Programming

with python™

By
Rami Tailakh
Senior Software Engineer and Data Science Practitioner
MSc in Applied Computing and Information Technology

Day-13 Agenda

- Course Recap
- Python GUI using TKInter
- REST API using Flask

Python GUI

- GUI stands for **g**raphical **u**ser **i**nterface
- There are many packages that supports developing
- graphical interface for a Python program
- Python GUI libraries are (but not limited): PyQt, wxPython and Tkinter. We will be focusing on Python **Tkinter.**
- Python Tkinter provides 19 kinds of widgets

Python Tkinter Widgets

- Tkinter provides 19 kinds of widgets

- The provided widgets are:
  label, LabelFrame, button, Canvas, Checkbutton, Entry Input, Frame, Listbox, Menu, Menu button, Radiobutton, Scale, Scrollbar, Text, tkMessageBox

Python Tkinter

- Tkinter in Python is for GUI Programming
- Tkinter is standard Python GUI library
- It gives an object-oriented interface to the Tk GUI toolkit
- It can be used by firstly importing it as follows:

```
>>> import tkinter
```

- If it is not installed, run the following command line on terminal:

```
$ pip install python-tk
```

Python Tkinter Basic Example

In PyCharm, create a new Python file and add the following lines, then, run the code 

tkinter_test.py

```
import tkinter

top=tkinter.Tk()
top.mainloop()
```

Python Tkinter Widgets-Button

- A button can be added to an interface, but we need to define a “Callback” function, as follows:

```
import tkinter
from tkinter import Button

top = tkinter.Tk()
top.geometry('300x200')

B = Button(top, text="Submit", command=buttonCallBack)
B.place(x=150, y=100)

top.mainloop()
```

Python Tkinter Widgets-MessageBox

- Tkinter also has the **messagebox** widget
- but it is required to define a “Callback” function,
- The messagebox can be added and executed as follows:

```
import tkinter
from tkinter import messagebox
from tkinter import Button

def buttonCallBack():
        msg = messagebox.showinfo("Message Title", "Message Body")

top = tkinter.Tk()
top.geometry('300x200')

B = Button(top, text="Submit", command=buttonCallBack)
B.place(x=150, y=100)

top.mainloop()
```

Python Tkinter Widgets-Label & Entry Text

- The library also provides adding an entry (input) widget and its label as follows:

```
import tkinter
from tkinter import Entry, Label, LEFT, RIGHT
top = tkinter.Tk()
top.geometry('300x200')

# Adding a Label
L = Label(top, text="Name:")
L.pack(side=LEFT)

# Adding an Entry
E = Entry(top, bd=3)
E.pack(side=RIGHT)

top.mainloop()
```

Python Tkinter Widgets-Practice 1

Show a message displays the entered text in an Entry item

```
import tkinter
from tkinter import messagebox, Button, Entry, Label, LEFT, RIGHT
.
.
top = tkinter.Tk()
top.geometry('300x400')
.
.
.
top.mainloop()
```

Python Tkinter Widgets-Checkbutton

- A Checkbutton is an input which can optionally be selected of left
- They can be as many checkboxes at once as required
- A Checkbutton can be added as follows:

```
import tkinter
from tkinter import IntVar, Checkbutton

top = tkinter.Tk()

top.geometry('200x150')

CheckVar1 = IntVar()
CheckVar2 = IntVar()
C1 = Checkbutton(top, text="Science", variable=CheckVar1, onvalue=1, offvalue=0, height=5, width=20)
C2 = Checkbutton(top, text="Technology", variable=CheckVar2, onvalue=1, offvalue=0, height=5, width=20)
C1.pack()
C2.pack()

top.mainloop()
```

Python Tkinter Widgets-Radiobutton

- A **Radiobutton** lets a user choose one item from a group of items at once, and it can be added as follows:

```
import tkinter
from tkinter import Label, IntVar, Radiobutton, StringVar, W
top = tkinter.Tk()
top.geometry('300x400')

def button_selected():
        selection = "You chose {}".format(var.get())
        label.config(text=selection)

var = StringVar()
R1 = Radiobutton(top, text="Arabic", variable=var, value='Arabic', command=button_selected)
R1.pack(anchor=W)
R2 = Radiobutton(top, text="English", variable=var, value='English', command=button_selected)
R2.pack(anchor=W)
R3 = Radiobutton(top, text="Turkish", variable=var, value='Turkish', command=button_selected)
R3.pack(anchor=W)
label = Label(top)
label.pack()

top.mainloop()
```

