OpenSubmit

Release 0.7.8

May 23, 2018

Contents

Warning: The manuals are work in progress and therefore incomplete. Feel free to help us with a pull request on GitHub.

CHAPTER 1

Students Manual

Welcome!

Your course responsible decided that it might be a good idea to manage the course assignments in OpenSubmit. This manual gives you a short introduction into the system, although most parts should be self-explanatory.

The most common use case for OpenSubmit are programming assignments, where your submitted solution is compiled and initially validated on dedicated test machines. In the further manual, we refer to this feature as *validation*.

The final grading of your solution is always done by humans. Validation just helps you and the correctors to have a working and gradable solution being submitted before the deadline.

1.1 Login

The URL of the OpenSubmit installation was announced by your course responsible. Depending on the configuration, you see one or more options for login. After that, you see the student dashboard.

Please note first the menu at the top of the window. It allows you to:

- See a list of withdrawn solutions in the archive.
- Activate the courses you are interested in.
- Adjust your user settings.

1.1.1 Setting your user details

Depending on the login method, some of your user details might be initially missing. This relates especially to your student ID and the study program you are in.

Since you are (hopefully) interested in not getting the grade of another person, it is highly recommended to fill out the missing fields after the first login. Click on *Settings* in the upper right corner to fix your user details.

The eMail address has a special relevance. When your submitted results are validated or graded, you are informed by eMail about the results. When your OpenSubmit installation offers multiple login methods, it is also used the match accounts to each other.

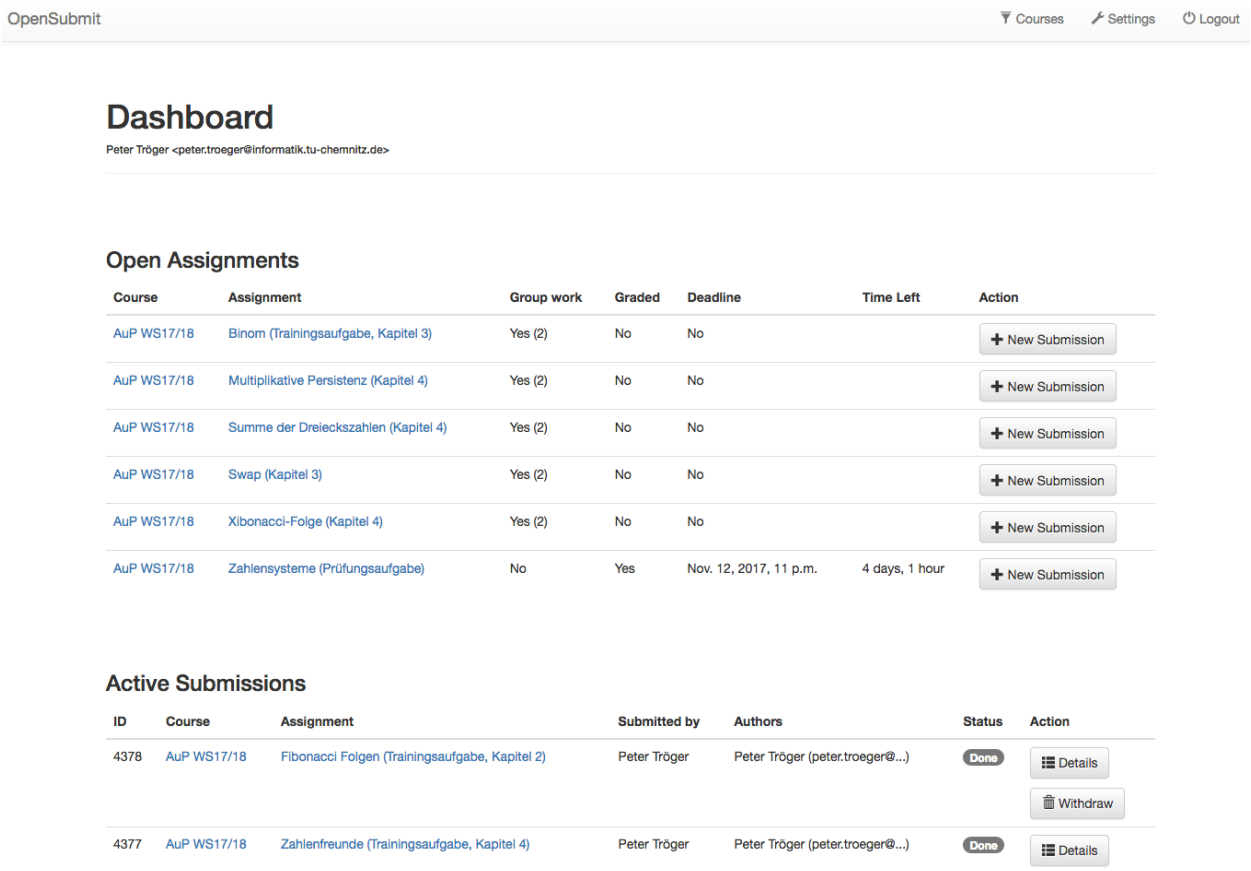
1.1.2 Choosing your courses

After your first login into the system, you might not see any open assignments, although they are already published. This is reasoned by the fact that you need to *activate* the according courses in OpenSubmit. Click on *Courses* in the upper right corner of the dashboard and choose them accordingly.

Activating and deactivating courses does not impact the status of your past and current submissions in the system.

Courses and the related assignments may disappear from your dashboard when the course owner disables them, f.e. at the end of the semester.

1.2 Dashboard



The dashboard is your central starting point for working with assignments in OpenSubmit. It is divided into three major sections:

Open Assignments that you can submit solutions for.

In progress Solutions that are currently validated or graded.

Finished Past solutions and assignments.

1.2.1 Open assignments

The list of open assignments shows all relevant information that you need:

Course A link to more information about the course were this assignment is offered.

Assignment A link to the assignment description.

Deadline The deadline for this assignment. When the deadline has passed, you can no longer submit a new solution.

Group Work The information if the submission of a single solution as group work is allowed, with the maximum number of authors in brackets. One student of the group is submitting the solution and specifies the other group members. OpenSubmit allows you to change your student group for every assignment, although this might not be allowed in your course. Check the assignment description. All group members have the same rights for the submission: Seing the status, getting notification mails, and withdrawing it before the deadline.

Graded The information if this is a graded assignment.

The *New Submission* button brings you to a separate screen where you can upload your assignment solution, either for yourself or your group of students.

The *Notes* fields allows you to drop additional information that is shown to the correctors.

Some assignments may expected the upload of a file. Please check your assignment description for the specific rules. Normally, this should be either a single file (source code, PDF file, DOCX file, image, ...) or a ZIP / TGZ archive of multiple files.

1.2.2 In Progress

Every uploaded file or archive is shown in the list of submissions that are *in progress*. Their assignments are then no longer shown in the list of open assignments.

The state of a submission in progress may be:

Waiting for grading Your solution is waiting for being graded by a human.

Waiting for validation test Your solution is queued for automated validation on a test machine.

Validation failed The automated validation failed. You need to act before the deadline.

You get an eMail when the state of your active submission changes, so you don't need to check the web pages all the time.

The *Details* button brings you to a separate page with all the neccessary information about your submission. It includes all information visible for the correctors (notes, uploaded file, declared authors), the results of the automated validation on the test machine, and eventually your final grade for this assignment.

When the assignment has a deadline, you are free to withdraw your submitted solution *as often as you want* before the deadline. The idea here is that you can use the automated validation in a trial-and-error fashion.

Withdrawn submissions are not considered for grading. They are still listed on the *Archive* page so that you can access your earlier attempts.

1.2.3 Finished

This section shows you finished work that no longer needs your active participation. This includes:

- Submission that were successfully validated and graded.
- Submissions for non-graded assignments that were successfully validated.
- Assignments for which you never submitted a valid solution.

1.2.4 Test Machines

The validation of student submissions is performed on dedicated test machines. For programming assignments, it is often needed to get specific technical details about the target machine. This information is summarized in the *Test Machines* section on the Dashboard.

CHAPTER 2

Teacher Manual

OpenSubmit was invented for making assignments more fun for the students, and less work for the teachers. Before you start to read into details, we recommend to get into the general *principles* we follow.

Student tutors, course owners and administrators (see also *Permissions*) all operate in the teacher backend, which is reachable by a link at the bottom of the student dashboard page, or directly via <*your OpenSubmit url*>*/teacher*.

2.1 Managing study programs

Location: `Teacher backend - System - Actions - Manage study programs`

This function is only available for users with according *permissions*.

