Hand-In-1

In this assignment you should hand in the tasklist app, allowing to login, create a new instance, show enabled or pending tasks and execute enabled tasks

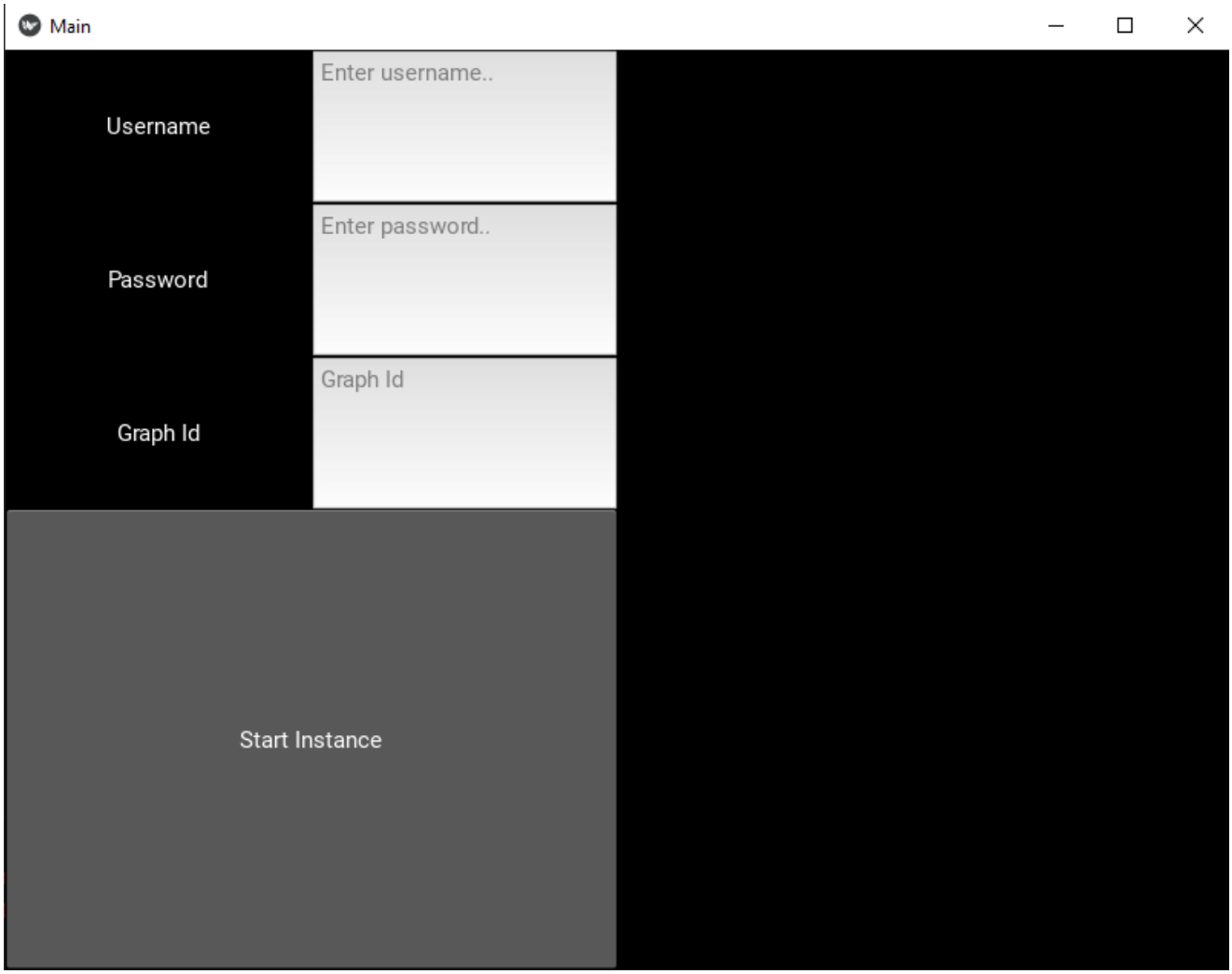
What to hand in

1. A brief report (handed in as a group). You may follow this structure
    - Introduction (approx 1/2 page, what is the report about)
    - Overview of how your app works/or how it should work and how it was tested (approx 1 page you may put screen shots and test results in appendix)
    - Overview of code - where did you add code and why (approx 1 page - put the relevant code in appendix)
    - 1/2 page conclusion
    - Appendixes:
        - Screenshots (how to use the app, result of tests)
        - Relevant code snippets
        - A picture and description of the DCR graph you use - including URL of the graph
2. A folder with your Python code

