Yaksh Documentation

Release April 2020

FOSSEE

Apr 07, 2020

Contents

Yaksh lets user create and take online programming quiz. Yaksh is an open source project developed by FOSSEE. The code is available on github.

The user documentation for the site is organized into a few sections:

CHAPTER 1

Introduction

This project provides an "exam" app that lets users take an online programming quiz.

1.1 Features

- Easy installation and cross platform support for Linux and Mac OS.
- Define fairly complicated programming problems and have users solve the problem.
- Immediate verification of code solution.
- Supports pretty much arbitrary coding questions in Python, C, C++, Java, Scilab and simple Bash and uses "test cases" to test the implementations of the students.
- Supports simple multiple choice questions, fill in the blanks questions and assignment evaluations.
- Since it runs on Python, you could technically test any Python based library.
- Completely safe from malicious code.
- Scales to over 500+ simultaneous users.
- Distributed under the BSD license.

CHAPTER 2

Installation

2.1 Requirements

Python 3.6, 3.7, 3.8

Django 3.0.3

2.2 Installing Yaksh

If Python 3.6 and above is not available in the system, then we recommend using miniconda

Installing Miniconda

1. Download miniconda from https://docs.conda.io/en/latest/miniconda.html according to the OS version.
2. Follow the installation instructions as given in https://conda.io/projects/conda/en/latest/user-guide/install/index.html#regular-installation
3. Restart the Terminal.

Pre-Requisite

- Ensure pip is installed

Installing Yaksh

1. **Clone the repository**:

```
git clone https://github.com/FOSSEE/online_test.git
```

2. **Go to the online_test directory**:

```
cd ./online_test
```

3. **Install the dependencies**:

- Install Django and dependencies:

```
pip install -r ./requirements/requirements-common.txt
```

- Install Code Server dependencies:

```
sudo pip3 install -r requirements/requirements-codeserver.txt
```

2.3 Quick Start

1. **Start up the code server that executes the user code safely:**
    - To run the code server in a sandboxed docker environment, run the command:

```
$ invoke start
```

Note: Make sure that you have Docker installed on your system beforehand. Find docker installation guide here.

    - To run the code server without docker, locally use:

```
$ invoke start --unsafe
```

Note: Note this command will run the yaksh code server locally on your machine and is susceptible to malicious code. You will have to install the code server requirements in sudo mode.

2. **On another terminal, run the application using the following command**
    - To start the django server:

```
$ invoke serve
```

Note: The serve command will run the django application server on the 8000 port and hence this port will be unavailable to other processes.

3. **Open your browser and open the URL** - `http://localhost:8000/exam`
4. **Login as a teacher to edit the quiz or as a student to take the quiz**
    - **Credentials:**

        For Student:

        - Username: student
        - Password: student

        For Teacher:

        - Username: teacher
        - Password: teacher

5. **User can also login to the Default Django admin by going to URL and entering the following admin credentials `http://`**

   For admin:

   - Username: admin
   - Password: admin

2.4 Production Deployment

- **Deploying Locally**

  Follow these steps to deploy locally on the server.

  - **Pre-Requisite**

    1. Ensure pip is installed
    2. Install dependencies, Run;

    ```
    pip3 install -r requirements/requirements-py3.txt # For Python 3.x
    ```

    3. Install MySql Server
    4. Install Python MySql support
    5. Install Apache Server for deployment
    6. Create a database named `yaksh` by following the steps below

    ```
    $> mysql -u root -p
    $> mysql> create database yaksh
    ```

    7. Add a user named `yaksh_user` and give access to it on the database `yaksh` by following the steps below

    ```
    mysql> grant usage on yaksh to yaksh_user@localhost identified
    by 'mysecretpassword';

    mysql> grant all privileges on yaksh to yaksh_user@localhost;
    ```

    8. Add `DATABASE_PASSWORD = 'mysecretpassword'` and `DATABASE_USER = 'yaksh_user'` to online_test/settings.py

  - **Installation & Usage**

    To install this app follow the steps below:

    1. Clone this repository and cd to the cloned repo.

    ```
    $ git clone  https://github.com/FOSSEE/online_test.git
    ```

    2. Rename the `.sampleenv` to `.env`
    3. In the `.env` file, uncomment the following and replace the values (please keep the remaining settings as is);

```
DB_ENGINE=mysql # Or psycopg (postgresql), sqlite3 (SQLite)
DB_NAME=yaksh
DB_USER=root
DB_PASSWORD=mypassword # Or the password used while creating␣
↪a Database
DB_PORT=3306
```

4. Run:

```
$ python manage.py makemigrations yaksh

$ python manage.py migrate yaksh
```

5. Run the python server provided. This ensures that the code is executed in a safe environment. Do this like so:

```
$ sudo python3 -m yaksh.code_server # For Python 3.x
```

Put this in the background once it has started since this will not return back the prompt. It is important that the server be running *before* students start attempting the exam. Using sudo is necessary since the server is run as the user "nobody". This runs the number ports configured in the settings.py file in the variable "N_CODE_SERVERS". The "SERVER_TIMEOUT" also can be changed there. This is the maximum time allowed to execute the submitted code. Note that this will likely spawn multiple processes as "nobody" depending on the number of server ports specified.

You can also use a Dockerized code server, see *Dockerized Code Server*

6. The `wsgi.py` script should make it easy to deploy this using mod_wsgi. You will need to add a line of the form:

```
WSGIScriptAlias / "/online_test/wsgi.py"
```

to your apache.conf. For more details see the Django docs here:

https://docs.djangoproject.com/en/2.0/howto/deployment/wsgi/

7. Create a Superuser/Administrator:

```
python manage.py createsuperuser
```

8. Go to http://desired_host_or_ip:desired_port/exam

And you should be all set.

9. Note that the directory "output" will contain directories, one for each user. Users can potentially write output into these that can be used for checking later.

10. As a moderator you can visit http://desired_host_or_ip/exam/monitor to view results and user data interactively. You could also "grade" the papers manually if needed.

- **Using Dockerized Code Server**

1. Install Docker

2. Go to the directory where the project is located

```
cd /path/to/online_test
```

3. Create a docker image. This may take a few minutes,

```
docker build -t yaksh_code_server -f ./docker/Dockerfile_codeserver
```

4. Check if the image has been created using the output of `docker images`
5. Run the invoke script using the command `invoke start` The command will create and run a new docker container (that is running the code_server.py within it), it will also bind the ports of the host with those of the container
6. You can use `invoke --list` to get a list of all the available commands

- **Deploying Multiple Dockers**

  Follow these steps to deploy and run the Django Server, MySQL instance and Code Server in seperate Docker instances.

  1. Install Docker
  2. Install Docker Compose
  3. Rename the `.sampleenv` to `.env`
  4. In the `.env` file, uncomment all the values and keep the default values as is.
  5. Go to the `docker` directory where the project is located:

```
cd /path/to/online_test/docker
```

  6. Build the docker images

```
invoke build
```

  7. Run the containers and scripts necessary to deploy the web application

```
invoke begin
```

  8. Make sure that all the containers are `Up` and stable

```
invoke status
```

  8. Run the containers and scripts necessary to deploy the web application, `--fixtures` allows you to load fixtures.

```
invoke deploy --fixtures
```

  10. Stop the containers, you can use `invoke restart` to restart the containers without removing them

```
invoke halt
```

  11. Remove the containers

```
invoke remove
```

  12. You can use `invoke --list` to get a list of all the available commands

- **Additional commands available**
  - **create_moderator** : Use this command to make a user as moderator.

```
python manage.py create_moderator <username>
```

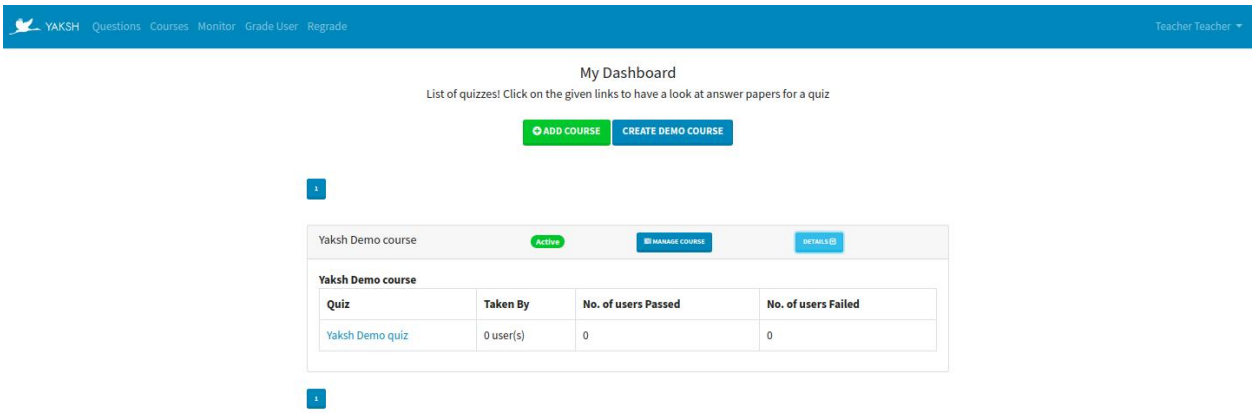
For more information on the command:

```
python manage.py help [command-name]
```

CHAPTER 3

Moderator's Dashboard

On logging in moderators see the following dashboard.



There are two options available:

- **Add Course** It allows to create a new course.
- **Create Demo Course** It creates a demo course contaning sample lesson and quiz with questions.