Python Tkinter Widgets-Listbox

- A **Listbox** is to add a list of options for the user to choose from
- A Listbox can be added as follows:

```
import tkinter
from tkinter import Listbox

LB1=Listbox(top)
LB1.insert(1,"Al-Quds")
LB1.insert(2,"Nablus")
LB1.insert(3,"Hebron")
LB1.insert(4,"Beitlahem")
LB1.insert(4,"Jenin")
LB1.pack()

top.mainloop()
```

Python Tkinter Widgets-Practice 2

Build an interface that validates either an email or a mobile Jawwal number.

Python Tkinter Widgets-Practice 3

Let's build an interface that help in creating objects for employees.

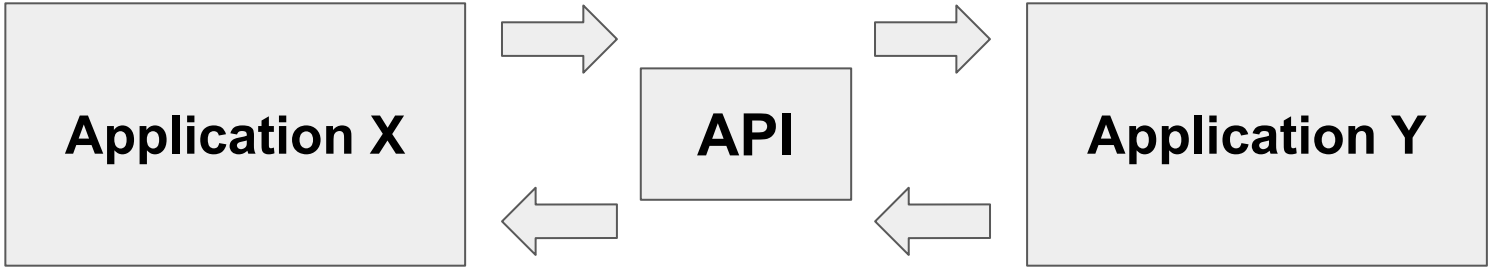
An employee has properties: *full name, date of birth, gender, address, mobile number, bank account, salary, department, skills*.

Python Tkinter Widgets-Practice 4

Building a REST API with Python Flask

What is an API?

- API stands for Application Programming Interface
- Generally speaking, API refers to the integration and communication between any two software applications
- AN API is just a medium that lets two software products <u>talk to each other</u>

