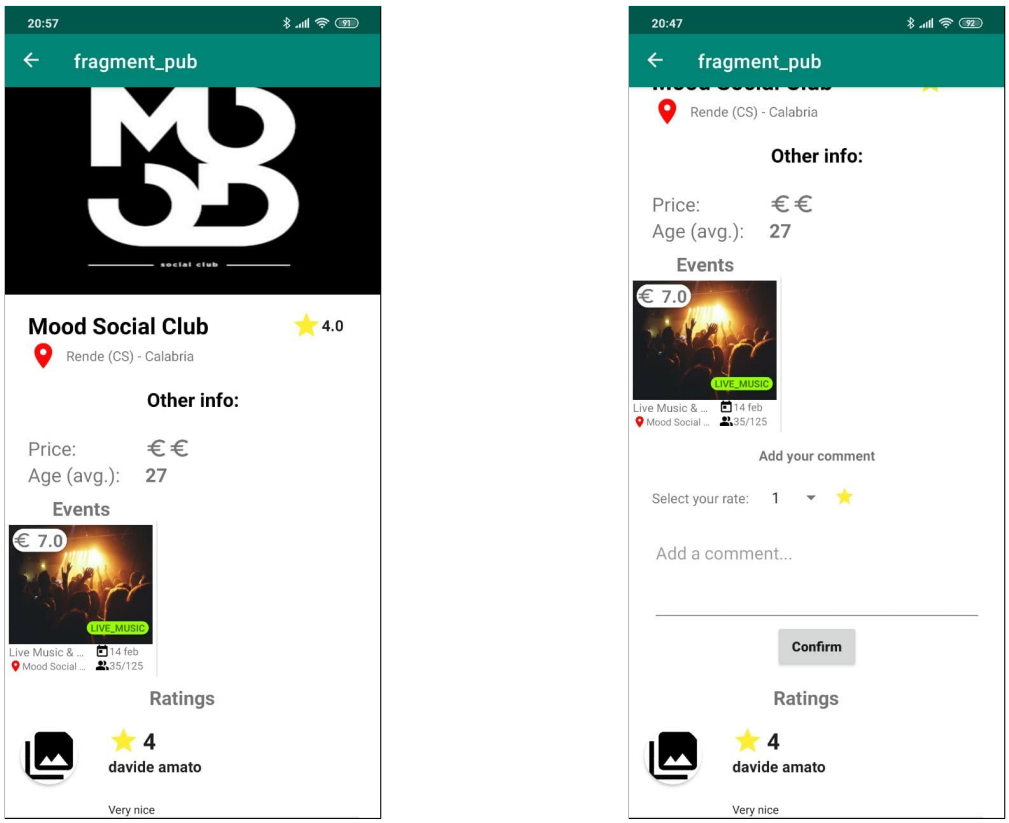
Whenever a user tap on a pub preview item in a list (e.g., in the Home, Find pubs or Followed pubs lists) or tap a pub's marker in the map view, he/she will be redirected to its page. When the user open a pub page, he/she can check the information relative to it, see the list of next events that will take place there, adding/removing it from the "followed pub" list (feature abailable only for logged in user) by pressing the "plus/minus" button at the bottom-right of the pub image and, by pressing the button "See All Ratings", he/she can give it a rating with a comment (feature available only for logged in user) and see all the ratings done by other users. An example of the page of a pub is given by the figure below.



(a) Guest user

(b) Logged in user.

