Flats

Design and Implementation of Mobile Applications

Design Document

A.Y. 2021/2022

Alessandro Polidori, Fabio Stecchi

POLITECNICO MILANO 1863

Contents

1 Introduction

1.1 Purpose

This is the *design document* (DD) of ***Flats*** application developed by *Alessandro Polidori* and *Fabio Stecchi* in the context of the *Design and Implementation of Mobile Application* course at Politecnico di Milano.
The document explains the most important design choices we made and the motivations behind them.

1.2 Scope

Flats is a multi-device application for smartphone and tablet and, the idea behind it, is to provide those who are looking for an apartment or a house to rent, a single platform offering the opportunity both to find and contact the owners and the possibility to post ads with the aim to find housemates to share the home and its rental cost.
The idea was born from the difficulty of finding houses to rent and housemates: in fact, nowadays, people are having hard time to find the right solution because they are disoriented by the several different sources to be analyzed (agency, private ads on the web, private ads on the social media). Moreover, due to the high investment involved, the request to share the rental cost is high but the market does not offer any tool which allows the possibility to easily find a housemate. The research of housemates is currently managed by word of mouth and it takes time. Flats is the ideal solution which allows to solve this need by taking advantage of a single application. Time saving, accurate info, real time updates just in one click.

1.3 Features description

The functionalities implemented in the app, trying to follow a possible order of interaction, are:

1.3.1 House navigation

It is the first screen shown when the app is loaded and also the only that can be used without the login or registration. It consists of a map of the city filled with markers associated with all available rental houses in that area. Users can click on the markers to obtain further information regarding the selected apartment.

1.3.2 Signup

If users are not yet registered, the signup screen will be shown when they will try to use the other app functions. They will be requested to enter their email and password in order to create their own profile.

1.3.3 Log in

The log in allows users to enter their credentials and log into the system to use all the features provided by the application.

1.3.4 Create Post

Users who find their ideal house solution and are also interested in finding a housemate can create a post by clicking on a button in the drop-down which contains information about the apartment in the House section. The post created will be shown to everyone in the Social section. Each post will be automatically linked to the apartment from which it was created and will contain information about it.

1.3.5 Send message

Users can start a chat in three different ways:

1. by looking for the user’s name in the proper bar in the chat section of the app.
2. by contacting the host of an apartment through a button in the window containing the details of an apartment in the House section.
3. by contacting a potential housemate by clicking the proper button on the respective post in the Social section.

1.3.6 Manage house

The last section of the app concerns those who want to put their property for rent. In this section hosts can add their property by entering the ad name, price, description and possible photos. In addition to this, hosts can also manage any property previously loaded with the possibility to delete it.

2 Architectural Design

We have decided to build ***Flats*** using Flutter. Flutter is an open source framework developed by Google for building natively compiled and multi-platform applications from a single codebase. Flutter is powered by Dart, a language optimized for fast apps on any platform. Our application uses some firebase backend services like Firebase Authenticator and Cloud Firestore and different Google APIs such as Google Places API and Google Maps API.

