1. **INTRODUCTION**

The auto grader is an application that allow talent sprint instructors to grade code submissions of students by running against certain test cases. The tests are conducted online according to the course that the student is in.

The student submits the code. Then the application saves the uploaded code file in a directory. And then queue the file using Celery. One by one all the codes submitted are validated if a valid file is submitted in Python.

The validated codes are executed in a sand box environment by using docker. And appropriate templet displayed using Django framework and in edx platform.

The instructor initially can schedule a test in the UI that is provided for them. After the examination is completed, they can go to the instructors UI to fetch the results.

**1.2 SDLC METHODOLOGY**

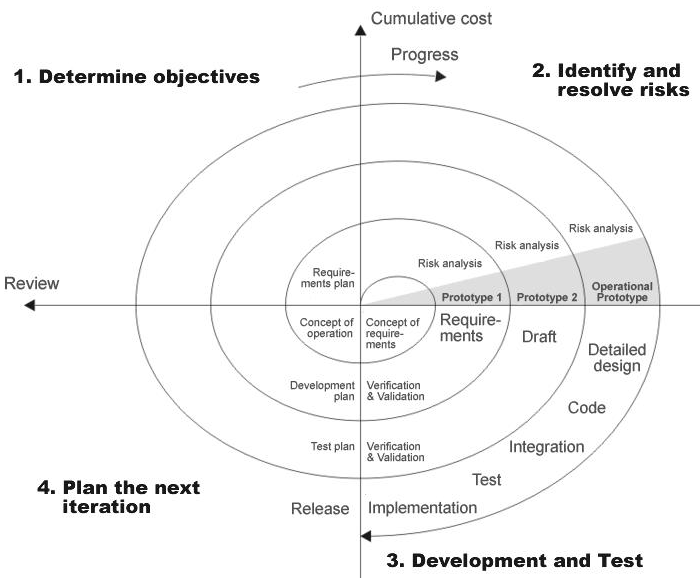
This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models. As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

* The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:
  + Evaluating the first prototype in terms of its strengths, weakness, and risks.
  + Defining the requirements of the second prototype.
  + Planning an designing the second prototype.
  + Constructing and testing the second prototype.
* At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.
* The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype.
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

The following diagram shows how a spiral model acts like:



**Fig 1.1-Spiral Model**

**Advantages**

* Estimates(i.e. budget, schedule etc .) become more relistic as work progresses, because important issues discoved earlier.
* It is more able to cope with the changes that are software development generally entails.
* Software engineers can get their hands in and start woring on the core of a project earlier.

**1.3 PROJECT MODULES**

The modle of the project are

* Compile box
* The grader database
* The client application
* The edX SGA module
  + 1. **Compile box**

Compile box is a NodeJS application that uses a *Docker*-based sandbox to compile and execute untrusted code and return the output. This can be used in two ways:

* Running the code against a certain set of test cases, and returning the results. The output can then be used to grade the code on the client.
* Compiling/Executing the code and returning the output.

Source Code:

The Compile box source code can be found on the server, [213.219.36.251](http://213.219.36.251:10001/compile), at /home/Compile box/Compile box/API. The main components are:

**app.js** : This is the starting point for the application. It handles HTTP requests and processes them as required. It then sends HTTP responses back to the clients, with the execution results.

**DockerSandbox.js** : This has the code responsible for creating a sandbox, pushing the code to be executed into it, calling the appropriate commands and storing the output.

**compilers.js** : This is an array of languages supported by Compile box, as well as other required information such as the command to run in the sandbox, the source code file, the expected file extension, etc.

Usage:

To use the Compile box in the client application, send a POST request to “<http://213.219.36.251:10001/compile>” with:

* Headers

Content-type: application/x-www-form-urlencoded

* Payload

“languageId” : the identifier which decides which compiler/interpreter to use on Compile box while evaluating the code. This must be hardcoded, and the mapping can be found in the Compile box source code, in the file compilers.js.

“code” : the code to be sent to the Compile box

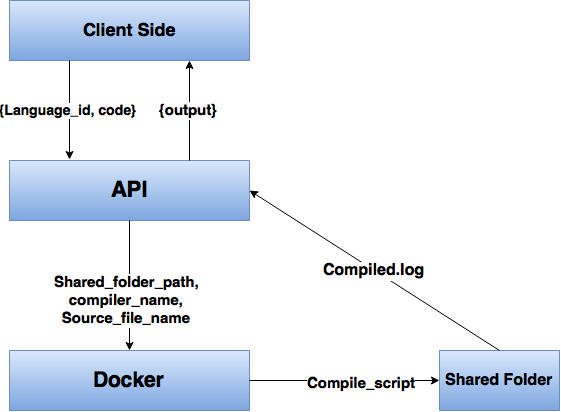
“fileName” : the name of the file in which the code will be stored, for compilation and execution in the Compile box. This is especially important for languages like Java.

“testCasePath”: the absolute path to the test cases that should be used in evaluating the code. This can be found in the grader database (i.e. the remote database) if required.

To start the Compile box on the server:

* Log into the server at 213.219.36.251, and navigate to /home/Compile box/Compile box/API.
* Start the nodejs application by running the command: sudo nodejs app.js

Compile box comes equipped with a large number of compilers and interpreters. However only some of them have been activated for use. To add support for a new language:

****

**Fig 1.2- Compile box Architecture**

**1.3.2 The grader database**

The MySQL database for the automatic grader keeps track of information that pertains to the system as a whole and is independent of the client using the grader application. This includes:

* Questions that can be used by the clients, as well as question papers that can be used for assessments.
* Submissions to the Compile box, and the results for each submission.

The database is on the server [213.219.36.251](http://213.219.36.251:10001/compile), and the name of the database is **‘graderapp’.**

**1.3.3 The client application**

The grader client application runs on whichever client want to access the Compile box and is responsible for the following tasks:

* Taking submissions from users, and updating the grader database with the details.
* Posting a request to the Compile box and processing the results, for example, calculating the grade based on test case output.
* Storing the results in the grader database as well as making it accessible to users and instructors, as required.

Source Code (EDX Client):

The client code is present on **sc.talentsprint.com: /home/edx/auto\_grader/.** The main components are:

**edx\_client.py** : This module uses a file monitor to check for code uploads and inserts the required details into a Celery queue.

**grader.py :** This module acts as a Celery worker. These workers pick up submissions from the queue, talk to the Compile box and process the results to be displayed.

**update\_local\_db.py (client-specific):** This is a collection of functions that talk to the EDX database to get information about code submissions or store results in a place that is locally accessible by EDX.

**update\_remote\_db.py (client-independant):** This is a collection of functons to interact with the remote grader database, to store submission data, or get information about a question, or get the results from Compile box.

Usage (EDX Client):

To start the grader application on edx:

* Navigate to **sc.talentsprint.com: /home/edx/auto\_grader/.**
* Start the upload monitor using the command: sudo python edx\_client.py
* Start Celery with a pool of workers using the command: celery -A grader worker -l info

**1.3.4 The edX SGA module**

The edX SGA module is an important part of the client application since it allows students to upload files, and provides instructors with a way to access and grade them.

1. **TECHNOLOGY DESCRIPTION**

**2.1 Requirements Gathering**

**2.1.1 Software requirements**

Operating System : Linux or Ubuntu

Server-side Scripting : JavaScript

Programming Language : Python

IDE/Workbench : VIM, Terminal

Server software Deployment : Docker, NodeJs

**2.1.2 Hardware requirements**

Processor : Pentium IV

Hard Disk : 40GB

RAM : 1GB or more

**2.2 Python**

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems. Using third-party tools, such as Py2exe or Pyinstaller, Python code can be packaged into stand-alone executable programs for some of the most popular operating systems, allowing the distribution of Python-based software for use on those environments without requiring the installation of a Python interpreter.

CPython, the reference implementation of Python, is free and open-source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and there are a number of language features which support functional programming and aspect-oriented programming (including by metaprogramming and by magic methods). Many other paradigms are supported using extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution.

The design of Python offers some support for functional programming in the Lisp tradition. The language has map(), reduce() and filter() functions; comprehensions for lists, dictionaries, and sets; and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The core philosophy of the language is summarized by the document "PEP 20 (The Zen of Python)", which includes aphorisms such as:

* Beautiful is better than ugly
* Explicit is better than implicit
* Simple is better than complex
* Complex is better than complicated
* Readability counts

**2.3 Django**

Django (*JANG-goh*) is a free and open-source web framework, written in Python, which follows the model–view–controller (MVC) architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent organization established as a 501(c)(3) non-profit.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Some well-known sites that use Django include Pinterest, Instagram, Mozilla, The Washington Times, Disqus, the Public Broadcasting Service, and Bitbucket.

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

**Ridiculously fast -** Django was designed to help developers take applications from concept to completion as quickly as possible.

**Reassuringly secure -** Django takes security seriously and helps developers avoid many common security mistakes.

**Exceedingly scalable -** Some of the busiest sites on the Web leverage Django’s ability to quickly and flexibly scale.

**2.4 Celery**

Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well.

The execution units, called tasks, are executed concurrently on a single or more worker servers using multiprocessing, [Eventlet](http://eventlet.net/), or [gevent](http://gevent.org/). Tasks can execute asynchronously (in the background) or synchronously (wait until ready). Celery is used in production systems to process millions of tasks a day.

## Easy to integrate

Celery is easy to integrate with web frameworks, some of which even have integration packages. Celery is written in Python, but the protocol can be implemented in any language. It can also operate with other languages using webhooks.

## Multi broker support

The recommended message broker is RabbitMQ, but support for Redis, Beanstalk, MongoDB, CouchDB, and databases (using SQLAlchemy or the Django ORM) is also available.

**2.5 Open edX**

edX is a massive open online course (MOOC) provider. It hosts online university-level courses in a wide range of disciplines to a worldwide student body, including some courses at no charge. It also conducts research into learning based on how people use its platform. EdX differs from other MOOC providers, such as Coursera and Udacity, in that it is a nonprofit organization and runs on open-source software.

The Massachusetts Institute of Technology and Harvard University created edX in May 2012. More than 70 schools, nonprofit organizations, and corporations offer or plan to offer courses on the edX website. As of 24 March 2016, edX has more than 7 million students taking more than 700 courses online.

EdX has been developed as open-source software and made available to other institutions of higher learning that want to make similar offerings. On June 1, 2013, edX open sourced its entire platform.

The source code can be found on GitHub.

**2.6 MongoDB**

MongoDB (from *humongous*) is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster. MongoDB is developed by MongoDB Inc. and is published as free and open-source software under a combination of the GNU Affero General Public License and the Apache License. As of July 2015[[update]](https://en.wikipedia.org/w/index.php?title=MongoDB&action=edit), MongoDB is the fourth most popular type of database management system, and the most popular for document stores.

Some of the features include:

**Ad hoc queries**

MongoDB supports field, range queries, regular expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions.

**Indexing**

Any field in a MongoDB document can be indexed – including within arrays and embedded documents (indices in MongoDB are conceptually similar to those in RDBMSes). Primary and secondary indices are available.

**Replication**

MongoDB provides high availability with replica sets. A replica set consists of two or more copies of the data. Each replica set member may act in the role of primary or secondary replica at any time. The all writes and reads are done on the primary replica by default. Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary. Secondaries can optionally serve read operations, but that data is only eventually consistent by default.

**Load balancing**

MongoDB scales horizontally using sharding. The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. (A shard is a master with one or more slaves.). Alternatively, the shard key can be hashed to map to a shard – enabling an even data distribution.

MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. MongoDB is easy to deploy, and new machines can be added to a running database.

**File storage**

MongoDB can be used as a file system, taking advantage of load balancing and data replication features over multiple machines for storing files.

This function, called Grid File System, is included with MongoDB drivers and available for many development languages (see "Language Support" for a list of supported languages). MongoDB exposes functions for file manipulation and content to developers. GridFS is used, for example, in plugins for NGINX and lighttpd. Instead of storing a file in a single document, GridFS divides a file into parts, or chunks, and stores each of those chunks as a separate document.

In a multi-machine MongoDB system, files can be distributed and copied multiple times between machines transparently, thus effectively creating a load-balanced and fault-tolerant system.

**Aggregation**

MapReduce can be used for batch processing of data and aggregation operations. The aggregation framework enables users to obtain the kind of results for which the SQL GROUP BY clause is used. Aggregation operators can be strung together to form a pipeline – analogous to Unix pipes. The aggregation framework includes the $lookup operator which can join documents from multiple documents.

**Server-side JavaScript execution**

JavaScript can be used in queries, aggregation functions (such as MapReduce), and sent directly to the database to be executed.

**Capped collections**

MongoDB supports fixed-size collections called capped collections. This type of collection maintains insertion order and, once the specified size has been reached, behaves like a circular queue.

**2.7 MySQL**

MySQL ("My S-Q-L",) is an open-source relational database management system (RDBMS); in July 2013, it was the world's second most widely used RDBMS, and the most widely used open-source client–server model RDBMS. It is named after Michael Widenius' (who is a co-founder of MySQL) daughter, My, while "SQL" stands as the abbreviation for Structured Query Language. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation. For proprietary use, several paid editions are available, and offer additional functionality.

MySQL is a popular choice of database for use in web applications, and is a central component of the widely used LAMP open-source web application software stack (and other "AMP" stacks). LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python". Free-software open-source projects that require a full-featured database management system often use MySQL. Applications that use the MySQL database include: TYPO3, MODx, Joomla, WordPress, phpBB, MyBB, Drupal and other software. MySQL is also used in many high-profile, large-scale websites, including Google (though not for searches), Facebook, Twitter, Flickr, and YouTube.

On all platforms except Windows, MySQL ships with no GUI tools to administer MySQL databases or manage data contained within the databases. Users may use the included command line tools, or install MySQL Workbench via a separate download. Many third party GUI tools are also available.

MySQL is written in C and C++. Its SQL parser is written in yacc, but it uses a home-brewed lexical analyzer. MySQL works on many system platforms, including AIX, BSDi, FreeBSD, HP-UX, eComStation, i5/OS, IRIX, Linux, OS X, Microsoft Windows, NetBSD, Novell NetWare, OpenBSD, OpenSolaris, OS/2 Warp, QNX, Oracle Solaris, Symbian, SunOS, SCO OpenServer, SCO UnixWare, Sanos and Tru64. A port of MySQL to OpenVMS also exists.

The MySQL server software itself and the client libraries use dual-licensing distribution. They are offered under GPL version 2, beginning from 28 June 2000 (which in 2009 has been extended with a FLOSS License Exception) or to use a proprietary license.

Support can be obtained from the official manual. Free support additionally is available in different IRC channels and forums. Oracle offers paid support via its MySQL Enterprise products. They differ in the scope of services and in price. Additionally, a number of third party organisations exist to provide support and services, including MariaDB and Percona.

MySQL has received positive reviews, and reviewers noticed it "performs extremely well in the average case". and that the "developer interfaces are there, and the documentation (not to mention feedback in the real world via Web sites and the like) is very, very good". It has also been tested to be a "fast, stable and true multi-user, multi-threaded sql database server".

**2.8 Docker**

Docker is all about making it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

And importantly, Docker is open source. This means that anyone can contribute to Docker and extend it to meet their own needs if they need additional features that aren't available out of the box.

**2.8.1 Who is Docker for?**

Docker is a tool that is designed to benefit both developers and system administrators, making it a part of many DevOps (developers + operations) toolchains. For developers, it means that they can focus on writing code without worrying about the system that it will ultimately be running on. It also allows them to get a head start by using one of thousands of programs already designed to run in a Docker container as a part of their application. For operations staff, Docker gives flexibility and potentially reduces the number of systems needed because of its small footprint and lower overhead.

**2.8.2 Docker and security**

Docker brings security to applications running in a shared environment, but containers by themselves are not an alternative to taking proper security measures.

Dan Walsh, a computer security leader best known for his work on SELinux, gives his perspective on the importance of making sure Docker containers are secure. He also provides a detailed breakdown of security features currently within Docker, and how they function.

