Modules: GUI Shop

Problems for exercise and homework for the Python Advanced Course @SoftUni.

It is important, when developing an application, to actually choose the tools and libraries, which will help us achieve the implementation. Today, we are going to use **tkinter** and some other dependencies for images and etc.

Please, feel free to change the implementation of steps, adding or removing certain pieces of code, debug the code and play around, so that you could fully understand the logic. You can look at the photos of the app's flow first, try to implement it, and then look at the description.

This section will show you the end result and some blurry code pics. Try to implement that on your own. If you face difficulties you can help yourself with the code from GitHub for the current commit (a link to the current commit will be left at the end of each section)

We have to define our main functionalities:

1.   Register and log in

2.   List all products

3.   Buy product

1. Global window

This part will create a canvas file that will be responsible for the tk object. We will use the file to import the tk wherever it is needed:

In "**canvas.py**" file, write the following:

```python
from tkinter import *


def create_app():
    tk = Tk()
    tk.geometry("700x600+0+0")
    tk.title("GUI Product shop")
    return tk


tk = create_app()
```

Now create a "**main.py**" file:

```python
from authentication import render_main_enter_screen
from canvas import tk


if __name__ == "__main__":
    render_main_enter_screen()
    tk.mainloop()
```

The last thing we need, before we actually start, will be to create a "**helpers.py**" file with:

```python
from canvas import tk


def clean_screen():
    for el in tk.grid_slaves():
        el.destroy()
```
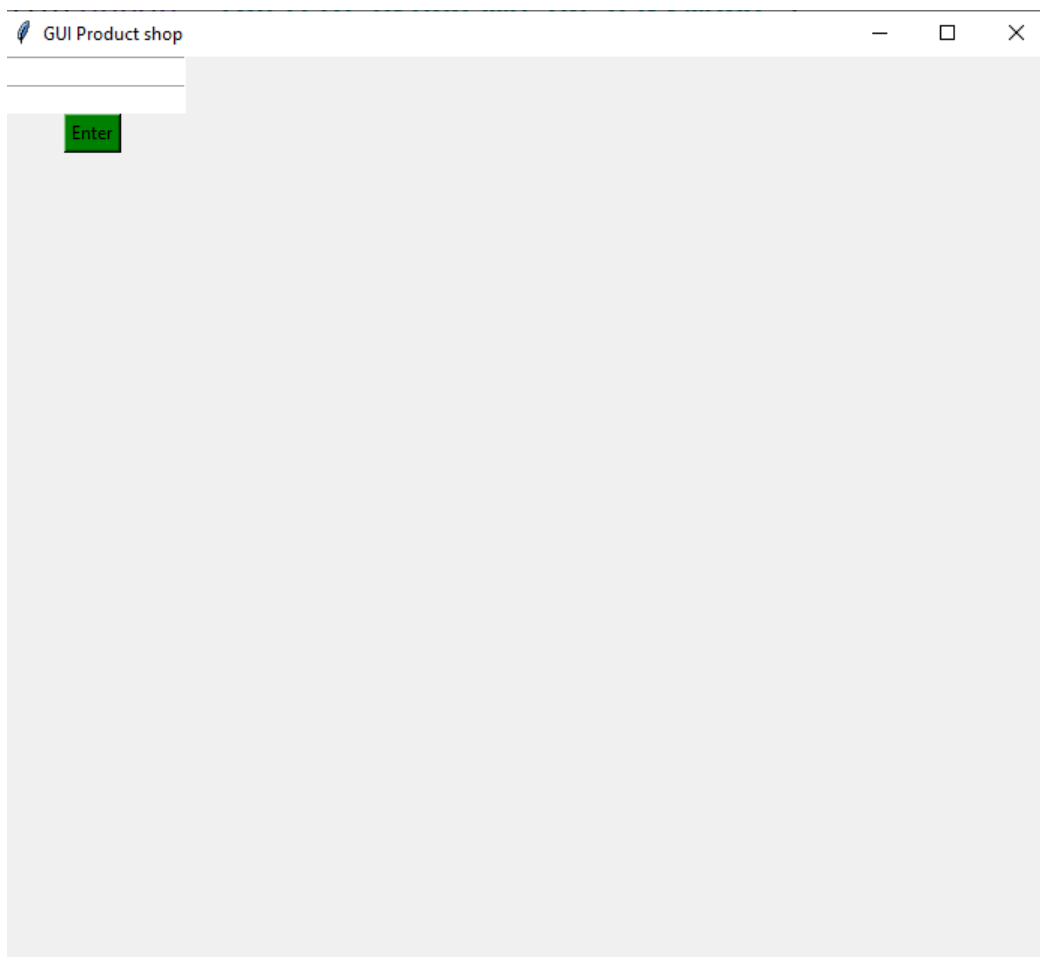
2. Login and register

