

Software Project  
Winter Semester 2016-2017

# **MoJEC**

**“Moodle JUnit Exercise Corrector  
(MoJEC)”**

# **Table Of Contents**

## **1. Introduction to MoJEC system**

- a. Goals**
- b. Architectural Overview**
- c. Usage**

## **2. Reasons for used applications**

- a. Docker**
- b. Jenkins**
- c. Maven**

## **3. Installation Guide**

- a. MoJEC Moodle Plugin**
- b. MoJEC Backend Web Service**
- c. License**

## **4. Technical Details**

- a. Frontend**
- b. Backend**
- c. Deployment**

## **5. Demo**

# **Introduction to Moodle JUnit Exercise Corrector (MoJEC)**

MoJEC enables a professor to set up moodle assignments for students, where the students can upload their Java task files in form of a single ZIP file. MoJEC builds the code and runs a predefined JUnit test suite over the code. The professor provides the JUnit tests as well as a single ZIP file when creating the assignment in Moodle. MoJEC shows the JUnit test results to the respective student and the professor similar to other test results in Moodle.

MoJEC verifies the type of the uploaded file. Only zip files can be uploaded to MoJEC. It display the test results of the uploaded Java task files, that means what tests have passed or failed and if there was any compilation error. The MoJEC Moodle plugin depends on the MoJEC web service.

## **Goals**

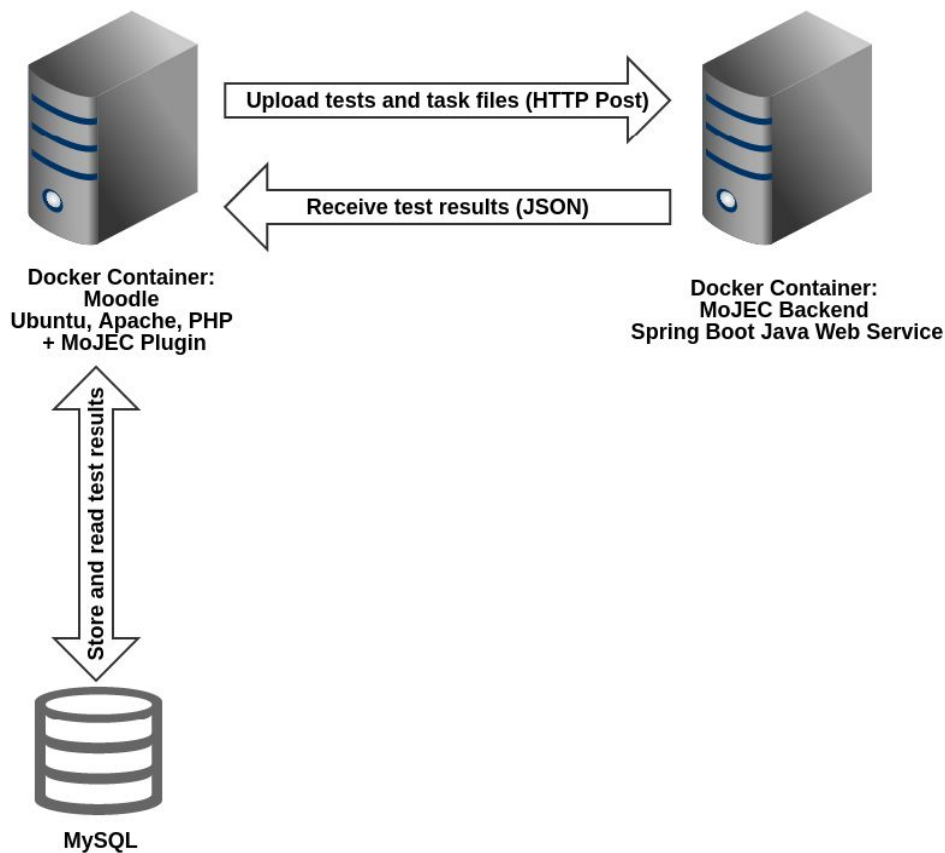
The main aim of this project is to save time and provide less effort for the professors who up until now have to manually download, unzip, compile, and run tests over the Java assignments handed in by students.

## **Architectural Overview**

The MoJEC system consists of two main parts:

- The MoJEC Moodle plugin
- The MoJEC web service

The Moodle plugin is of the type assignment submission and connects to the web service using REST and the JSON file format.



## Usage

### Teacher

- Create an Assignment with the type “JUnit Exercise Corrector”
- Upload a single ZIP file containing your JUnit tests in the corresponding JUnit test file upload environment.  
Note: The JUnit test files cannot be organized into Java packages. The default package needs to be used.
- After the students uploaded their assignments the aggregated test results can be viewed in the grading table column named “JUnit Exercise Corrector”.
- Detailed results of a particular submission can be viewed by clicking the magnifier icon in the respective cell of the JUnit Exercise Corrector column of the grading table.
- Download all MoJEC submissions by selecting the Grading Action Download all submissions.

### Student

- Navigate to the assignment.
- Press Add Submission respectively Edit Submission.

- Upload a single ZIP file containing the Java files to be tested and click Save changes.
- View your test results in the JUnit Exercise Corrector row of the submission status table.

## Reasons for used applications:

### Docker

- Easy multi-server maintainability due to just executing the image on the server.
- Keeping track of the container version, thus providing version control of the containers.
- Lightweight since the containers are operating on the process level, making it perfect for software delivery.
- Very low memory usage, since it only requires an operating system, supporting libraries, and system resources to run a specific program.
- Reliable, since it runs the tests on the same image as the production environment.

### Jenkins

- Open-source
- Very well documented and has a wide range of useful plugins.
- Allows for running builds for multiple branches dynamically.
- Ability to send email notifications.

### Maven

- Maven supports dependency management and will retrieve them transitively
- Maven forces you to have a standard directory structure
- Like ANT it is used to automate the build process. But compared to ANT it is easier to use and comes with a sophisticated plugin ecosystem.
- All Maven projects have a common structure, which makes it easier to understand each project.
- It is declarative. All you have to do was create a .xml file and put your source in the default directory. Maven takes care of the rest.
- It has a lifecycle, which is invoked when you execute `mvn install`. This command tells Maven to execute a sequence of steps until it reaches the lifecycle goal.

# Installation Guide

## MoJEC Moodle Plugin

Download the plugin archive from GitHub

[https://github.com/HFTSoftwareProject/moodle-assignsubmission\\_mojec](https://github.com/HFTSoftwareProject/moodle-assignsubmission_mojec) and extract it to **<pathToMoodle>/mod/assign/submission/mojec** of your local Moodle installation. Then visit the admin's notification page.

