Assignment 1: Part 1 of 2 (handed out 18.11.2024)

Assignment 1 consists of 2 parts. This is Part 1.

You will create the skeleton of the app which uses dummy data. Your hand-in for Part 1 consists only of Exercise 1.2 and will be a video screen cast of the running app demonstrating the user interface design and interactions.

Part 2 will be handed out after the first lecture on 20.11.2024.

The handin for Assignment 1 (Part 1 and 2) will be on Tuesday the 26.11.2024.

Exercise 1.1: Tutorials for Beeware and Toga

Follow the beeware tutorial at:

https://docs.beeware.org/en/latest/index.html

Follow the Tutorials 0 to 8 (skip 6). For tutorial 5 try connecting your phone to your laptop (in file transfer mode) and see if you can select it to install the app on it.

Your app must run both on your local machine and on your phone (android, ios) either as a virtual phone or on your physical device.

Follow the toga tutorial at:

https://toga.readthedocs.io/en/stable/tutorial/index.html

Follow Tutorials 0 to 2.

Note 1: The API Reference for Toga will be relevant as you develop your project throughout this course. So make sure to bookmark it and look up information inside it from time to time:
https://toga.readthedocs.io/en/stable/reference/api/index.html

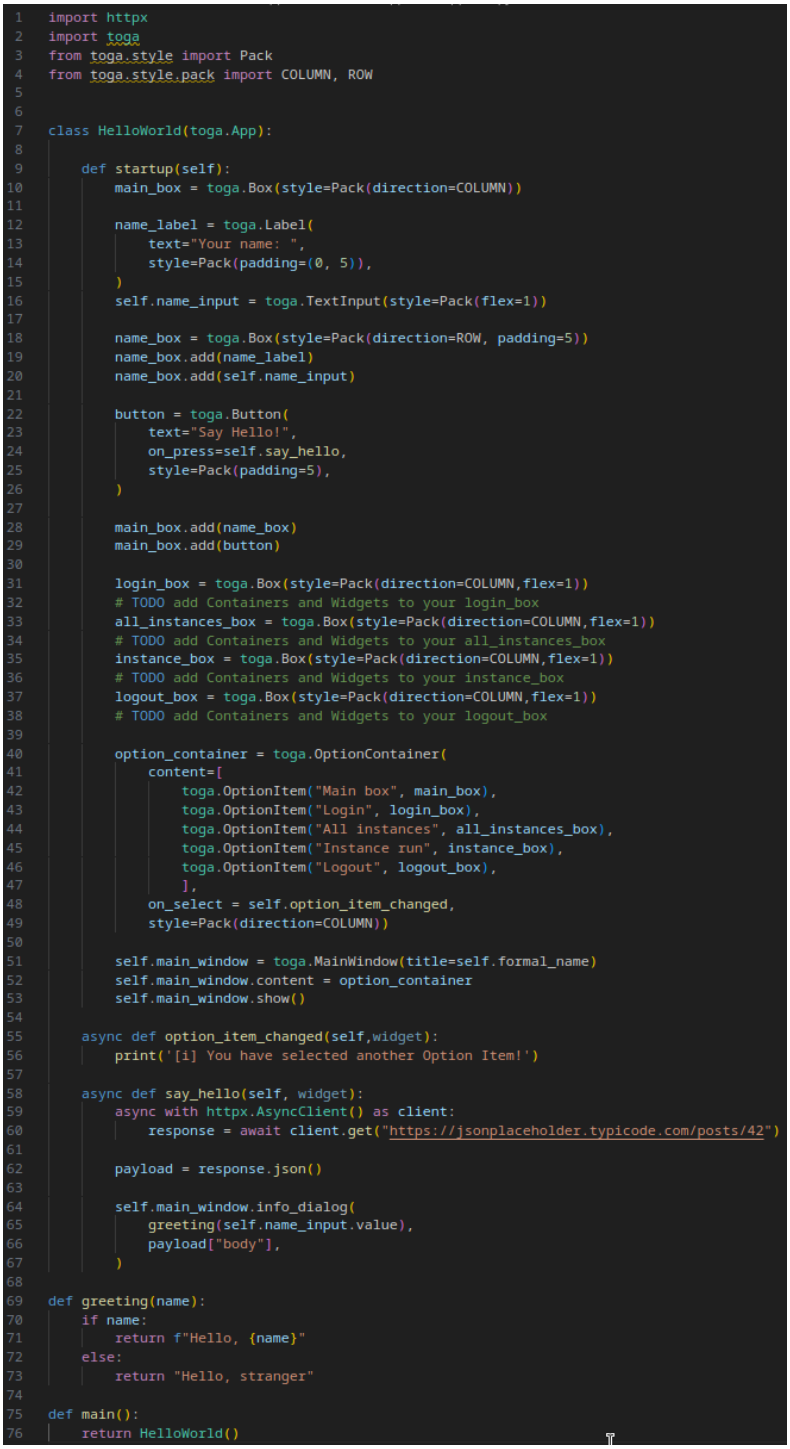
Note 2: The same code you used in the Toga tutorial can be used in the app you built using the beeware tutorial.