The dashboard contains all the courses. Each course provides two options

- **Manage Course** Click on this button to manage the course. See the Manage course section in the *Courses* page for more details.
- **Details** Clicking on the Details button shows all the quizzes in the course. Click on the quiz link to monitor the quiz.

The following pages explain the various functions available for moderators

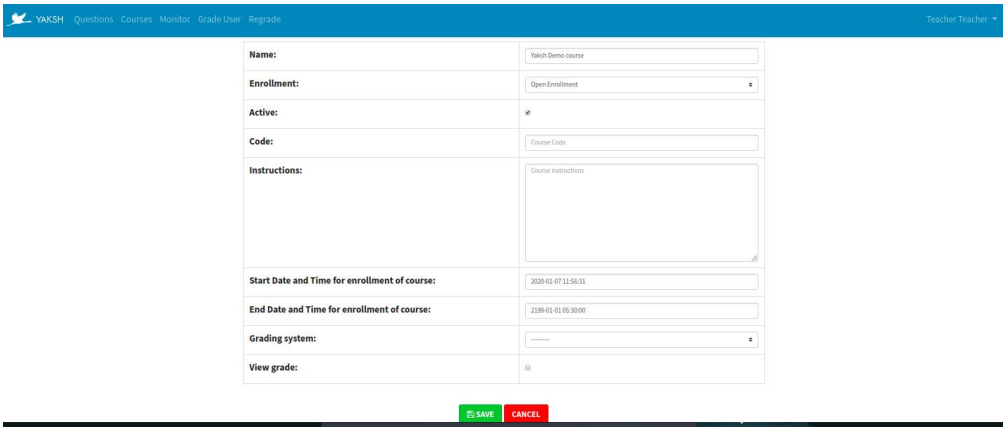
3.1 Courses

For students to take a quiz, it is imperative for the moderator to create a course first. A course can contain several modules and a module can contain several lessons and/or quizzes.

To create modules, lessons and quizzes go to the *Lessons and Modules* and *Quizzes* section of the documentation.

3.1.1 Setting up a new course

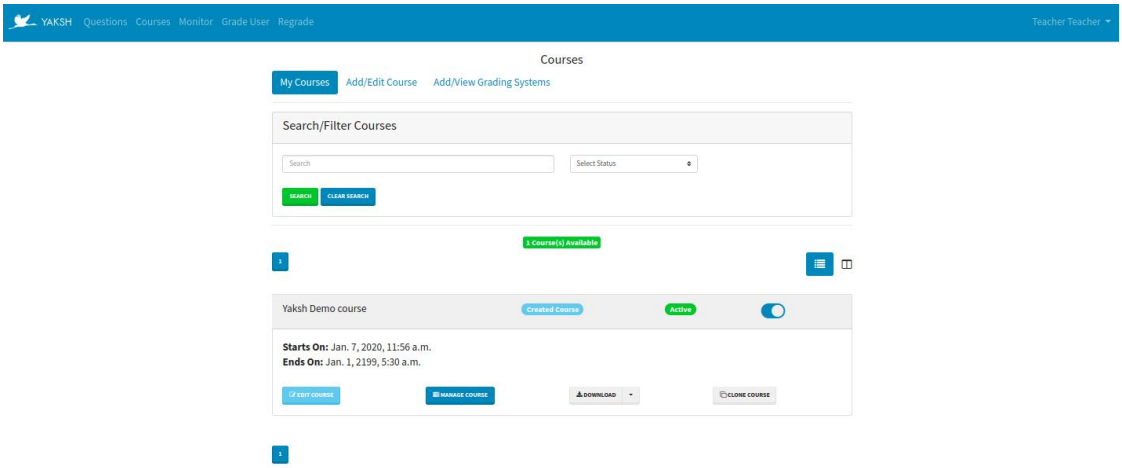
To create a course, click on the Add New Course button on the moderator's dashboard. This will lead you to a create add course page, where you need to fill in the following fields.



- **Name** Name of the Course
- **Enrollment** Open enrollment is open to all students. Enroll Request requires students to send a request which the moderator can accept or reject.
- **Active** If the course should be active for students to take the quiz. The status of the course can be edited later.
- **Code** If the course should be hidden and only accessible to students possessing the correct course code.
- **Instructions** Instructions for the course.
- **Start Date and Time for enrollment of course** If the enrollment of the course should be available only after a set date and time.
- **End Date and Time for enrollment of course** If the enrollment of the course should be available only before a set date and time.
- **Grading System** Add a grading system to the course.
- **View Grade** This field allows the student to view the grade if checked else grade is not visible to student.

3.1.2 Features in Courses

Click on the Courses link on the navigation bar.



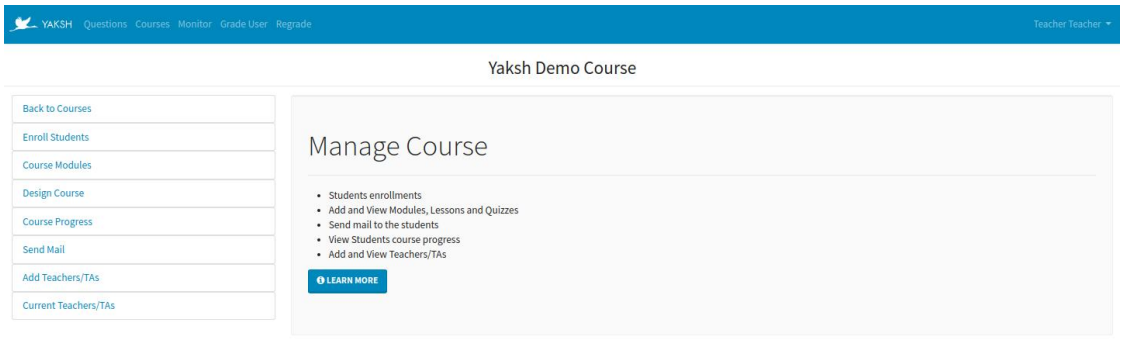
This page shows all the courses created by a moderator and all the courses allotted to a moderator.

The following options are available in the courses page

- **My Courses** Click to show all the courses created by you.
- **Add/Edit Course** Click to add the details of a new course.
- **Add/View Grading Systems** Add or view grading systems. More info on creating grading system
- **Search/Filter Courses** Search the courses by name or filter the course with active and inactive status.
- **Course Name** Shows course name of all the created and allotted courses.
- **Edit Course** Click this button to edit the corresponding course details.
- **Manage Course** This provides more options for the course. For e.g. setting up modules, lessons, quizzes, practice exercises, students enrollments etc.
- **Download** This button provides two options. One is to download the course CSV containing student data, Other is to download entire course for offline viewing.
- **Clone Course** Click to create a copy of a course along with its modules, lessons and quizzes.
- **Activate/Deactivate Course** Toogle to activate or deactivate the course.

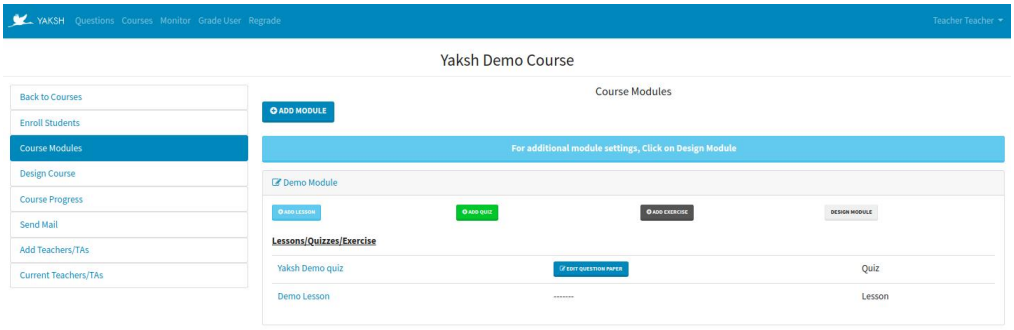
3.1.3 Manage Course

Click on the Manage course button to view the course details page.



Following are the features for course details -

- **Enroll Students**
  - **Upload Users** Create and enroll users automatically by uploading a csv of the users. The mandatory fields for this csv are - **firstname, lastname, email**. Other fields like **username, password, institute, roll_no, department, remove** fields are optionals.
  - **Requests** This is a list of students who have requested to be enrolled in the course. Moderator can enroll or reject selected students.
  - **Enrolled** This is a list of students who have been enrolled in the course. Moderator can reject enrolled students.
  - **Rejected** This is a list of students who have been rejected for enrollment in a course. Moderator can enroll rejected students.
- **Course Modules** Moderator can send mail to all enrolled students or selected students.