## 2.8.3 How does this help you build better software?

When your app is in Docker containers, you don’t have to worry about setting up and maintaining different environments or different tooling for each language. Focus on creating new features, fixing issues and shipping software.

**Accelerate Developer Onboarding**

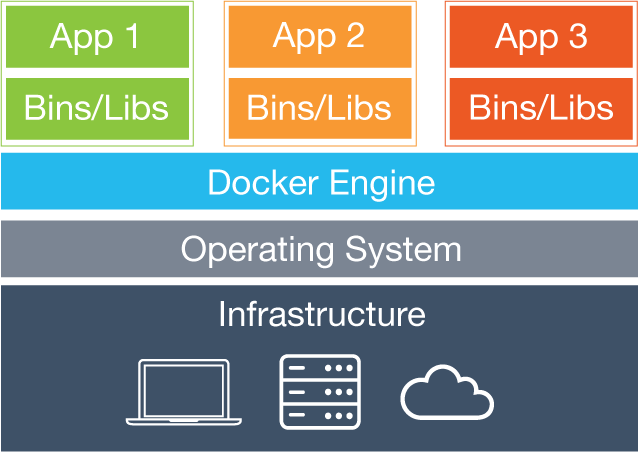
Stop wasting hours trying to setup developer environments, spin up new instances and make copies of production code to run locally. With Docker, you can easily take copies of your live environment and run on any new endpoint running Docker.

**Empower Developer Creativity**

The isolation capabilities of Docker containers free developers from the worries of using “approved” language stacks and tooling. Developers can use the best language and tools for their application service without worrying about causing conflict issues.

**Eliminate Environment Inconsistencies**

By packaging up the application with its configs and dependencies together and shipping as a container, the application will always work as designed locally, on another machine, in test or production. No more worries about having to install the same configs into a different environment.



**Fig 2.1- Docker Architecture**

### Containers

Containers include the application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in userspace on the host operating system. They’re also not tied to any specific infrastructure – Docker containers run on any computer, on any infrastructure and in any cloud.

**2.9 JavaScript**

JavaScript is a script-based programming language that was developed by Netscape Communication Corporation. JavaScript was originally called Live Script and renamed as JavaScript to indicate its relationship with Java. JavaScript supports the development of both client and server components of Web-based applications. On the client side, it can be used to write programs that are executed by a Web browser within the context of a Web page. On the server side, it can be used to write Web server programs that can process information submitted by a Web browser and then update the browser’s display accordingly

Even though JavaScript supports both client and server Web programming, we prefer JavaScript at Client side programming since most of the browsers supports it. JavaScript is almost as easy to learn as HTML, and JavaScript statements can be included in HTML documents by enclosing the statements between a pair of scripting tags

<SCRIPTS>.. </SCRIPT>.

<SCRIPT LANGUAGE = “JavaScript”>

JavaScript statements

</SCRIPT>

Here are a few things we can do with JavaScript:

Validate the contents of a form and make calculations.

Add scrolling or changing messages to the Browser’s status line.

Animate images or rotate images that change when we move the mouse over them.

Detect the browser in use and display different content for different browsers.

Detect installed plug-ins and notify the user if a plug-in is required.

We can do much more with JavaScript, including creating entire application.

**Advantages**

* JavaScript can be used for Sever-side and Client-side scripting.
* It is more flexible than VBScript.
* JavaScript is the default scripting languages at Client-side since all the browsers supports it.

Client-side components usually consist of HTML, CSS, JavaScript, Ajax, JS libraries, images, and whatever other files that are to be downloaded to the browser. On the server, you need a listener to process requests, fetch resources or information, and manipulate them so that they can be sent back to the client. This is usually accomplished using XML, JSON, or HTML-Formatted text, which is sent across the wire using Ajax. There are a number of competing technologies to choose from here. Depending on your traffic, hardware, O/S, bandwidth, IT expertise, and numerous other factors, there is a technology for every taste and occasion. Popular server-side languages at this time include PHP, Java, and .NET, to name only a few. There is presently a ServerJS movement whose goal is to eliminate the gap between client and server. Exponents of the group want to keep with HTML, JavaScript, and CSS, which are most familiar to the end-users. What makes server-side JavaScript possible is a web server than can process the code. One such server is called Jaxer. Developed by Aptana, Jaxer is an open source Ajax web server for building rich web pages and applications using a unified Ajax model that can be written entirely using JavaScript. Writing code for Jaxer is the focus of this article.

**2.10 Shell script**

A shell script is a computer program designed to be run by the Unix shell, a command line interpreter. The various dialects of shell scripts are considered to be scripting languages.

Typical operations performed by shell scripts include file manipulation, program execution, and printing text. A script which sets up the environment, runs the program, and does any necessary cleanup, logging, &c is called a wrapper.

The term is also used more generally to mean the automated mode of running an operating system shell; in specific operating systems they are called other things such as batch files (MSDos-Win95 stream, OS/2), command procedures (VMS), and shell scripts (Windows NT stream and third-party derivatives like 4NT—article is at cmd.exe), and mainframe operating systems are associated with a number of terms.

The typical Unix/Linux/Posix-compliant installation includes the Korn Shell (ksh) in several possible versions such as ksh88, Korn Shell '93 and others. The oldest shell still in common use is the Bourne shell (sh); Unix systems invariably include also the C Shell (csh), Bourne Again Shell (bash), a remote shell (rsh), a secure shell for SSL telnet connections (ssh), and a shell which is a main component of the Tcl/Tk installation usually called tclsh; wish is a GUI-based Tcl/Tk shell. The C and Tcl shells have syntax quite similar to that of said programming languages, and the Korn shells and Bash are developments of the Bourne shell, which is based on the ALGOL language with elements of a number of others added as well. On the other hand, the various shells plus tools like awk, sed, grep, and BASIC, Lisp, C and so forth contributed to the Perl programming language.

Other shells available on a machine or available for download and/or purchase include ash, msh, ysh, zsh (a particularly common enhanced Korn Shell), the Tenex C Shell (tcsh), a Perl-like shell (psh) and others. Related programmes such as shells based on Python, Ruby, C, Java, Perl, Pascal, Rexx &c in various forms are also widely available. Another somewhat common shell is osh, whose manual page states it "is an enhanced, backward-compatible port of the standard command interpreter from Sixth Edition UNIX."

Windows-Unix interoperability software such as the MKS Toolkit, Cygwin, UWIN, Interix and others make the above shells and Unix programming available on Windows systems all the way down to such things as signals and other Interprocess Communication, system calls and APIs; the Hamilton C Shell is a Windows shell very similar to the Unix C Shell, and Microsoft distributes Services for Unix for use with its NT-based operating systems in particular, which have a Posix environmental subsystem.

**2.11 JUNIT**

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit.

JUnit is linked as a JAR at compile-time; the framework resides under package junit.framework for JUnit 3.8 and earlier, and under package org.junit for JUnit 4 and later.

A research survey performed in 2013 across 10,000 Java projects hosted on GitHub found that JUnit, (in a tie with slf4j-api), was the most commonly included external library. Each library was used by 30.7% of projects.

## Example of JUnit test fixture

A JUnit test fixture is a Java object. With older versions of JUnit, fixtures had to inherit from junit.framework.TestCase, but the new tests using JUnit 4 should not do this. Test methods must be annotated by the @Test annotation. If the situation requires it, it is also possible to define a method to execute before (or after) each (or all) of the test methods with the @Before (or @After) and @BeforeClass (or @AfterClass) annotations.

import org.junit.\*;

public class FoobarTest {

@BeforeClass

public static void setUpClass() throws Exception {

// Code executed before the first test method

}

@Before

public void setUp() throws Exception {

// Code executed before each test

}

@Test

public void testOneThing() {

// Code that tests one thing

}

@Test

public void testAnotherThing() {

// Code that tests another thing

}

@Test

public void testSomethingElse() {

// Code that tests something else

}

@After

public void tearDown() throws Exception {

// Code executed after each test

}

@AfterClass

public static void tearDownClass() throws Exception {

// Code executed after the last test method

}

}

**2.12 NodeJs**

As an asynchronous event driven framework, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection the callback is fired, but if there is no work to be done Node is sleeping.

const http = require('http');

const hostname = '127.0.0.1';

const port = 1337;

http.createServer((req, res) => {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('Hello World\n');

}).listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

This is in contrast to today's more common concurrency model where OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node are free from worries of dead-locking the process—there are no locks. Almost no function in Node directly performs I/O, so the process never blocks. Because nothing blocks, less-than-expert programmers are able to develop scalable systems.

Node is similar in design to and influenced by systems like Ruby's [Event Machine](http://rubyeventmachine.com/) or Python's [Twisted](http://twistedmatrix.com/). Node takes the event model a bit further, it presents the event loop as a language construct instead of as a library. In other systems there is always a blocking call to start the event-loop. Typically one defines behavior through callbacks at the beginning of a script and at the end starts a server through a blocking call like EventMachine::run(). In Node there is no such start-the-event-loop call. Node simply enters the event loop after executing the input script. Node exits the event loop when there are no more callbacks to perform. This behavior is like browser JavaScript — the event loop is hidden from the user.

HTTP is a first class citizen in Node, designed with streaming and low latency in mind. This makes Node well suited for the foundation of a web library or framework.

Just because Node is designed without threads, doesn't mean you cannot take advantage of multiple cores in your environment. You can spawn child processes that are easy to communicate with by using our [child\_process.fork()](https://nodejs.org/api/child_process.html#child_process_child_process_fork_modulepath_args_options) API. Built upon that same interface is the [cluster](https://nodejs.org/api/cluster.html) module, which allows you to share sockets between processes to enable load balancing over your cores.

**3 DESIGN**

**3.1 UML Diagrams**

The unified modeling language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

**User Model View**

This view represents the system from the users perspective. The analysis representation describes a usage scenario from the end-users perspective.

**Structural Model View**

In this model the data and functionality are arrived from inside the system.This model view models the static structures.

**Behavioral Model View**

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

**Implementation Model View**

In this the structural and behavioral as parts of the system are represented as they are to be built.

**Environmental Model View**

In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

UML Analysis modeling, which focuses on the user model and structural model views of the system.

UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

Use case Diagrams represent the functionality of the system from a user’s point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view. Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer …etc., or another system like central database.

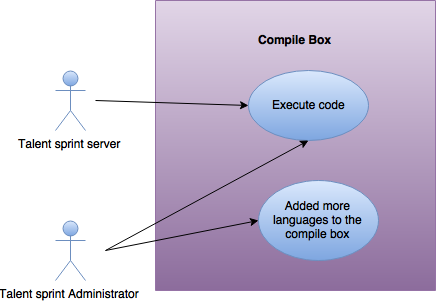
* + 1. **Use Case Diagrams**

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases.

So we can say that use cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system.

The actors can be human user, some internal applications or may be some external applications. So in a brief when we are planning to draw an use case diagram we should have the following items identified.

* Functionalities to be represented as an use case
* Actors
* Relationships among the use cases and actors.

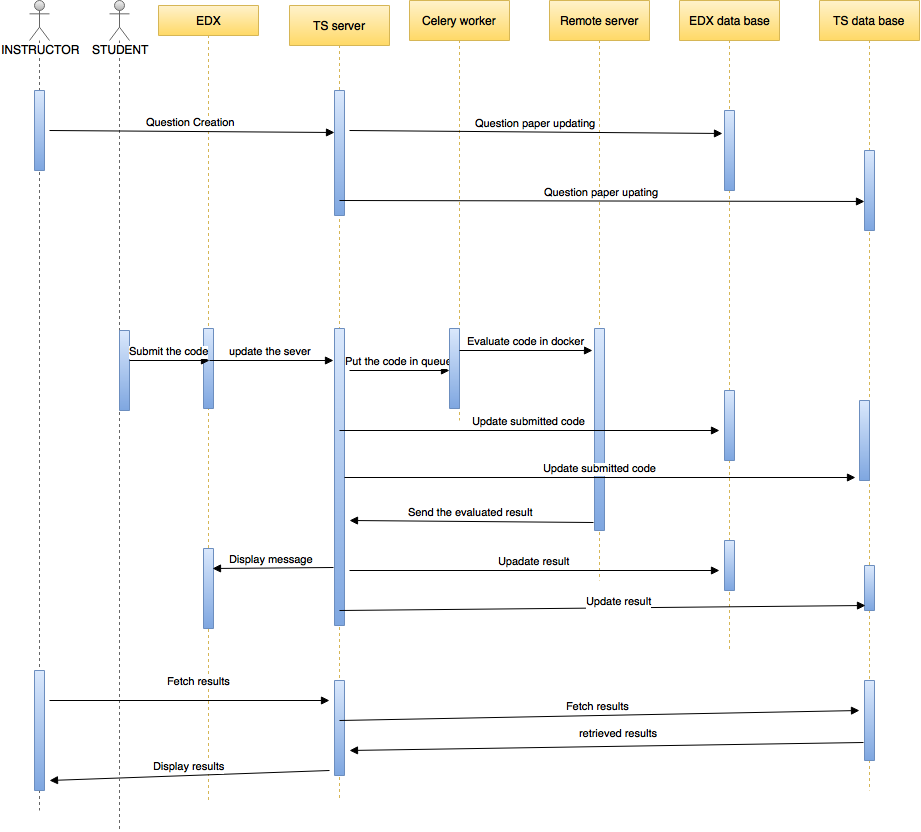
****

**Fig 3.1- Use case diagram**

* + 1. **Sequence Diagram**

A Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



**Fig 3.2- Sequence diagram**

**3.2 Data Flow Diagrams**

A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also know as a data flow graph or a bubble chart. DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system’s structure charts. The Basic Notation used to create a DFD’s are as follows:

1. **Dataflow**

Data move in a specific direction from an origin to a destination

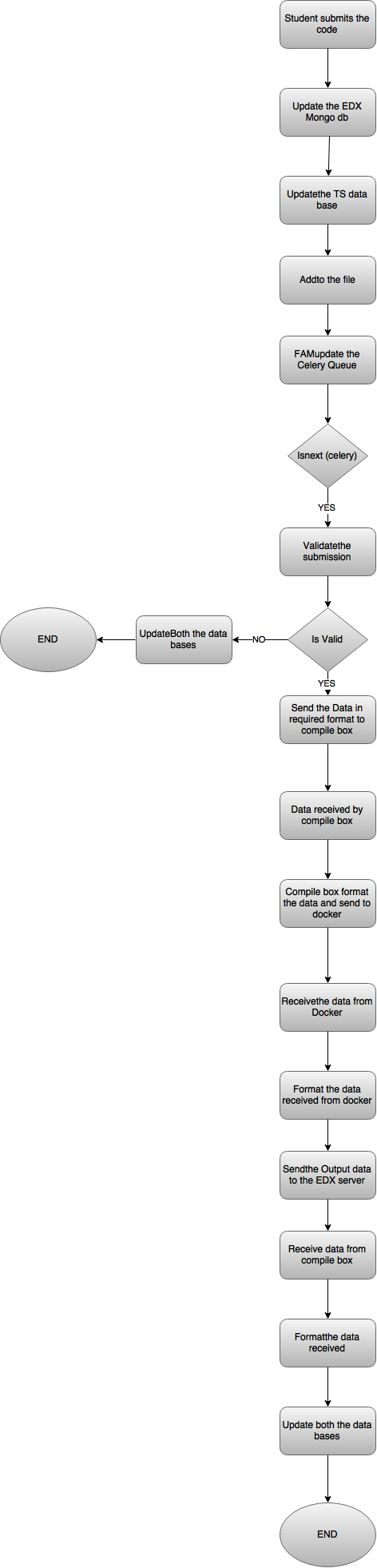
1. **Process**

People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.

