ACADEMY OF TECHNICAL-ART PROFESSIONAL STUDIES BELGRADE

DEPARTMENT OF HIGH SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTING

Rilak Nikola

PYTHON APPLICATION FOR RECORD OF EMPLOYEES

- final work -

Belgrade, November 2021

Candidate: **Rilak Nikola**

Index number: **73/18**

Study program: **New computer technologies**

The theme: **Python application for employee records**

Basic tasks:

1. Description of client and server side technologies.

2. Implementation of Python application for employee records.

3. User interface description.

Mentor:

Belgrade, November 2021.

Dr. Perica Strbac, professor ATUSS

SUMMARY:

Employee record software was created with the idea and goal of enabling one company to send all information about employees in an easier way. In this way, the responsible person has the space to manage the database, review the data and have an insight into all events in the company. The software itself provides the competent person with various options as well as insight into some important information, and some of them are:

- time of arrival, departure of workers and total time spent at the workplace; all
- information about the employee, such as his personal data and photographs;
- database manipulation, adding new employees, deleting and searching.

The software itself is not required in the sense that it does not require additional employee training and thus makes itself easier to commercialize and use globally.

Key words: Software, database, identification, employees

ABSTRACT:

Employee evidentication software was created with the idea and goal of enabling one company to monitor all the data and information about the employees in an easier way. In this way, the responsible person has the space to manage the database, review the data and have an insight into all events in the company. The software itself provides the competent person with various options as well as insight into some important information, and some of them are:

- time of arrival, departure of workers and total time spent at the workplace; all
- information about the employee, such as his personal data and photo; database
- manipulation, adding new employees, deleting and searching.

The software itself is not required in the sense that it does not require additional employee training and thus makes itself easier to commercialize and use globally.

Key words: Software, databse, identification, employee

THE CONTENT:

Employee identification software is a system designed to easily perform any quick check of an employee and his information such as arrival time, working hours or his name and date of birth. The idea is realized by giving each employee their personal identification card, which can be seen in Figure 1.1. and applies for / unsubscribes from it. Check-in and check-out are done by a single scan of the card, which sends data to the administrator or program manager.

In addition to their personal photo, name, surname, date of birth and position at work, the employee also received his / her unique identification number, which is presented in a bar code. The number consists of 10 randomly selected digits and is unique in itself. With the help of this number, each employee is registered in the database and in that way his data is read. See example in photo 1.1.

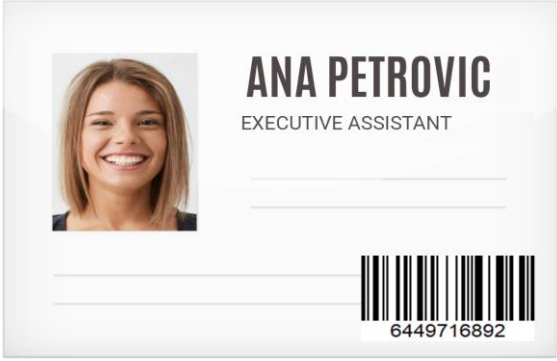

Figure 1.1 Example of an identification card

The main idea is that at the very entrance to the company there is a panel with a bar code reader where employees check in and out. In addition to the panel, green and red lights would certainly be added physically, which indicates success and failure. In that way, regular registration of working hours is established without any problems, as well as all other important information that is later served and forwarded to other staff, e.g. human resources staff, from where they can read the monthly working hours on the basis of which they make the calculation of income.

Since the needs of today's organizations are growing and computers and software are replacing every day some jobs of workers who used to work by hand, it is easy to say that every company needs such software for easier organization, management and insight into information. The physical needs of this software are not too demanding, since only one server and at least two login and logout devices are required. Usage is simplified and designed to be simple, so the software is ideal for use in a smaller company, according to its current options.

In the software description chapter, we discuss a detailed description of all interface and code aspects, including images and detailed explanations.

The development process chapter consists of a detailed description of all implementations of the software, where each development step is described in detail and further explains the things that are important for the application.

In the practical application of software, we are talking about the use of software today as well as methods of use, which further explains the ways of use.

2. SOFTWARE DESCRIPTION

First of all, the idea itself arose as a result of the need for organization and fast access to data. Although this system is certainly implemented in most of today's companies, I am sure that in smaller local companies, where the number of workers is between 10 and 20, this type of organization is really necessary. No training is required to use the software, both for administrators and employees, as the software is customized so that anyone can handle it. It is important to mention that the administrator has many more options in the software itself than the employee.

2.1. User part of the interface

The user part requires a previously turned on server, ie the administrator part of the software. After the server is turned on, the user part connects and connects to it, after which the software is completely ready for use.

Figure 2.1.1 User part of the interface

The panel or user interface itself contains only the basic necessary functions such as login, logout and current time insight (see Figure 2.1.1). After selecting one of these options (login / logout), the user is offered to read or enter his unique number or bar code, which can be seen in Figure 2.1.2.

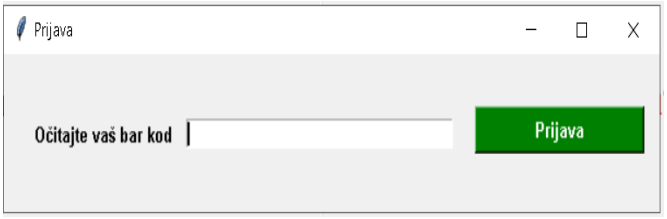

Figure 2.1.2 Bar code entry panel

It is important to note that the sign-in bar code entry panel is completely identical to the sign-in panel. In the extension we can see the login code which consists of the phase of checking the bar code, reading the database and sending the information to the server.

prijava.py

- **connecting to the server**

```
host = "127.0.0.1" #we use the ip address of the local host
port = 12345 #we take port 12345 assuming it is open s = socket.socket
(socket.AF_INET, socket.SOCK_STREAM) def connect (): #server connection
function
    s.connect (host, port) # we create an object that we place two main items,
                                        the first indicates the ipv4 address while the second indicates that u
                                          issue tcp protocol
    print ("Linked to:", host)
mainThread = threading.Thread (target = connect, args = ()). start () #placing the function in a thread
```

prijava.py

- **Check the bar code**

```
flag = "LOGIN" #we use this flag to let the server know which data to read. In case of unsubscribe, the flag would
be "Unsubscribe"
def checkBarcode (): #We check the entry or bar code
    if len (e1.get ()) == 0: #setting all the conditions so as not to make a mistake
        messagebox.showerror (title = "Error", message = "You did not enter a barcode!")
    elif len (e1.get ())> 1:
        conn = sqlite3.connect ("employee.db") cursor
        = conn.cursor ()
        bar_code = e1.get ()
        query = f "" "
                        SELECT *
                        FROM employees
                        WHERE id = '{bar_code}' AND onBusiness = 'No'
                "" "
        cursor.execute (inquiry)
        result = cursor.fetchall ()
        conn.close ()
        if result:
            messagebox.showinfo (title = "Successful login", message = "You have successfully signed up for
their working hours. ")
            check-in()
            sys.exit ()
```

```
        else:
            messagebox.showerror (title = "Error", message = "No employees were found or you are
already registered! ")
```

prijava.py

- **Send feedback to the server**

```
def login (): # We send the server data with the "LOGIN" flag so that the server knows that
                the client reports
    #query
    working hours = "START"
    request = e1.get () + "" + flag + "" + working hours s.sendall
    (request.encode ())
# FORMAT ----> 11/11/2021 - 22:03:36: employee 6449716892 SIGNED UP.
```

However, if the bar code is not correct or if the employee simply does not exist in the database, the employee receives a message informing him that an error has occurred and that bar code is not valid or that it is simply already registered / deregistered. (see Figure 2.1.3 and Figure 2.1.4).

