Assignment 2 Part 1 (hand in 10.12.2024)

Now the UI of the app is complete, and you have a process model in the dcrgraphs.net portal. Next, we will add functionality to our app by connecting to the dcrgraphs.net REST API service. This will allow you to create instances (simulations) of the DCR graph you've created in Assignment 1. Inside each instance you will be able to retrieve the events and the roles. You will be able to execute events based on the chosen role in the app.

Note: Remember that what we call instance in our app is called sim or simulation in the REST API.

What to hand for Part 1:

For **B)** create a video screencast of **step 6** for your DCR graph in the portal and the open terminal. Starting from the login and execute at least 3 events or until none of the events are pending.

For **C)** create a video screencast of your phone (virtual or physical) showing the exact same steps as **B) step 6**.

Part 1

A) Create a standalone DCR Active Repository class.

1. Download the python template from absalon called "dcr_active_repository.py"
2. In your code folder (at the same level as the app.py file you edited in assignment 1) create one python package called "services" (a directory where you create an empty file called "__init__.py")
3. Place "dcr_active_repository.py" inside "services"
4. Fill in the "dcr_active_repository.py" template with the correct code
5. Open pyproject.toml and under "sources" add the "services" folder as "src/helloworld/services" (replace helloworld if you changed the name of your folder where "app.py" resides)

B) Once you filled in the "dcr_active_repository.py" template you can test it in the terminal.

1. The template has a main() method that can be run as a script.
2. Inside the main() method assign the graph_id variable with your own graph id from the dcrgraphs.net portal
3. Open a terminal and navigate to the services folder.

4. Make sure you activate your python virtual environment
5. Run the command: “python dcr_active_repository.py”
6. Test that the functionality works as expected by starting side by side a simulation in the dcrgraphs.net portal and executing events in the same order as you do in the script code.

(Semi-)Automating briefcase

Finally, you must now make sure the app can run on your phones (virtual or physical). To aid in the development process you can use the provided scripts (you can download them from absalon). Place them inside the same folder as your “pyproject.toml”.

- desktop.sh (or desktop.ps1 if you are using windows and powershell)
- android.sh (or android.ps1 if you are using windows and powershell)
- ios.sh

On linux and macOS for the .sh scripts remember to run “chmod +x desktop.sh”, “chmod +x android.sh”, “chmod +x ios.sh” to allow you to execute the scripts. On Windows and .ps1 you might need to execute “Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser” before running the actual .ps1 scripts (similar to https://docs.beeware.org/en/latest/tutorial/tutorial-0.html).

For android and ios you need to replace the text "add your device id here" (quotes also need to be removed) with your device id. You can find this id when running “briefcase run android/” or “briefcase run ios” and selecting the target device.

Note: The scripts assume your app folder is named helloworld. Edit the scripts if this is not the case.

C) Inside “app.py” add the function calls from the DCR REST API into the appropriate handlers and replace the dummy data for the “instances” and “events” with real data from your process.

Hints: Help yourselves to code snippets from the “main()” method in the “dcr_active_repository.py”. For references to toga see:
https://toga.readthedocs.io/en/stable/reference/api/containers/optioncontainer.html and
https://toga.readthedocs.io/en/stable/reference/api/widgets/button.html

First let's remove the functionality from the beeware tutorial, if you haven’t already done so. Remove the “Main box” OptionItem from the OptionContainter and all the associated code related to the main_box, including the say_hello and greeting methods. Then add two class parameters:

```
from services.dcr_active_repository import check_login_from_dcr, DcrActiveRepository, EventsFilter, DcrUser

class CloudApp(toga.App):
    graph_id = "your graph id" # change to your own graph id
    dcr_ar = None # we instantiate it after we login successfully
```

one to hold your graph_id and the other to hold a reference to the DcrActiveRepository class once we have logged in, abbreviated as dcr_ar. Fill the graph_id with your own.

Make the following behavior (1 to 6):

1. When the app starts only the “Login” OptionItem is enabled.

In the end of the startup() method add these lines:

```
self.option_container.content['Logout'].enabled = False
self.option_container.content['All instances'].enabled = False
self.option_container.content['Instance run'].enabled = False
```

2. Once there is a successful login to the DCR REST API the “Login” OptionItem should be disabled and the user is moved into the “All instances” OptionItem.

Remember that the methods inside the dcr_active_repository.py are asynchronous and you must use the await keyword before the method call.

```
connected = await check_login_from_dcr(self.user_input.value, self.password_input.value)
```

Inside the login_handler(widget) you must first add the logic for calling the check_login_from_dcr and then only if the login is successful you must update the enabled OptionItems:

```
self.option_container.content['All instances'].enabled = True
self.option_container.content['Instance run'].enabled = True
self.option_container.content['Login'].enabled = False
self.option_container.content['Logout'].enabled = True

self.option_container.current_tab = 'All instances' #Must match the name of an OptionItem
```

To programmatically navigate through the tabs we change the parameter current_tab with the name of one of our for OptionItems. Here we navigate to the ’All instances’ OptionItem.