## MoJEC Backend Web Service

The web service which is used for this plugin can be downloaded from here:

<https://github.com/HFTSoftwareProject/MoJEC-Backend>

It can be built using below command

```
mvn clean install
```

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

There is also a ready-to-go Docker container available:

<https://hub.docker.com/r/hftstuttgart/mojec-backend/>

This can be run using Docker: `run -p 8080:8080 hftstuttgart/mojec-backend`

## License

GNU General Public License, Version 3, is a free and copyleft license for software. The GNU makes sure it remains free software for all its users, when speaking of free software, it is referred to freedom, not price. General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software, that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, but it is not allowed to relicense the changed source code. If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Developers that use the GNU GPL protect your rights with two steps:

(1) assert copyright on the software

(2) offer you this License giving you legal permission to copy, distribute it.

For more information on this please go to the following link:

<https://www.gnu.org/licenses/gpl-3.0.html>

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt to propagate or modify it is not acceptable, and will automatically terminate your rights under this License.

## Technical Details

- The backend is a RESTful web service that allows you to upload the test files and the task files as HTTP POST parameters.
- The data format for this communication is JSON, i.e when you upload the task zip, the Java files will be checked against the previously uploaded tests and the result will be returned as a JSON string.
- The default timeout set in moodle for the requests to our backend is set at 30sec.
- The backend is based on Spring Boot and uses the JavaCompiler API to compile the tests and tasks.
- The backend uses Maven as a build and dependency management tool.
- For continuous deployment Jenkins 2.0 is used.
- For the virtualization technology we chose to use Docker.
- The continuous deployment environment runs on a KVM machine. It uses Jenkins' Maven plugin, bash scripts and the docker CLI to deploy the new code to the productive environment.
- Jenkins builds the Docker container and pushes it to Docker Hub. On the docker server the newly build container is pulled from Docker Hub.
- Here's an example JSON response after uploading the task Java file.

```

{
  "testResults": [
    {
      "testName": "CalculatorTest",
      "testCount": 5,
      "failureCount": 0,
      "successfulTests": [
        "div",
        "mult",
        "sub",
        "add",
        "sum"
      ],
      "testFailures": []
    },
    {
      "testName": "CalculatorSecondTest",
      "testCount": 5,
      "failureCount": 1,
      "successfulTests": [
        "add2",
        "sub2",
        "div2",
        "sum2"
      ],
      "testFailures": [
        {
          "testHeader": "mult2(CalculatorSecondTest)",
          "message": "expected:<15.0> but was:<10.0>",
        }
      ]
    }
  ]
}

```

The above shows the result of two JUnit test files (CalculatorTest and CalculatorSecondTest). The field "testCount" indicates the number of test methods within the test file. The field "failureCount" indicates how many tests have failed and the field "successfulTests" indicates the method names of passed tests. In case a test failed, the necessary information can be found as an entry in the "testFailures" array.

If there was an compilation error the relevant information is part of the "compilationErrors" array as shown below.



```
"compilationErrors": [  
  {  
    "code": "compiler.err.expected",  
    "columnNumber": 0,  
    "kind": "ERROR",  
    "lineNumber": 0,  
    "message": "',' expected",  
    "position": 46,  
    "filePath": "/tmp/TaskNotCompilable.java",  
    "startPosition": 46,  
    "endPosition": 46  
  }  
]  
}
```

## Interface description

The web service offers three REST endpoints:

### **POST /v1/unittest**

Used for uploading and creating of assignments. The body needs to contain two fields as form data:

*assignmentId*: The ID of the created assignment. This is created by moodle.

*unitTestFile*: The zip file containing the unit tests for this assignment.

### **DELETE /v1/unittest?assignmentId=<111>**

Delete the created assignment. The assignment ID of the unit tests which need to be deleted is passed as a query parameter.

### **POST /v1/task**

The upload of the Java files to be tested. The body needs to contain two form fields:

*taskFile*: The zip file containing the java files.

*assignmentId*: The id of the assignment. Provided by moodle

## Frontend

Moodle originally has been an acronym for Modular Object-Oriented Dynamic Learning Environment. The modularity of Moodle is achieved by having a sophisticated plugin environment, providing vast amounts of existing plugins to choose from – or in case of developing your own plugin – several plugin types to build upon.

This section describes the plugin developed, starting with an overview of the chosen plugin type, then describing the file structure of the plugin and finally its usage.

Part of the following documentation is oriented towards the official Moodle documentation for the Assignment Submission Plugin which can be found here: [https://docs.moodle.org/dev/Assign\\_submission\\_plugins](https://docs.moodle.org/dev/Assign_submission_plugins)

## **Plugin Type – Assignment Submission Plugin**

Moodle provides more than 50 standardized plugin types to choose from when writing a plugin and if none of the standardized types fit there is the “local” type for a generic plugin for local customisation.

For our case the Assignment Submission Plugin type was the best fit. It allows you to display custom form fields to the students when they are editing their assignment submission as well as to the teachers when there are editing the assignment settings. It also has full control over the display of the submitted assignment to graders and students.

In short the main features are:

- An assignment submission plugin can add settings to the module settings page.
- An assignment submission plugin can show a summary of the submission to students and graders.
- An assignment submission plugin can add form fields to the student submission page.

## **File Structure**

All the files of the mojec assignment submission plugin should be located under “mod/assign/submission/mojec” within the root folder of the Moodle installation.

This section briefly describes the files and their purpose within the plugin. Some files will be explained in more detail where applicable. For more information about the files described please have a look at the official Moodle documentation and/or the source code of the plugin:

[https://github.com/HFTSoftwareProject/moodle-assignsubmission\\_mojec](https://github.com/HFTSoftwareProject/moodle-assignsubmission_mojec)

### **version.php**

This file is used to tell Moodle the version information about our plugin, so that it can be installed and upgraded correctly. This information is added to version.php, which is also the case for any other type of Moodle plugin.

For more information please refer to: [version.php](#)

## **settings.php**

This settings file allows us to add custom settings to the system wide configuration page for our plugin.

As is there are two settings described in this file for the mojec plugin:

- default: A checkbox to indicate if the plugin should be enabled by default when creating a new assignment.
- Web service base url: A textfield to define the base url of the web service that is used to communicate the files to and perform the actual tests.

## **lang/<country\_code>/assignsubmission\_mojec.php**

These are the language files for the plugin. Depending on the language to support, the language files reside in a different subfolder of lang. For example:

- English: lang/en/assignsubmission\_mojec.php
- German: lang/de/assignsubmission\_mojec.php
- The filename itself should be the same as the component name of the plugin. The component name of the plugin has the form of <plugintype>\_<pluginname>, so assignsubmission\_mojec.