Figure 2.1.3 Logout error

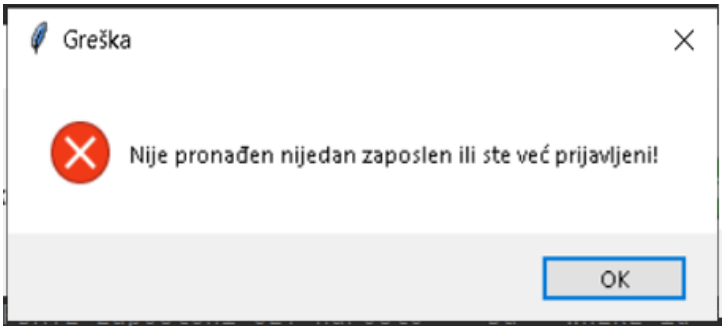

Figure 2.1.4 Login error

If the bar code is still valid and if the employee applied successfully, a message will be sent to him with the text that he was successfully registered and from that moment his working hours are running and we can see that in Figure 2.1.5.

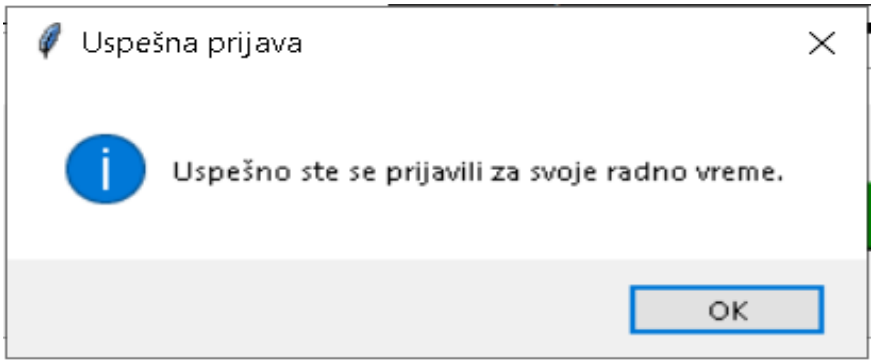

Figure 2.1.5 Successful application

2.2. Administrative part of the interface

Unlike the user interface, the admin panel or window administrative part of the interface is much more advanced and complex. It has various options such as database manipulation and employee presence testing, which can be seen in Figure 2.2.1. The administrator part itself is actually the server part of the software, through which the interface we mentioned earlier is "uploaded". So, it could be said that this is the main part of the software from which other segments come. The interface is easy to use and does not require any training, at least not in this current version.

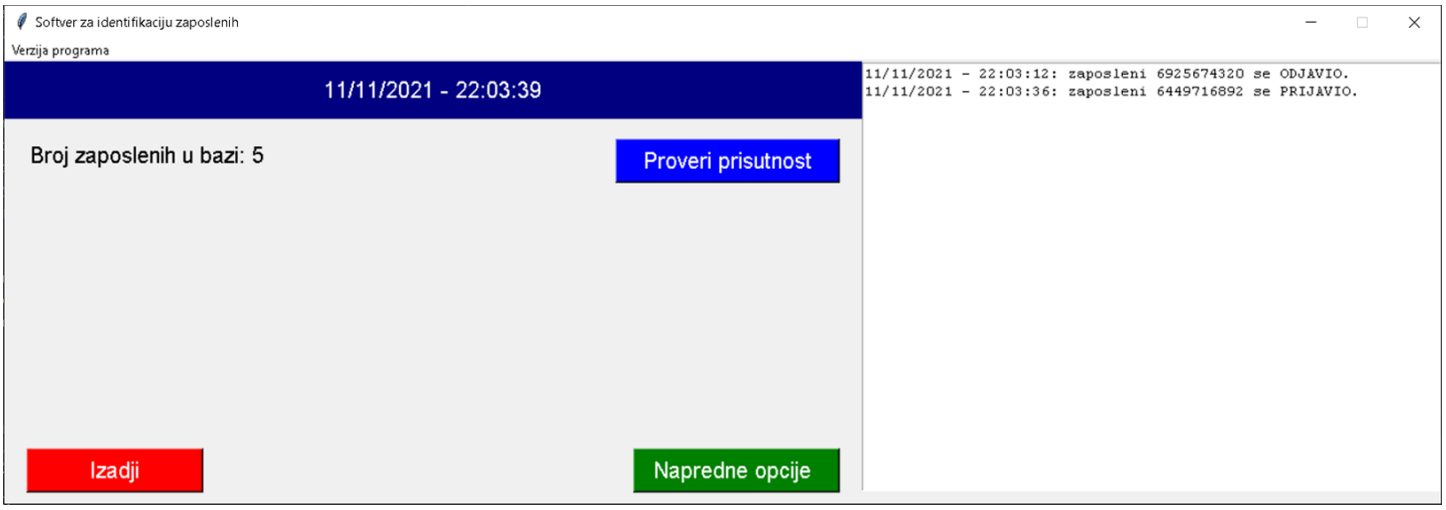

Figure 2.2.1 Administrator part of the interface

main.py

- **The structure of the main menu using tkinter (eng. Tkinter) library**

```
root = tk.Tk ()# root object in which we place the main window of the tkinter library

root.geometry ("1300x400") # geometry

root.title ("Employee Identification Software")
root.resizable (False, False) # we set the ability to expand the window to false



def Program version (): # a small function to display the version of the program built into the main
                                    window menu
    messagebox.showinfo (title = "Version", message = "v0.1")
```

```
main_menu = tk.Menu (root)
root.config (menu = main_menu)
main_menu.add_command (label = 'Program Version', command = Program Version)

root.var = tk.StringVar () # a variable that stores the value of the display label
                                                    number of employees
# control
label1 = tk.Label (root, text = "Time", fg = "white", bg = "navy", width = 70, font = "15", height = 2) label1.grid (row
= 0, column = 0, sticky = tkinter.N)

label2 = tk.Label (root, textvariable = root.var, font = "10") label2.grid (row = 0, column
= 0, sticky = tkinter.NW, pady = 70, padx = (20, 0))

koJePrisutan = tk.Button (root, padx = 15, width = "15", text = "Check Presence", bg = "blue", fg = "white",
command = koJeTu, font = "2")
koJePresut.grid (row = 0, column = 0, sticky = tkinter.NE, pady = 70, padx = (0, 20))

back = tk.Button (root, padx = 50, text = "Get out", bg = "red", fg = "white", command = ugasiProgram, font =
"10")
back.grid (row = 0, column = 0, sticky = tk.NW, pady = (350, 0), padx = (20, 0))

advancedOptionsButton = tk.Button (root, padx = 15, text = "Advanced Options", bg = "green", fg = "white",
command = advancedOptions, justify = tk.LEFT, anchor = "w", font = " 10 ")
advancedOptionsButton.grid (row = 0, column = 0, sticky = tkinter.NE, pady = (350, 0), padx = (0, 20))

text1 = tkinter.Text (root, width = "70")
text1.grid (row = 0, column = 1)
```