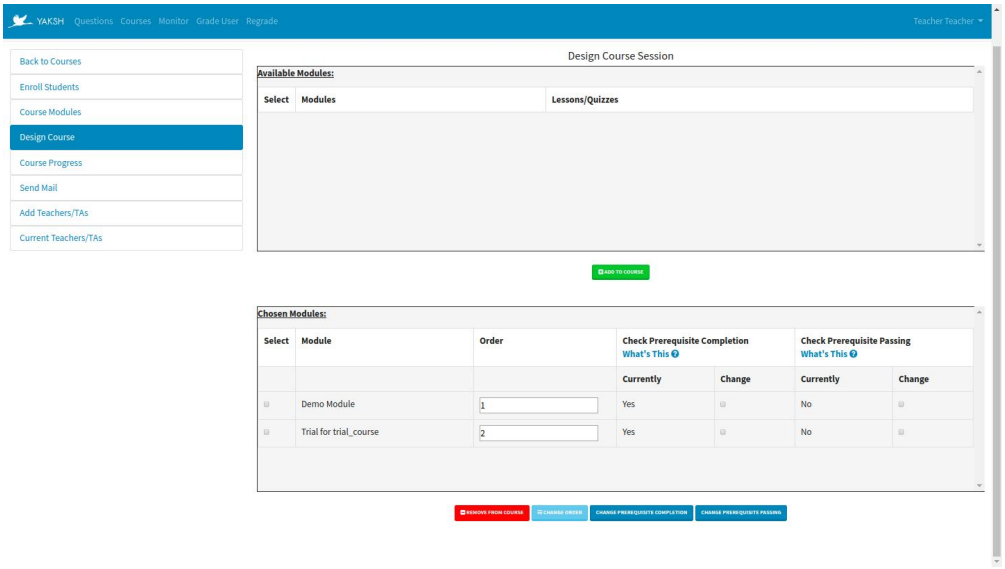
  - **Add Module** Click on this button to add a module to the course. Fill the details of the module and save it.

    After creating a module for the course, following options are available:

    * **Add Lesson** Add lesson to the corresponding module.
    * **Add Quiz** Add a graded quiz to the correspoding module.
    * **Add Exercise** Add a ungraded practice exercise to the corresponding module.

  * **Design Module** This option allows you to change the order of the units added to the module, check for prerequisites of the module and remove a unit from the module.

- **Design Course** Clicking on **Design Course** will show the below page.



  - **Available Modules** contains all the modules that are not added to a course.

    To add a module to the course select the checkbox besides the desired module to be added and click **Add to course** button.

  - **Chosen Modules** contains all the modules that are added to a course.

    Following parameters can be changed while designing a course:

    * **Order** Order in which modules are shown to a student.

      To change a module's order change the value to a desired order in the textbox under **Order** column and click **Change order**.

    * **Check Prerequisite Completion** Check if previous module is completed. Default value is **Yes**.

      For e.g., Assuming a course contains modules **Demo Module** and **Trial for trial_course** in the given order; a student has to first complete **Demo module** to attempt **Trial for trial_course** if the **Check Prerequisite** value for **Trial for trial_course** is checked **Yes**.

      Currently column shows the current value of **Change Prerequisite Completion** which in this case is **Yes**.

      Select the checkbox from **Change** column under **Check Prerequisite Completion** and click **Change Prerequisite Completion** button to change the value.

    * **Check Prerequisite Passing** Check if previous module is completed. Default value is **Yes**. This is similar to **Check Prerequisite Completion** except that it checks if all the quizzes in the module are passed or not.

      Currently column shows the current value of **Change Prerequisite Passing** which in this case is **Yes**.

Select the checkbox from **Change** column under **Check Prerequisite Passing** and click **Change Prerequisite Passing** button to change the value.

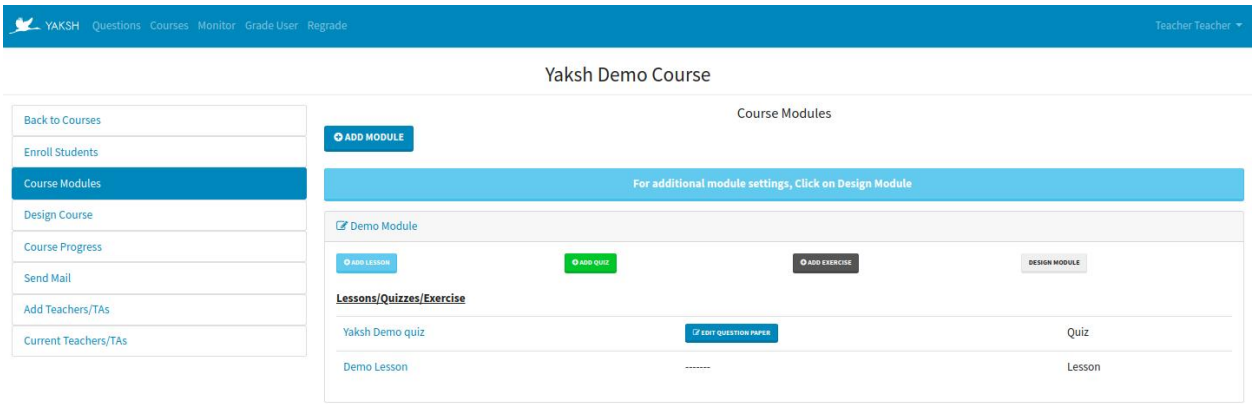
* **Remove Module** To remove a module from the course select the checkbox beside every module and click **Remove from course** button.

- **Course Progress** It shows progress made by the students in the course. Moderator can also download the course progress data.
- **Send Mail** Moderator can send mail to all enrolled students or selected students.
- **Add Teachers/TAs** Moderator can search for the users by username, email, first name and last name to add as Teacher/TA to the course.
- **Current Teachers/TAs** It shows all the added Teachers/TAs to the course. Added users can view and edit the course, modules, lessons and quizzes available in the course.

3.2 Quizzes

Quizzes are intrinsically associated with a course, hence to view and/or edit a quiz, we need to navigate to the course details page by clicking on **Manage Course** button.

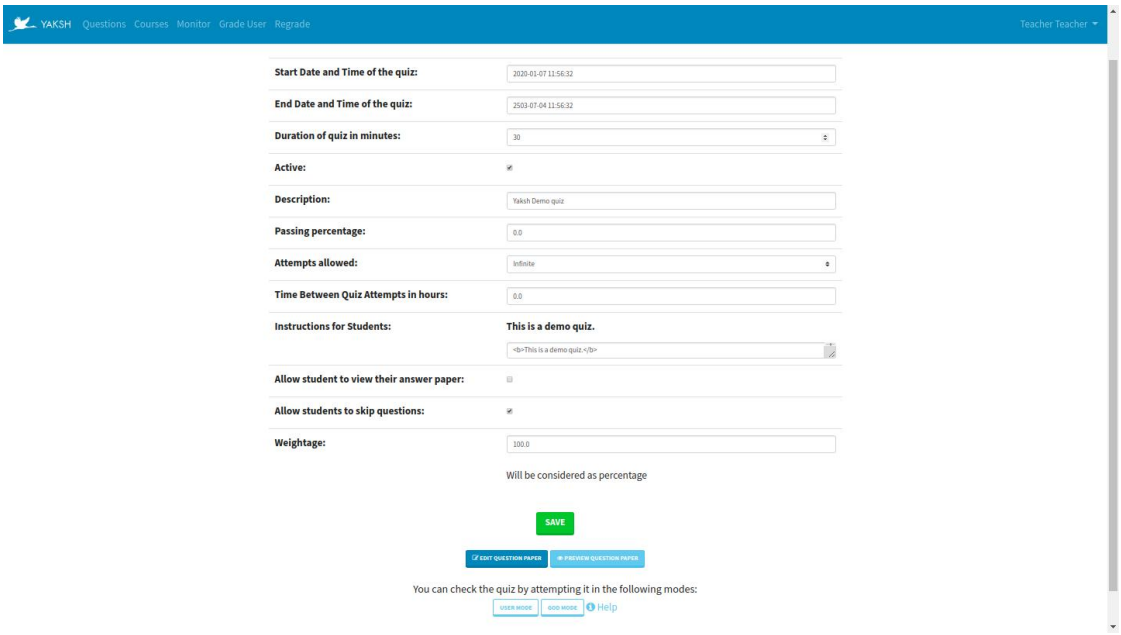
Clicking on **Course Modules** in the course details page will open the page as shown below



This page shows all the modules with quizzes and exercises.

3.2.1 Creating a Quiz

Click on **Add Quiz** button to add a quiz.



Note: It is important to have created or uploaded questions before creating a quiz.

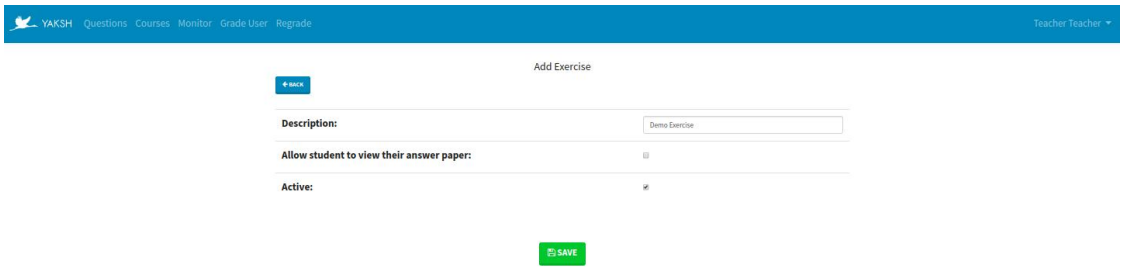
- **Start Date and Time of quiz** - The date and time after which the quiz can be taken.
- **End Date and Time of quiz** - The date and time after which the quiz is deactivated and cannot be attempted.
- **Duration** - Duration of quiz to be written in minutes.
- **Active** - Check the checkbox to activate/deactivate quiz.
- **Description** - Description or name of the quiz.
- **Passing Percentage** - Minimum percentage required to pass the test.
- **Attempts allowed** - Number of attempts that a student can take of the current quiz.
- **Time Between Quiz Attempts in hours** - For a quiz with multiple attempts this value can be set so that student can attempt again after the specified time.
- **Instructions for students** - Additional instructions for students can be added. Some default instructions are already provided.
- **Allow student to view answer paper** - Click on this checkbox to allow student to view their answer paper.
- **Allow student to skip questions** - Click on this checkbox to allow/disallow student to skip questions for a quiz. Value defaults to allow skipping questions.
- **Weightage** - Every quiz will have weightage depending on which grades will be calculated.

