Assignment 2 - Part 2

We will now setup a database in the cloud and connect to it with our app. This will allow us to store the latest DCR role for the logged in user, as well as to select instances that are specific to the user. We will also store the acceptance criteria (no events can be left pending at the end of a run) of the instance in order to not delete any instances that are not valid (i.e. do not meet the acceptance criteria).

What to hand in for Part 2

Part 2 of Assignment 2 you must hand in your `app.py` file together with the `services` folder containing both the `dcr_active_repository.py` and the `database_connection.py` files. Rename the `app.py`, `dcr_active_repository.py`, `database_connection.py` files with your group name as it appears on absalon, for example: `app_group1.py`, `dcr_active_repository_group1.py`, `database_connection_group1.py`. In `app_group1.py` add a link to your DCR graph as a code comment next to the `graph_id` and update the `import` statement for the `dcr_active_repository_group1`, `database_connection_group1` imports.

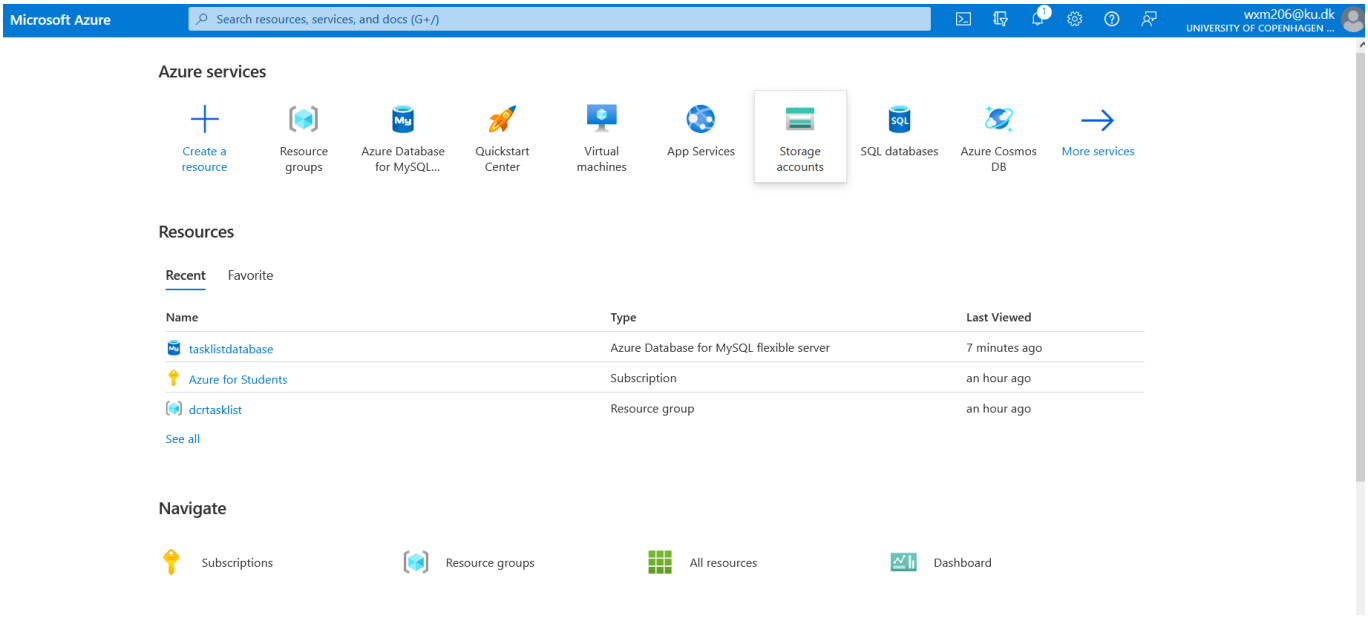
1. Setup your Cloud service on Azure

We're using Microsoft Azure services for this: https://azure.microsoft.com/da-dk/free/students. If your KU email does not work use this instead: https://azure.microsoft.com/da-dk/free/ (You might need to use your card details for this)

Go to "Start Free" or "Free Account". Use your KU email and create an account. You get 100 virtual dollars to use on cloud resources.

You will get an "Azure for Students" subscription. A cloud service subscription is like your mobile phone subscription. When you add services and increase the usage it will also you increase your cost.

