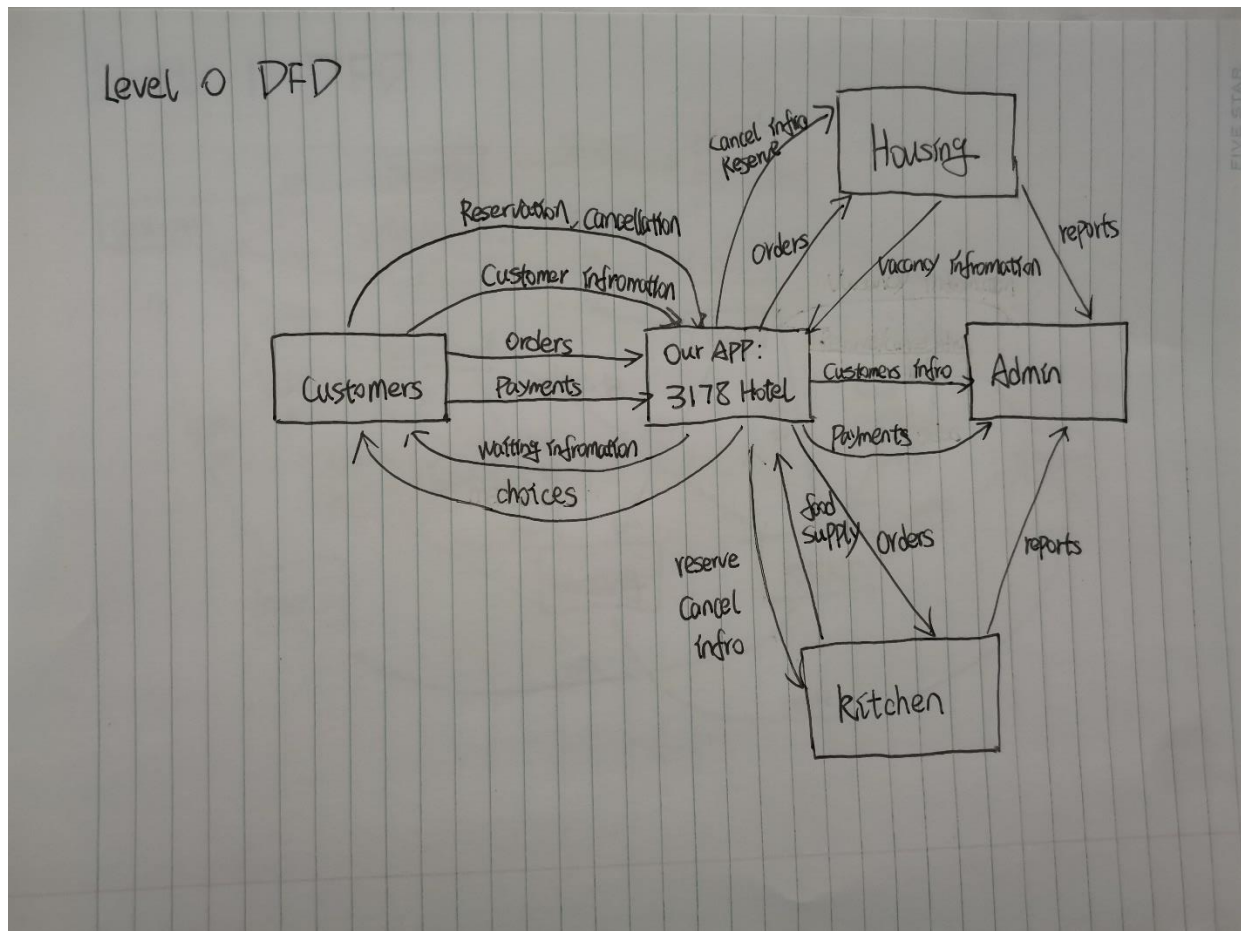
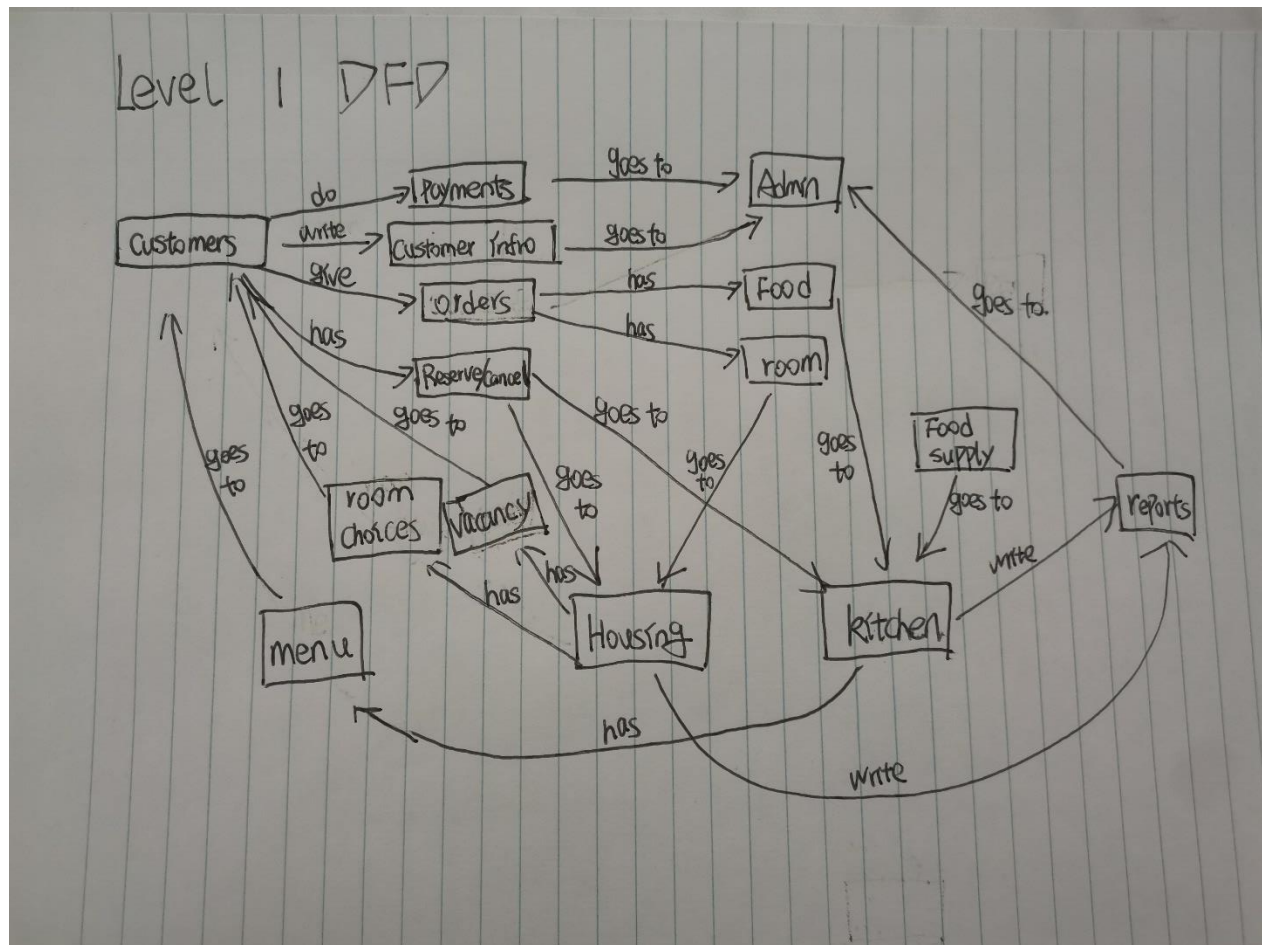


