Introduction to CI with TravisCI and Maven

# Required software

To start with the task, you must first install Java JDK and an IDE. In this guide I use Java 10.0.2 and Eclipse Oxygen 3.

* Java: <http://www.oracle.com/technetwork/java/javase/downloads/jdk10-downloads-4416644.html>
* Eclipse: <https://www.eclipse.org/downloads/packages/release/oxygen/3/eclipse-ide-java-ee-developers>

As editor for the Travis and Bash files Atom should be installed. Alternatively, you can install a Bash and a Yaml plugin for eclipse.

* Atom Editor: <https://atom.io/>

A GitHub and TravisCI account and an active internet connection are also required.

# Preparations

After the necessary software has been installed, we create an empty project. For this we will use the Spring Initializr. First go to <https://start.spring.io/>.

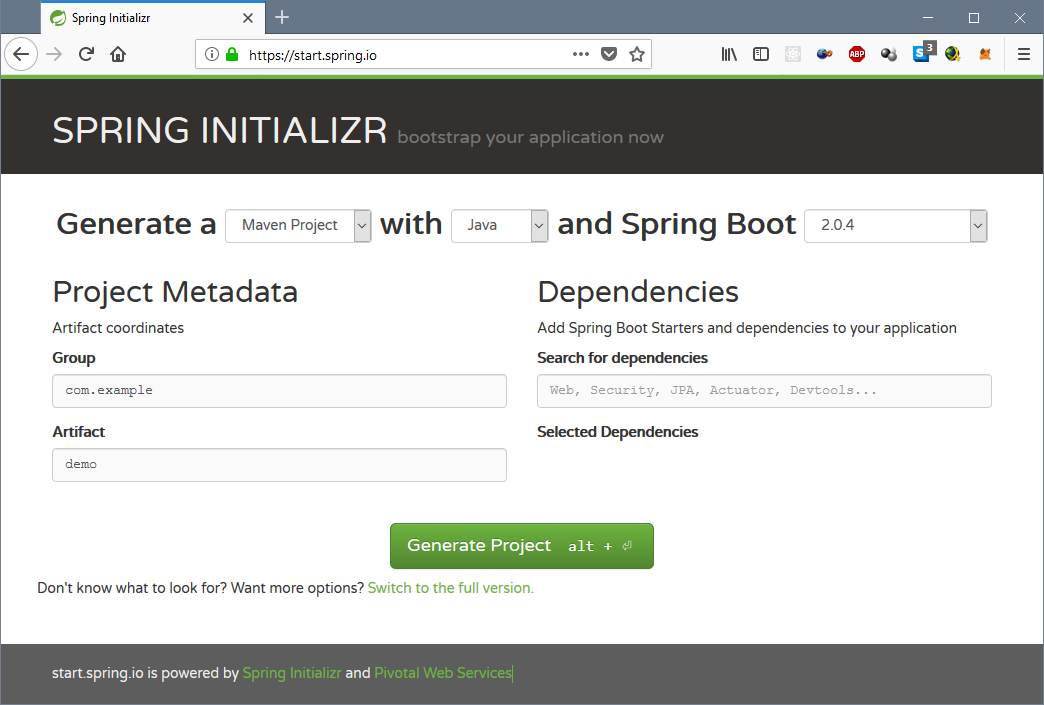


Figure 1: Empty Spring Initializr

Then fill in the fields as follows:

As a group we use "com.aut.StudentID". As an Artifact we use "TravisSpringDemo".

