UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

# Project Partyholic

Real-Time Mobile NLP Analysis WS20/21
Natural Language Processing

Abdulsatar Sayed (3072353)
Ahmad Hussien (3085179)
Ashraf Hashash (3063034)

Get access to the repository of Partyholic on Github:

https://github.com/smmohash/Partyholic-0.1.0

## Contents

## Intro

Partyholic is an Android app that allows the user to capture a photo with their camera or choose one from their gallery, with the photo being of a poster of a party or a festival. Then, Partyholic extracts the time, date, and location of the event from the poster and uses it to search on Google Maps and show the destination directly on a Google Maps windows inside the app screen and zoom in onto it, while leaving the user the option to go back and forth between their current location and the event location (the destination). It also calculates the time remaining until the event and shows it on top of the screen. This means, the user has just one job, which is to take a photo or choose one from their gallery. There is no need to type any text with your fingers. Instead, you, as someone who is wandering the streets looking for the next party, can enjoy your time with less disruption.

## Assumptions

We assume that the poster contains at least:

- the address of the event, which we assume, consists of (in the following order):
  - the name of a street
  - the number of a building
  - a ZIP code (German: PLZ) which is a five-digit number (in Germany)
  - name of the town/city/village/etc.…
- the time of the event.
- the date of the event.

Also, we assume that the operating system of the phone is Android and that the phone has a camera with a reasonable image capturing resolution and a working internet connection.

An important thing is, that the user will have to give permission to Partyholic to use their GPS and camera sensors and their internet connection and access the device storage as well.

In order for Google Maps service to work, the user will also have to be signed in into their Google account.

UNIVERSITÄT DUISBURG ESSEN

*Offen im Denken*

## Structure

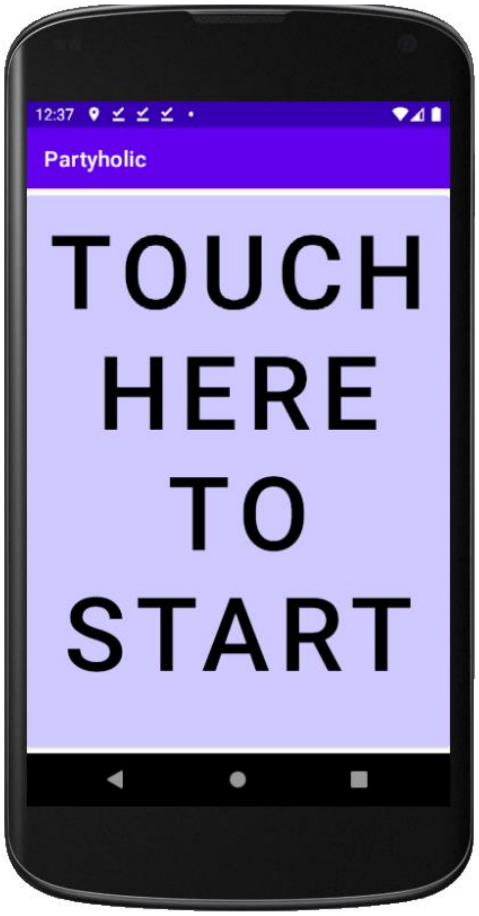
Since our app involve image processing and optical character recognition (OCR) it is more efficient to do it with server-client architecture because the mobile phone processor is not as powerful as a computer processor.

Partyholic is two parts:

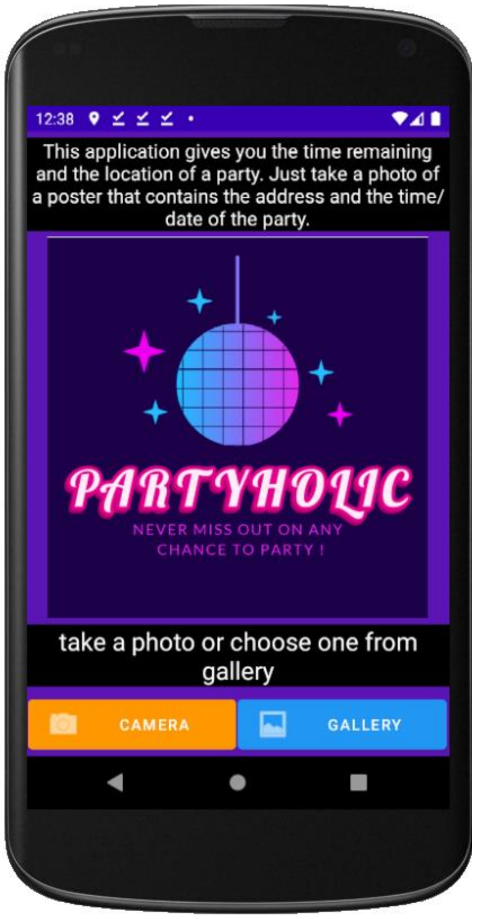
- frontend: the part which the user accesses directly by installing the .apk file on their device.
- backend: the part which the software installed on the device accesses via internet and is where the image processing happens, which in turn communicates with APIs.

The Android app has 6 activities (screens):

Initial screen that asks the user for the previously mentioned permissions.



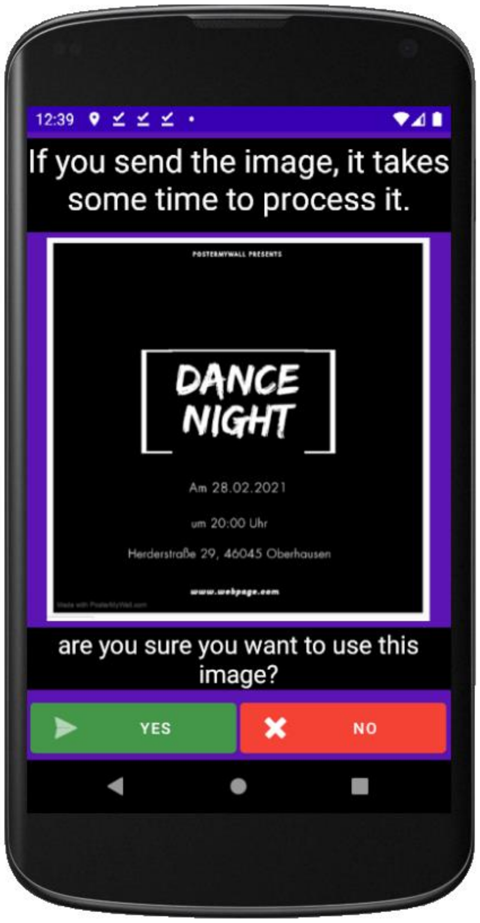
Main screen with Partyholic logo and description and a dialog and two buttons: Camera and Gallery.


12:38

This application gives you the time remaining and the location of a party. Just take a photo of a poster that contains the address and the time/ date of the party.

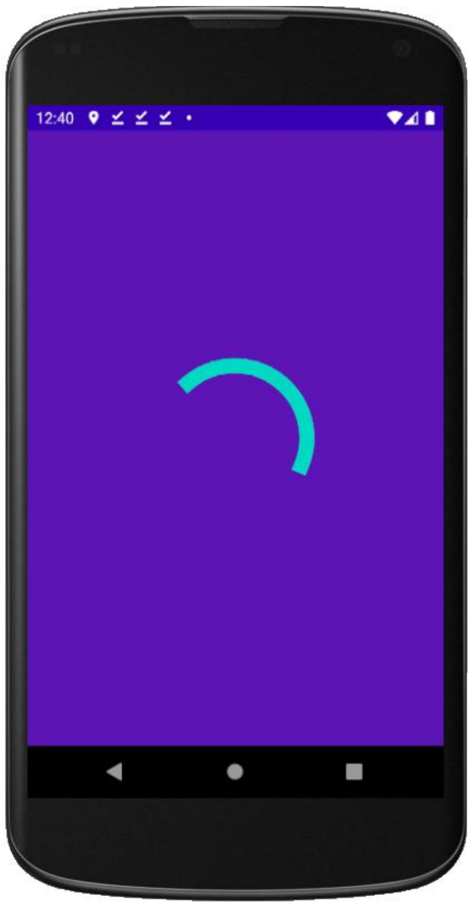
take a photo or choose one from gallery

Confirmation screen which views the selected/captured image one last time and asks the user to confirm they want to use it or not.