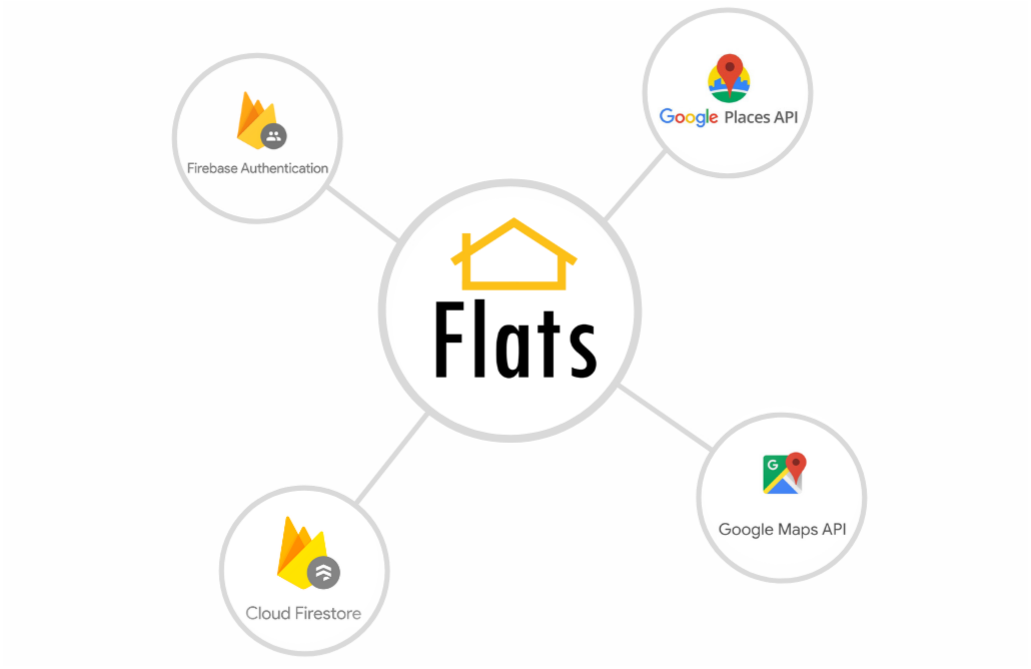


Figure 2.1: Architecture overview

2.1 Firebase Authenticator

We have chosen Firebase Authenticator because it provides easy-to-use back-end services to authenticate users to our app.
We have opted for an Email and password based authentication. Once a user

signs in for the first time, a new user account is created and linked to a new unique user ID that identifies the user across our app.

2.2 Cloud Firestore

We have chosen Cloud Firestore to store and retrieve all the data. Cloud Firestore is Firebase's newest database for mobile app development. It is a NoSQL, document-oriented database that means that data are stored in documents organized into collections. Unlike a SQL database, there are no tables or rows. Each document contains a set of key-value pairs.
All our data are organized in 4 different collections and their structure is described here below.

ChatRoom is the collection used for the chat service. It contains documents that are characterized by the email of the two users involved in each chat. The best way to store messages in this scenario is by using subcollections which means a collection associated with each document. In addition to this, each document has also different fields such as:

- *emails*: an array that contains emails of both users.
- *lastMessage*: it contains the last message.
- *lastMessageSendBy*: the author of the last message.
- *lastMessageSendTs*: timestamp of the last message.

Location is the collection used to storage data relative to each available apartment/house. Each document is associated to a single unit and it is characterized by a uniqe auto-generated ID when a unit is added by a user. Each document contains the following fields:

- *description*: it contains a description of the rental ad.
- *hostMail*: it contains the mail of the user creating the ad.
- *location*: it contains the coordinates of the location in GeoPoint type.
- *name*: it contains the name of the rental ad.
- *price*: it contains the rental price.
- *uid*: it contains the userID of the user creating the ad.

- *urls*: it is an array containing all the urls of the image uploaded by the user.

Post is the collection used to handle the social section. When a user creates a Post, a new document with a uniqe auto-generated ID is created as well. Each document contains the following fields:

- *content*: it contains the content of the post.
- *flatId*: it contains the ID of the document of the location to which the post is associated. This Id is used to get the details of the apartment.
- *title*: it contains the title of the post.
- *uid*: it contains the userID of the user creating the post.
- *userMail*: it contains the mail of the user creating the post.

User is the collection containing information about each user who has a single dedicated document holding the following fields:

- *email*: email of the user.
- *pic_url*: url of the profile image of the user.
- *username*: username of the user.