Students register in OpenSubmit by themselves, simply by using one of the configured authentication methods (see *Authentication methods*). After the first login, there are asked to complete their user details (see also *Setting your user details*).

One part of the user details dialogue is the choice of the study program, e.g. *computer science (Bachelor)* or *Greek philosophy (Master)*. When more than one study program is configured in OpenSubmit, then this choice becomes mandatory. If only a single or no study program is configured, then the students are not forced to make that choice.

The study program is shown in the *User management* and the *Grading table*. Is has no further impact on the operation of OpenSubmit, but can help with the grading of mixed courses.

2.2 Managing courses

Location: `Teacher backend - System - Actions - Manage courses`

This function is only available for users with according *permissions*.

Assignments for students belong to a course. The registered students can choose (see also *Choosing your courses*) which course they participate in. This is different to many other learning management systems, which offer dedicated course permission systems (see also *The Zen of OpenSubmit*).

2.2.1 Course creation

Location: `Teacher backend - System - Actions - Manage courses - Add course`

The following settings must be configured for a new course:

Title The title of the course.

Course owner A user who automatically gets *course owner* permissions for this course. His email address is used as sender in student notifications.

Tutors A set of users that get *student tutor* permissions for this course.

Course description link A URL for the course home page. Used in the student dashboard.

Active The flag decides if any assignments from this course are shown to the students, regardless of their deadlines. This allows to put courses in an 'archive' mode after the term is over.

LTI key / LTI passphrase OpenSubmit supports the LTI protocol, so that you can integrate it into other learning management systems (LMSs) such as Moodle.

The LMS needs a consumer key and a shared secret resp. passphrase that you configure separately for each OpenSubmit course. This makes sure that the system knows automatically the course in which the external LMS user is interested in. Such users don't need to perform any authentication, OpenSubmit blindly believes in the identify information forwarded by the LMS. If a user already exists with the same email address, the LMS identity is added to his social login credentials.

Using LTI authentication can lead to duplicate accounts. You can *merge users* to fix that.

The machine-readable configuration for LTI, which can be parsed by some LMS systems, is available under the relative URL `/lti/config/`.

2.2.2 Providing a link to the course assignments

Location: `Teacher backend - Course - Info`

After using an *authentication provider*, students get automatically an account and end up on the student dashboard, where none of the OpenSubmit courses is activated for them. This leads to the fact that they can't see any course assignments by default.

If you want to make sure that students automatically see the assignments for your course, you need to tell OpenSubmit the course ID when entering the system. This can be done by linking to a course-specific OpenSubmit URL. It is shown in the course details on the teacher backend landing page.

2.2.3 Grading scheme creation

Location: `Teacher backend - System - Actions - Manage grading schemes`

Before you can create assignments for students, you must think about the grading scheme. A grading scheme is an arbitrary collection of gradings, were each grading either means 'pass' or 'fail'.

Grading schemes can later be used in the creation of assignments.

2.2.4 Assignment creation

Location: `Teacher backend - Course - Actions - Manage assignments - Add assignment`

With an existing course and an appropriate grading scheme, you can now create a new assignment:

Title (mandatory) The title of the assignment.

Course (mandatory) The course this assignment belongs to.

Grading scheme (optional) The grading scheme for this assignment. If you don't chose a grading scheme, than this assignment is defined as ungraded, which is also indicated on the student dashboard. Ungraded assignments are still validated.

Max authors (mandatory) For single user submissions, set this to one. When you choose larger values, the students get a possiblity to define their co-authors when submitting a solution.

Student file upload (mandatory) If students should upload a single file as solution, enable this flag. Otherwise, they can only enter submission notes. Students typically submit archives (ZIP / TGZ) or PDF files, but the system puts no restrictions on this.

Description (mandatory) The assignment description is linked on the student dashboard. It can either bei configured as link, f.e. when you host it by yourself, or can be uploaded to OpenSubmit.

Publish at (mandatory) The point in time where the assignment becomes visible for students. Users with teacher backend access rights always see the assignment in their student dashboard, so that they can test the validation before the official start.

Soft deadline (optional) The deadline shown to the students. After this point in time, submission is still possible, although the remaining time counter on the student dashboard shows zero.

If you leave that value empty, then the hard deadline becomes the soft deadline, too.

The separation between hard and soft deadline is intended for the typical late-comers, which try to submit their solution shortly after the deadline. Broken internet, time zone difficulties, dogs eating the homework ... we all know the excuses.

Hard deadline (optional) The deadline after which submissions for this assignment are no longer possible.

If you leave that value empty, then submissions are possible as long as the course is *active*.

Validation test (optional) The uploaded *validation test* is executed automatically for each student submission and can lead to different subsequent *states* for the submission. Students are informed about this state change by email. The test is executed before the hard deadline. It is intended to help the students to write a valid solution.

Download of validation test (optional) The flag defines if the students should get a link to the *validation test*. This makes programming for the students much more easy, since the can locally if their uploaded code would pass the validation checks.

Full test (optional) The uploaded *full test* is executed automatically for each student submission and can lead to different subsequent *states* for the submission. Students are *not informed* about this test. The test is executed after the hard deadline. It is intended to support the teachers in their grading with additional information.

Support files (optiona) A set of files that you want to have in the same directory when the *validation test* or the *full test* is running.

Test machines (mandatory in some cases) When you configure a *validation test* or *full test*, you need to specify the :*test machines* that run it. When chosing multiple machines, the testing load is distributed.

2.3 Managing submissions

A submission is a single (archive) file + notes handed in by a student. Every submission belongs to a particular assignment and its according course in OpenSubmit.

A student submission can be in different states. Each of the states is represented in a different way in student frontend and the teacher backend:

```
    # State description in teacher backend
    STATES = (

        # The submission is currently uploaded,
        # some internal processing still takes place.
        (RECEIVED, 'Received'),

        # The submission was withdrawn by the student
        # before the deadline. No further automated action
        # will take place with this submission.
        (WITHDRAWN, 'Withdrawn'),

        # The submission is completely uploaded.
        # If code validation is configured, the state will
        # directly change to TEST_VALIDITY_PENDING.
        (SUBMITTED, 'Submitted'),

        # The submission is waiting to be validated with the
        # validation script on one of the test machines.
        # The submission remains in this state until some
        # validation result was sent from the test machines.
        (TEST_VALIDITY_PENDING, 'Validity test pending'),

        # The validation of the student sources on the
        # test machine failed. No further automated action will
        # take place with this submission.
        # The students get informed by email.
        (TEST_VALIDITY_FAILED, 'Validity test failed'),

        # The submission is waiting to be checked with the
        # full test script on one of the test machines.
        # The submission remains in this state until
        # some result was sent from the test machines.
        (TEST_FULL_PENDING, 'Full test pending'),

        # The (compilation and) validation of the student
        # sources on the test machine worked, only the full test
        # failed. No further automated action will take place with
        # this submission.
        (TEST_FULL_FAILED, 'All but full test passed, grading pending'),

        # The compilation (if configured) and the validation and
        #  the full test (if configured) of the submission were
        # successful. No further automated action will take
        # place with this submission.
        (SUBMITTED_TESTED, 'All tests passed, grading pending'),

        # Some grading took place in the teacher backend,
        # and the submission was explicitly marked with
        # 'grading not finished'. This allows correctors to have
        # multiple runs over the submissions and see which
        # of the submissions were already investigated.
        (GRADING_IN_PROGRESS, 'Grading not finished'),

        # Some grading took place in the teacher backend,
        # and the submission was explicitly marked with
        # 'grading not finished'. This allows correctors
```

(continues on next page)

(continued from previous page)

```
        # to have multiple runs over the submissions and
        #  see which of the submissions were already investigated.
        (GRADED, 'Grading finished'),

        # The submission is closed, meaning that in the
        # teacher backend, the submission was marked
        # as closed to trigger the student notification
        # for their final assignment grades.
        # Students are notified by email.
        (CLOSED, 'Closed, student notified'),

        # The submission is closed, but marked for
        # another full test run.
        # This is typically used to have some post-assignment
        # analysis of student submissions
        # by the help of full test scripts.
        # Students never get any notification about this state.
        (CLOSED_TEST_FULL_PENDING, 'Closed, full test pending')
    )

    # State description in student dashboard
    STUDENT_STATES = (
        (RECEIVED, 'Received'),
        (WITHDRAWN, 'Withdrawn'),
        (SUBMITTED, 'Waiting for grading'),
        (TEST_VALIDITY_PENDING, 'Waiting for validation test'),
        (TEST_VALIDITY_FAILED, 'Validation failed'),
        (TEST_FULL_PENDING, 'Waiting for grading'),
        (TEST_FULL_FAILED, 'Waiting for grading'),
        (SUBMITTED_TESTED, 'Waiting for grading'),
        (GRADING_IN_PROGRESS, 'Waiting for grading'),
        (GRADED, 'Waiting for grading'),
        (CLOSED, 'Done'),
        (CLOSED_TEST_FULL_PENDING, 'Done')
    )
```

2.3.1 Submission grading

Location: `Teacher backend - Course - Manage submissions`

Location: `Teacher backend - Course - Manage assignments - Show submissions`

The grading of student submissions always follows the same workflow, regardless of the fact if you are using the automated testing facilities or not.