12:39

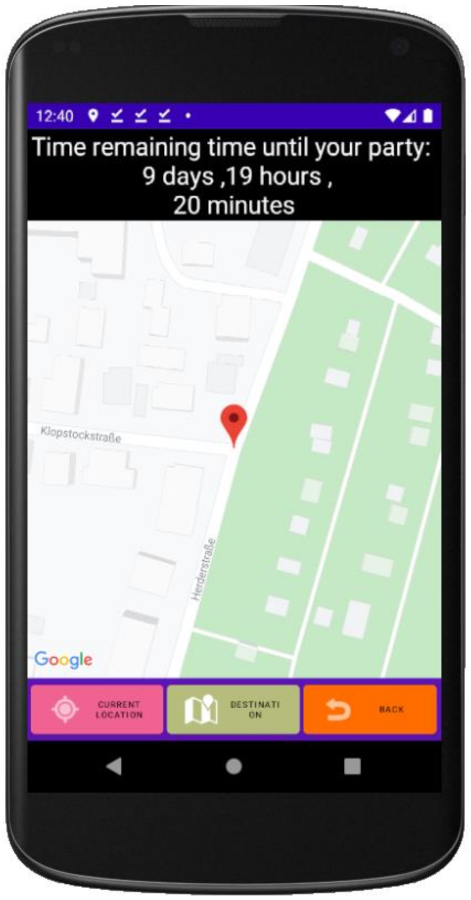
If you send the image, it takes some time to process it.

are you sure you want to use this image?



Loading screen which shows up once the user decides to use the image, since that is the most critical part and since it takes time the most. The loading screen serves two purposes:

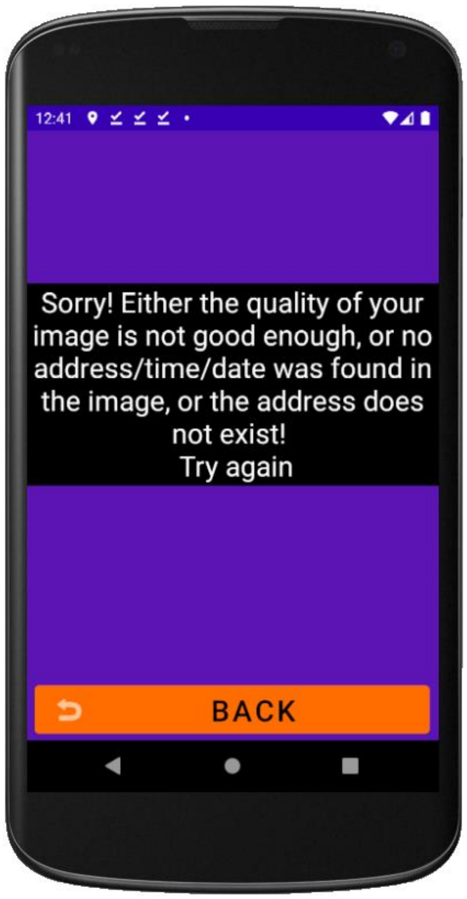
- Indicates to the user that they have to wait a few seconds.
- Prevents the user from issuing more orders while an order is already being processed.



Time remaining time until your party:
9 days ,19 hours ,
20 minutes

Result screen which has a panel with a text being the time remaining until the event, or in case of failure to find time/date, an apology message, and a Google Maps window that shows the destination and the current location of the user.

Apology screen which shows up in case the address of the event could not be recognized due to the low quality of the image, or it not containing an address at all, or because the address couldn't be found (for example, because it does not exist in the real word).