The README file is being edited on the GitHub.

GitHub URL: <https://github.com/COSC310-Assignment-2/Assignment-2-.git>



In level 0 DFD, we can see customers have the power of inputting orders, paying orders, giving app their information, having the right to take the reservation and cancellation, and receiving the waiting information and choices from kitchen and housing. On our app, it has the functions of gathering all the data that customers input, separating them by the functions and giving the data to housing, administration, kitchen. Besides sending the data, our app has the receiving functions from all those three related stake holders. One important thing is that administration has the highest power of getting reports from kitchen and housing, and administration doesn't need to output anything.



In level 1 DFD, we can clearly see all the functions are listed and how they worked from the arrows. This graph is adding on the level 0 graph, so it is a bigger graph to see each individual function and how they acted as the sending and receiving functions.

Since we are not familiar with GitHub (all of us is first time using this), we don't know that each individual can upload their file on the GitHub, we are sending the file to one person and he is uploading it in the end, so we are saying that everyone has approximately equal work, and you can go ahead to check on that one person's final version of our project which he is uploading in the GitHub, and we will show some comments in the project to state who did what.

```

String content = content1.getText().toString();
porterStemmer = new PorterStemmer();
String stem = porterStemmer.stem(content);
rep=new msg(stem, type: 1, da.format(new Date()));

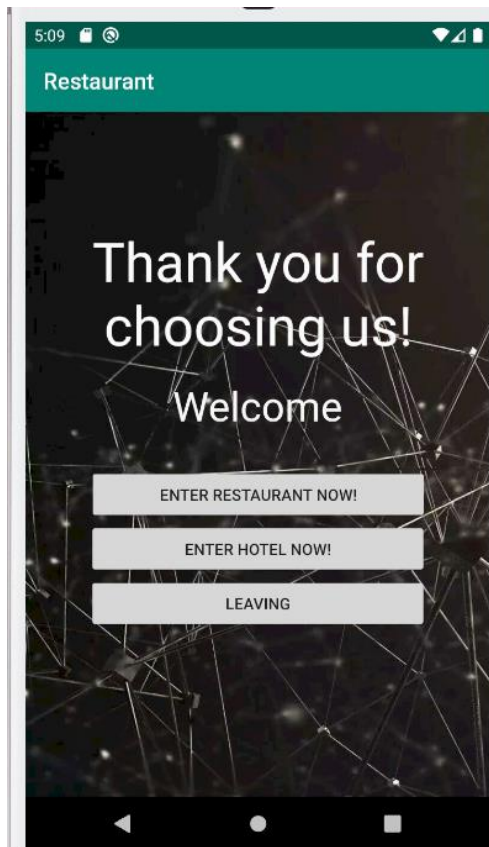
Adap.addItem(rep);
chatlist.scrollToPosition(Adap.getItemCount()-1);
content1.setText("");
new Handler().postDelayed(new Runnable() {
    public void run() {
        chat(rep,Adap);
        chatlist.scrollToPosition(Adap.getItemCount()-1);//execute the task
    }
}, delayMillis: 500);
}

try {

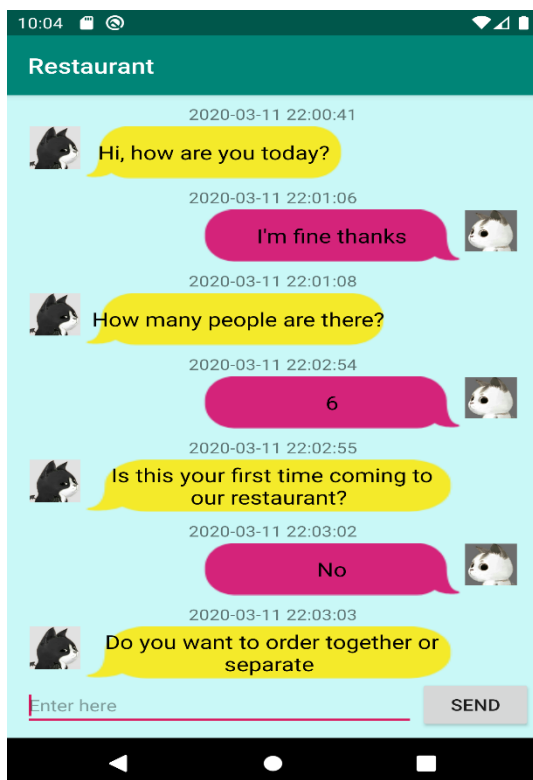
    inputStream = getAssets ( ).open ( fileName: "en-ner-person.bin");
    Log.e( tag: "name", msg: "1");
    TokenNameFinderModel tokenModel =
        new TokenNameFinderModel(inputStream);
    Log.e( tag: "name", msg: "2");
    nameFinderME = new NameFinderME(tokenModel);
    Log.e( tag: "name", msg: "3");
    Span nameSpans[] = nameFinderME.find(respon);
    Log.e( tag: "name",nameSpans.toString());
    myname=nameSpans[0].toString();
    Log.e( tag: "nname",nameSpans.toString());
} catch (IOException e) {
    e.printStackTrace();
}

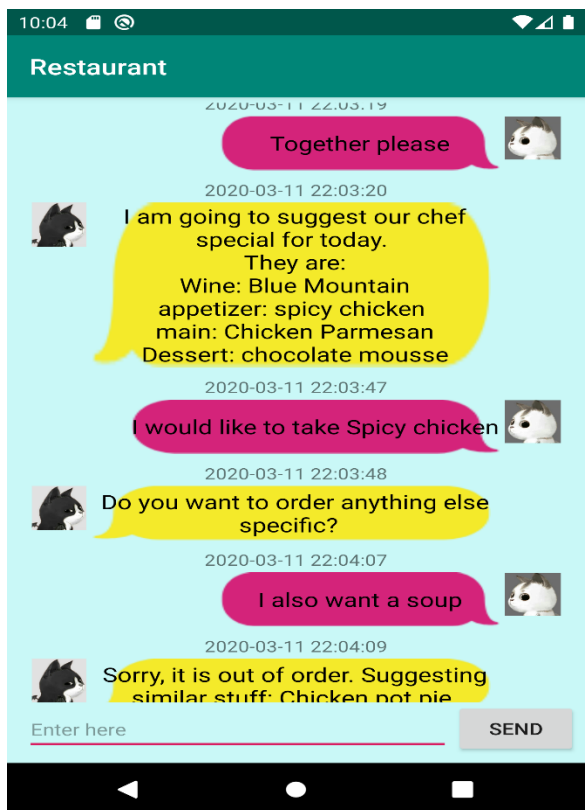
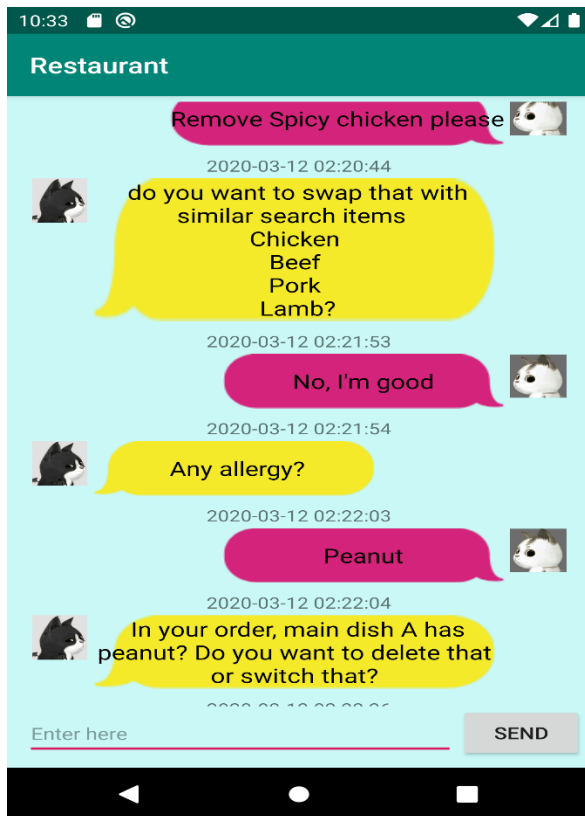
adap.addItem(new msg( content: "Ok "+myname+"! How many people are there?" , type: 0, da.format(new Date())));

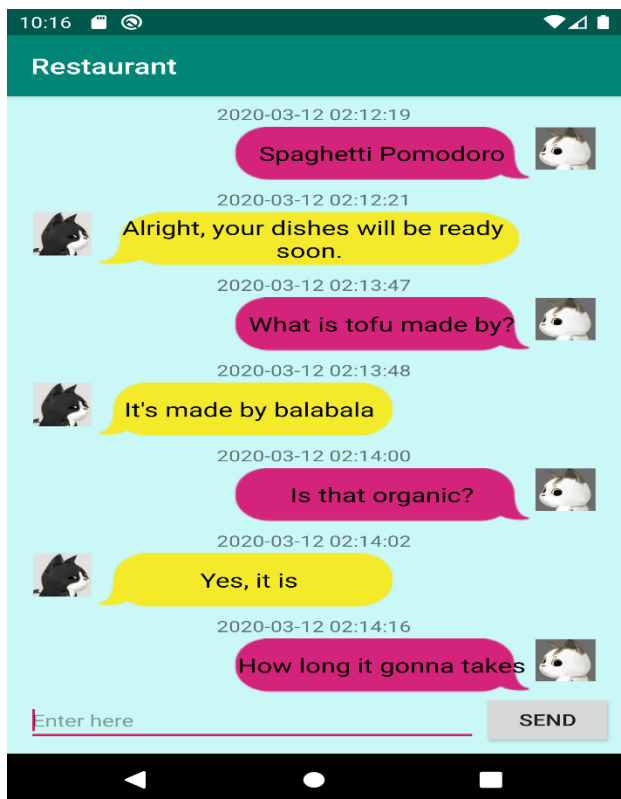
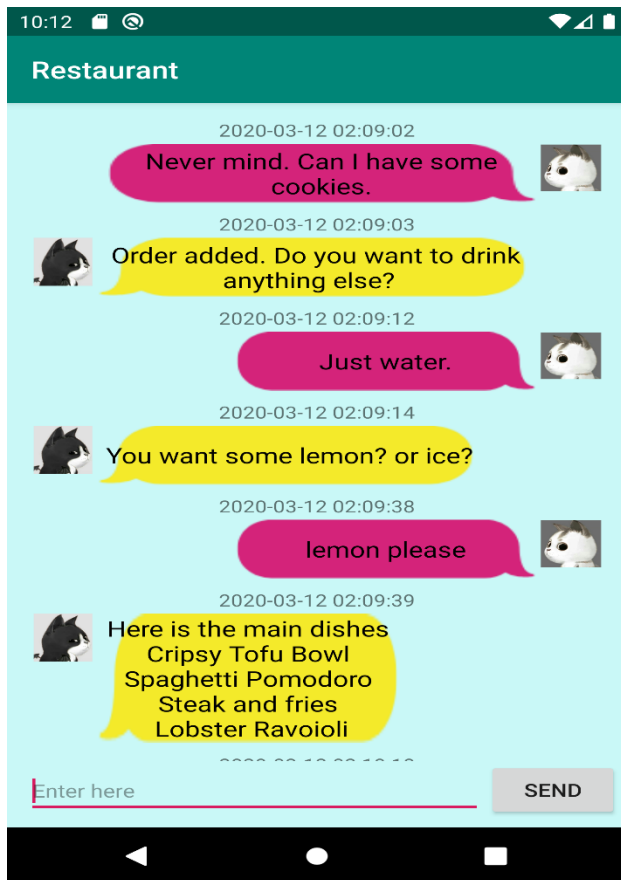
```

