TECHNISCHE UNIVERSITÄT
CHEMNITZ

Department of Electrical Engineering and Information Technology

Chair of Measurement and Sensor Technology

Project Documentation

„Intelligente Sensorsysteme“

Group: QR Code Reader Software

Members: Jeremy Bieling
Nils Wittek
Liu Jiulin
Nicolas Sammler

Supervisor: Dhia Haddad

Project: Software QR Code Reader

Date: 02.01.2023

Camera	Camera was successfully set up. We are able to select specifies cameras when the device has multiple.
GUI	Graphical interface where users can see if the code was detected and if there registered in the course
Upload TUC Cloud	Saves the name and date in a file on TUC Cloud / Nextcloud
QR Code Reading	The code can recognize QR codes and returns “false” if there is no one while he retunes “true” if there is one in the field of view of the camera

1 Abstract

For conferences, courses or congresses its useful to track who visited. With our QR code scanner everyone can check in individually. Later the organizers can generate certificates based on the Lists who visited which course or seminars.

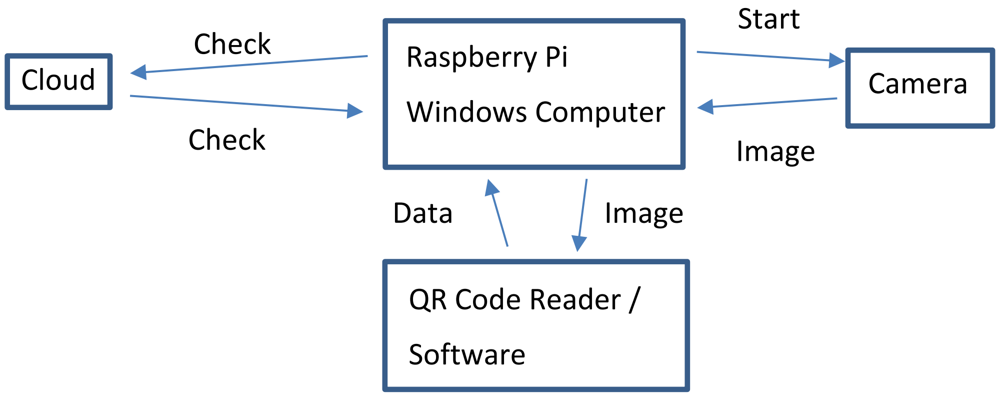
2 Member Responsibilities

Nils Wittek	GUI
Jeremy Bieling	QR Code Reader
Nicolas Sammler	Camera implementation
Liu Jiulin	Cloud upload

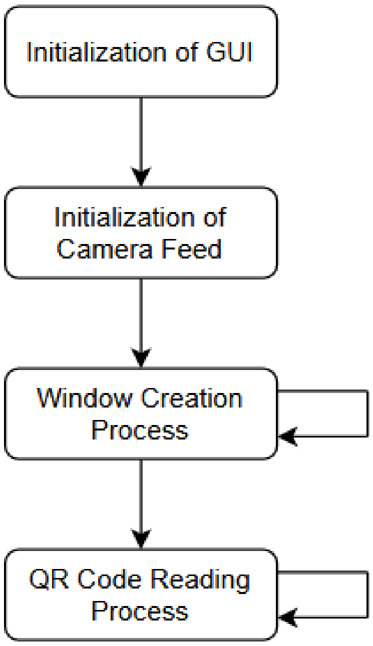
3 Functional Description

3.1 Overview

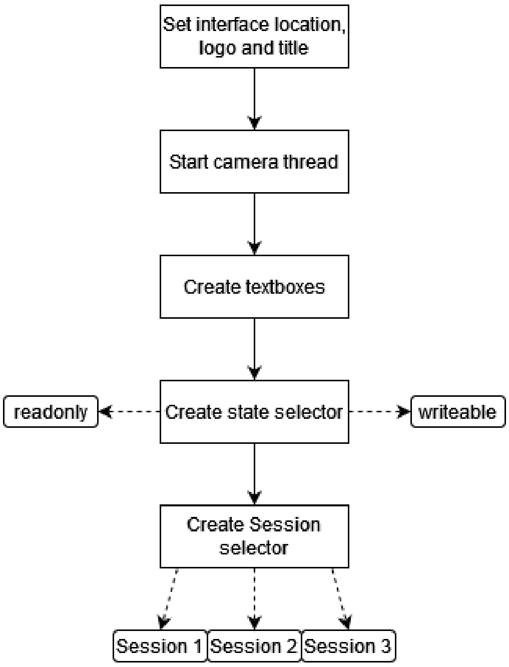
Place a short description of how to setup hardware and how to use the system for persons not being in charge with this project.