Sorry! Either the quality of your image is not good enough, or no address/time/date was found in the image, or the address does not exist!
Try again

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

## Techniques and protocols used

### Programming languages:

- frontend : Java
- backend: Python

### Frameworks:

- Heroku: cloud platform for hosting the backend part.
- Android Studio: for the frontend
- Flask: for the backend

We chose the Python-based framework (Flask) to build our backend since it is easy to deploy on Heroku.

### External APIs:

- Google Maps: it is used inside Partyholic in a Fragment.
- Nominatim (OpenStreetMap): a free geocoding API finds coordinates of addresses.
- ocr.space: optical character recognition for extracting the text from the image.

First, we tried using the google-api-geocoder, but it is a paid service. So, we switch to Nominatim. It does not require an API-KEY to use it.

### View & Fragment:

These are the general terms for the objects that the layout of the Android app consists of.

"View" is basically like "node". Let's take the main screen for example. The main screen is basically a linear layout (horizontal) that contains:

- a textview which has the welcoming text for Partyholic.
- a space which is a separated .xml file included in the .xml file of the main
- an imageview which has the Partyholic logo.
- another inclusion of space
- another textview
- another inclusion of space
- a linear layout (vertical) that contains:
  - another inclusion of space (vertical)
  - camera button
  - another inclusion of space
  - gallery button
  - another inclusion of space

The Google Maps window is a Fragment. It is basically a layout file (.xml) with its corresponding .java class. It gets "inflated" into the result screen every time this screen is opened.

## Intents:

Intent is a Java class by Google. It allows developers to access a wide range of the Android device functionalities. With Intents we achieved:

- opening the camera and retrieving the captured image
- opening the gallery and retrieving the chosen image
- saving the retrieved image into the directory designated to Partyholic.

## Threading:

In order to implement the loading screen so that it shows up right after the app sends an image to the server and then disappear right after the app finishes dealing with the server and gets the results and is ready to show them, we had to extend the class Task in Java.

## Regular expressions:

The backend program uses **rule-based NLP**. We did the job using regular expression. That is because of two reasons:

- That gets the job done, since addresses in Germany have this somewhat similar structure. The pattern goes as follows : street name > building number > postal code -> city name.
  The name of street also, usually, ends with something like : "str", "straße", "weg", etc. We took advantage of that.
- Our knowledge of the other possible methods to handle the issue (maybe using machine learning) is still limited for the time being. We are still at doing the course by Prof. Dr.-Ing. Torsten Zesch, Sprachtechnologie (at the department : Language Technology Lab) and intending to do the exam this study term.

We define some patterns to detect the time of the event, which could be as follows:

(um 20:00 uhr ) or (um 18 uhr)

as well as for the date, for example :

(am 12.04.2021) or (am 12-januar-2021)

and of course, for detecting the addresses too.

So, we just tried to catch the most address patterns and it seems that the word "Straße" appeared on the most street name's in Germany.

## HTTPS:

The communication between the frontend and the backend was achieved with Https. The frontend is sent in binary decoding, writing it into an output stream. Also, when the backend responds, it sends the data in a .json file.

## OCR:

The Process of identifying the text in an image is called optical character recognition (OCR)

There is an open-source engine to apply OCR on an image called Tesseract. First, we tried to use it.

But it seems that this engine is optimized to be applied on scanned book documents and it gives, in that case,  very accurate and reliable results.