main.py

- **Function for listening to client connections and messages**

```
local_pc = "" # we set the ip address or host to empty because the server itself
                takes your ip address
port = 12345 # we take port 12345 assuming it is free

s = socket.socket (socket.AF_INET, socket.SOCK_STREAM) # we create the object we are placing
    two main items, the first indicates the ipv4 address while the second indicates that it is a tcp protocol
```

```
s.setsockopt (socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # we set the possibility
                                                        reusing the same address
def listenConnections(): # a function for listening to all connections, which is later placed in the thread
    s.bind (local_pc, port) # setting a predefined ip address and port in our facility
or socket
    s.listen () # we put the object in a state of listening for new connections
    connection = sqlite3.connect ("employee.db") # connecting to a database
    cursor = connection.cursor() # creating an object with the help of which we manage the base
    print ("Listening ...")
    global start_time
    while True: # a loop that regulates the successful connection of the client
        conn, addr = s.accept () # the two most important objects with which we can see the content of the message and the
address from which the message was sent
        print ("Upcoming connection:", addr)
        try:
            while True:
                data = conn.recv (1024) # an object that contains a received message with a size buffer
1024 bytes
                request = data.decode() #      we decrypt the message since all content is over the network
sends in encoded form


                if ("REGISTRATION" and requirements): # we make inquiries to differentiate between what is an application and what is
unsubscribe in message
                    currentTime = datetime.datetime.now (). strftime (% d "+" / "+"% m "+" / "+"% Y
-% H:% M:% S ")
                    name = request.split ("") [0]
                    text1.insert (END, currentTime + ": employees" + name + "LOGIN. \ n")


                    f = open ("log.txt", "a")
                    f.write (currentlyTime + ": employees" + name + "SIGNED IN. \ n")
                    f.close ()
                    start_time = time.time ()
                    start_time_base = (int (start_time))
                    cursor.execute ("UPDATE EMPLOYEES SET at BUSINESS = ?, start_time =? WHERE id =?",
                                    ('Yes', start_time_base, name)
```

```
                    connection.commit ()


                if ("LOGOUT" and requests):
                    currentTime = datetime.datetime.now (). strftime (% d "+" / "+"% m "+" / "+"% Y
-% H:% M:% S ")
                    name = request.split ("") [0]
                    text1.insert (END, currentTime + ": employee" + name + "has signed out. \ n")


                    query1 = f "" "
                                SELECT seconds
                                FROM employees
                                WHERE id = {name}
"" "


                    cursor.execute (inquiry1)
                    seconds = cursor.fetchall () [0]
                    rowsCut1 = str (seconds) .split ('(', 1)          # we shorten the string before the number
                    preString1 = rowsCut1 [1]
                    posleString1 = preString1.split (",", 1)          # we shorten the string after the number
                    numberSeconds = afterString1 [0] # result as an integer
                    print (numberSeconds)


                    query2 = f "" "
                                                SELECT start_time
                                                FROM employees
                                                WHERE id = {name}
"" "


                    cursor.execute (inquiry2)
                    start = cursor.fetchall () [0]
                    row = str (start) .split ('(', 1)          # we shorten the string before the number
```

```
                                pre = row [1]
                                after = pre.split (",", 1) # we shorten the string after the number
                                start_number = after [0] # result as an integer


                                f = open ("log.txt", "a")
                                f.write (currentlyTime + ": employees" + name + "have logged out. \ n")
                                f.close ()
                                e = int (time.time () - float (start_number))
                                print (s)
                                e2 = int (numberSeconds)
                                e3 = e + e2
                                print (e3)
                                hour = ('{: 02d}: {: 02d}'. format (e3 // 3600, (e3% 3600 // 60)))
                                print (hour)
                                cursor.execute ("UPDATE EMPLOYEES SET at WORK = ?, WORK HOURS = ?, seconds =? WHERE
id =? ", ('No', hour, e3, name))
                                connection.commit ()
                            break
                    except:
                        print ("---")


tk1 = threading.Thread (target = listenConnections, args = ()). start () # Putting a function under the thread so that it
can always receive connections, regardless of whether some actions are happening or not
```

2.2.1 Basic options

If we look at the previous photo 2.2.1, we can see some of the options that the administrator has. One of the most important options is the ability for the administrator to check the presence and thus it is very easy to get information about who is currently at work and who is not. In the extension follows the code of the function for dynamic display of employees, where the rule applies:

- Workers who they are not those present have gray photographs
- Workers who they did attendees have color photographs

ko-je-tu.py

- **Function for creating the interface of the present workers**

```
    class DynamicGrid (tk.Frame): # In this case, we used the class for easier dynamics
                                                  data display
     def __init __ (self, parent, * args, ** kwargs):
            tk.Frame .__ init __ (self, parent, * args, ** kwargs)
            frame = tk.Frame (self, width = 300, height = 300) # the frame object it already uses
                                                                  as the parent tkinter root window
            frame.pack (expand = True, fill = tk.BOTH)
            canvas = tk.Canvas (frame, bg = '# FFFFFF', width = 300, height = 300, scrollregion = (0, 0, 500,
500))
            hbar = tk.Scrollbar (frame, orient = tk.HORIZONTAL) hbar.pack (side =
            tk.BOTTOM, fill = tk.X) # we add a scrollbar hbar.config (command =
            canvas.xview)
            # vbar = tk.Scrollbar (frame, orient = tk.VERTICAL)
            # vbar.pack (side = tk.RIGHT, fill = tk.Y)
            # vbar.config (command = canvas.yview) canvas.config (width =
            300, height = 300) canvas.config (xscrollcommand = hbar.set)
            canvas.pack (side = tk.LEFT, expand = True, fill = tk .BOTH)



            self.text = tk.Text (canvas, wrap = "char", borderwidth = 0, highlightthickness = 0,
                                        state = "disabled", cursor = "arrow")


            self.text.pack (fill = "both", expand = True)
            self.boxes = []

     def add_box (self, color = None): # frame-making function


            conn = sqlite3.connect ("employee.db") cursor
            = conn.cursor ()


            query = f "" "
                                    SELECT *
                                    FROM employees
                        "" "


            cursor.execute (inquiry)
            result = cursor.fetchall ()
```

```
            conn.close ()

            for i in result:
                box = tk.Frame (self.text, bd = 1, relief = "sunken",
                                width = 25, height = 25)
                bar_code = f "{i [0]}"
                at work = f "{i [7]}"
                image = f "{i [9]}"

                img = (Image.open ("" + image)) if
                ("No" in Business):
                    resized_image = img.resize (150, 150), Image.ANTIALIAS) .convert ('L') else:

                    resized_image = img.resize (150, 150), Image.ANTIALIAS)
                new_image = ImageTk.PhotoImage (resized_image)
                label2 = tk.Label (box, image = new_image)
                label2.image = new_image
                label2.grid ()

                btn = tk.Button (box, text = bar_code, cursor = "hand2")
                btn.grid ()
                self.boxes.append (box)
                self.text.configure (state = "normal")
                self.text.window_create ("end", window = box)
                self.text.configure (state = "disabled")

class display (object):
    def __init __ (self):
            self.root = tk.Tk ()
            self.root.title ("Who's present?")
            self.dg = DynamicGrid (self.root, width = 500, height = 200)
            self.dg.pack (side = "top", fill = "both", expand = True)

    def start (self):
            self.dg.add_box ()
            self.root.mainloop ()

display (). start ()
```

By analyzing the previous code, we can see how to structure and write the appropriate data in the details panel itself. A query from the database is used, which returns all employees, and based on that, images of employees are created later in the extension of the same function. The photographs themselves are painted in gray if the employee is absent and remain in color if present. This is done by a simple query whose result is shown in Figure 2.2.1.2.