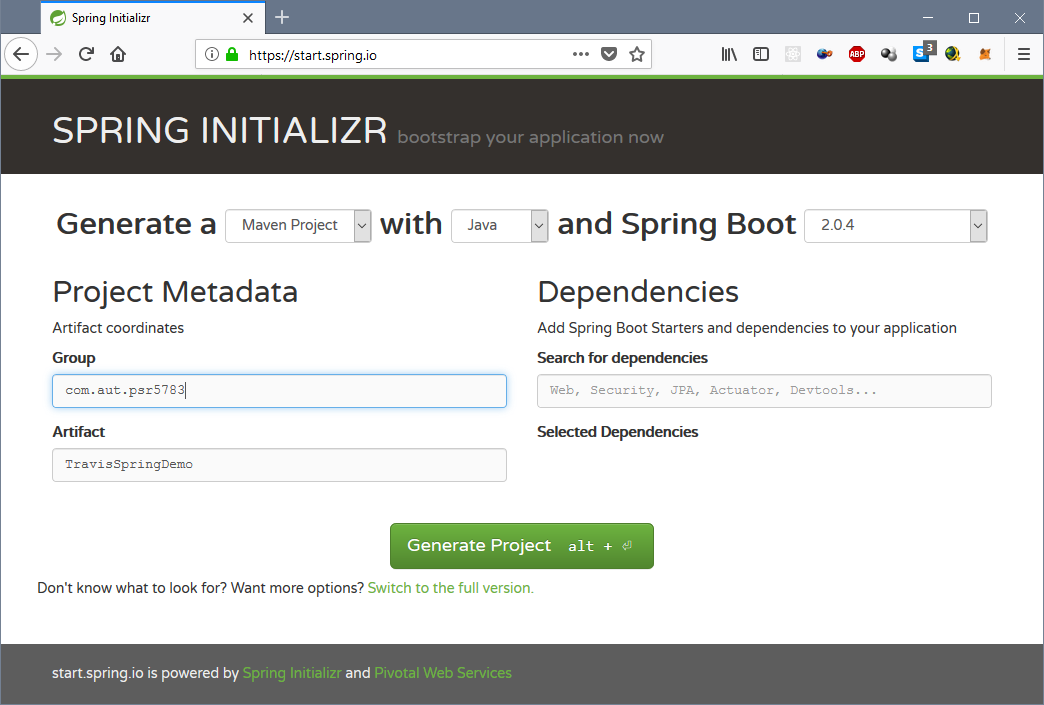


Figure 2: Example configuration for the "Spring Initializr"

Now you can click on "Generate Project" to download your empty project.

# GitHub and TravisCI setup

The next step is to create a GitHub repo and link it to TravisCI. First create an empty GitHub Repo. My GitHub Repo is called "MavenTravisGitDemo" and is published on GitHub (<https://github.com/fwidder/MavenTravisGitDemo>).

In the second step we will connect TravisCI with GitHub. To do this, we go to <https://travis-ci.com/>. After registering, we click on the profile icon in the upper right corner and click on "Profile". After connecting our Travis profile to our GitHub profile, we should see all available repos.

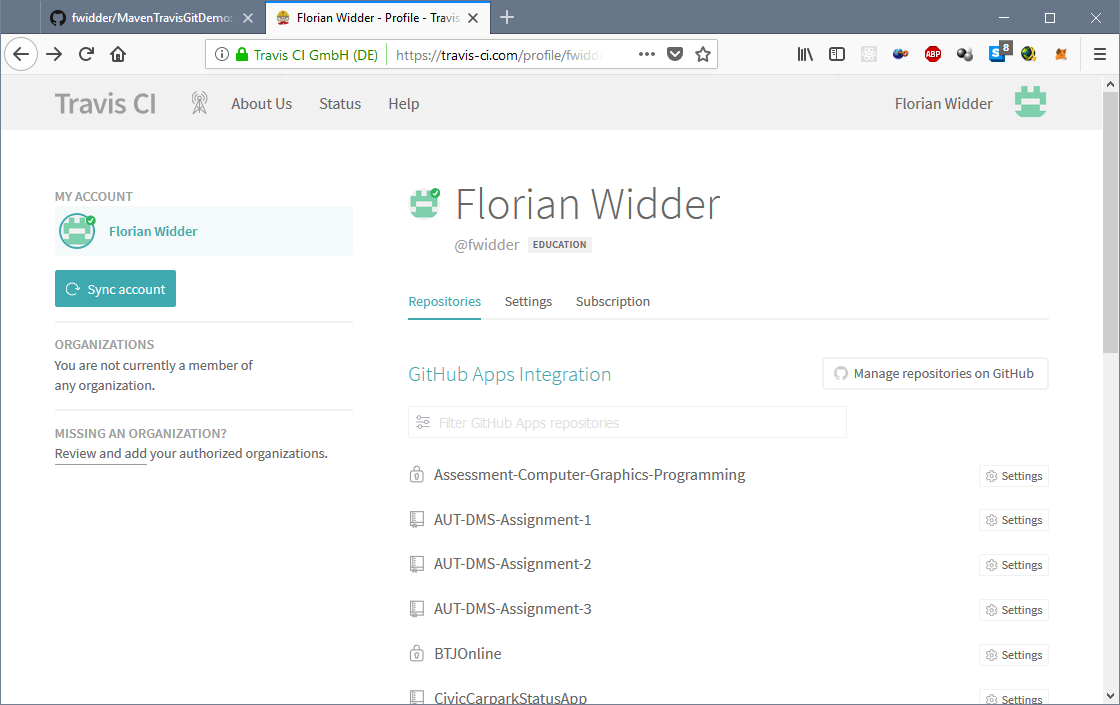


Figure 3: View of all repos

Find your newly created repo and click on the gear behind the name of the repo. Make sure the settings for your repo are the same as shown in Figure 4: Settings for TravisCI.