Short version:

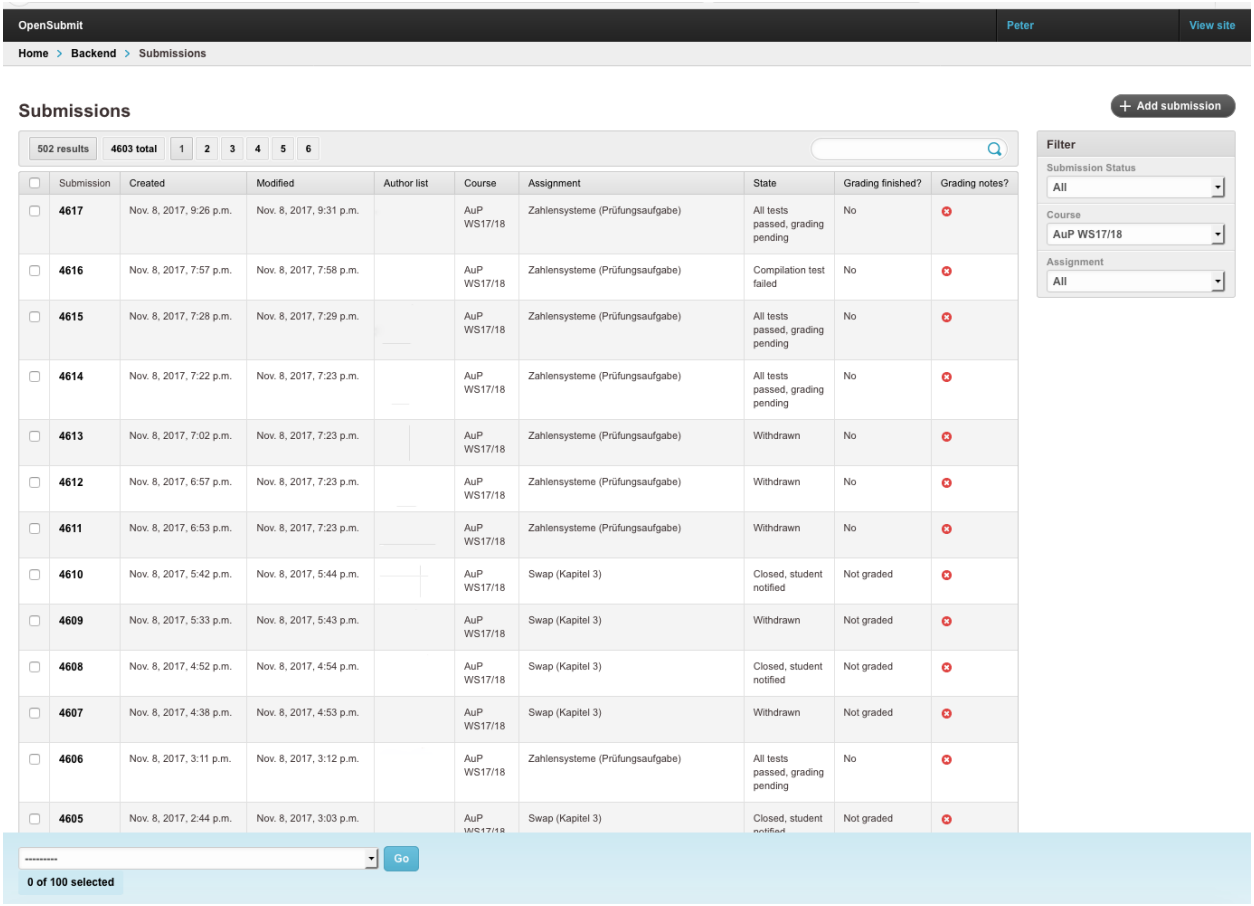
- For every submission:
  - Open the submission in the teacher backend.
  - Use the preview function for inspecting uploaded student archives.
  - Check the output from validation test and full test.
  - Optional: Add grading notes and a grading file for the student as feedback.
  - Decide for a grading, based on the provided information.

  - Mark the submission as **grading finished** if you are done with it.

- Close and notify all finished submissions as bulk action.

Long version:

On the right side of the submissions overview page, different filtering options are available.

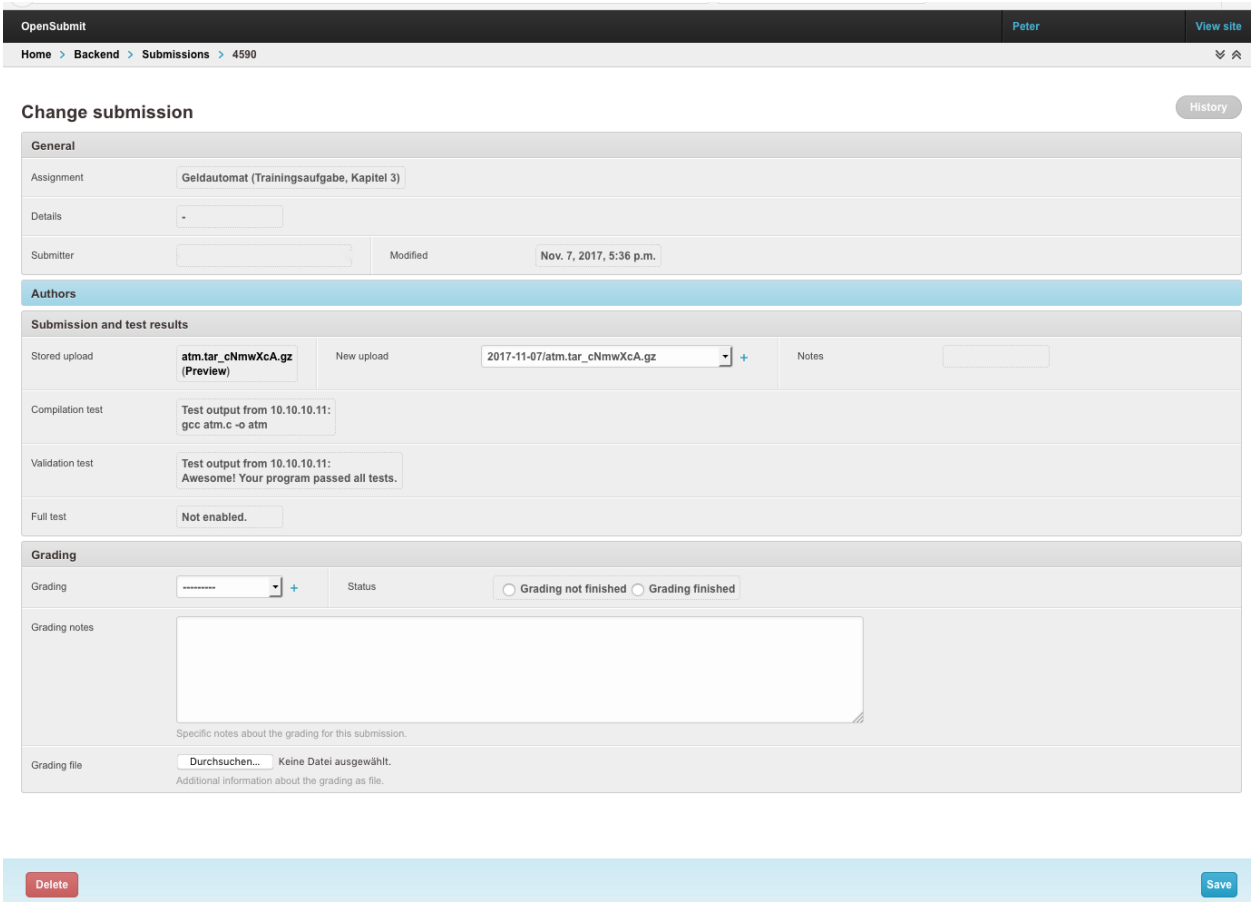


The most important thing is the distinguishing between **non-graded**, **graded** and **closed** submissions:

Non-graded submissions are the ones that were submitted (and successfully validated) before the hard deadline. Your task is to go through these submissions and decided for a particular grading. If this is done, than the grading is marked as being completed for this particular submission. This moves it into the **graded** state.

