Technical Project Description

Tech Challenge 2020 – Team DataMo

1. Hardware components:

a. NFC-tags:

For our prototype we used free programmable NFC-tags which can easily be purchased in electrical stores or over the internet for less than two euros each. As storage capacity 50 bytes are sufficient. We decided to use NFC-tags like key rings because they are weatherproof and can easily be mounted to any device using cable ties. The latter is another argument for the key ring like NFC-tags compared to NFC-stickers because these can easily be ripped off by any person. Whereas for removing cable ties tools are needed.
For programming the NFC-tags we used freely available software like the “NFC Tools” app which is available on the Google Play Store. After storing the necessary URLs on the NFC-tags (see below), the tags get password protected to prevent any unauthorized person to change the information stored.

b. Webserver hardware:

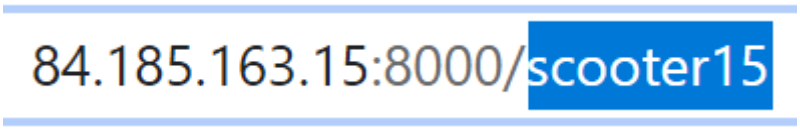
As webserver an internet capable device being able to execute python3 code is needed. Furthermore, outside access to the device ports is needed for which it could be necessary to enable the on-device firewall. If the server is part of a local network, the internet interface of that network (e.g. the router) must allow outside access of the server. For this the settings of the router must be changed for which most producer provide guidelines. It is recommended to use a static local IP-address for the server.
Note: The use of any DynDNS like service is possible. In this case the respective URLs of the DynDNS provider must be stored on the NFC-tags.
Any commercial cloud-server being able to execute python3 code is also sufficient.

2. Software components:

a. Server software

The used code is written in python3. Only the in-build modules “socket” and “time” are used. The installation of any further modules is not needed.

The code is encapsulated in four functions. The functions “createSocket()” and “bindSocket()” are used for creating a server-socket and for binding the specified host-IP and port to it. In case of exceptions, the “bindSocket()” function is retrying seven times using recursion. This number of retries can be changed inside the function. Both functions are using functions of the socket-module. The function “getHtml()” is reading from a text-file and returning its content as a string. This is used for accessing the predefined html-webpages and sending over the internet as a string.
The prototype uses four different webpages. A “scooter-page” if a scooter-request gets send, a “station-page” if a station-request gets send, a “success-page” if both – scooter- and station-requests meet the constraints (see below) and a “general-page” if no further request is send to the server.



Example request: The client is sending “scooter15” as the respective scooter-tag got scanned

The function “sendAndRecieve()” is accepting new connections in a “while-True-loop”, evaluating the client-requests, verifying if the specified constraints for a successful parking are met and sending the respective webpage to the client.
For verifying a valid parking, the IP-address of the client as well as the time of the scan/ client-request is saved. Because of privacy and technical reasons only the first 24 bit of the client address are used. For a parking to be valid a scooter and a station request must come from the same client address and be inside of a pre-defined timespan. This timespan in seconds can be changed inside the “sendAndRecieve()” function. Valid parking’s are printed out instantly and are also saved together with the corresponding timestamp. This history, storing all valid parking’s during the server session, is automatically printed out when the server is turned off. “sendAndRecieve()” also uses functions from the module “socket”, as well as functions from the module “time”.

Link to code repository:

b. Click dummy

For demonstrating a possible cooperation with a scooter provider, we created our own click dummy representing the app of the scooter provider “Lime”. This click dummy was created using the software “Adobe xD”. A browser version of the click dummy can be viewed using the link above.

Link to click dummy: https://xd.adobe.com/view/68429d80-cc19-4470-6ace-6a526129d32e-1e73/

3. Step-by-Step instructions:

i. Select a device you want to use as webserver (see “Webserver Hardware” above) and note the local IP-address of your device as well as the global IP-address of your local network. To find out your global IP, you can use the website https://icanhazip.com/ - for your local IP you can type “ifconfig” in your prompt for Linux or “ipconfig” for Windows.
   You can also us a cloud-server and note the IP-address under which the server is accessible.
   If you want to test the code only on your computer, you can also use the local IP “127.0.0.1” as local and global IP-address and open the above explained URL in your computer browser
   If you want to test the code locally inside your local network, use the local IP-address also as the global one. Be aware that server device and smartphone must be in the same local network.
ii. Download the server code as well as the html-files to your server.
iii. Open the server code and edit the following parameters:
   1. “IP_host “= local IP-address
   2. “port”: choose a port. A high number is recommended e.g. 8000
   3. “htmlFilePaths”: insert the corresponding file paths of the html-files
iv. Store the URLs on the NFC-tags using e.g. the “NFC Tools” app.
   Composition of the URL is:
   http://global-IP:port/request
   request can be the following using “X” as any digit:
   - “scooterXX”
   - “stationXX”
   - or empty
v. run code on the server.
vi. scan NFC-tags with any smartphone being NFC-capable (remember to turn NFC on) and open the URL with your browser.