Once you have created the subscription, go to: https://portal.azure.com/ and login with your KU account.



2. Create an Azure Database for MySQL Flexible Server

You will use one database per group.

1. On the front page go to the search bar and type: "Azure Database for MySQL Flexible Server"
2. Pres the "+ Create" button.
3. Keep the default "Flexible Server" and click "Advanced Create"
4. Fill in the forms.
    - Create a new Resource Group (This is a logical set of resources that are project specific)
    - Enter a server name (Make sure to not name your database server the same as other groups. e.g. "tasklistdatabasegroup1")
    - For Region select "UK South" (if the region does not work try another)
    - MySQL version: 8.0
    - Workload type: "For development or hobby projects"
    - Compute + storage: (leave as is) "1 vCores, 2 GiB RAM, 20 GiB storage, Auto scale IOPS" and Geo-redundancy: Disabled
    - Availability zone: (leave as is) "No preference"
    - Authentication method: MySQL authentication only
    - Admin username and password: Add a database username and password (these are to connect to the database, do not use any personal passwords as we will store it in the app as plain text)
5. Click on "Next: Networking"
    - Under "Firewall rules" press add "+ Add 0.0.0.0 - 255.255.255.255". You get a message saying that this will expose your database to the internet unrestricted. Press "Continue" for to this message.
6. Click on "Review and create" (If you used the basic configuration, on the right you should see an estimated total cost of USD 10/month)
7. Click on "Create".
8. While you wait for it to be created download the MySQL Workbench.

3. MySQL

Go to: https://dev.mysql.com/downloads/ and install "MySQL Workbench" on your machine.

Once installed you need to configure the connection and database:

1. Open MySQL Workbench and under "Database" go to "Manage connections"
2. Give it a connection name (e.g. tasklist) and once the MySQL Database is deployed in the Azure portal open that resource in your browser, we will need to copy some information from there.
3. In the Azure portal under the "tasklistdatabase" resource find the "Settings" sub-menu on the left panel:
    - Go to "Databases" and press "+ Add" give it a name (e.g. tasklistdatabase)
    - Now go to "Networking" press "Download SSL Certificate" in the top menu bar. This will download a file to your computer. Take that file and store it in your Beeware project under the "services" folder, i.e. next to the "dcr_active_repository.py" file.
    - Now go to "Connect" find the "MySQL Workbench" tab and follow the instructons from there. (Note: You will also see a "Connect from your app" tab with a python tab which we will use later)

4. Now you are connected to the database. In MySQL Workbench, open an "SQL Tab"  paste the following:

```
CREATE TABLE Instances (
    InstanceID int,
    IsInValidState bool,
    CONSTRAINT PK_Sim PRIMARY KEY (InstanceID)
);

CREATE TABLE DCRUsers (
    Email varchar(255),
    Role varchar(255),
    CONSTRAINT PK_Email PRIMARY KEY (Email)
);

CREATE TABLE UserInstances (
    Email varchar(255),
    InstanceID int,
    CONSTRAINT PK_UserInstance PRIMARY KEY (Email, InstanceID),
    CONSTRAINT FK_Email FOREIGN KEY (Email) REFERENCES DCRUsers(Email),
    CONSTRAINT FK_Instance FOREIGN KEY (InstanceID) REFERENCES
Instances(InstanceID)
);
```

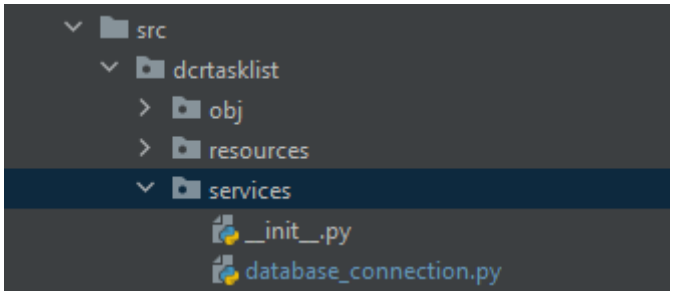
5. Press the  "Execute Button" (or CTRL + SHIFT + ENTER) to create the table. (Note: if you need to change the table column values you can always delete the table by executing `DROP TABLE DCRUsers;` and recreate it by doing step 4 and 5 again).
6. Insert the emails and desired roles for all group members into the database. Use the same email you use for dcrgraphs.net and the REST API authentication and use a role that you have defined in your dcr graph:

```
INSERT INTO DCRUsers VALUES ('{your dcr email}','{the role when executing
events}');
```

Note: The role has to be a role that exists for your own DCR graph. Paste it from your graph into the SQL query above. It will become your default role when we later sync the app with the database.