When all gradings are done, then the submissions can be **closed**. This is the point in time were the status for the students changes, before that, no notification is done. The idea here is to first finish the grading - maybe with multiple people being involved - before notifying all students about their results. Only submissions in the **graded** status can be closed. This is a safeguard to not forget the finishing of some grading procedure.

The submission details dialogue shows different information:



The assignment may allow the students to define co-authors for their submission. You can edit this list manually, for example when students made a mistake during the submission. The according section is hidden by default, click on the *Authors* tab to see it.

The original *submitter* of the solution is stated separately. Submitters automatically become authors.

Students can always add notes to their submission. If file upload is disabled for the assignment, this is the only gradable information.

The *file upload* of the students is available for direct download, simply by clicking on the file name. This is especially relevant when having text or PDF document as solution attachment. The *Preview* link opens a separate web page with a preview of the file resp. the archive content.

When *testing* is activated for this assignment, then the according result output is shown in the submission details.

The choice of a *grading* is offered according to the *grading scheme* being configured for the particular assignment. The *grading notes* are shown in the student frontend, together with the grade, when the submission is closed.

The *grading file* is also offered after closing, and may - for example - contain some explanary template solution or a manually annotated version of the student file upload.

The radio buttons at the bottom of the page allow to mark the submission as **non-graded** or **graded**.

When all submissions are finally graded, it is time to release the information to the students. In order to do this, mark on the overview page all finished submissions. This can be easily done by using the filters on the right side and the 'mark all' checkbox in the upper left corner. The choose the action 'Close graded submissions + send notification'.

2.3.2 Grading table

Location: `Teacher backend - Course - Show grading table`

If you want to have a course-level overview of all student results so far, use the *grading table* overview. It is available as action in the *Courses* section of the teacher backend.

2.3.3 Duplicate report

Location: `Teacher backend - Course - Manage assignments - Show duplicates`

A common task in assignment correction is the detection of cheating. In OpenSubmit terms, this leads to the question if different students have submitted identical, or at least very similar, solutions for an assignment.

Checking arbitrary code for similarities is a complex topic by itself and is closely related to the type and amount of code being checked. OpenSubmit follows it general *principles* here by not restricting the possible types of submission for a perfect duplicate detection. Instead, we encourage users with specific demands to use such services in their *testing scripts*.

OpenSubmit provides a basic duplicate checking for submitted files based on weak hashing of the student archives content. This method works independently from the kind of data and can, at least, detect the most lazy attempts of re-using other peoples work.

Based on the hashing results, the duplicate report shows groups of students that may have submitted the same result. This list must be treated as basic indication for further manual inspection. The report works independently from the course and the status of the submissions. Withdrawn solutions are skipped in the report.

2.4 Automated testing of submissions

The automated testing of submissions is performed by a Python 3 script that you, the assignment creator, have to write. This script is executed by OpenSubmit on some configured *test machines*. You are completely free in what you want to do in this script - at the end, OpenSubmit just needs an indication about the result. Common tasks, such as code compilation and execution, are supported by helper functions you can use in this script.

You can upload such a script in two ways:

- As single Python file named *validator.py*.
- As ZIP / TGZ archive with an arbitrary name, which must contain a file named *validator.py*.

The second option allows you to deploy additional files (e.g. profiling tools, libraries, code not written by students) to the test machine. OpenSubmit ensures that all these files are stored in the same directory as the student code and the Python script.

2.4.1 How to write a test script

Test scripts are written in Python 3.4 and will be directly called by the OpenSubmit daemon running on test machines.

You can install this daemon, which is also called *executor*, on your own computer easily. This gives you an offline development environment for test scripts while you are working on the assignment description.

Similar to the installation of *test machines*, the following procedure (for Debian / Ubuntu systems) gives you a testing environment:

- Install Python 3: `sudo apt-get install python3 python3-pip`

To keep your Python installation clean, we recommend to use Virtualenv:

- Install the Virtualenv tool: `sudo pip3 install virtualenv`
- Create a new virtual environment, e.g. in `~/my_env`: `python3 -m virtualenv ~/my_env`
- Activate it with `source ~/my_env/bin/activate`
- Install the OpenSubmit validator library / executor inside: `pip3 install opensubmit-exec`
- Develop the *validator.py* for your assignment.

Examples for test scripts can be found online.

We illustrate the idea with the following walk-through example:

Students get the assignment to create a C program that prints 'Hello World' on the terminal. The assignment description demands that they submit the C-file and a *Makefile* that creates a program called *hello*. The assignment description also explains that the students have to submit a ZIP archive containing both files.

Your job, as the assignment creator, is now to develop the `validator.py` file that checks an arbitrary student submission. Create a fresh directory that only contains an example student upload and the validator file:

```
1 def validate(job):
2     job.run_make(mandatory=True)
3     exit_code, output = job.run_program('./hello')
4     if output.strip() == "hello world":
5         job.send_pass_result("The world greets you! Everything worked fine!")
6     else:
7         job.send_fail_result("Wrong output: " + output)
```

The `validator.py` file must contain a function `validate(job)` that is called by OpenSubmit when a student submission should be validated. In the example above, this function performs the following steps for testing:

- Line 1: The validator function is called when all student files (and all files from the validator archive) are unpacked in a temporary working directory on the test machine. In case of name conflicts, the validator files always overwrite the student files.
- Line 2: The *make* tool is executed in the working directory with `run_make()`. This step is declared to be mandatory, so the method will throw an exception if *make* fails.
- Line 3: A binary called *hello* is executed in the working directory with the helper function `run_program()`. The result is the exit code and the output of the running program.
- Line 4: The generated output of the student program is checked for some expected text.
- Line 5: A positive validation result is sent back to the OpenSubmit web application with `send_pass_result()`. The text is shown to students in their dashboard.
- Line 6: A negative validation result is sent back to the OpenSubmit web application with `send_fail_result()`. The text is shown to students in their dashboard.

Test scripts are ordinary Python code, so beside the functionalities provided by the job object, you can use any Python functionality. The example shows that in Line 4.

If any part of the code leads to an exception that is not catched inside `validate(job)`, than this is automatically interpreted as negative validation result. The OpenSubmit executor code forwards the exception as generic information to the student. If you want to customize the error reporting, catch all potential exceptions and use your own call of `send_fail_result()` instead.

To check if the validator is working correctly, you can run the command `opensubmit-exec test <directory>` in your VirtualEnv. It assumes the given directory to contain a validator script resp. archive and the student submission file resp. archive. The command simulates a complete validation run on a test machine and prints exhaustive debugging information. The last line contains the feedback sent to the web application after finalization.

2.4.2 Test script examples

The following example shows a validator for a program in C that prints the sum of two integer values. The values are given as command line arguments. If the wrong number of arguments is given, the student code is expected to print *"Wrong number of arguments!"*. The student only has to submit the C file.

```
1  from opensubmitexec.compiler import GCC
2
3  test_cases = [
4      [['1', '2'], '3'],
5      [['-1', '-2'], '-3'],
6      [['-2', '2'], '0'],
7      [['4', '-10'], '-6'],
8      [['4'], 'Wrong number of arguments!'],
9      [['1', '1', '1'], 'Wrong number of arguments!']
10 ]
11
12 def validate(job):
13     job.run_compiler(compiler=GCC, inputs=['sum.c'], output='sum')
14     for arguments, expected_output in test_cases:
15         exit_code, output = job.run_program('./sum', arguments)
16         if output.strip() != expected_output:
17             job.send_fail_result("Oops! That went wrong! Input: " + str(arguments) +
   ↪", Output: " + output, "Student needs support.")
18             return
19     job.send_pass_result("Good job! Your program worked as expected!", "Student seems␣
   ↪to be capable.")
```

- Line 1: The *GCC* tuple constant is predefined by the OpenSubmit library and refers to the well-known GNU C compiler. You can also define your own set of command-line arguments for another compiler.
- Line 3-10: The variable *test_cases* consists of the lists of inputs and the corresponding expected outputs.
- Line 13: The C file can be compiled directly by using `run_compiler()`. You can specify the used compiler as well as the names of the input and output files.
- Line 14: The for-loop is used for traversing the *test_cases*-list. It consists of tuples which are composed of the arguments and the expected output.
- Line 15: The arguments can be handed over to the program through the second parameter of the `run_program()` method. The former method returns the exit code as well as the output of the program.
- Line 16: It is checked if the created output equals the expected output.
- Line 17: If this is not the case an appropriate negative result is sent to the student and teacher with `send_fail_result()`
- Line 18: After a negative result is sent there is no need for traversing the rest of the test cases, so the *validate(job)* function can be left.
- Line 19: After the traversal of all test cases, the student and teacher are informed that everything went well with `send_pass_result()`