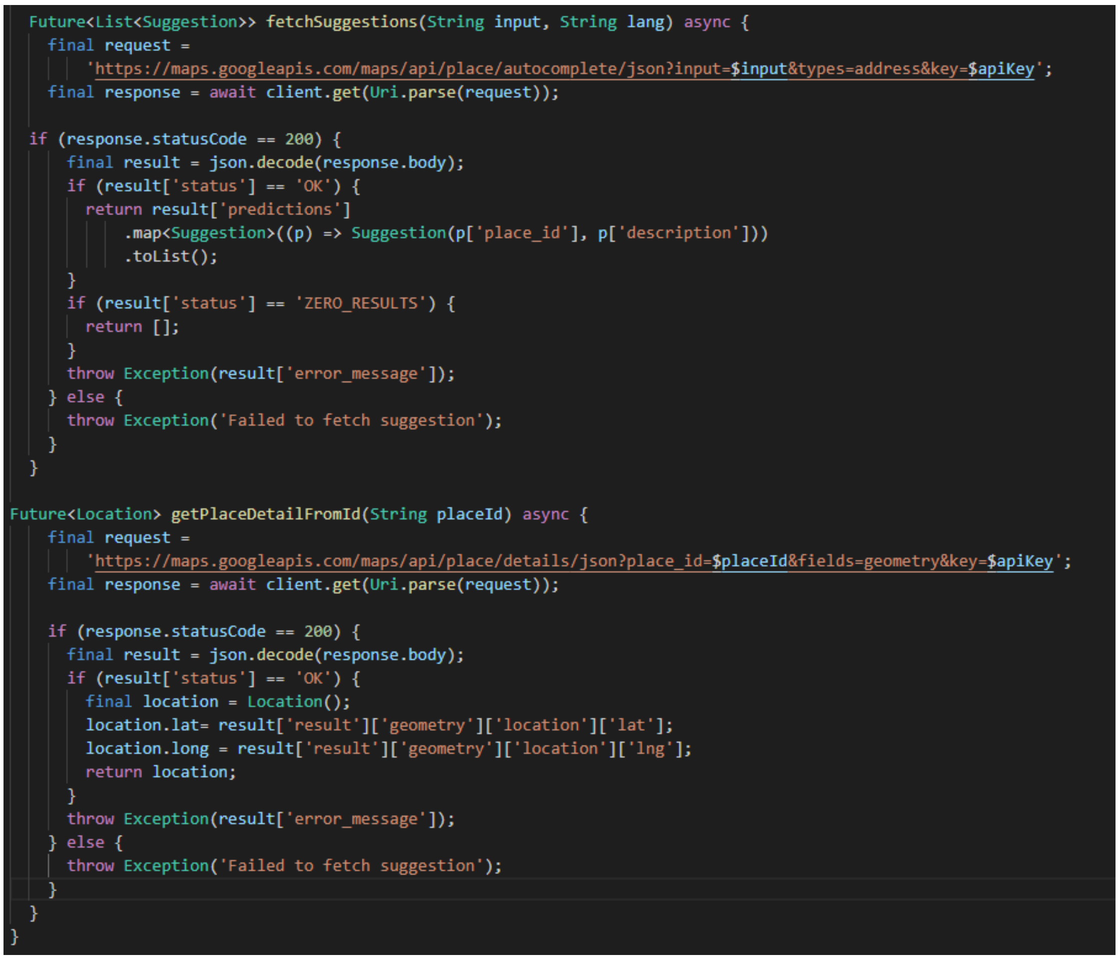
We have used Cloud Firebase Storage to store and serve user-generated content, such as photos of the profile or of the rental ads.

2.3 Google Maps API

Considering that the visualization on the map of the geographic locations is user friendly and represents the best way to immediately identify the houses available for rent, we have opted for the Google Maps API which is a robust tool to add a custom map using Google Maps data, map displays and map gesture responses. Moreover it allows to add information on the map with customized markers.

2.4 Google Places API

Google Places API helps users to find points of interest and allows us to use also the Place Autocomplete service, a web service that returns place predictions in response to an HTTP request. The request specifies a textual search string and optional geographic bounds. The service can be used to provide autocomplete functionality for text-based geographic searches. Google place APIs are not as expensive as other APIs and help users when they insert address to add a new house into the system and help us from retrieving coordinates.



```
  Future<List<Suggestion>> fetchSuggestions(String input, String lang) async {
    final request =
        'https://maps.googleapis.com/maps/api/place/autocomplete/json?input=$input&types=address&key=$apiKey';
    final response = await client.get(Uri.parse(request));

  if (response.statusCode == 200) {
      final result = json.decode(response.body);
      if (result['status'] == 'OK') {
        return result['predictions']
            .map<Suggestion>((p) => Suggestion(p['place_id'], p['description']))
            .toList();
      }
      if (result['status'] == 'ZERO_RESULTS') {
        return [];
      }
      throw Exception(result['error_message']);
    } else {
      throw Exception('Failed to fetch suggestion');
    }
  }

Future<Location> getPlaceDetailFromId(String placeId) async {
    final request =
        'https://maps.googleapis.com/maps/api/place/details/json?place_id=$placeId&fields=geometry&key=$apiKey';
    final response = await client.get(Uri.parse(request));

    if (response.statusCode == 200) {
      final result = json.decode(response.body);
      if (result['status'] == 'OK') {
        final location = Location();
        location.lat= result['result']['geometry']['location']['lat'];
        location.long = result['result']['geometry']['location']['lng'];
        return location;
      }
      throw Exception(result['error_message']);
    } else {
      throw Exception('Failed to fetch suggestion');
    }
  }
}
```

Figure 2.2: Place Autocomplete requests via Google Places API

3 User Interface Design

In this section are present some screenshots by ***Flats*** mobile application. The application is developed to adapt itself to smaller screen (smartphone) and larger ones (tablet).

