

## RouteYOU – app design to support outdoor achievement recording

Szymon Węgliński <sup>1</sup>, Dawid Kotrys <sup>2</sup>

<sup>1</sup> student of Geoinformatics, Faculty of Geodesy and Cartography, Warsaw University of Technology, plac Politechniki 1, Warsaw 00-661, Poland, [szymon.weglinski.stud@pw.edu.pl](mailto:szymon.weglinski.stud@pw.edu.pl)

<sup>2</sup> PhD, Department of Mathematics, Faculty of Mechanical Engineering and Computer Science, University of Bielsko-Biala, Willowa 1, Bielsko-Biala 43-300, Poland, [dkotrys@ubb.edu.pl](mailto:dkotrys@ubb.edu.pl)

**Abstract:** The article is devoted to the process of creating a map application, whose purpose is to provide organizational support for the School Club of the Polish Tatra Society "Pionowy Świat" at the 5<sup>th</sup> High School in Bielsko-Biała. However, the program has been generalized for all types of outdoor activities where the spatial location of stops and characteristic points of the route are important. Additionally, the application allows for the generation of rankings of the most active users and the most visited places, with the option to export this data to HTML files. There is also broad potential for further development, which will allow continuous improvements and modifications to ensure that the provided functionalities are as accessible and useful as possible for the target group of users.

**Keywords:** geoinformatics, python, tkinter library, tkintermapview library, desktop application, RouteYOU, outdoor activities, tourism, HTML;

## RouteYOU – projekt aplikacji wspierającej rejestrowanie osiągnięć na świeżym powietrzu

Szymon Węgliński <sup>1</sup>, Dawid Kotrys <sup>2</sup>

<sup>1</sup> student Geoinformatyki, Wydział Geodezji i Kartografii, Politechnika Warszawska, plac Politechniki 1, Warszawa 00-661, Polska, [szymon.weglinski.stud@pw.edu.pl](mailto:szymon.weglinski.stud@pw.edu.pl)

<sup>2</sup> doktor, Katedra Matematyki, Wydział Budowy Maszyn i Informatyki, Uniwersytet Bielsko-Bialski, Willowa 1, Bielsko-Biała 43-300, Polska, [dkotrys@ubb.edu.pl](mailto:dkotrys@ubb.edu.pl)

**Streszczenie:** Artykuł jest poświęcony procesowi tworzenia aplikacji mapowej, której głównym celem jest wsparcie organizacyjne Szkolnego Koła Polskiego Towarzystwa Tatrzańskiego „Pionowy Świat” przy V Liceum Ogólnokształcącym w Bielsku-Białej. Program został jednak uogólniony dla wszelkich typów aktywności zewnętrznych, w których istotne jest położenie przestrzenne przystanków oraz punktów charakterystycznych trasy. Oprócz tego aplikacja pozwala prowadzić ranking najaktywniejszych użytkowników, najczęściej odwiedzanych miejsc oraz dane te eksportować do plików HTML. Istotnym jest również szerokie pole rozwoju, które pozwoli nieustannie ulepszać oraz modyfikować program, aby dostarczane funkcjonalności były możliwie najbardziej przystępne oraz przydatne dla docelowej grupy użytkowników.

**Słowa kluczowe:** geoinformatyka, python, biblioteka tkinter, bibliotek tkintermapview, aplikacja komputerowa, RouteYOU, aktywności na świeżym powietrzu, turystyka, HTML;

## 1. Introduction

Clive Humby once said: “Data is the new oil. It’s valuable, but if unrefined it cannot really be used”. While this statement applies to all data, it holds even greater significance when it comes to spatial data. This type of information is unquestionably valued nowadays. However, its wide research spectrum and vast range of potential applications require careful organization to effectively extract the data we are interested in. For this reason, created programs should respond to user needs, which often focus on diverse aspects and require tailored approaches.

This perspective, along with the insufficient number of functions provided by commercial maps, highlighted the need to create an application whose primary goal is to automate the process of managing individual routes and transferring manually collected data to a digital medium. That is why I decided to create the foundation for a map application whose functionalities sufficiently accelerate the process of managing achievements and maintaining lists of the most active users and visited points. At the same time, since this is my first application, I implemented many simplifications during its development, which I plan to replace in the future with specialized tools and methods based on advanced geoinformation techniques, the specifics and use of which I will hopefully learn during my engineering studies.

## 2. App description

### a. Programming decisions

I began developing the application by making key programming decisions. I selected Python (*version 3.12.6*) as my language of choice. This decision was driven due to Python’s extensive use in geosciences and its rich collection of scientific and user interface libraries, which offer many useful functions. Another advantage of Python is its readability, which simplifies both writing and modifying parts of the code [1].

Given that this is my first venture into more advanced applications, I also made a design decision regarding code organization. I decided to program the entire application in a single file to avoid complications with file management. However, in the future, following current trends in application development [2], I plan to split the code into thematic files or even folders to improve its readability and organization. This will not only make future work easier but also reduce the potential failure rate of individual application components.