Once a quiz parameters have been set click on **Save** button to save the quiz.

To create a Question paper, Click on **Add Question Paper** link located besides the created quiz.

3.2.2 Creating a Exercise

Click on **Add New Exercise** button to add a exercise.



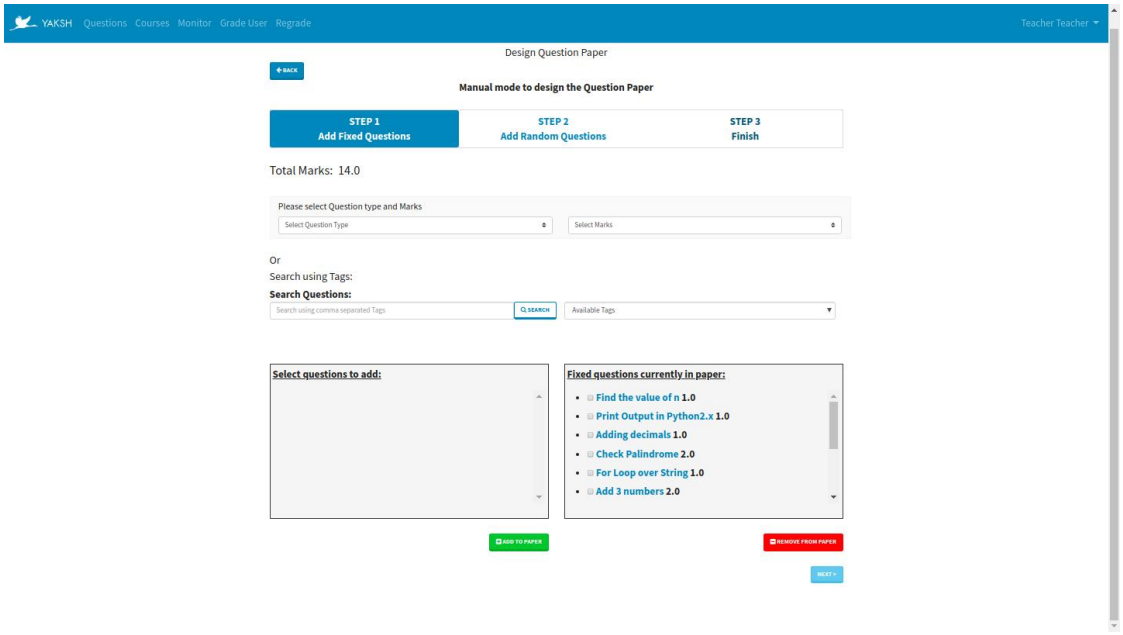
Exercise is similar to quiz with a difference that exercise has infinite attempts and infinite time. It also does not allow a student to skip the question. Each question in an exercise can be timed i.e. time to solve a particular question. Once the question time expires, question solution is shown to the student.

All the parameters are set by default only below parameters can be changed.

- **Description** - Description or name of the exercise.
- **Allow student to view answer paper** - Click on this checkbox to allow student to view their answer paper.
- **Active** - Select the checkbox to activate/deactivate exercise. Default value is active.

To create a Question paper, Click on **Add Question Paper** link located besides the created exercise.

3.2.3 Designing Question Paper



A quiz/exercise can have fixed as well as random questions. Fixed questions are those question that are bound to appear for every student taking the quiz. In random questions a pool of questions is given and

number of questions to be picked from the pool is set. Hence for different students, different questions from the pool will appear.

To add questions to a questionpaper

- Select Question type and marks and a list of questions will be displayed will be in the **select questions to add** section. Do this for both fixed questions and random questions.
- You can also search for questions using the tags added while creating the question. All the available tags are shown.
- After adding questions click on **Next** button to go to **Step 3 Finish**.
- Select **Shuffle questions' order for each student** if you want to jumble up the question sequence for every student and for every attempt.
- Select **Shuffle MCQ/MCC options for each student** if you want to jumble up the MCQ/MCC question options for every student and for every attempt.
- Click on **Save** to save it.

3.2.4 Editing a Quiz/Exercise

Click on the quiz/exercise link to edit, change the parameters and click on Save. Options Available once the Quiz is created.

- **Preview Question Paper** Click on the Preview Question Paper button located at the bottom of the page to preview all the questions available in the question paper of the quiz.
- **User Mode**

  Attempt quiz the way student will attempt i.e. - Quiz will have the same duration as that of the original quiz.

  Quiz won't start if the course is inactive or the quiz time has expired.
- **God Mode** Attempt quiz without any time or eligibilty constraints.

3.2.5 Editing a QuestionPaper

Click on the **Edit Question Paper** besides Quiz/Exercise and follow steps from Design Question Paper.

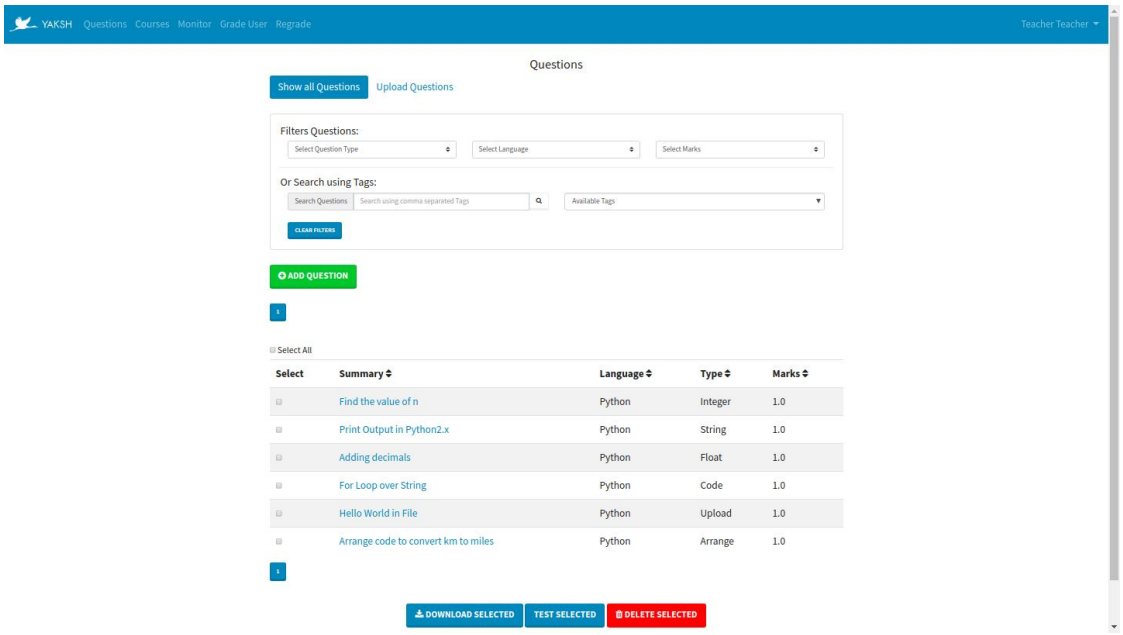
If the questions are already added to a Question Paper then they are shown in the **Fixed Questions currently in the paper** section.

3.3 Questions

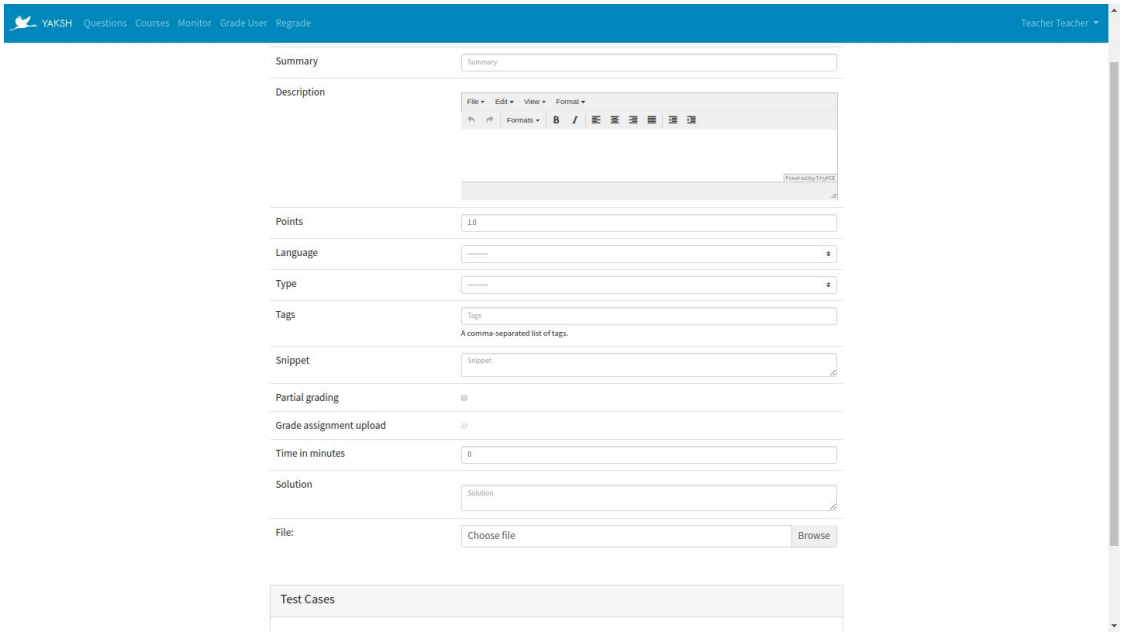
3.3.1 Setting up questions

Setting up questions is the most important part of the Yaksh experience. Questions can be of multiple types i.e Multiple choice questions (MCQ), Multiple correct choices (MCC), Coding questions, assignment upload, fill in the blanks.