The following example shows a validator for a C program that reads an positive integer from standard input und prints the corresponding binary number.

```
1  from opensubmitexec.exceptions import TerminationException
2
3  test_cases = [
4      ['0', '0'],
```

(continues on next page)

(continued from previous page)

```
    ['1', '1'],
    ['8', '1000'],
    ['9', '1001'],
    ['15', '1111']
]

def validate(job):
    job.run_build(inputs=['dec_to_bin.c'], output='dec_to_bin')
    for std_input, expected_output in test_cases:
        running = job.spawn_program('./dec_to_bin')
        running.sendline(std_input)
        try:
            running.expect(expected_output, timeout=1)
        except TerminationException:
            job.send_fail_result("Arrgh, a problem: We expected {0} as output for the input {1}.".format(expected_output, std_input), "wrong output")
            return
        else:
            running.expect_end()
    job.send_pass_result("Everything worked fine!", "Student seems to be capable.")
```

- Line 1: A *TimeoutException* is thrown when a program does not respond in the given time. The exception is needed for checking if the student program calculates fast enough.
- Line 3-9: In this case the test cases consist of the input strings and the corresponding output strings.
- Line 12: The method `run_build()` is a combined call of *configure*, *make* and the compiler. The success of *make* and *configure* is optional. The default value for the compiler is GCC.
- Line 13: The test cases are traversed like in the previous example.
- Line 14: This time a program is spawned with `spawn_program()`. This allows the interaction with the running program.
- Line 15: Standard input resp. keyboard input can be provided through the `sendline()` method of the returned object from line 14.
- Line 17-20: The validator waits for the expected output with `expect()`. If the program terminates without producing this output, a *TerminationException* exception is thrown.
- Line 22: After the program successfully produced the output, it is expected to terminate. The test script waits for this with `expect_end()`
- Line 23: When the loop finishes, a positive result is sent to the student and teacher with `send_pass_result()`.

Warning: When using `expect()`, it is important to explicitely catch a *TerminationException* and make an explicit fail report in your validation script. Otherwise, the student is only informed about an unexpected termination without further explanation.

The following example shows a validator for a C program that reads a string from standard input and prints it reversed. The students have to use for-loops for solving the task. Only the C file has to be submitted.

```
from opensubmitexec.exceptions import TimeoutException
from opensubmitexec.exceptions import TerminationException

test_cases = [
```

(continues on next page)

(continued from previous page)

```
    ['hallo', 'ollah'],
    ['1', '1'],
    ['1234', '4321']
]

def validate(job):
    file_names = job.grep('.*for[:space:]*(.*;.*;.*).*')
    if len(file_names) < 1:
        job.send_fail_result("You probably did not use a for-loop.", "Student is not able to use a for-loop.")
        return

    job.run_build(inputs=['reverse.c'], output='reverse')
    for std_input, expected_output in test_cases:
        running = job.spawn_program('./reverse')
        running.sendline(std_input)
        try:
            running.expect(expected_output, timeout=1)
        except TimeoutException:
            job.send_fail_result("Your output took to long!", "timeout")
            return
        except TerminationException:
            job.send_fail_result("The string was not reversed correctly for the following input: " + std_input, "The student does not seem to be capable.")
            return
        else:
            running.expect_end()
    job.send_pass_result("Everything worked fine!", "Student seems to be capable.")
```

- Line 1: A *TimeoutException* is thrown when a program does not respond in the given time. The exception is needed for checking if the student program calculates fast enough.
- Line 2: A *TerminationException* is thrown when a program terminates before delivering the expected output.
- Line 4-8: The test cases consist of the input strings and the corresponding reversed output strings.
- Line 11: The `grep()` method searches the student files for the given pattern (e.g. a for-loop) and returns a list of the files containing it.
- Line 12-14: If there are not enough elements in the list, a negative result is sent with `send_fail_result()` and the validation is ended.
- Line 16-24: For every test case a new program is spawned with `spawn_program()`. The test script provides the neccessary input with `sendline()` and waits for the expected output with `expect()`. If the program is calculating for too long, a negative result is sent with `send_fail_result()`.
- Line 25: If the result is different from the expected output a *TerminationException* is raised.
- Line 26-27: The corresponding negative result for a different output is sent with `send_fail_result()` and the validation is cancelled.
- Line 28-29: If the program produced the expected output the validator waits with `expect_end()` until the spawned program ends.
- Line 30: If every test case was solved correctly, a positive result is sent with `send_pass_result()`.

2.5 Developer reference

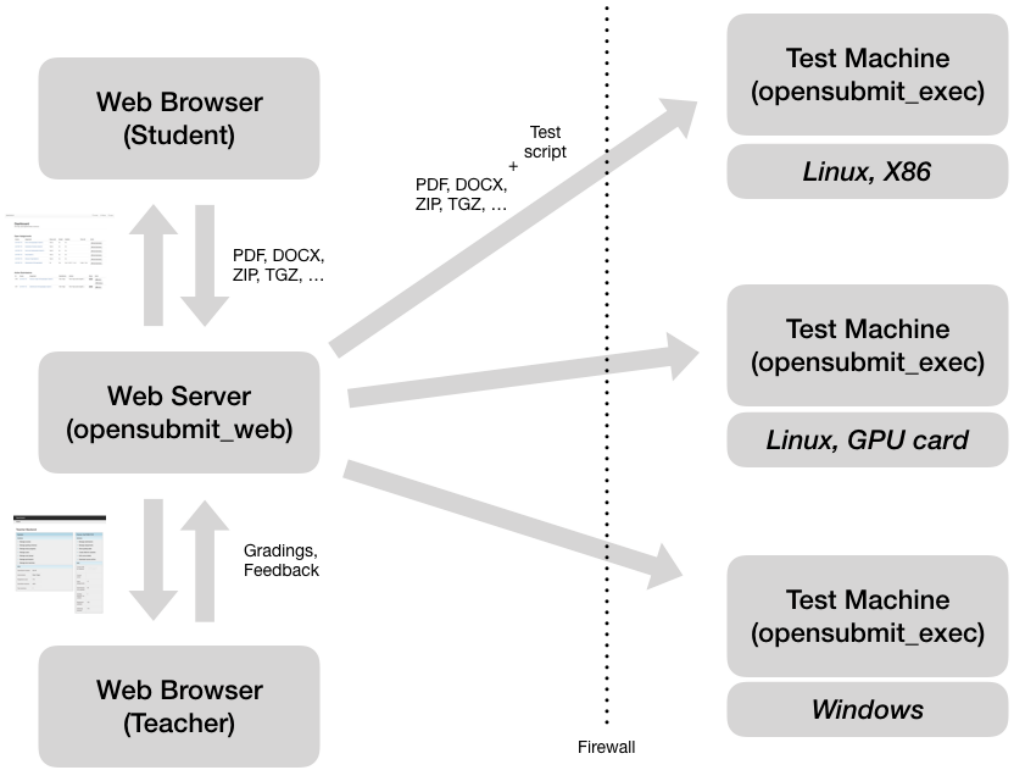
The Job class summarizes all information about the submission to be validated by the test script. It also offers a set of helper functions that can be directly used by the test script implementation.

Test scripts can interact with a running student program, to send some simulated keyboard input and check the resulting output for expected text patterns.

CHAPTER 3

Administrator Manual

OpenSubmit consists of two parts: The web application and the executor daemon. The web application provides the user interface for students and teachers, while the executor daemons evaluate student code with according test scripts.



If you just want to play around, use our demo installation.

If you want your own production setup, go the manual way for *web application* and *test machines*.

Please note that OpenSubmit *does not support password-based login*. You need to work with one of the supported *Authentication methods*.

3.1 Full-stack installation with Terraform

The source code repository offers a Terraform script for deploying a complete OpenSubmit environment on a single machine running in the Google Compute Engine. The installation procedure deploys the Docker containers described in the following sections. For such an installation:

- Install Terraform on your local machine.
- Clone the Git repository for OpenSubmit and adjust the variables in terraform.tf.
- Call `terraform apply`.

This setup is not recommended for production, since the database is installed as Docker image.