Such a language file contains several key,value entries in the form of \$string[“key”] = “Value”; where the key is the same throughout the different language files and the value is depending on the given language.

Moodle provides a dedicated String API that allows – given a key – the retrieval of the value depending on the selected language (e.g. get\_string(“key”, “default value”)).

## **db/access.php**

This is where any additional capabilities are defined if required. This file can be omitted if there are no capabilities added by the plugin. The MoJEC plugin does not add any additional capabilities, this file only exists as a placeholder as it seems to be a good practice.

See [Activity\\_modules#access.php](#) for more information.

## **db/upgrade.php**

This file handles upgrading the plugin to match the latest version. If for example a newer version of the plugin requires additional database tables or columns, this is the place to define them.

See [Activity\\_modules#upgrade.php](#) for more information.

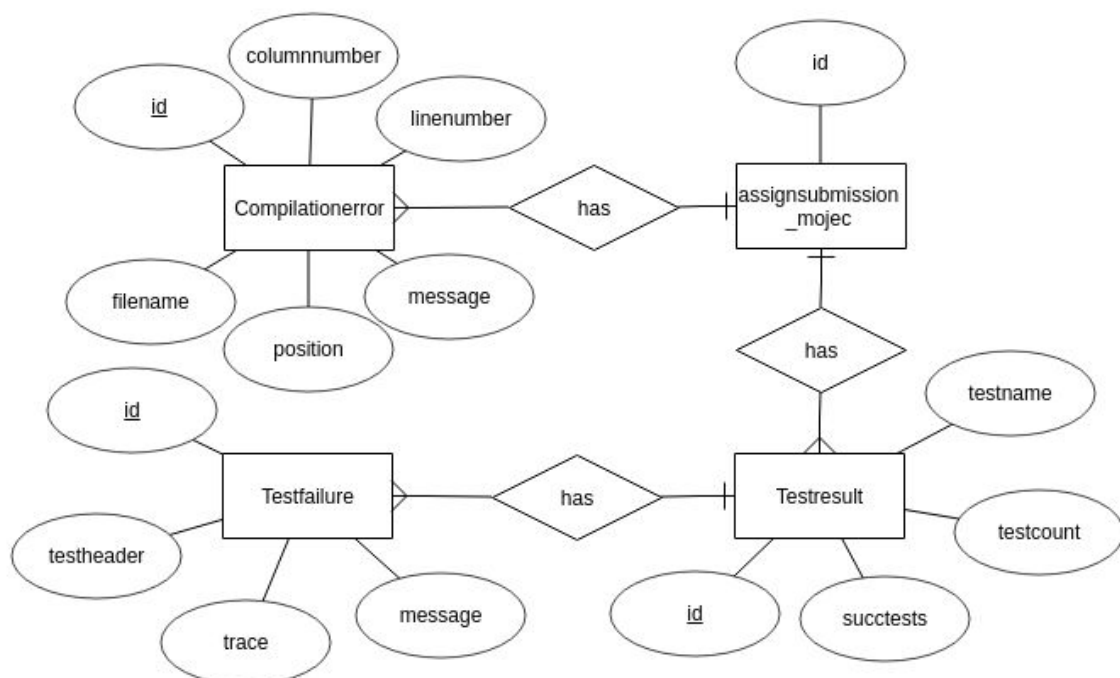
## db/install.xml

This is where any database tables required to save this plugins data are defined. Moodle provides a dedicated XML schema to model this kind of information with elements such as TABLENAME, FIELDS and KEY. In addition Moodle comes with a XMLDB editor that supports the creation of the install.xml without having to directly get in touch with XML.

The code below shows for example how the table “mojec\_testfailure” is defined, with all its fields as well as primary and foreign key definitions.

```
<TABLE NAME="mojec_testfailure" COMMENT="Info about the failures occurred during test
execution.">
  <FIELDS>
    <FIELD NAME="id" TYPE="int" LENGTH="10" NOTNULL="true" SEQUENCE="true"/>
    <FIELD NAME="testresult_id" TYPE="int" LENGTH="10" NOTNULL="true" SEQUENCE="false"/>
    <FIELD NAME="testheader" TYPE="char" LENGTH="255" NOTNULL="true" SEQUENCE="false"/>
    <FIELD NAME="message" TYPE="char" LENGTH="255" NOTNULL="false" SEQUENCE="false"/>
    <FIELD NAME="trace" TYPE="text" NOTNULL="false" SEQUENCE="false"/>
  </FIELDS>
  <KEYS>
    <KEY NAME="primary" TYPE="primary" FIELDS="id"/>
    <KEY NAME="fk_testresult" TYPE="foreign" FIELDS="testresult_id"
    REFTABLE="mojec_testresult" REFFIELDS="id"/>
  </KEYS>
</TABLE>
```

The general structure of data is depicted in the following ER-Model.



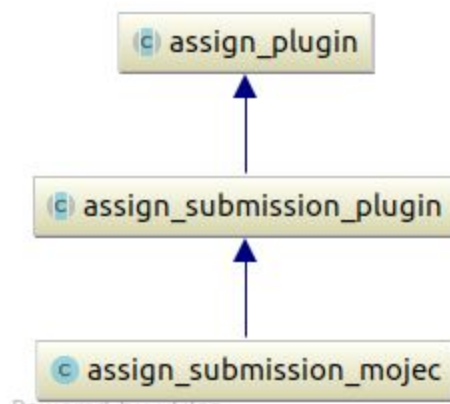
Every time a new MoJEC assignment is created in Moodle, a new instance of an “assignsubmission\_mojec” entity is created. Again, the name of this table has to be in the form of <plugintype>\_<pluginname>. On submission of a task file, the backend web service will be contacted and responds with one or more test results or compilation errors encoded as a JSON-String. The JSON-String will then be parsed and the data is stored in the corresponding database tables for future retrieval.

### locallib.php

This is where all the functionality for this plugin is defined. All submission plugins must define a class with the component name of the plugin that extends *assign\_submission\_plugin*.

```
class assign_submission_mojec extends assign_submission_plugin {
```

That means we have the following class hierarchy:



The *assign\_submission\_plugin* class is an abstract base class all assignment submission plugins must extend. It contains a small number of additional functions that only apply to submission plugins.

The *assign\_plugin* class is an abstract class that is the base class for all assignment plugin (feedback or submission plugins). It provides access to the *assign* which represents the current assignment instance through “\$this->assignment”.

Overall those two classes provide a number of public functions, so called hooks, that can be overridden in order to implement the functionality needed.

In the following a few selected functions will be shortly described to give an impression of the kind of hooks that are present.

