**Student View**

This document contains details about the functionality and various modules of the student view. It also contains details of the models,views and templates used by the student view.

**Functionality :**

* A way for students to view their progress and grades in the course.
* The list of modules they are proficient in.
* The list of exercises they are proficient in.
* Student’s score based on the exercises they are proficient in.
* The list of exercises the student needs to be proficient in, to be proficient in a module.
* A list of non-proficient and uncompleted exercises.

**Technical Details :**

The ‘controller’(in Model View Controller) code for the student view is present in the file Aalto/aaltoplus/opendsa/views.py. ‘**module\_list**’ is the function that renders the student view. The data for the controller comes from file models.py which contains a set of classes for each table in the database. Each class has a set of member functions for performing certain utility operations(example : to check if the user is proficient in a exercise). Since the model fetches all the data from the corresponding tables, filtering of data at the controller level (views.py) proves to be a costly affair. For example, to extract all the entries for a particular user from table ‘user\_exercise’, initially we used the filter function *userExercise.objects.filter* which does the filtering after the data is returned from the database. To improve the performance the controller calls static member functions of respective classes in models.py. These methods use cursor to execute appropriate stored procedure and returns the result. o execute a query we use django.db.connection.connection.cursor() to get a cursor object. Then, we execute the query through statement cursor.execute(sql, [params]). Static methods inside the models and procedures are not the same. Static methods inside the class in models will call the stored procedure available in the database.

The various stored procedures that needs to be run in the database are :

*Delimiter $$*

*Create PROCEDURE GetUserExerciseByUser\_Id(in userId INTEGER) Begin Select \* from opendsa\_userexercise where user\_id = userId; End $$*

*Create PROCEDURE GetUserModuleByUser\_Id(in userId INTEGER) Begin Select \* from opendsa\_usermodule where user\_id = userId; End $$*

*Create PROCEDURE GetUserDataByUser\_Id(in userId INTEGER) Begin Select \* from opendsa\_userdata where user\_id = userId; End $$*

The use of stored procedures instead of using data from the models directly helped in reducing the time taken for the page to load. The page now loads in about a second as compared to 20 seconds that it took when data was used from the django models.

The classes used in **views.py** are

1)Useruiexec (Fields : exercise, prof, points)

This class renders the exercises under the modules.

exercise : Contains the exercise object

prof : indicates the proficiency - if the person is proficient or not.

points : indicates points scored by the user. If a student is proficient in a exercise then based on the exercise type, value is set to this field.

2)Profuiexec (Fields : userexercise, points)

This class renders the proficiency exercises.

userexercise : contains the userexercise object which has all the data from the table user\_exercise.

points : indicates points scored by the user. If a student is proficient in a exercise then based on the exercise type, value is set to this field.

3)Useroutputmodule (Fields : name, covers, authors, prof, countexec, prerequisites, userexecs)

This class renders the modules in the ‘Module Summary’ tab. It contains information about the module along with the exercises under it.

name : name of the module

covers : concepts which the module covers

authors : various authors of this module.

prof : this field is used to indicate if the module is proficient or not.

prerequisites : contains details of the modules which must be completed before starting the current module.

userexecs : a array of exercises which the module contains.

countexec : contains the count of number of exercises inside a module.

4)Function module\_list(request):

The function module\_list takes the request object as a parameter, reads the session id from the cookies and uses the session id to get the information about the logged in user. The cookies field in the request object is a dictionary. So request.cookies[“sessionid”] contains the session id. With the user id the function fetches the data from the database. The function first fetches the list of all modules and the list of all module entries for the logged in user and maps them to decide the proficiency of the user in various modules. It then fetches all the exercises and inserts apporpriate exercise inside each ‘Useroutputmodule’ object. The function then constructs objects for proficient, nonproficient exercises, and exercises to be taken. These objects are then written out to the response stream and passed to the template. Thus the response stream now has an array of ‘Useroutputmodule’ objects which corresponds to the tab ‘Module Summary’, a array of ‘Profuiexec’ objects corresponding to proficient exercises, and an array of ‘userexercise’ and ‘exercise’ objects for non-proficient and proficient exercises respectively.

**The template file : module\_list.html:**

This file contains the html code for the student view. It reads the entries from the objects written out to the response and constructs the DOM elements. The interactions inside the page such as switching tabs or expanding modules/exercises are achieved through jQuery.