Exercise 1.2: The Cloud App User Interface Design

Inside the app.py file, clean the app.py startup(self) method so that you have an empty starting point. Keep the rest of the file as is. Specifically, in the startup(self) method we add UI components using Toga. We add **option_container = toga.OptionContainer(...)***. We now create 5 boxes for the OptionContainer content add the 5 boxes as **toga.OptionItem(...).** Look at the API reference for the **OptionContainer** for guidance: https://toga.readthedocs.io/en/stable/reference/api/containers/optioncontainer.html.

*("..." means you continue writing the code yourself, do not copy the "..." in your python code)

The starting point for your app python class, with the OptionContainer and the 5 boxes, should look like the following code snippet (also available for download as app.py on absalon):



```python
import httpx
import toga
from toga.style import Pack
from toga.style.pack import COLUMN, ROW


class HelloWorld(toga.App):

    def startup(self):
        main_box = toga.Box(style=Pack(direction=COLUMN))

        name_label = toga.Label(
            text="Your name: ",
            style=Pack(padding=(0, 5)),
        )
        self.name_input = toga.TextInput(style=Pack(flex=1))

        name_box = toga.Box(style=Pack(direction=ROW, padding=5))
        name_box.add(name_label)
        name_box.add(self.name_input)

        button = toga.Button(
            text="Say Hello!",
            on_press=self.say_hello,
            style=Pack(padding=5),
        )

        main_box.add(name_box)
        main_box.add(button)

        login_box = toga.Box(style=Pack(direction=COLUMN,flex=1))
        # TODO add Containers and Widgets to your login_box
        all_instances_box = toga.Box(style=Pack(direction=COLUMN,flex=1))
        # TODO add Containers and Widgets to your all_instances_box
        instance_box = toga.Box(style=Pack(direction=COLUMN,flex=1))
        # TODO add Containers and Widgets to your instance_box
        logout_box = toga.Box(style=Pack(direction=COLUMN,flex=1))
        # TODO add Containers and Widgets to your logout_box

        option_container = toga.OptionContainer(
            content=[
                toga.OptionItem("Main box", main_box),
                toga.OptionItem("Login", login_box),
                toga.OptionItem("All instances", all_instances_box),
                toga.OptionItem("Instance run", instance_box),
                toga.OptionItem("Logout", logout_box),
                ],
            on_select = self.option_item_changed,
            style=Pack(direction=COLUMN))

        self.main_window = toga.MainWindow(title=self.formal_name)
        self.main_window.content = option_container
        self.main_window.show()

    async def option_item_changed(self,widget):
        print('[i] You have selected another Option Item!')

    async def say_hello(self, widget):
        async with httpx.AsyncClient() as client:
            response = await client.get("https://jsonplaceholder.typicode.com/posts/42")

        payload = response.json()

        self.main_window.info_dialog(
            greeting(self.name_input.value),
            payload["body"],
        )

def greeting(name):
    if name:
        return f"Hello, {name}"
    else:
        return "Hello, stranger"

def main():
    return HelloWorld()
```

We will now design the User Interface (UI), without any functionality, think about it geometrically, it helps to draw the UI on paper. Keep in mind a few guiding principles:

- Containers (Box, OptionContainer, ScrollContainer) allow us to design the structure of our App.
- Containers can be stacked inside other containers:
    - We use toga.Box as our main building blocks

    - We use toga.ScrollContainer when we list multiple Box or Widget components of the same type (we want to scroll through them)
    - We use one toga.OptionContainer as our navigation Container for the app. Our app has only one toga.MainWindow and all the content is provided through boxes inside the toga.OptionContainer
- Every time we switch direction between COLUMN and ROW we need a new toga.Box
- Widgets (Labels, TextInput, PasswordInput, Buttons, Selection) are the main ways we interact with the app, we update widget labels, add content through inputs, click on buttons and select items from a dropdown selection.
- Widgets are always added inside Containers.