### b. Python libraries

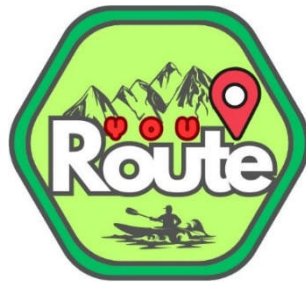
As I mentioned, Python’s strength lies in its multitude of libraries, which are generally well-documented and tested. The final version of my application uses the following libraries:

- **Tkinter** – a standard Python library for creating graphical user interfaces (GUI). In my program I use it for enabling the application to respond to user requests [3],
- **datetime** – handles operations involving dates and times. In this case, it calculates the difference between two given dates,
- **Pillow** – an image processing library. In my program it is responsible for displaying PNG files associated with the application’s logo,
- **folium** – a library for creating interactive maps. It allows spatial maps to be generated and saved as HTML files.
- **TkinterMapView** – an extension of Tkinter that adds interactive map functionality directly within the application’s window. It also helps to display points and routes selected by the user [4],
- **geopy.distance** – part of the geopy library. Used for calculating the distance between two geographic points based on their latitude and longitude,
- **json** – provides tools for working with JSON data. This is a temporary solution for storing the application’s state,
- **os** – a standard library for interacting with the operating system. It is used to manage and manipulate file paths,
- **webbrowser** – allows the program to open URLs in the default web browser. In my application, this is useful for collecting user-submitted error reports.

### c. Logo and name

The name of my program consists of two combined components. “Route” reflects the nature of the application and its connection to geopositioning, while “YOU” (written in capital letters) emphasizes a strong focus on individual route personalization.

I placed similar emphasis on the application’s logo, which, along with the name, highlights the interdisciplinary potential of the application. The graphic file was created using the features provided by **Canva** [5].



**Figure 1.** “RouteYOU” – application logo

### 3. From project to realization

As mentioned, a graphical user interface has been created using Tkinter library. The window configuration is as follows:

```
# Import of necessary libraries and color initialization

window = tk.Tk() # Window initialization
window.title('RouteYOU') # Setting the window title
window.geometry('1200x600') # Setting a fixed window size
window.config(bg='lightlime', bd=0) # Setting background color and border width
window.resizable(False, False) # Disabling window resizing for proper elements scaling

# Application code
window.mainloop() # Running the Tkinter event loop
```

#### a. Interface sections

To improve GUI and separate the most important parts of the program, I used the canvas from the Tkinter library and I created a special function for lists with a scrollbar. The code is shown below:

```
def scrollbar_listbox(root, title, width=23, height=5, fg='black', bg='white',
                    relx=0.5, rely=0.5, anchor='center', x=0, y=0, font_th=12):
    # Initializing and positioning a new frame within the window
    frame = tk.Frame(root)
    # Setting the frame's position in the window
    frame.place(relx=relx, rely=rely, anchor=anchor, x=x, y=y)

    # Creating and adding a title label
    list_title = tk.Label(frame, text=title, font=('Arial', font_th, 'bold'), fg=fg)
    # Positioning the title label at the frame's top
    list_title.pack(side=tk.TOP, pady=5)

    # Creating a listbox for displaying content
    listbox = tk.Listbox(frame, width=width, height=height, fg=fg, bg=bg)
    # Positioning the listbox on the frame's left
    listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

    # Creating a vertical scrollbar linked to the listbox
    scrollbar = tk.Scrollbar(frame, orient=tk.VERTICAL, command=listbox.yview)
    # Positioning the scrollbar on the frame's right
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

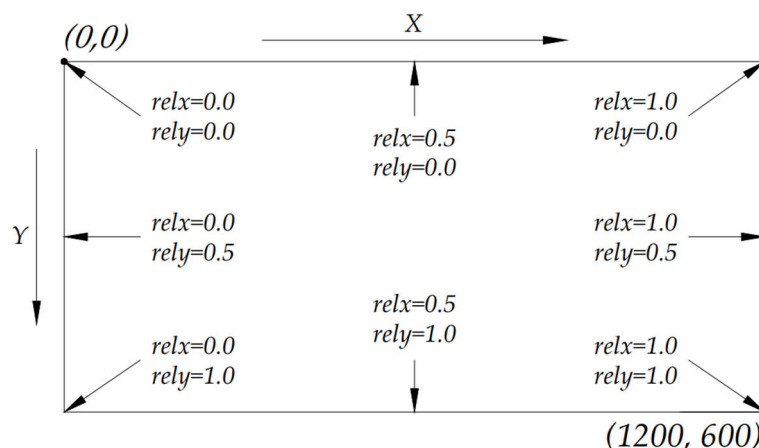
    # Updating the scrollbar with the listbox scrolling
    listbox.config(yscrollcommand=scrollbar.set)

    return listbox, list_title
```

The interface was enhanced with buttons, all of which were added based on the following scheme:

```
button_example = tk.Button(window, text='Example', bg='color',
                           activebackground='color', command=function, width=10, height=1)
button_example.place(relx=1.0, rely=1.0, anchor='se', x=±shift, y=±shift)
```

It is important to note that the positioning of window elements uses relative coordinates. The method for defining them is illustrated in Figure 2, where the rectangle represents the application window area.



**Figure 2.** Interpretation of element positioning within the window in the Tkinter library (figure made in: **AutoCAD 2023**)