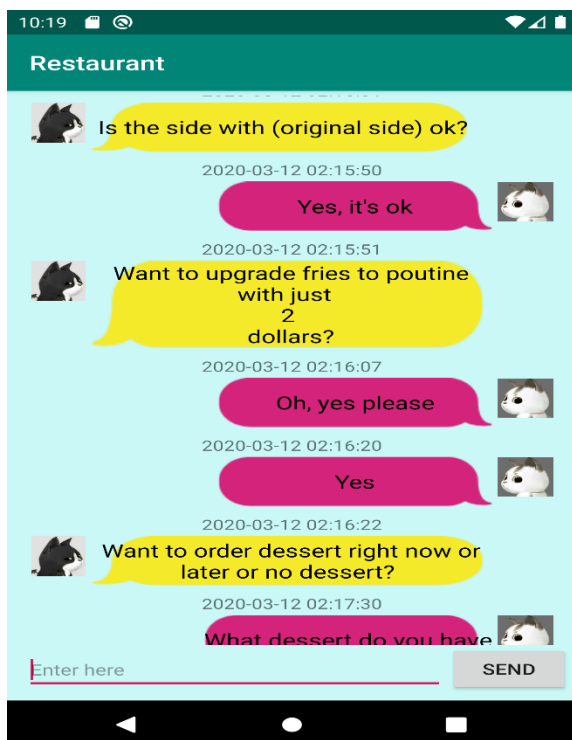
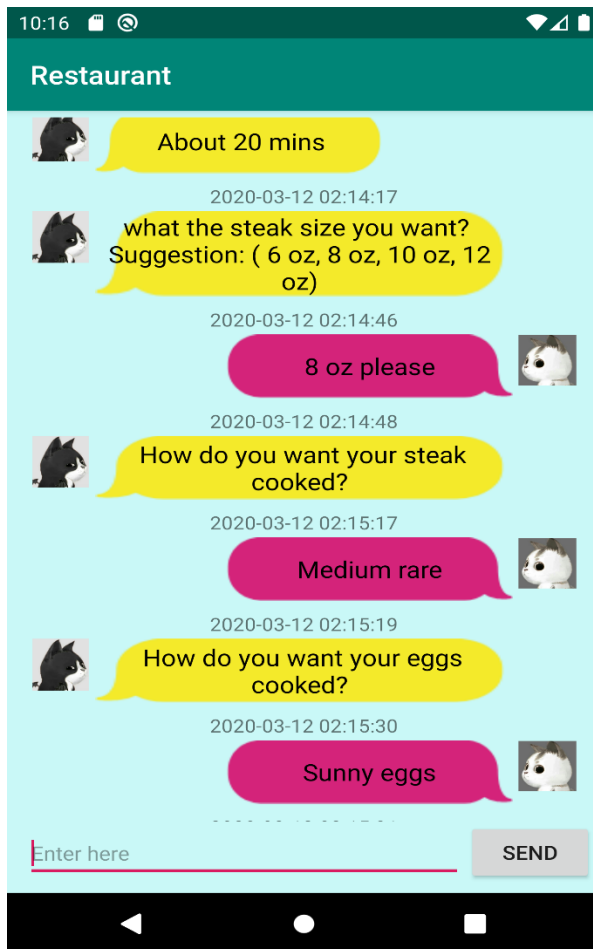


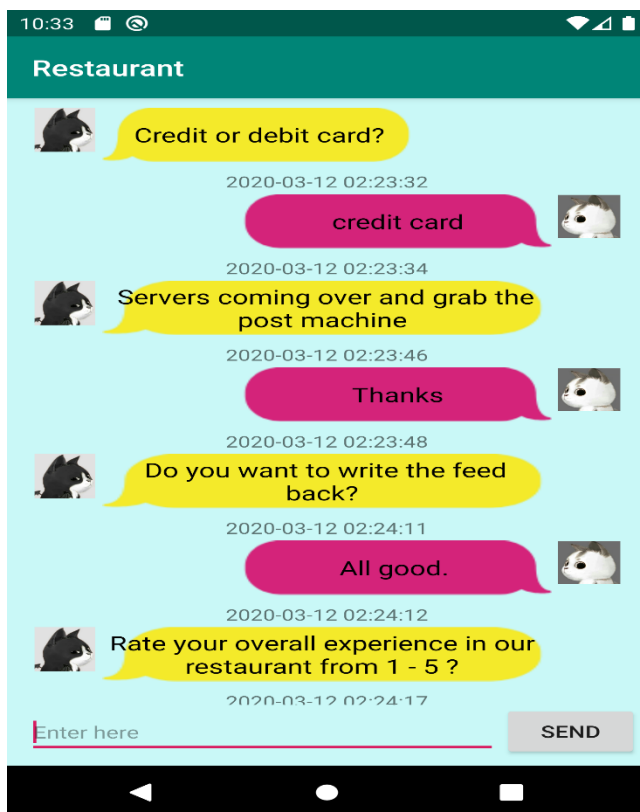
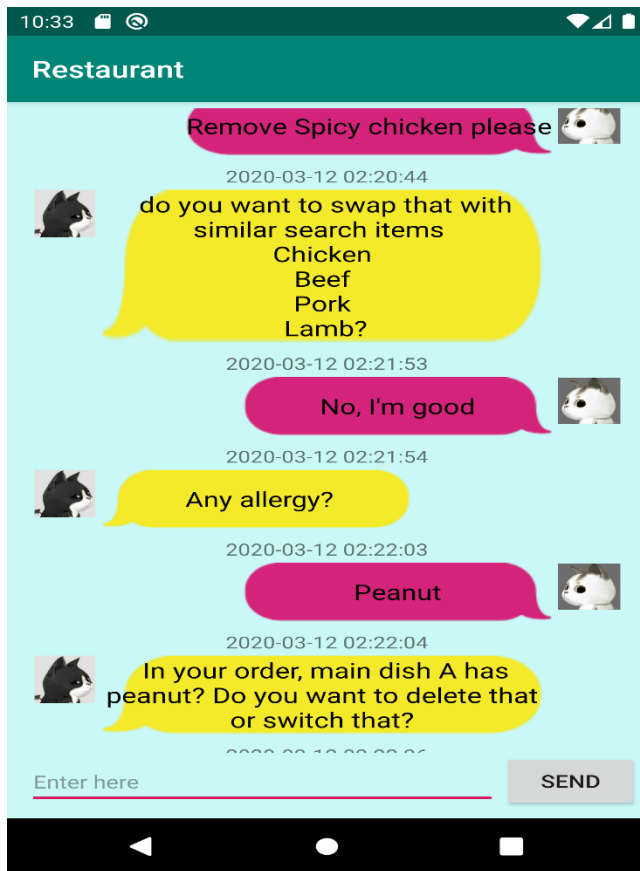
Restaurant first testing:



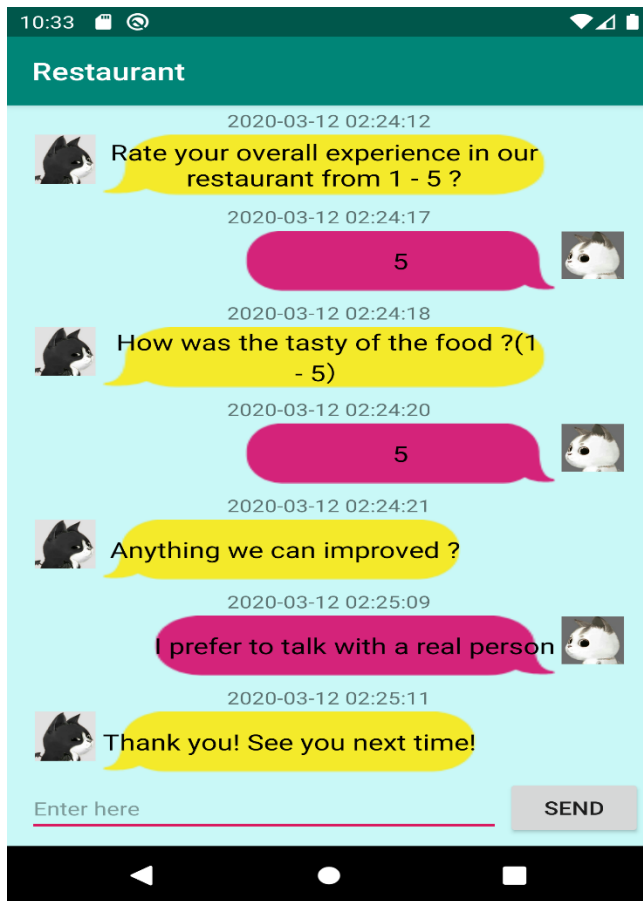




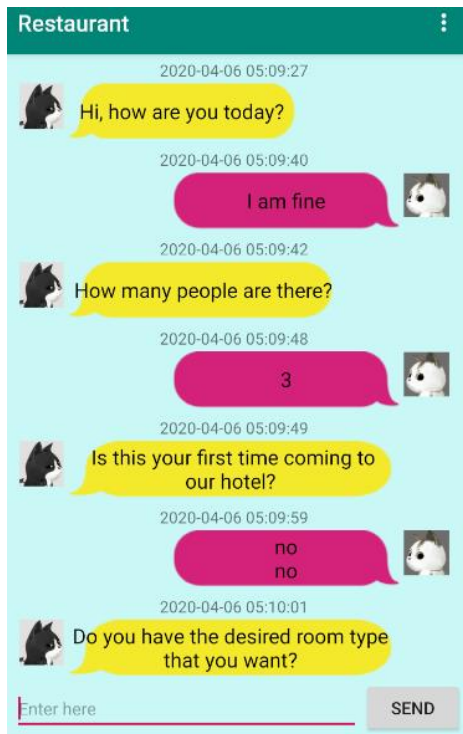


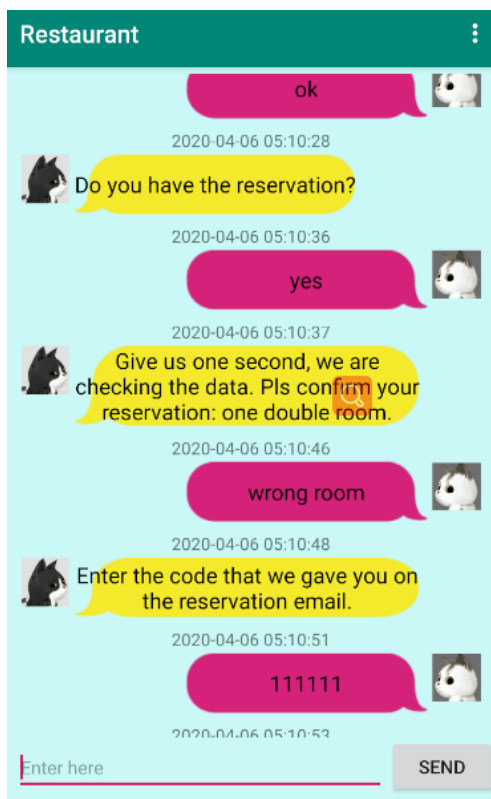


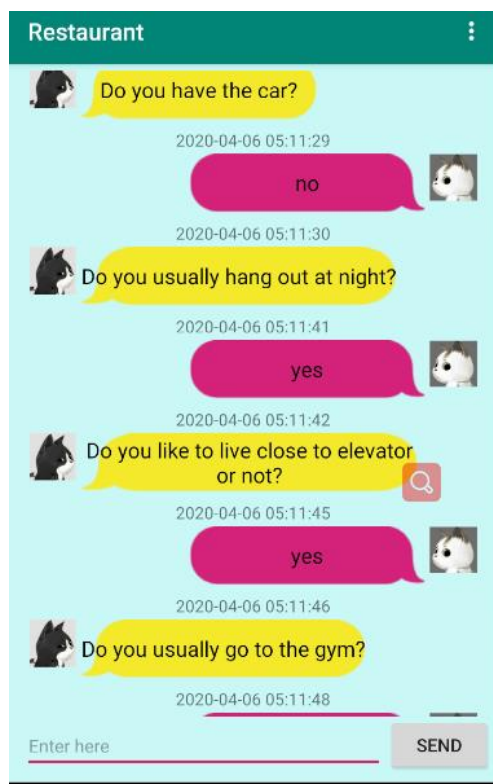
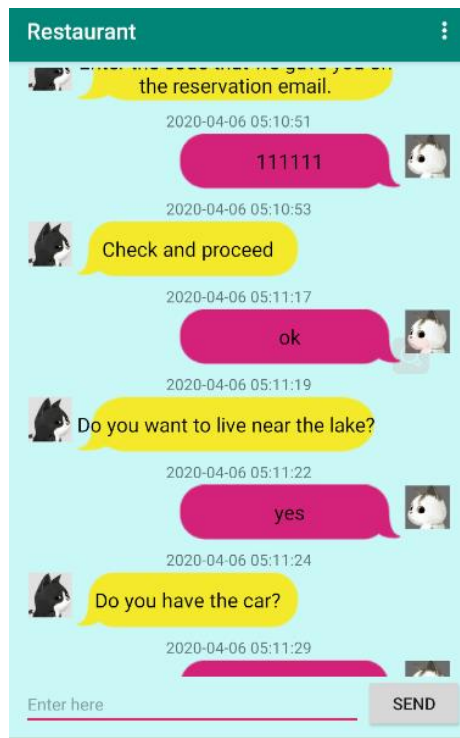