The user can navigate inside our app via a *BottomNavigationBar*.

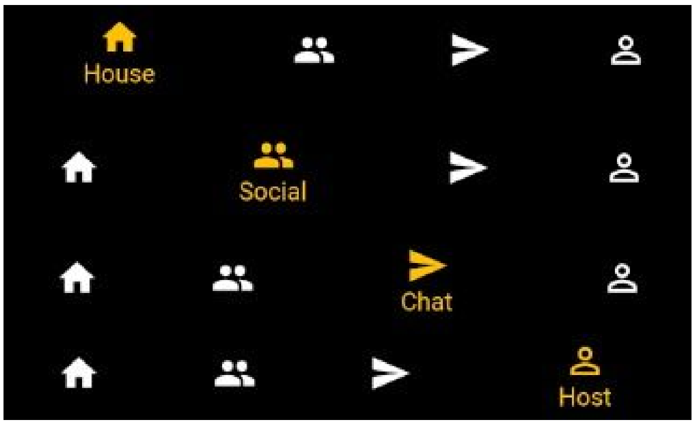


Figure 3.1: BottomnavigationBar

3.1 Splash

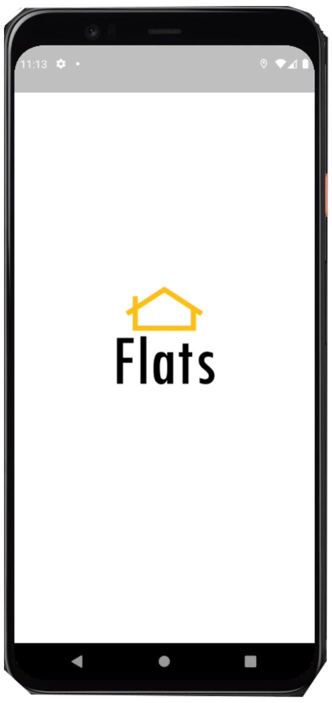


Figure 3.2: Slash Screen

The Splash Screen welcomes the user when the application is starting; meanwhile it restores the state of the application or loads the default one.

3.2 Login/SignUp

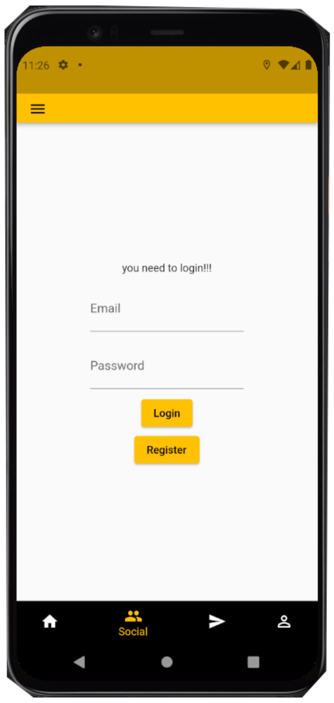
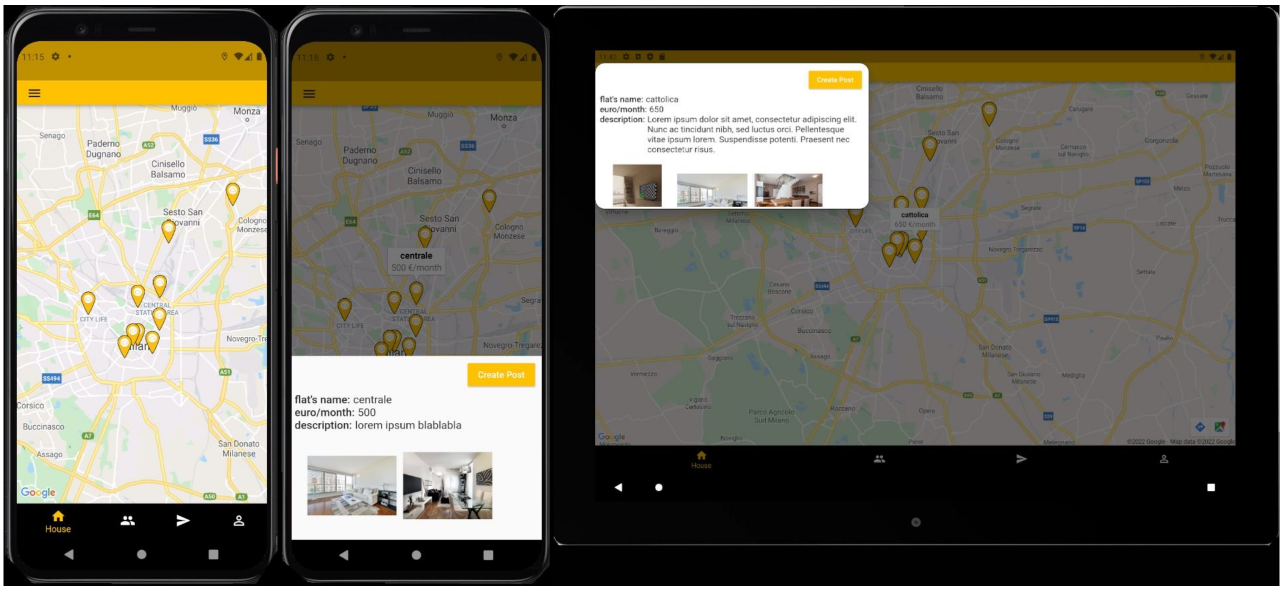