To set up a question click on the questions link in the navigation bar.



To add a question click on the **Add Question** button



- **Summary**- Summary or the name of the question.
- **Language** - Programming language on which the question is based.
- **Type** - Type of the question. i.e Multiple Choice, Multiple Correct Choice, Code, Assignment Upload etc.
- **Points** - Points is the marks for a question.
- **Description** - The actual question description.
- **Tags** - Type of label or metadata tag making it easier to find specific type of questions.
- **Solution** - Add solution for the question.

- **Snippet** - Snippet is used to give any default value or default code or command. This will be displayed in the students answer form. This is used only for code questions.
- **Minimum time(in minutes)** - This value can be set for questions which will be added to a Exercise. Exercise time will depend on this time.
- **Partial Grading** - Click this checkbox to enable partial grading feature.
- **Grade Assignment Upload** - Click this checkbox if the assignment upload based question needs evaluation. Evaluation is done with **Hook based TestCase** only.
- **File - File field is used to upload files if there is any file based question.** For e.g. The question is reading a file say **dummy.txt** and print its content. You can then upload a file **dummy.txt** which will be available to the student while attempting the quiz.
  - **Some file features:**
    1. To delete a file click the delete checkbox and click on **Delete Selected Files button**.
    2. **To extract a file for e.g. say dummy.zip click the extract checkbox and click on Save button.** If **extract** is selected, the file will be extracted while checking the student submitted code.
    3. To hide any file from student click the hide checkbox and click on Save button.

Note: We only support **zip** extension for **extract file** feature.

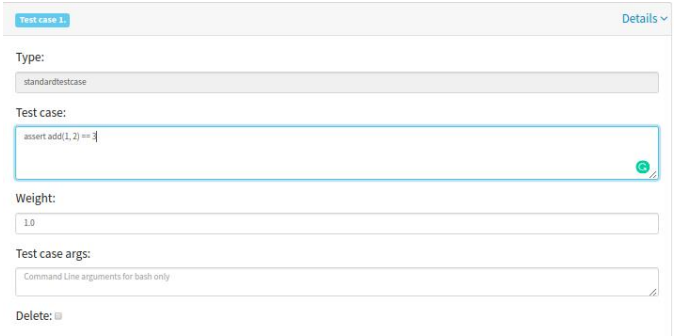
3.3.2 How to write Test cases

After saving the question with the necessary details, you will be able to add the test cases. A drop down **Add Test case** will be available to add the test case in the Test Cases section.

The following explains different methods to write test cases.

- **Create Standard Test Case**

  Select Standard from Add Test Case field. Sample Testcases are given for all languages.



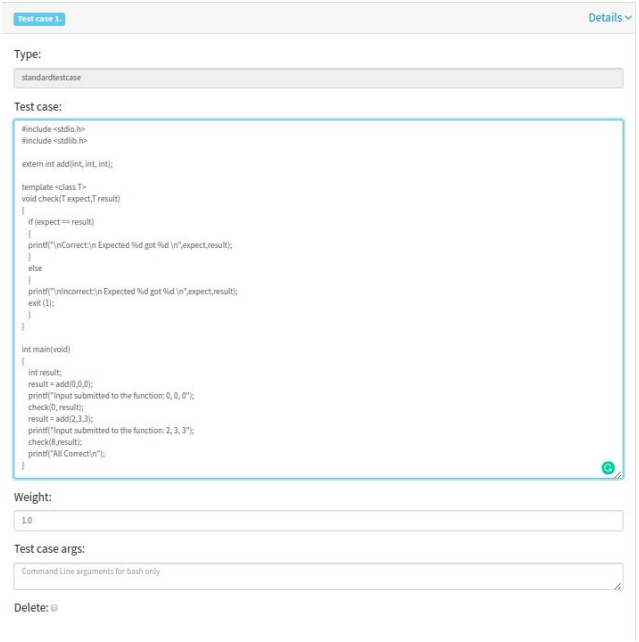
  - **For Python:**

    In the test case field write a python assert to check the user code. For e.g.

```
assert add(1, 2) == 3
```

    for program of addition.

  - **For C, C++:**



Consider a Program to add three numbers. The code in the Test case field should be as follows:

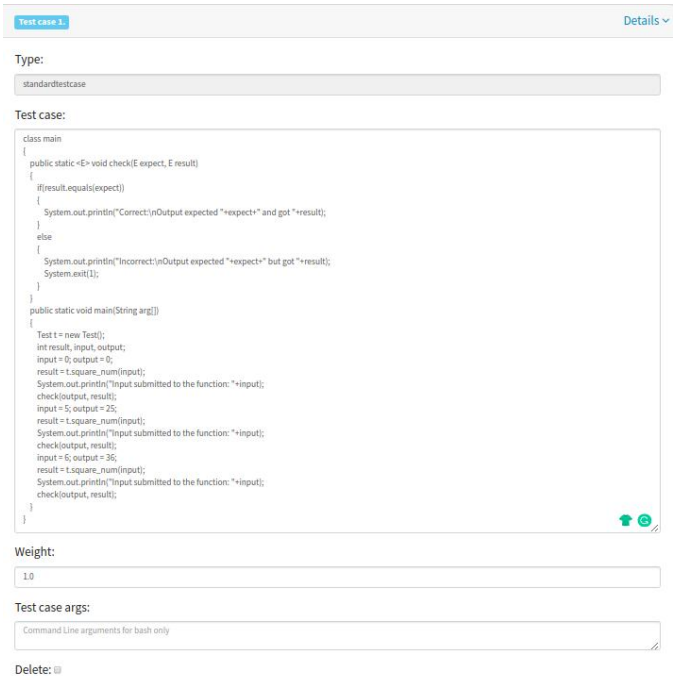
```
#include <stdio.h>
#include <stdlib.h>

extern int add(int, int, int);

template <class T>
void check(T expect,T result)
{
    if (expect == result)
    {
    printf("\nCorrect:\n Expected %d got %d \n",expect,
↪result);
    }
    else
    {
    printf("\nIncorrect:\n Expected %d got %d \n",
↪expect,result);
    exit (1);
    }
}

int main(void)
{
    int result;
    result = add(0,0,0);
    printf("Input submitted to the function: 0, 0, 0");
    check(0, result);
    result = add(2,3,3);
    printf("Input submitted to the function: 2, 3, 3");
    check(8,result);
    printf("All Correct\n");
}
```

Assuming Students answer to be as below:

```
int add(int a, int b, int c)
{
    return a+b+c;
}
```

Note:

1. In the above example, **add** in the main function is obtained from student code.
2. Please make sure that the student code function and testcase calling function should be same which in this case is **add**.



– **For Java:**

Consider a Program to find square of a number. The code in the Test case Field should be as follows:

```
class main
{
    public static <E> void check(E expect, E result)
    {
        if(result.equals(expect))
        {
            System.out.println("Correct:\nOutput expected
↪"+expect+" and got "+result);
        }
        else
        {
            System.out.println("Incorrect:\nOutput␣
↪expected "+expect+" but got "+result);
            System.exit(1);
```

(continues on next page)

(continued from previous page)

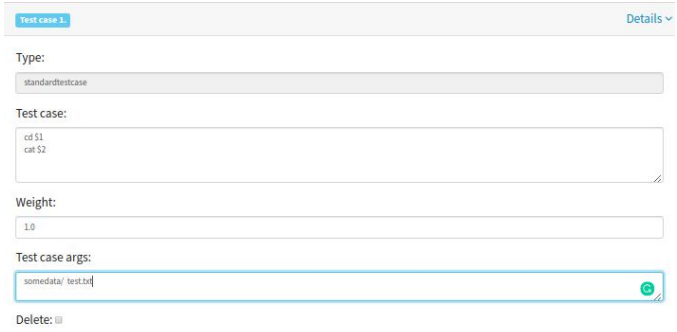
```
        }
    }
    public static void main(String arg[])
    {
        Test t = new Test();
        int result, input, output;
        input = 0; output = 0;
        result = t.square_num(input);
        System.out.println("Input submitted to the␣
↪function: "+input);
        check(output, result);
        input = 5; output = 25;
        result = t.square_num(input);
        System.out.println("Input submitted to the␣
↪function: "+input);
        check(output, result);
        input = 6; output = 36;
        result = t.square_num(input);
        System.out.println("Input submitted to the␣
↪function: "+input);
        check(output, result);
    }
}
```

Assuming Students answer to be as below:

```
class Test
{
    int square_num(int num)
    {
        return num*num;
    }
}
```

Note:

1. For Java, class name should always be **main** in testcase.
2. In the above example, **Test** is the class of student's code.
3. Please make sure that the student's code class and calling class in testcase is always **Test**. (square_num is the function inside Test class.)