The window is complemented by the previously mentioned canvas areas and a map embedded within the window using the TkinterMapView library. In addition, the basic route statistics are located in the lower part of the central area. In my program, this is represented as the code below:

```
# RIGHT SIDE
# Creating background on the right side
canvas_right = tk.Canvas(window, width=180, height=600, bg=muddygreen, highlightthickness=0)
# Attaching the background to the right side of the window
canvas_right.place(relx=1.0, rely=0.5, anchor='e')
# Creating a scrollbar listbox for route points
listbox_route_points, label_route_points = scrollbar_listbox(window, 'ROUTE POINTS',
                                                             fg='green', relx=1.0, rely=0.5, height=15, anchor='e', x=-10, y=-10)
# Opening the logo image
logo = Image.open(r'C:\Users\HP\Desktop\RouteYOU\RouteYOU_logo.png')
# Resizing the image to a fixed size
logo = logo.resize((128, 128), Image.LANCZOS)
# Converting the image to a format compatible with the tkinter library
app_logo = ImageTk.PhotoImage(logo)
# Setting the background for the logo
app_logo_label = tk.Label(window, image=app_logo, bd=0, bg=muddygreen)
# Attaching the logo to the correct position in the window
app_logo_label.place(relx=1.0, rely=0.0, anchor='ne', x=-25, y=+10)

# LEFT SIDE
# Creating background on the left side
canvas_left = tk.Canvas(window, width=180, height=600, bg=muddygreen, highlightthickness=0)
# Attaching the background to the left side of the window
canvas_left.place(relx=0.0, rely=0.5, anchor='w')
# Creating scrollbar listboxes for most active users, most visited places and future
# functionalities
listbox_ranking, label_ranking = scrollbar_listbox(window, 'MOST ACTIVE', fg='red', height=10,
                                                    relx=0.0, rely=0.0, anchor='nw', x=+10, y=+10)
listbox_visited, label_visited = scrollbar_listbox(window, 'MOST VISITED', fg='darkorange',
                                                    height=5, relx=0.0, rely=0.5, anchor='sw', x=+10, y=+35)
listbox_, label_unnamed = scrollbar_listbox(window, '- - -', fg='darkblue', relx=0.0, rely=0.8,
                                             anchor='sw', x=+10, y=-20)
```

```

# CENTER
# Creating the background canvas for the map
canvas_center = tk.Canvas(window, width=750, height=510, bg='grey', highlightthickness=1,
    highlightbackground='darkgrey')
# Attaching background to the top part of the central area
Canvas_center.place(relx=0.5, rely=0.0, anchor='n', y=+9)
# Creating the map widget using the tkintermapview library
map_widget = TkinterMapView(window, width=750, height=510, corner_radius=0)
# Attaching map to the correct position
map_widget.place(relx=0.5, rely=0.0, anchor='n', y=+10)
# Setting the initial map view parameters
map_widget.set_position(49.7678810, 19.1610286) # Coordinates of Czupel (Little Beskids)
map_widget.set_zoom(17) #
# Adding an option to add an individual marker not connected to route points scrollbar
map_widget.add_right_click_menu_command(label='New tag', command=add_marker_event,
    pass_coords=True)

# Creating a canvas for statistics
canvas_stats = tk.Canvas(window, width=840, height=70, bg='darkgreen', highlightthickness=0)
# Pinning to lower part of central area
canvas_stats.place(relx=0.5, rely=1.0, anchor='s')
# Updating statistics with implemented function for refreshing and displaying component elements
update_statistics(distance=global_distance, duration=global_duration, speed=global_speed)

```

TkinterMapView uses OpenStreetMap as its representation of topography and as an underlay for placing markers and lines. One of its key benefits is that it is free to use and easy to implement, along with straightforward options for customization. However, this is a temporary solution in my application due to the limited functionalities provided by the library. In the future, I want to utilize data from [geoportal.gov.pl](https://geoportal.gov.pl) [6] and ArcGIS software [7][8]. In addition to using dedicated systems for spatial information, these tools will enable more effective statistical analysis and data modeling, while also improving the visual quality of the interface.

## b. Memory management

In “RouteYOU”, memory is stored as JSON and saved when the application closes. When reopened, data from the same file is retrieved, and then the program recreates the last stage displayed on the user’s screen. This concept is implemented with two simple functions described below.

```

# Import of necessary libraries, initialization of global variables and rest of the code

def save_state():
    # Creating a data instance
    state = {
        # Saving status of the tables and last-set language
        'people_list': people_list,
        'visited_points': visited_points,
        'current_language': current_language,
        # Saving the center of displayed map as the last map position
        'map_position': {
            'latitude': map_widget.get_position()[0],
            'longitude': map_widget.get_position()[1]
        }
    }
    # Opening in write mode to save data to a JSON file
    with open('app_state.json', 'w') as f:
        json.dump(state, f, indent=4)

```

```

def load_state():
    global people_list, visited_points, current_language
    # Checking if the file exists
    if os.path.exists('app_state.json'):
        # Opening the file in read mode
        with open('app_state.json', 'r') as f:
            try:
                # Reading the contents of the JSON file
                state = json.load(f)
                people_list = state.get('people_list', [])
                visited_points = state.get('visited_points', [])
                current_language = state.get('current_language', 'EN')
                map_position = state.get('map_position', {'latitude': 49.767881, 'longitude':
19.161029})
                # Setting the map position as the last displayed
                map_widget.set_position(map_position['latitude'], map_position['longitude'])
                # Handling incorrect JSON structure by setting base values for each dataset
            except json.JSONDecodeError:
                people_list = []
                visited_points = []
                current_language = 'EN'
                map_widget.set_position(49.767881, 19.161029)
    # Setting empty or base values if file is not found
    else:
        people_list = []
        visited_points = []
        current_language = 'EN'
        map_widget.set_position(49.7678810, 19.1610286)

    # Refreshing scrollbar listboxes and language status for correct display
    update_people()
    update_visited()
    relaunch_language()

# Rest of the code

```

In the current version, I chose the JSON format because it is a lightweight data interchange format that excels in data serialization due to its simplicity and widespread support. Another benefit is its ease of exchange between different programming languages [9]. In the future, I plan to create a unique and dedicated database for my application that will connect components. However, for now, this solution is more than sufficient.

### c. Delivered functionalities in window interface

The interface was created using snippets of the code described in section a., with the final version shown in Figure 3.