4. Python + MySQL

1. Now we are ready to test the connection to our app.
2. In your terminal, with your virtual environment active, execute: `pip install mysql-connector-python`

3. Remember to add `mysql-connector-python` in the `pyproject.toml`/`requires` so that beeware imports the pip package correctly.
4. Create a seperate file for the database connection `database_connection.py` under your `services` python package. Like so: 

5. Fill in the `database_connection.py` with the following code and replace the placeholder strings "your db password from azure" , "your db user from azure" and "your db name from azure" with the ones from azure:

```
from mysql.connector import connect

db_password = 'your db password from azure'

sql_query_template = {}

sql_query_template['get_dcr_role'] = f"SELECT Role FROM DCRUsers WHERE
Email = %(email)s"
#TODO: fill in these templates with the right SQL query
sql_query_template['get_dcr_role'] = f""
sql_query_template['update_dcr_role'] = f""
sql_query_template['get_all_instances'] = f""
sql_query_template['get_instances_for_user'] = f""
sql_query_template['insert_instance'] = f""
sql_query_template['insert_instance_for_user'] = f""
sql_query_template['update_instance'] = f""
sql_query_template['delete_instance_from_user_instance'] = f""
sql_query_template['delete_instance'] = f""

def db_connect():
   from pathlib import Path

   resources_folder = Path(__file__).parent.resolve()
   cert_filepath =
str(resources_folder.joinpath("DigiCertGlobalRootCA.crt.pem"))
   cnx = mysql.connector.connect(user="your db user from azure",
                                 password=db_password,
                                 host="your db name from azure
.mysql.database.azure.com",
                                 port=3306,
                                 database="your db name from azure",
                                 ssl_ca=cert_filepath,
                                 ssl_disabled=False)
   print(f'[i] cnx is connected: {cnx.is_connected()}')
   return cnx

def get_dcr_role(email):
```

```
    try:
        cnx = db_connect()
        cursor = cnx.cursor(buffered=True)
        cursor.execute(sql_query_template['get_dcr_role'], {'email':email})
        query_result = cursor.fetchone()[0]
        cursor.close()
        cnx.close()
        return query_result
    except Exception as ex:
        print(f'[x] error get_dcr_role! {ex}')
        return None

def update_dcr_role(email,role):
    try:
        cnx = db_connect()
        cursor = cnx.cursor(buffered=True)
        cursor.execute(sql_query_template['update_dcr_role'], {'role':role,
'email':email}, multi=False)
        cnx.commit()
        cursor.close()
        cnx.close()
    except Exception as ex:
        print(f'[x] error update_dcr_role! {ex}')

def get_all_instances():
    try:
        cnx = db_connect()
        cursor = cnx.cursor(buffered=True)
        cursor.execute(sql_query_template['get_all_instances'])
        query_result = cursor.fetchall()
        cursor.close()
        cnx.close()
        return query_result
    except Exception as ex:
        print(f'[x] error get_all_instances! {ex}')
        return None

def get_instances_for_user(email):
    try:
        cnx = db_connect()
        cursor = cnx.cursor(buffered=True)
        cursor.execute(sql_query_template['get_instances_for_user'],
{'email':email})
        query_result = cursor.fetchall()
        cursor.close()
        cnx.close()
        return query_result
    except Exception as ex:
        print(f'[x] error get_instances_for_user! {ex}')
        return None

def insert_instance(id, valid, email):
    try:
        cnx = db_connect()
```

```
        cursor = cnx.cursor(buffered=True)
        cursor.execute(sql_query_template['insert_instance'],
{'id':id,'valid':valid}, multi=False)
        cursor.execute(sql_query_template['insert_instance_for_user'],
{'email':email,'instance_id':id}, multi=False)
        cnx.commit()
        cursor.close()
        cnx.close()
    except Exception as ex:
        print(f'[x] error insert_instance! {ex}')

def update_instance(id, valid):
    try:
        cnx = db_connect()
        cursor = cnx.cursor(buffered=True)
        cursor.execute(sql_query_template['update_instance'],
{'id':id,'valid':valid}, multi=False)
        cnx.commit()
        cursor.close()
        cnx.close()
    except Exception as ex:
        print(f'[x] error update_instance! {ex}')

def delete_instance(id):
    try:
        cnx = db_connect()
        cursor = cnx.cursor(buffered=True)

cursor.execute(sql_query_template['delete_instance_from_user_instance'],
{'id':id}, multi=False)
        cursor.execute(sql_query_template['delete_instance'], {'id':id},
multi=False)
        cnx.commit()
        cursor.close()
        cnx.close()
    except Exception as ex:
        print(f'[x] error delete_instance! {ex}')
```