Here is the complete login_handler:

```
async def login_handler(self,widget):

    connected = await check_login_from_dcr(self.user_input.value, self.password_input.value)

    if connected:
        self.user = DcrUser(self.user_input.value,self.password_input.value)
        self.dcr_ar = DcrActiveRepository(self.user)

        self.option_container.content['All instances'].enabled = True
        self.option_container.content['Instance run'].enabled = True
        self.option_container.content['Logout'].enabled = True

        self.option_container.current_tab = 'All instances'

        self.option_container.content['Login'].enabled = False
    else:
        print(f"[x] Login failed try again!")
```

3. The “All instances” box should be refreshed whenever:
    3.1. The user navigates to the “All instances” OptionItem
    3.2. The user deletes one instance
    3.3. The user deletes all instances

Refreshing an OptionItem means that the box inside it needs to be re-created and the instances retrieved using the dcr_ar REST API. Since this can happen from multiple user interactions we will create an async method that holds this logic. Name this method show_instances_box() and from the startup() method move all the code responsible for creating the all_instances_box into it. In the method you must first clear the existing widgets and in the end refresh the box to display the new content, as shown here:

```
    async def show_instances_box(self):
        self.all_instances_box.clear()
        #your code for the all_instances_box from startup(self) goes here
        self.all_instances_box.refresh()
```

In addition you must make the all_instance_box a class variable using the self. parameter.

Finally replace the list of dummy instances with a call to the dcr_ar like so:

Replace:

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

With:

```
self.instances = {}
dcr_ar_instances = await self.dcr_ar.get_instances(self.graph_id)
if len(dcr_ar_instances)>0:
    self.instances = dcr_ar_instances
```

To achieve the functionality required for 3.1 update the option_item_changed callback handler with:

```
async def option_item_changed(self, widget):
    print('[i] You have selected another Option Item!')
    if widget.current_tab.text == 'All instances':
        await self.show_instances_box()
```

Finally for 3.2 and 3.3 make sure to update delete_instance_by_id and delete_all_instances accordingly. First call the delete_instance method from the dcr active repository and then call the show_instances_method. Note, when deleting all instances you need to loop through self.instances.

4. When the user creates a new instance or clicks on an instance the user is moved to the “Instance run” OptionItem.

In the show_instance and create_new_instance handlers you create this behavior by changing the current_tab of the OptionContainer to the “Instance run” OptionItem. To keep track of the currently selected instance you need to create variable self.current_instance_id. It gets assigned the widget.id in the show_instances method. It also gets assigned the result of calling the self.dcr_ar.create_new_instance in the create_new_instance method.

5. The “Instance run” box should be refreshed whenever:
    - 5.1. The user clicks on an instance or creates a new instance
    - 5.2. The user changes roles
    - 5.3. The user executes an event

This should be done in a similar manner to what you did in step 3. Name the method responsible for refreshing the instance as show_instance_box:

```
async def show_instance_box(self):
    self.instance_box.clear()
    #your code for the instance_box from startup(self) goes here
    self.instance_box.refresh()
```

Again make sure you add self. to the instance_box, and replace the dummy data with data from the dcr active repository.

Replace:

```
role_items = list([' ','Doctor','Nurse','Patient'])
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'}]
```

With:

```
events = await self.dcr_ar.get_events(self.graph_id,self.current_instance_id,EventsFilter.ALL)
role_items = []
if self.user.role:
    role_items.append(self.user.role)
for event in events:
    event_role = event.role
    if event_role not in role_items:
        role_items.append(event_role)

self.role_selection = toga.Selection(
        items=role_items,
        on_change=self.role_changed,
        style=Pack(padding=5)
)
if len(role_items)>0:
    self.role_selection.value = role_items[0]
    self.user.role = self.role_selection.value
```

Finally, to accurately represent the DCR Graphs execution semantics we can add visual cues for enabled, and pending events, as well as only enable events that match the currently selected role. Update the events loop for the instance_box with:

```
for event in events:
    color = None
    btn_enabled = True
    text = event.label
    if event.enabled:
        color = "green"
    if event.pending:
        color = "blue"
        text = text + " !"
    if len(event.role)>0:
        if event.role != self.user.role:
            btn_enabled = False
        text = text + f" (role: {event.role})"
    if event.enabled:
        event_button = toga.Button(
            text=text,
            style=Pack(padding=5, color=color),
            id=event.id,
            on_press=self.execute_event,
            enabled=btn_enabled
        )
        events_box.add(event_button)
```

For 5.2 remember to update the self.user.role in the role_changed handler before calling show_instance_box.

When deleting an instance or deleting all instances you might end up with the “Instance run” OptionItem still holding a deleted instance. To prevent this you need to check if the instance id about to be deleted by delete_instance_by_id is the same as the self.current_instance_id. You can then safely clean the self.instance_box like so:

```
self.instance_box.clear()
self.instance_box.add(toga.Label('Select an instance from All instances or Create new!'))
self.instance_box.refresh()
```

6. When the user logs out, by clicking the Logout button in the “Logout” OptionItem, the user is moved back to the “Login” OptionItem and all other OptionItems are disabled.

To achieve the logout we set the enabled parameter for Login OptionItem to True, we switch to it as the current tab and we set the enabled parameter to False for the following OptionItems: All instances,

Instance run and Logout. We also remove any stored username and password by setting those variables to = None.