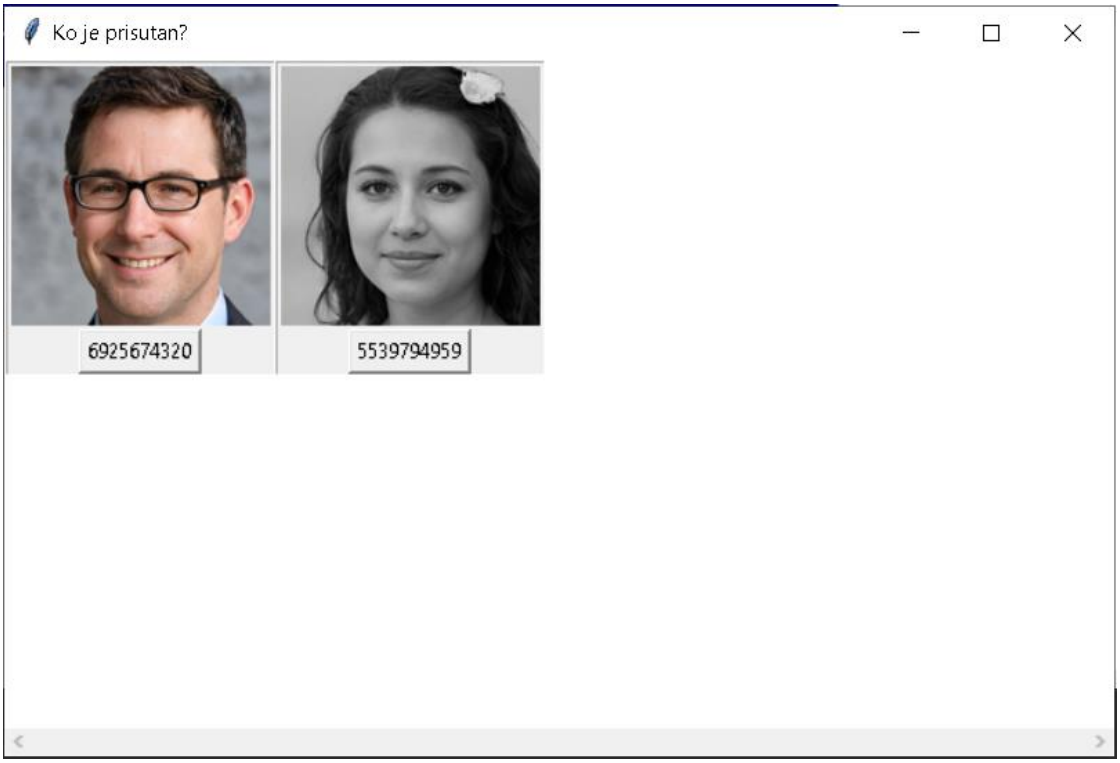

Figure 2.2.1.2 Who's present interface

The idea itself is, as we have already said, that the images of employees who are present are in color, while on the other hand, employees who are not at work have a gray image. From this point of view, there is enough information (ID number), but in the future it would certainly become an option to display all the information of that employee, such as we want or red light next to the pictures, for easier recognition. It should certainly be taken into account that as the needs of the company grow, more options and various possibilities would be added.

In the continuation of the description of the code, on the right side of the interface we can see the existence of a text box or text field in which information is printed about when the employee checked in or checked out. There is also a check-in / check-out date and time and a unique number of employees (bar code). This information is also written to the text file "log" (below is an example). The exceptional importance of this data is that it gives the administrator a complete insight into the time of registration or deregistration, as well as the worker who registered or deregistered. Without this option, it would be difficult to keep track of who came to work or who left.

06/11/2021 - 20:30:47: Employee 6449716892 LOGOUT. 06/11/2021 - 20:30:59: employee 5539794959 LOGOUT. 06/11/2021 - 20:31:10: employee 6925674320 LOGOUT. 06/11/2021 - 20:31:28: employee 5539794959 SIGNED UP. 06/11/2021 - 20:38:05: employee 6925674320 APPLIED. 06/11/2021 - 20:44:46: employee 6925674320 LOGOUT. 06/11/2021 - 20:44:49: employee 5539794959 LOGOUT. 06/11/2021 - 20:46:40: employee 5539794959 SIGNED UP. 06/11/2021 - 20:49:20: Employee 6449716892 SIGNED UP. 06/11/2021 - 20:51:52: employee 6449716892 LOGOUT. 06/11/2021 - 20:51:57: employee 5539794959 LOGOUT. 06/11/2021 - 20:55:55: employee 5539794959 APPLIED. 06/11/2021 - 20:57:30: employee 5539794959 LOGOUT. 06/11/2021 - 21:28:49: employees 6925674320 APPLIED. 06/11/2021 - 21:29:05: employee 6449716892 SIGNED UP. 07/11/2021 - 14:08:11: employee 6925674320 LOGOUT. 07/11/2021 - 14:08:15: employee 6449716892 LOGOUT. 08/11/2021 - 12:06:15: employee 5539794959 APPLIED. 08/11/2021 - 12:07:04: Employees 6925674320 APPLIED. 08/11/2021 - 12:07:16: employee 6449716892 SIGNED UP. 08/11/2021 - 12:07:31: employee 5539794959 LOGOUT. 08/11/2021 - 12:16:08: employee 6449716892 LOGOUT. 08/11/2021 - 12:18:40: Employee 6449716892 SIGNED UP.

Part of a document that records all check-ins and check-outs.

There is an idea as well as the possibility that there is a special database with all this information, so based on that, all the options for working with this data could be added. For example. search when the employee under ordinal number 23241414 checked in or checked out or whether the same worker checked out that day at that time or earlier.

- **Day**/**date**/**Year** - **A clock**:**Minute**:**Second**: Staff **ID_NUMBER** se **logged in / out.**

A concrete example follows:

- 14/08/2021 - 14:39:01: employees **6548954123** signed up.
- 14/08/2021 - 17:10:48: employees **1264333994** checked out.

2.2.2 Advanced options

In addition to the basic ones, there are also advanced options that the administrator can access, so in the picture below we can see that there are various ways to manage the database. The first and most important option is to add a new worker, which we will describe in detail later. In addition to this main function, there are functions for listing, deleting and searching, as we can see in Figure 2.2.2.1. Each function is performed on the basis of a unique number of employees.

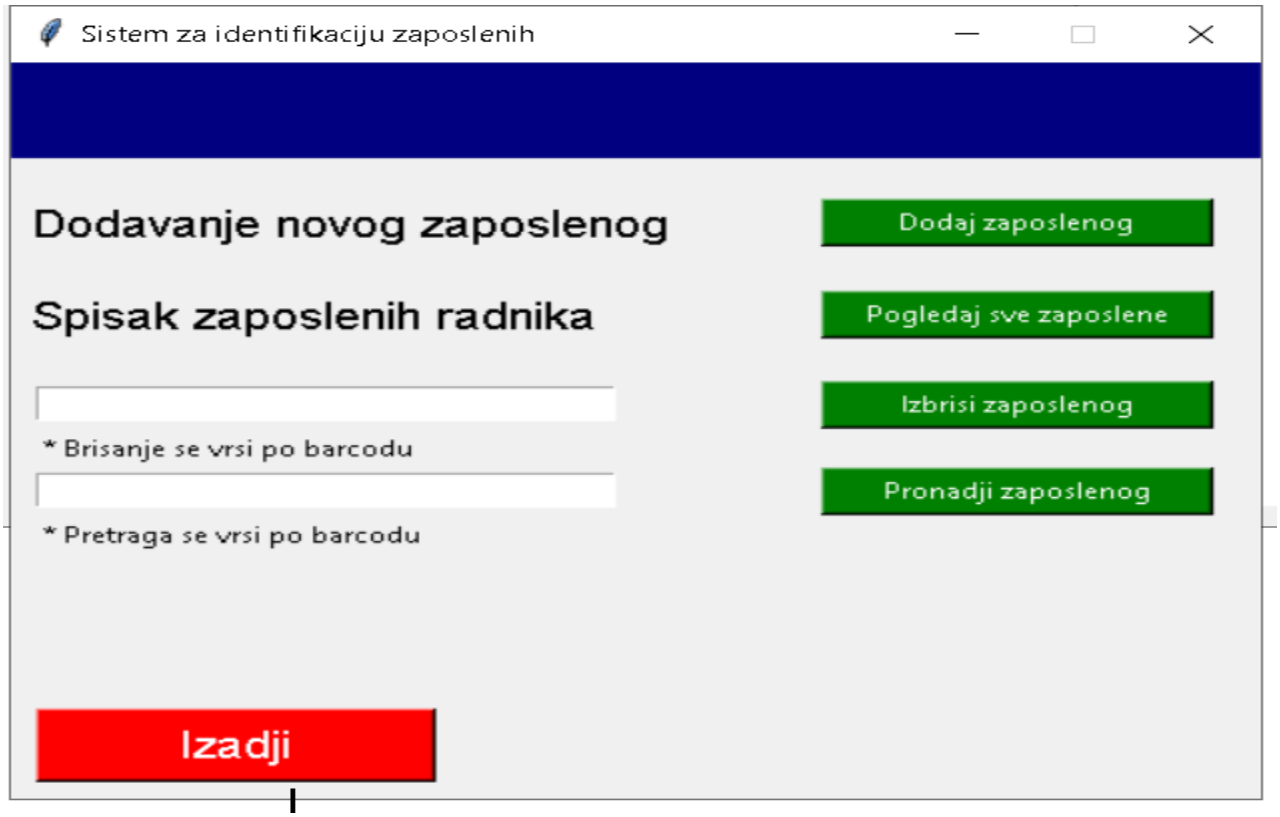

Figure 2.2.2.1 Advanced Administrator Options

Advanced options include:

- adding a new employee;
- list of all employees;
- deletion of an employee;
- employee search.

