## **ACIT 2515 – Object Oriented Programming – Assignment 3**

**You will be continuing this assignment with the same code and partner as Assignment 2.**

**Goals**

* To build a RESTful API using JSON and Python Flask
* To persist your entity data using JSON

**Work on entities:**

* Make sure your entities have an attribute "\_id" that can be used to uniquely reference them. For a person, it may be their name, for an apartment - the address, for a car - the license plate, etc. It is much easier if this attribute is an integer, but it can also be a string.
* In your abstract entity, create a **to\_dict()** method that returns the instance state as a dictionary.
* Make this method an abstract method and implement it in your child classes.

**Work on the entity manager:**

* Your entity manager will now persist data - which means that it can save its state in a file on disk, and load entities from the disk.
* The data will be persisted in a file, using JSON format.
* The entity manager must have a new attribute: "\_filepath". This attribute contains the path to the JSON file that will be used to read / write entities from / to the file.
* Create two new methods:
  + **\_read\_from\_file**() will be called from your \_\_init\_\_ method. It will open the file located at \_filepath, read its contents as JSON, and create entities accordingly.
  + **\_write\_to\_file()** will be called from each method that modifies either the entities or the entity manager. It will export the entities as a JSON serialized list in the file specified at \_filepath.

**Work on your unit tests:**

Update your unit tests as follows:

* SpecificEntity1 and SpecificEntity2 – Add a unit test to verify the new to\_dict() method
* EntityManager – Modify your file as follows:
  + **\_\_init\_\_** – Add tests for parameter validation on the filepath parameter
  + **\_read\_from\_file** – **Mock** this method so it does nothing, but make sure it is called upon construction of EntityManager.
  + **\_write\_to\_file** – **Mock** this method so it does nothing. But make sure it is called on each method in EntityManager that updates an Entity (i.e., add, update, delete).

**Create a RESTful API:**

Create a RESTful API for your EntityManager class with the following API endpoints. Update the naming of your API resources based on the names of your EntityManager and Entity classes (but don’t use abstract in the entities resource name).

**You must provide example JSON files (see the Submission below) for the add (POST) and update (PUT) API endpoints otherwise you will lose 50% of your marks on this section.**

**POST /entitymanager/entities**

* Request: Entity JSON
* Response: Id of new Entity OR error message
* Returns 200 OK on success
* Returns 400 and an error message if the entity object is invalid

**PUT /entitymanager/entities/<entity\_id>**

* Request: Entity JSON (without id)
* Response: None OR error message
* Returns 200 OK on success
* Returns 404 and an error message if the entity object does not exist

**DELETE /entitymanager/entities/<entity\_id>**

* Request: None
* Response: None OR error message
* Returns 200 OK on success
* Returns 404 and an error message if the entity object does not exist

**GET /entitymanager/entities/<entity\_id>**

* Request: None
* Response: Entity JSON OR error message
* Returns 200 OK on success
* Returns 404 and an error message if the entity object does not exist

**GET /entitymanager/entities/all**

* Request: None
* Response: List of Entity JSON (can be empty)
* Returns 200 OK on success

**GET /entitymanager/entities/all/<type>**

* Request: None
* Response: List of Entity JSON (can be empty) of the given type OR error message
* Returns 200 OK on success
* Returns 400 and an error message if the type is not supported

**GET /entitymanager/entities/stats**

* Request: None
* Response: Entity Statistics JSON
* Returns 200 OK on success

**Make sure you test file persistence. If you shutdown or restart the Flask server, your Entity objects should be available when it is started again.**

To ensure you thoroughly test your APIs, take two screenshots for each API endpoint:

* 1 success – 200 OK
* 1 error – 400 or 404 if relevant

Make sure the Method, URI, Request, Response and Response Code is clearly visible in the screenshot. Combine all your screenshots in a single PDF document.

**If you don't combine the screenshots or submit them in a different format than PDF, you will get 0 for the RESTful section.**

**Submission**

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

**Deliverables - JSON**

JSON: Design the JSON structure for each of your SpecificEntity classes. Create sample JSON for the following:

* SpecificEntity1 – Add
* SpecificEntity1 – Update
* SpecificEntity2 – Add
* SpecificEntity2 – Update

Use a **type** field in the JSON to distinguish each SpecificEntity so the API code can create the correct object (SpecificEntity1 or SpecificEntity2).

***If you do not provide valid example JSON, you will lose 50% of your marks on your total grade for this assignment because your marker will be unable to test it. Make sure they are valid and work with your RESTful API.***

***Be careful when using variables: id and type are Python built-in functions, so do not use them.***

**Deliverables - GitHub repository**

Your Git repo should contain (at least) the following files:

* Updated UML Design in source format
* JSON samples for add and update:
  + in a json\_samples folder
  + add1.json, add2.json
  + update1.json, update2.json
* Screenshot in ONE PDF file
* Entity Manager Class (i.e., employer.py)
* Entity Manager Unit Test (i.e., test\_employer.py)
* Abstract Entity Class (i.e., abstract\_employee.py)
* Specific Entity 1 Class (i.e., manager.py)
* Specific Entity 1 Unit Test (i.e., test\_manager.py)
* Specific Entity 2 Class (i.e., engineer.py)
* Specific Entity 2 Unit Test (i.e., test\_engineer.py)
* RESTful API Module (i.e., employer\_api.py)

Your D2L submission should include your revision ID (i.e., commit id) and URL to the Git repository with your code.

**Grading Summary**

|  |  |
| --- | --- |
| Design planning   * JSON examples | 2 marks |
| File persistence   * Read on construction (2 marks) * Write on updates (2 marks) * JSON Storage (1 mark) | 5 marks |
| Unit tests   * EntityManager test updates (4 marks)   + Includes mocking * SpecificEntity test updates (1 mark) | 5 marks |
| RESTful API   * 3 marks for each endpoint with an error response * 1 mark for each endpoint without an error response | 15 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).