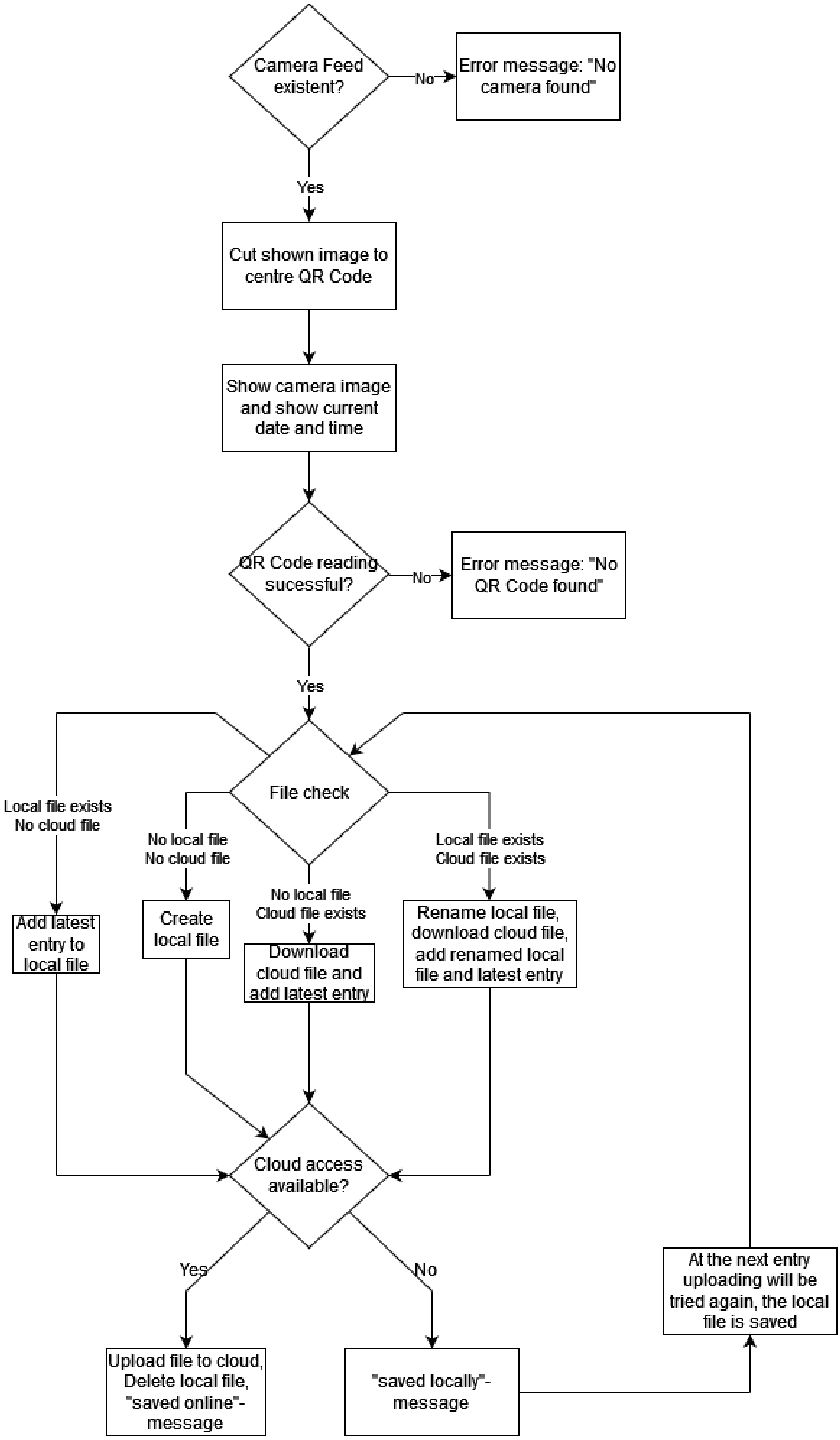
3.2 Software



Flowchart software overview



Flowchart for the window creation process


Camera Feed existent?
No
Error message: "No camera found"
Yes
Cut shown image to centre QR Code
Show camera image and show current date and time
QR Code reading sucessful?
No
Error message: "No QR Code found"
Yes
File check
Local file exists No cloud file
No local file No cloud file
No local file Cloud file exists
Local file exists Cloud file exists
Add latest entry to local file
Create local file
Download cloud file and add latest entry
Rename local file, download cloud file, add renamed local file and latest entry
Cloud access available?
Yes
No
Upload file to cloud, Delete local file, "saved online"-message
"saved locally"-message
At the next entry uploading will be tried again, the local file is saved

Flowchart of the QR code reading process

Algorithms:

The class “CameraFeed” is the software behind the app, it includes a Initialization function, a QR reading function a status message creator and a information writing function apart from the main running function. Furthermore. Initialization is used to select the camera port. The QR reading function tries to read data from a QR code by using functions from the OpenCV2 library.