It is important to note that both search and deletion are performed via a bar code. The reason why the bar code is used is that it is unique for each worker and is treated as an identification number (ID). In order for deletion and search to be accurate, it is necessary to submit precise data for validation, which e.g. name or surname is not the case. Guided by the logic of, say, using a search name, there are ten employees with the name Milos when searching, and that is at best, it should always be taken into account that there may be 100 employees named Milos. Based on that, the base can be congested and traffic slowed down. It is certainly a better option to search by bar code, but I also think that in the future I would definitely add a search by other parameters, in order to complete these two functions.

In the extension we can see the search and delete codes as well as the way I solved the conditions and set the boundaries of the functions and also solved all possible errors. Both functions are performed in the same way, respecting the same order of the algorithm.

advanced_options.py

- Bar code employee search function

```
def findEmployee ():
    if len (e2.get ()) == 0: # A small check of the entry field to avoid checking
                                empty field.
        messagebox.showerror (title = "Error", message = "You did not enter a barcode!")
    elif len (e2.get ())> 1: # Check again if the field is empty.
                                It was expected not to, for that reason the function continues


        conn = sqlite3.connect ("employee.db") # Opening a connection to the base
        cursor = conn.cursor ()
        bar_code = e2.get ()
        query = f "" "
                            SELECT *
                            FROM employees
                            WHERE id = '{bar_code}'
                    "" " # Making queries
        cursor.execute (inquiry)
        global score
        result = cursor.fetchall ()
      if result:
            resultWindow ()
        else:
            messagebox.showerror (title = "Error", message = "No employees were found under this
ID! ")
```

advanced_options.py

- **The “resultWindow ()” function that returns the interface with all search data.** Without
- it, it would be worthless to search, since every function strives for a result.

```
def resultWindow ():
    global score

    id = result [0] [0]
    name = result [0] [1]
    last name = result [0] [2]
    date of birth = result [0] [3]
    position = result [0] [4]
    working hours = result [0] [5]
    at work = result [0] [7]
    image = result [0] [9]

    master = tk.Toplevel ()
    master.title ("Result")
    master.geometry ("360x450")
    master.resizable (False, False)

    idLabel = tk.Label (master, text = "Employee ID", font = 8, justify = tk.LEFT)
    idLabel.grid (row = 0, column = 1, sticky = tk.W)
    #
    e1 = tk.Entry (master, width = 25)
    e1.grid (row = 0, column = 2, padx = (13, 0))
    e1.insert (0, id)
    e1.configure (state = tk.DISABLED)

    #
    nameLabel = tk.Label (master, text = "Employee Name", font = 8, justify = tk.LEFT)
    nameLabel.grid (row = 1, column = 1, sticky = tk.W)
```

```
#
e2 = tk.Entry (master, width = 25)
e2.grid (row = 1, column = 2, padx = (13, 0))
e2.insert (0, name)
#
last nameLabel = tk.Label (master, text = "Employee last name", font = 8, justify = tk.LEFT) last
nameLabel.grid (row = 2, column = 1, sticky = tk.W)
#
e3 = tk.Entry (master, width = 25) e3.grid (row =
2, column = 2, padx = (13, 0)) e3.insert (0,
surname)
#
yearLabel = tk.Label (master, text = "Date of birth", font = 8, justify = tk.LEFT) yearLabel.grid (row
= 3, column = 1, sticky = tk.W)
#
e4 = tk.Entry (master, width = 25) e4.grid (row =
3, column = 2, padx = (13, 0)) e4.insert (0, date
of birth)
#
positionLabel = tk.Label (master, text = "Employee position", font = 8, justify = tk.LEFT) positionLabel.grid
(row = 4, column = 1, sticky = tk.W)
#
e5 = tk.Entry (master, width = 25) e5.grid (row =
4, column = 2, padx = (13, 0)) e5.insert (0,
position)

radniSatiLabel = tk.Label (master, text = "Radni sati", font = 8, justify = tk.LEFT)
radniSatiLabel.grid (row = 5, column = 1, sticky = tk.W)
#
e6 = tk.Entry (master, width = 25)
e6.grid (row = 5, column = 2, padx = (13, 0))
e6.insert (0, working hours)
```

```
    naPosluLabel = tk.Label (master, text = "Na poslu:", font = 8, justify = tk.LEFT)
    naPosluLabel.grid (row = 6, column = 1, sticky = tk.W)
    #
    e7 = tk.Entry (master, width = 25) e7.grid (row =
    6, column = 2, padx = (13, 0)) e7.insert (0,
    naPoslu)
    #
    # labelEmpty = tk.Label (master, height = 10)
    # labelPrazna.grid (row = 5, column = 2)
    #
    label = tk.Label (master, text = "")
    label.grid (row = 7, column = 2)

    img = (Image.open ("" + image))
    reduced_image = img.resize (180, 180), Image.ANTIALIAS)
    new_image = ImageTk.PhotoImage (thumbnail)

    label2 = tk.Label (master, image = new_image, width = 150, height = 245)
    label2.image = new_image
    label2.grid (row = 7, column = 1)
```

advanced_options.py

- **Function for deleting an employee from the database**
- The only feature that requires administrator confirmation to avoid an unwanted error

```
def validationForDelete ():
    if len (e3.get ()) == 0:
        messagebox.showerror (title = "Error", message = "You did not enter a barcode!")
    elif len (e3.get ())> 1:
        conn = sqlite3.connect ("employee.db")
        cursor = conn.cursor ()
        bar_code = e3.get ()
```

```
        query = f "" "
                        SELECT *
                        FROM employees
                        WHERE id = '{bar_code}'
                "" "
        cursor.execute (inquiry)
        global score
        result = cursor.fetchall ()
        if result:
            id = result [0] [0]
            name = result [0] [1]
            last name = result [0] [2]
            date of birth = result [0] [3]
            position = result [0] [4]
            msg = messagebox.askquestion (title = "Confirmation", message = "The next employee will be
deleted from database: \ t \ n \ n "
                                                    "ID:" + id + "\ n" +
                                                    "Name:" + name + "\ n" +
                                                    "Last name:" + last name + "\ n" +
                                                   "Date of birth" + date of birth + "\ n" +
                                                   ("Position:" + position + "\ n")
            if msg == 'yes':
                print ("Successful")
                query = f "" "
                                DELETE FROM employees
                                WHERE id = {id}
                        "" "
                cursor.execute (inquiry)
                conn.commit ()
                messagebox.showinfo ("Successful deletion!", message = id + "deleted!")
        else:
            messagebox.showerror (title = "Error", message = "No employees were found under this
ID! ")
```

As we mentioned earlier, the main function is the option to add a new employee by which the administrator can, manually via the interface, supplement the database with a new employee including all his arbitrary attributes and even a unique number or bar code. This is especially important because the administrator is left with the option to change the pre-generated 10-digit number to another number. Of course, some limits need to be set, such as searching for that bar code in case it already exists in the database. This is done to avoid errors and also to deprive the administrator of the ability to make errors or misuse the software. The option of adding workers includes data necessary for one worker, which can also be seen in Figure 2.2.2.2, and they are:

- Employee ID, this item is already filled in with a number of 10 randomly selected digits; The name
- of the employee;
- Employee last name;
- Date of Birth;
- Employee position;
- Photography.