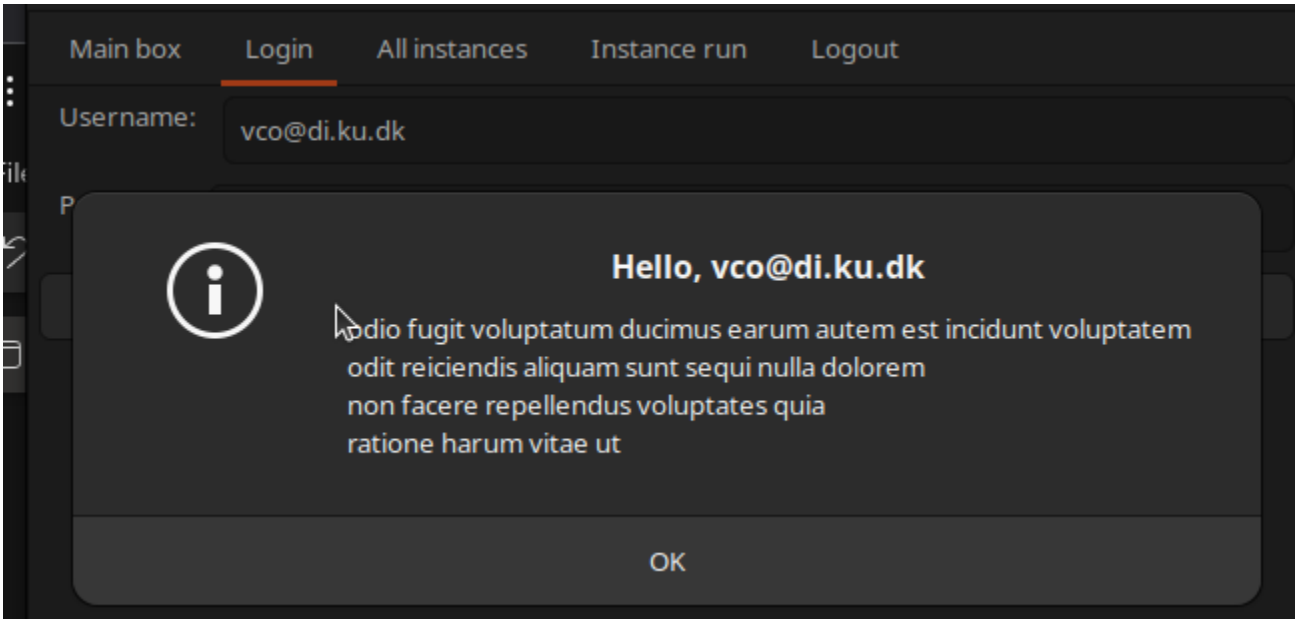
Now we focus on each box individually:

1.2.1 The login_box

For the login_box we need the user to provide a username and password. Add a Label and a **TextInput** Widget for the username and a Label and **PasswordInput** for the password. Add these pairs of Labels and Inputs inside their individual boxes with **direction=ROW**.

Finally add a login Button. Create a new method in the app class that will be used as a handler for the login Button. In the Beeware tutorial you saw how the "Say Hello!" button has the handler **self.say_hello** which displays the **self.name_input.value** of the **self.name_input = toga.TextInput(...)** value. Create the same behavior for your username **TextInput** and login Button.

The end result should look like this (when running **briefcase dev** inside the terminal):



Remember, if you get stuck you can always look up the toga Widgets and Containers in the API reference: https://toga.readthedocs.io/en/stable/reference/api/index.html

1.2.2 The all_instances_box

For the all_instances_box, first add a label saying “All instances will be listed here” and in a box add two buttons “Create new instance” and “Delete all instances” (choose which direction COLUMN or ROW you prefer for the box holding the buttons). Each button needs its own **on_press** handler. Create the methods for the handlers like so:

```
async def delete_all_instances(self,widget):
    print('[i] Delete all instances!')

async def create_new_instance(self,widget):
    print('[i] Create new instance!')
```

For now these methods will only print messages in the command line. Remember to assign the **on_press=self.create_new_instance** and **on_press=self.delete_all_instances** to the appropriate buttons.

Now we want to list several instances of a process. These instances will be retrieved from an online service in the future. For now we will use some dummy data provided as a list of python dictionary objects.