– **For Bash:**

In Test case Field write your bash script. For e.g. the question is to move to a particular directory and read a file **test.txt** The Test case code shown is:

```
cd $1
cat $2
```

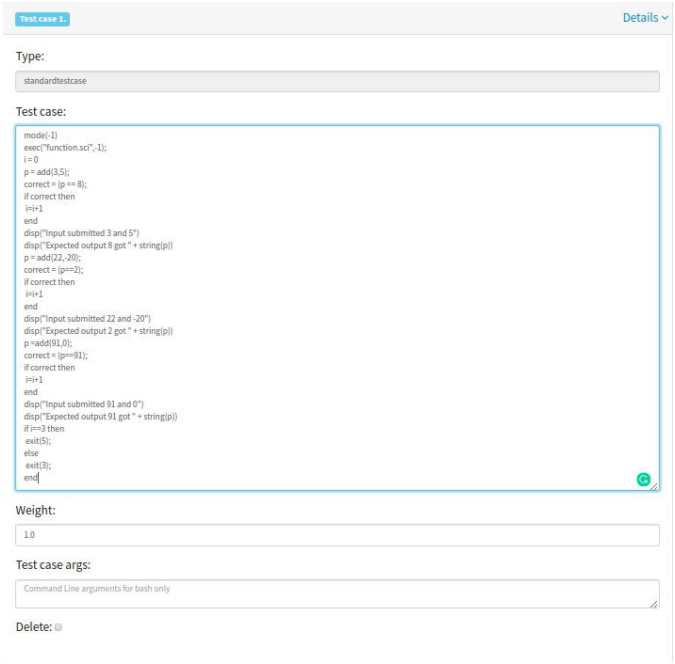
In **Test case args** Field type your Command line arguments.

In this case the test case args are:

```
somedata/  test.txt
```

Note:

1. **Test case args** field is used only for bash.
2. Each argument should be separated by **space**.
3. This field can be left blank.



– **For Scilab**

Consider a Program to add two numbers. The code in the Test case Field should be as follows:

```
mode(-1)
exec("function.sci",-1);
i = 0
p = add(3,5);
correct = (p == 8);
if correct then
 i=i+1
end
disp("Input submitted 3 and 5")
disp("Expected output 8 got " + string(p))
p = add(22,-20);
```

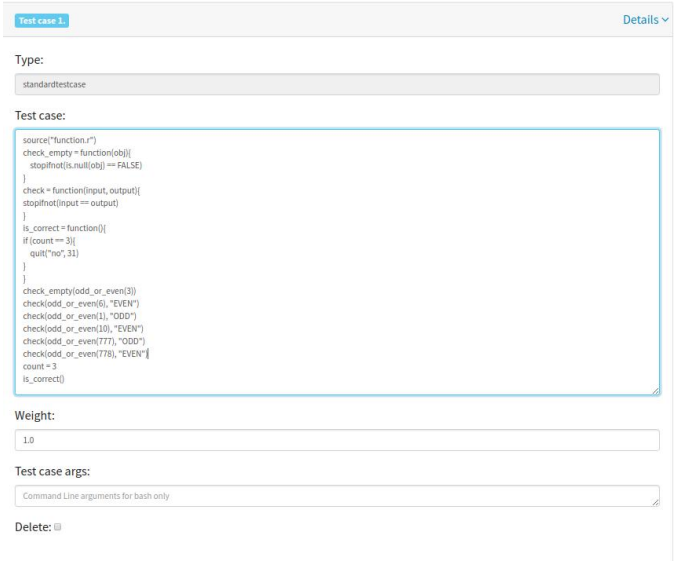
(continues on next page)

(continued from previous page)

```
correct = (p==2);
if correct then
 i=i+1
end
disp("Input submitted 22 and -20")
disp("Expected output 2 got " + string(p))
p =add(91,0);
correct = (p==91);
if correct then
 i=i+1
end
disp("Input submitted 91 and 0")
disp("Expected output 91 got " + string(p))
if i==3 then
 exit(5);
else
 exit(3);
end
```

Assuming Students answer to be as below:

```
funcprot(0)
function[c]=add(a,b)
c=a+b;
endfunction
```



– **For R**

Consider a Program to print even or odd number. The code in the Test case Field should be as follows:

```
source("function.r")
check_empty = function(obj){
    stopifnot(is.null(obj) == FALSE)
}
check = function(input, output){
stopifnot(input == output)
}
is_correct = function(){
```

(continues on next page)

(continued from previous page)

```
if (count == 3){
    quit("no", 31)
}
}
check_empty(odd_or_even(3))
check(odd_or_even(6), "EVEN")
check(odd_or_even(1), "ODD")
check(odd_or_even(10), "EVEN")
check(odd_or_even(777), "ODD")
check(odd_or_even(778), "EVEN")
count = 3
is_correct()
```

Assuming Students answer to be as below:

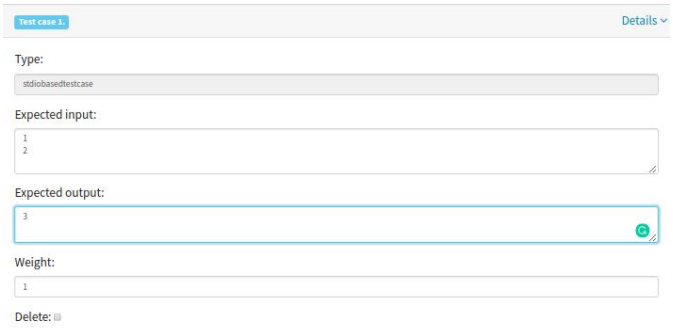
```
odd_or_even <- function(n){
  if(n %% 2 == 0){
    return("EVEN")
  }
  return("ODD")
}
```

Check **Delete** Field if a test case is to be removed.

Finally click on **Save** to save the test case.

- **Create Standard Input/Output Based Test Case**

  Select StdIO from Add Test Case field.



In Expected input field, enter the value(s) that will be passed to the students' code through a standard I/O stream.

Note: If there are multiple input values in a test case, enter the values in new line.

In Expected Output Field, enter the expected output for that test case. For e.g type 3 if the output of the user code is 3.

Setting up Standard Input/Output Based questions is same for all languages.

Note: Standard Input/Output Based questions is available only for the languages Python, C, C++, Java, Bash.

- **Create Hook based Test Case**

  Select Hook from Add Test Case field.

  In Hook based test case type, moderator is provided with a evaluator function called **check_answer** which is provided with a parameter called **user_answer**.

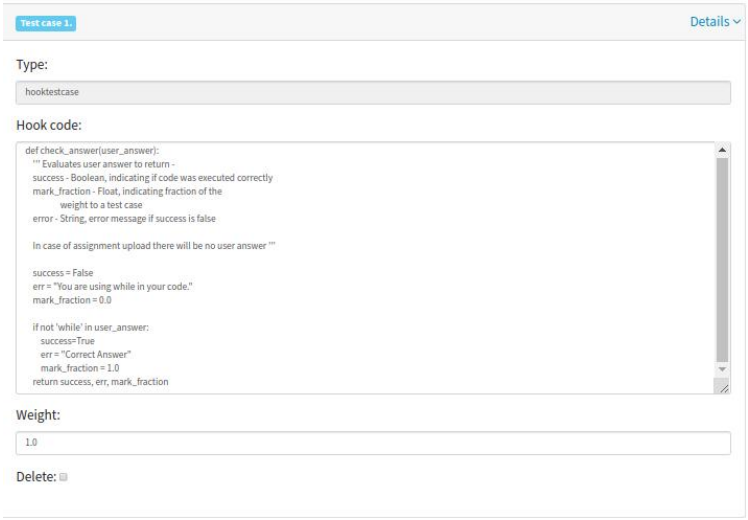
  user_answer is the code of the student in string format.

  Note: For assignment upload type question there will be no **user answer** File uploaded by student will be the answer.

  Suppose the student needs to upload a file say **new.txt** as assignment. Sample Hook code for this will be as shown below.

```
def check_answer(user_answer):
    ''' Evaluates user answer to return -
    success - Boolean, indicating if code was executed correctly
    mark_fraction - Float, indicating fraction of the weight to␣
↪a test case
    error - String, error message if success is false

    In case of assignment upload there will be no user answer ''
↪'

    success = False
    err = "Incorrect Answer" # Please make this more specific
    mark_fraction = 0.0

    try:
        with open('new.txt', 'r') as f:
            if "Hello, World!" in f.read():
                success = True
                err = "Correct Answer"
                mark_fraction = 1.0
            else:
                err = "Did not found string Hello, World! in␣
↪file."
    except IOError:
        err = "File new.txt not found."
    return success, err, mark_fraction
```

  A moderator can check the string for specific words in the user answer and/or compile and execute the user answer (using standard python libraries) to evaluate and hence return the mark fraction.



- **Create MCQ or MCC Based Test Case**

  Select MCQ/MCC from Add Test Case field.