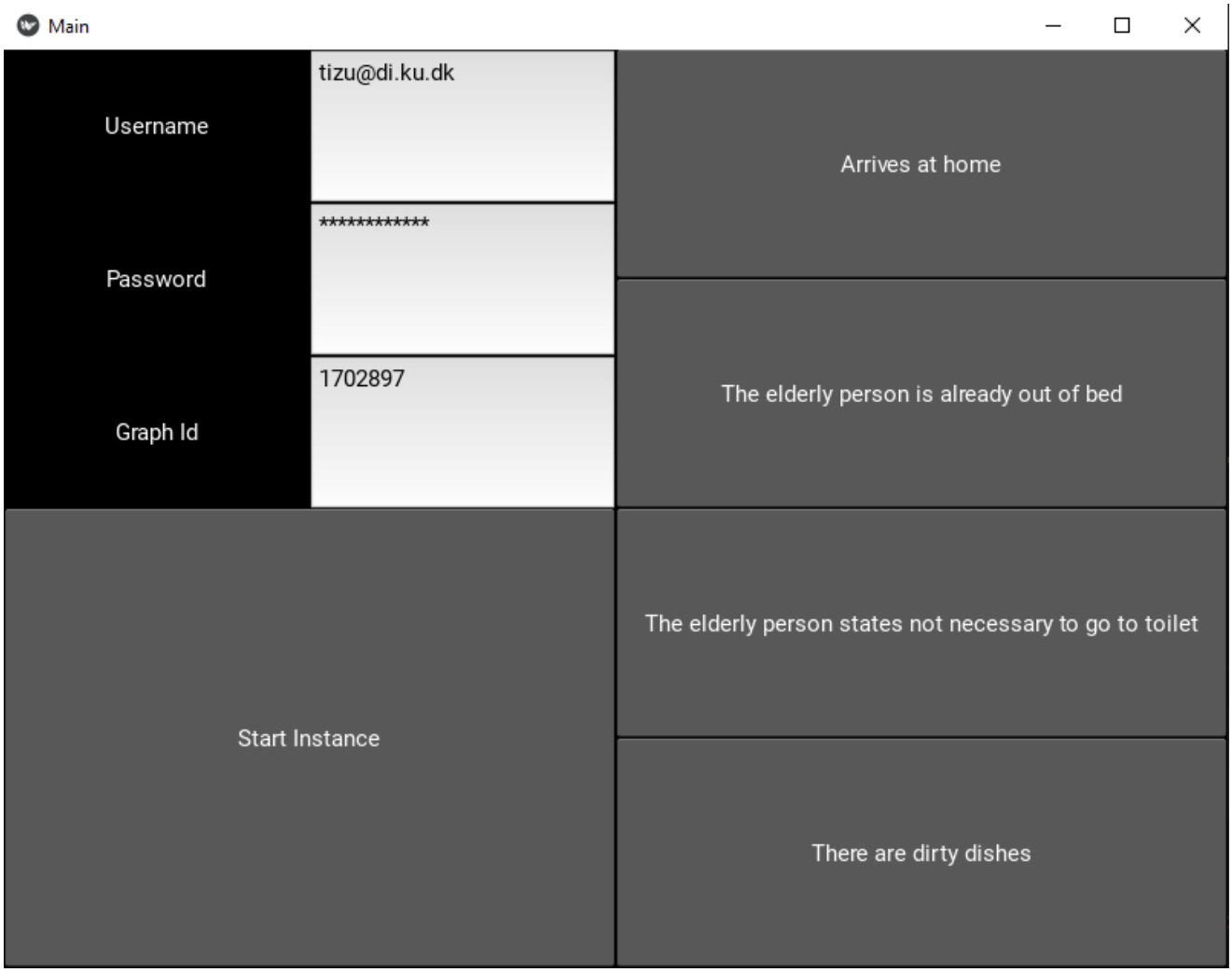
Instructions

In this hand-in, you will put the pieced together, that you already implemented in the course of the exercise.