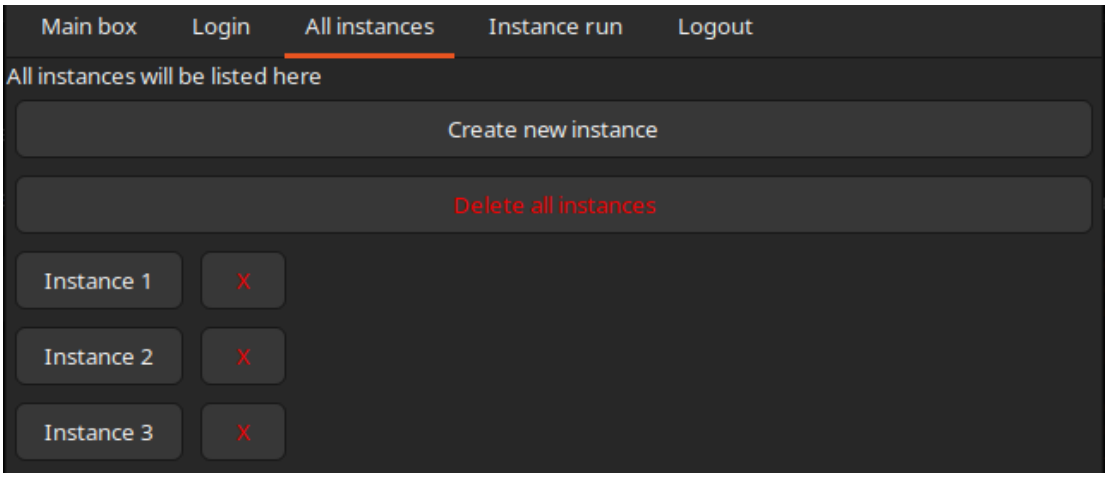
Use the code given here:

```
all_instances_container = toga.ScrollContainer(horizontal=False,style=Pack(direction=COLUMN,flex=1))
instances_box = toga.Box(style=Pack(direction=COLUMN))

instances = [{'id':1,'name':'Instance 1'},
             {'id':2,'name':'Instance 2'},
             {'id':3,'name':'Instance 3'}]

for instance in instances:

    buttons_box = toga.Box(style=Pack(direction=ROW))
    instance_button = toga.Button(
        instance['name'],
        on_press=self.show_instance,
        style=Pack(padding=5),
        id = instance['id']
    )
    buttons_box.add(instance_button)
    del_button = toga.Button(
        "X",
        on_press=self.delete_instance_by_id,
        style=Pack(padding=5, color="red"),
        id = f"X{instance['id']}"
    )
    buttons_box.add(del_button)
    instances_box.add(buttons_box)
all_instances_container.content = instances_box
all_instances_box.add(all_instances_container)
```

We have dynamically created buttons for each instance in the instances list and added them inside a ScrollContainer. The **on_press** methods for the **instance_button** and **del_button** need to be added as new methods, which for now will print the id of the instance in the command line. Use these methods:

```
async def show_instance(self, widget):
    print(f'[i] You want to show: {widget.id}')

async def delete_instance_by_id(self, widget):
    print(f'[i] You want to delete: {widget.id[1:]}')
```

The **all_instances_box** should look like this (when running **briefcase dev** inside the command line and if you have chosen the **direction=COLUMN** for the button box on the top):



When you make the window height smaller than the height of the ScrollContainer you should see that the Instance 1,2,3 buttons will be scrollable.

Clicking on the Instances buttons should print to the command line:

```
briefcase dev

[cloudapp] Starting in dev mode...
========================================================================
[i] You want to show: 1
[i] You want to show: 1
[i] You want to delete: 1
[i] You want to delete: 1
[i] You want to delete: 2
[i] You want to delete: 2
[i] You want to show: 2
[i] You want to show: 3
[i] You want to show: 3
[i] You want to delete: 3
[i] You want to delete: 3
```

1.2.3 The instance_box

In the instance box we will execute events related to a specific process instance. Let's assume our process is: A patient visiting a hospital for treatment. We have the following events happening during the hospital visitation process:

- A "Doctor" can "Diagnose" and "Operate"
- A "Nurse" can "Give treatment"
- A "Patient" can "Take treatment"

We therefore have the roles "Doctor", "Nurse", "Patient" and the events:

- "Diagnose" and "Operate" performed by the role "Doctor";
- "Give treatment" performed by the role "Nurse";
- "Take treatment" performed by the role "Patient".

Our app will need to support executing events in a process and changing roles. The **instance_box** UI will be similar to the **all_instances_box** UI design.

First add a label at the top of the **instance_box** saying "Each instance will appear here".