- **get\_settings():** The get\_settings function is called when building the settings page for the assignment. It allows this plugin to add a list of settings to the form. In case of the MoJEC plugin a file manager to allow the teachers to upload their JUnit test zip is added. Its overridden from the assign\_plugin class.
- **save\_settings():** The save\_settings function is called when the assignment settings page is submitted, either for a new assignment or when editing an existing one. In the MoJEC plugin this function saves the JUnit test zip selected by the teacher and transfers the file to the backend web service. Its overridden from the assign\_plugin class.
- **get\_form\_elements\_for\_user():** This function is called when building the submission form and allows (like the get\_settings function for the settings) to add a list of elements to the submission form. In case of the MoJEC plugin this function adds file manager to allow the students to upload their task zip file. Its overridden from the assign\_plugin class.
- **save():** This function is called to save a user submission. Within the MoJEC plugin this function does the following things:
  - Save the uploaded task zip file in the Moodle database.
  - Call the backend web service to transfer and the test the file
  - Receive and the process the web service response
  - Save the test results in the Moodle database
 Its overridden from the assign\_plugin class.
- **view\_summary():** This function is called to display a summary of the submission to both markers (teachers) and students. For the students this summary will be shown within the submission status table and for the teachers within a column of the grading table. In the MoJEC plugin this method returns a more compact view (only essential data) for the grading table and a detailed view for the students submission status table.

```
public function view_summary(stdClass $submission, & $showviewlink) {
    global $PAGE;

    if ($PAGE->url->get_param("action") == "grading") {
        return $this->view_grading_summary($submission, $showviewlink);
    } else {
        return $this->view_student_summary($submission);
    }
}
```

- **delete\_instance():** This function is called when the assignment has been deleted and is used for clean-up purposes. For the MoJEC plugin this means all the test result, compilation error records etc. are deleted. In addition the backend web service is called to trigger the deletion of the assignment's test files.

## **lib.php**

This file is the entry point to many standard Moodle APIs for plugins. An example is that in order for a plugin to allow users to download files contained within a filearea belonging to the plugin, they must implement `componentname_pluginfile` function in order to perform their own security checks. In case of the MoJEC plugin this function is named “`assignsubmission_mojec_pluginfile`” and checks for example if the user requesting the file download is actually logged in and has the necessary permissions.

## **Backend**

### **Framework**

This is the web service used for the moodle plugin. It is written in Java and uses the Spring Boot framework. Spring is a largely used Java framework and with the Spring Boot extension it provides a fast gettings started experience for Spring development.

### **Build**

As build and dependency management tool, Apache Maven is used. The application can be build using mvn package.

### **Application configuration**

The backend web service is using the `application.properties` file to configure our application.

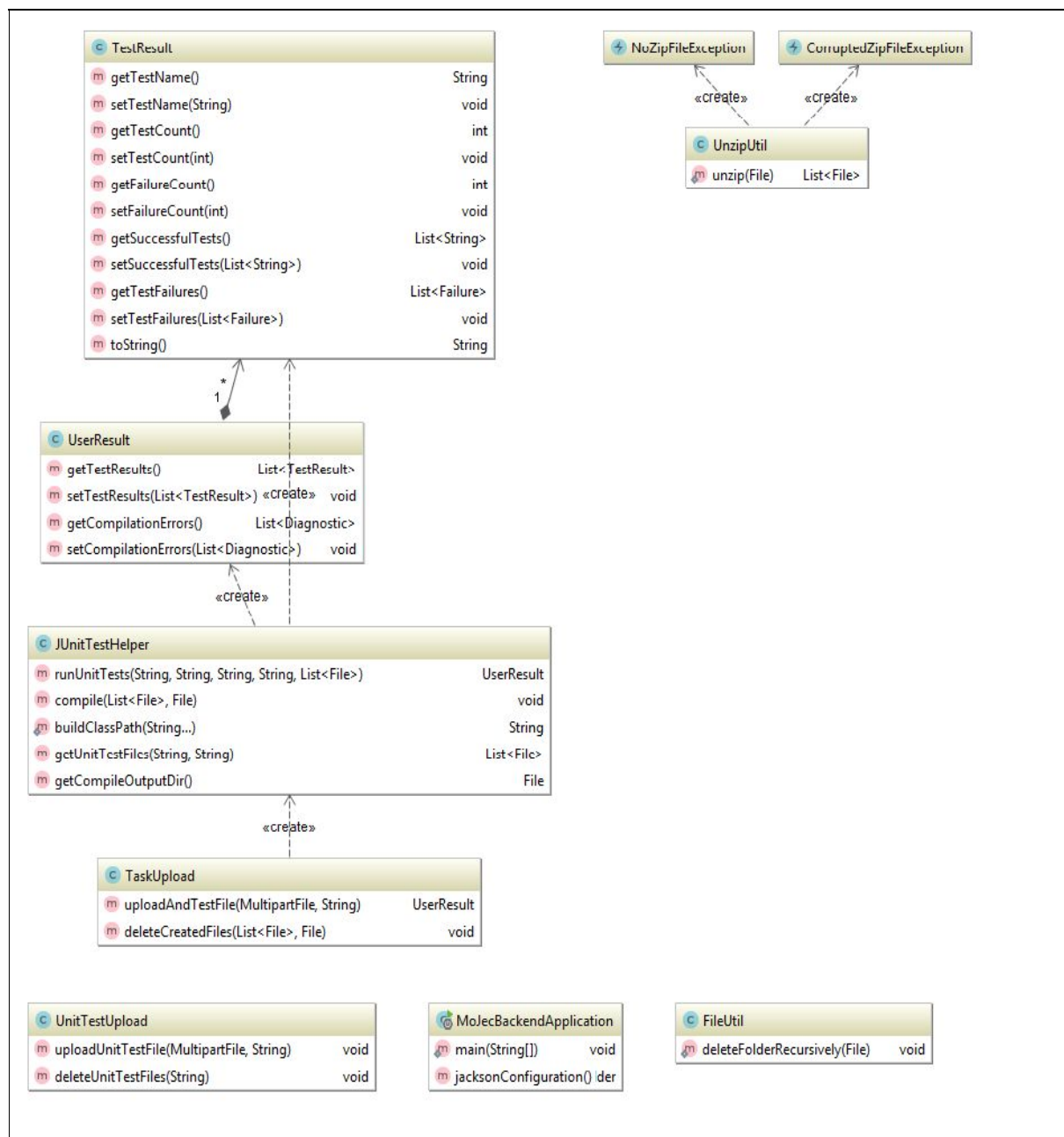
To configure your local configuration create a file called `application-local.properties` in `/src/main/resources/` and override the properties. Afterwards configure the application to use the local profile using the run configuration or adding `spring.profiles.active=local` to the global `application.properties` file.

### **Integration tests**

MoJEC-Backend has some rudimentary API tests using Spring Boot Testing. This tests assure that there won't be any regressions in the API when changing the backend code.