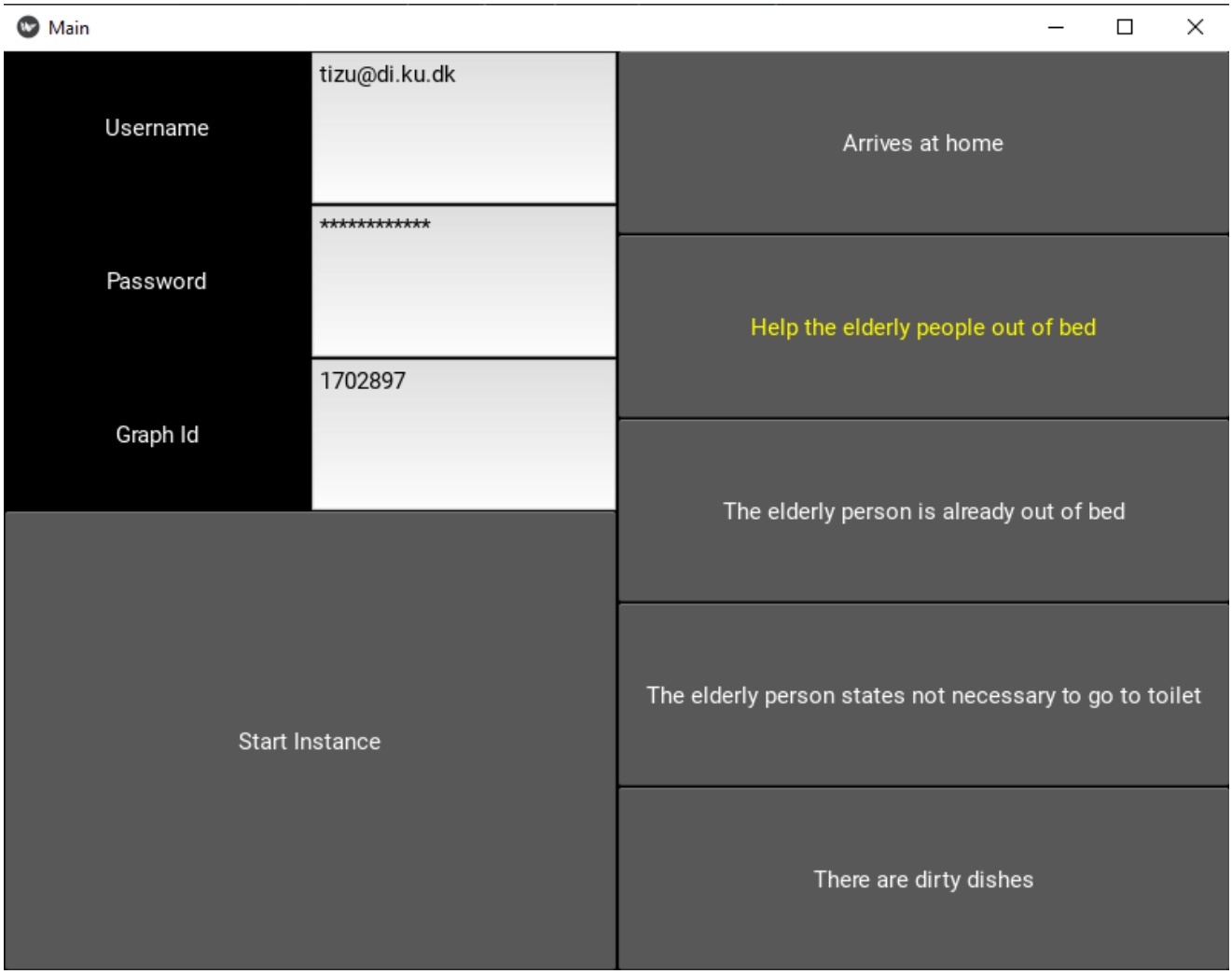
Final Result:



The final result contains three `TextInput` fields, one for a password, one for a username and one for the graph id you would like to work with. They are accompanied by 3 `Label`s explaining the purpose of the text fields. We have a button `Start Instance` that can connect to the dcr server and start a new simulation using the authentication and the graph id. When we press the button `Start Instance` the application should show all possible events as buttons.



You previously had used labels, but we want to use buttons here, so that we can communicate to the server, that we have performed a certain event by clicking on the button. When clicking `Arrives at home` for example, we get a new event `Help the elderly people out of bed`.



The button is colored yellow, because the event is pending. This means that the event has to be performed before the process can be finished. It should always be possible to click `Start Instance` to create a new instance and run a fresh simulation.

To implement the task app, you can use the source code of the exercises. We recommend following the workflow:

(1) start to create a class `MainApp(App)` in which you set up a constructor, that contains all the elements of the gui. Place them in layouts until you have the structure that we are looking for.
Remember, that the buttons to trigger events are added dynamically in the source code and can not be added at the start of the application. However, it is useful to add an empty container (`BoxLayout`) that later stores the buttons.