1. **Source**

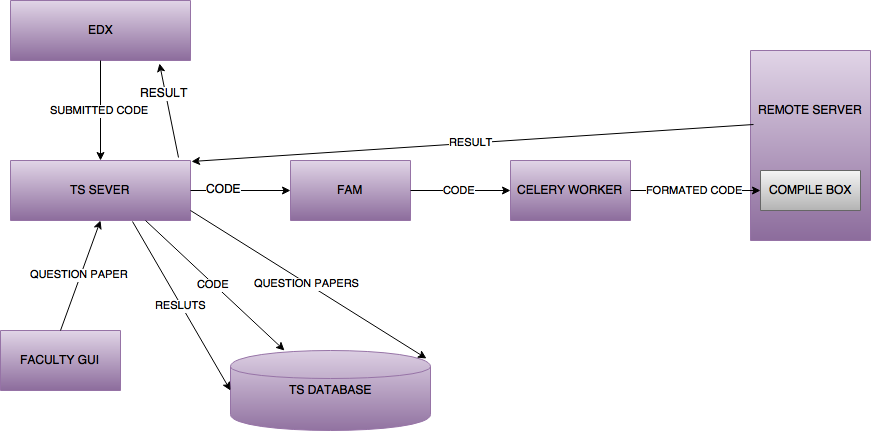
External sources or destination of data, which may be People, programs, organizations or other entities.

1. **Data Store:** Here data are stored or referenced by a process in the System.



**Fig 3.3- Data flow diagram**

* 1. **Architecture Diagram**

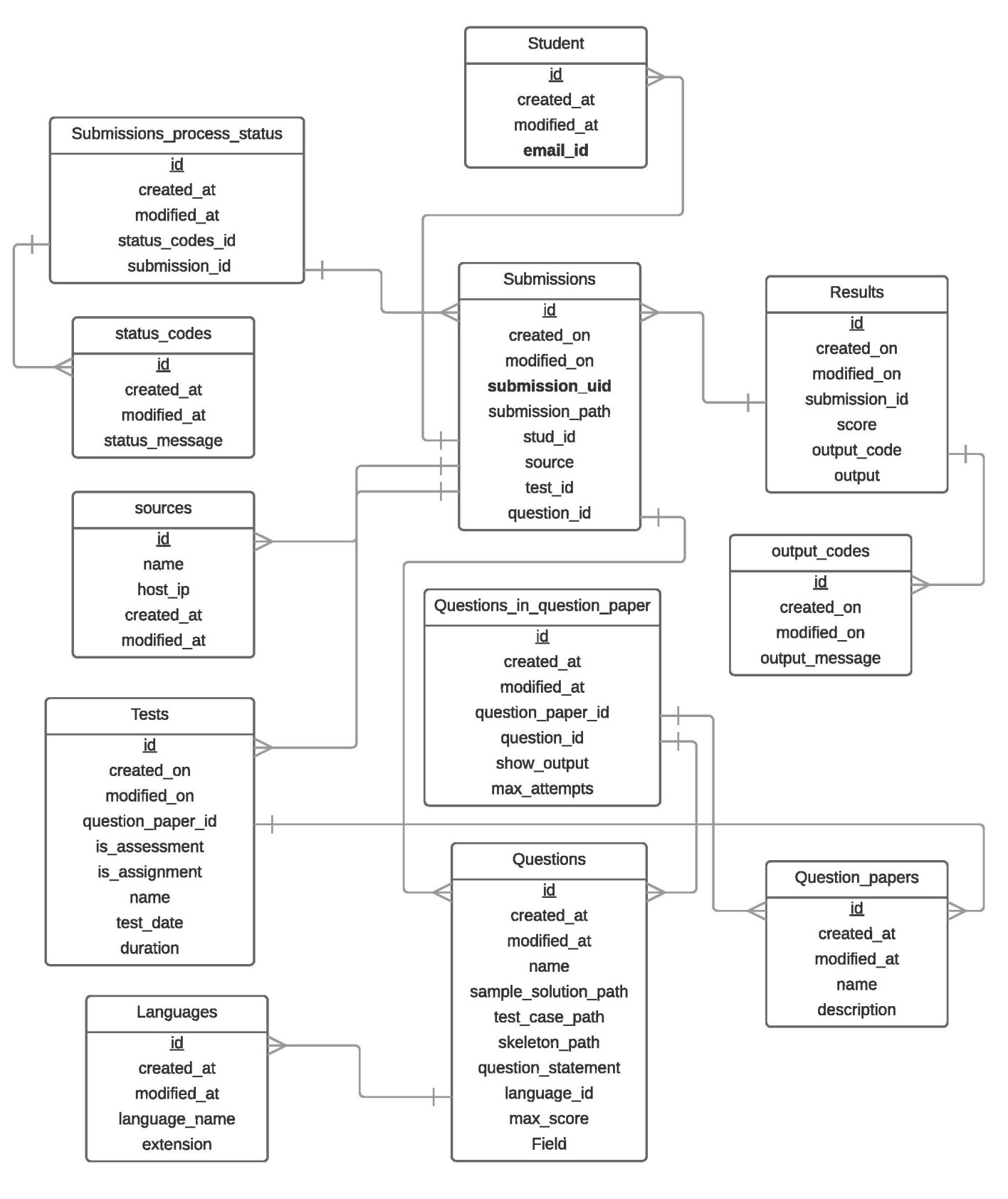


**Fig 3.4- Architecture diagram**

* 1. **SQL database structure**

Database design is the process of producing a detailed data model of a database. This data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

The term database design can be used to describe many different parts of the design of an overall database system. Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the database management system (DBMS).

****

**Fig 3.5- SQL database structure**

1. **IMPLEMENTATION**
   1. **Coding**

**Monitor.py**

import pyinotify

import os.path,time

import datetime

import pull\_grader\_client

from settings import TSPullGrader

class MyEventHandler(pyinotify.ProcessEvent):

def process\_IN\_CREATE(self, event):

print "CREATE event:", event.pathname

print "size===",os.path.getsize(event.pathname)

cd= datetime.datetime.fromtimestamp(os.path.getctime(event.pathname)) + datetime.timedelta(hours=-5) + datetime.timedelta(minutes=-30)

print cd.strftime('%Y-%m-%d %H:%M:%S')

if os.path.isfile(event.pathname):

filename=event.pathname.split('/')[-1]

filepath=event.pathname.split('/')[-2]

print "Filename: " + filename

print "Filepath: " + filepath

#Temporary fix, remove from final version.

if filename in ["FindSquare.java", "FindSquare.class"]:

return

time.sleep(3)

pull\_grader\_client.main(filepath, filename, cd.strftime('%Y-%m-%d %H:%M:%S'))

def main():

# watch manager

wm = pyinotify.WatchManager()

wm.add\_watch(TSPullGrader.SUBMISSION\_PATH, pyinotify.ALL\_EVENTS, rec=True,auto\_add=True)

# event handler

eh = MyEventHandler()

# notifier

notifier = pyinotify.Notifier(wm, eh)

notifier.loop()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Settings.py**

import logging

class TSGrader():

#Path where the submitted files are uploaded

SUBMISSION\_PATH = "/edx/var/edxapp/uploads"

TEMP\_PATH = "/tmp"

TESTCASE\_MARKER = "#####"

TESTCASE\_DELIMITER = "|"

SCORE\_MARKER = "/"

AUTO\_GRADED = {"invalid":0, "assessment":1, "assignment":2}

GRADE\_MESSAGE = {0: "Congratulations! Your code has compiled with no errors.", 1: "Check for Errors:"}

USE\_FORMATTER = 1

GRADING\_STATUS = {

"new\_upload": 1,

"processing": 2,

"graded": 3

}

DEFAULT\_LOG\_LEVEL = logging.INFO

LOGGER\_NAME = "TSgrader"

class TSEdx():

#EDX SQL database credentials

DB\_HOST = "localhost"

DB = "edxapp"

DB\_USER = "edxapp001"

DB\_PASSWORD = "password"

#EDX MongoDB credentials

MONGO\_HOST = "localhost"

MONGO\_PORT = 27017

MODULE\_TYPE = "edx\_sga"

TABLES = {

"student\_item" : "submissions\_studentitem",

"submission": "submissions\_submission",

"score": "submissions\_score",

"score\_summary": "submissions\_scoresummary",

"student\_module":"courseware\_studentmodule",

"student\_anonymousid": "student\_anonymoususerid"

}

SOURCE\_ID = 1

class RemoteGrader():

DB\_HOST = "213.219.36.251"

DB = "graderapp"

DB\_USER = "edxgrader"

DB\_PASSWORD = "grader123"

ERRORS = {

"invalid\_filename": 1,

"unmet\_constraints": 2,

"compilation\_error": 3,

"runtime\_error": 4,

"no\_errors": 5,

"server\_error": 6

}

SUBMISSION\_STATUS = {

"new\_submission": 1,

"processing": 2,

"graded": 3

}

**Validator.py**

def not\_multiple\_file\_extensions(file\_name):

return len(file\_name.split('.')) == 2

def not\_empty\_file\_name(file\_name):

return not file\_name.split('.')[0] == ""

def not\_empty\_file\_extension(file\_name):

return not file\_name.split('.')[1] == ""

def not\_wrong\_file\_extension(file\_name, valid\_extension):

print file\_name

print valid\_extension

print "."+file\_name.split('.')[1] == valid\_extension

return "."+file\_name.split('.')[1] == valid\_extension

def not\_wrong\_file\_name(file\_name, valid\_name):

print file\_name

print valid\_name

return file\_name == valid\_name

#must check not\_multiple\_file\_extensions(file\_name) function before others

def validate\_constraints(file\_name, constraints):

file\_name = file\_name.strip()

valid\_name = constraints["expected\_filename"].strip()

valid\_extension = constraints["expected\_extension"].strip()

return not\_wrong\_file\_extension(file\_name, valid\_extension) and not\_wrong\_file\_name(file\_name, valid\_name)

#must check not\_multiple\_file\_extensions(file\_name) function before others

def validate\_filename(file\_name):

return not\_multiple\_file\_extensions(file\_name) and not\_empty\_file\_name(file\_name) and not\_empty\_file\_extension(file\_name)

**DockerSandbox.js**

var DockerSandbox = function(timeout\_value,path,folder,vm\_name,compiler\_name,file\_name,output\_command,languageName,e\_arguments,extension, testExtension, language\_id, code, stdin\_data, test\_case\_path, file\_name)

{

this.timeout\_value = timeout\_value;

this.path = path;

this.folder = folder;

this.vm\_name = vm\_name;

this.compiler\_name = compiler\_name;

this.file\_name = file\_name;

this.output\_command = output\_command;

this.langName = languageName;

this.extra\_arguments = e\_arguments;

this.stdin\_data = stdin\_data;

this.extension = extension;

this.testExtension = testExtension;

this.code = code;

this.lanfuage\_id = language\_id;

this.test\_case\_path = test\_case\_path;

this.file\_name = file\_name;

this.test\_case\_file\_name = "";

}

DockerSandbox.prototype.run = function(success)

{

var sandbox = this

this.prepare( function(){

sandbox.execute(success);

});

}

DockerSandbox.prototype.prepare = function(success)

{

var exec = require('child\_process').exec;

var fs = require('fs');

var sandbox = this;

//Creating the file that is to be sent to docker by adding the input file

exec("mkdir "+ this.path+this.folder + " && cp "+this.path+"/Payload/\* "+this.path+this.folder+"&& chmod 777 "+ this.path+this.folder,function(st)

{

fs.writeFile(sandbox.path + sandbox.folder+"/" + sandbox.file\_name, sandbox.code,function(err)

{

if (err)

{

console.log(err);

}

else

{

console.log(sandbox.langName+" file was saved!");

exec("chmod 777 \'"+sandbox.path+sandbox.folder+"/"+sandbox.file\_name+"\'");

fs.writeFile(sandbox.path + sandbox.folder+"/inputFile", sandbox.stdin\_data,function(err)

{

if (err)

{

console.log(err);

}

else

{

console.log("Input file was saved!");

}

if(sandbox.test\_case\_path != "")

{//If path exists then it interprets it to be a test cases program

console.log("---------------------------------------------");

console.log(sandbox.folder);

console.log("cp \'"+sandbox.test\_case\_path+"\' \'"+sandbox.path+sandbox.folder+"\'");

exec("cp \'"+sandbox.test\_case\_path+"\' \'"+sandbox.path+sandbox.folder+"\'");

}

success();

});

}

});

});

}

//This function runs the Docker container and execute script.sh inside it. Return the output generated and delete the mounted folder

DockerSandbox.prototype.execute = function(success)

{

var exec = require('child\_process').exec;

var fs = require('fs');

var myC = 0; //variable to enforce the timeout\_value

var sandbox = this;

if(this.test\_case\_path != "")

{

var test\_case\_file = this.test\_case\_path.split("/");

console.log(test\_case\_file[test\_case\_file.length-1].split(".")[0]);

test\_case\_file = test\_case\_file[test\_case\_file.length-1].split(".")[0];

//This statement is executed for all the codes that are to be verified with test case

var st = this.path+'DockerTimeout.sh ' + this.timeout\_value + 's -u mysql -e \'NODE\_PATH=/usr/local/lib/node\_modules\' -i -t -v "' + this.path + this.folder + '":/usercode ' + this.vm\_name + " /usercode/script.sh \'" + this.compiler\_name + "\' " + this.file\_name + " \'" + this.output\_command+' '+test\_case\_file+ "\' " + this.extra\_arguments;

}

else

{

//This statement is executed for all the codes which require no test cases to execute

var st = this.path+'DockerTimeout.sh ' + this.timeout\_value + 's -u mysql -e \'NODE\_PATH=/usr/local/lib/node\_modules\' -i -t -v "' + this.path + this.folder + '":/usercode ' + this.vm\_name + ' /usercode/script.sh ' + this.compiler\_name + ' ' + this.file\_name + ' ' + this.output\_command+ ' ' + this.extra\_arguments;

}

//log the statement in console

console.log(st);

//execute the Docker, This is done ASYNCHRONOUSLY

exec(st);

console.log("------------------------------")

//Check For File named "completed" after every 1 second

var intid = setInterval(function()

{

//Displaying the checking message after 1 second interval, testing purposes only

//console.log("Checking " + sandbox.path+sandbox.folder + ": for completion: " + myC);

myC = myC + 1;

fs.readFile(sandbox.path + sandbox.folder + '/completed', 'utf8', function(err, data) {

//if file is not available yet and the file interval is not yet up carry on

if (err && myC < sandbox.timeout\_value)

{

//console.log(err);

return;

}

//if file is found simply display a message and proceed

else if (myC < sandbox.timeout\_value)

{

console.log("DONE")

//check for possible errors

fs.readFile(sandbox.path + sandbox.folder + '/errors', 'utf8', function(err2, data2)

{

if(!data2) data2=""

console.log("Error file: ")

console.log(data2)

console.log("Main File")

console.log(data)

var lines = data.toString().split('\*-COMPILE BOX::ENDOFOUTPUT-\*')

data=lines[0]

var time=lines[1]

console.log("Time: ")

console.log(time)

success(data,time,data2)

});

//return the data to the calling functoin

}

//if time is up. Save an error message to the data variable

else

{

//Since the time is up, we take the partial output and return it.

fs.readFile(sandbox.path + sandbox.folder + '/logfile.txt', 'utf8', function(err, data){

if (!data) data = "";

data += "\nExecution Timed Out";

console.log("Timed Out: "+sandbox.folder+" "+sandbox.langName)

fs.readFile(sandbox.path + sandbox.folder + '/errors', 'utf8', function(err2, data2)

{

if(!data2) data2=""

var lines = data.toString().split('\*---\*')

data=lines[0]

var time=lines[1]

console.log("Time: ")

console.log(time)

success(data,data2)

});

});

}

//now remove the temporary directory

console.log("ATTEMPTING TO REMOVE: " + sandbox.folder);

console.log("------------------------------")

exec("rm -r " + sandbox.folder);

clearInterval(intid);

});

}, 1000);

}

module.exports = DockerSandbox;

**compilers.js**

/\*

This file stores the compiler/interpretor details that are provided to DockerSandbox.sh by the app.js

The index is the key field,

First column contains the compiler/interpretor that will be used for translation

Second column is the file name to use when storing the source code

Third column is optional, it contains the command to invoke the compiled program, it is used only for compilers

Fourth column is just the language name for display on console, for verbose error messages

Fifth column is optional, it contains additional arguments/flags for compilers

Sixth column is the code extention of that language file

Seventh columnb is the test case extention and it is optional

You can add more languages to this API by simply adding another row in this file along with installing it in your

Docker VM.