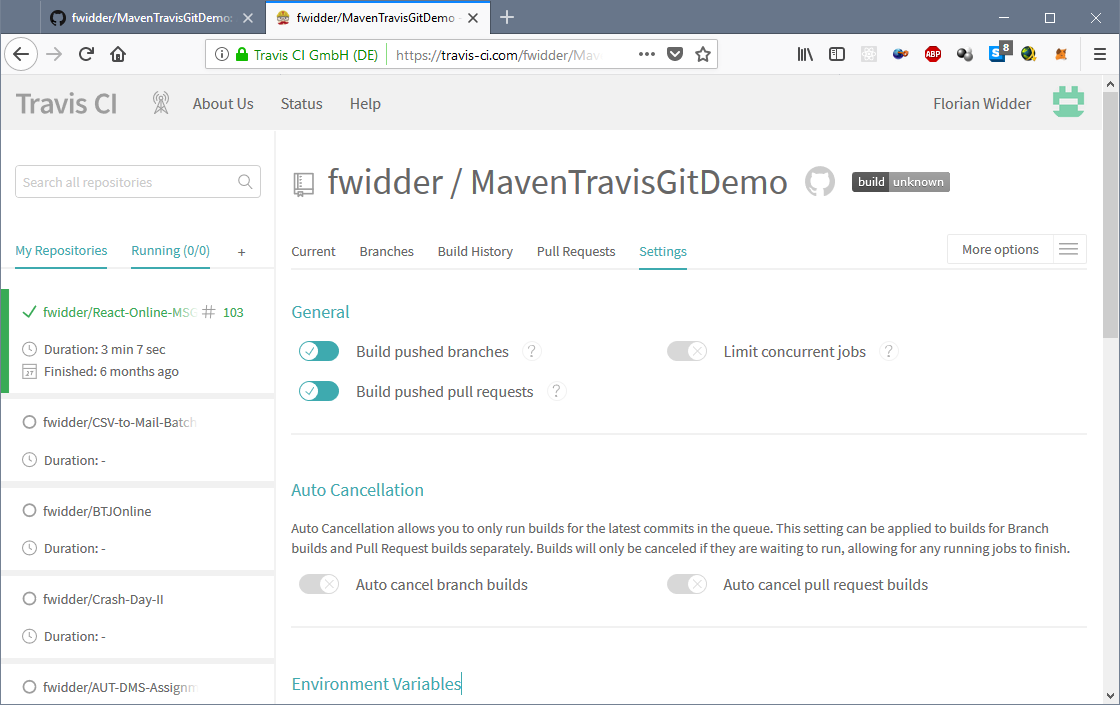


Figure 4: Settings for TravisCI

# Importing and filling the empty project into Eclipse

First extract the ZIP folder containing the empty project into your Eclipse workspace (e.g. "C:\Users\Flori\eclipse-workspace-aut"). Then right click on "Project Explorer" and go to "Import" → "Import...".