To be able to run the integration tests the system where the tests are executed needs to be a \*nix System because a /tmp/ folder must exist. Also the needed libraries JUnit and Hamcrest need to be downloaded into /opt/mojec/junit/. This is the reason why the tests are disabled by default so it can be build on MS Windows systems. The tests can be enabled by setting -DskipTests=false

## Overview of Mojec Class UML

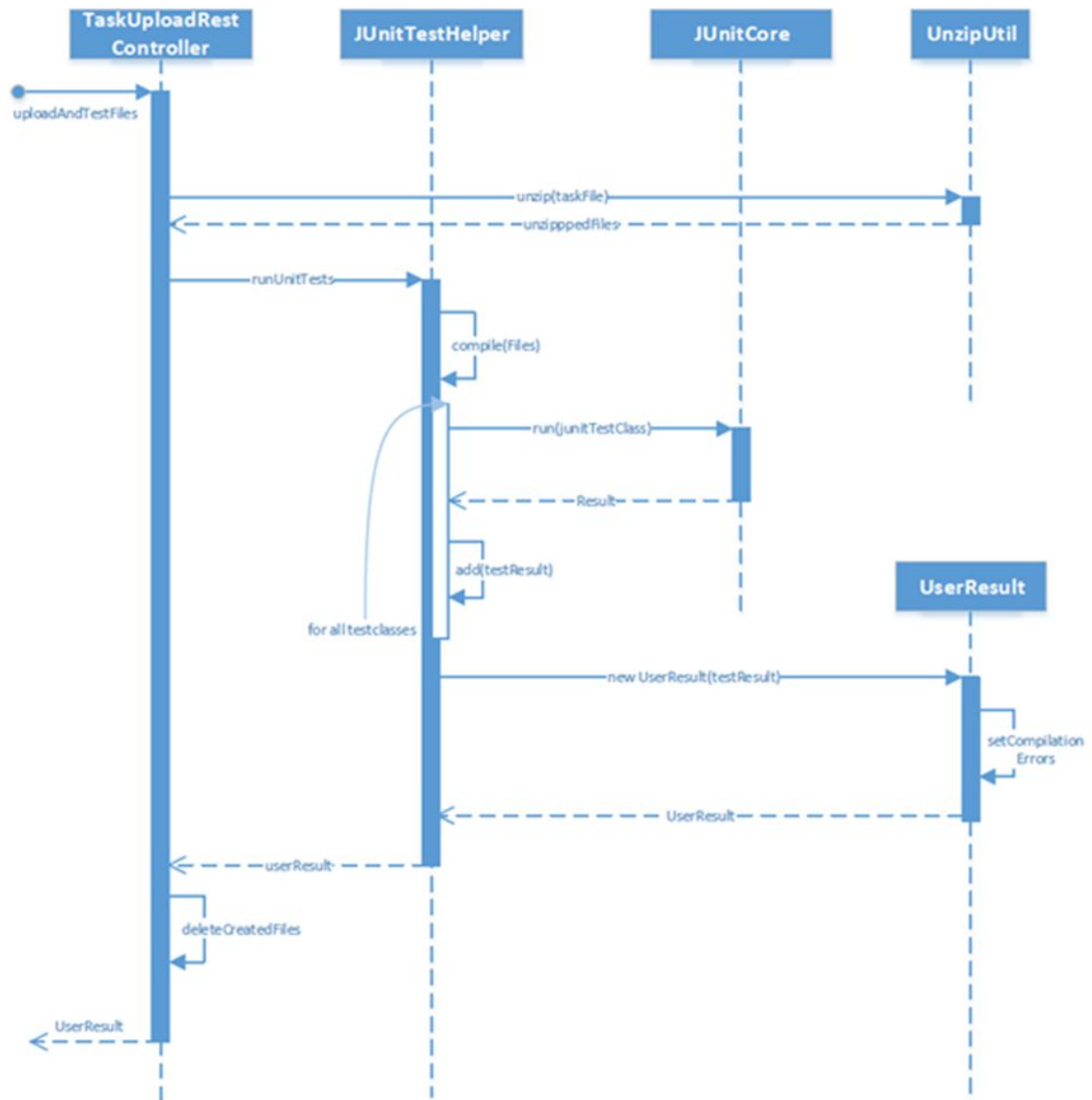




## List of important Classes & Functions

1. UnzipUtil:
  - a. unzip(file): Utility function to unzip the uploaded files and saves them to disk. Also checks if the zip file is valid.
2. JUnitTestHelper:
  - a. runUnitTests: creates a temporary folder for the compilation output, loads compiled classes into the classloader and runs JUnit tests. Returns a list of all successful, failed and not compilable tests.
  - b. compile: Sets the compiler option for a specific output path, compiles it, and if the compilation fails, tries to compile again without the not compilable file.
  - c. buildClassPath: Builds a custom class path. This is needed because the JUnit.jar dependency needs to be in the classpath when compiling the JUnit tests.
  - d. getUnitTestFiles: Gets all the JUnit Test files from the specified path.
3. UnitTestUpload
  - a. uploadUnitTestFile: REST resource for the JUnit test upload: Creates one folder per assignment and unzips the JUnit files into this subfolder.
  - b. deleteUnitTestFiles: REST resource for delete uploaded JUnit tests.
4. TaskUpload:
  - a. uploadAndTestFile: REST resource for the upload of the Java files. Unzips the files into the subfolder and runs the tests. Afterwards it creates / formats the result for the frontend.
5. MyRunListener:
  - a. testFailure: Method which is called when a Unit test fails. In here we save a list of all failed tests.
  - b. getSuccessfulTestNames: Creates a list of all successful tests.

## Sequence Diagram Task Upload Use Case:



# Deployment

## Docker installation:

- Installing Docker Toolbox adds these software:
  - For Windows:
    - Docker Client for Windows
    - Docker Machine for Windows
    - Docker Compose for Windows
    - VirtualBox
    - Kitematic for Windows (Alpha)
    - Git for Windows
  - For Linux:
    - `sudo apt install docker.io`
- Installing Docker Toolbox provides a Docker Terminal for running all Dockerfiles & images
- Then comes creating Dockerfiles as per the requirements
- The Dockerfile runs using Docker Terminal

## Dockerfile for MoJEC:

- Two Dockerfiles were created. One for Moodle including the MoJEC plugin and other one for the MoJEC-Backend
- The MoJEC Frontend Docker file installs and runs Moodle 3.1 stable, with an external MySQL Database. (This includes latest Ubuntu 16.04, PHP7 and Moodle 3.1)
  - Click [here](#) for the Moodle docker image
- MoJEC-Backend Docker file includes downloading libraries needed for compilation
  - Click [here](#) for the backend docker image

## Continuous Deployment environment:

The software installation required on KVM server:

- JDK: Execute following commands on the terminal for installation
  - `sudo apt-get install default-jdk`
  - `sudo apt-get install default-jre`
- Maven: Execute below command on the terminal to install Maven
  - `sudo apt-get install maven`
- Jenkins: Execute below command on the terminal to get jenkins
  - `wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war`
  - save it in accessible folder, go inside that folder using `cd`

- Execute below command on the terminal to install Jenkins  
`java -jar jenkins.war`
- To push the latest image on the Docker machine. To pull the latest changes for the plugin to update the MOJEC Front-End plugin.