\*/

exports.compilerArray= [ ["python", "file.py", "", "Python", "", "", ""],

["javac -cp ./usercode/junit-4.10.jar:./usercode: ", "file.java", "java -cp ./usercode/junit-4.10.jar:./usercode/: org.junit.runner.JUnitCore", "JUnit", "", ".java", "Test.class"], ["javac","file.java","\'./usercode/javaRunner.sh\'","Java","", ".java", ""],

];

**app.js**

var express = require('express');

var arr = require('./compilers');

var sandBox = require('./DockerSandbox');

var app = express.createServer();

var port=10001;

var ExpressBrute = require('express-brute');

var store = new ExpressBrute.MemoryStore();

// stores state locally, don't use this in production

var bruteforce = new ExpressBrute(store,{

freeRetries: 50,

lifetime:-10

});

app.use(express.static(\_\_dirname));

app.use(express.bodyParser());

app.all('\*', function(req, res, next)

{

res.header('Access-Control-Allow-Origin', '\*');

res.header('Access-Control-Allow-Methods', 'PUT, GET, POST, DELETE, OPTIONS');

res.header('Access-Control-Allow-Headers', 'Content-Type');

next();

});

function random(size) {

//returns a crypto-safe random

return require("crypto").randomBytes(size).toString('hex');

}

app.post('/compile',bruteforce.prevent,function(req, res)

{

console.log(req.body)

var languageId = req.body.languageId;

var code = req.body.code;

var stdin = req.body.stdin;

var testCasePath = req.body.testCasePath;

var fileName = req.body.fileName;

var folder = 'temp/' + random(10); //folder in which the temporary folder will be saved

var path=\_\_dirname+"/"; //current working path

var vm\_name='virtual\_machine'; //name of virtual machine that we want to execute

var timeout\_value=20;//Timeout Value, In Seconds

arr.compilerArray[languageId][1] = fileName;

//details of this are present in DockerSandbox.js

var sandboxType = new sandBox(timeout\_value,path,folder,vm\_name,arr.compilerArray[languageId][0],arr.compilerArray[languageId][1],arr.compilerArray[languageId][2],arr.compilerArray[languageId][3],arr.compilerArray[languageId][4],arr.compilerArray[languageId][5],arr.compilerArray[languageId][6], languageId, code, stdin, testCasePath, fileName);

//data will contain the output of the compiled/interpreted code

//the result maybe normal program output, list of error messages or a Timeout error

sandboxType.run(function(data,exec\_time,err)

{

console.log("\*\*\*\*"+{output:data, langid: languageId,code:code, errors:err, time:exec\_time}+"\*\*\*\*\*\*\*\*");

console.log(data);

console.log(err);

res.send({output:data, langid: languageId,code:code, errors:err, time:exec\_time});

});

});

app.get('/', function(req, res)

{

res.sendfile("./index.html");

});

console.log("Listening at "+port)

app.listen(port);

* 1. **User manual**

Setting Up New Tests on the Automatic Grader

Before the day of the test:

● Select the questions you want to use: add new questions or use existing ones.

● Select a question paper for the test: create a new question paper or use an existing one.

● Schedule a test.

● Add questions to the client platform, and set them up for file uploads and automatic grading.

During the test:

● Monitor the submissions for the test.

● Get real time statistics on submissions and outputs.

After the test:

● Download a summary of the results for the test.

● Generate emails to each student with the results of their coding submissions.

**Step By Step**

Enabling File Uploads on EDX

To allow students to upload files on EDX, an advanced component called the EDX Staff Graded Assignment (EDX SGA) module is used. By default, this module is not available while creating course content.

To be able to use the module, the course team need to change the Advanced Settings for the course in Studio:

● Open the course you are authoring in Studio and select "Settings" ⇒ "Advanced

Settings.

● Navigate to the section titled "Advanced Module List".

● Add "edx\_sga" to module list. If the list is not empty, separate each module with a “,” (comma).

● Studio should save your changes automatically.

To add questions to your course using the SGA module:

● Go to the Course Outline for the course you are authoring.

● Create a Section, Sub­section and Unit, if you haven't already.

● While editing the Unit, in the "Add New Component" interface, you should now see an

"Advanced" button.

● Click "Advanced" and choose "Staff Graded Assignment". This will add a component to your question that will allow students to upload files.

Keep in mind:

● Since the SGA module doesn't support text within the problem, it is recommended to precede the SGA component with a Text or HTML component with the question statement and other instructions for the student.

● The SGA module doesn’t support more than one upload component for each Course

Section. Create a new section for each problem that need file uploads within a course.

For more information on the EDX­SGA module, visit: https://github.com/mitodl/edx­sga

**Setting Up Tests to Use the Automatic Grader**

The automatic grader only processes content (i.e., questions, test papers) that is registered on its database. If you want an assessment or assignment to be automatically graded you must use the GUI to register the questions, create a question paper, and schedule a test using that question paper.

The interface will then generate some content for you that you must copy onto your problem statement. This is essential, since without this generated content, EDX would not be able to recognise that the question must be automatically graded.

**Before the Test :**

The following steps must be completed before the assessment to ensure the submissions are automatically graded.

**Adding New Questions to the Automatic Grader**

If you have a new question that needs to be autograded, it needs to be registered with the autograder. To do so, use the Question Manager :

● Click on “Add a Question”.

● Fill out the required details for the question such as its name, the maximum possible score, the question statement, the expected filename (with the extension) and the language the solutions will be in.

● Upload the necessary files: the skeleton, the testcases and a sample solution. Keep in mind the test case file must be the executable file.

● Click on “Save” to complete the process.

**Creating a Question Paper**

To use a set of questions in a test, a question paper must be created and added to the system. This can be done using the Question Paper Manager:

● Click on “Add a Question Paper”

● Fill the details about the question paper such as the name, the description, whether the output should be shown to users and the maximum number of attempts allowed for questions in that question paper.

● Select the questions to be used in the question paper. From the list of questions on the select, click “Add to Pool” to add a question to the pool of questions that will be used in the question paper. The list of questions in the new question paper will appear on the right.

● When you are satisfied with your select, click on “Save” to create the question paper.

**Scheduling a Test**

Scheduling a test allows you to monitor submissions in real time and generate emails with the students results. Without scheduling the test, you also won’t be able to add the html content required in your question statements (on EDX). To schedule a test, use the Test Manager:

● Click on “Add a Test”.

● Fill in the required details: name of the test, the date of the test (YYYY­mm­dd HH:MM:SS), the test duration (in seconds) and whether it is an assessment or assignment.

● Choose the question paper to be used for the test, and click “Save”.

**Adding Questions To EDX**

The next step is to add the questions to the client platform, where they will be visible to the students, and configuring them so they can be automatically graded.

● Navigate to your course, create a new Section, and a new Unit.

● In the unit, create a HTML Text component for the problem statement. Click on “Edit”, and then click on “HTML”. This is where the question statement in HTML format must be added.

● Go to the HTML Content Generator and click on the “Select” button next to the test. This will show you the list of questions that are being used.

● Click on “Generate HTML Content” and you will see a modal window with the problem statement enclosed in html tags, along with important information that is required by the autograding module. This information will be hidden from students.

● Copy the content and paste it onto the HTML editor window on EDX. Format it as required and save your changes.

● While editing the Unit, in the "Add New Component" interface, you should see an

"Advanced" button.

● Click "Advanced" and choose "Staff Graded Assignment". This will add a component to your question that will allow students to upload files.

● EDX should save your changes automatically.

● Repeat all of the above steps for each question on the test.

Keep in mind:

● Since the SGA module doesn't support text within the problem, it is recommended to precede the SGA component with a Text or HTML component with the question statement and other instructions for the student.

● The SGA module doesn’t support more than one upload component for each Course

Section. Create a new section for each problem that need file uploads within a course.

**On the Day of the Test:**

**Publish the Test and Set the Appropriate Due Date**

On Studio, publish the test so that it is visible to the students. Make sure the due date for the test is set properly, otherwise students may not be able to upload their files.

**Monitor Submissions Live**

Once the test starts, you can navigate to the GUI for Live Monitoring. Here you will be able to see submissions to the automatic grader in real time, as well as statistics about the grading results. To start the monitoring:

● Go to the Live monitoring GUI, and click on “Start Live Monitoring”. Once the monitoring has started the page will be updated every 5 seconds.

● You will be able to see information about each submission (email, the question for which the answer was submitted and the result after grading).

● You will also be able to see statistics such as the total number of submissions, the number of submissions that raised a compilation or runtime error, etc.

● To stop the monitoring, click on “Stop Live Monitoring”.

● If you want to see the monitoring statistics for a test that has already taken place, click on “Get Summary For a Test”, and select the test from the dropdown. Test will only be available in this list after the date of the test has passed.

**After the Test:**

**Get a Summary of the Results**

After the test has been conducted, a summary of the coding scores and results can be downloaded from the Results Generator. Click on “Download Results” to get a CSV file that has information about each submission, such as the student’s email, the question for which the answer was submitted, the maximum possible score, the score for that submission and the grader’s output.

**Automatically Generate Results Emails**

In the same GUI, mails can be automatically generated that will tell each student his total coding score for the test and the output for each of his submissions. To do so:

● Click on “Generate Results Emails” button next to the test.

● A modal window will appear where you can edit the HTML template used for generating the mails. Make the changes as required.

● Enter the email subject in the space provided.

● Click on “Download mails.csv” to get the CSV file with the personalised mails for each student who participated in the test.

**Sending Results Emails Using the CSV File and ThunderBird MailMerge.**

Thunderbird is a mail client that allows a user to generate mails from a CSV file like the one the

Results Generator provides.

Write the mail:

● Open ThunderBird. Click on the “Write” tab to start writing a new mail.

● In the space for the ‘To’ address, type “{{ email }}”.

● In the space for the email subject, type “{{ subject }}”.

● In the email body, type “{{ content }}”.

● From the toolbar, select “Options => Delivery Format => Rich Text (HTML) only”

Select MailMerge and configure the appropriate options:

● Click on the dropdown menu near the “Send” button and you will see an option “MailMerge”. Click on “MailMerge”. A modal window will appear where you can configure the settings required to use mail merge.

● For “Source”, select “CSV”.

● For “Deliver Mode”, select “Send Later”. This will ensure that the generated mails will first be sent to your outbox. There you can inspect their contents before sending the mails to the students.

● For “File”, use the file browser to select the CSV file that you downloaded from the

Results Generator (mails.csv).

● Set the “Character Set” to “UTF­8”, “Field Delimiter” to “Tab”, and “Text Delimiter” to

“Single Quote”

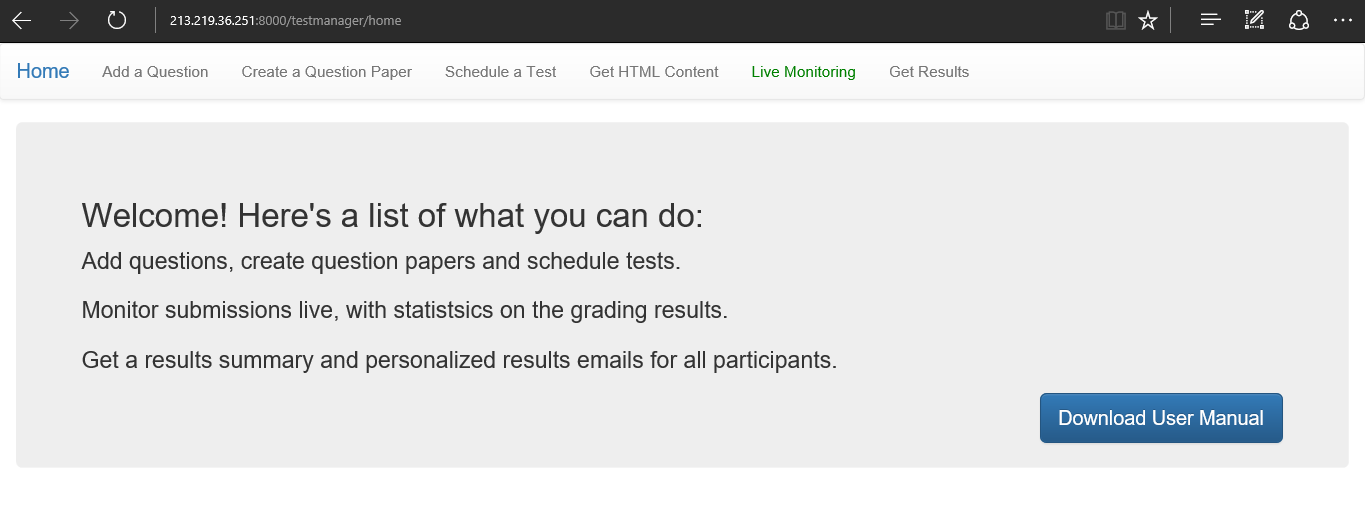
● Once all the configurations are complete, click on “OK” to start generating the mails.

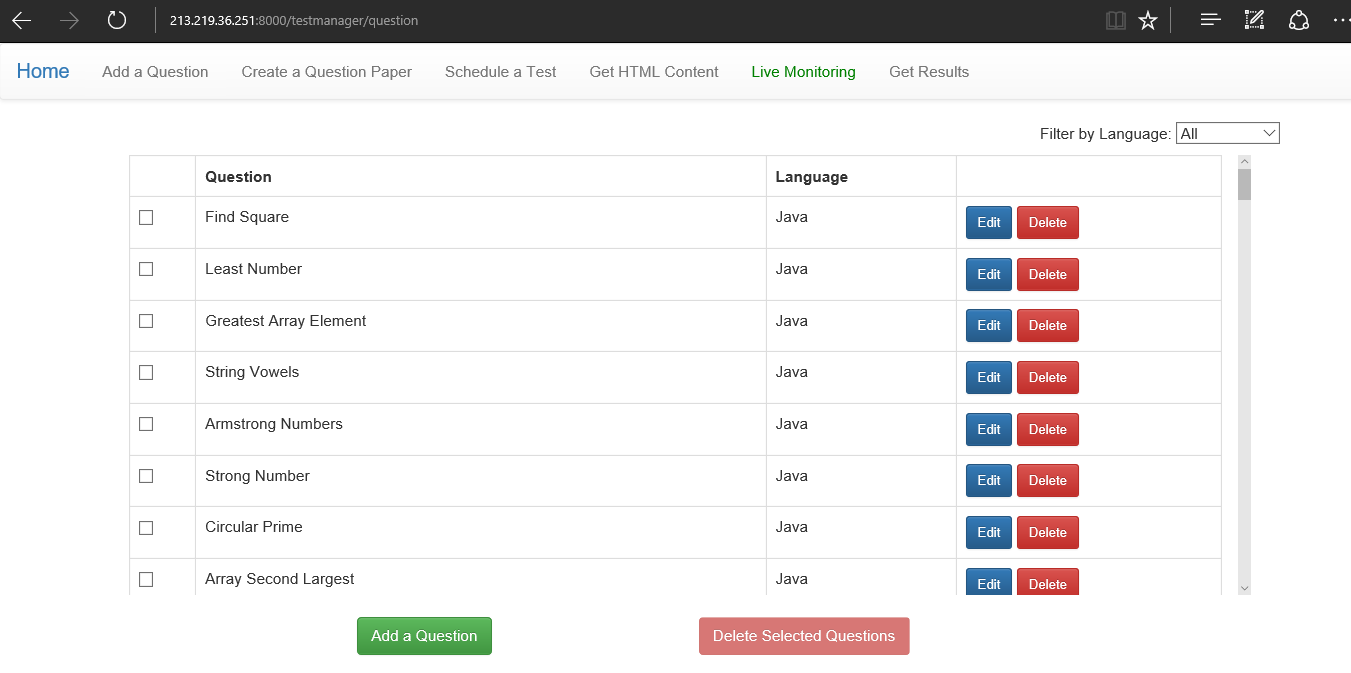
Inspect the generated mails and send:

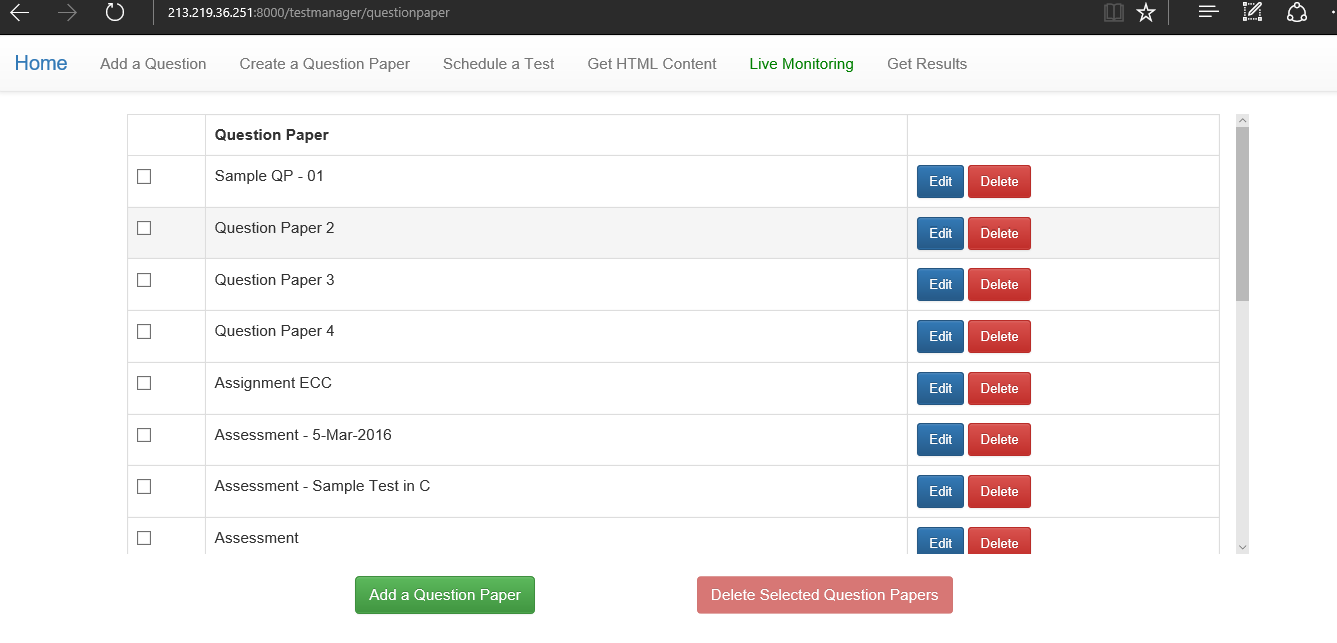
● The mails will be sent to your Outbox.

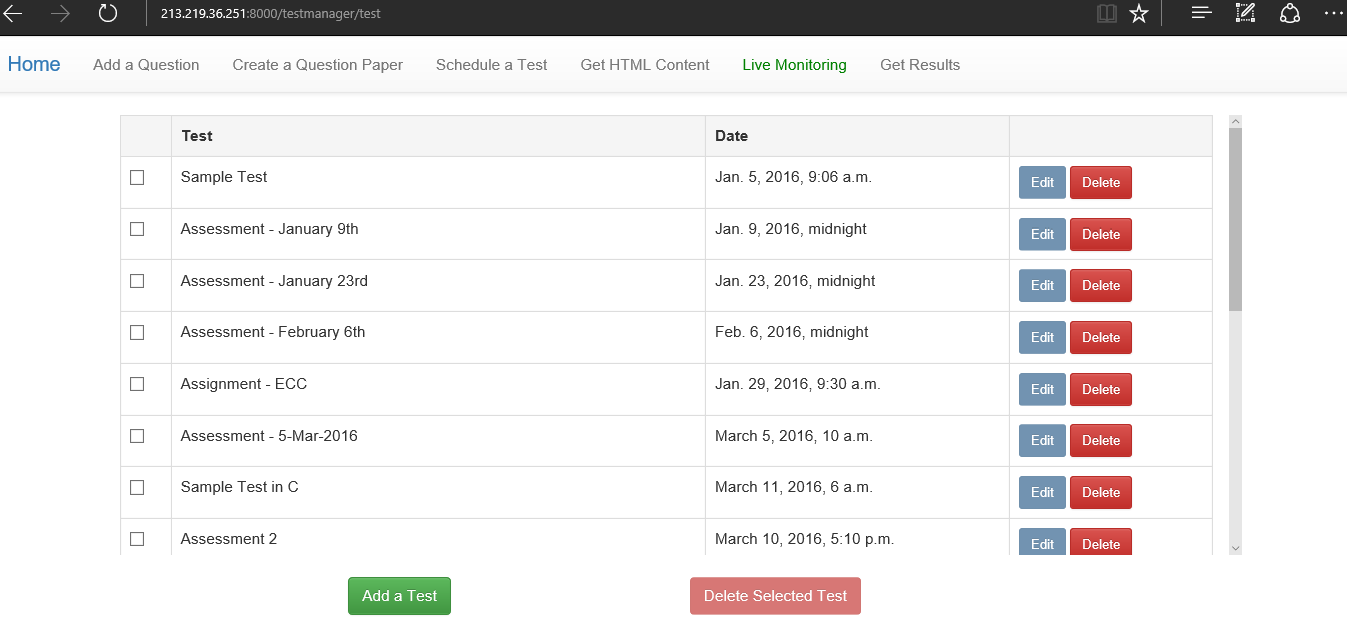
● Inspect their content and send them to the students. You can also delete them if there are any errors.

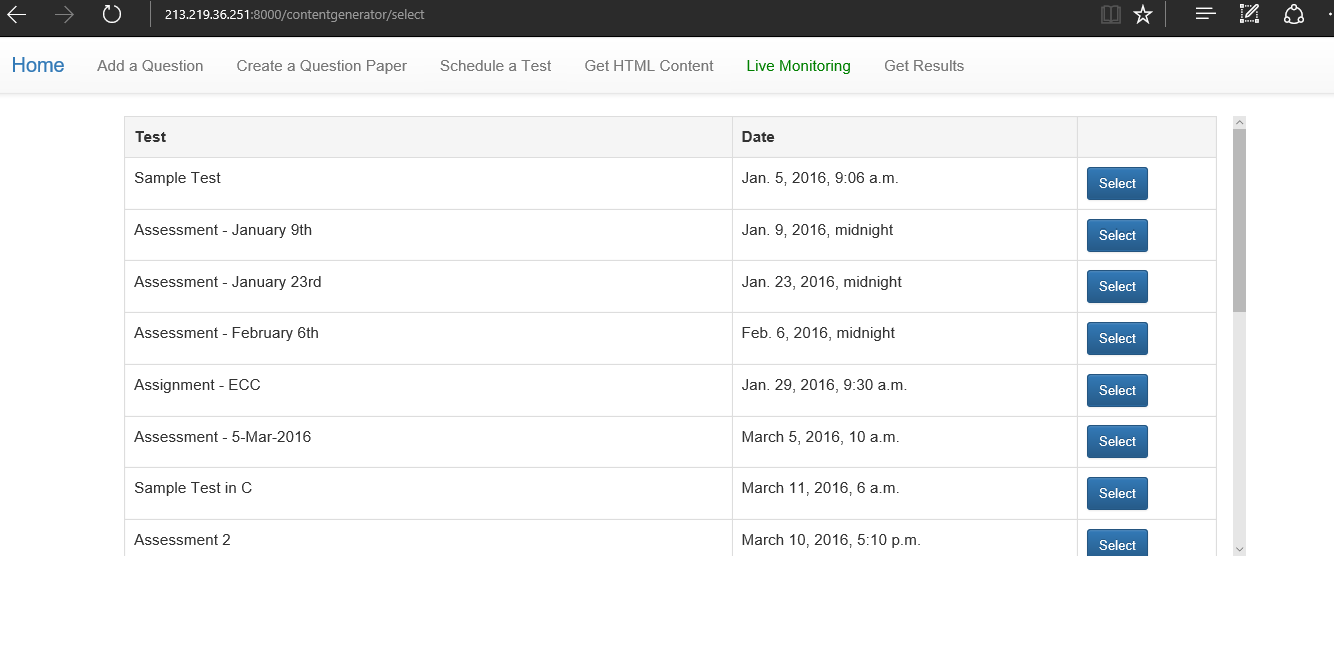
* 1. **Screenshots**

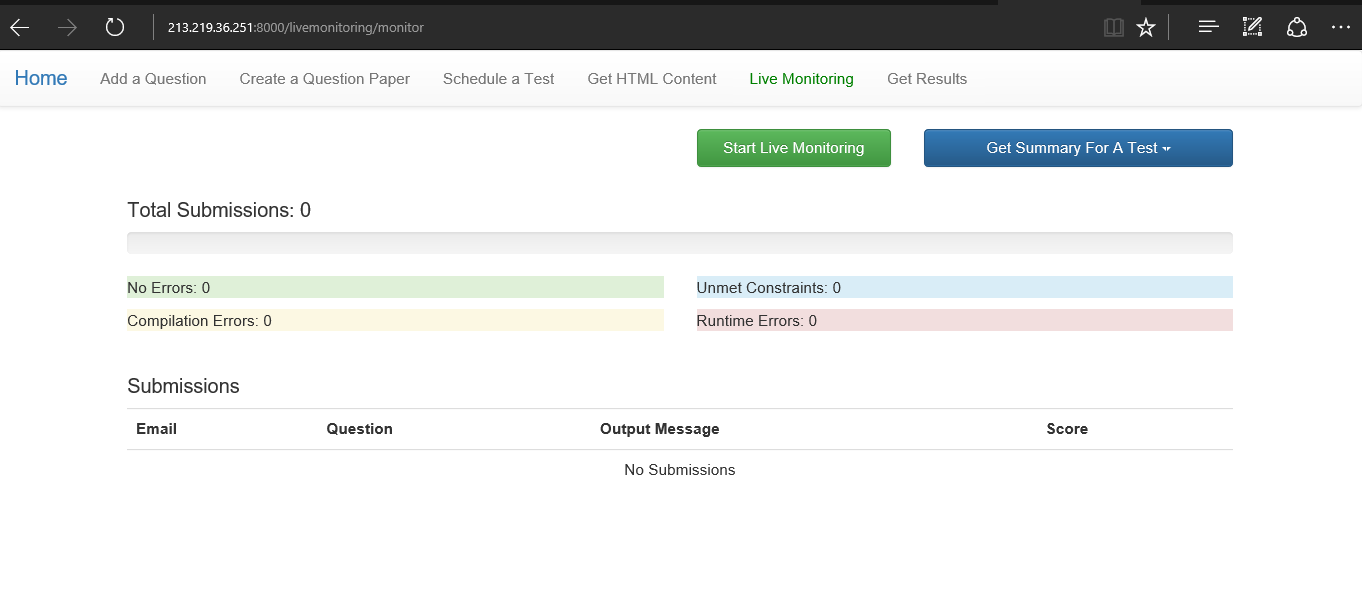
**Fig 4.1- Welcome screen for Test Preperation**

**Fig 4.2- Add a question screen**

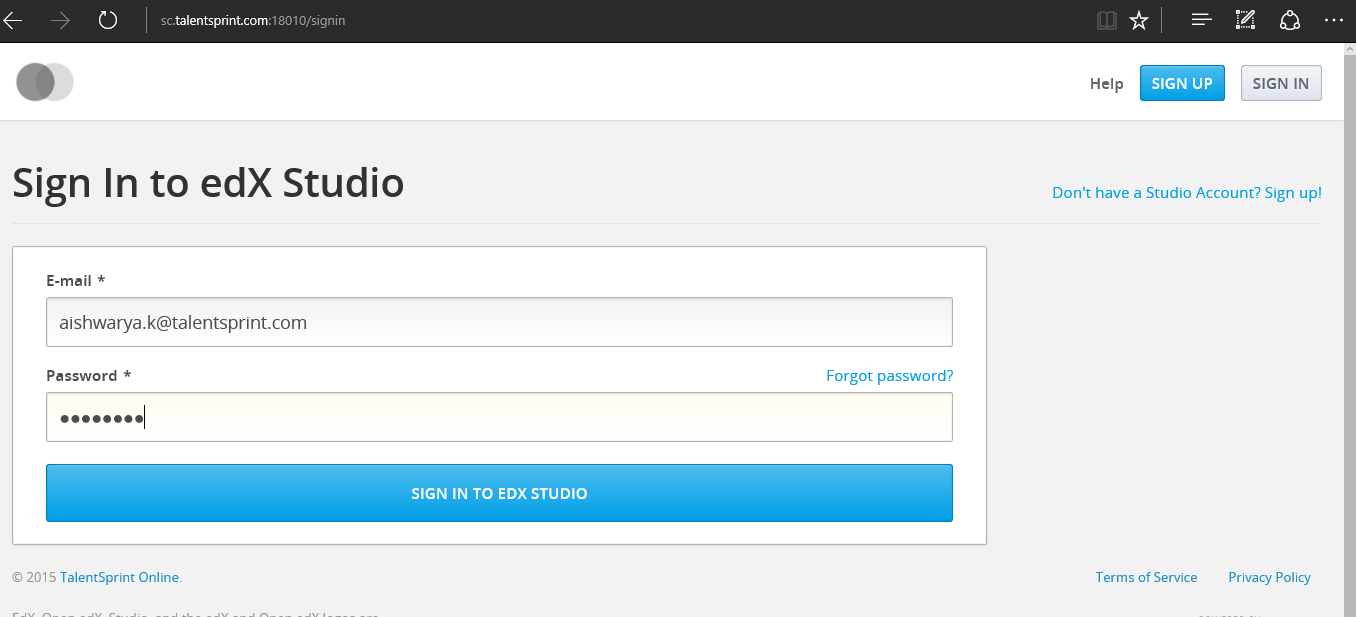
**Fig 4.3- Create a question paper screen**

**Fig 4.4- Schedule a test screen**

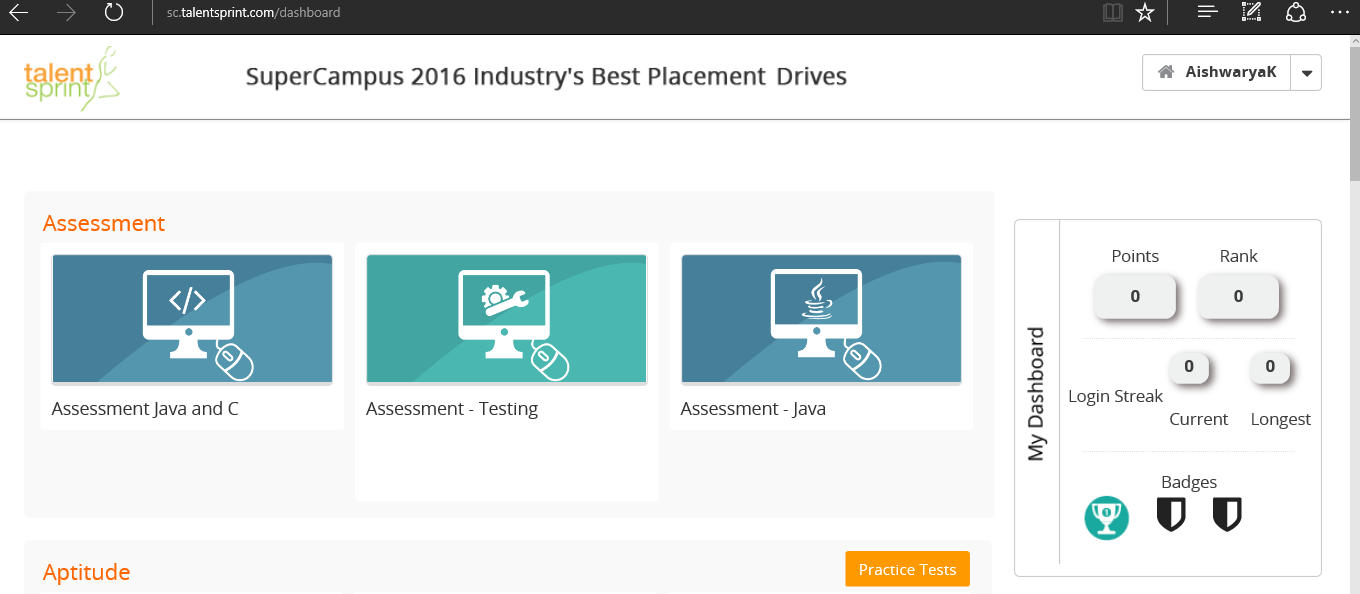
**Fig 4.5- Get HTML content screen**

**Fig 4.6- Live monitoring screen**

**Fig 4.7- Get results screen**



**Fig 4.8- Student Login screen**



**Fig 4.8- Student home screen**

* 1. **Testing**

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.

Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

This tutorial will give you a basic understanding on software testing, its types, methods, levels, and other related terminologies.

**4.4.1 Functional testing**

Functional testing is a quality assurance (QA) process and a type of black-box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing). Functional testing usually describes *what* the system does.

Functional testing does not imply that you are testing a function (method) of your module or class. Functional testing tests a slice of functionality of the whole system.

Functional testing differs from system testing in that functional testing "*verifies* a program by checking it against ... design document(s) or specification(s)", while system testing "*validate[s]* a program by checking it against the published user or system requirements" .

Functional testing has many types:

* Smoke testing
* Sanity testing
* Regression testing
* Usability testing

Functional testing typically involves six steps

1. The identification of functions that the software is expected to perform
2. The creation of input data based on the function's specifications
3. The determination of output based on the function's specifications
4. The execution of the test case
5. The comparison of actual and expected outputs
6. To check whether the application works as per the customer need.

**4.4.2 Unit testing**

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for component testing.

Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

## Benefits

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

**Finds problems early:** Unit testing finds problems early in the development cycle. This includes both bugs in the programmer's implementation and flaws or missing parts of the specification for the unit. The process of writing a thorough set of tests forces the author to think through inputs, outputs, and error conditions, and thus more crisply define the unit's desired behavior. The cost of finding a bug before coding begins or when the code is first written is considerably lower than the cost of detecting, identifying, and correcting the bug later; bugs may also cause problems for the end-users of the software. Some argue that code that is impossible or difficult to test is poorly written, thus unit testing can force developers to structure functions and objects in better ways.

In test-driven development (TDD), which is frequently used in both extreme programming and scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is considered to be a bug either in the changed code or the tests themselves. The unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development team of the problem before handing the code off to testers or clients, it is still early in the development process.

**Facilitates change:** Unit testing allows the programmer to refactor code or upgrade system libraries at a later date, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract.

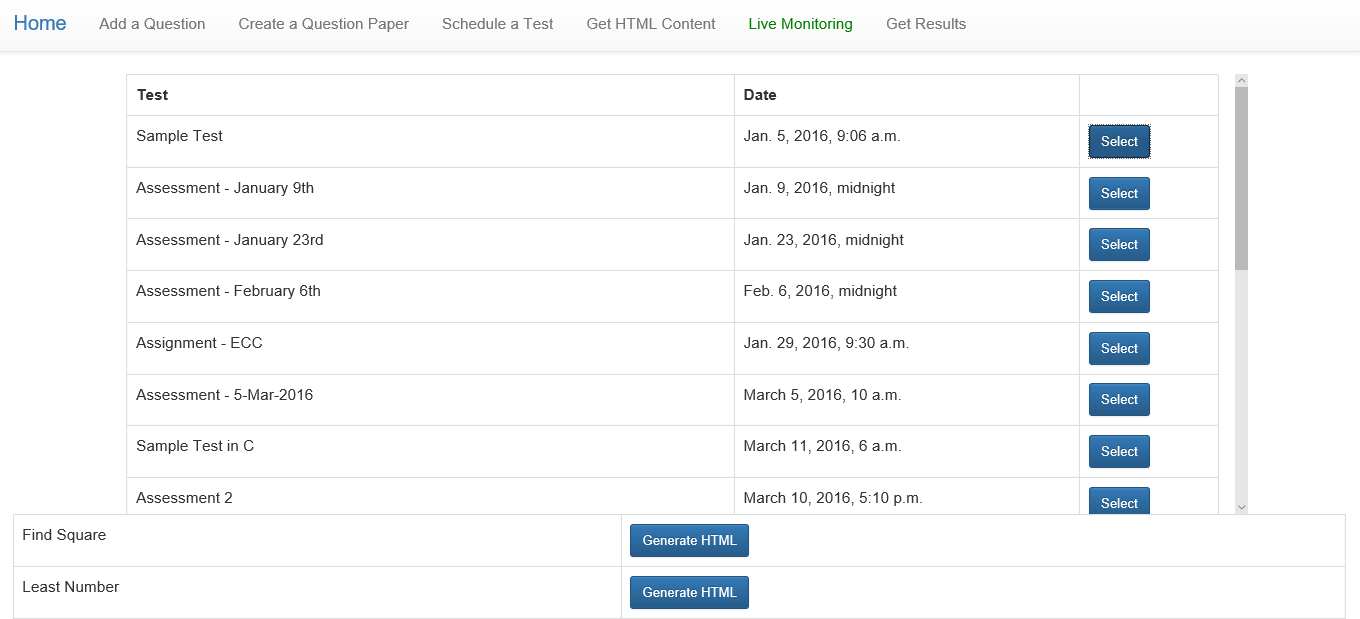
Simplifies integration: Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

### Documentation: Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit, and how to use it, can look at the unit tests to gain a basic understanding of the unit's interface (API).

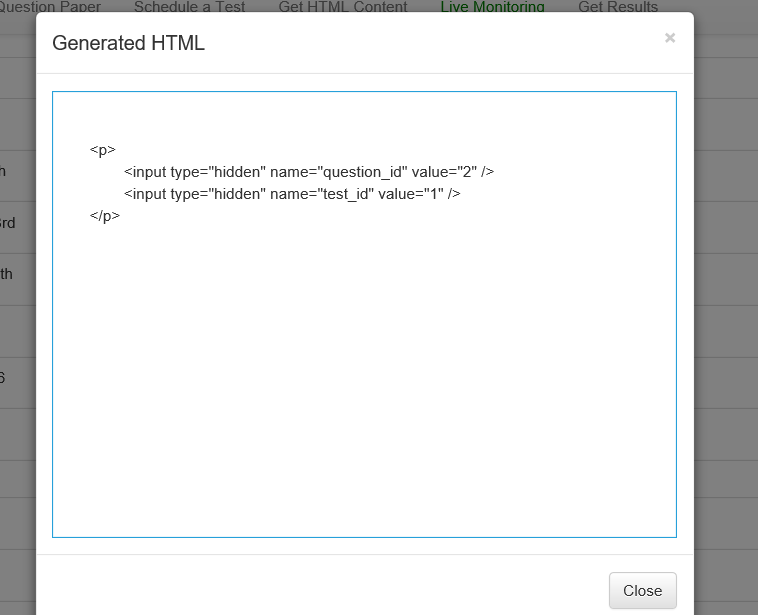
Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

1. **OPERATING SEQUENCE**

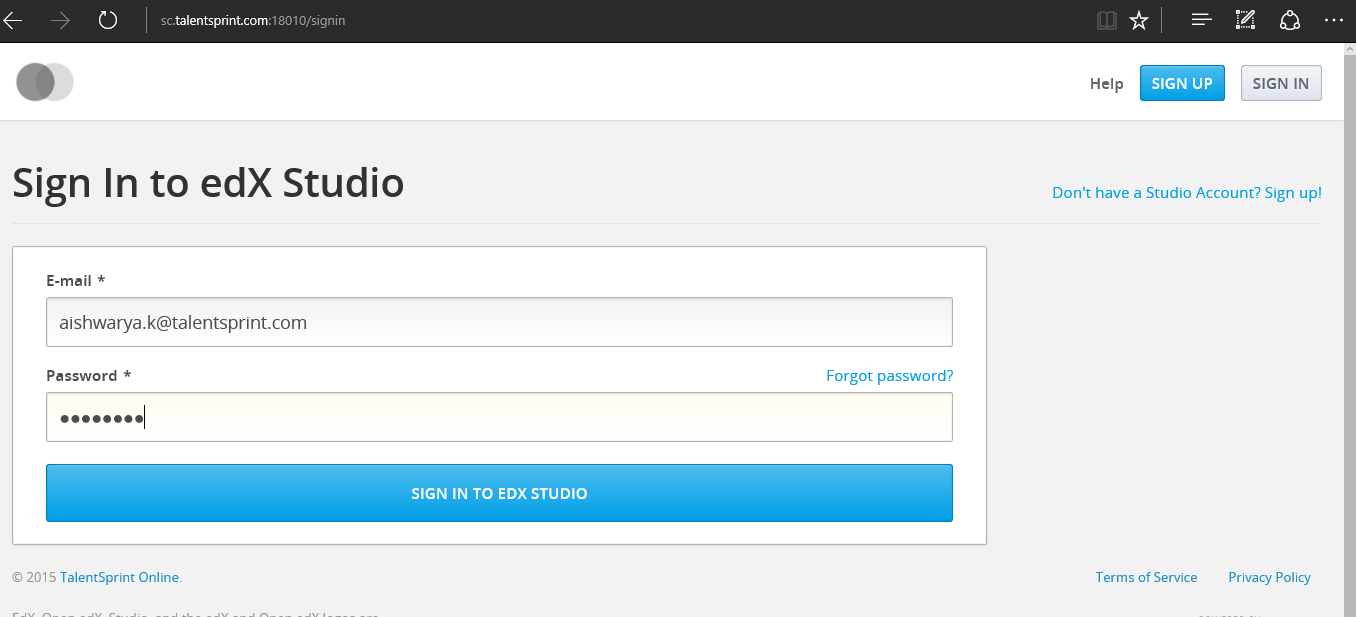
INPUT:



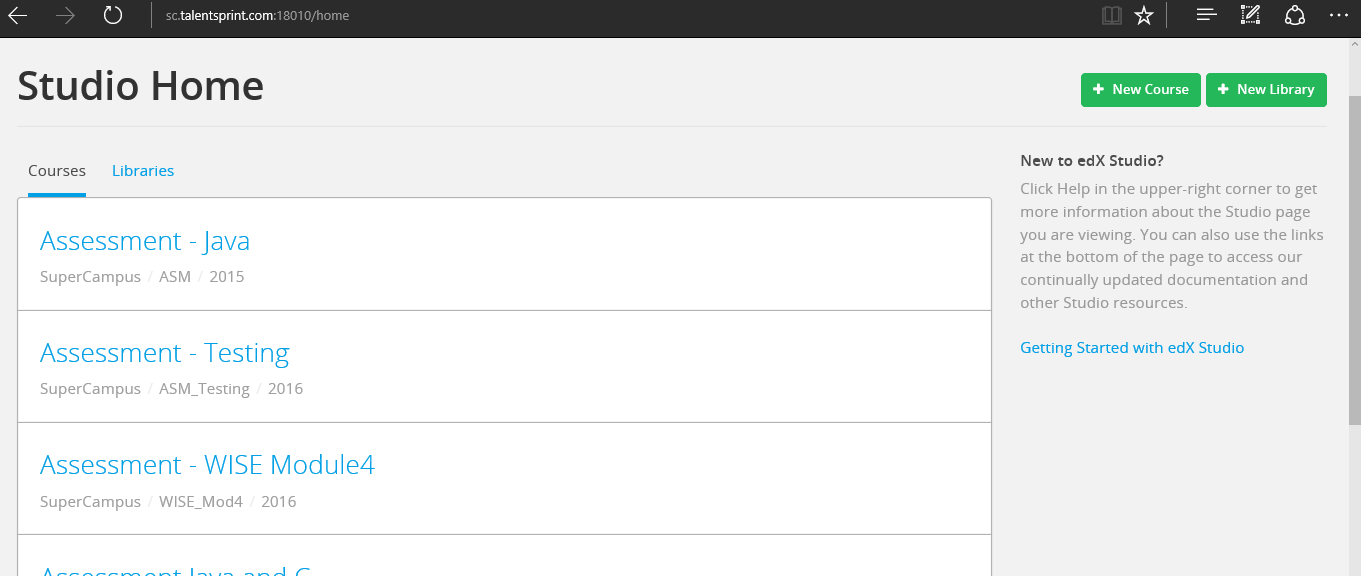
**Fig 5.1- Get HTML Content for requied Code**



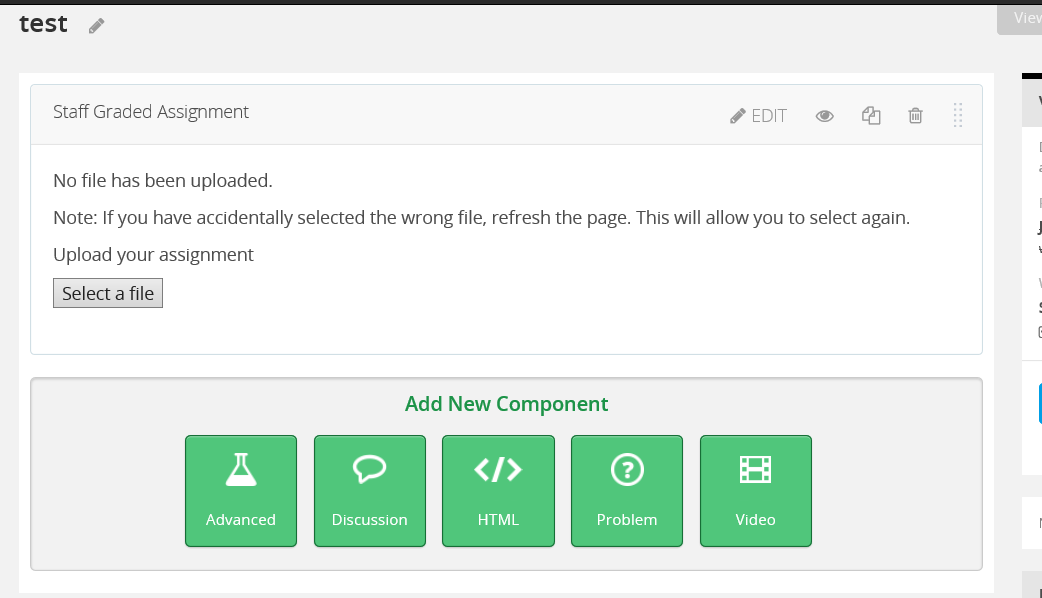
**Fig 5.2- Display of HTML content to be inserted**