  Fig (a) showing MCQ based testcase

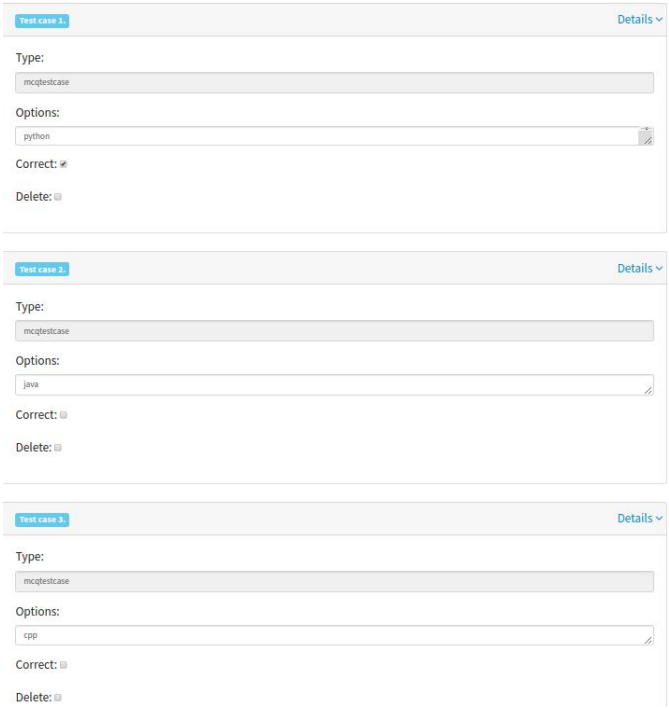
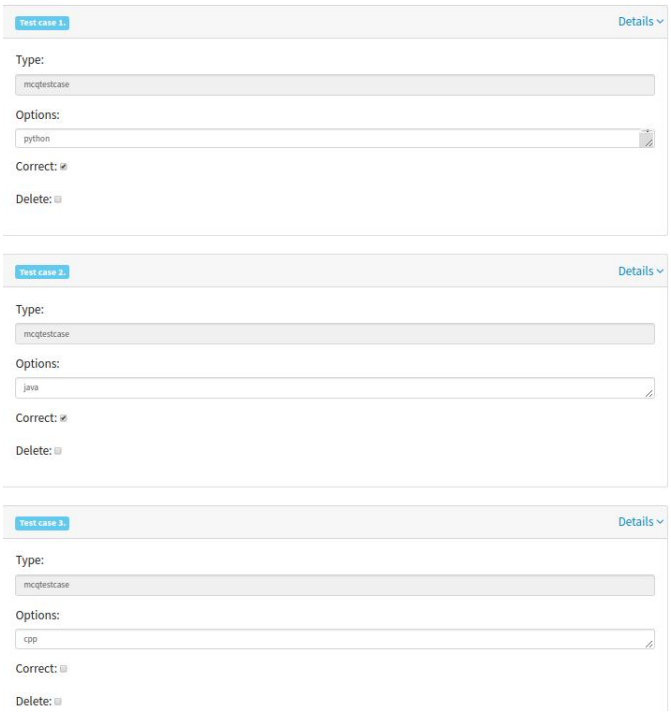


  Fig (b) showing MCC based testcase



In Options Field type the option check the correct checkbox if the current option is correct and click on Save button to save each option.

For MCC based question, check the correct checkbox for multiple correct options.

- **Create Integer Based Test Case**

  Select **Answer in Integer** from Type field.

  Select Integer from Add Test Case field.

  In the Correct field, add the correct integer value for the question.

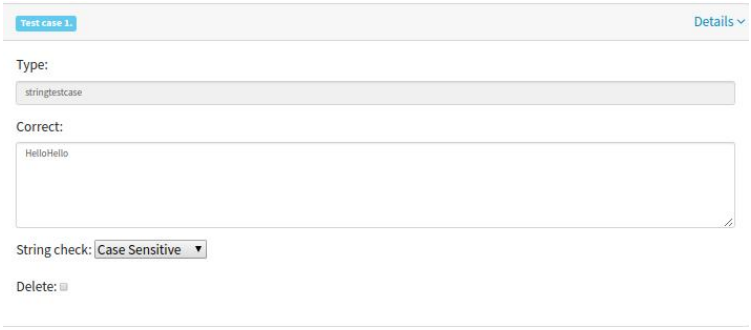
- **Create String Based Test Case**

  Select **Answer in String** from Type field.

  Select **String** from Add Test Case field.

  In the **Correct** field, add the exact string answer for the question.

In String Check field, select if the checking of the string answer should be case sensitive or not.



- **Create Float Based Test Case**

  Select **Answer in Float** from Type field.

  Select **Float** from Add Test Case field.

  In the **Correct** field, add the correct float value for the question.

  In the **Error Margin** field, add the margin of error that will be allowed.

3.3.3 Features in Question

- **Download Questions**

  Select questions from a list of questions displayed on the Questions page. Click on the Download Selected button to download the questions. This will create a zip file of the Questions selected. The zip will contain yaml file and an folder called **additional_files** which will contain files required by questions downloaded. Finally one can also download a template yaml file and modify it to add his/her questions.

- **Upload Questions**

  Click on the **Upload Questions** tab in the **Question Page**. One can upload Yaml file with extensions .yaml or .yml. Please note that you cannot upload files associated to a question. Yaml file can have any name.

  One can also upload zip with the following zip structure -

```
.zip
|-- .yaml or .yml
|-- .yaml or .yml
|-- folder1
|   |-- Files required by questions
```

(continues on next page)

(continued from previous page)

```
|-- folder2
|   |-- Files required by questions
```

- **Test Questions**

  Select questions from the list of question displayed on the Questions page. Click on Test selected button. This will take you to a quiz with the selected questions.

  Note: This will not create an actual quiz but a trial quiz. This quiz is hidden from the students and only for moderator to view.

- **Filter Questions**

  You can filter questions based on type of question, language of question or marks of question.

  1. Click Select Question Type to filter question based on type of the question.
  2. Click Select Language to filter question based on language of the question.
  3. Click Select marks to filter question based on mark of the question.

- **Search by tags**

  1. You can search the questions by tags added during question creation.
  2. Click on the Available tags to view all the available tags. Select any tag from available tags and click **Search**.
  3. Enter the tag in the search bar and click on **Search Icon** respective questions will be displayed.

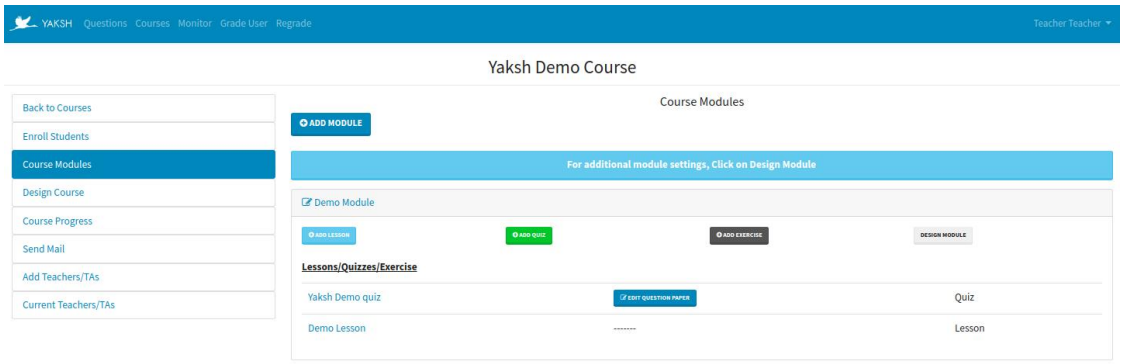
3.4 Lessons and Modules

Courses can have lessons and quizzes encapsulated using a module.

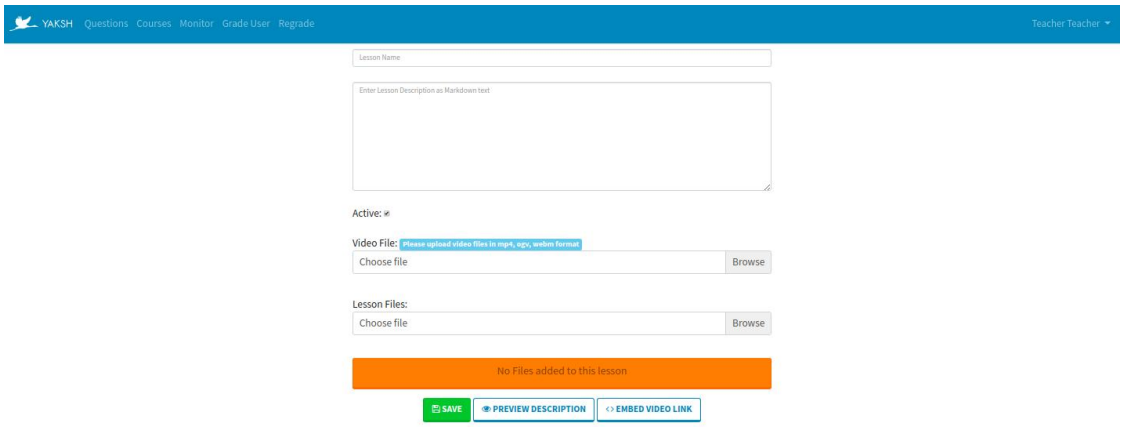
- **What is a lesson?** A lesson can be any markdown text with/or an embedded video of a particular topic.
- **What is a module?** A Module is a collection of lessons, quizzes and practice exercises clubbed together by similar idea/content. A module can have its own description as a markdown text with/or an embedded video.

3.4.1 Setting up a Lesson

This page shows all the lessons created by you and added to a module.



To create a new lesson click on **Add Lesson** button in the module.