Figure 3.3: Login Screen

The login screen is composed by a form in which the users inserts the credentials to log into the system. If users are not yet registered, they can sign up.

3.3 House Screen



flat's name: centrale
euro/month: 500
description: lorem ipsum blablabla

flat's name: cattolica
euro/month: 650
description: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc ac tincidunt nibh, sed luctus orci. Pellentesque vitae ipsum lorem. Suspendisse potenti. Praesent nec consectetur risus.

Figure 3.4: House Screen and apartment details on smartphone and tablet

The House Screen is the first screen that users see once the application is loaded. Users can click on any marker to see further information about houses. After clicking, users can see two different screen layouts according to the device they are using.

3.4 Social Screen

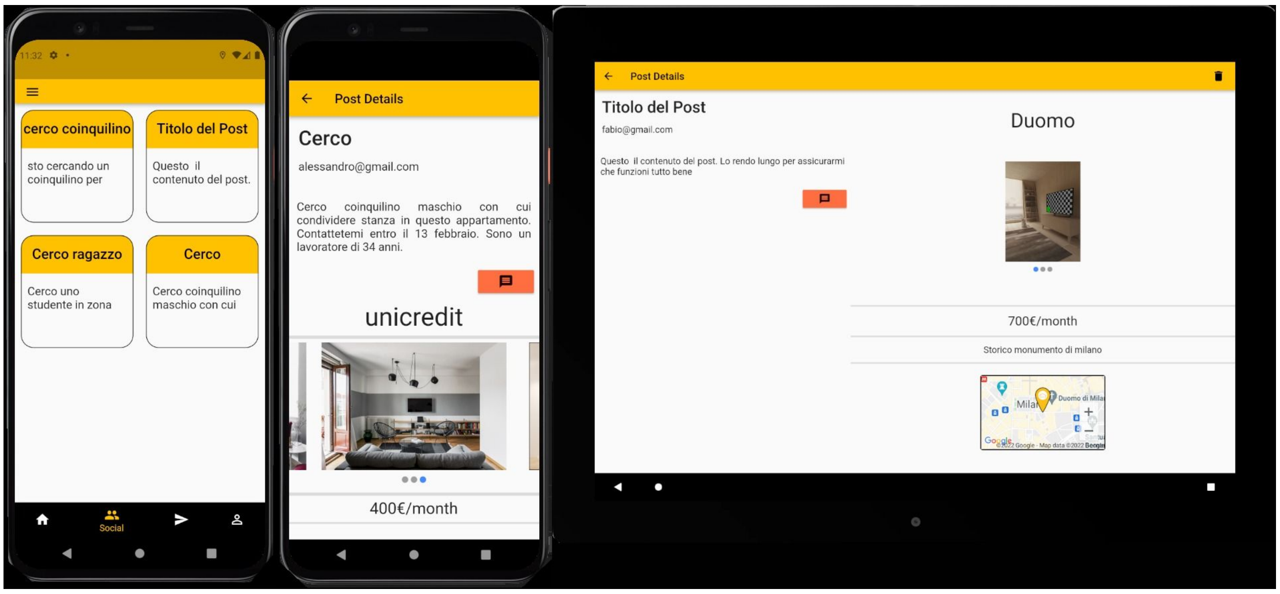