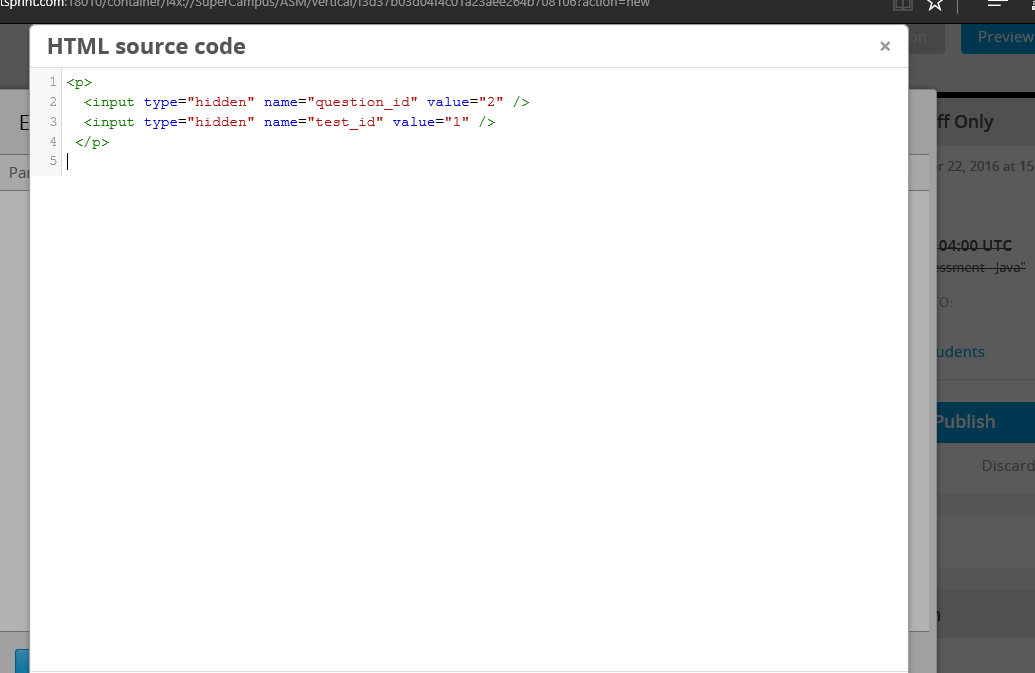
**Fig 5.3- Instructor login screen**



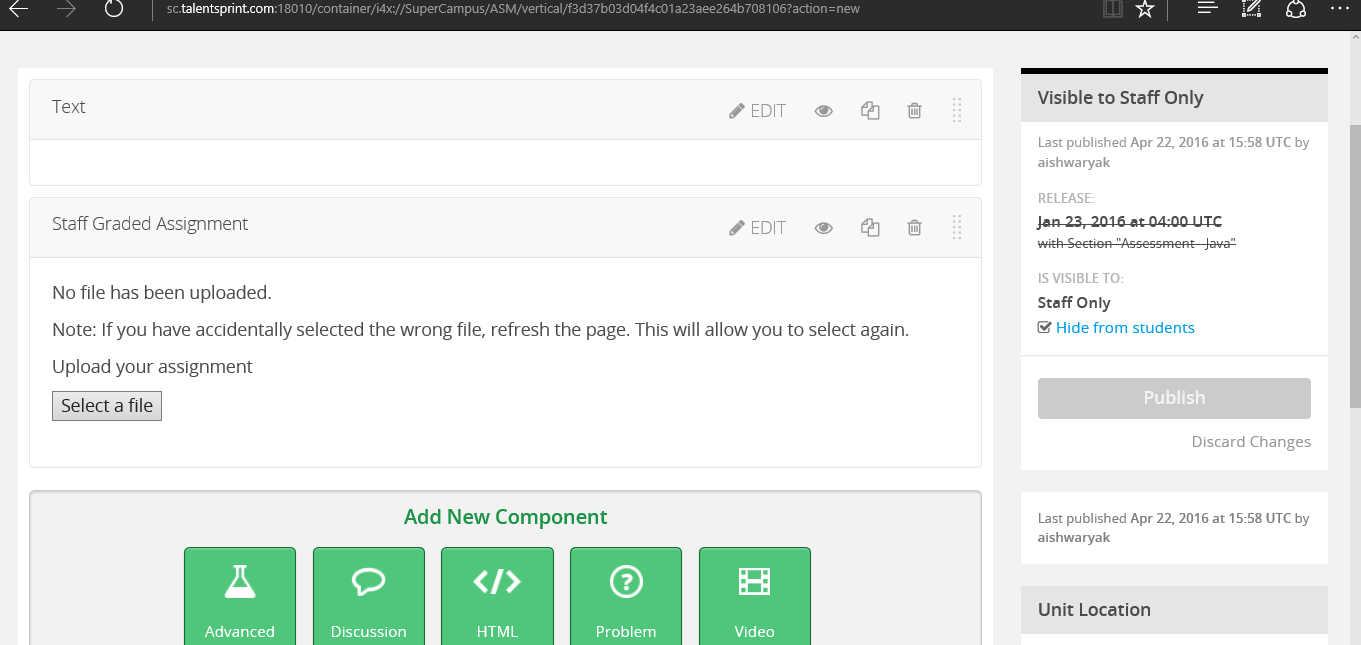
**Fig 5.4- Instructor home screen**



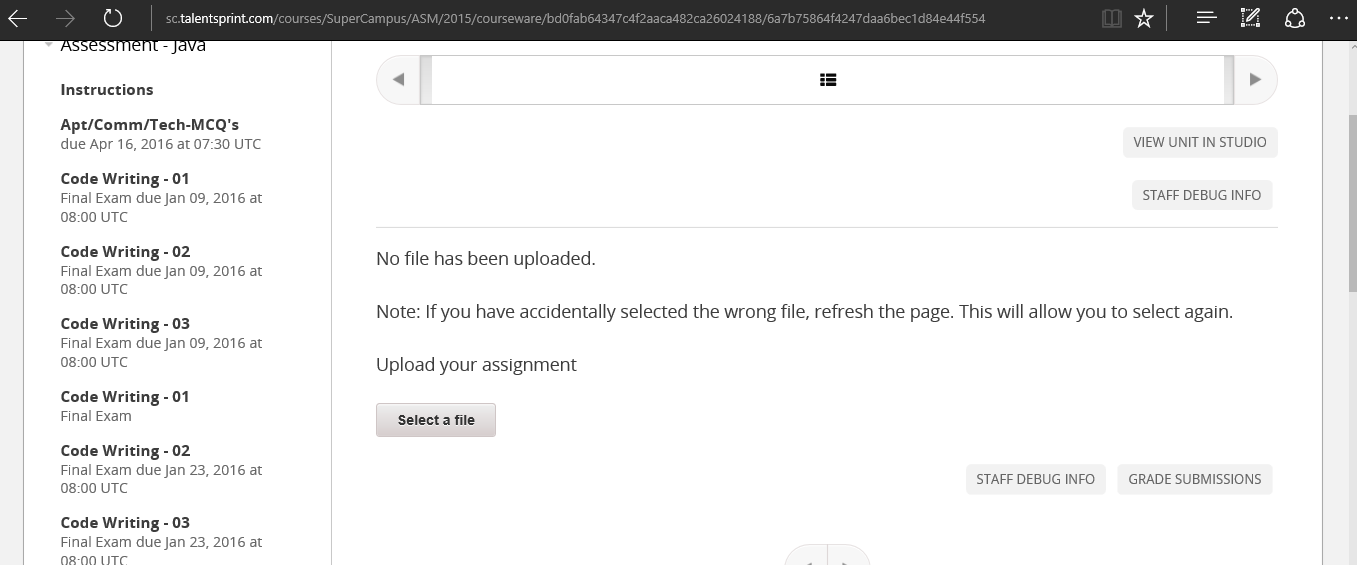
**Fig 5.5- Insertion of required module in selected section**



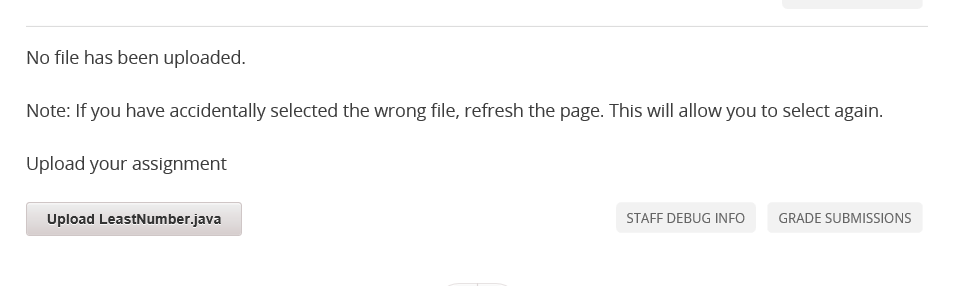
**Fig 5.6- Insertion of HTML code in the selected question**



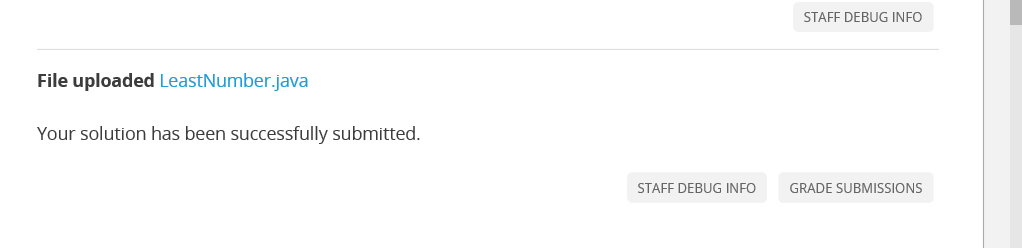
**Fig 5.7- Prepare the question to be published**



**Fig 5.8- Students view of published question**

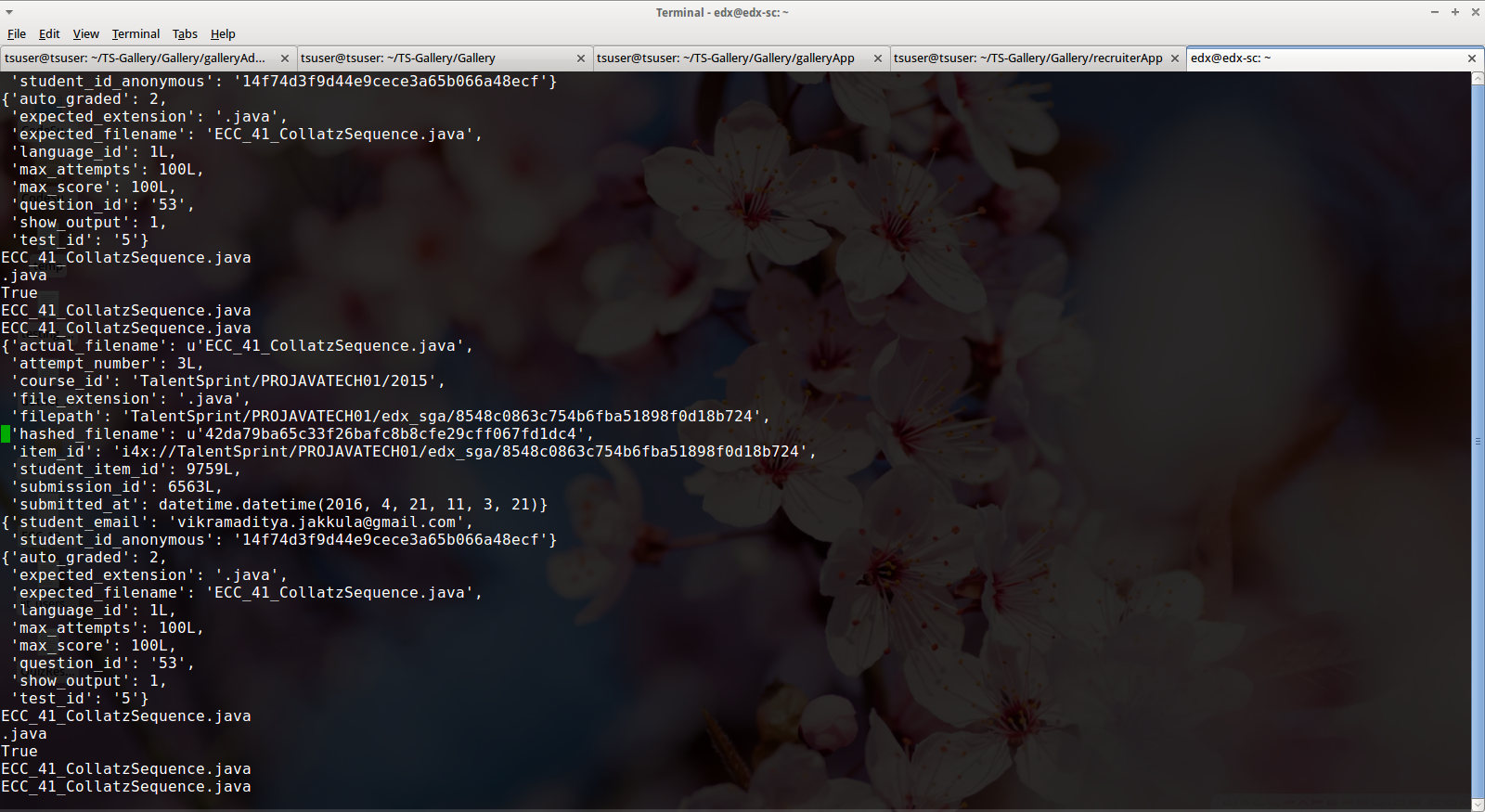


**Fig 5.9- Student uplading the code**

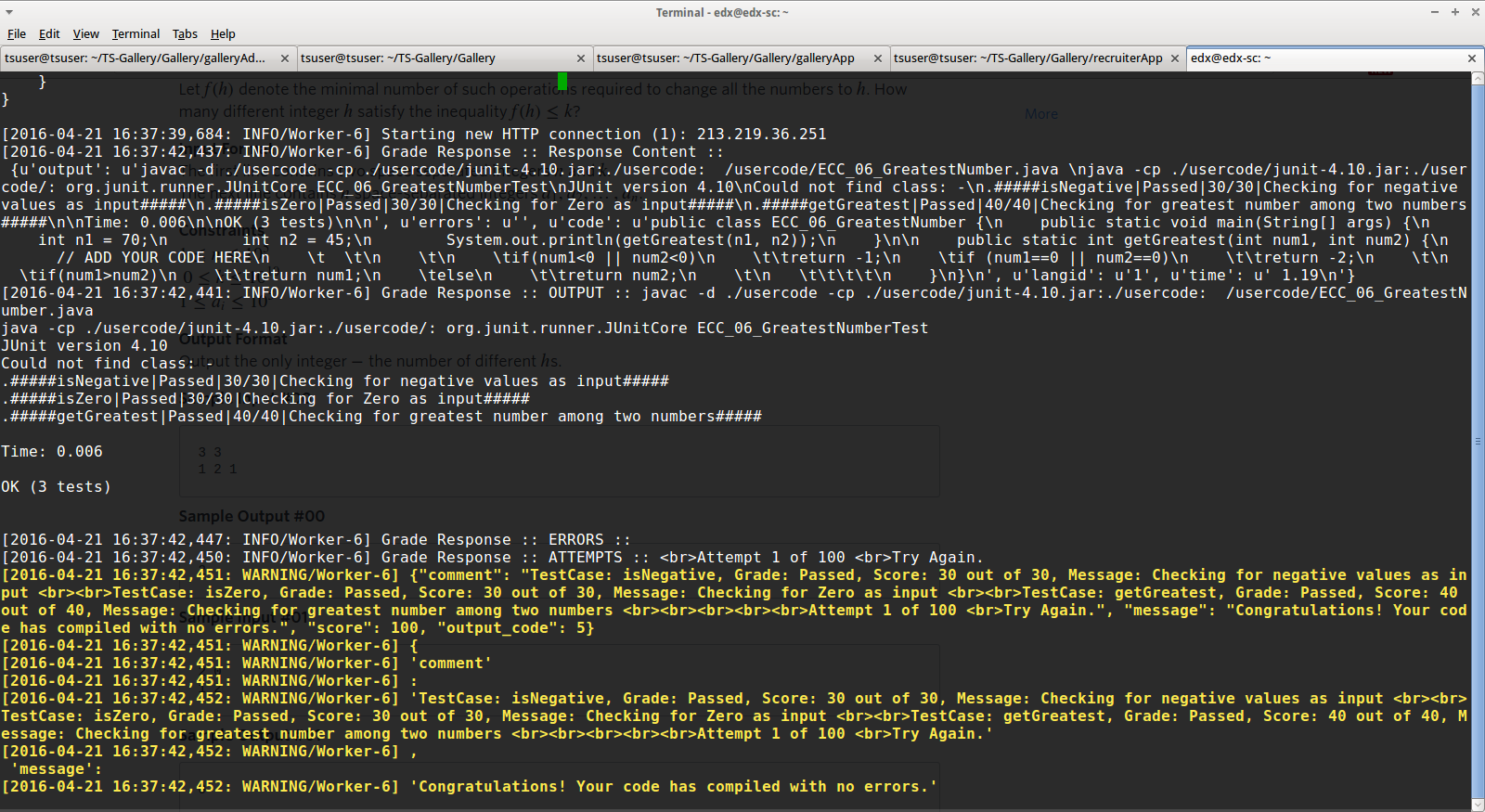


**Fig 5.10- Student code been submitted**

OUTPUT:

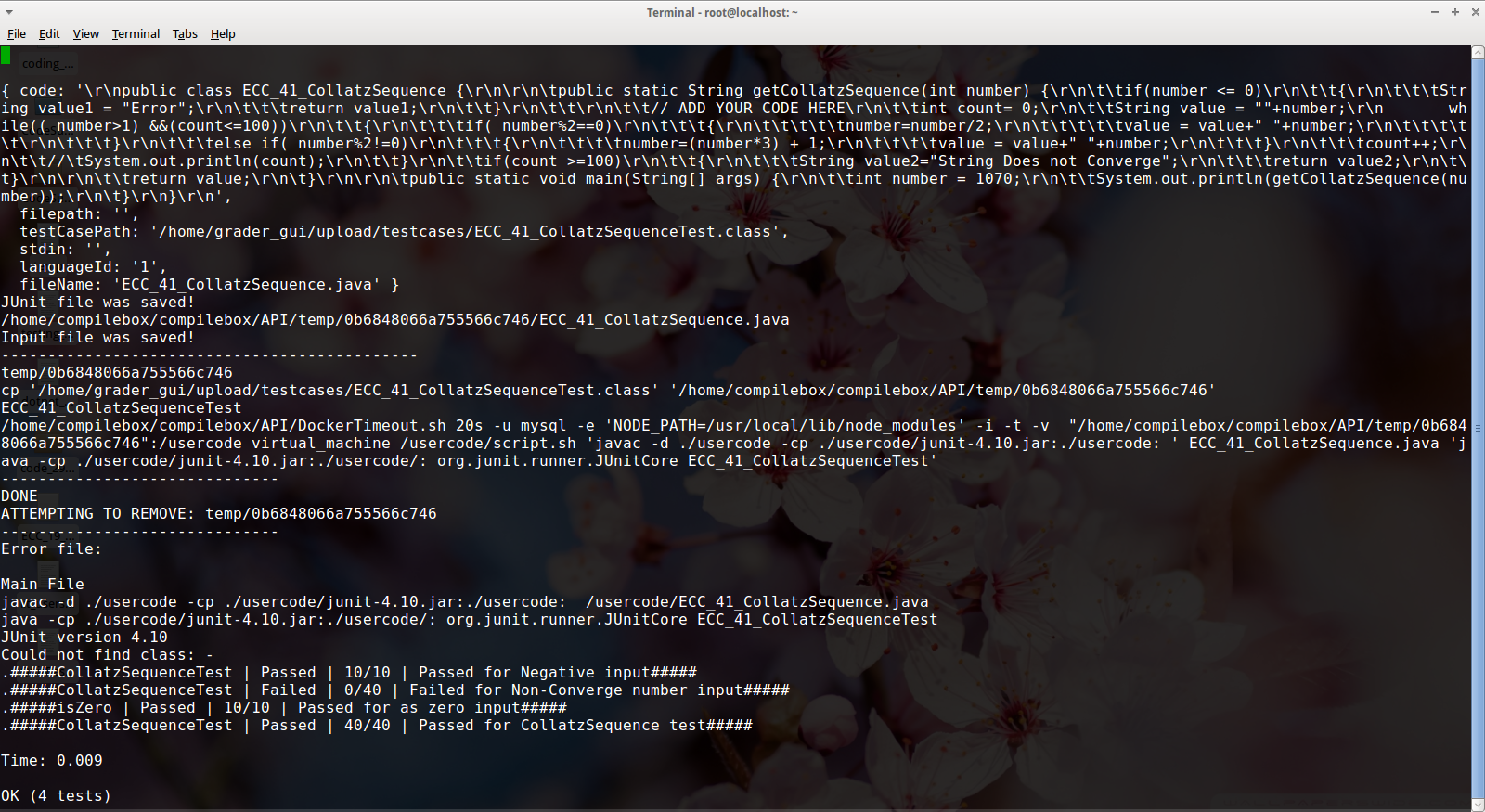


**Fig 5.11- Monitor status**

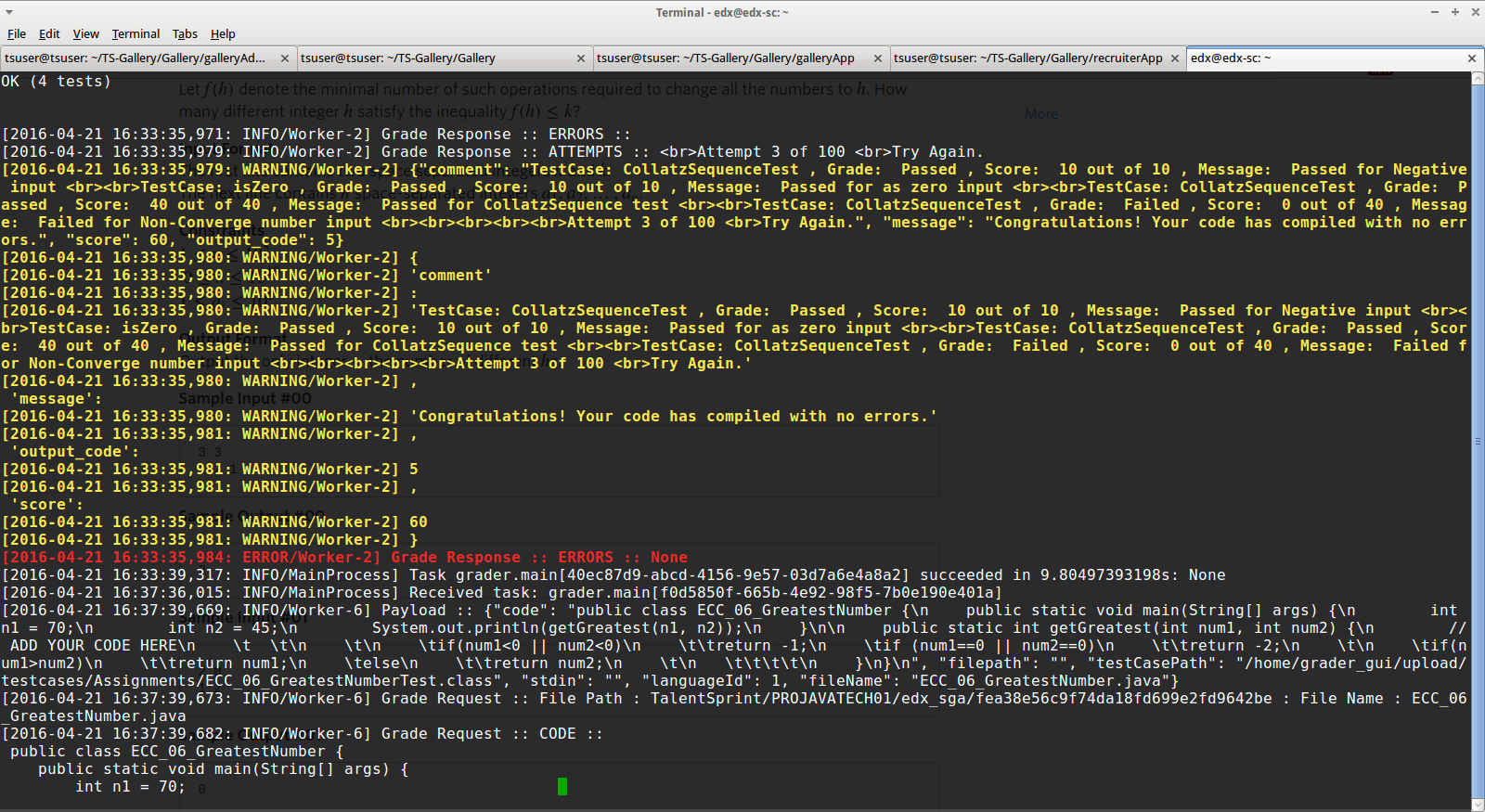


**Fig 5.12- Celery start connection**

**If correct code submitted**

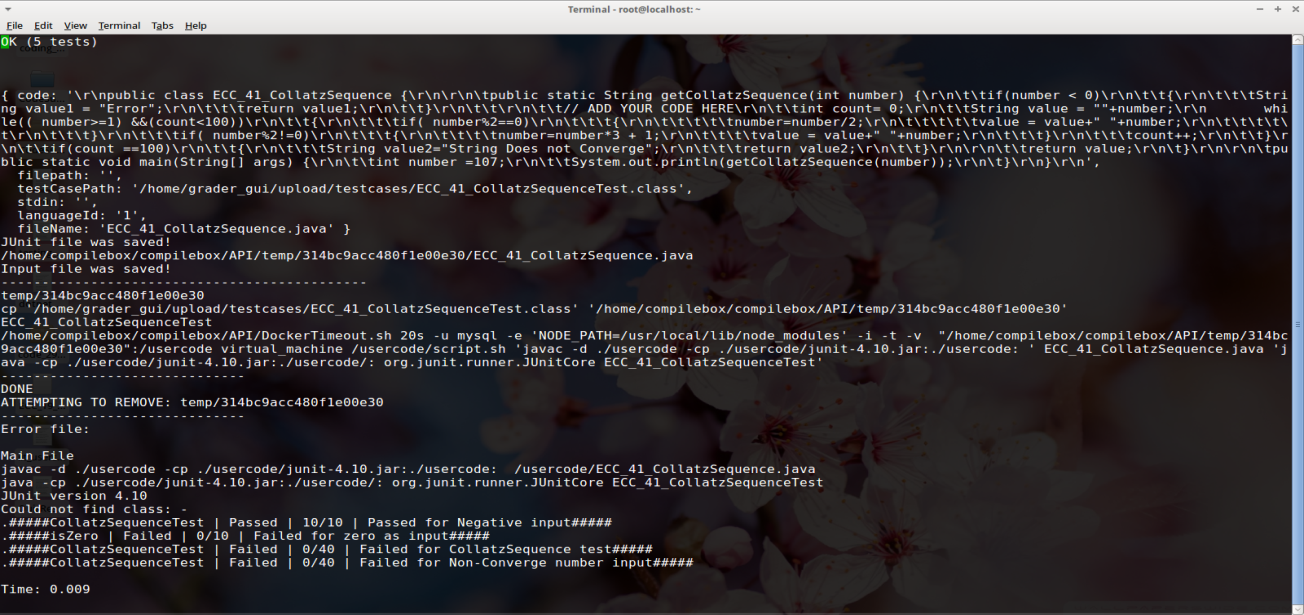


**Fig 5.13- Submitted code correct then Compile box Status**

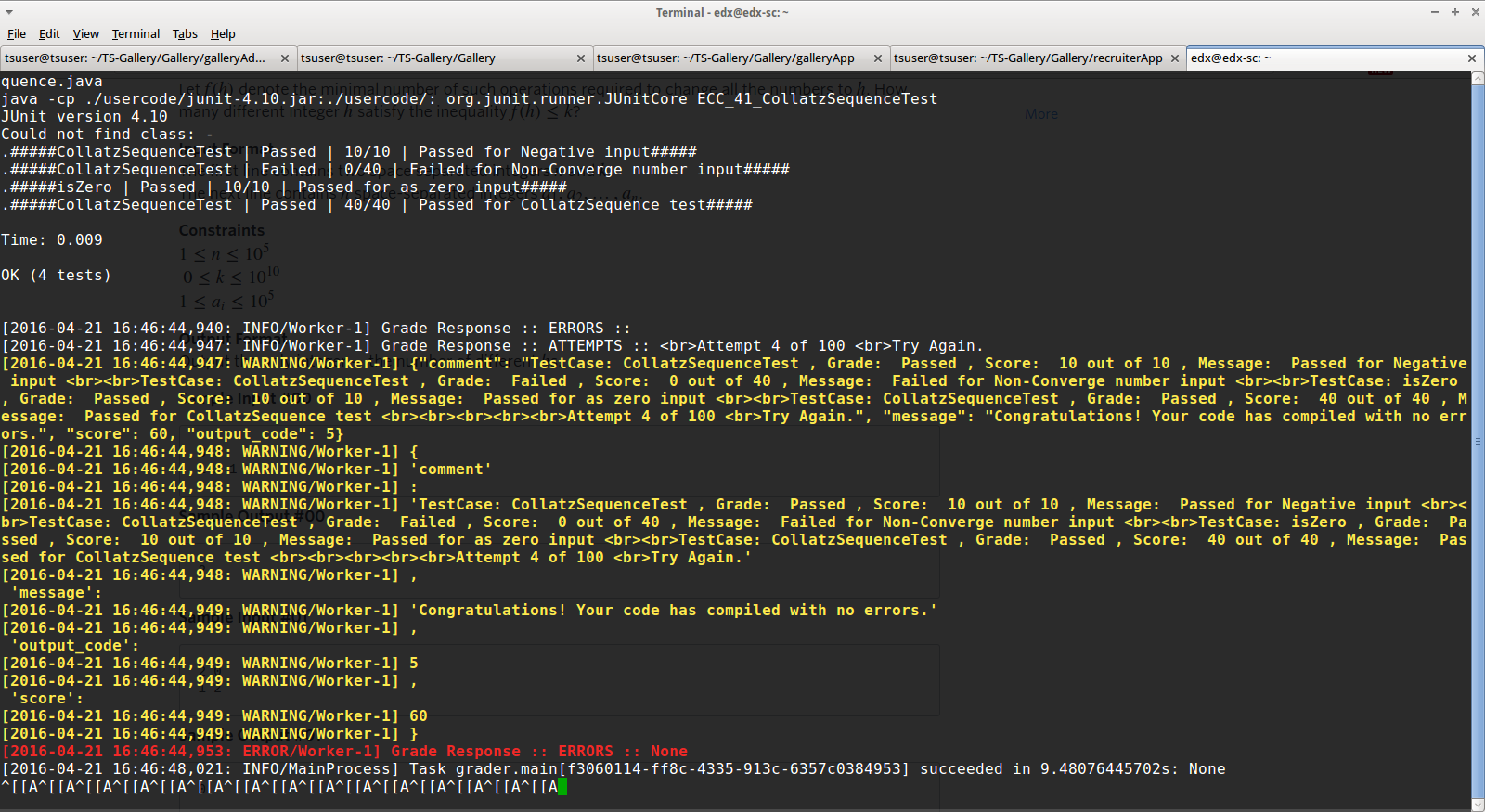


**Fig 5.14- Submitted code correct then Celery Status**

**If wrong code submitted**



**Fig 5.15- Submitted code wrong then Compile box Status**



**Fig 5.16- Submitted code wrong then Celery Status**

1. **CONCLUSION AND FUTURE SCOPE**

The automatic grader is an application to allow talent sprint instructors to grade code submissions of students by running against certain test cases.

This auto grader is currently underutilization in talent sprint.

It can record all submissions stored in a database, making it easier to generate reports and analyze them. And Compilation/execution is done in taking place in sandbox. The database and the compilation/execution box are completely independent of the frontend for submissions, so that new clients can be added with minimal effort. Instructors are also provided with the UI so that they can prepare test question paper and schedule a test. This UI is makes it easy for the instructors to even fetch the results after the exam. And the future extension for the project is that it should be made completely EDX independent platform i.e., working of this project can be done without any dependency on Open edX platform or edX database.

1. **REFERENCES**

* [**https://www.docker.com/**](https://www.docker.com/)
* [**https://nodejs.org/**](https://nodejs.org/)
* [**https://docs.python.org/3/**](https://docs.python.org/3/)
* [**http://www.tutorialspoint.com/javascript/**](http://www.tutorialspoint.com/javascript/)
* [**http://www.celeryproject.org/**](http://www.celeryproject.org/)
* [**https://docs.djangoproject.com/en/1.9/**](https://docs.djangoproject.com/en/1.9/)
* [**https://open.edx.org/**](https://open.edx.org/)
* [**https://github.com/edx/**](https://github.com/edx/)
* [**https://www.mysql.com/**](https://www.mysql.com/)
* [**http://www.tutorialspoint.com/mongodb/**](http://www.tutorialspoint.com/mongodb/)