Create **a file** called "**authentication.py**" and **a directory** called **db** with the files "**user_credentials_db.txt**" and "**users.txt**" for storing the data. Below, you can find the flow that should be implemented:
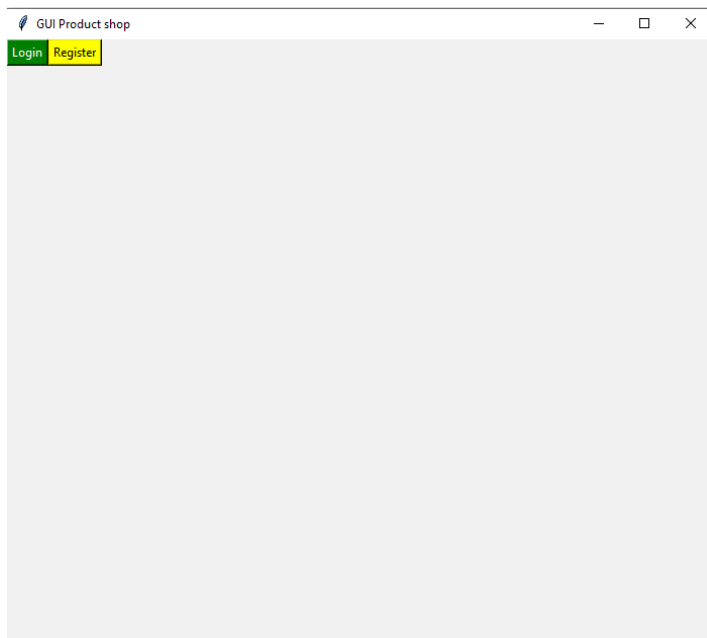
The two buttons lead to different forms – the **login form** and the **register form**.
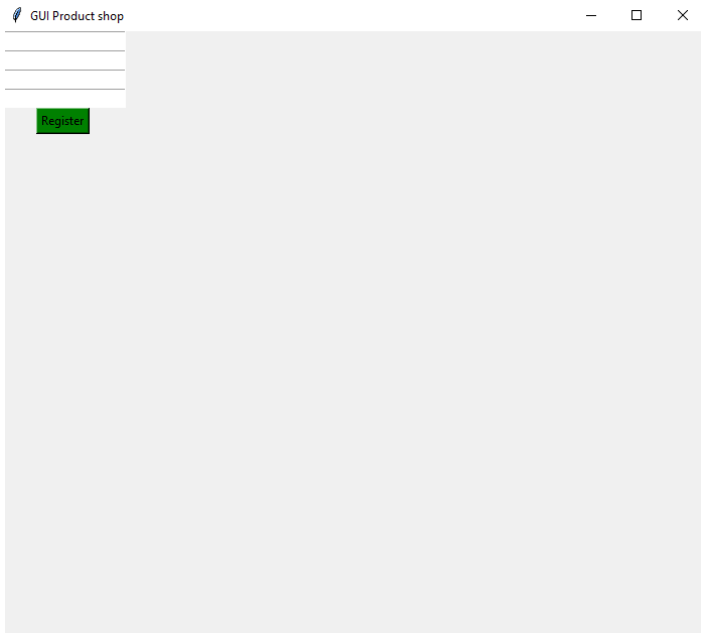
Login:

When a **username** and **password** are **entered**, it should **check** if such a username and password combination exists from the user credentials file. If so, let the **user to the product screen** (we will implement that later). If the username and/or password is invalid, a message should appear:
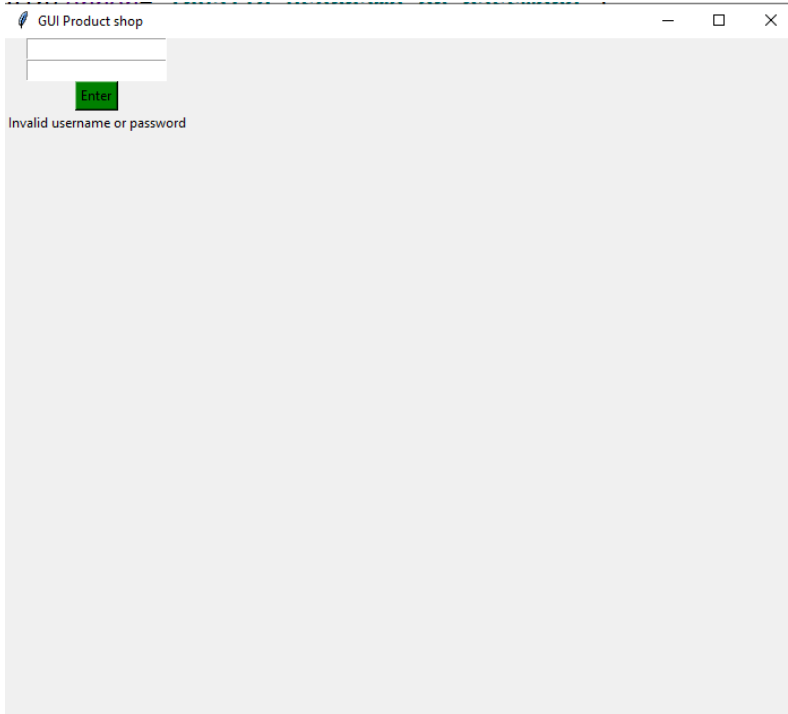
<div align="center">

"Invalid username/password"

</div>

Register form:

It should **save the user's information** to the file "`users`" as a **dictionary** (**hint,** look for `json.dumps` and how you can store complex objects in files) and the **credentials** in the "**credentials**" **file**. After that, it should render the user to the login page.

Message for invalid **username/password**:

The current implemented commit - link

Here, you have some blurry pictures if you want to take a look at how it could be implemented:

```python
import json
from tkinter import *

from canvas import tk
from heplers import clean_screen
from products import render_products


def login(username, pword):
    with open("db/                    , "r") as file:
        data = file.readlines()
        for line in data:
            name, password = line[:-1].split(", ")
            if n                        :
                with open("db/current_user.txt", "w") as file:
                    file.write(name)
                render_products()
                return

    render_login(error="Invalid username or password")
```

```python
def register(**user):
    user.update({"products": []})
    with open("db/u████t", "a", newline='\n') as file:
        file.write(json.dumps(user))
        file.write('\n')
    with open("db/user_credentials_db.txt", "a", newline='') as file:
        file.write(f"{user['username']}, {user['password']}\n")
    render_login()


def render_login(error=None):
    clean_screen()
    username = Entry(tk)
    username.grid(row=0, column=0)
    password = ████████
    password ███████████

    if error is not None:
        Label(tk, text="Invalid username or password").grid(row=3, column=0)
    Button(tk, text="Enter", bg="green", command=lambda: login(username.get(), password.get())).grid(row=2, column=0)

def render_register():
    clean_screen()
    username = ████████
    username ███████████
    password = ████████
    password ███████████
    first_name = ████████
    first_name ███████████
    last_name = ████████
    last_name ███████████
    # Make validations for names length and username to be unique and with no special chars
    Button(
        tk,
        text="Register", bg="green",
        command=lambda: register(username=username.get(), ████████████████████
        ████████████████).grid(row=4,
                                 column=0)


def render_main_enter_screen():
    Button(tk, text="Login", bg="green", fg="white", command=render_login).grid(row=0, column=0)
    Button(tk, text="Register", ██████████████████████████████████)
```
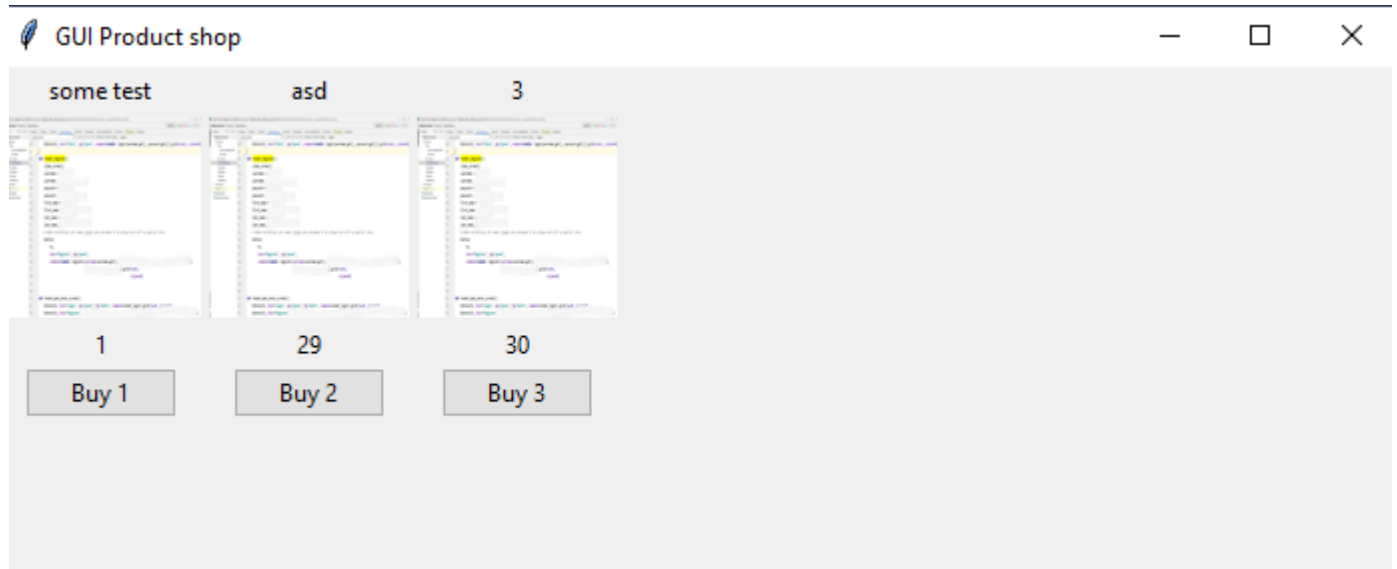
3. List all product

Here, you can see how the grid should look like:

This is the **view**. It **should be rendered** with data, fetched from **products.py which you have to create**.

The file can look like this:

```
{"id": 1, "name": "some test", "img_path": "1.png", "count": 1}
{"id": 2, "name": "asd", "img_path": "1.png", "count": 28}
{"id": 3, "name": "3", "img_path": "1.png", "count": 30}
```

The button should be bound to the product id on click. When the user "buys" some product, you should update the user file as well for the current user's products:

```
{"username": "Ines", "password": "123", "first_name": "d", "last_name": "d", "products": [1, 2, 2]}
{"username": "Inesa", "password": "123", "first_name": "a", "last_name": "s", "products": []}
```