### Docker Toolbox installation:

Click the below link to install the Docker tool box on windows machine

[https://docs.docker.com/toolbox/toolbox\\_install\\_windows/](https://docs.docker.com/toolbox/toolbox_install_windows/)

### MoJEC - Docker Hub:

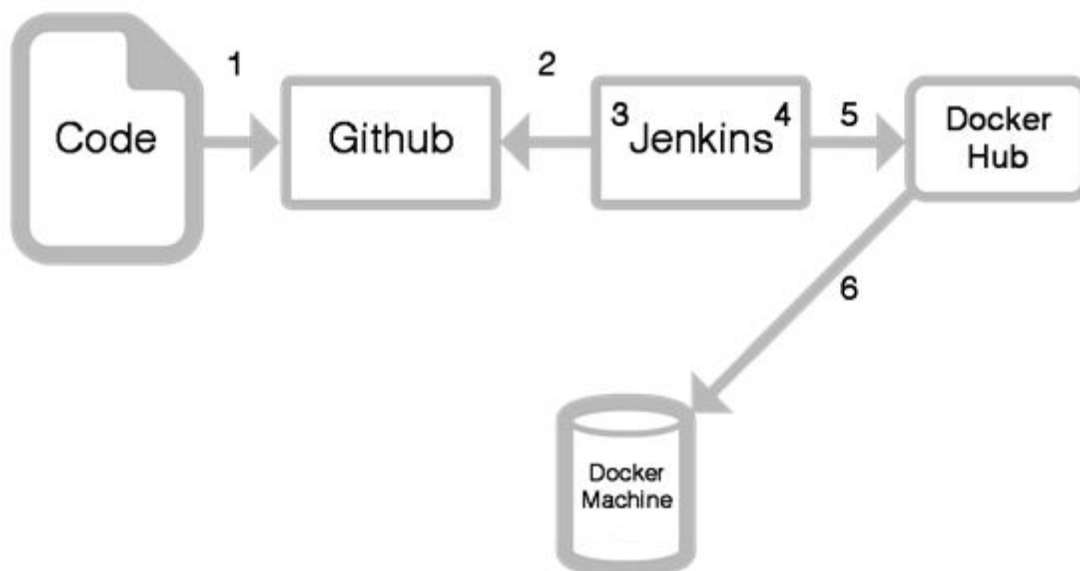
<https://hub.docker.com/r/hftstuttgart/moodle/https://hub.docker.com/r/hftstuttgart/mojec-backend>

### Jenkins installation:

Click the below link to get the Jenkins war file for installation

<https://jenkins.io/>

### Continuous deployment workflow



1. A developer makes any changes to the code and pushes the new changes to GitHub repository
2. These changes to the GitHub would be detected by Jenkins.(through Build Trigger configuration in Jenkins)
3. Jenkins will then take the new code or changes from the GitHub and compile it (as mentioned in Build configuration in Jenkins)
4. Further Jenkins will Build the Docker container(as mentioned in Build configuration in Jenkins)
5. And then Jenkins will push this Docker container to Docker Hub(as mentioned in Build configuration in Jenkins using Docker Build and Publish plugin)
6. Finally Jenkins will run a shell script which will pull and start new Docker container on Docker server(done via Shell Execution command done in Jenkins)

## **Information on SSH Communication**

To pull the latest changes of the Moodle plugin we created a shell script that will be used to execute a git pull and get the latest changes for the MoJEC Plugin.

This is done via Shell Execution command in Jenkins

This will connect to the Docker machine via ssh and executes a script (which exists in the Jenkins machine)

To connect to the Docker Machine via ssh, username and password entry are required at each connection, which made a problem in the automation process. To bypass this problem the authentication using keys is used.

### **SSH Login without username and password**

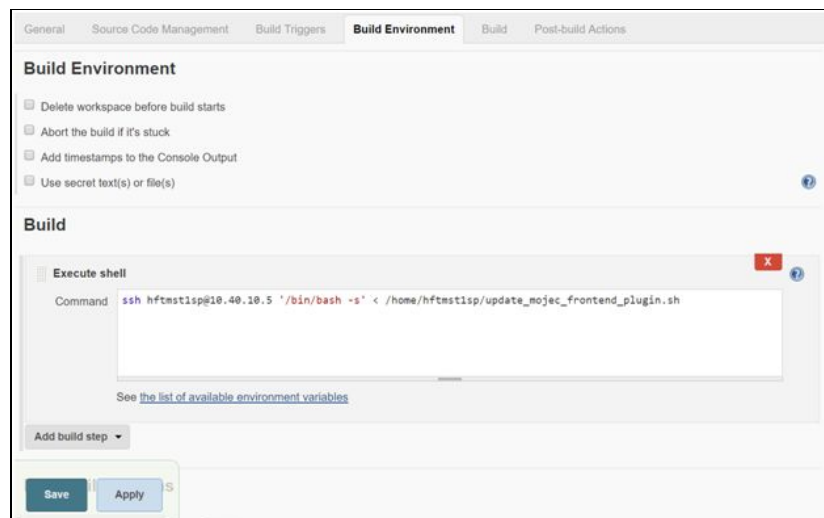
To be able to login from Jenkins to the Docker Machine using ssh the following steps should be done:

1. Run terminal on the Jenkins machine.
2. Execute “ssh-keygen” to generate the public and private key of the Jenkins Machine.
3. Execute “ssh-copy-id -i ~/.ssh/id\_rsa.pub 10.40.10.5” (10.40.10.5 Docker Machine IP) which appends the public key to the authorized\_keys file in the Docker Machine.
4. Enter the Docker Machine Password.

After following the above steps, all ssh connections from the Jenkins Machine to the Docker Machine will run without any other request for username or password.