(2) Implement a method on top level, that connects to the DCR server and gets the enabled events
(The source code to this can be completely found in the previous exercise)

```
def get_enabled_events(graph_id: str, sim_id: str, auth: (str, str)):
...
```

(3) Implement a method `def start_sim(self, instance):` inside the `MainApp(App)` class that triggers the behavior of the button `Start new instance`. Remember that this method should also work when you are in a running simulation. Thus, it makes sense to clear the layout containing the event buttons before filling it with buttons again. The structure of your method may look like this:

```
def start_sim(self, instance):
        #- contact dcr server to put a new simulation
        #- store the simulation id
        #- create button of enabled events <- this is explained in the further
section (5)
```

(4) In order to let the server know, which event we want to trigger we need to store the information about the !!!event id!!! inside the button. It is not the label of the event that is send to the server, but the id. For this, we introduce a new widget, called `SimulationButton`. The source code is completely given bellow:

```
class SimulationButton(Button):
    def __init__(self, event_id: int,
                 graph_id: str,
                 simulation_id: str,
                 username: str,
                 password: str,
                 text: str):
        Button.__init__(self)
        self.event_id = event_id
        self.text = text
        self.graph_id = graph_id
        self.simulation_id = simulation_id
        self.username = username
        self.password = password
        self.manipulate_box_layout: BoxLayout = BoxLayout()
        self.bind(on_press=self.execute_event)
```

Aside from working as a normal button, this button stores the event id as `self.event_id`. As you may notice, the behavior of the button when pressed is `execute_event`. However, we are missing the execute event function. It is your task to include the method inside of the `SimulationButton` class. It is structured like this:

```
def execute_event(self, instance):
    url = (f"https://repository.dcrgraphs.net/api/graphs/{self.graph_id}/sims/"
           f"{self.simulation_id}/events/{self.event_id}")
    # send a post request to dcr server with basic authentication
    # create the buttons of new enabled events <- explained in (5)
```

(5) We are missing one piece to the puzzle, namely how we actually create the buttons inside our empty BoxLayout just waiting to be filled. For this, we can add a function on top level besides `get_enabled_events..` called `create_buttons_of_enabled_events`. The signature of the function looks like this:

```
def create_buttons_of_enabled_events(
        graph_id: str,
        sim_id: str,
        auth: (str, str),
        button_layout: BoxLayout):
```

It is handed all the information about the graph, the simulation id and the basic authentication, and additionally, we also pass a BoxLayout `button_layout` that is the layout we want to render the buttons in. It is your task now to use the `get_enabled_events` function inside this function to get all enabled events from the server. For each event that you have received, you want to create a new button (our SimulationButton) that contains the necessary informations about the event. The following line of code achieve this:

```
events_json = get_enabled_events(
    graph_id,
    sim_id,
    auth)
# cleanup of previous widgets
button_layout.clear_widgets()
events = []
# distinguish between one and multiple events
if not isinstance(events_json['events']['event'], list):
    events = [events_json['events']['event']]
else:
    events = events_json['events']['event']
# add a custom button, that stores the event id
for e in events_json['events']['event']:
    s = SimulationButton(
            #the actual event id
        e['@id'],
        graph_id,
        sim_id,
        auth[0],
        auth[1],
        #the label of the event
        e['@label']
    )
    s.manipulate_box_layout = button_layout
#add a line of code that colors pending events
#to distinguish them from non pending events
button_layout.add_widget(s)
```

(6) Link everything together and try your application

Code Structure:

The initial structure with the information given here, can look like this:

```
import httpx
import xmltodict
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput

def get_enabled_events(graph_id: str, sim_id: str, auth: (str, str)):
        pass


def create_buttons_of_enabled_events(
        graph_id: str,
        sim_id: str,
        auth: (str, str),
        button_layout: BoxLayout):
    pass
        # source code provided in exercise sheet


class SimulationButton(Button):
    def __init__(self, event_id: int,
                 graph_id: str,
                 simulation_id: str,
                 username: str,
                 password: str,
                 text: str):
        Button.__init__(self)
        self.event_id = event_id
        self.text = text
        self.graph_id = graph_id
        self.simulation_id = simulation_id
        self.username = username
        self.password = password
        self.manipulate_box_layout: BoxLayout = BoxLayout()
        self.bind(on_press=self.execute_event)

    def execute_event(self, instance):
            pass

class MainApp(App):
```

```
    def __init__(self):
        App.__init__(self)
        self.login_button = Button(text='Start Instance')
        # initialize all elements here
        def build(self):
                pass

        def start_sim(self, instance):

if __name__ == '__main__':
    mainApp = MainApp()
    MainApp().run()
```

Finally, this might be the most complex bit of code you have yet seen, but don't get discouraged. You will get it eventually.

Happy Coding and good luck!