After the **user has bought an item**, you should **render the same view** with **the updated values** for the **product's quantity**:
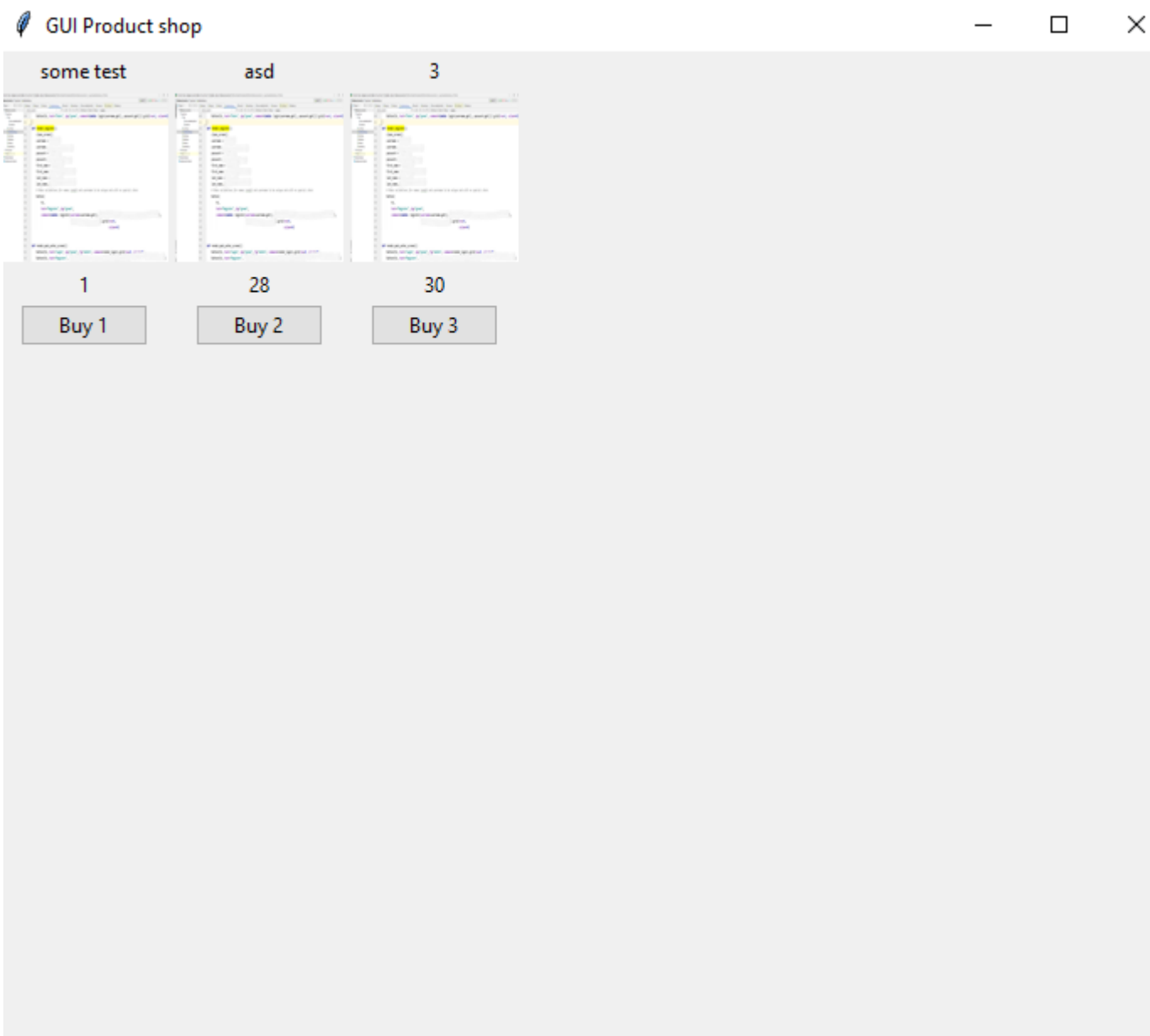
You can view the code for this commit here

4. BONUS

•   Validations to all of the fields and cases you can think of – **last_name**, **first_name length**, a **unique username**, **password validation for length and special characters** and etc.

•   Better "**CSS**" – to actually **rearrange the grid** and **colors** so that they provide a better user experience

•   Admin part for **adding products** – here you may change the "`users.txt`" structure to contain an "`is_admin`" property. A **button can be rendered**, for example "`Add product`", only **if the user is admin**. After that you should **render a new view with a form for product data**, add **a picture** for this product to **the images folder**. When **a button for creating is clicked**, you should **open a file** and **add a new product** to the file. Be careful – **the data should be always consistent**, do not miss some fields, otherwise, unexpected behavior could appear.