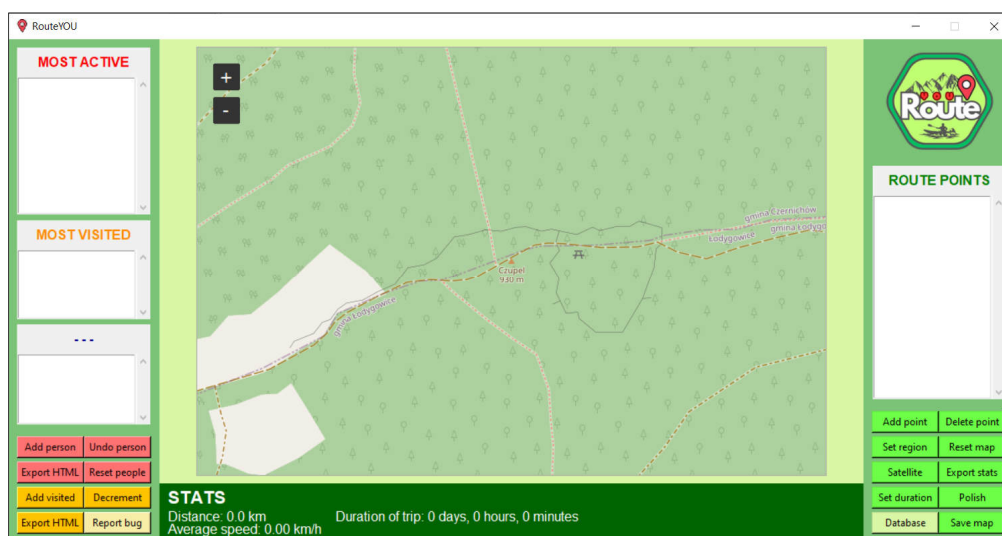


Figure 3. Interface of an application

Now, the functions represented by buttons in the interface will be examined closely. However, to keep the content concise, I will aim to avoid including long code snippets in the descriptive sections.

#### i. Language button

Since the target audience of the application is Polish, the program supports language switching (Figure 4). To do this, simply press the “Polish” button if the language is set to English, or “Angielski” if the language is set to Polish. In addition to button labels, windows and messages also change displayed language. However, this is implemented slightly differently. The names of buttons and table labels are replaced using the following example function:

```
# Rest of the code

def relaunch_language():
    if current_language == 'PL':
        button_language.config(text='Angielski')
        # Rest of the buttons
    else:
        button_language.config(text='Polish')
        # Rest of the buttons

# Rest of the code
```

The switching is implemented by the code below.

```
def switch_language():
    global current_language # Using global variable for language
    if current_language == 'EN':
        current_language = 'PL'
    else:
        current_language = 'EN'
    relaunch_language()
```

Whereas, the windows and messages are changed using simple conditional statements, which adjust the window’s title and content depending on the selected language. An example of this can be found below.

```
# Needed parts of the code

title = 'Warning' if current_language == 'EN' else 'Uwaga'
message = 'Table has been cleared.' if current_language == 'EN' else 'Tabela wyczyszczona.'
messagebox.showinfo(title, message)

# Rest of the code
```

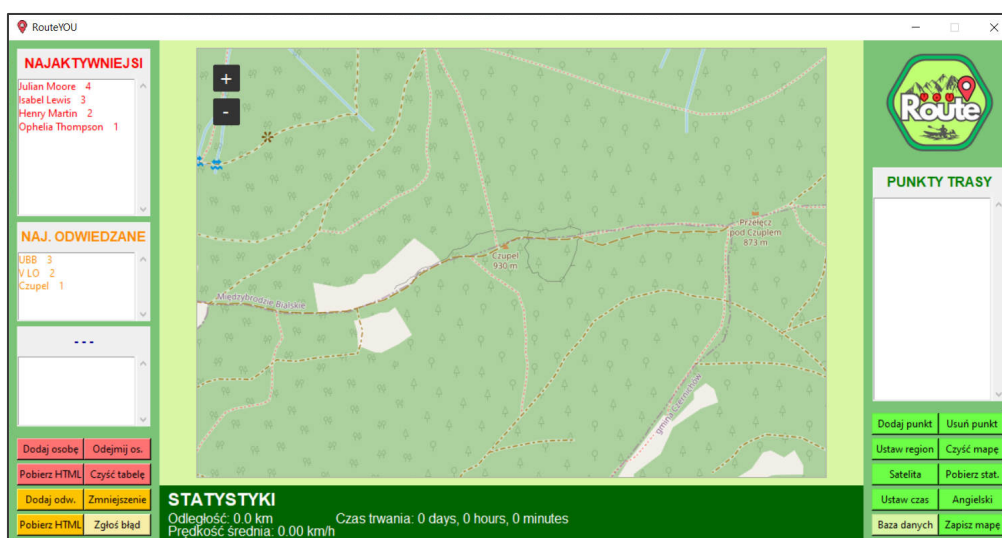


Figure 4. Polish version of interface

## ii. “Add point” and “Delete point” buttons

As the names of these buttons suggest, the first is used to add a new point, while the other one is used to delete the selected point, if it exists. The procedure for adding a point consists of three steps:

1. Click the button
2. Enter the name in the appropriate subwindow (if this step is skipped, the name will default to “Unnamed point”, with each subsequent unnamed point receiving a sequential identifier starting from 1) (Figure 5)
3. Click on the map to place a marker with the user-provided name.

Each new point creates a marker with a name and a polyline connecting the two most recently added points with a blue segment. Based on this, the Euclidean distance [10] is calculated using geopy.distance library [11]. This method is beneficial when adding route point for water activities, but adding on-earth trails requires a denser distribution of points to accurately approximate the distance travelled. Each newly added (or removed) point updates the total distance (Figure 6), which is stored as a global variable, along with other statistics.