When in running mode the camera is activated and tries to read a QR code with the before mentioned function. If no camera is found an error message and icon is displayed. Otherwise the user can see the recorded image and the current date and time. If unsuccessful an error message is displayed on the contrary the program will write the person’s name and login time into the attendance list when successful. Unless problems occur the user receives confirmation about his enrolment in text and image form. In the other case the user is also notified about the problem and can try again. Lastly an error message will be displayed if the attendance file can’t be uploaded to the cloud, in that case the upload is tried again at a later point in time.

If no attendance file exist at the start of registration a new local one is created. In case of a file existing in the cloud but not locally it will be downloaded and edited. When there is no internet connection the local file will be edited. The normal operation mode consist of downloading the cloud file adding information to it and then reuploading. Exception handling in case of multiple local file still being existed is also implemented. It should be noted that in order to access the file from the cloud a Nextcloud folder needs to be created. The steps therefore are listed below, it should not be forgotten to adjust the link and password accordingly.

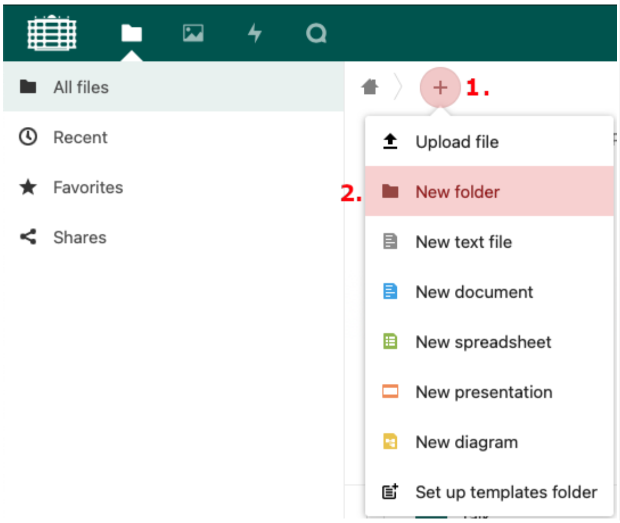
The class “app” is used to create the GUI for the user. Once again there is an initial setup which defines where the interface is located. Secondly there is a create function which creates the selection windows for the user. Here the session can be selected and the state of the app can be changed.

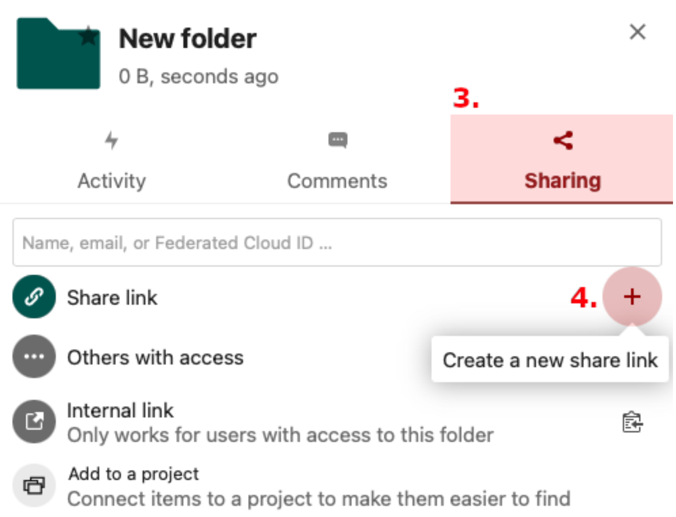
With the initialization of the “app” class the “CameraFeed” class is started as well. Both processes then loop indefinitely till the program is turned off.