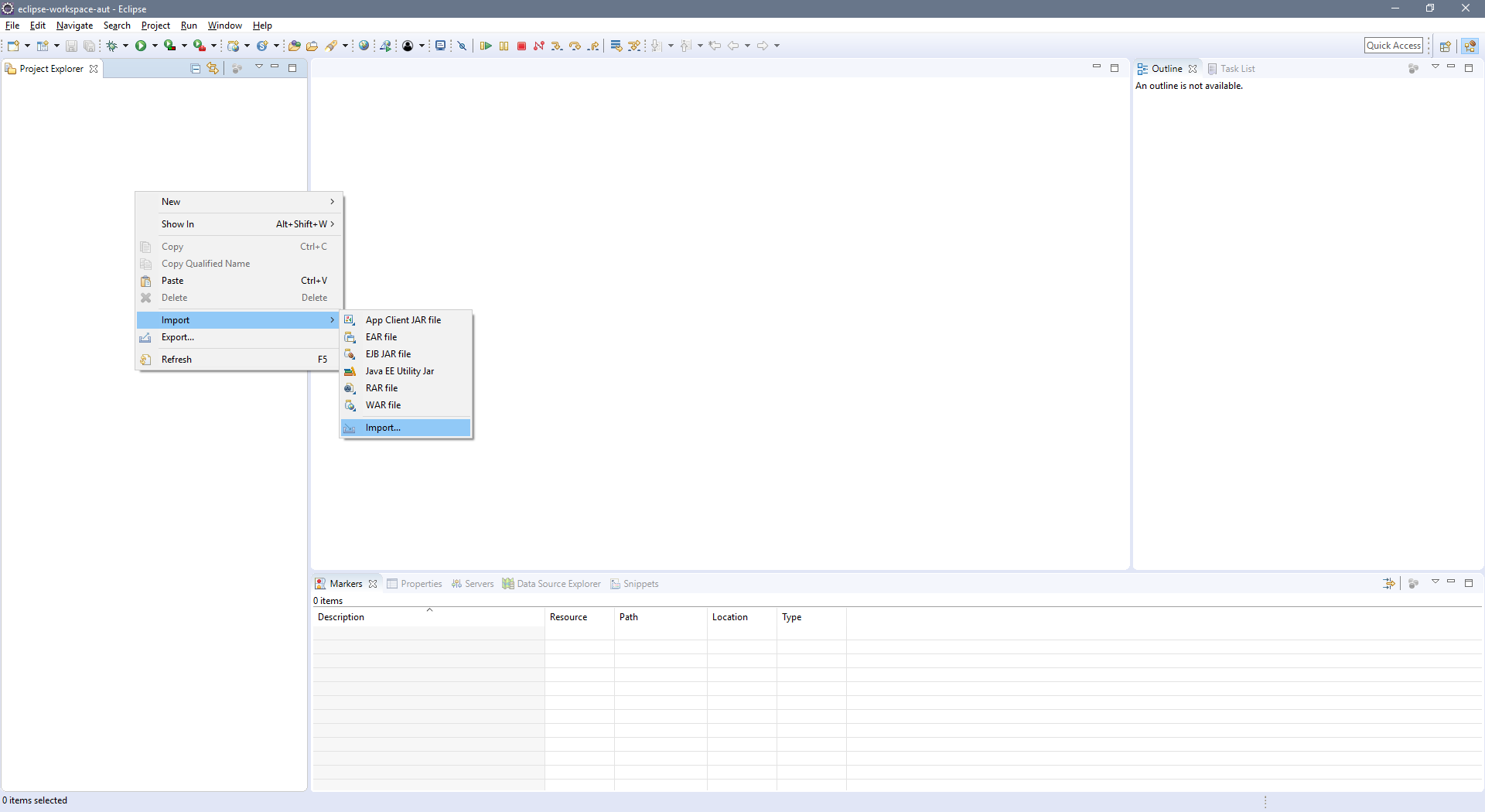


Figure 5: Import menu 1

Select Existing Maven Projects and click on “next”.

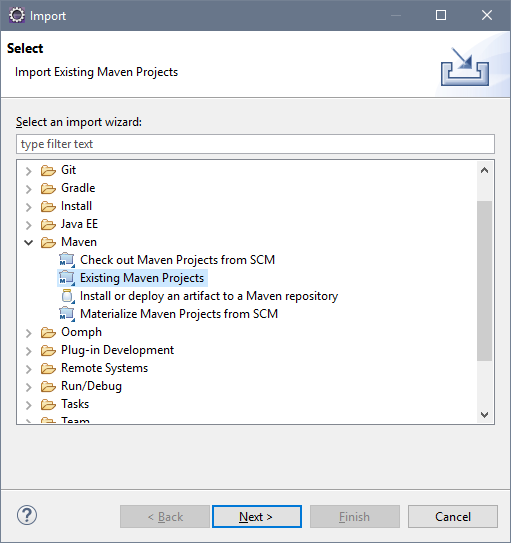


Figure 6: Import menu 2

Select with a click on "Browse" the folder in which your project is located and then click on "Finish".

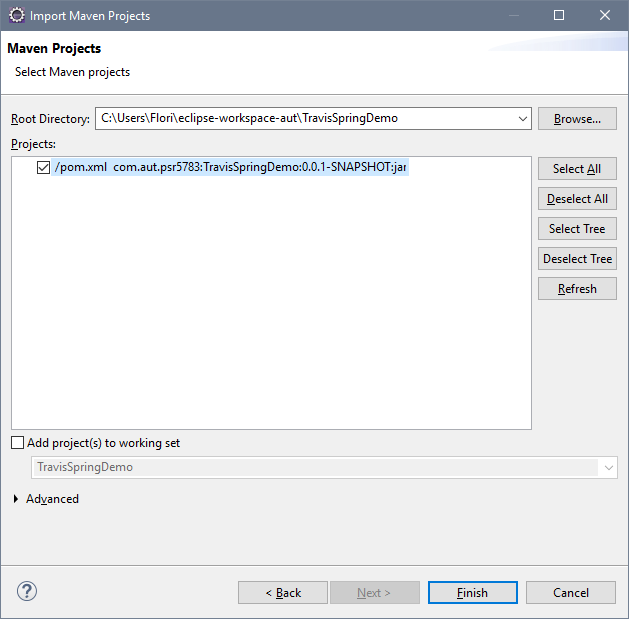


Figure 7: Import menu 3

Then we create the class "SimpleMath" in "src/main/java - com.aut.psr5783.TravisSpringDemo" and the class "SimpleMathTests in "src/test/java - com.aut.psr5783.TravisSpringDemo".Then we fill the two classes with some content. Just copy the code following the code into the classes:

|  |  |
| --- | --- |
| SimpleMath | SimpleMathTests |
| /\*\*  \* **@author** Florian Widder  \*  \* An example class with simple calculations.  \*/  **package** com.aut.psr5783.TravisSpringDemo;  **public** **class** SimpleMath {  /\*\*  \* Stores the number of performed calculations.  \*/  **private** **long** operations;  /\*\*  \* Standard Constructor  \*/  **public** SimpleMath() {  operations = 0;  }  /\*\*  \* Adds two numbers.  \*  \* **@param** a  \* **@param** b  \* **@return** a + b  \*/  **public** **double** add(**double** a, **double** b) {  operations++;  **return** a + b;  }  /\*\*  \* Subtracts two numbers from each other.  \*  \* **@param** a  \* **@param** b  \* **@return** a - b  \*/  **public** **double** subtract(**double** a, **double** b) {  operations++;  **return** a - b;  }  /\*\*  \* Divides two numbers.  \*  \* **@param** a  \* **@param** b  \* **@return** a / b  \*/  **public** **double** divide(**double** a, **double** b) {  operations++;  **return** a / b;  }  /\*\*  \* Multiply two numbers.  \*  \* **@param** a  \* **@param** b  \* **@return** a \* b  \*/  **public** **double** multiply(**double** a, **double** b) {  operations++;  **return** a \* b;  }  /\*\*  \* Returns the number of calculations performed.  \*  \* **@return** number of calculations performed  \*/  **public** **long** getNumberOfCalculatedOperations() {  **return** operations;  }  } | /\*\*  \*  \* **@author** Florian Widder  \*  \* Tests for SimpleMath  \*  \*/  **package** com.aut.psr5783.TravisSpringDemo;  **import** **static** org.junit.Assert.\*;  **import** org.junit.Before;  **import** org.junit.Test;  **public** **class** SimpleMathTests {  SimpleMath math;  @Before  **public** **void** setUp() **throws** Exception {  math = **new** SimpleMath();  }  @Test  **public** **void** testAdd() {  *assertTrue*(math.add(2, 3) == 5);  }  @Test  **public** **void** testSubtract() {  *assertTrue*(math.subtract(6, 3) == 3);  }  @Test  **public** **void** testDivide() {  *assertTrue*(math.divide(10, 2) == 5);  }  @Test  **public** **void** testMultiply() {  *assertTrue*(math.multiply(12, 3) == 37);  }  @Test  **public** **void** testGetNumberOfCalculatedOperations() {  **long** operationsOld = math.getNumberOfCalculatedOperations();  math.add(1, 1);  math.add(2, 2);  math.add(3, 3);  math.add(4, 4);  *assertEquals*(math.getNumberOfCalculatedOperations(), operationsOld + 4);  }  } |

After that it should look like this.