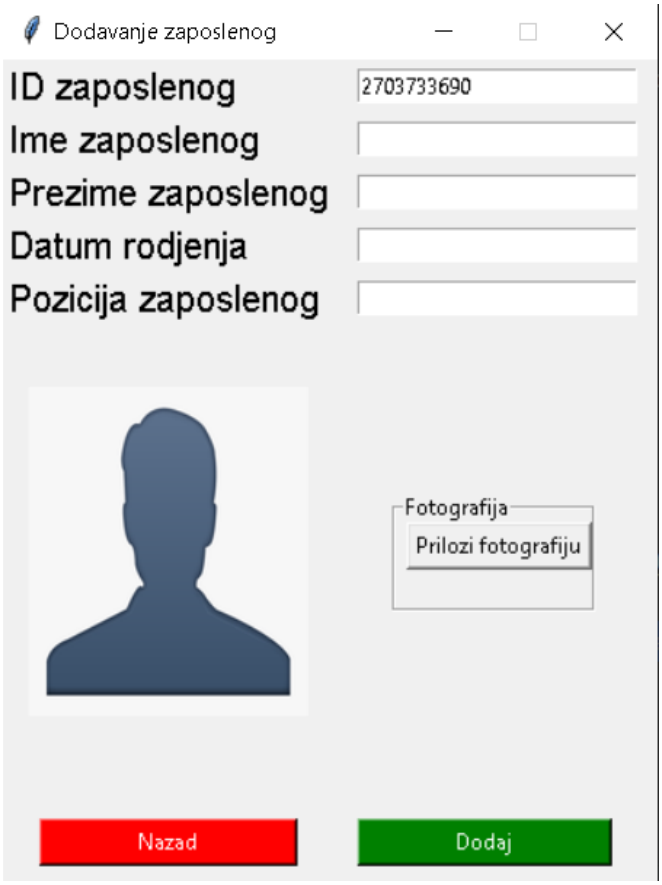

Figure 2.2.2.2 Adding an employee

In the extension, it follows the code of the function for adding a new employee, in which the bar code or the identification number of the employee is also generated - the most important attribute of the employee. In addition, some data that are not visible to the administrator, the program itself generates and enters them into the database. The reason for this is that this data is not important for the interface nor is it there to be displayed but has other functions, it is more detailed in the code.

add_employee.py

- **Add employee function**

```
def addEmployee ():
    global number # global variable
    global trajectory # global variable
    id = e1.get ()
    name = e2.get ()
    last name = e3.get ()
    date of birth = e4.get ()
    position = e5.get () #We capture text box entries (entry box) and
                         we place them in variables
    working hours = 0
    at work = "No" #The default attribute for a worker is not at work
    source_path = r "" + path
    destination_path = r "C: \ Users \ Rile \ PycharmProjects \ Graduate \ employee" + "\\" + first name + "-" + last name +
    ".jpg"
    shutil.copy (source_path, destination_path) #copy a photo to a directory and
                                                image name formatting
    conn = sqlite3.connect ("employee.db")
    seconds = 0
    conn.execute ("insert into Employee (id, name, surname, date of birth, position, working hours, seconds, at work,
picture) values   (?,?,?,?,?,?,?,?,?)",
                  (id, name, surname, date of birth, position, working hours, seconds, at work,
destination_path)) # Enter all data in the database, including those that
                     are not visible to the administrator
    print ("Successful database entry.")
    conn.commit ()
    conn.close ()
    messagebox.showinfo (title = None, message = "You have successfully added a worker!")
```

After analyzing the previous code used to add an employee, in addition to all these options, some of which are less and some more important, there is an option to view all employees, which the administrator has an insight into each employee, his data and general verification. employees at work. One of the more important data is the number of working hours, as we see in Figure 2.2.2.3. in extension. This data is extremely important for the entire company and we can freely say that this is one of the most important data of any company, given that based on it (working hours) are calculated

income of one employee. The way in which the system of collecting working hours and entering them is designed is that the registration time is entered in the database under the start_time column, so the same value is loaded and the time spent is logged out after the user logs off. The system has been tested several times with different methods and has always given a successful result. We can also see more in the code that follows.

prikaz.py

- **Function for graphical display of employees**

```
root = tkinter.Tk () # the building in which we place the main window of the library library
root.title ("Employees currently at work")
root.geometry ("475x520")
rows = 0

def update_scroll (event):
    photoCanvas.configure (scrollregion = photoCanvas.bbox ("all")) # update function
                                                                    scrollbar positions
photoFrame = tkinter.Frame (root, bg = "# EBEBEB", width = "500", height = "400")
photoFrame.grid ()
photoFrame.rowconfigure (0, weight = 1)
photoFrame.columnconfigure (0, weight = 1)

photoCanvas = tkinter.Canvas (photoFrame, bg = "# EBEBEB", height = "500", width = "450")
photoCanvas.grid (row = 0, column = 0, sticky = "nsew")

canvasFrame = tkinter.Frame (photoCanvas, bg = "# EBEBEB", width = "500")
photoCanvas.create_window (0, 0, window = canvasFrame, anchor = 'nw')
# we make a frame and a canvas in which we place that frame

def create_frame_frame (): # a function for creating a frame in which everyone can be accommodated later
                           employee data
    conn = sqlite3.connect ("employee.db") # connecting to a database
    cursor = conn.cursor () # we create an object in which we place the cursor with the help of which we work
                            inquiries
```

```
        query = f "" " # we make an inquiry based on which we take only users who are currently at work
                                SELECT *
                                FROM employees
                                WHERE on Business = 'Yes'
                        "" "


        cursor.execute (inquiry) # we execute the query
        result = cursor.fetchall ()
        conn.close ()


        for i in result:
                        step = tkinter.LabelFrame (canvasFrame, text = "Details:", width = "400")
                        step.grid (row = rows, columnspan = 7, sticky = 'W', padx = 5, pady = 5, ipadx = 5, ipady = 5)
                        tkinter.Label (step, text = "ID", font = "Arial 8 bold"). grid (row = 0, sticky = 'E', padx = 5,
pady = 2)
                        tkinter.Label (step, text = "Name", font = "Arial 8 bold"). grid (row = 1, sticky = 'E', padx = 5,
pady = 2)
                        tkinter.Label, step, text = "Surname", font = "Arial 8 bold "). grid (row = 2, sticky = 'E',
padx = 5, pad = 2)
                        tkinter.Label (step, text = "Date birth ", font = "Arial 8 bold "). grid (row = 3,
sticky = 'E', padx = 5, pady = 2)
                        tkinter.Label (step, text = "Position", font = "Arial 8 bold"). grid (row = 4, sticky = 'E',
padx = 5, pad = 2)
                        tkinter.Label (step, text = "Working hours", font = "Arial 8 bold"). grid (row = 5, sticky = 'E',
padx = 5, pad = 2)
                        tkinter.Label (step, text = "At work", font = "Arial 8 bold"). grid (row = 6, sticky = 'E',
padx = 5, pad = 2)
                        for x in range (13):
                                tkinter.Label (step, text = "", font = "Arial 8 bold italic"). grid (row = 5, column = x,
sticky = 'E', padx = 5, pady = 2)
                        canvas = tkinter.Canvas (step, bg = "red", height = "100", width = "100")
                        canvas.grid (row = 9, column = 3, sticky = 'E', padx = 0, pady = 0)
                        image = f "{i [9]}"
                        img = (Image.open ("" + image))
                        resized_image = img.resize (150, 150), Image.ANTIALIAS)
                        new_image = ImageTk.PhotoImage (resized_image)
```

```
                label2 = tkinter.Label (canvas, image = new_image)
                label2.image = new_image
                label2.grid ()

                e1 = tkinter.Entry (step)
                e2 = tkinter.Entry (step)
                e3 = tkinter.Entry (step)
                e4 = tkinter.Entry (step)
                e5 = tkinter.Entry (step)
                e6 = tkinter.Entry (step)
                e7 = tkinter.Entry (step)
                e1.insert (0, f "{i [0]}")
                e2.insert (0, f "{i [1]}")
                e3.insert (0, f "{i [2]}")
                e4.insert (0, f "{i [3]}")
                e5.insert (0, f "{i [4]}")
                e6.insert (0, f "{i [5]}")
                e7.insert (0, f "{i [7]}")

                e1.grid (row = 0, column = 1, columnspan = 7, sticky = "WE")
                e2.grid (row = 1, column = 1, columnspan = 7, sticky = "WE")
                e3.grid (row = 2, column = 1, columnspan = 7, sticky = "WE")
                e4.grid (row = 3, column = 1, columnspan = 7, sticky = "WE")
                e5.grid (row = 4, column = 1, columnspan = 7, sticky = "WE")
                e6.grid (row = 5, column = 1, columnspan = 7, sticky = "WE")
                e7.grid (row = 6, column = 1, columnspan = 7, sticky = "WE")
                rows + = 1

photoScroll = tkinter.Scrollbar (photoFrame, orient = tkinter.VERTICAL)
photoScroll.config (command = photoCanvas.yview)
photoCanvas.config (yscrollcommand = photoScroll.set)
photoScroll.grid (row = 0, column = 1, sticky = "ns")
```

```
canvasFrame.bind ("<Configure>", update_scrollregion)

# we insert the scrollbar into the canvas we created earlier

create_detail_frame ()
```