3.2 Full-stack installation with Docker Compose

You can replicate our demo installation on your own machine with Docker Compose, which comes as part of a normal Docker installation. Our compose file relies on the official Docker images for the web application and the executor.

- Download the compose file on the machine.
- Call `docker-compose up` to download, configure and start the OpenSubmit Docker containers and a separate database container.
- Got to `http://localhost:8000` and use one of the configured authentication methods.

This setup is not recommended for production, since the database is installed as Docker image.

3.3 Single installation of the web application

The OpenSubmit web application runs with Python 3.4 or newer versions. There are two options:

3.3.1 Docker-based installation

The latest official release of the OpenSubmit web application is available as single opensubmit-web Docker image. It expects a couple of environment variables to be set, check the *configuration section* for further details.

3.3.2 Manual installation

This is the recommended approach for production environments. You need to follow these steps:

- Prepare a Python 3 web hosting environment.
  - Debian / Ubuntu: `apt-get install libapache2-mod-wsgi-py3 apache2 sqlite python3`.

- Run `pip install opensubmit-web` as root or in a virtualenv environment. If you get error messages about unresolved dependencies, try running `pip install -U opensubmit-web`. PIP should come as part of your Python installation.
- Run `opensubmit-web configcreate` to create an OpenSubmit configuration file. Check the *configuration section* for details.
- Run `opensubmit-web configtest` to check your configuration.
- Run `opensubmit-web apachecreate` to generate a default Apache 2.4 configuration for mod_wsgi, which is stored in `/etc/opensubmit/apache24.config`. You can include this file in some virtual host configuration.
- Restart your web server.
- Got to the OpenSubmit start page and use your configured authentication method.
- Run `opensubmit-web makeadmin <email>` to make the created user an administrator in the system.

Updating an existing manual installation is easy:

- Run `pip install --upgrade opensubmit-web` as root or in the virtualenv environment.
- Run `opensubmit-web configtest` to perform neccessary database updates.
- Restart your web server.

3.3.3 Configuration of the web application

OpenSubmit searches for a configuration file in `/etc/opensubmit/settings.ini`. This file should be initially created by calling `opensubmit-web configcreate`. This management command allows to pre-define specific configuration options via command-line or environment variables, and creates an according config file. Check `opensubmit-web configcreate -h` for details.

Impress and privacy policy

There are several European regulations that expect a web page to provide both an impress and a privacy policy page (GDPR / DSGVO). There are two ways to achieve that:

- Option 1: Your configuration file defines name, address, and email of an administrator. The according options for `opensubmit-web configcreate` are `--admin_name`, `--admin_email`, and `--admin_address`. If you want to modify settings.ini directly, add `ADMIN_NAME`, `ADMIN_EMAIL` and `ADMIN_ADDRESS` in the `[admin]` section. The first two settings are mandatory anyway. Given that information, OpenSubmit will provide a default impress and privacy policy page.
- Option 2: Your configuration file defines alternative URLs for impress page and privacy policy page. The according options for `opensubmit-web configcreate` are `--admin_impress_page` and `--admin_privacy_page`. If you want to modify settings.ini directly, add `IMPRESS_PAGE` and `PRIVACY_PAGE` options with the links in the `[admin]` section.

3.3.4 Authentication methods

OpenSubmit supports different authentication methods, as described in the following sections. It *does not support password-based logins* - authentication is always supposed to be handled by some third-party service.

If you need another authentication method for your institution, please open an according issue.

Authentication methods show up on the front page when the according settings are not empty. You can therefore disable any of the mechanisms by commenting them out in settings.ini.

Login with OpenID

If you want to allow users to login with OpenID, you need to configure the following settings:

- `LOGIN_DESCRIPTION: <visible button title>`
- `OPENID_PROVIDER: <provider URL>`

The standard OpenSubmit installation already contains an example setting for using StackExchange as OpenID provider.

Login with Shibboleth

If you want to allow users to login with Shibboleth, you need to configure the following settings:

- `LOGIN_SHIB_DESCRIPTION: <visible button title>`

You also need a fully working installation of the Apache 2.4 mod_shib module. The authentication module of OpenSubmit assumes that, as result of the work of *mod_shib*, the following environment variables are given:

- `REMOTE_USER`: The user name of the authenticated user.
- `HTTP_SHIB_ORGPERSON_EMAILADDRESS`: The email address of the authenticated user.
- `HTTP_SHIB_INETORGPERSON_GIVENNAME`: The first name of the authenticated user.
- `HTTP_SHIB_PERSON_SURNAME`: The last name of the authenticated user.

Note: If you are using Apache 2.4 with *mod_wsgi*, make sure to set `WSGIPassAuthorization On`. Otherwise, these environment variables may not pass through.

Login with Google account

If you want to allow users to login with an Google account, you need to configure the following settings:

- `LOGIN_GOOGLE_OAUTH_KEY: <OAuth key>`
- `LOGIN_GOOGLE_OAUTH_SECRET: <OAuth secret>`

A new pair can be created in the Google API Console. The authorized forwarding URL should be `<base url of your installation>/complete/google-oauth2/`.

You also need to activate the Google+ API, so that OpenSubmit is able to fetch basic user information from Google.

Login with Twitter account

If you want to allow users to login with an Twitter account, you need to configure the following settings:

- `LOGIN_TWITTER_OAUTH_KEY: <OAuth key>`
- `LOGIN_TWITTER_OAUTH_SECRET: <OAuth secret>`

A new key / secret pair can be created in the Twitter Application Management. The authorized forwarding URL should be `<base url of your installation>/complete/twitter/`. We recommend to modify the application access to *Read only*, and to allow access to the email addresses.

Login with GitHub account

If you want to allow users to login with an GitHub account, you need to configure the following settings:

- `LOGIN_GITHUB_OAUTH_KEY: <OAuth key>`
- `LOGIN_GITHUB_OAUTH_SECRET: <OAuth secret>`

A new key / secret pair can be created in the OAuth application registration. The authorized forwarding URL should be `<base url of your installation>/complete/github/`.

3.3.5 User management

One of the core concepts of OpenSubmit is that users register themselves by using an external authentication provider (see *Authentication methods*).

Based on this, there are different groups such a registered user can belong to:

- *Students* (default): Users who cannot access the teacher backend.
- *Student Tutors*: Users with limited rights in the teacher backend.
- *Course Owners*: Users with advanced rights in the teacher backend.
- *Administrators*: Users will unrestricted rights.

Permissions

The following table summarized the default permissions for each of the user groups.

Permission	Students	Student Tutors	Course Owners	Administrators
Student Frontend	Yes	Yes	Yes	Yes
• Create submissions	Yes	Yes	Yes	Yes
• Withdraw submission	Yes	Yes	Yes	Yes
• See unpublished assignments	No	Yes	Yes	Yes
Teacher Backend	No	Yes	Yes	Yes
• eMail to participants	No	Yes[1]	Yes[2]	Yes[2]
• Manage/grade submissions	No	Yes[1]	Yes[2]	Yes[2]
• Manage assignments	No	No	Yes[2]	Yes[2]
• Manage grading schemes	No	No	Yes	Yes
• Manage study programs	No	No	Yes	Yes
• Manage courses	No	No	Yes	Yes
• Manage users	No	No	No	Yes
• Manage test machines	No	No	No	Yes
• Manage custom permissions	No	No	No	Yes

Administrators can create custom user groups and permissions. Normally this should be avoided, since some permissions have a non-obvious impact on the usage of the teacher backend.

[1] Only for courses where the user was chosen as tutor.
[2] Only for courses where the user was chosen as tutor or course owner.

Assigning users to groups

There are two ways to assign users to user groups, assuming that they logged-in once for registration:

- In the teacher backend, as administrator (see *Authentication methods*).
- With the `opensubmit-web` command-line tool.

The first option is the web-based configuration of user groups, which is only available for administrators. Click on *Manage users* and mark all user accounts to be modified. After that, choose an according action in the lower left corner of the screen.

The second option is the `opensubmit-web` command-line tool that is available on the web server. Calling it without arguments shows the different options to assign users to user groups.

Merging accounts

Since OpenSubmit users always register themselves in the platform (see *Authentication methods*), it can happen that the same physical person creates multiple accounts through different authentication providers. The main reason for that is a non-matching or missing email address being provided by the authentication provider.

Administrators can merge users in the teacher backend. Click on *Manage users*, mark all user accounts to be merged, and choose the according action in the lower left corner. The nect screen shows you the intended merging activity and allows to chose the "primary" account by flipping roles. The non-primary account is deleted as part of the merging activity.