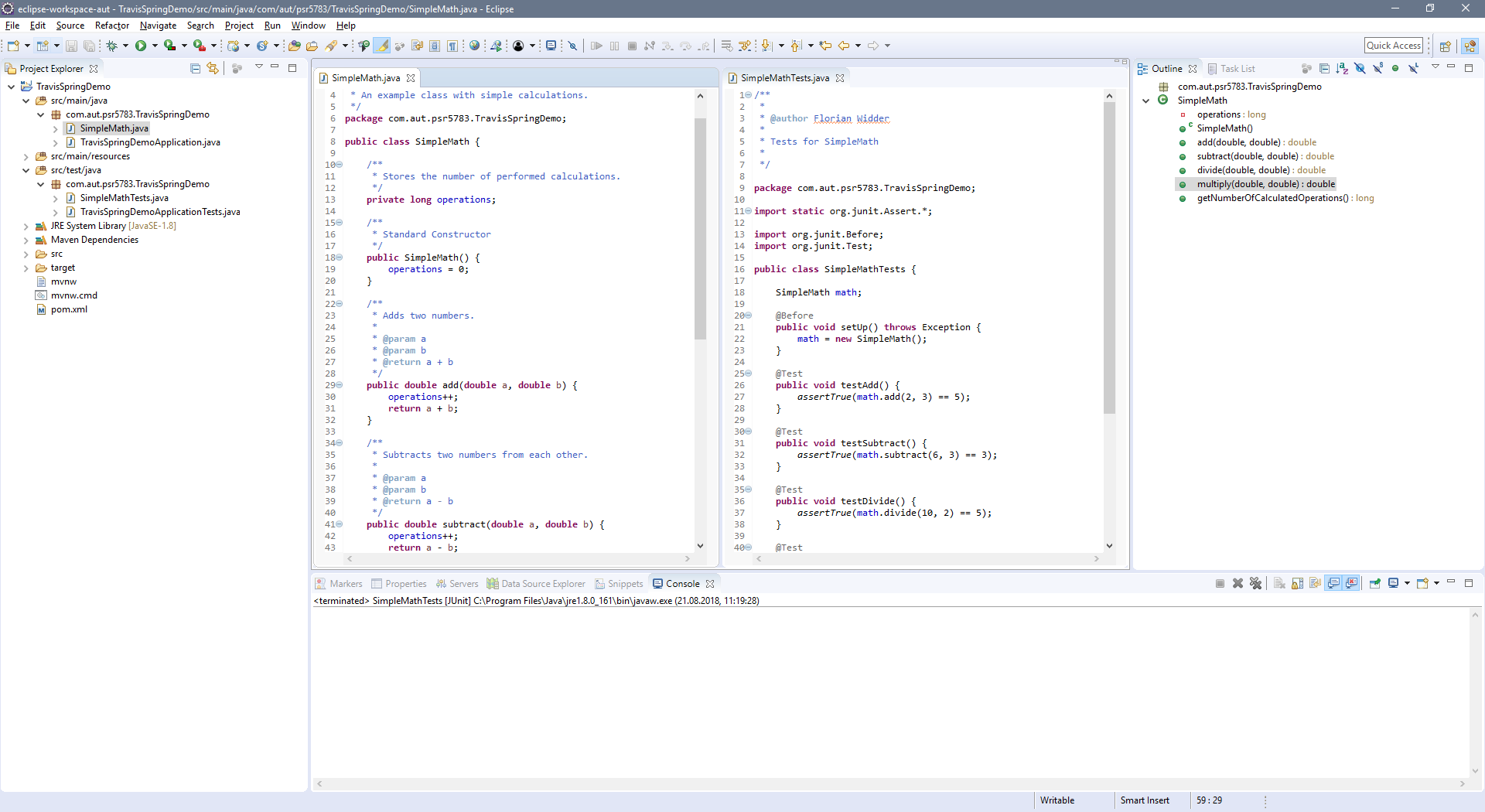


Figure 8: Eclipse after creating the Java code

# Creating the TravisCI script

Now we best switch to Atom because it supports the required syntax highlighting. Just open your Eclipse workspace as a project in Atom. Then create a file named “. travis.yml”. If there is a file with this name in any of your TravisCI related projects, Travis will automatically try to build the project. The file is built according to the normal YAML syntax.

Writing a Travis script that performs our JUnit tests only requires a very simple script.

First we have to define the language we want to use: Java.

Our script needs administrator rights so we have to set sudo to true.

The Java version we want to use is 9 or 10, I have chosen 10 in the example. You can also choose between openjdk and oraclejdk. I have chosen the Oracle version here.

In the next step we define what needs to be done in preparation (before executing the actual script). For us, this is making the Maven wrappers executable and downloading the dependencies. the commands are defined in a YAML array. last but not least, we test our software. this is done with the simple"./mvnw test" command. if we want to do other things after that (such as test case coverage evaluations), we would simply add them to the array.

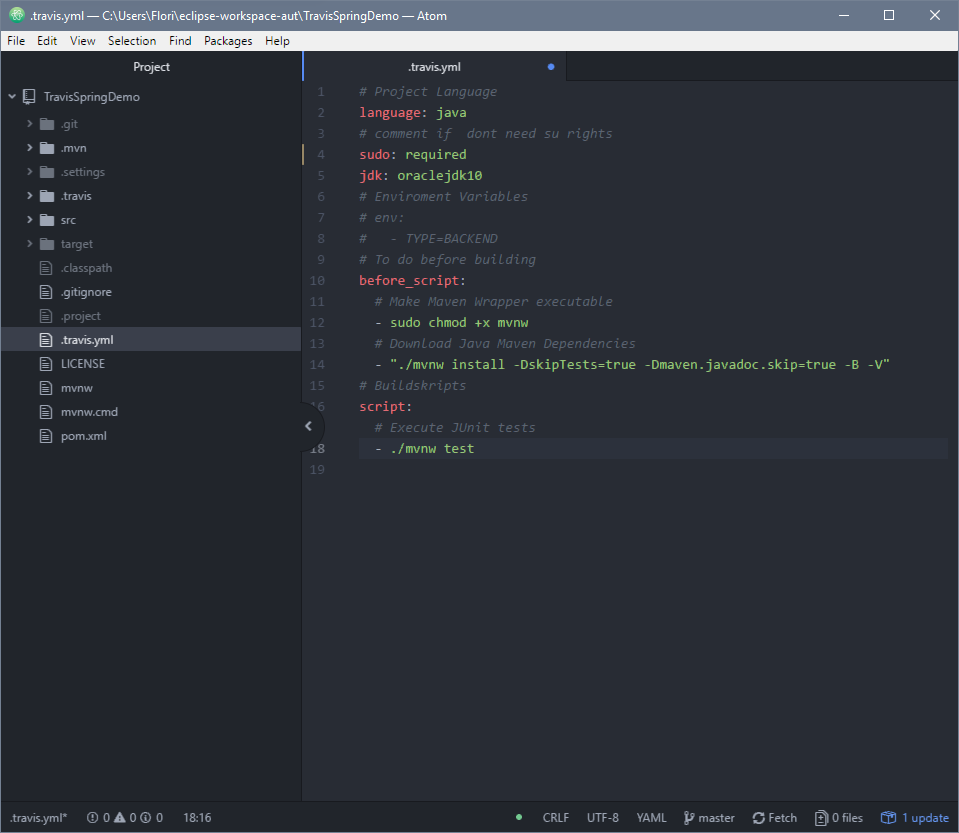


Figure 9: Travis Script

# Commit to GitHub and build on TravisCI

To upload the code to GitHub we must open a command line in our project folder. I will work with the Windows Powershell.