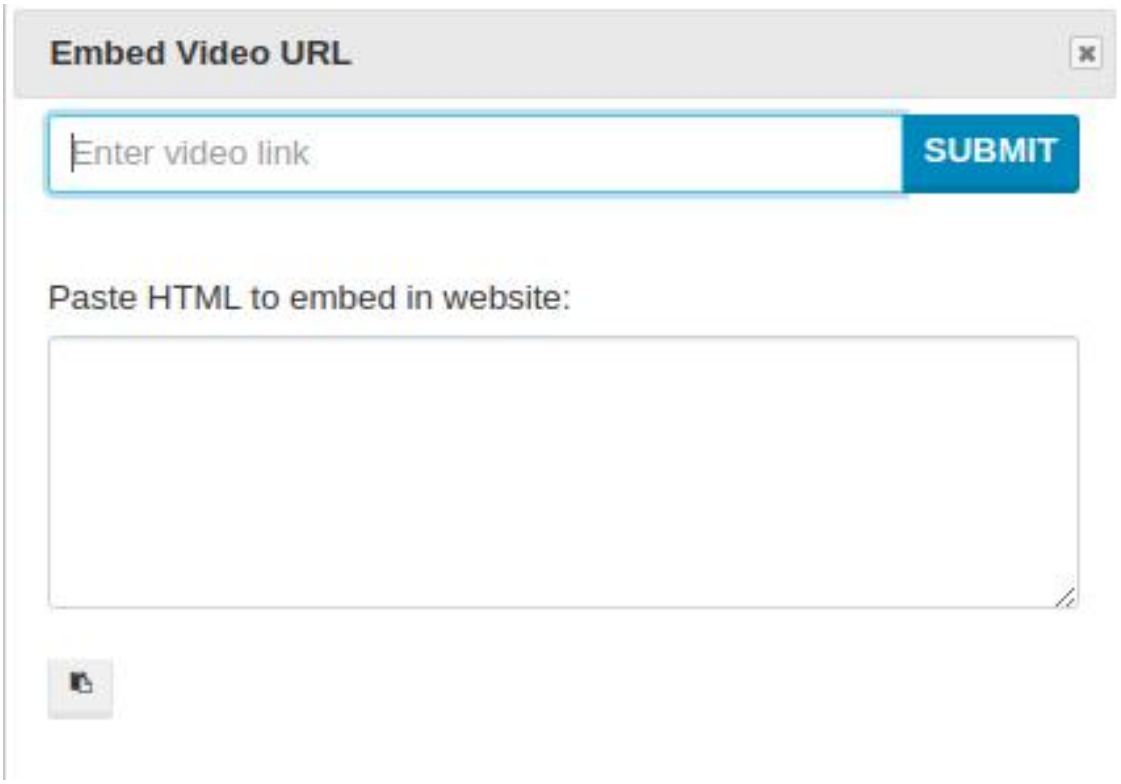
- **Name** - Name of the lesson.
- **Description** - Description can be any markdown text or embedded video link.
- **Active** - Activate/Deactivate a lesson
- **Video File** - Upload a video file for the lesson
- **Lesson files** - Add files to the lesson which will be available for students to view and download. All the uploaded files will be shown below.

Click on **Save** to save a lesson.

Click on **Preview Lesson Description** to preview lesson description. Markdown text from the description is converted to html and is displayed below.

Select the checkbox beside each uploaded file and click on **Delete files** to remove files from the lesson.

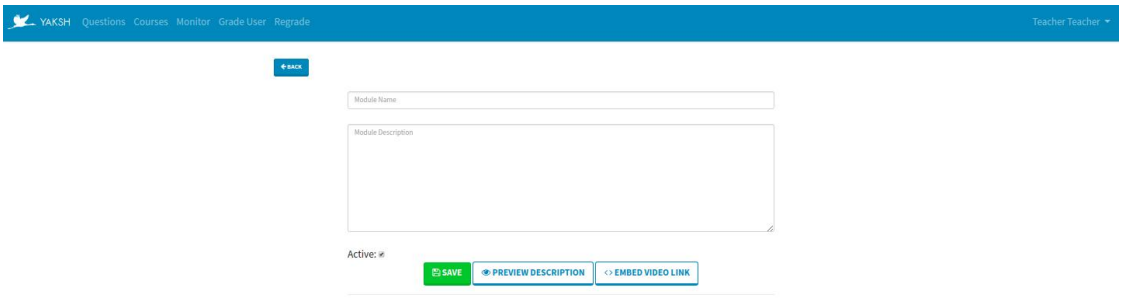
Click on **Embed Video Link** to embed a video. On clicking a pop-up will be shown.



Enter the url and click on **Submit** a html div is generated in the text area below. Click on the button below the textarea to copy the textarea content. This html div can then be added in the lesson description.

3.4.2 Setting up a Module

To create a new module click on **Add Module** button from course modules section of course details page.



- **Name** - Name of the module.
- **Description** - Description can be any markdown text or embedded video link.

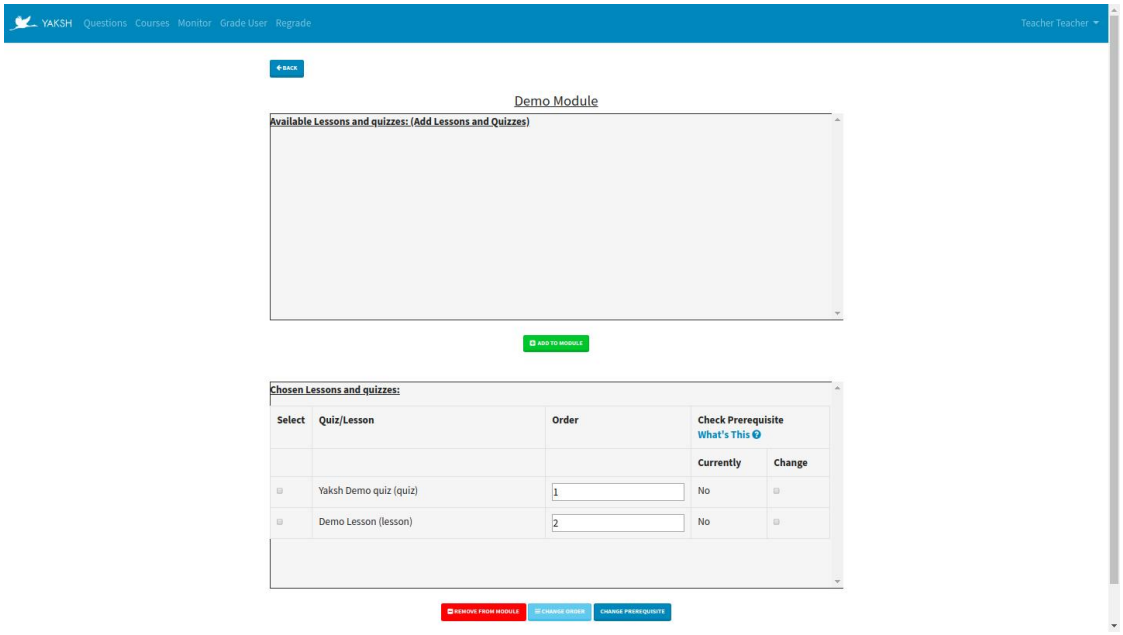
Click on **Save** to save a module.

Click on **Preview Lesson Description** to preview lesson description. Markdown text from the description is converted to html and is displayed below.

Click on **Embed Video Link** to embed a video.

3.4.3 Design a Module

To add lessons or quizzes to a module click on **Design Module**.



Available Lessons and quizzes contains all the lessons and quizzes that are not added to a module.

To add a lesson or a quiz to the module select the checkbox beside every lesson or quiz and click **Add to Module** button.

Chosen Lesson and quizzes contains all the lessons and quizzes that are added to a module.

A lesson or quiz added to a module becomes a unit. A unit has following parameters to change:

Order - Order in which units are shown to a student.

To change a unit's order change the value in the textbox under **Order** column and click **Change order**.

Check Prerequisite - Check if previous unit is completed. Default value is Yes. For e.g. A student has to first complete **Yaksh Demo quiz** to attempt **Demo Lesson** if the **Check Prerequisite** value for **Demo Lesson** is checked **Yes**.

Currently column shows the current value of **Check Prerequisite** which in this case is **Yes**.

Select the checkbox from **Change** column under **Check Prerequisite** and click **Change Prerequisite** button to change the value.

To remove a lesson or a quiz from the module select the checkbox beside every lesson or quiz and click **Remove from Module** button.

3.5 Other Features

3.5.1 Grade User

Grade User is a feature of Yaksh to access students' answer papers for each quiz and grade them where necessary.

Clicking on the **Grade User** link from the nav bar will show all the courses.

Click on the **Details** button to show the quizzes associated to a particular course.

Click on the **Grade User** button next to the quiz name to view all the students who have attempted the quiz.

Click on the student name to view their submissions.

3.5.2 Monitor

Monitor is a feature of Yaksh where the moderator can monitor a quiz and view statistics.

Clicking on the **Monitor** link from the nav bar will show all the courses.

Click on the **Details** button to show the quizzes associated to a particular course.

Click on the **Monitor** button next to the quiz name to view all the students who are attempting the quiz.

Click on the student name to view their submissions.

3.5.3 Grader

Click the **Grader** link on the navigation bar. This allows you to regrade answerpapers of students using three ways:

1. **Question wise regrade** Answerpapers can be regraded per Question.
2. **Quiz wise regrade** Answerpapers can be regraded per Quiz.
3. **User wise regrade** Answerpaper can be regraded for a particular student.

CHAPTER 4

About Yaksh

4.1 History

At FOSSEE, Nishanth had implemented a nice django based app to test for multiple-choice questions. Prabhu Ramachandran was inspired by a programming contest that he saw at PyCon APAC 2011. Chris Boesch, who administered the contest, used a nice web application Singpath that he had built on top of GAE that basically checked your Python code, live. This made it fun and interesting.

Prabhu wanted an implementation that was not tied to GAE and hence wrote the initial cut of what is now 'Yaksh'. The idea being that anyone can use this to test students programming skills and not have to worry about grading their answers manually and instead do so on their machines.

The application has since been refactored and maintained by FOSSEE Developers.

4.2 Contact

For further information and support you can contact

Python Team at FOSSEE

4.3 License

4.4 Authors

FOSSEE Developers