After analyzing the previous code for graphical presentation of employees, where each online worker is presented in a frame with all the data, we come to the conclusion that this type of information review brings the best results because it is reviewed and easy to use.

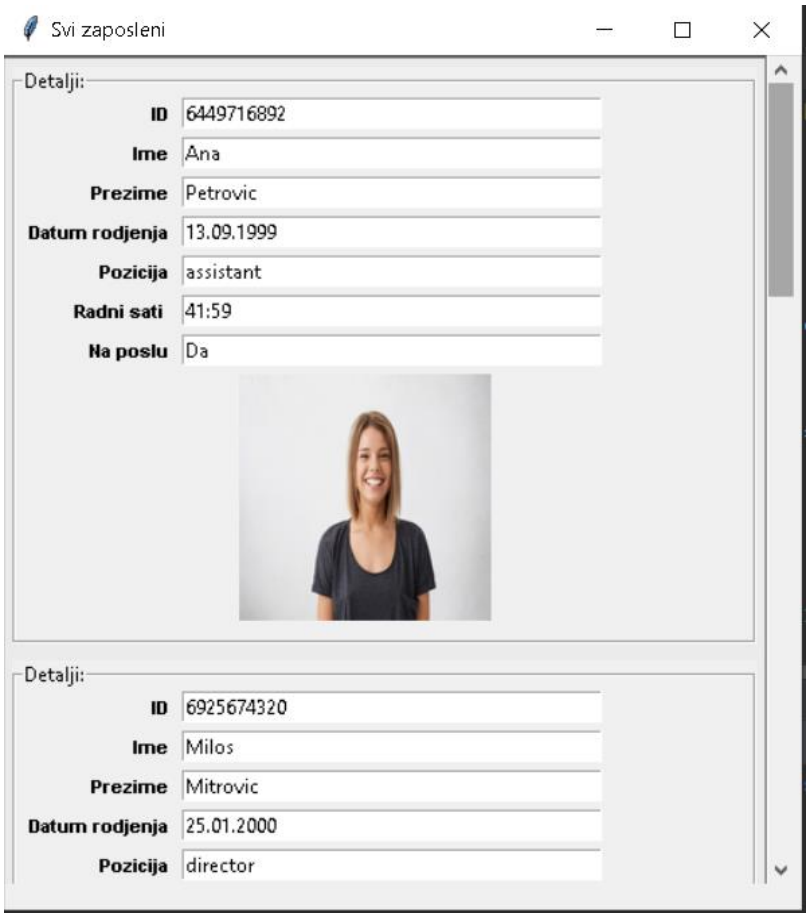

Figure 2.2.2.3 List of employees

If we look at the previous photo 2.2.2.3, we can see that for each worker there is a detailed frame with all the data, starting from the basic identification number to whether the employee is at work. Another way for the administrator to see if an employee is present or not.

3. MAKING PROCEDURE

The development process is a description of all the ideas and problems when creating software. Specifically in the case of Employee Identification Software, the development process is a complex process consisting of specially selected segments where each has its own part of the implementation. The process of making by the procedures can be seen in the extension, with detailed descriptions.

3.1. Software idea

The idea itself arose as a need of today's companies to have insight into information about workers and to manage them more easily. Employee identification software was created due to the idea that there is a global system to which workers log in and out. The software requires constant communication between the server and the client and also requires the existence of a local area network (LAN). Without the server, the software would not work because all the information is written to the database via the server, which is for security reasons. Only superiors have access to the server, while on the other hand, all employees can access the client. When we talk about the client, it is nothing but an ordinary device (or apparatus) that physically exists in front of the door with which workers read their bar code numbers or cards.

3.2. Realization of an idea

The realization of the idea consisted of a few segments and original questions to which a simple answer should be given while at the same time meeting all the necessary criteria of such software. Realization of the idea in stages and phases:

1) Interface prototype design
2) Creating an accessible interface based on a prototype
3) The idea of a strong connection between client and server
4) Testing and solving all communication problems
5) Optimization of data flow over the network for easier entry into the database
6) Testing of all software capabilities and conducting stress tests
7) Add new options, maintain and update the system

1) Interface prototype design

The prototype of the interface, ie the idea of the software layout, looked exactly the same from the beginning of the implementation to the final part, without any major changes and design. The idea is that the interface is accessible and easy to use, which is especially important when hiring an employee as an administrator, which means that no special training of that staff would be needed. This contributes to easier software management and easier access to it.

2) Creating an accessible interface based on a prototype

Merging the idea into the layout and later the entire software was an easier part of making it, as the software is easy to use, as we mentioned earlier. When creating the interface and the software itself, I spent a lot of time on the colors of the texts and controls, as well as the fonts and sizes of the controls. Such an approach to software development, where each detail is created separately and each segment is also gradually created, is in itself promising.

3) The idea of a strong connection between client and server

When we talk about the connection between the server and the client, the most ideal software environment would be the existence of a local network in which data would be exchanged, and in addition a server with large space and fast network flow and of course adequate professional staff. These three features make the ideal software and the ideal environment for seamless and accurate data exchange. In principle, it is very important to always anticipate all possible software needs as well as leave space and ways to improve in the future.

4) Testing and solving all communication problems

When communicating between two computers or a server and a client, one should always anticipate some of the possible errors and the sequence of events, especially when it comes to software that uses the network as a permanent communication tool. It is really inevitable that there are errors in communication and that is why it is necessary to anticipate them and then solve them or simply set the conditions for their events.

The first problem I encountered while implementing the library **socket** (Socket, network communication) in the software was a constant need for the server to receive data, although it must always be ready for other actions such as searching, deleting and adding an employee. So, in essence, the administrator of the software (server) should not be hindered in performing these operations in case the server receives connections and data from the client - in layman's terms, receiving data should not interfere with his work. The solution to this was to use the option **nor** in the programming language **python (**eng. Python), this way the software can perform multiple actions at the same time without those actions depending on each other. In principle, this relationship between functions and actions is independent and it is actually possible to create countless threads that would do their part of the job regardless of the work of the program. This could be seen as running some program algorithm in the background of the program while on the surface everything is still the same and ready for any work. The functions provided by threads are extremely important and of special importance and should be used especially during network operations because it is important that the socket itself be independent.

We mentioned the English word **socket.** A socket is not just a network communication but means a pair of ip addresses and ports, ie two-way communication between two computers on the network, is more detailed in Figure 3.2.1.

Figure 3.2.1 Socket address format

In our case of software and testing itself, the server uses the IP address of the local host **127.0.0.1.** which represents **local** the address of each computer. This address can be used only in the case of our computer, and on the other hand, in the case of commercializing the software, it would be necessary to provide a static IP address that would be a "hosting address" (e.g.**68.148.05.15**) and through which all internet traffic would be performed.