Deleting a point involves selecting it from the “Route Points” list and then clicking the appropriate button. This action will remove the corresponding marker, connect the two adjacent points with a line segment, and recalculate the distance. It’s important to note, that if we attempt to delete a point when only one segment (two points) remains on the map – both points will be deleted.

The function below is responsible for calculating the distance using geodetic coordinates.

```
# Rest of the code (with initialization of global variable for distance)

def calculate_polyline_length(coords):
    total_distance = 0
    for i in range(len(coords) - 1):
        total_distance += geodesic(coords[i], coords[i + 1]).kilometers
    return total_distance

# Rest of the code
```

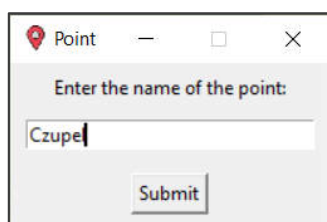


Figure 5. Subwindow for adding a new point

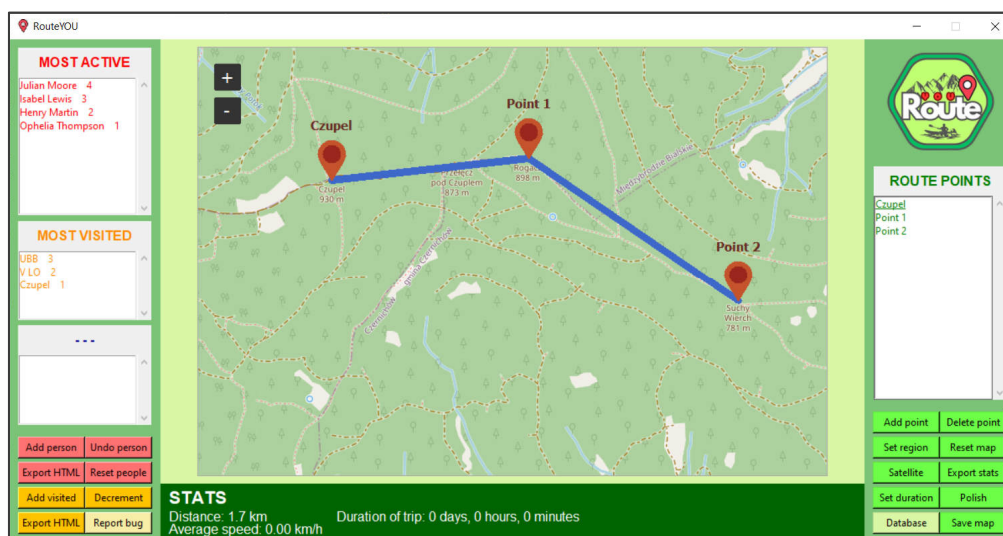


Figure 6. Adding new points and recalculating the distance after each



There are plans to create a pathfinding algorithm on the map that will enable finding a route along existing paths between two points, making it easier to add stops on land routes.

### iii. Other buttons connected with map management

- **Set region** – clicking this button opens a combo box (Figure 7) that allows the user to select a region from an alphabetical list. These regions are stored in a global array that holds both the name and coordinates to move to, as shown in the example below. This feature is designed to simplify map navigation for the user, and the straightforward storage format makes it easy for programmer to add new regions.

```
# Rest of the code

regions = {
  'Tatry': (49.250954, 19.934102),
  # (...)
}

# Rest of the code
```

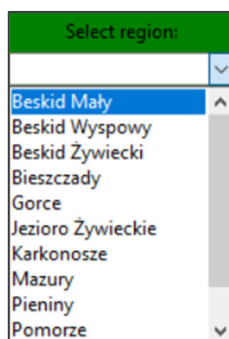


Figure 7. Combo box with regions

- **Reset map** – the function represented by this button, allows the user to clear “Route Points” table and remove all marker points with polylines from the map. The deletion of points is permanent and (in the current version of the application) cannot be undone.
- **Satellite/Map** – this button is used to change the base layer of the map. After clicking, it allows the user to switch between Google Maps Satellite and OpenStreetMap (Figure 8) while maintaining the current position. The function responsible for this part is shown below.

```
# Initialization of global variables and rest of the code

def toggle_map():
    global current_server # Global variable for storing current base layer
    if current_server == 'OpenStreetMap': # Switching to Satellite if Map
        map_widget.set_tile_server('https://mt0.google.com/vt/lyrs=s&hl=en&x={x}&y={y}&z={z}&s=Ga',
                                   max_zoom=19)
        current_server = 'Google Satellite'
        if current_language == 'EN': # Language management
            button_toggle_map.config(text='Map')
        else:
            button_toggle_map.config(text='Mapa')
    else: # Switching to Map if Satellite
        map_widget.set_tile_server('https://a.tile.openstreetmap.org/{z}/{x}/{y}.png')
        current_server = 'OpenStreetMap'
        if current_language == 'EN': # Language management
            button_toggle_map.config(text='Satellite')
        else:
            button_toggle_map.config(text='Satelita')

# Rest of the code
```