Figure 3.11: "Pub page" view comparison between *Guest* and *Logged in user*



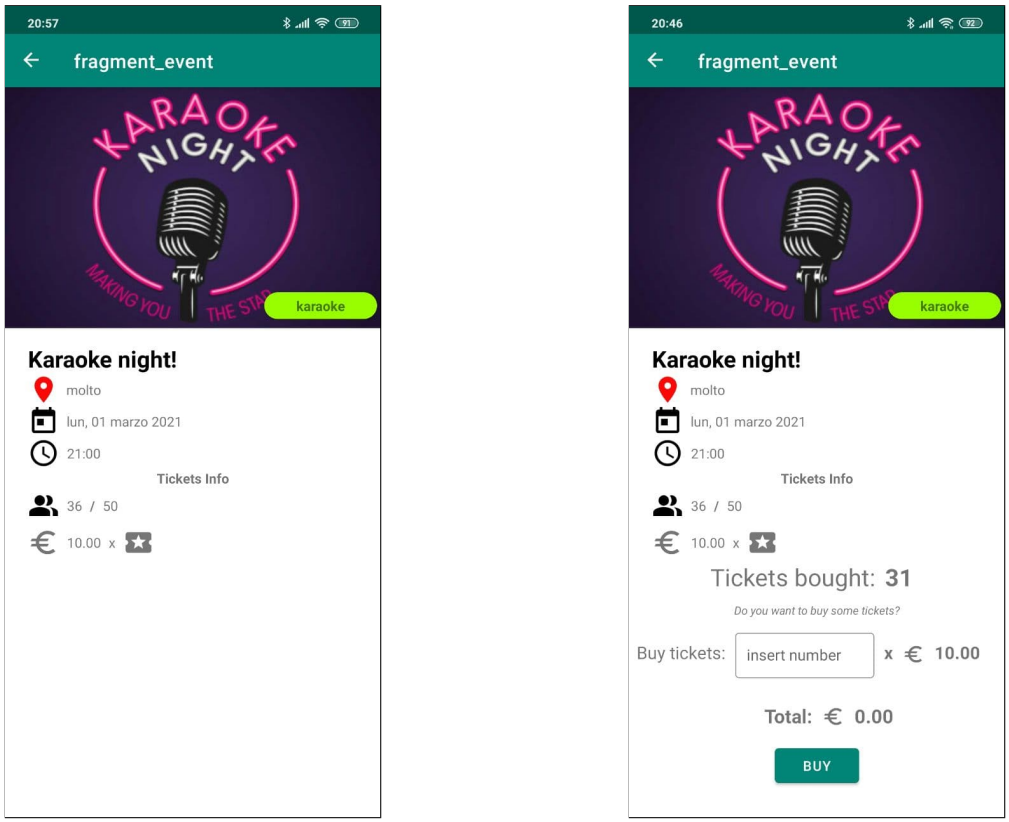
(a) Guest user

(b) Logged in user.

Figure 3.12: "Pub page" view comparison between *Guest* and *Logged in user* after pressing "See All Ratings" button

Event page

Opening a page relative to an event, the user can visualize all the most important information and, if the user is logged in, show up a section that allows user to buy some tickets. The figure below gives an example of an event page visualized from both guest and logged in user.



(a) Guest user

(b) Logged in user.

Figure 3.13: "Event page" view comparison between *Guest* and *Logged in user*

3.3.1 Suggestions

PubEvents is able to suggest pubs and events to the logged in user by computing functions that give a score to each pub and event based on some parameters and choosing the ones with the highest scores. Important factors that come into play for choosing the pubs or events to suggest are the *last pubs followed* by the user and the *last events* for which the user bought some tickets. The score is computed in a slightly different way for pubs and events. So, the core of the algorithm is the same and is described below.

General Suggestion Algorithm (GSA)

Here, Pubs and Events are generally considered as **Items**, while **Candidates** are object composed by an Item and a score value. The suggestions are computed in the following way:

1. Create a list of **candidate selectors**.
2. Retrieve the list of **all items** from DB.
3. Create a list of **candidates** from the *all items* list, setting the score of each candidate to 0.
4. Retrieve the list of **last items** from DB.
5. **For each candidate selector** in *candidate selectors*:
   (a) **Compute internal parameters** based on *last items* list.
   (b) **Compute the score** of each candidate of the *candidates* list.
6. **Sort** *candidates* list in descending order based on their score.
7. **Output the sorted candidates list**.

Candidate Selectors

A candidate selector is an object the aim of which is to compute a score for each candidate of the list based on the type of selector it is. In order to compute the score, it must first compute some internal parameters. Detailed information will be given later.

Pub Suggestions

Pubs suggestions are computed by using the GSA where *items* are instances of *pubs* and there are some other details that deserve more attention:

- the list of *candidate selectors* (GSA's 1. point) contains only a **Distance Candidate Selector**.
- the *all items* list (GSA's 2. point) is actually the list of *all pubs*.
- the *last items* list (GSA's 4. point) coincides with the list of *last followed pubs*.

Distance Candidate Selector (DCS): this type of selector has two internal parameters:

- a *double* **MAX_DISTANCE**: this parameter is a constant whose value is 100000.
- a *GeoPoint* **meanPoint**: this parameter is computed during GSA's 5.(a) point.

DCS computes the internal parameter *meanPoint* value in this way:

DCS Parameter Algorithm (DCS PA)

1. Initialize doubles avgLat = 0.0 , avgLong = 0.0 .
2. For each item i in *lastItems** list:
   (a) avgLat += i.getLatitude();
   (b) avgLong += i.getLongitude();
3. avgLat /= lastItems.size();
4. avgLong /= lastItems.size();
5. **meanPoint** = new GeoPoint(avgLat,avgLong);

***Note**: DCS is a candidate selector used for both pubs and events suggestions. Hence *lastItems*, in case of pubs coincides with *last followed pubs* and in case of events will coincides with *last events for which user bought some tickets*.

