Data Store Challenge:

=================

API Task:

————

The folder *’database\_operations’*  consists of Data bases utility module and SQL ORM Schema mapper module.

Under the folder *‘SmartFab’ ,* File: “*user\_store\_api.py”* contains the API to access the store.

In order for the solution to be scalable, that is if in future any change in the DB technology or the implementation, the user must still be able to use the same API. I have implemented an interface class - ‘iMetaStore’ in file *‘store\_interface.py’* . This module contains abstract methods. These abstract method logic are implemented in the ‘*user\_store\_api.py*’. The user will have to import *user\_store\_api* module to access the meta data store. The usage of interface will allow to match the signatures of the method defined in the interface and the API

The API names are simple to understand and are highly correlated with their functionality. Two features are implemented, Type 1 and Type 2 in the challenge. The API’s are not generic. They are specific to the requirements of the user.

*‘test.py’ :* Consists of sample on importing the API , invoking the functionalities and print the results.

Any new API’s can be added in the interface file and made available for the user.

With respect to performance, if the DB does not reside on the host computer, there could be a latency observed in the query/update process. The queries are made using ORM, sqlalchemy internally converts to SQL statements and pushes it to the DB. In the case of heavy or large database rows per request ORM is costly. The internal caching capability uses memory unfairly. On the other hand for smaller number of rows the ORM is relatively faster. In the later case, we could replace ORM with traditional SQL mapping.

Architecture:

==========

SQL Object Relational Mapper

(*sql\_orm.py*)

Data Base Utility

(*db\_utilities.py*)

Interface/Template

(*store\_interface.py*)

API

(*user\_store\_api.py*)

User

Meta Data Store

(MySQL)