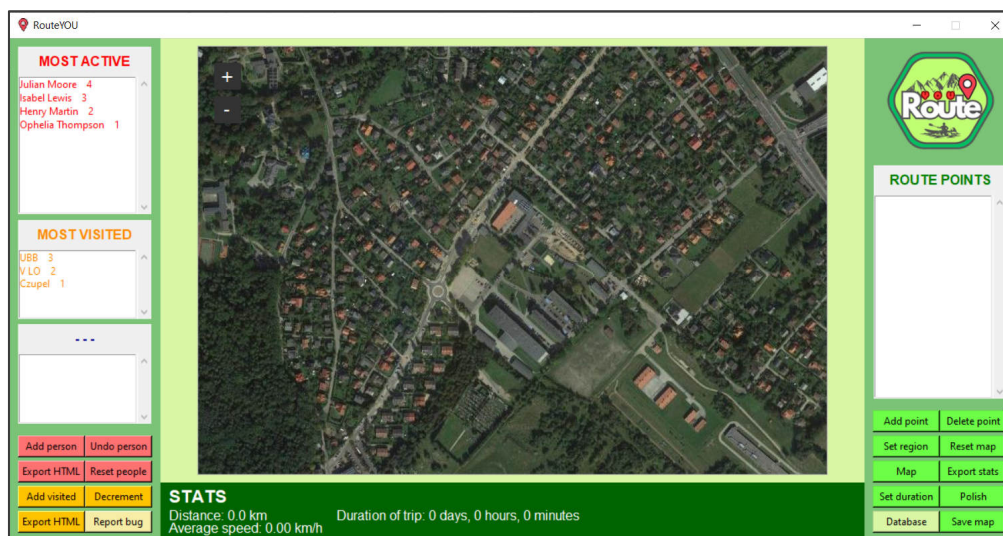


Figure 8. Interface with changed underlay (region: Campus of the University of Bielsko-Biala)

- **Save map** – this button allows user to save map, including markers and polylines, to HTML file in a user-chosen location with a user-chosen name. If current view is set to satellite, the button will save the map in its default (non-satellite) view instead.
- **Database** – in the current version of the program, this button only displays a simple message and is not connected to any useful functionalities. Future plans include the previously mentioned program database, which will allow for the analysis and storage of application data and modified map in one place.

#### iv. Operations on statistics

Currently, the statistics are quite basic, focused on building concept that will be expanded in future, more advanced versions. At present, the application calculates the distance of the added route (section ii. of this article) and allows the user to set the trip duration by pressing the “Set duration” button and filling in the required fields in a new subwindow (Figure 9). If incorrect data is entered, the code detects it and displays a message indicating that the date or time input is incorrect. If the provided data is valid, the statistics will be updated with the new duration and average speed (if a distance is set; otherwise, this part will remain at 0.00 km/h).

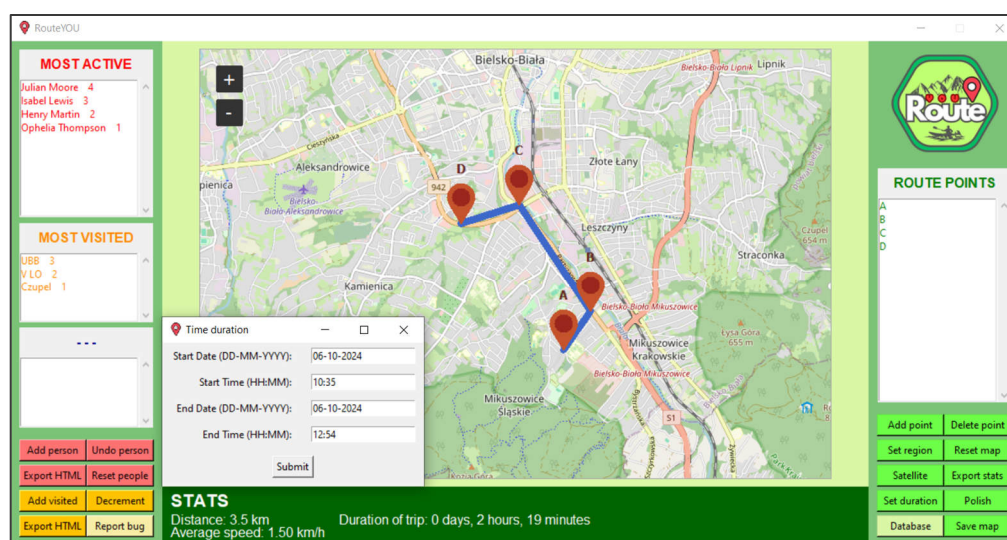


Figure 9. Interface with “Time duration” subwindow

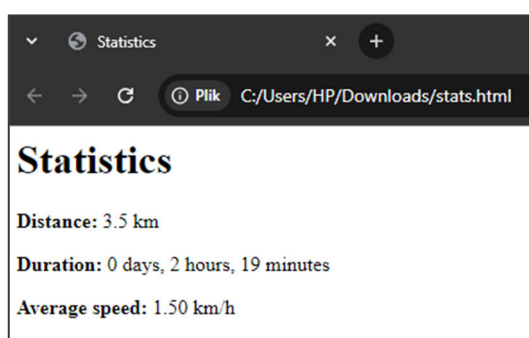
The “Export stats” button allows the user to download all statistics as simple HTML file. After clicking it, the code shown below is saved to the file specified by the user, and the result of opening the file in a browser window is shown in Figure 10. A subwindow confirms the successful saving and shows a path to the file.

```

<html>
  <head>
    <title>Statistics</title>
  </head>
  <body>
    <h1>{main_label}</h1>
    <p><strong>{distance_label}</strong> {global_distance:.1f} km</p>
    <p><strong>{duration_label}</strong> {global_duration}</p>
    <p><strong>{average_speed_label}</strong> {global_speed:.2f} km/h</p>
  </body>
</html>

```

where: `main_label`, `distance_label`, `duration_label` and `average_speed_label` store different content based on the current language set in the window at the moment the button is clicked.