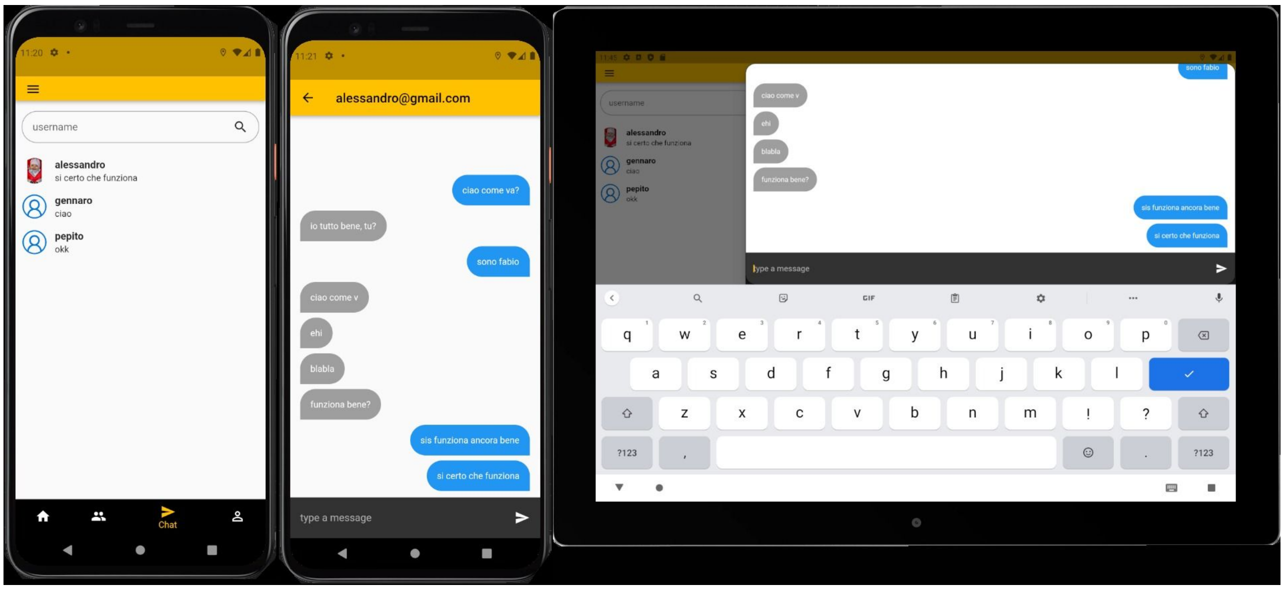

cerco coinquilino
sto cercando un coinquilino per
Titolo del Post
Questo il contenuto del post.
Cerco ragazzo
Cerco uno studente in zona
Cerco
Cerco coinquilino maschio con cui
Social
Post Details
Cerco
alessandro@gmail.com
Cerco coinquilino maschio con cui condividere stanza in questo appartamento. Contattetemi entro il 13 febbraio. Sono un lavoratore di 34 anni.
unicredit
400€/month
Post Details
Titolo del Post
fabio@gmail.com
Questo il contenuto del post. Lo rendo lungo per assicurarmi che funzioni tutto bene
Duomo
700€/month
Storico monumento di milano

Figure 3.5: Social Screen and post details on smartphone and tablet

The social screen is organized as a bulletin board with different post-it attached to it. If users click on a post they can see further information in different layouts for smartphone and tablet.