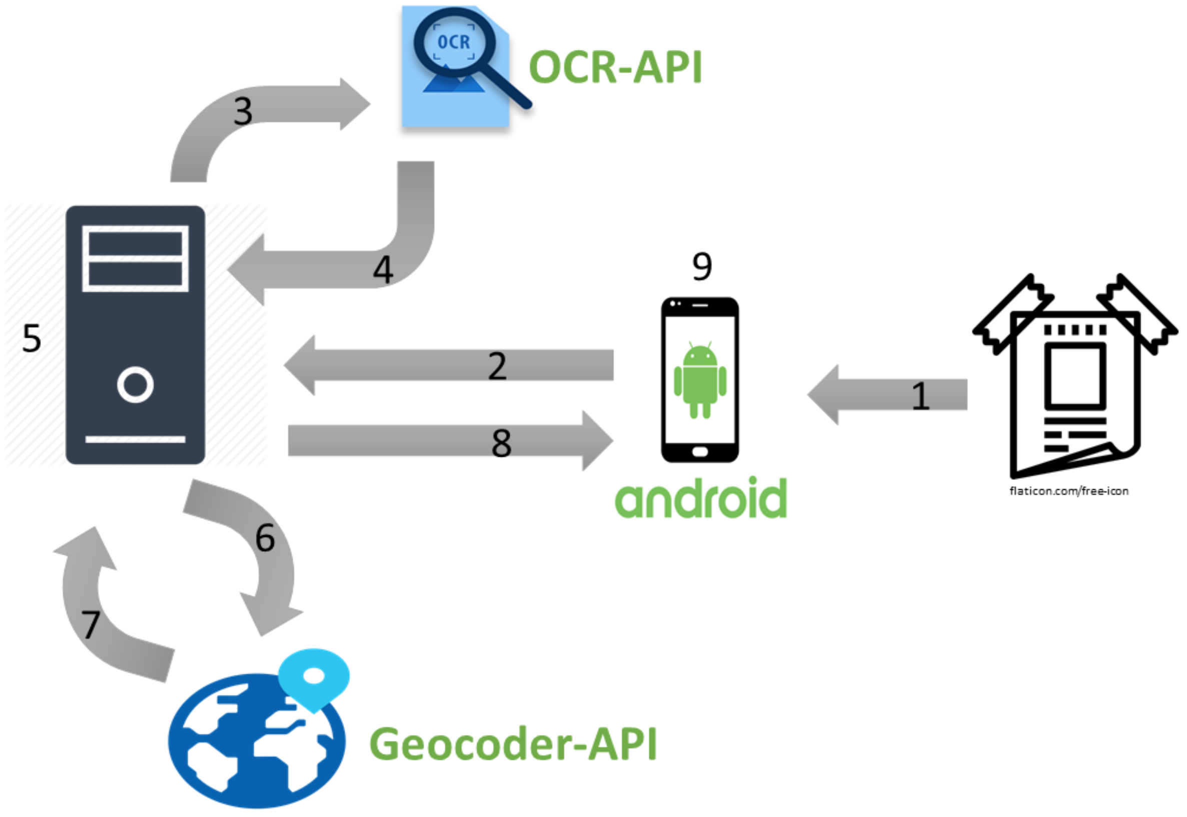
But in our project, the data we need to process, are colored images. And after using Tesseract on some test cases, we did not get accurate results even after many experiments with reprocessing as filtering , re-scaling , blurring, and  image thresholding.

Because of that we chose to do this step by a free, third party API : ocr.space. They offer a free API-KEY with some limitations. One of these limitations is that the file size should be less than 1 MB. And therefore, we compress the image so that it will be always within the allowed size.

UNIVERSITÄT DUISBURG ESSEN

*Offen im Denken*

## How Partyholic works (in chronical order):



(1) capturing an image (or choosing from gallery):
The user takes an image or choses one from gallery and click the “yes” button.

(2) sending the image to the backend:
The image is sent to the server as a stream of bytes whereas the server will receive this stream of bytes and write it in the home directory and give this image a unique name to distinguish it from other requests since the server can receive more than one request in the same time from different users .

(3) applying the OCR on the received image:
The image is sent to the OCR API to get it processed.

(4) getting the textual content of the image.

(5) extracting the relevant information from the text.

(6) sending the address to the geo coding API.

(7) getting the coordinates corresponding to the address.

(8) sending the response to the frontend:
In case of failure of identification, “Unknown” is sent to the frontend. When the image is not needed anymore after sending the response to the app, it gets deleted so that the home directory is not filled with unnecessary files and the storage will not run out of capacity.

(9) showing the result to the user.

## Conclusion

Now we have vision of how to build an application in all its stages, starting from discussing the idea of the project and the problems that we may face us within a working group and how to find solutions and search in different places sites. We learned how to design the graphic user interface, build the backend, and link the user interface to the appropriate API, in addition to testing the program to find out and troubleshoot problems.