We start to create a local repo. Then we add all files in the current folder to our repo.

git init

git add .

Then we have to allow travis to run the maven wrapper.

git update-index --chmod=+x mvnw

After that we must commit the changes.

git commit -m "First Version"

Finally, we push everything to our GitHub repo (you have to replace the URL with your repo).

git remote add origin https://github.com/fwidder/TestRepo.git

git push -u origin master

If you did everything right, travis should now have started building your repo. This should only take a few minutes. *And it should fail…*

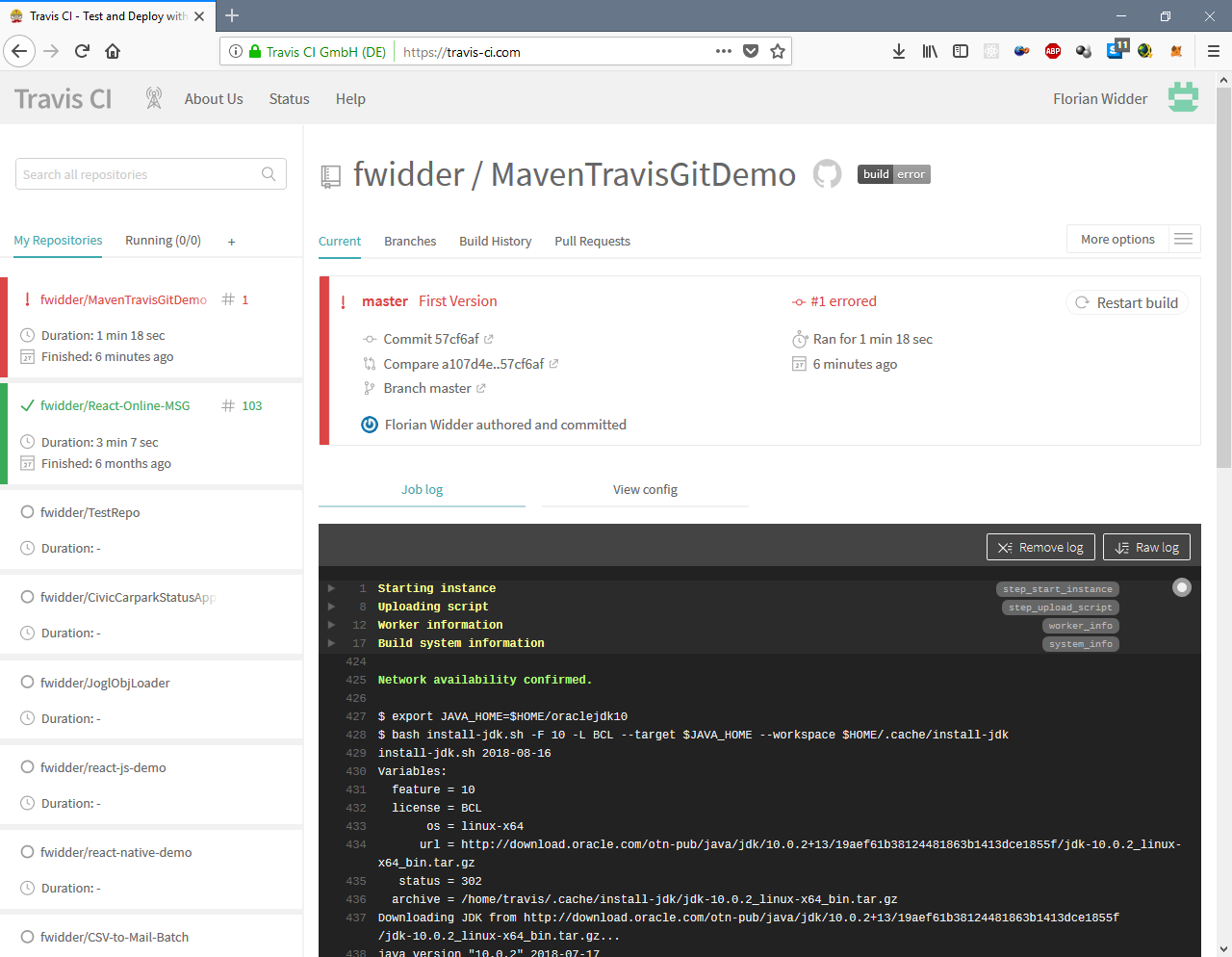


Figure 10: Failed TravisCI Build

If you take a closer look at the log on TravisCI (especially the end) you will notice that one of our tests failed (line 1914 in my log).