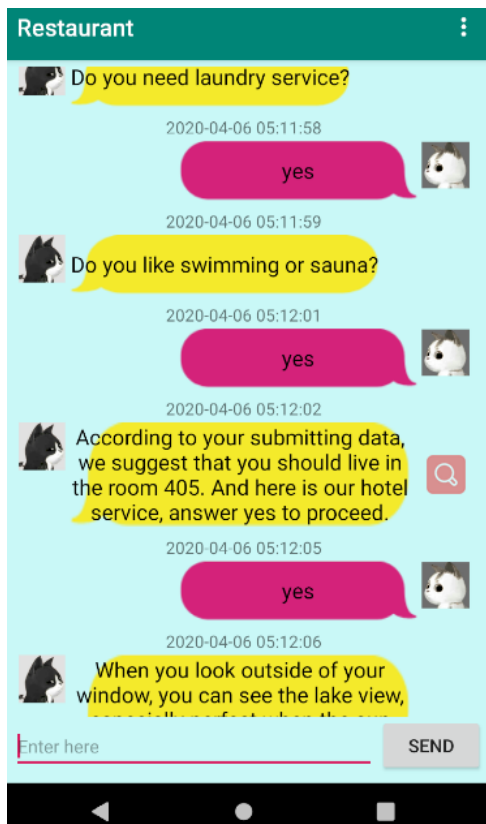
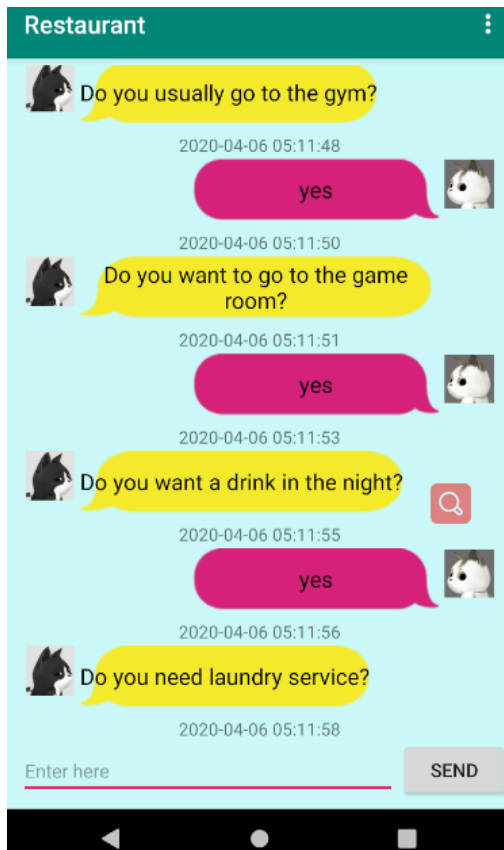


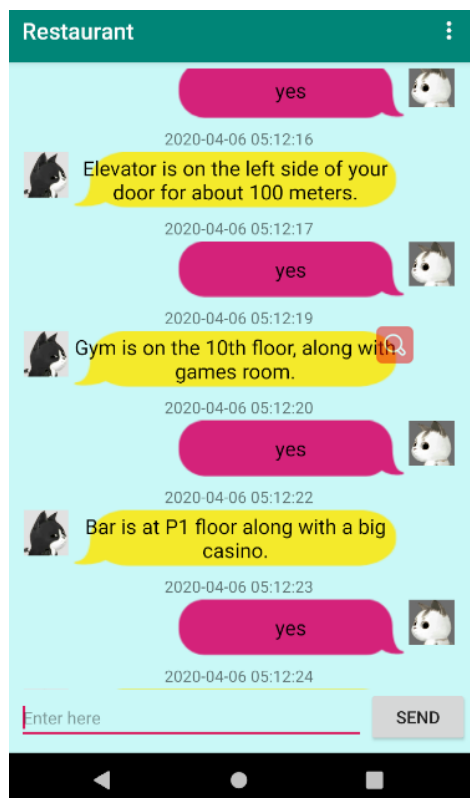
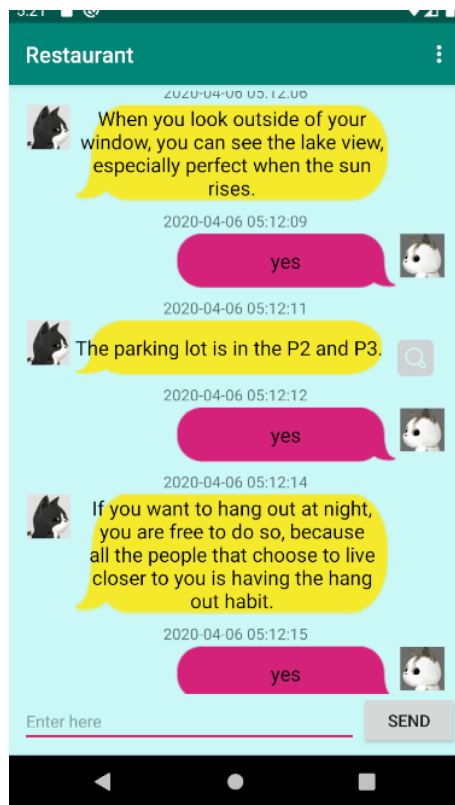
Next is the hotel testing

