## **ACIT 2515 – Object Oriented Programming – Final assignment**

**This is a group assignment. You will complete it with your partner(s).**

**Due date: Friday 10 April at 6PM. NO LATE SUBMISSIONS ACCEPTED.**

**Note:**You will need to set up a time to meet and demo your application and answer questions. This must happen before Friday 6 April, 6PM. We will meet on Zoom for the demos.

Please organize it as soon as possible - there is a limited number of hours per day. I will not accept late submissions. Students who are not present during their group demo will receive 0.

**Goals**

* To persist (i.e., store) your entities to a Sqlite database.
* To map your entities to database tables using an ORM.
* To create a Graphical User Interface (GUI) in Tkinter that integrates with your RESTful API using the Python requests package. The GUI will allow users to create, update, delete and query the entities stored in the database.

**Overview**

In the previous assignment, you implemented persistence using JSON files. For this final part you will change the persistence engine from JSON files to a SQLite database. You can use **peewee** to declare your models and interact with the database. Remember that attributes on peewee declaratives are typically public.

Notes:

* Preliminary initialization is no longer needed as the entities come from the database
* You might need to pass around the db object from peewee. Put it in a separate module to avoid circular dependencies.
* The \_read\_entities\_from\_file and \_write\_entities\_to\_file methods are no longer needed. Reading and saving can now be done from / to the database.
* DB models have public attributes, so getter and setter methods are not required anymore.

**Database persistence (15 marks)**

For this assignment you will store your entity records in a single table in a Sqlite database. Design your database table so that it can store the attributes for both entities.

A few hints on database design:

* Think about the size of your data (ex varchar columns). Make sure they are big enough to hold your data. Make sure the parameter validation in your entity class constructors reject values that are too big and cannot be stored properly in the database.
* Mandatory columns (i.e., not nullable). Note that the attributes specific to your child entity classes must be nullable because they will not be populated for the other entity type. If they are mandatory, that must be enforced through parameter validation – both in the constructor in the db model and the GUI.

You need to create the following files:

* create\_tables.py – This script will create your table and all the required columns to support both your specific entity types, including a column to hold the type.
* drop\_tables.py – Drops your table.

You will be using **peewee** as the ORM to interact with your database table. Please refer to previous labs in order to setup your objects.

You need to update the following files:

* Your specific entity classes. These must be re-written as peewee db models. They should have all the same helper methods as before including a constructor with parameter validation.
* Your entity manager class. This must be re-written to use peewee to create the connection to your Sqlite database and all methods refactored to create, update, delete and query the database using SQLAlchemy.
  + All previous public methods remain, just the implementation changes
  + All previous parameter validation and exceptions must remain

**Unit Testing of the entity manager (5 marks)**

Update the unit test for your entity manager to work with your refactored entity manager class.

There should NOT be significant changes to the unit tests, since your public interface is largely unchanged, except:

* Your setUp method should create a test sqlite database (it should have a different name than the one used in your application).
* The tearDown method should delete the test sqlite database.
* Note: you may want to use the setUpClass and tearDownClass methods to set up test data. setUp / tearDown are called before / after each individual test method, whereas setUpClass / tearDownClass are called once before all test methods are run.
* You no longer need the mocks (i.e., the @patch decorations) on your test methods because we are using a dedicated test database for our unit tests (ie we control and accept side effects).

**Graphical User Interface (10 marks)**

Create a GUI for your entities that supports the following:

* Displays a summary list of the specific entities (stored in your database) and the entity statistics
  + User may toggle the list of specific entities between each of the two types
  + The entity stats are displayed adjacent to the summary list
  + The summary list and entity stats automatically update if an entity is added, updated or deleted
* Displays the full details of a single entity upon selection from the list above (both types supported)
* Add a new entity of each type
* Update an existing entity (both types supported)
* Delete an existing entity (both types supported)

Make sure the GUI displays errors returned from the backend in a user friendly manner. It should not just display the error code (i.e., 200, 400, 404) to the user. It should display messages like:

* Employee successfully added
* Employee could not be added because data is missing or invalid
* Etc.

The GUI must call the API endpoints of your RESTful API for each of the above actions using the *requests* Python package. It must NOT integrate directly with your entity manager – all interaction between the GUI and the backend should be through the RESTful API.

Grading on the GUI will be on both the functionality AND the usability. If GUI is unorganized or difficult to use, you will lose marks.

Note that to run and test your application, both the RESTful API main application (i.e., employer\_api.py) and the GUI main application (i.e., employer\_gui.py) must be running at the same time.

**Submission**

**Your code must be in your Git repo with a specific revision id (submitted to D2L). Otherwise you will receive zero marks.**

Your Git repo should include the following files:

* + All the code necessary to run your two applications, including the two main applications:
    - RESTful API module (i.e., employer\_api.py)
    - GUI module (i.e., employer\_gui.py)
  + Unit test class (i.e., test\_employer.py)

Your D2L submission should include:

* Revision ID (i.e., commit id) and URL to the Git repository with your code

**Demonstation**

* You must arrange a time to demo your running application to your instructor on or before 6 April 2020, 6PM.
* You must be running the code from your Git repo on the same revision as submitted.
* Only the team members present for the demo will receive marks.
* All team members are expected to be able to run the program and answer questions.
* The demo should take approximately 5-10 minutes (max 20).
* See the Grading Summary for the specific items to demo

**Grading Summary**

|  |  |
| --- | --- |
| Database Persistence Updates   * Database create/drop scripts (2 marks) * Updates to the Specific Entity classes (4 marks) * Refactored Entity Manager class (4 marks)   **If the REST API does not run, you get 0 marks for this section.** | 10 marks |
| Unit Test Update   * EntityManager test updates to use the new DB models (1 marks) * update tests to use temporary SQLite DB (2 marks)   **If your unit tests do not run or have failures, you get 0 marks for this section.** | 3 marks |
| Unit Test and GUI Demo   * Code is running from your Git repo (1 mark) * Unit Tests all pass (1 mark) * Add Specific Entity 1 (1 mark) * Add Specific Entity 2 (1 mark) * Entity Summary List, including auto update (2 marks) * Entity Statistics, including auto update (1 mark) * Entity Details Specific Entity 1 (1 mark) * Entity Details Specific Entity 2 (1 mark) * Update Specific Entity 1 (1 mark) * Update Specific Entity 2 (1 mark) * Delete Entity (1 mark) * Test one error condition, i.e., error message for invalid data entry (1 mark) * Show the updated database table through DB Browser (1 mark)   **If the GUI does not run, you will receive 0 for this section. If the GUI has missing features, you will receive at most 8 marks (50% penalty)** | 14 marks |
| UX / ergonomy of the GUI   * Text is easy to read * Buttons are logically placed * Usage flow is logical and consistent | 3 marks |
| **Total** | **30 marks** |

Marks will be subtracted for violations of best practices covered so far in this course (i.e., naming, DocString, constants for magic numbers, etc).

There will be bonus marks if you implement additional complex features or pay extra attention to the look'n'feel of your application.

**No late submissions will be accepted. If you do not demo your program, you will get 0.**