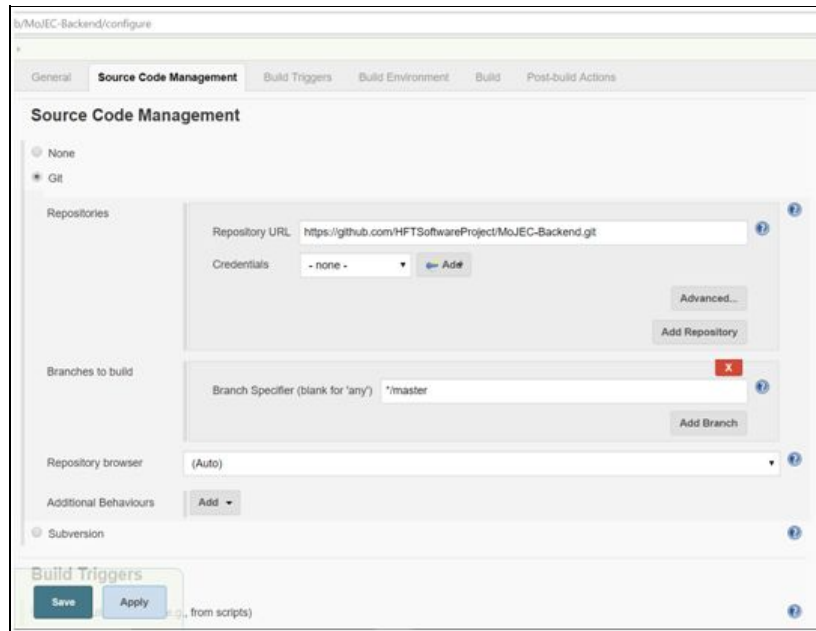
## Jenkins Job: Settings for MoJEC-Frontend:

1. Create new item with Free style project. **MoJEC-Frontend**
2. Once the New Item is created go to the Configuration to apply settings as per requirement, you can provide Project Name & Description
3. **Source Code Management:** In source code management we will set the URL of the Git repository from where the code is to be referred
4. **Build Triggers:** Here we specify how often the Jenkins should trigger the Build. In the configuration settings, we have selected Poll SCM and the Schedule as H/2 \* \* \* \* that means it will trigger Build every 2 minutes
5. **Build Environment:** Here settings related to what exactly has to be done is entered. In the below screen-shot: Execute Shell: Connects to the Docker machine via ssh and executes a script (which exists in the Jenkins machine) to execute a git pull and get the latest changes for the MoJEC Plugin.



## Jenkins Job: Settings to Automate MoJEC-Backend:

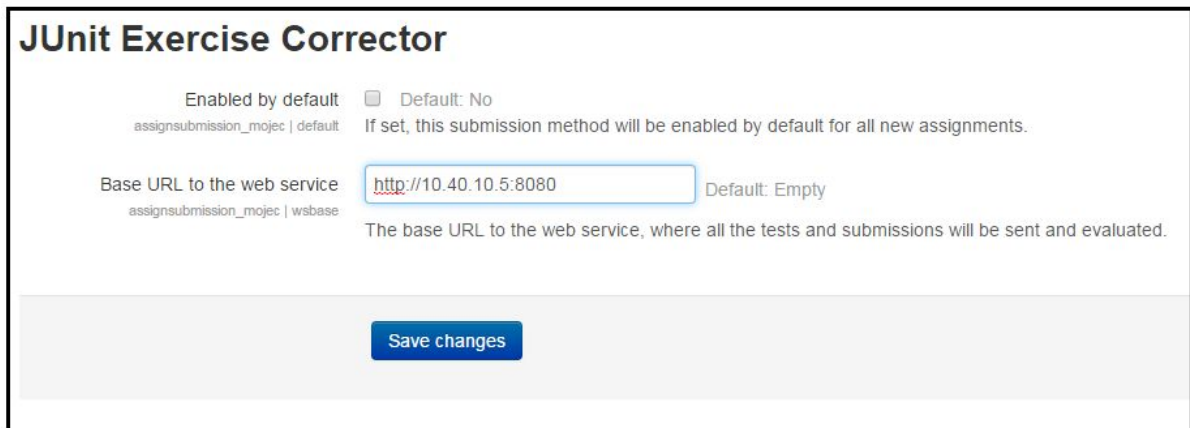
1. Create a new item with Free style project. **MoJEC-Backend**.
2. Once the New Item is created go to the Configuration to apply settings as per requirement, you can provide Project Name & Description.
3. **Source Code Management:** Here we will set the URL of the Git repository from where the code is to be referred.



4. **Build Triggers:** Here we specify how often the Jenkins should trigger the Build. In the configuration settings, we have selected Poll SCM and the Schedule as H/2 \* \* \* \* that means it will trigger Build every 2 minutes.
5. **Build Environment:** Here settings related to what exactly has to be done is entered In the configuration settings: Clean package using Maven plugin: After the latest changes are pulled from the GitHub, this will remove and again package all. Docker Build & Publish Plugin: To compile, build the new image and push the new image to Docker Hub. Execute Shell: Connects to the Docker machine via ssh and executes a script (which exists in the Jenkins machine) to pull and run the new image.  
The plugin used to perform docker operation was 'Cloudbees Docker Build Push & Plugin'

# Demo

The backend webservice has to be configured for the backend. This can be done in moodle: Site administration -> Plugins -> Plugins Overview -> Find "JUnit Exercise Corrector". Go to the settings of the plugin and configure the backend URL as shown below.



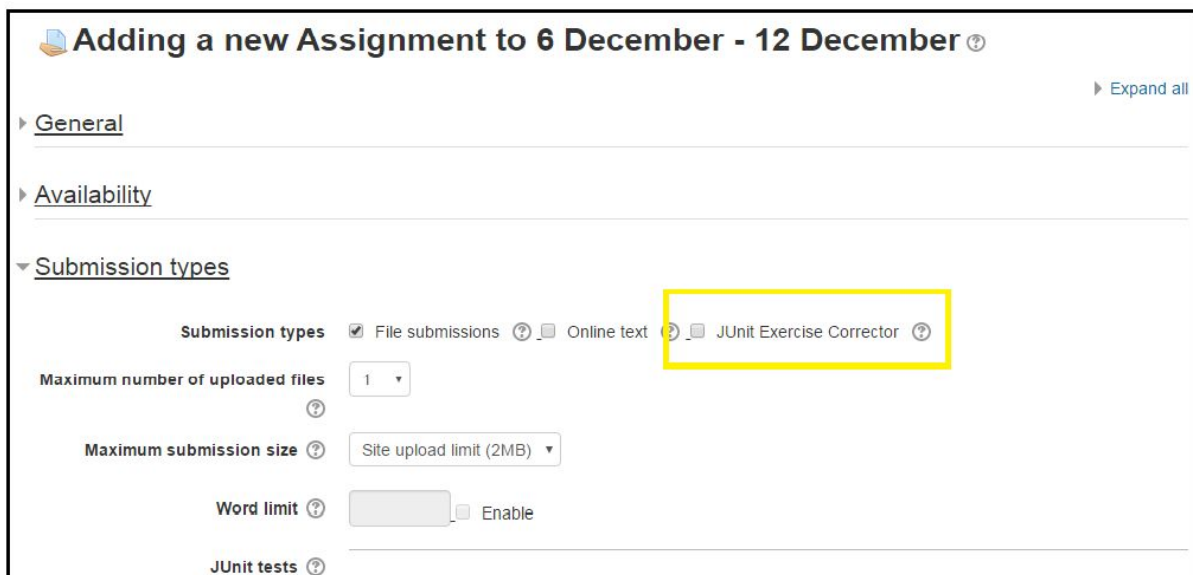
**JUnit Exercise Corrector**

Enabled by default ☐ Default: No  
assignsubmission\_mojec | default If set, this submission method will be enabled by default for all new assignments.