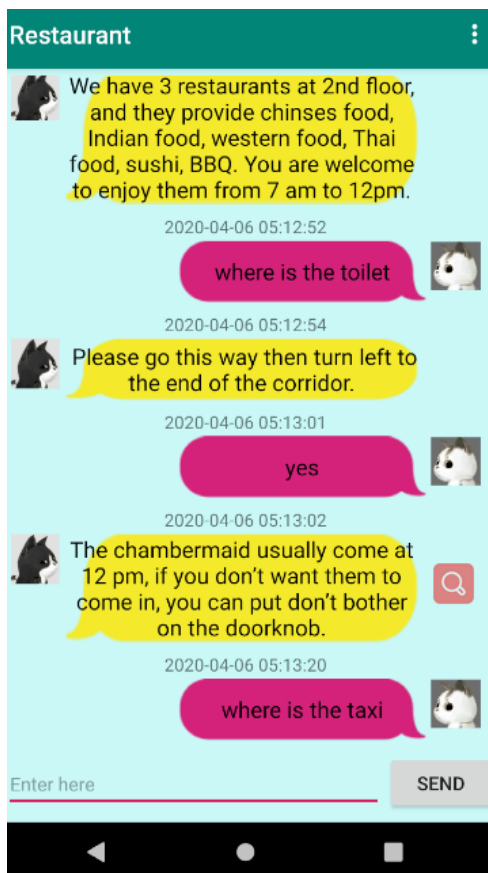
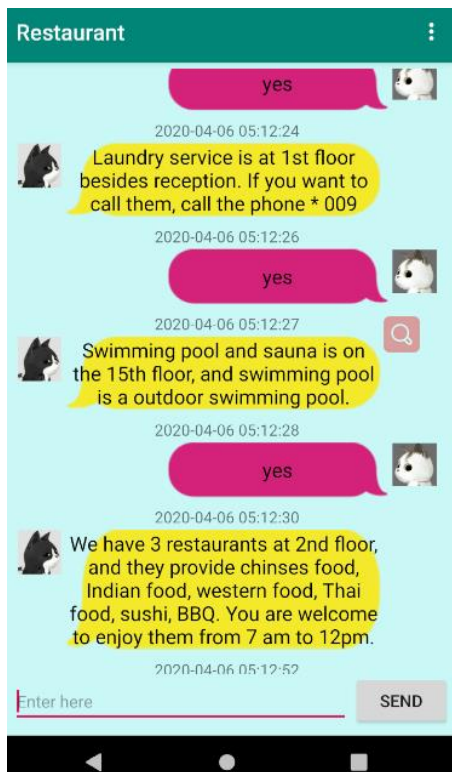


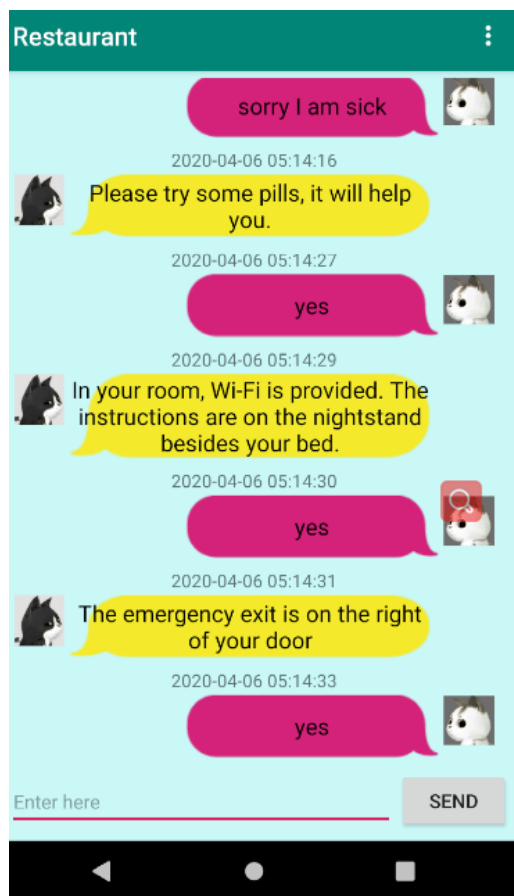
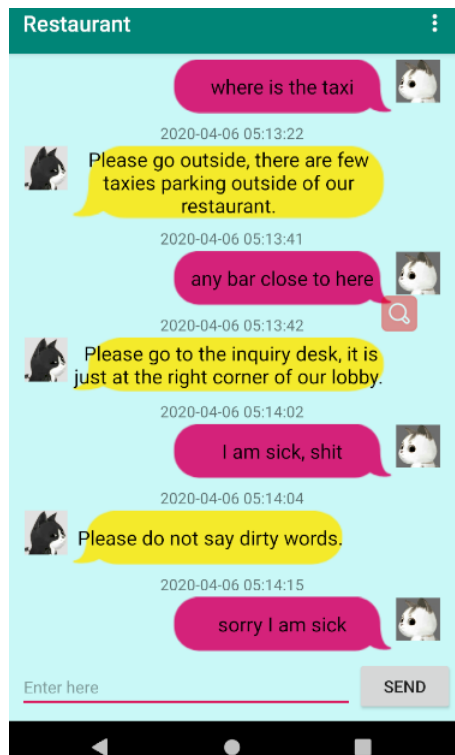


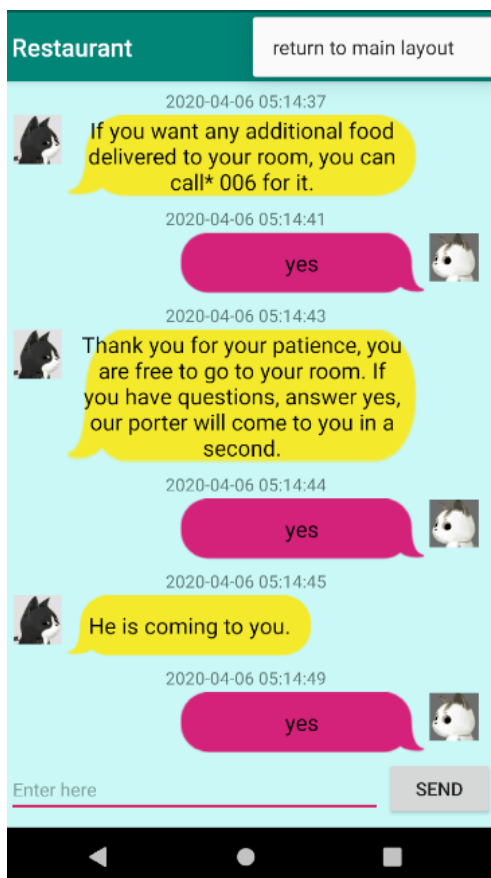
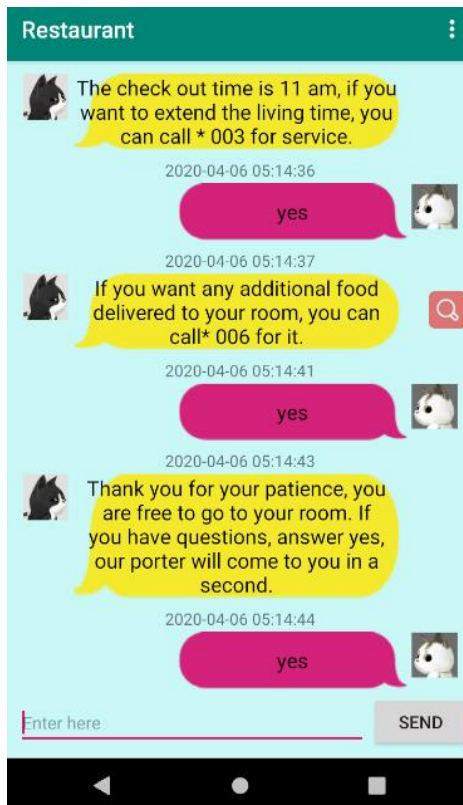