3.5 Chat Screen


alessandro
si certo che funziona
gennaro
ciao
pepito
okk

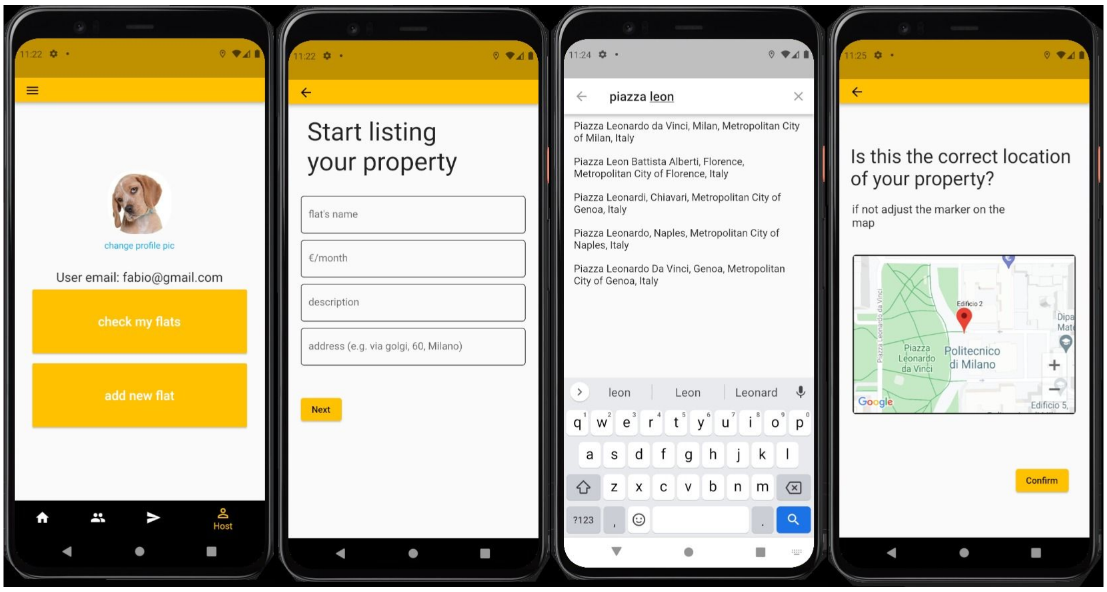
alessandro@gmail.com

ciao come va?
io tutto bene, tu?
sono fabio
ciao come v
ehi
blabla
funziona bene?
sis funziona ancora bene
si certo che funziona

Figure 3.6: Chat Screen and single chat on smartphone and tablet

Chat screen is a list of all chats started by the logged in user. When users click on a chat, the chat is displayed with a different layout according to the device they are using.

3.6 Host Screen



User email: fabio@gmail.com

Start listing your property

flat's name

€/month

description

address (e.g. via golgi, 60, Milano)

piazza leon

Piazza Leonardo da Vinci, Milan, Metropolitan City of Milan, Italy

Piazza Leon Battista Alberti, Florence, Metropolitan City of Florence, Italy

Piazza Leonardi, Chiavari, Metropolitan City of Genoa, Italy

Piazza Leonardo, Naples, Metropolitan City of Naples, Italy

Piazza Leonardo Da Vinci, Genoa, Metropolitan City of Genoa, Italy

Is this the correct location of your property?

if not adjust the marker on the map

Figure 3.7: Host screen, add house and autocomplete screen

The Host screen has two main buttons: the first allows to see all his properties and the second one allows to add a new property. When clicking on "add" button, a new screen appears together with a form in which the users inserts the details. When the host inserts the address, the autocomplete screen appears. At the end of the process, the host will be requested to confirm or adjust the location moving the marker in a small map.

4 Testing

The application shows all houses available for rent on a map through Google Maps API. Because of Google Maps API and Firebase, the possibility to run several unit tests is limited. We consequently targeted our tests aiming to check the correct performance of the forms and to verify if errors messages are shown when needed. In particular, we have tested login and register form and added house form in Host section. Regarding registration phase, we have checked that emails must be in a correct format and password must be longer than 6 characters.

On the other hand, We made a lots of widget tests to verify that the widget's UI looks and interacts as expected.

When testing, we stressed the app by performing several unusual actions by the user and by checking that the interface kept working as intended in order to exclude any graphic glitches.

5 Effort Spent

The development of our app took place between November 2021 and January 2022.

We worked in team on site and remotely to setup the crucial parts of the application but we also worked on our own with continuous mutual contacts.

We didn't use any professional time tracker, but we estimate that the hours spent are around 150 per worker and can be roughly splitted as follows:

- 15% platform meet and greet
- 65% development
- 10% testing
- 10% document drafting