7. In your `app.py` file do:

   - at the top add the import `from services import database_connection as dbc`
   - in the `async def login_handler(self,widget):` method after assigning the `self.user` variable:

```
    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)
```

add this:

```
self.user.role = dbc.get_dcr_role(email=self.user.email)
print(f'[i] Role: {self.user.role}')
```

It should print out `[i] cnx is connected: True` and `[i] Role: {your role}` in the terminal after you have started the app and logged in.

8. Proceed to create the remaining SQL queries from the top of the `database_connection.py` file:

```
#TODO: fill in these templates with the right SQL query
sql_query_template['get_dcr_role'] = f""
sql_query_template['update_dcr_role'] = f""
sql_query_template['get_all_instances'] = f""
sql_query_template['get_instances_for_user'] = f""
sql_query_template['insert_instance'] = f""
sql_query_template['insert_instance_for_user'] = f""
sql_query_template['update_instance'] = f""
sql_query_template['delete_instance_from_user_instance'] = f""
sql_query_template['delete_instance'] = f""
```

SQL refresher: https://www.w3schools.com/MySQL/default.asp

Hints:

- What you pass as input to the SQL queries in the `WHERE` part of the query can be read from the input dictionary given in the `cursor.execute` for each of the methods in the file.
- Use the MySQL syntax for `%(...)s` inserting the parameter values where you must replace `...` with the key of the dictionary given in the `cursor.execute` that is specific to the method where the query is used.

For example: We defined the query `SELECT Role FROM DCRUsers WHERE Email = %(email)s` and used `email` for the input parameter as it was given in the input dictionary `{'email':email}` for `cursor.execute` of the `get_dcr_role` method. The query `get_dcr_role` only returns the `Role` column.

- The `get_all_instances` templates require a `SELECT ... FROM ... INNER JOIN ... ON` SQL query. It returns the `InstanceID` and `IsInValidState` from the `Instances` table and `Email` from the `UserSimulations` table.
- The `get_instancess_for_user` template requires a `SELECT ... FROM ... INNER JOIN ... ON ... WHERE ...` SQL query. It returns the `InstanceID` and `IsInValidState` from the `Instances` table. Note that unlike `get_all_instances` this query requires the `email` as input.
- All the `update_` templates require an `UPDATE ... SET ... WHERE ...` SQL query. It does not return a value.
- All the `insert_` templates require an `INSERT INTO ... VALUES (...,...)` SQL query. It does not return a value.

- All the `delete_` templates require a `DELETE FROM ... WHERE ...` SQL query. It does not return a value.

9. Back in your `app.py` add the `dbc` method calls as indicated:

    - In the method `delete_all_instances` only delete the instances for the logged in user that are in a valid state (instances that are not pending). You must first check that the instance is valid from the database, then you must delete the instance in the DCR Active Repository and finally the database entry for the instance.
    - In the method `delete_instance_by_id` you are allowed to delete invalid instances, after you have deleted them from the DCR Active Repository.
    - In the method `create_new_instance` you must insert the new instance with the correct `valid` state according to the DCR acceptance criteria. You must check using the DCR Active Repository that none of the events are initially pending.
    - In the method `role_changed` you must update the user role in the database.
    - In the method `execute_event` you must update the `valid` state of the instance in the database (by checking whether or not there are pending events after executing the event).
    - In the method `show_instances_box` you must only add to `self.instances` those instances that also exist in the database irrespective of the logged in user. You must create a list called `my_instances` in which you only append the instances belonging to the currently logged in user. You must disable any button for which the instance `id` is not in `my_instances` .