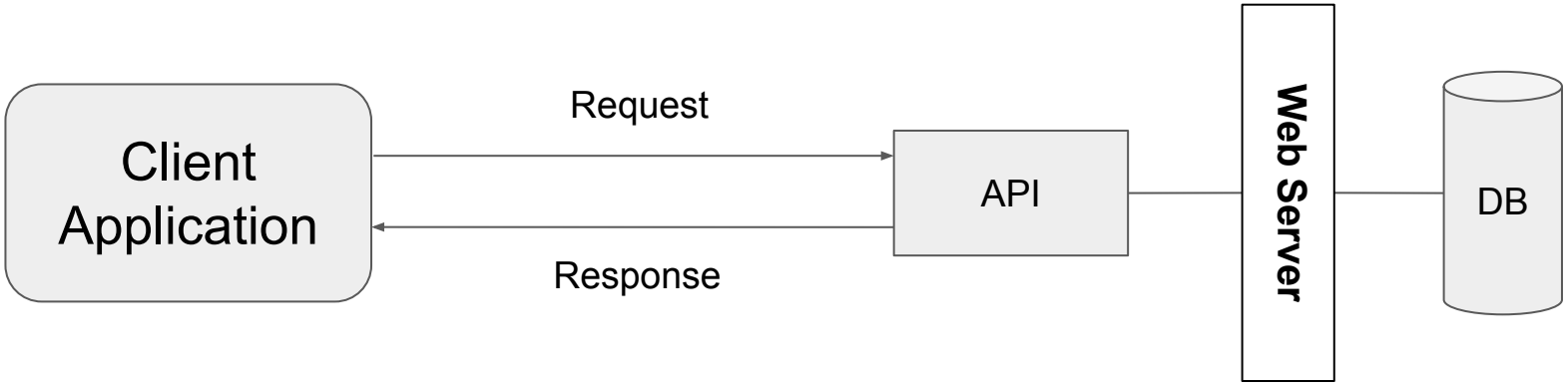


What is REST?

- **REST** stands of Representational State Transfer
- It is for designing standards, for distributed systems, between servers in order to make it easier to communicate with each other
- More specifically, REST is a set of rules to be followed when creating APIs.
- **RESTful** term is typically used when an API (web service) implementing REST architecture using HTTP

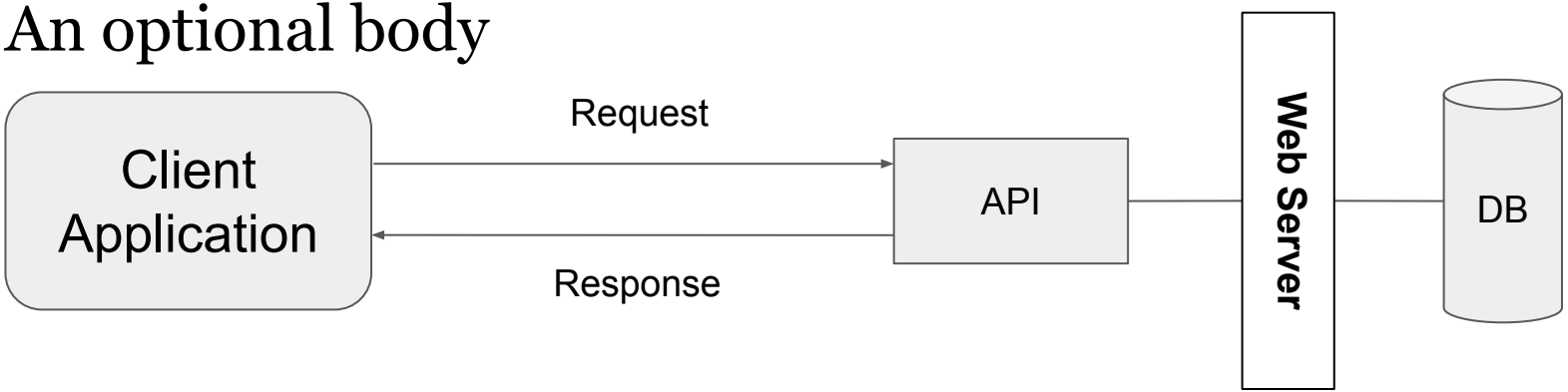
Client-Resource

- **Client** refer to a software program or application which uses an API.
- **Resource** describes an object, data, or piece of information required to be retrieved, stored, or sent to other services. <u>When a client sends a request to the server, it receives access to a resource</u>.



HTTP Request

- **HTTP** is a protocol that allows to fetch resources
- When a client sends an HTTP request, the server will respond with an HTTP response
- An HTTP request usually contains the following:
  1. A header; HTTP verb, URI and an HTTP `GET /home HTTP/1.1`
  2. A blank line separating the header from the body
  3. An optional body



HTTP verbs (methods)

- **GET:** is only used to obtain a resource from a given server. Requests using this method should have no other effect on the data