3.4 Single installation of a test machine

Test machines are used to run the validation scripts (see *Automated testing of submissions*) for student submission. Pending validation jobs are fetched from the OpenSubmit web server in regular intervals and executed on a test machine.

The creator of an assignment can chose which test machines are used for the validation. This enables a flexible setup with dedicated test machines for special assignments, e.g. GPU programming.

There are two options for installation:

3.4.1 Docker-based installation

The latest official release of the OpenSubmit executor application is available as opensubmit-exec Docker image. It expects a couple of environment variables to be set, check the *configuration section* for details.

3.4.2 Manual installation

Both the validator library and the job fetching is implemented in a Python package called `opensubmit-exec` (the *executor*). It runs with Python 3.4 or newer versions. For an installation, you need to follow these steps:

- Choose a dedicated machine beside the web server. It will compile (and run) the student submissions.
- Think again. IT WILL RUN THE STUDENT SUBMISSIONS. Perform all neccessary security precautions, such as network isolation and limited local rights.
- Install Python >= 3.4 on the machine. e.g. through `sudo apt-get install python3 python3-pip`.

- Run `pip3 install opensubmit-exec` as root or in a virtualenv environment. If you get error messages about unresolved dependencies, try running `pip install -U opensubmit-exec`. PIP should come as part of your Python installation.
- Create an initial configuration as described in the *configuration section*.
- Run `opensubmit-exec configtest` to check your configuration.
- Add a call to `opensubmit-exec run` to cron, so that it regulary asks the web server for fresh work. We have good experiences with a 30s interval. You can also do it manually for testing purposes.

Smart students may try to connect to machines under their control in their code, mainly for copying validation scripts. An easy prevention mechanism is the restriction of your test machine network routing so that it can talk to the web server only.

The fetching of validations is protected by a shared secret between the web application and the executor installations. Check both the `settings.ini` on the web server and `executor.ini` on the test machines.

Updating an existing manual executor installation consists of the following steps:

- Run `pip install --upgrade opensubmit-exec` as root or in a virtualenv environment.
- Run `opensubmit-exec configtest` to check the configuration for compatibility.

3.4.3 Configuration of the executor

OpenSubmit searches for a configuration file in `/etc/opensubmit/executor.ini`. This file should be initially created by calling `opensubmit-exec configcreate`. This management command allows to pre-define specific configuration options via command-line or environment variables, and creates an according config file. Check `opensubmit-exec configcreate -h` for details.

CHAPTER 4

Developer Manual

The development of OpenSubmit is coordinated on GitHub. We need help in everything. Feel free to join us.

The central Makefile is a good starting point. It supports several targets for preparing a development environment:

make venv Prepares a virtualenv with all neccessary packages for packaging and running OpenSubmit.

make runserver Perform all neccessary preparations to run the Django development server. This includes the creation of a configuration file, the execution of the neccessary database creation / migration steps and the startup of the server.

make tests Run the test suite.

make coverage Run test suite and create code coverage analysis.

make docs Build the HTML documentation.

make build Create Python installation packages (wheels).

make docker-build Create Docker images for web application and executors.

make docker Run Docker containers for the web application, executors and a PostgreSQL database.

4.1 Writing documentation

The OpenSubmit manuals are hosted on ReadTheDocs and are generated with Sphinx. You find the according `.rst` files in the docs/ folder.

CHAPTER 5

About

5.1 The Zen of OpenSubmit

Following an old tradition, there is a set of guiding principles for the design of OpenSubmit:

Minimalism is good. OpenSubmit follows the philosophy that teachers know the best how their teaching works. This leads to the conclusion that teaching policies and workflows do not belong into our code or complicated configuration options. Assignment rules vary widely in different institutions and groups. Given that, it is a main philosophy of OpenSubmit to reduce the functionality to the submission and validation of of student submissions. And nothing else. This simplifies the student user interface and clarifies the teacher workflow. Whenever some design decision is restricting what teachers can do with the framework, it might be bad.

Passwords are bad. History has shown that even the largest companies fail at implementing a secure password authentication mechanism. Doing this properly includes captcha management, email roundtrips, recovery procedures, two-factor magic, identity checks, permanent software updates, and solid basic crypto knowledge. There are better ways to spend our restricted resources. OpenSubmit therefore does not have a password-based authentication mechanism. Instead, we support the authentication through third-party services. Since every eductional institution already has an existing scheme for that, we focus on integrating them properly instead.

Machines don't grade. Humans do. Even though OpenSubmit is focusing on the automated validation of student submissions, we do not aim for automated grading. These ideas became popular in the context of MOOCs, but cannot work in an educational environment where the future of humans depends on the certificates they get. OpenSubmit is therefore focusing on supporting teachers in their grading in every possible way, so that bulk activities (grading teams, duplicate checks a.s.o) becomes a fast and painless activity.

Students are too creative. OpenSubmit is intended to deal with the fact that student are extremely creative in what they submit as solution. Especially with code. The tool should be the forgiving middleman that translates the arbitrary student package into something that can be graded fast and easily.

If you are interested in the why's and how's of these principles, check our (slightly outdated) `presentation` from LinuxTage 2016.

5.2 License

OpenSubmit is licensed under the AGPL Version 3. This means you are allowed to:

- Install and run the unmodified OpenSubmit code at your site.
- Re-package and distribute the unmodified version of OpenSubmit.
- Modify and re-publish (fork) the sources, as long as your modified versions are accessible for everybody.

In short, AGPL forbids you to distribute / run your own modified version of OpenSubmit without publishing your changes. This does not relate to configuration files.

5.3 Acknowledgements

People who contributed to this project so far:

- Peter Tröger (project owner)
- Jafar Akhundov (testing)
- Omar Alhaffar (testing)
- Srikar Beechu (testing)
- Kai Fabian (code, testing)
- Frank Feinbube (patches, testing)
- Sven Köhler (patches, testing)
- Jens Pönisch (testing)
- Bernhard Rabe (patches, testing)
- Martin Richter (docs, patches, testing)
- Matthias Werner (testing)

CHAPTER 6

Changelog

6.1 v0.7.8 Release

This is a stable release which adds support for compliance to the GDPR / DSGVO regulations in Europe.

If you upgrade from a v0.6.x release, make sure that you read the *v0.7.0 release notes*!

Here is the list of changes since the last official release:

- OpenSubmit now provides an impress and a privacy policy page. Details about the configuration can be found in the documentation.
- Script download was broken on the executors and is now fixed.
- The release changelog moved to the official documentation.

6.2 v0.7.4 Release

This is a stable release that only brings internal and administrative improvements, with no visible impact for end users.

If you upgrade from a v0.6.x release, make sure that you read the *v0.7.0 release notes*!

Here is the list of changes:

- We now offer Docker images for the web application and the executor. The *Administrator Manual* was updated accordingly.
- We now offer a demo installation at http://www.demo.open-submit.org (#98). This lead to a new configuration option called `DEMO`, which allows to enable passthrough login buttons on the landing page.
- We now offer a Terraform-based installation of OpenSubmit on cloud computing resources. Check the Terraform section in the admin manual for further details.
- The traditional `opensubmit-web configure` call is now split up into three explicit commands:

opensubmit-web configcreate Creates a new config file for OpenSubmit. Supports several command-line options and environment variables for pre-defining configuration options, as described in the manual section about *Configuration of the web application* (#238).

opensubmit-web apachecreate Creates a new Apache configuration snippet, based on an existing OpenSubmit configuration.

opensubmit-web configtest Checks the current configuration for validity. Supposed to be called after updates.

- The new `HOST_ALIASES` configuration option allows you to set alternative host names for your OpenSubmit web machine. This makes sure that the CSRF protection does not prevent users from entering the site under a different name.
- All views are now Django class-based views, which eases the future development and implicitely improves the catching of illegal HTTP requests (#233).
- We switched to Django 1.11.
- We switched to a new LTI support library, which hopefully improves the compatibility to LMS systems. There is now also support for *automated LTI configuration*.

Make sure that you run `opensubmit-web configtest` resp. `opensubmit-exec configtest` after installation.

This release is compatible to executors from the v0.7 series.

Installation is possible with:

```
pip install --upgrade opensubmit-web; opensubmit-web configtest; service
apache2 restart
```

```
pip install --upgrade opensubmit-exec; opensubmit-exec configtest
```

6.3 v0.7.3 Release

This is a stable release with some urgent patches and minor updates for the 0.7 series functionalities.

If you upgrade from a v0.6.x release, make sure that you read the *v0.7.0 release notes*!