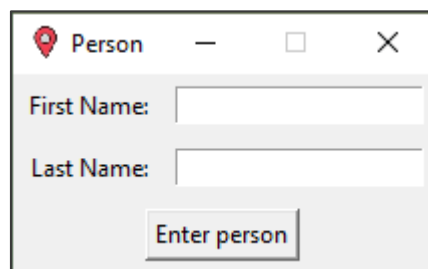


**Figure 10.** Content of HTML file with statistics

v. Scrollbar listbox: “Most active” table management

The “Most active” table is designed to store and rank the most active people, highlighting those who go out frequently. In the current version of the program, these individuals are not yet linked to the route, but in future, with the addition of a database, I plan to integrate these components of the application. For now, the table’s operation is managed using 4 buttons (in red, like the scrollbar listbox title). The table can be modified using:

- “Add person” button – initially, after pressing this button, the subwindow “Person” (Figure 11) opened. Then, the user could input the full name in two separated fields (First and last name). However, after consulting this functionality with the first user of the “RouteYOU” application – Sebastian Kulinski PhD (supervisor of SK PTT “Pionowy Świat”) – this concept was changed due to the tedious process of adding multiple people from one trip. The updated version first asks for the number of people and then opens exactly that many windows one after the other. This particularly solved faced difficulty, but in the future, I will add the possibility to import data from a file (if list of the users will be available as computer version).



**Figure 11.** Subwindow for adding people to “Most active” listbox

- “Undo person” button – allows the user to remove a specified number from the selected record (Figure 12). If the number is greater than what the person currently has, that row will be removed from the table. This button is useful for correcting trips that were unintentionally added to a person.

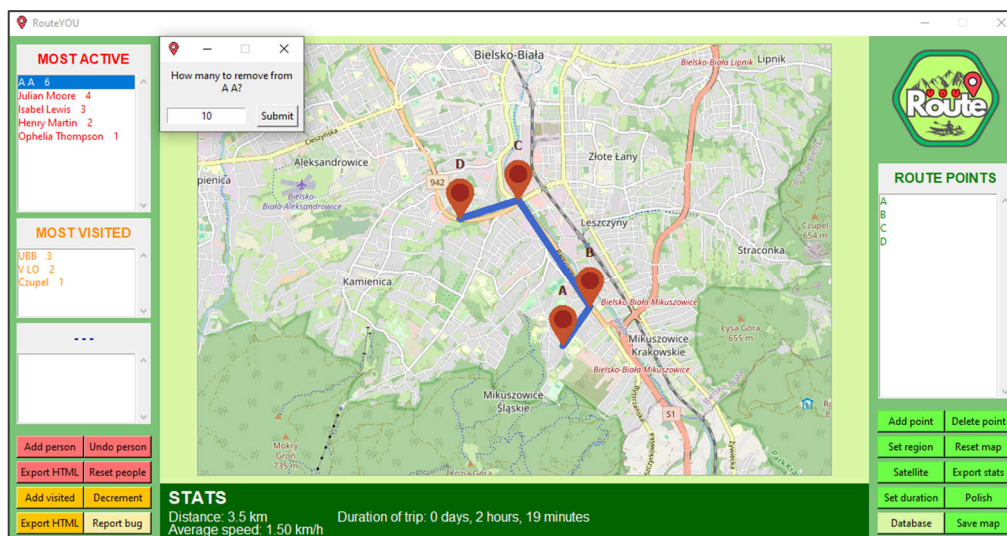


Figure 12. Removing trials from person

- “Reset people” button – removes all records from the listbox. This operation requires confirmation to make the user aware that it might destroy important data and affect many rows. It is useful, for example, when we want to create a new ranking at the start of a semester.
- “Export HTML” button – works similarly to the “Export stats” button (section iv. of this article). It creates a portion of HTML code that is saved in a user-specified file. After opening (Figure 13), a table will be displayed in which top three positions will be highlighted with the appropriate colors. This function works perfectly with the previous button, as we can export the table before resetting it and store it for future use.

Full name	Count
Julian Moore	4
Isabel Lewis	3
Noah Taylor	3
Henry Martin	2
Ophelia Thompson	1

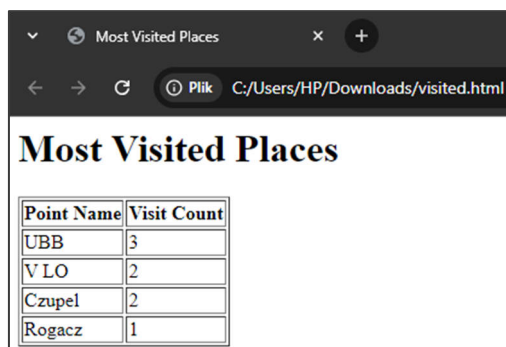
Figure 13. Exported “Most active” table (filled with random data)

#### vi. Scrollbar listbox: “Most visited” table management

In addition to the table collecting the most active individuals, the program also supports the ranking of the most frequently visited places (with buttons and table title in orange color). The operation scheme is quite similar to the “Most active” table, but there is no button to clear the entire table, as we generally want to keep the places we visit for as long as possible. However, if there is a need to add such a function, the ability to reset the table will be implemented in future versions. The listbox is managed in the interface through buttons:

- “Add visited” button – when clicked, adds the selected point from the “Route Points” table to the “Most visited” table or increments the existing row with the same name by 1,
- “Decrement” button – when clicked, decrements the selected record in the “Most visited” table by 1.

As before, there is also button “Export HTML” which allows exporting this data to an HTML file. The procedure is identical to that of the previous buttons of this type. The result of the operation associated with this button is shown in Figure 14. It is important to note that the first three ranking positions are not highlighted.