After calculating internal parameters, DCS computes the score for each candidate (GSA's 5.(b) point) in this way:

DCS Score Algorithm (DCS SA)

1. For each candidate c in candidates:
   (a) Double distance = distanceBetween(c.item.getGeoPoint(), meanPoint);
   (b) if distance $\geq$ MAX_DISTANCE:
      i. candidates.remove(c);
   (c) else:
      i. double partial_score = ScoreCalculator.computeScoreInverse(3, 1, MAX_DISTANCE, dist);
      ii. c.setScore(c.getScore() + partial_score)

As can be seen from DCS SA's 1.(b)-i. point, if the distance between a candidate's location and *meanPoint* is greather or equal than MAX_DISTANCE, the candidate will be removed from the list. This is done because candidates that are too far away from the meanPoint are not considered interesting for the user and, also, in order to lighten computations of the next selectors.
The partial score that has to be added to the current candidate's score is computed by **computeScoreInverse(***rangeMax, rangeMin, maxDomainValue, currentDomainValue***)(CSI)** function where:

- *rangeMax* and *rangeMin*: these two values indicate the range within which we want to obtain the score. In this case we want to obtain a distance score value between 3 to 1.

- *maxDomainValue*: this is the maximum distance value that is considered
- *currentDomainValue*: it is the distance between the current candidate and *meanPoint*.

In this case, DCS uses an *inverse* function because it assigns a score between a **MAX** and a **min** value where **MAX means <u>minimum distance</u>** from *meanPoint* and **min means <u>maximum distance</u>** from *meanPoint*.
In detail, the mathematical function behind CSI is the following:

$$\frac{rangeMax}{\textbf{computeScore}(rangeMax, rangeMin, maxDomainValue, currentDomainValue)}$$

ComputeScore (CS) is a function that returns a score between a **MAX** (*rangeMax*) and a **min** (*rangeMin*) value where **MAX means <u>maximum distance</u>** from *meanPoint* and **min means <u>minimum distance</u>** from *meanPoint*. For this reason, to give highest score to candidates with minimum distance, this function is inverted in **CSI**.

Events Suggestions

Events suggestions are computed by using the GSA where *items* are instances of *events* and:

- the list of *candidate selectors* (GSA's 1. point) contains a **Distance Candidate Selector** and a **Event Type Candidate Selector**.
- the *all items* list (GSA's 2. point) is actually the list of *all events*.
- the *last items* list (GSA's 4. point) coincides with the list of *last events for which user bought some tickets*.

Distance Candidate Selector (DCS): it has already been explained in detail (section 3.3.1 - Pub Suggestions).

Event Type Candidate Selector (ETCS): this type of selector has one internal parameter, an *HashMap<EventType, Double>* named **typeScores**. This hashmap will contain a score for each type of event. The score of a type of event is based on the quantity of events of that type among the last events for which the user bought some tickets. The hashmap is filled during GSA's 5.(a) point in the following way:

ETCS Parameter Algorithm (ETCS PA)

1. typeScores = new HashMap<>();
2. For each event *e* in lastEvents:
   (a) if *e*.getType() is not in typeScores:
      i. typeScores.put(*e*.getType(), 0);
   (b) else:

i. Double tmp = typeScores.get(*e*.getType());
ii. tmp += 1.0;
iii. typeScores.put(*e*.getType(),tmp);

3. double maxOccurrences = getMaxValue(typeScores);
4. For each eventType *et* in typesScores.keySet():
   (a) double normalized_score = ScoreCalculator.computeScore(2,1,maxOccurrences,typeScores.get(t));
   (b) typeScores.put(*et*, normalized_score);

As you can notice, ETCS PA first fills the hashmap assigning to each event type the relative number of occurrences among the last events (ETCS PA's 2. point). Later, it will take the maximum number of occurrences (ETCS PA's 3. point) and use it as MAX value in **computeScore** function (already seen in section 3.3.1 - Pub Suggestions - DCS SA).
In this case we don't need the inverse function (CSI). In fact, The final score of each event type in typeScores is calculated taking the normalized value between 2 and 1 (ETCS PA's 4.(a) point) where **2 means maximum number of occurrences** and **1 means minimum number of occurrences**.

After calculating internal parameters, ETCS simply assigns the score to each candidate (GSA's 5.(b) point) in this way:

ETCS Score Algorithm (ETCS SA)

1. For each candidate *c* in candidates:
   (a) Event e = *c*.item;
   (b) *c*.setScore(typeScores.get(e.getType()));

Related Work

Two of the most used platforms that offer many of the features provided by PubEvents are **Google** and **Facebook**. They are free and widely used all over the world but they are not specifically suited for this domain.
The most accurate alternative that I found and that is already present in the Android Marketplace is "**HypeSpots - Find Clubs, Pubs and Events Near You**". The name is pretty self-explenatory. But it is not used world wide. Its last update dates back to the 8th of February 2020 and it counts only 500+ downloads.

Future Work

The app is just a prototype. This means that many feature are not already implemented and many things need/have to be polished/fixed. Among the features that are not already implemented but may be in the future we can list:

- Visual directions shown in a map that help user to reach a pub/event.
- Notification system that notify user about imminent events and that suggest some event or pub close to him/her.
- Social media features like: share a pub/event with friends, chats.
- Add the possibility to login/register with Facebook and other accounts.

Instead, among the things that have to be polished/fixed we can list:

- Choose a proper design, logo, colors, structure and so on.
- Make the suggestions system more effective and accurate.