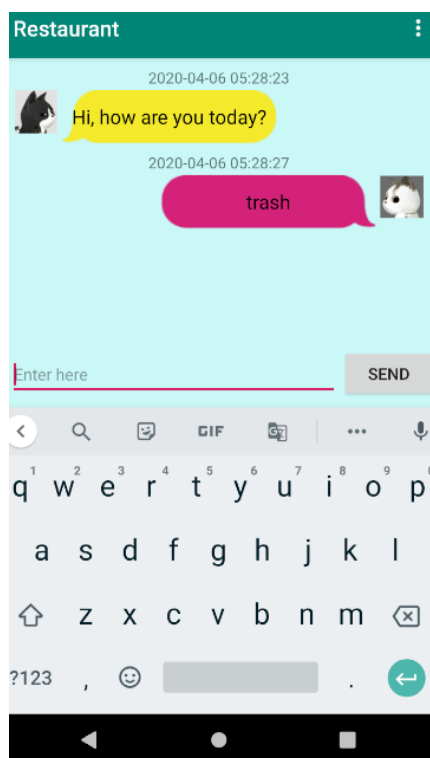
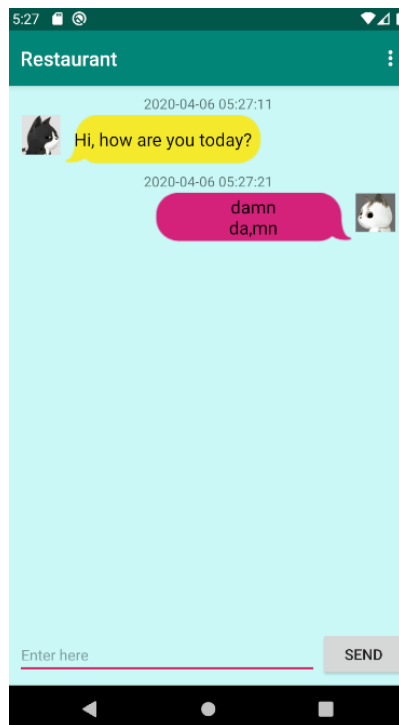




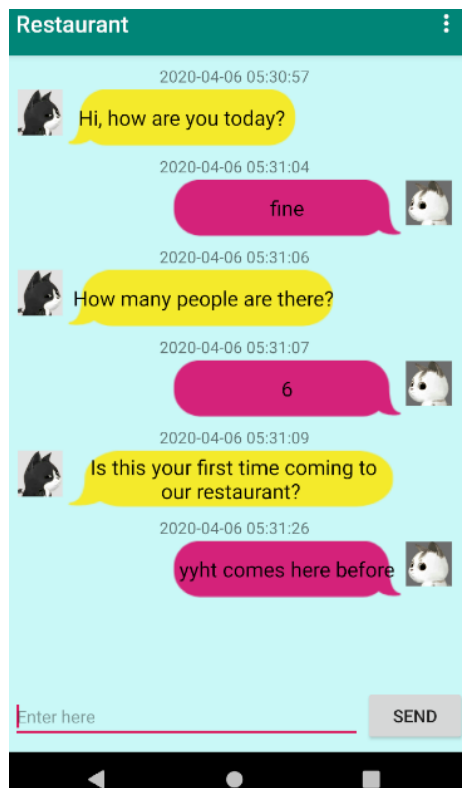
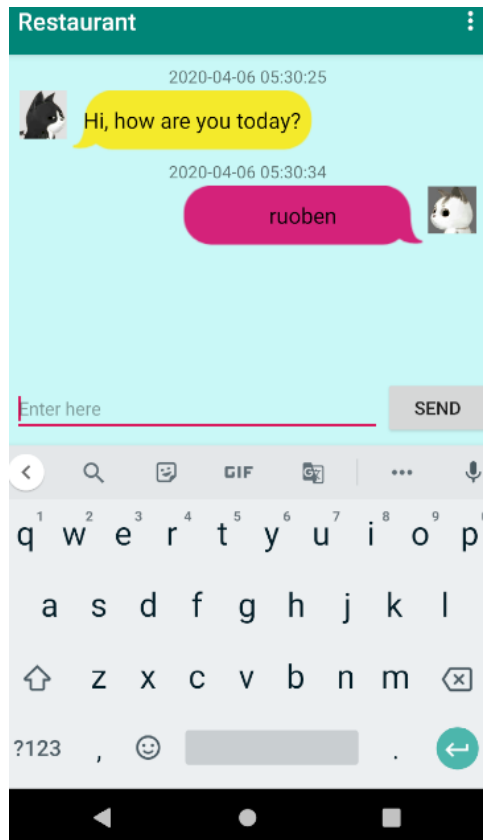




Limitation: we can't handle some dirty words since we only added a little bit of dirty word in the recognition software.



We can't catch some weird English names.



Lists of useful features that we can use in the future:

- 1 Message class which contains three attributes of the message object which can help us define the message is from user or agent
- 2 Override adaptors to fit the chat window.
- 3 Create our own view holders, one for agent, and another one for user.
- 4 An xml layout named right chat and left chat can represent the message object.
- 5 the real time above the message.