Then add the following dummy data:

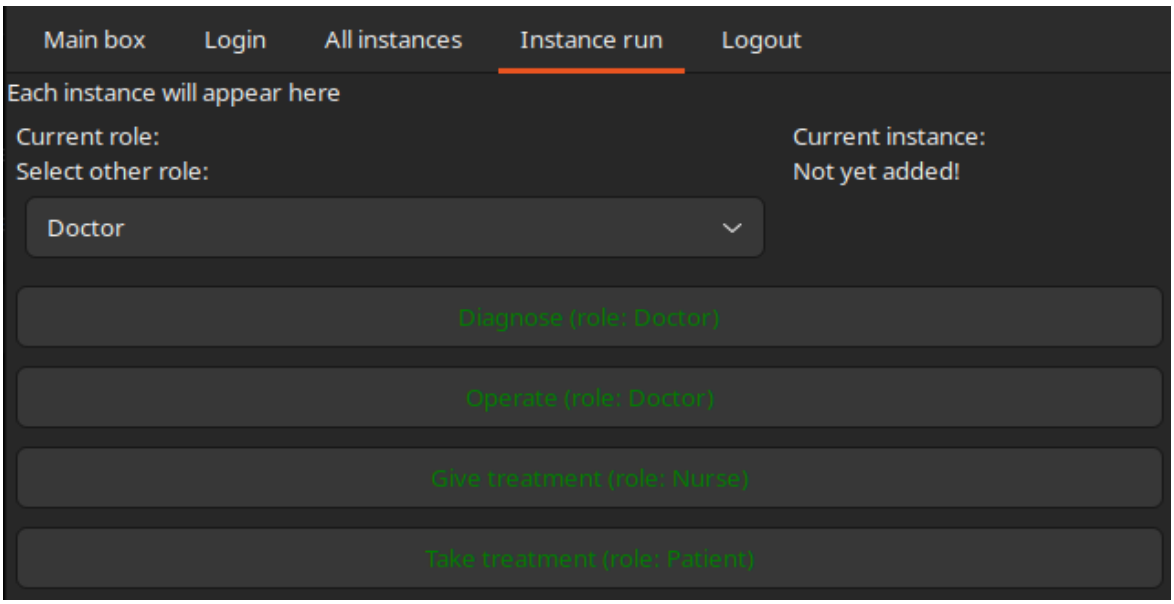
```
role_items = list([' ','Doctor','Nurse','Patient'])
selected_role_item = 'Doctor'
events = [{'id':'Diagnose','label':'Diagnose','role':'Doctor'},
          {'id':'Operate','label':'Operate','role':'Doctor'},
          {'id':'Give treatment','label':'Give treatment','role':'Nurse'},
          {'id':'Take treatment','label':'Take treatment','role':'Patient'}]
```

Now create the following structure:

- toga.Box (with the direction ROW)
    - toga.Box (with the direction COLUMN)
        - toga.Label with **text ="Current role:"**
        - toga.Label with **text="Select other role:"**
        - toga.Selection with **items=role_items, value=role_items[0]** and **on_change=self.role_changed**. Remember to create the **def role_changed(self,widget)** method in the App class. The method should print the currently selected role value. You will need to add the toga.Selection instance to the class "self." like so: **self.role_selection = toga.Selection**(...). Then, similarly to how you printed the username in the login box you can retrieve the value and print it in the command line like so: **print(f'[i] You changed the role to {self.role_selection.value}!')**

    - toga.Box (with the direction COLUMN)
        - toga.Label with **text =“Current instance:”**
        - toga.Label with **text=“Not added yet!”**
- toga.ScrollContainer
    - toga.Box
        - ***For*** each event in the **events** list of dummy data add an **event_button = toga.Button** where the **text= f"{event['label']} (role: {event['role']})"**, the **id=event[‘id’]** and you have an **on_press=self.execute_event** button handler. Remember to also create the **execute_event(self, widget)** method which should **print (f'[i] You want to execute event: {widget.id}').** Look at how the ScrollContainer was populated in the **all_instances_box**. The main difference is that in this ***for*** loop we only need to add one button.

In the end your **instance_box** should look like this:



Clicking on events should print to the command line:

```
 briefcase dev

[cloudapp] Starting in dev mode...
========================================================================
[i] You want to execute event: Diagnose
[i] You want to execute event: Operate
[i] You want to execute event: Give treatment
[i] You want to execute event: Take treatment
```

1.2.4 The logout_box

For the logout_box Add a single Logout button with an **on_press** handler printing to the command line **“You want to logout!”**.

The hand in for Assignment 1 Part 1

Create a video screen cast showing the user interface design and interactions from Exercise 1.2. Run **briefcase dev** and demonstrate the 4 implemented boxes by typing inside the input widgets and clicking through each button (for the buttons in the scroll containers it is enough to click one button from each group), show also the command line output (See the example screen cast in the assignment 1 page on absalon).