# Code Checker Web Documentation - CCW

## Intro

CCW was developed for TU Graz in the 2019/2020 academic year as a prototype system to assist students in learning the “Java” programming language through solving tasks given by their mentors through an interactive online interface. The system is primarily meant for solving programming tasks for practice but could also be built up to be used for homework assignments and exams.

## Developer installation guide

1. Extract the zip file to a folder.

2. Install and start a database, use MySQL database. Note `username`,`password`, `port` and `host`

3. Create a schema and note its name.

4. Open the file `src/main/resources/application.properties`

5. Replace the following with appropriate values:

1. `spring.datasource.url=jdbc:mysql://<host>:<port>/<schema\_name>`

2. `spring.datasource.username=<username>`

3. `spring.datasource.password=<password>`

6. Make sure you have [JDK 8](https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html) installed and added to your path. It will not run with just JRE.

1. To test run `javac -version` and you should see `javac 1.8.XXXX`

7. Make sure you have [Maven] (https://maven.apache.org/install.html) installed and added to path.

1. To test run `mvn -version` and you should see detailed output about location of install etc.

8. If using windows, use `mvnw.bat [goals]`:

1. run `mvnw.bat install spring-boot:run`

2. to package and generate war, run `mvnw.bat clean package`

9. If using \*nix like any Linux distro, Unix or Mac OS X:

1. run `chmod +x mvwn` for the first time.

2. run `mvnw install spring-boot:run`

3. to package and generate war, run `mvnw clean package`

## System set up guide JAR

1. Place the first ccw\_create.jar into your preferred directory. This will create the DB schema.
2. Run it and check on [https://[ IP](https://[IP) ADDRESS]:8443/login if you get a login page.
3. Click on register and make an account that you will turn into an admin account.
4. Go into the table “Users” and change the value of the “ROLES” field from “ROLE\_USER” to “ROLE\_ADMIN”.
5. Shut down the ccw\_create.jar.
6. Start up the ccw\_update.jar. The system should now be fully utilisable.

## Task creation guide

Here we will describe what is currently possible when it comes to creating task logic. It is assumed that a user will always make **a function** which will then be tested for correctness, the function must always be **public.** This is the general syntax to specify the function and what it should return (// used to describe what should be inputted in what field):

{

"function": [

"Fib", // Name of the function

"int", // Return type of the function as dictated by the java language specification ex. String

["int"] // One or more function argument types separated by commas (,)

],

"tests": [ //a list of test pairs with the following syntax [[function arg1, arg2, argn], expected return value]

[["0"], "0"],

[["1"], "1"],

[["2"], "1"], //for example here the 2 is the argument for the test and 1 is the expected return

[["42"], "267914296"]

]

}

Another example but this time with lists:

{

"function": [

"Sorter",

"int[]",

["int[]"]

],

"tests": [

[["5"], "5"],

[["3,2,1"], "1,2,3"], //Lists are defined as strings separated by commas (,)

[["5,6,5,4"], "4,5,5,6"]

]}

Palindrome example:

{

"function": [

"Palindrome",

"Boolean",

["String"]

],

"tests": [

[["a"], "true"],

[["aaa"], "true"],

[[""], "true"],

[["ab"], "false"],

[["absxsafsgagasgsd"], "false"],

[["abaaaba"], "true"]

]

}

Note: All examples can be found in the appendix along with proper solutions.

## General flow

The structure of the application should be understandable to anyone with a basic knowledge of Spring Boot and its design philosophy so here we will focus more on how the compilation and testing works for future development to be as easy to get into as possible.

As soon as a user clicks **Solve** the **Solution** entity is sent to the **CoordinatorService** via the sendSolution() function. The coordinator’s main role is to coordinate the testing of solutions. The solution is placed into its queue and every 10 seconds the queue is emptied and the testing is done on all the tasks in the solutions\_queue. ProcessSolution(Solution s) then receives the individual solution. The line cmsg\_list = cs.start(tsk.getTests(), tsk.getMaxtime(), s.getAnswer()); extracts all the necessary elements for testing and compilation and returns an array with the test results.

The **CompilerService** (above as cs) is responsible for testing and code compilation. As previously mentioned **public** ArrayList<String> start(String tests, Long maxtime, String user\_code) receives the tests in the format as described in the task creation guide. The maximum amount of time that anyone test is allowed to run and the user submitted code solution. The user submitted function is placed inside a dummy class for execution via the line

String contents = String.*format*("package test1; import java.util.Arrays; public class Dummy { %s } ", user\_code);. What follows is Java Source file construction as well as a method to capture the output of the java compiler which goes directly into standard output. Finally *jCompile*(files) is called on the in-memory java source file created before. If the compilation is successful, an “ok” is added to the result array. For testing two arrays are then prepared ArrayList<Object[]> ex\_inputs and String ex\_outputs[] which stand for expected inputs and outputs respectively. Essentially one describes the parameters that will go into a function and the other the expected outputs that a function will return as defined in the task creation section.

The *testIt*(ex\_outputs, cname, fname, params, ex\_inputs, maxtime) function then does the actual execution of the user code and testing. With thisClass.getDeclaredMethod(fname, params); and java reflection the correct function for testing is retrieved from the JVM environment from it’s name and parameters. The next for loop goes through all the tests and uses the expected inputs from each and forwards them to the retrieved function and executes the whole thing in a new thread. The program then waits until maxtime passes and if the thread has not terminated in the elapsed time an exception is raised on it via the deprecated method cex.~~stop~~();. It is worth noting that this solution is not ideal as it goes against java good practice. With cex.getRet(); we get the return value of the function. It is then tested and categorised and appropriate points are added.