5) Optimization of data flow over the network for easier entry into the database

Another key thing in creating software, which uses network communication as a writing tool, is to optimize data transmission and writing. Imagine a database of over 10,000 workers in which at least 90% of that number enrolls and leaves work every day. First and foremost, a huge database would be needed to store this information, and constant use of the network without traffic congestion would also be needed. This amount of flow requires error-free flow and no additional traffic problems because it is already burdensome in itself. For this reason, developers around the world are devising and designing the best ways to write and read data from a database without burdening it.

In the specific case of Employee Identification Software, there was no traffic congestion problem nor was there a memory type problem because simply the software was not used for a large number of people nor was it planned for such a large thing. But certainly, I have applied to the software some of the types of optimizations that are generally known and which every developer should adhere to in order not to create additional problems in the operation of the software.

Optimization and a good way of writing codes are done in several ways, and only some of them are:

- Use of adequate data types (eg data type integer for number entry purposes and not string data type or text for number use purposes);
- Enter only the necessary data, without burdening the database with unnecessary typing and printing that further consumes memory and space;
- Enable the server to have enough space for data storage, ie predict the scope of the database and prepare all possible scenarios based on that;
- Like many other features, you need to pay attention to every part of the code.

6) Testing of all software capabilities and conducting stress tests

In addition to small tests of all functions during the development of the program, which each programmer does during implementation, it is necessary to perform some heavier load or stress tests. The so-called stress tests are performed at the final stage of software development, when a lot of functions have already been done and when the program is simply ready for commercialization and commissioning. There is usually a team that performs these stress tests and they are based on a complete test of all the functions and capabilities of the software in all possible ways, which is what the name of this test says. In many cases, hidden bugs are found at this stage of testing, which are later returned to the developers to fix.

In our case of Employee Identification Software, the stress test was conducted in the end but only by me personally. I did not encounter any errors during such testing, and that is probably because I have already tested the program well when writing code and programming. But it is certainly important to implement this step because it can always give some feedback that can help in solving certain problems.

7) Add new options, maintain and update the system

Like any project in any field of work, it is important to leave space for improving and adding functions as well as maintaining the same. When it comes to Employee Identification Software, I think there is a huge amount of room for improvement and adding opportunities. The reason for this is that when it comes to this type of software, which helps to manage the company, it is always possible to supplement and create new additional options. Some of these ideas are:

- Creating video surveillance and implementing monitoring systems in the software itself Creating
- space for new data of each employee that would further improve the convenience Creating
- opportunities for direct printing of cards with employee data
- Since we already use network communication, the implementation of a communication system between administrators, or some correspondence system
- Creating an administrator user system, where each administrator has their own password and username with which to log in to the system
- Option to calculate working hours for a specific month or period of working
- days Ability for users to check their working hours at the login terminal
- Storing all data currently entered in log.tht in a database intended only for that purpose and based on that to have access to all data on logins and logouts
- as well as many other options.

These are just some of the options that are actually ideas, while on the other hand I have a lot of other ideas that would improve this software and take it to the next level. I also believe that every software in the world has room to improve and add new features.

3.3. Problems encountered during implementation

As problems in code and implementation are an inevitable part of every project, so in my case there were a couple of problems. Some were smaller and harmless which I managed to solve very quickly while a couple of problems required a longer solution. One such problem was that when restarting the server, the login time of the workers was always reset. This type of problem is something that should always be kept in mind and the software should always be prepared in advance for such cases, because there can always be a malfunction and a server restart. In this case, I solved the problem by instead of sending the initial time of the worker directly to the server, I entered that time in the database and during the later logout of the worker, I read the same time and thus worked working hours. The solution was really simple and didn't take too long to implement.

It is also important to note that the old start time is always "trampled" and changed with the new start time, so that the base is not loaded. With this system, I have provided dynamic writing without loading the server and the database in general, which is of great importance.

4. PRACTICAL APPLICATION OF THE SOFTWARE

When we talk about the practical application of one software in the world, its breadth can be great, since software is made today as a solution and answer to all kinds of problems. In our case of Employee Identification Software, the use can indeed be wide. The reason for this is that we can always find a company that needs this type of maintenance software, targeting smaller companies or firms because these larger ones probably already have such software created. This type of software is especially desirable in companies that manage big data, and based on that request, their work can be made easier. Employee identification software contains several useful functions that allow you to easily manage a smaller database and system. It has a system for tracking workers, ie their arrival and departure from the company, a system for adding, deleting and searching employees as well as a system for checking employees who are currently at work. These few systems implemented in the software enable quick and easy management of the software, and for that reason it is in itself accessible and amenable to commercialization.

3.1. Method of use

The method or manner of using employee identification software is broad. It can be used for several purposes and the goal of each purpose is the successful management of one organization.

To use this software from the administrator's side, no special training of professional staff is required as the software is easy and accessible to use, at least in this current version (see 0.1). Employee identification software is easy to use in itself because it is graphically presented so that everyone can understand the functions and the system itself. Also, all this applies to users or employees of the company who need to use this system every day and use it to log in and out. Check-in and check-out functions are based on the same principle, so their use is understandable and fairly easy to use. By accessing the device, the worker reads his bar code with two clicks and in that way secures for himself the entry or exit from the company.

Table 2.1 - Overview of roles in the system

	The role	Degree of hierarchy	Authorization
Staff	Login and logout	I	No authorization
Administrator	System and database management data	II	Full authority over the system

5. CONCLUSION

Using all the above technologies and taking into account all the problems that arose during the implementation, it could easily be said that the software for employee identification has all the predispositions to be commercialized in every sense of the word. The main technology used is python (eng.**Python**), with which the whole interface was created and the whole code was written. The project itself is sufficiently accessible and easy to use, so its use is practically for all staff, such as human resources staff. The experience gained during the development of this software is significant because it has greatly improved my knowledge and understanding of the technologies used, so this project can also be used as part of a portfolio or reference for future work. In addition, the same technologies are in great demand in today's IT world, so this project is a big plus and advantage in itself.

I believe that continuous improvement of skills is the best way to achieve success, so there is room in this software for further improvement and advancement so that it would be perfect.

- **INDEX OF TERMS**

A

B

Z

I

K

M

P

R

S

F

- # LITERATURE

[1] Python Software Foundation, https://docs.python.org/3/ , retrieved: November 2021

[2] SQLite Open-Source, https://www.sqlite.org/index.html , retrieved: November 2021

[3] socket - Low-level networking interface, https://docs.python.org/3/library/socket.html , retrieved: November 2021

[4] tkinter - Python interface to Tcl / Tk, https://docs.python.org/3/library/tkinter.html , retrieved: November 2021

- **CONTRIBUTIONS**

- **STATEMENT OF ACADEMIC HONORABILITY**

STATEMENT OF ACADEMIC HONORABILITY

Student (name, name one parent and surname):	Nikola Radiša Rilak
Index number:	BAT - 73/18

Under full moral, material, disciplinary and criminal responsibility, I declare that the final work, entitled:

• result of own research work;
• that this work, neither in whole nor in parts, has not been reported to other higher education institutions;
• that I have not infringed the copyrights or misused the intellectual property of others;
• that I have indicated or cited the work and opinions of other authors that I have used in this paper in accordance with the Instruction;
• that all papers and opinions of other authors are listed in the bibliography / references which is an integral part of this paper, listed in accordance with the Instructions;
• that I am aware that plagiarism is the use of other people's work in any form (such as quotations, paraphrases, images, tables, diagrams, designs, plans, photographs, film, music, formulas, websites, computer programs, etc.) without citation author or presentation of other people's copyrighted works as mine, punishable by law (Law on Copyright and Related Rights), as well as other laws and relevant acts of the College of Electrical Engineering and Computing Vocational Studies in Belgrade;
• that the electronic version of this paper is identical to the printed copy of this paper and that I agree to its publication under the conditions prescribed by the acts of the College of Electrical Engineering and Computing Vocational Studies in Belgrade;
• that I am aware of the consequences if this work is proven to be plagiarism.

In Belgrade, __. __. 201_. years

Student's handwritten signature

_____