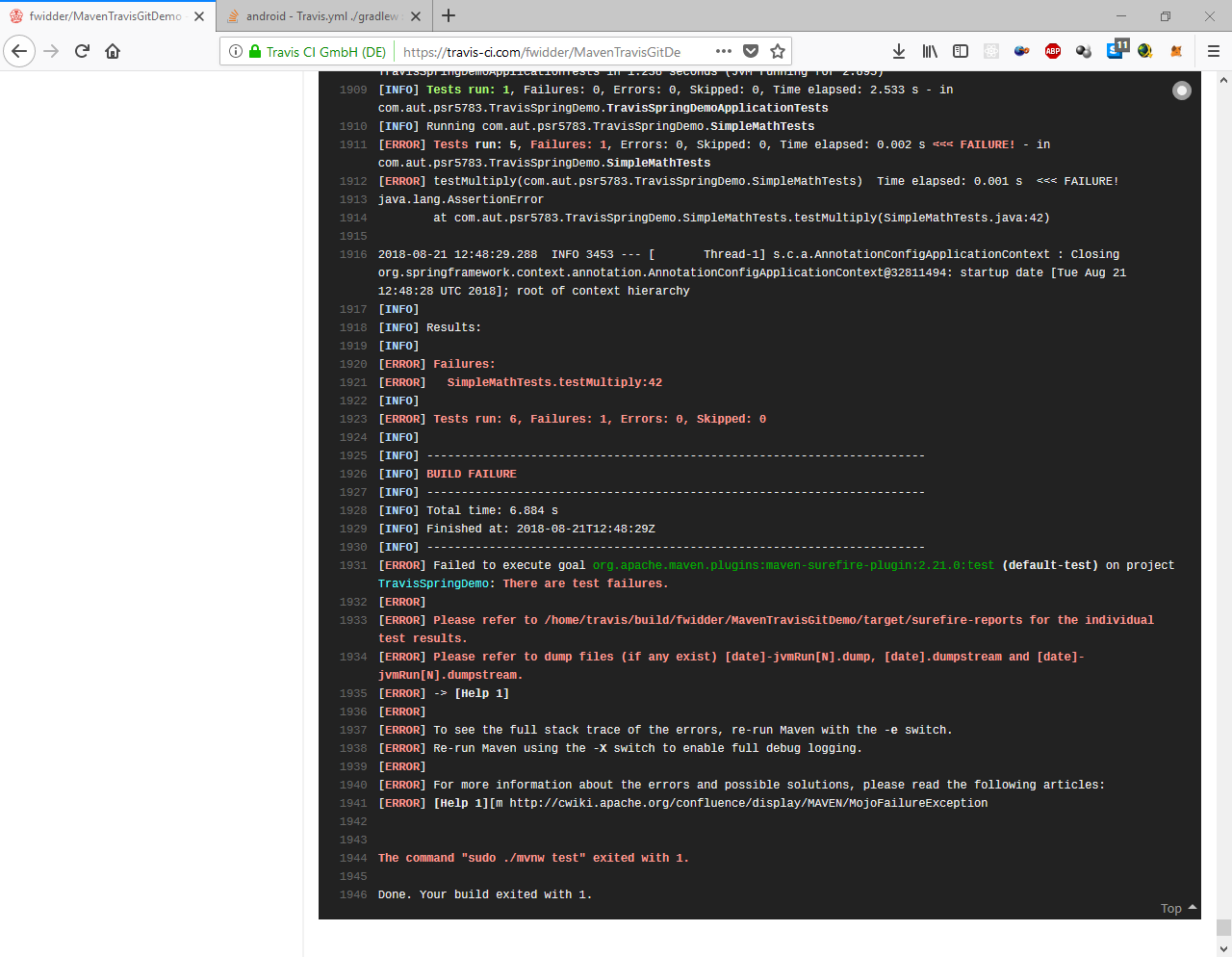


Figure 11: Test failure

If you look at the given line in the source code, you will see that 3 \* 12 = 36 and not 37. If you change this and commit and push your changes again, everything should go well this time.

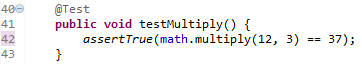


Figure 12: Bug in SimpleMathTests

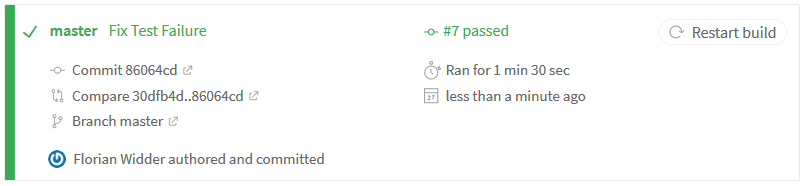


Figure 13: Successful build

# Reading

TravisCI Documentation: <https://docs.travis-ci.com/>

GitHub Apps: <https://github.com/marketplace>

Spring Boot: <https://spring.io/projects/spring-boot>

Maven: <https://maven.apache.org/guides/>

Junit: <https://junit.org/>