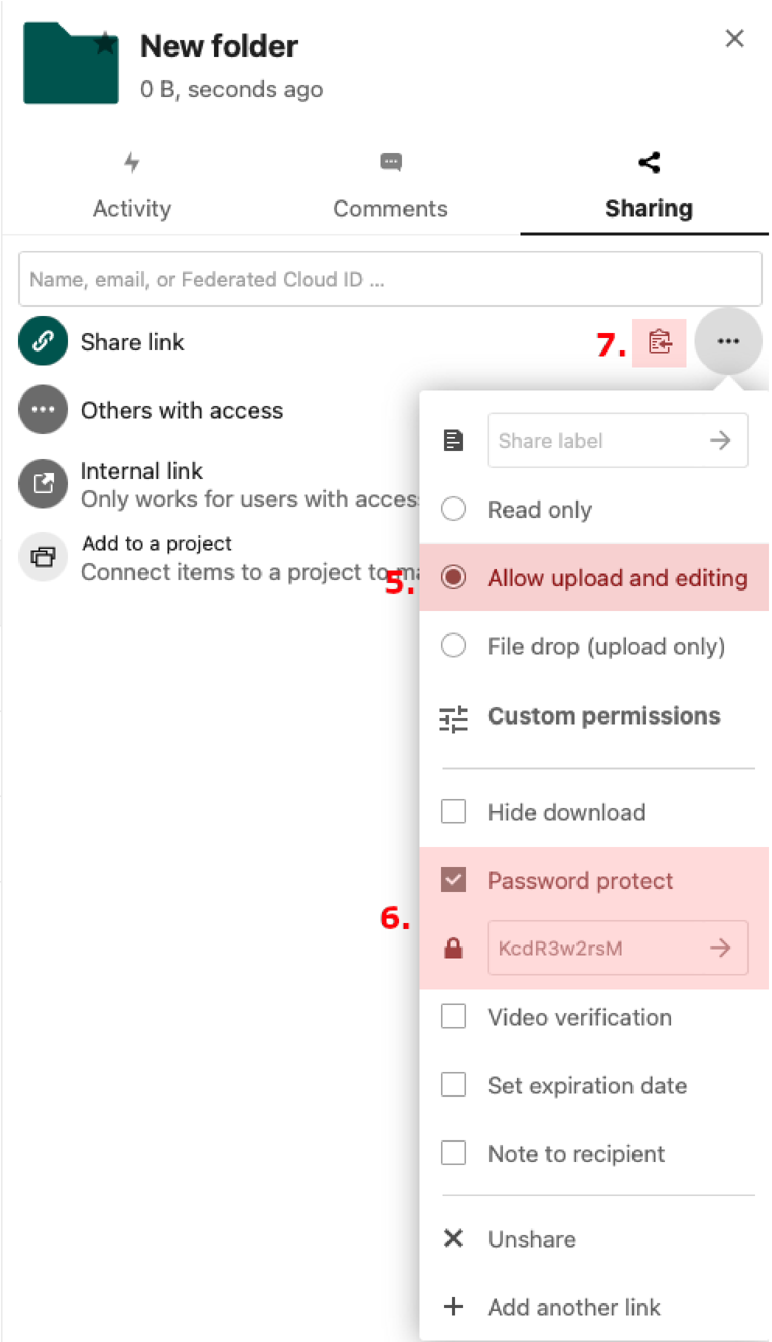
Note: For the program to work the following libraries have to be installed:

- opencv-python
- pyocclient
- pandas
- pillow
- openpyxl

Next Cloud folder creation




New folder
0 B, seconds ago
3.
Activity
Comments
Sharing
Name, email, or Federated Cloud ID ...
Share link
4.
Create a new share link
Others with access
Internal link
Only works for users with access to this folder
Add to a project
Connect items to a project to make them easier to find



○ Read only

● Allow upload and editing

○ File drop (upload only)