Base URL to the web service  Default: Empty  
assignsubmission\_mojec | wsbase The base URL to the web service, where all the tests and submissions will be sent and evaluated.

[Save changes](#)

When adding new assignment, the JUnit Exercise Corrector is the type of submission to be selected to submit and run JUnit programs.



**Adding a new Assignment to 6 December - 12 December** [Expand all](#)

▸ General

▸ Availability

▼ Submission types

**Submission types** ☒ File submissions [?](#) ☐ Online text [?](#) ☒ JUnit Exercise Corrector [?](#)

**Maximum number of uploaded files** [?](#)

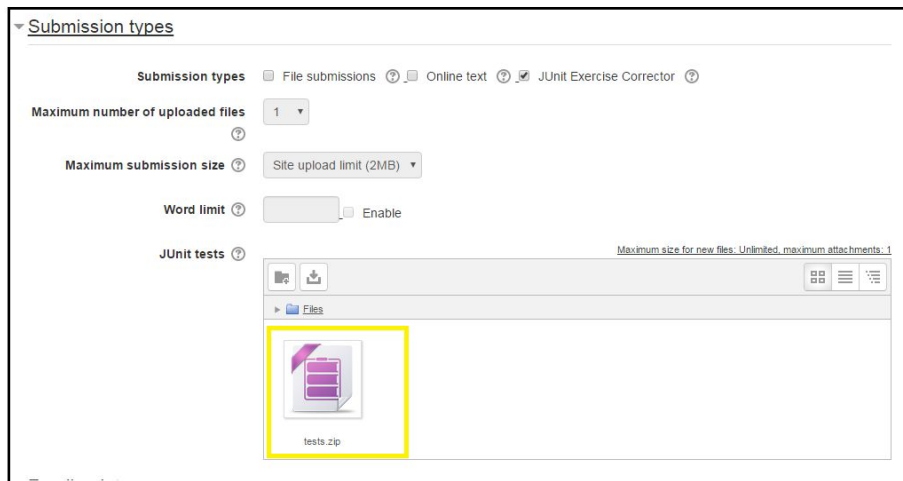
**Maximum submission size** [?](#)

**Word limit** [?](#)  ☐ Enable

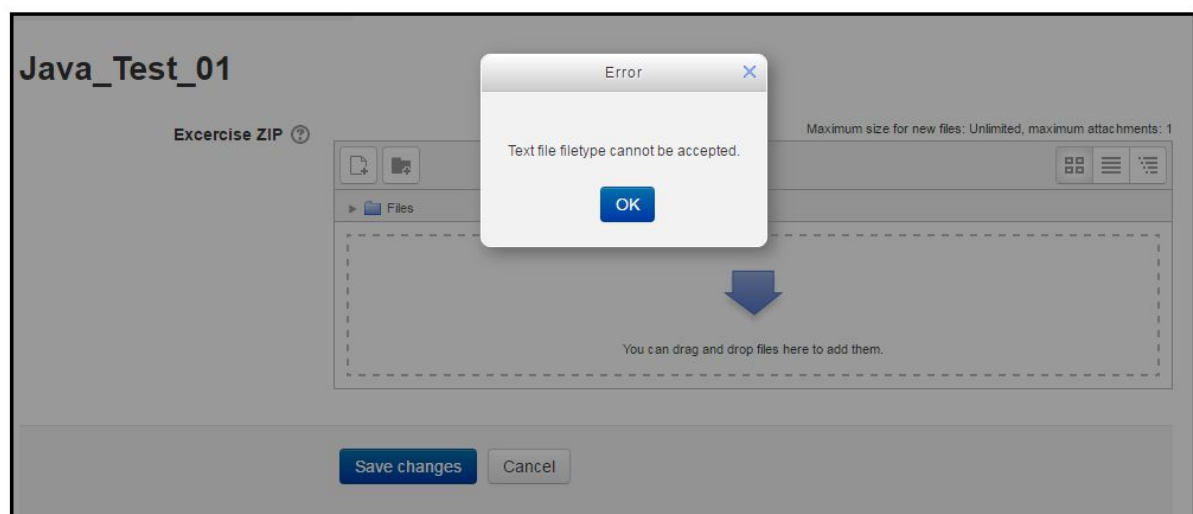
**JUnit tests** [?](#)



After the selection of the submission types, the zipped program should be submitted to run the test and get back the weighted results.



Make sure the submission of the file should be a zipped format, the Exercise Corrector does not accept normal files and throws back Error message to the user.



After successfully adding the zip file, the JUnit corrector processes, run and compile the zip file. This will then calculate the number of successful tests and the compilation error, then submitted for grading.

**CalculatorTest**  
**Successful Tests**

- add
- div
- sub
- sum
- mult

**Compilation errors**

Filename: TaskNotCompilable.java  
Message: ';' expected  
Column-No.: 0  
Line-No.: 0  
Position: 46

Submission comments
▶ Comments (0)

Edit submission

Make changes to your submission

The overall results are presented so the professor grades the students exercise.

Submission status	Submitted for grading
Grading status	Not graded
Due date	Monday, 26 December 2016, 12:00 AM
Time remaining	6 days 10 hours
Last modified	Monday, 19 December 2016, 1:39 PM
JUnit Exercise Corrector	<div>  tasks.zip </div> <b>Overall results</b> Comp. Err.: 1 Tests: 9/10 (90%) <b>CalculatorSecondTest</b> <b>Successful Tests</b> <ul style="list-style-type: none"> <li>• add2</li> <li>• div2</li> <li>• sub2</li> <li>• sum2</li> </ul> <b>Failed Tests</b> <p> Testheader: mult2(CalculatorSecondTest)  Message: expected:&lt;15.0&gt; but was:&lt;10.0&gt;  Trace: <a href="#">show trace</a> </p>