Here is the list of changes:

- The student frontend got a small design change (#219). Withdrawn submissions are now collected on a separate page (“Archive”). The landing page provides three sections with open work (=open assignments the student can submit for), work in progress (=submissions under validation / grading) and finished work (=submissions that where graded, positively validated or where the deadline is over). This also allows to access assignments from the past, even when the deadline is over, as long as the course remains active. The student manual was updated accordingly.
- You can now send mails to a set of students (#123) from the list of submissions.
- The grading table got more powerful, you can now enable / disable the assignments to be shown (#214).
- Validation scripts can produce dedicated messages that are only visible to tutors. They are now also shown in the teacher backend (#213).
- The documentation is now clearer about the Job.expect() interface and the role of TimeoutException.
- The link to the assignment download in the submission details is now fixed. It also shows more details with this update.
- Assignment lists in the teacher backend are now sorted.

- Error code generated by student programs are no longer modified, but reported as-is by the executors (#215).
- The output of student programs was saved with double new-lines. This is fixed now (thanks to @tttee).
- The footer now links to the student / teacher manual page. The teacher backend link now only shows when the user has the according rights.
- The code base is now automatically checked for security bugs in the dependencies. Keyboard input created by the validation script is no longer double-echoed (#229).
- We got a logo!

Make sure that you run `opensubmit-web configure` resp. `opensubmit-exec configure` after installation.

This release is compatible to executors from the v0.7 series.

Installation is possible with:

```
pip install --upgrade opensubmit-web; opensubmit-web configure; service
apache2 restart
```

```
pip install --upgrade opensubmit-exec; opensubmit-exec configure
```

6.4 v0.7.2 Release

This is a stable release with some minor fixes.

If you upgrade from a v0.6.x release, make sure that you read the *v0.7.0 release notes*!

Here is the list of changes:

- Fixed a bug that prevented executors from removing their generated temporary files. (#210)
- Executors now also stop working, with an error report for every tested submission, when they run out of disk space. (#208)
- The file preview loads faster and shows line numbers. (#162)
- Full tests can now only be started for submissions that are not already under test. (#211)
- The configured maximum number of authors for an assignment is now checked in the student frontend (#205) Thanks to @tzwenn for reporting this issue.
- The teacher manual now provides a lot more information and examples about writing validation test scripts (#207, #209).

Make sure that you run `opensubmit-web configure` resp. `opensubmit-exec configure` after installation.

This release is compatible to executors from the v0.7 series.

Installation is possible with:

```
pip install --upgrade opensubmit-web; opensubmit-web configure; service
apache2 restart
```

```
pip install --upgrade opensubmit-exec; opensubmit-exec configure
```

6.5 v0.7.0 Release

After several months of beta testing, this is the largest release ever made for OpenSubmit.

There are two major changes that make this upgrade more important (and more painful) than the ones before:

- OpenSubmit no longer supports Python 2. You need Python 3.4 or newer, both on the web server and and test machines.
- The programming model for test scripts has changed in an incompatible way.

With this release, we also introduce the new home page at http://open-submit.org. It currently offers a set of (unfinished) manuals for students, course owners and administrators.

This update is the first major change, since 2012, in the way how test scripts are written. We hope that the new features and future possibilities are convincing enough for the additional upgrade efforts.

Thanks to @tttee and @tzwenn for contributing patches to this release.

6.5.1 Changes in comparison to v0.6.12

- The web application (opensubmit_web) and the executor daemon (opensubmit_exec) are now written in Python 3. You need to adjust your web server configuration and, in case, your Virtualenv installation accordingly (see below).
- The separation between admin backend and teacher backend is gone (#179). There is only a teacher backend now. Administrative actions are offered in the 'System' section of the teacher dashboard. Everbody, including the administrators, is therefore now forced to go through the student authentication page.
- Since admins have no longer a separate user name / password entry into the system, they need a different way to manage initial user permissions. This is realized with new features in the `opensubmit-web` command-line tool. It supports explicit role assignment (`make_student`, `make_owner`, `make_admin`), based on an user email address. As an alternative, these actions are also offered in the user section of the teacher backend. (#9)
- The `opensubmit-web` tool now also has a `create_demo` command. It installs a set of dummy courses, dummy assignments and dummy users for quick testing.
- Assignments can now be non-graded, simply by not chosing a grading scheme in the assignment configuration. Assignments can now also be published without a deadline. Both things are indicated in the student dashboard, the ordering was adjusted accordingly. (#183, #198, #177)
- Several list views in the teacher backend now have advanced sorting and search support.
- File names of student submissions are now kept. This ensures that Makefiles being provided by the validator package always work. (#149)
- Test machines can now be disabled. This gives you an upgrade path when switching to v0.7-style test scripts - disable all test machines, exchange the test scripts in the assignments, and re-enable them.
- Student eMails are now more detailed. (#202)
- Test machines now can have a human-readable name. If this is not given, than the old naming scheme applies (#201).
- Assignment descriptions can now be uploaded to, and served by the OpenSubmit installation. You are still able to use an external link for the assignment description. (#172, #174)

Beside these changes, there were also several internal improvements:

- Since we switched to Python 3, all installation packages are now wheels.
- Since we switched to Python 3, all UTF-8 rendering issues are now solved (#182, #184).

- There is improved support for contributors by integrating Travis CI and Scrutinizer, by making PEP-8 a reality in many code parts, and by supporting Anaconda as default IDE.
- Due to the complete re-write of the executor code, the error reporting and internal logging is now much more detailed (#191, #193, #196). The new executor checks by itself if it is still compatible to the contacted version of the OpenSubmit web application.
- OpenSubmit will now start to follow the PEP-440 version scheme. This allows us to release beta versions that are not installed during a regular upgrade procedure of your Python installation.
- Many little bugs were fixed (#181, #185, #186, #197, #203, #200, #199, #180, #190).

6.5.2 The new test script format

The newly offered OpenSubmit manual is the central source of information for how to write a test script. Here is the short overview of differences for upgrading users:

- A validation test or full test script can now only be written in Python >=3.4. It contains a single function `validate(job)` that is called by the executor. It still must be named validator.py, but can be stored within an archive with additional support files.
- All information about the student submission is available in the provided `Job` object. Check the manual for more details. (#113)
- The `Job` object also offers a set of convinience functions, such as searching for keywords in the submitted student files. Check the manual. (#6, #124)
- The result reported to the student is now sent explicitely by the test script, and no longer implicitely derived from the exit code of the script. If you forget to send a result in your validator, then every function run not throwing an exception is reported as success with a default message. Check the online examples.
- Calling `configure`, `make` or the compiler is now an explicit activity in the test script. This reduces the amount of options for assignments in the web interface, and increases the flexibility on the testing side. It also leads to the fact that support files are no longer an extra thing, since they can be simply added to the test script archive (#189). We hope that this fundamental architectural change, and the complete re-factoring of the code, helps to solve traditional problems with Windows-based test machines (e.g. #144). This one is for you, @thehappyhippo.
- Based on the fantastic *pexpect* library, you can now interact with the running student application in your test script code. This includes the support for student applications that expect a TTY. Check the example.

There are updated online examples for test scripts in the new format. We are also still working on imroving the manual for teachers - stay tuned.

6.5.3 Upgrade hints

The upgrade from an existing v0.6.12 installation demands a little bit more effort. We recommend to follow this procedure:

- Make a database backup. Seriousely.
- Install Python 3.4 or better on your web server, including `pip3` for getting Python 3 packages.
- Make sure that your web server can run Python 3 code, f.e. by installing `libapache2-mod-wsgi-py3`.
- Run `pip3 install --upgrade opensubmit-web` to fetch OpenSubmit into your Python 3 installation.

- Run `opensubmit-web configure`, as usual. The configuration file format did not change, but there is a larger set of database migrations that must be executed for this release. The Apache 2.4 configuration is also re-generated in a format that fits to `libapache2-mod-wsgi-py3`.
- Restart the web server.
- Go to the teacher backend and disable all test machines.
- Install Python 3.4 or better on your test machines, including `pip3` for getting Python 3 packages.
- Run `pip3 install --upgrade opensubmit-exec` to fetch OpenSubmit into your Python 3 installation.
- Run `opensubmit-exec configure`, as usual. If you see strange error messages, try to delete `/etc/opensubmit/executor.ini` and re-run `opensubmit-exec configure` to create a new one. In case, adjust it accordingly.
- Start to port your test scripts to the new format, and upload them for your assignments.
- Re-enable the test machines and check if the validation works again.

This release is, obviously, only compatible to executors from the v0.7 series.

6.6 Releases before v0.7.0

All release notes before v0.7.0 used to live on GitHub, and where accidentially deleted in February 2018. Don't play around with `git tag -d ...`