Custom permissions

☐ Hide download

☑ Password protect

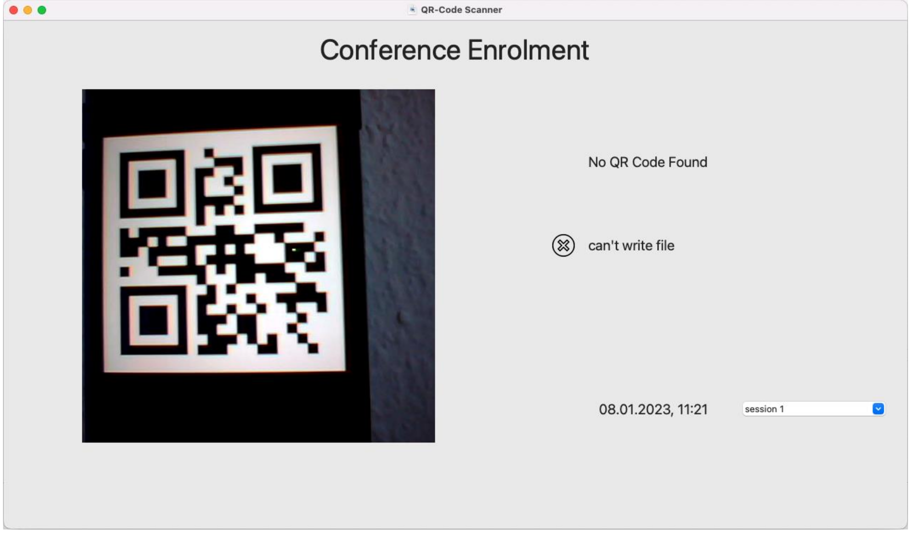
KcdR3w2rsM

☐ Video verification

☐ Set expiration date

☐ Note to recipient

Possible GUI states



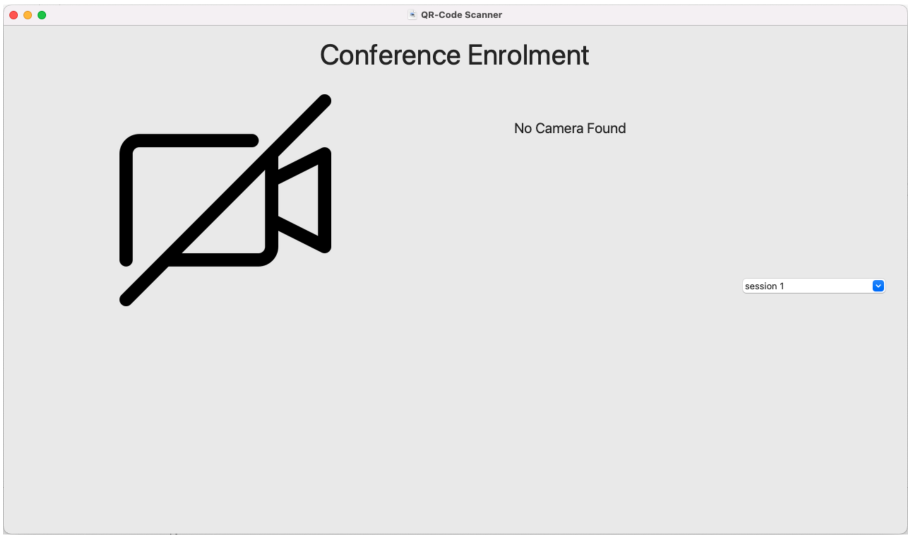
Conference Enrolment

No QR Code Found

can't write file

08.01.2023, 11:21

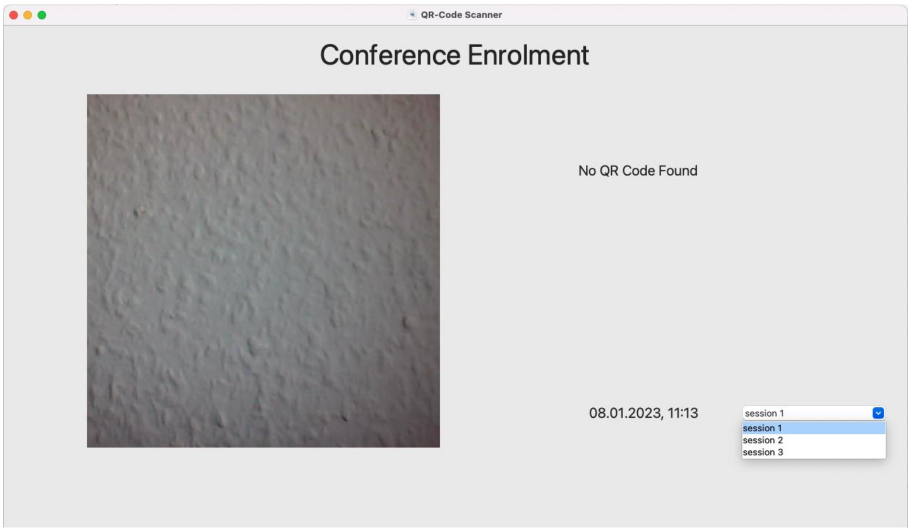
GUI when the file can’t be written



Conference Enrolment

No Camera Found

GUI when no camera signal is found

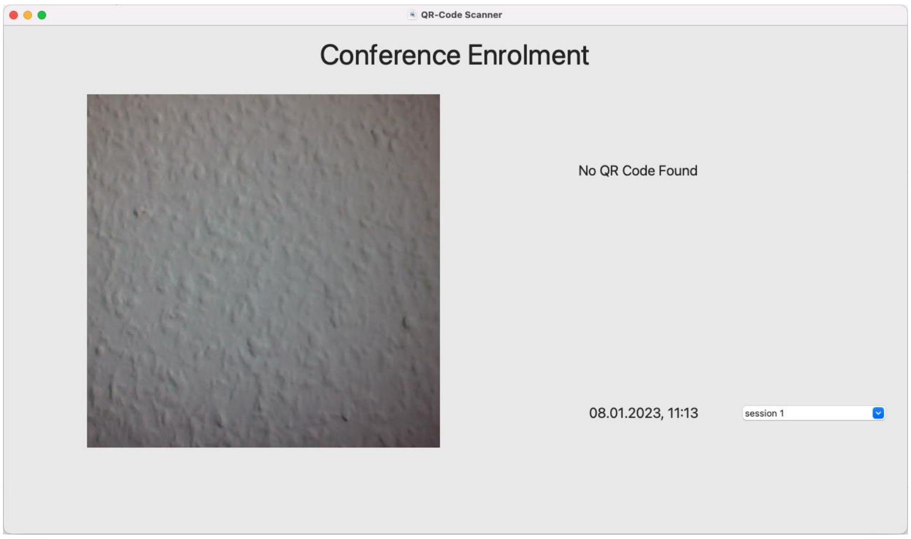


Conference Enrolment

No QR Code Found

08.01.2023, 11:13

Session selection in the GUI



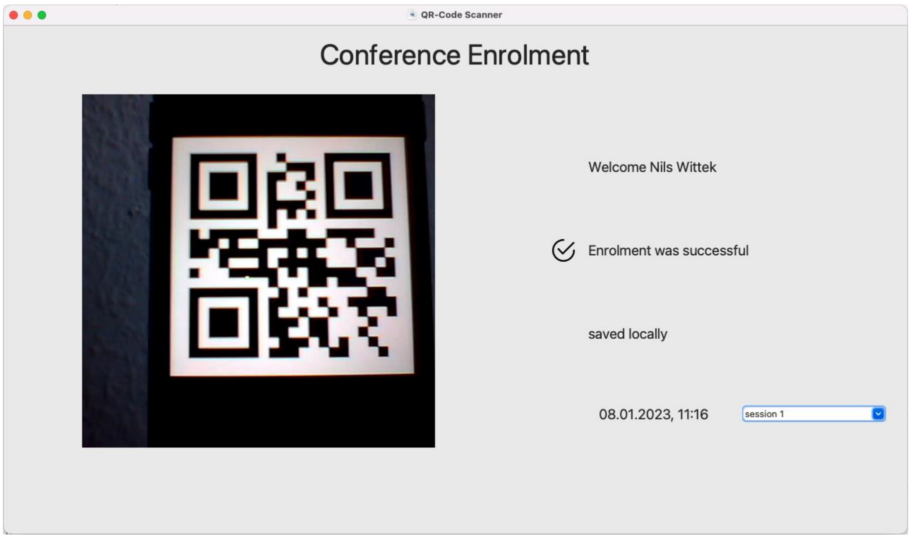
Conference Enrolment

No QR Code Found

08.01.2023, 11:13

session 1

GUI when no QR code is found



Conference Enrolment

Welcome Nils Wittek

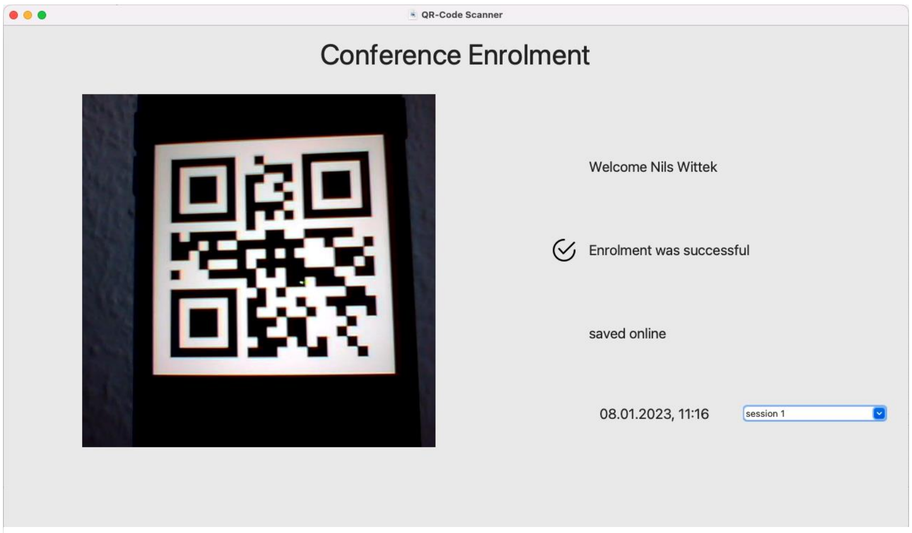
✓ Enrolment was successful

saved locally

08.01.2023, 11:16

session 1

GUI after successful enrolment with no internet connection



Conference Enrolment

Welcome Nils Wittek

✓ Enrolment was successful

saved online

08.01.2023, 11:16

session 1

GUI after successful enrolment with internet connection

3.3 Hardware

The code is written in Python and is meant for an use on a Raspbery Pi. For our development we tested it with Windows computers.

3.4 Interaction with users as well as hard- and software

The hardware allows the user to select which session he wants to attend. Afterwards this person has to scan their personal QR code in order to be written into the attendance list. Feedback regarding a successful check-in or malfunctions is given in the form of text and icons appearing.