- **POST:** is used to create new resource (data) back to the server

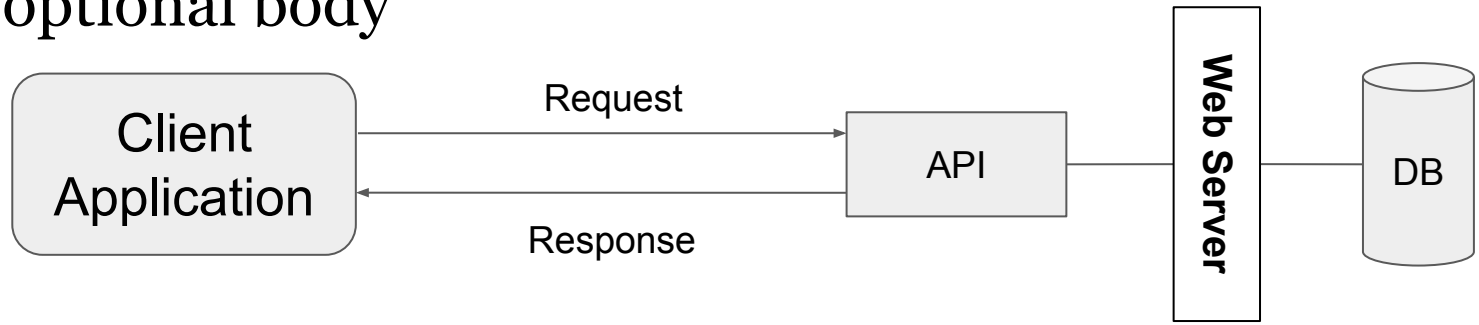
- **PUT:** is used to update a target resource

- **DELETE:** is used when deleting a resource given by a URI

HTTP Response

- When a server receives a request, it sends a message back to the client.
- If the request is successful, it returns the target resource, otherwise, it will return an error
- An HTTP response usually contains the following:

1. A header
2. A blank line separating the header from the body
3. An optional body



HTTP Response/Continue

- The **header** contains the HTTP version, status code, and explanation about the status code in plain language
- Here are some common status code examples:
  - **200 OK:** the request was successful
  - **201 Created:** a resource has been created
  - **400 Bad Request:** The request cannot be processed because of bad request syntax
  - **404 Not Found:** This says the server was not able to find the requested page
  - **500 Internal Server Error:** when an unexpected condition was experienced

Creating an API with Python Flask

Here are the basic steps for building a basic app:

1. Open PyCharm
2. Create an empty project
3. Create app.py file
4. Add the basic code in the [app.py example on Github]
5. Run flask server ▶ or by running the following on terminal:

   ```
   $  python app.py
   ```

6. If you experience the following error:

   ```
   ModuleNotFoundError: No module named 'flask'
   ```

   Run the following command line on terminal:

   ```
   $  pip install Flask
   ```

Creating an API with Python Flask/Continue

Let's improve our basic example by passing a person name with the request and returning "Hello <name>".

Run [app_person_name.py example on Github](). .

http://127.0.0.1:5000/sayHello/Adam

Hello Adam!

List resources with Python Flask

Here, we can list all employees saved in employees.csv on Github

Run get_all_employees.py example on Github.

http://127.0.0.1:5000/api/employees


```
{
"employees": [ .. ]
}
```

List resources with Python Flask/Continue

Let's improve the example by only retrieving employees in a specific department for example:

http://127.0.0.1:5000/api/employees/Sales


```
{
"employees": [ .. ]
}
```


How can we do that without displaying the department name in the URL? What best practice can be done?

Creating a Resource with Flask API

Here, we can add a new employee by running the [create_employee.py example on Github](#).

URI	/api/employees
Method	POST

Request Body Example	{
	"Name": "Samer Khouri",
	"Department": "IT",
	"Salary": 4000
	}

Response Body	{"employee": { "Name": "Samer Khouri", "Department": "IT", "Salary": 4000 }, "Status": "A new employee was created successfully"}

Creating an API with Python Flask-Practice 1

Build a flask API that can do:

- Create a new employee
- Make sure no duplication
- List all employees
- List employees by department
- List employees by sales
- Search for employees by name
- Adding bonus to (an) employee(s)
- Virtually Delete an employee
- Move an employee from a department to another

Note: an employee data should have an “employee_id” attribute in addition to the attribute “employment_status”.

Creating an API with Python Flask-Practice 2

Build a flask API that can validate:

- an email
- a Jawwal mobile number
- A URL

The response should only contain what is not valid, e.g:

```
{
    "validation_errors”: ["Email is not valid", "URL is not valid"]
}
```

Thank you