Point Name	Visit Count
UBB	3
V LO	2
Czupel	2
Rogacz	1

Figure 14. Exported “Most visited” table (filled with random data)

#### vii. Handling user-experienced bugs

The application interface is complemented by an “Report error” button. When clicked, it opens the default email client with a pre-filled recipient address and subject. To complete the report, the user needs to enter the message content and send the email. This feature is implemented using the webbrowser [12] library, as shown in the code below.

```
# Rest of the code

def report_bug():
    email_address = 'szweglinski@gmail.com' # Contact address
    if current_language == 'EN': # Language management
        subject = 'Bug Report'
        body = 'Please describe the bug you encountered here.'
    else:
        subject = 'Zgłoszenie o błędzie'
        body = 'Opisz błąd, którego doświadczyłeś podczas korzystania z aplikacji'
    mailto_link = f'mailto:{email_address}?subject={subject}&body={body}' # Creating link
    webbrowser.open(mailto_link)

# Rest of the code
```

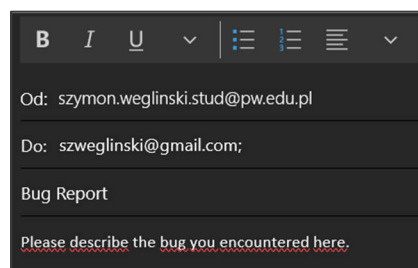


Figure 15. Screenshot of the Outlook inbox

## 4. Conclusions

The “RouteYOU” application will now be used by the Mountain Club at the 5<sup>th</sup> High School in Bielsko-Biala. However, it is perfect for all types of outdoor activities, on land (running, hiking), on water (kayaking, sailing) and in the air (paragliding). A key feature here is the route length calculation. For now, with particularly winding routes, there is a need to place points more frequently to achieve more accurate distance measurements. My further research will focus on improving this functionality by introducing a “route-finding” algorithm that connects two points, not with a straight line, but by simplifying the original trail shape. I also aim to make this algorithm interdisciplinary, because in the future I plan to replace OpenStreetMap with a more sophisticated detailed map, especially for trails.

A key component of the next versions of “RouteYOU” will be the integration of its own spatial database, allowing map-related attributes to be combined with user data. In the long term, I plan to create a mobile app closely linked to the desktop application. My goal is to use phone location data to enhance the process of adding new trails. With user consent, it will be possible to collect positioning data and transfer it in compact, user-friendly packages. After calculating and processing in the database, this data will be accessible on both devices.

Another area of development is the ongoing improvement and modernization of the application's front-end, focused on enhancing clarity and visual quality. Among the many user-experience upgrades I have in mind, the most interesting and challenging one will definitely be the introduction of 3D route models, allowing users to not only view their tracks from different angles, but also to display statistics and route details. This feature will also be a major focus of my future research. In my head, there are also plans for changing the input for the date, from simple but tedious adding parameters to opening two little calendars with clocks to set the time limits. I am convinced that this small change might really improve the experience of using "RouteYOU" application and will decrease the possibility of making mistakes in the process. Additionally, a usability enhancement will be the ability to import data from a file, particularly useful for adding lists of people associated with specific trails, making the process faster and more efficient for the user.

But for now, the most important goal – creating a complex but functional interface that helps in collecting statistics – has been successfully achieved. At the same time, I am pleased that this application remains open to further modifications, including a complete overhaul of the interface and environment. All critical user data, such as map routes or tables with statistics can be exported to an HTML file from which this data can be easily retrieved. Looking ahead, I hope that next year will be focused on developing and extending existing and new functionalities.

## REFERENCES

1. Rashed G., Ahsan R.: Python in Computational Science: Applications and Possibilities. *International Journal of Computer Applications* (0975 – 8887). Volume 43 – No.20. May 2012. pp. 26-30
2. Splitting Python Source Code Into Multiple. Available online: <https://dnmtechs.com/splitting-python-source-code-into-multiple-files> (accessed on 03.10.2024)
3. Graphical User Interfaces with Tk. Available online: <https://docs.python.org/3/library/tk.html> (accessed on 05.10.2024)
4. TkinterMapView. Available online: <https://github.com/TomSchimansky/TkinterMapView> (accessed on 05.10.2024)
5. Canva. Available online: <https://www.canva.com> (accessed on 03.10.2024)
6. Geoportal. Available online: <https://www.geoportal.gov.pl/en> (accessed on 05.10.2024)
7. Österman A.: Map visualization in ArcGIS, QGIS and MapInfo. KTH Royal Institute of Technology. Stockholm. 2014. Available online: <https://kth.diva-portal.org/smash/get/diva2:729183/FULLTEXT01.pdf> (accessed on 05.10.2024)
8. ArcGIS. Available online: <https://www.esri.com/en-us/arcgis/products/arcgis-pro/overview> (accessed on 05.10.2024)
9. Python: Saving Objects Using JSON. Available online: <https://medium.com/@sarperismetmakas/python-saving-objects-using-json-1b93370f7fa9> (accessed on 05.10.2024)
10. Euclidean distance. Available online: [https://en.wikipedia.org/wiki/Euclidean\\_distance](https://en.wikipedia.org/wiki/Euclidean_distance) (accessed on 05.10.2024)
11. GeoPy: Calculating distance. Available online: <https://geopy.readthedocs.io/en/stable/index.html#module-geopy.distance> (accessed on 05.10.2024)
12. Webbrowser library. Available online: <https://docs.python.org/3/library/webbrowser.html> (accessed on 06.10.2024)