4 Description of files

project_documentation.docx	project documentation
final_presentation.pptx	final presentation slides

additional files:

enrolment_sucessful.png	Icon for successful check-in
enrolment_failed.png	Icon for failed check-in
no-video.png	Icon for no camera signal
qr-code.ico	Icon of the program
qr-code-reader.py	Code of the program

5 References

6 Annex

QR Code Reader

January 9, 2023

[ ]:

```
import os
from os.path import exists
import tkinter as tk
from tkinter import ttk
from tkinter import font
import cv2 as cv
from PIL import Image,ImageTk
import time
import datetime
from threading import Thread
import owncloud
import pandas as pd

class CameraFeed(Thread):
    def __init__(self):
        super().__init__()

        self.cam = cv.VideoCapture(0)   # change the camera port

    def QR_read(self, image):
        try:
            detect = cv.QRCodeDetector()
            value, points, straight_qrcode = detect.detectAndDecode(image)
            return value, points
        except:
            return None

    def modifyFile(self, saveFile, data, value):
        readData = pd.read_excel(saveFile, index_col=None, header=None) # read␣
 ↪data from the given excel file
        rows = len(readData.index)  # determine the row count
        with pd.ExcelWriter(saveFile, mode='a', if_sheet_exists="overlay") as␣
 ↪writer:    # append one entry to the next empty row
            data.to_excel(writer, header=False, index=False, startrow=rows)
        app.name.config(text = "Welcome " + value)  # display the name from the␣
 ↪QR code
        self.statusMessage("success")
    def statusMessage(self, status):    # display status message + image
        if status == "file":
            message = "can't write file"
            statusImage = "enrolment_failed.png"
        if status == "success":
            message = "Enrolment was successful"
            statusImage = "enrolment_successful.png"
        app.info.config(text = message)
        newImg = ImageTk.PhotoImage((Image.open(statusImage)).resize((32,32),␣
↪Image.Resampling.LANCZOS))
        app.indicator.config(image = newImg)
        app.indicator.image = newImg

    def run (self):
        while(True):
            result, image = self.cam.read()
            currentTime = datetime.datetime.now()
            if result:
                value, points = self.QR_read(image)

                img = cv.cvtColor(image, cv.COLOR_BGR2RGB)
                img = Image.fromarray(img)

                # scale the camera image to always fit in the given space
                finalHeight = 500
                width, height = img.size
                scale = finalHeight / height

                img = img.resize((int(width * scale), finalHeight), Image.
↪Resampling.LANCZOS)
                img = img.crop(box = (int(((width * scale) / 2) - (finalHeight /
↪ 2)), 0, int(((width * scale) / 2) + (finalHeight / 2)), finalHeight))
                img = ImageTk.PhotoImage(img)
                app.imageLabel.config(image = img)
                app.imageLabel.image = img

                app.time.config(text = currentTime.strftime('%d.%m.%Y, %H:%M'))

                if points is None or value == "":   # no QR code detected
                    app.name.config(text = "No QR Code Found")
                    app.info.config(text = "")
                    app.cloudInfo.config(text = "")
                    app.indicator.config(image = '')
                    app.indicator.image = ''
                else:
                    saveFile = str(app.selected_entry.get()) + ".xlsx"  # get␣
↪the filename from the selected session entry
                    data = pd.DataFrame([[str(value), str(currentTime)]])   #␣
↪data format for storing the new entry
                    local_file_exists = exists(saveFile)    # check if a local␣
↪file exists (only when previously not transmitted to the cloud)
                    cloud_file_exists = False
                    cloud_access = True
                    file_error = False

                    oc = owncloud.Client.from_public_link('https://tuc.cloud/
↪index.php/s/areq9npZmFrsara', folder_password = "nnxoP5Y4BR")  # connect to␣
↪the cloud
                    try:
                        if oc.file_info(saveFile) != None:  # check if file␣
↪already exists on the cloud and if cloud access is available
                            cloud_file_exists = True
                    except owncloud.owncloud.HTTPResponseError:
                        cloud_file_exists = False
                    except:
                        cloud_access = False

                    if(cloud_file_exists and local_file_exists):
                        try:
                            os.rename(saveFile, "tmp.xlsx") # rename local file
                            oc.get_file(saveFile, saveFile) # download remote␣
↪file
                            readData = pd.read_excel(saveFile, index_col=None,␣
↪header=None)
                            rows = len(readData.index)
                            tmpData = pd.read_excel("tmp.xlsx", index_col=None,␣
↪header=None)    # append contents of local file to downloaded file
                            with pd.ExcelWriter(saveFile, mode='a',␣
↪if_sheet_exists="overlay") as writer:
                                tmpData.to_excel(writer, header=False,␣
↪index=False, startrow=rows)
                            self.modifyFile(saveFile, data, value)  # add␣
↪recently scanned entry to downloaded file
                            os.remove("tmp.xlsx") # delete the old local file
                        except:
                            self.statusMessage("file")
                            file_error = True

                    if(cloud_file_exists and not local_file_exists):
                        oc.get_file(saveFile, saveFile) # download remote file
                        try:
                            self.modifyFile(saveFile, data, value)  # add␣
↪recently scanned entry to downloaded file
                        except:
                            self.statusMessage("file")
                            file_error = True

                    if(not cloud_file_exists and local_file_exists):
                        try:
                            self.modifyFile(saveFile, data, value)  # add␣
↪recently scanned entry to local file
                        except:
                            self.statusMessage("file")
                            file_error = True

                    if(not cloud_file_exists and not local_file_exists):
                        try:
                            with pd.ExcelWriter(saveFile, mode='w') as writer: ␣
↪# create a new file
                                data.to_excel(writer, header=False,␣
↪index=False, startrow=0)
                            app.name.config(text = "Welcome " + value)
                            self.statusMessage("success")
                        except:
                            self.statusMessage("file")
                            file_error = True

                    if cloud_access and not file_error:
                        try:
                            oc.drop_file(saveFile)  # upload file to cloud
                            os.remove(saveFile)     # delete local file when␣
↪upload successful
                            app.cloudInfo.config(text = "saved online")
                        except:
                            app.cloudInfo.config(text = "saved locally")
                    elif not file_error:
                        app.cloudInfo.config(text = "saved locally")

                    time.sleep(3)
            else:
                app.name.config(text = "No Camera Found")   # display message␣
↪and image if no camera found
                img = (Image.open("no-video.png"))
                img = img.resize((300,300), Image.Resampling.LANCZOS)
                img = ImageTk.PhotoImage(img)
                app.imageLabel.config(image = img)
                app.imageLabel.image = img
```

```
class App(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title("QR-Code Scanner")
        #self.attributes('-topmost', 1)         # optionally keep window always␣
 ↪in foreground
        #self.attributes('-fullscreen', True)   # optionally set window to␣
 ↪fullscreen
        self.iconbitmap("qr-code.ico")

        window_width = 1280                     # define window size
        window_height = 720

        screen_width = self.winfo_screenwidth()     # determine screen size
        screen_height = self.winfo_screenheight()

        center_x = int(screen_width/2 - window_width / 2)   # determine the␣
 ↪center of the screen
        center_y = int(screen_height/2 - window_height / 2)

        self.geometry(f'{window_width}x{window_height}+{center_x}+{center_y}') ␣
 ↪# center the window on the screen

        self.columnconfigure(0, weight = 5) # set the size ratio of the␣
 ↪different columns
        self.columnconfigure(1, weight = 1)
        self.columnconfigure(2, weight = 5)

        self.camera_thread = CameraFeed()
        self.camera_thread.daemon = True    # necessary to stop the thread when␣
 ↪exiting the program
        self.create_window()        # create the GUI window
        self.camera_thread.start()  # start the camera thread

    def create_window(self):
        standardFont = font.nametofont("TkDefaultFont")

        self.programName = ttk.Label(self, text = "Conference Enrolment", font␣
 ↪= (standardFont, 40))
        self.name = ttk.Label(self, font = (standardFont, 20))  # display the␣
 ↪scanned name
        self.indicator = ttk.Label(self)    # image if enrolment was successful␣
 ↪or not
```

```
        self.info = ttk.Label(self, text = "", font = (standardFont, 20))   #␣
↪text if enrolment was successful or not
        self.cloudInfo = ttk.Label(self, text = "", font = (standardFont, 20))␣
↪# text for upload status
        self.imageLabel = ttk.Label(self)   # place for the camera image
        self.time = ttk.Label(self, text = "", font = (standardFont, 20))   #␣
↪current time
        self.selected_entry = tk.StringVar()
        self.selector = ttk.Combobox(self, textvariable=self.selected_entry,␣
↪state = 'readonly')    # session selector
        self.selector['values'] = ['session 1', 'session 2', 'session 3']   #␣
↪custom sessions can be entered here
        self.selector.current(0)    # make the first entry the default one

        # place all GUI elements on the grid layout
        self.programName.grid(column=0, row=0, columnspan=3, padx=15, pady=15)
        self.name.grid(column=2, row=2, sticky=tk.W)
        self.indicator.grid(column=1, row=3, padx=15, pady=15, sticky=tk.E)
        self.info.grid(column=2, row=3, sticky=tk.W)
        self.cloudInfo.grid(column=2, row=4, sticky=tk.W)
        self.imageLabel.grid(column=0, row=1, rowspan=5, padx=15, pady=15)
        self.time.grid(column=2, row=5, padx=15, pady=15, sticky=tk.W)
        self.selector.grid(column=2, row=5, padx=30, pady=15, sticky=tk.E)

if __name__ == "__main__":  # lauch the main GUI loop
    app = App()
    app.mainloop()
```