GDAŃSK UNIVERSITY OF TECHNOLOGY

Large-scale Enterprise Applications (LSEA)
Laboratory 1: Version Control System

In the LSEA laboratory, students are developing an application whose topic will be selected at the first laboratory. As part of subsequent tasks, the project will be supplemented with many features, which include e.g. multithreading, business logic of operations on the proposed data model classes, user interface, network services, database access mechanisms and more solutions showing the capabilities of the Java SE and Java EE platform. After each laboratory, please keep the developed project for use in the next task. For the purposes of the laboratories, groups of 3-5 people should be formed and cooperate.

First of all, the topic of the realized application should be chosen before the first laboratory. Please be aware that the topic of the application should be universal in that way that can be easily expanded, e.g., by database, should have possible multiple types of data and operations on them and… possible enterprise application at the end with multilayered construction (server-business logic-presentation logic). The relatively easy way to get the data in future laboratories should be also taken into consideration.

The selected topic cannot be repeated within students' group and should be presented (PDF document) and uploaded with all files in Laboratory 1 section in the eNauczanie platform, with all details (but no more than 1 A4 page)! Of course, it can be the project which can be used by students in the future work or can expand their specific interests, e.g. application for football match results analysis (this subject is already reserved for the teacher ☺).

If the topic of the application will not be uploaded into the eNauczanie platform it will result in 0 points for this exercise.

Be aware, that to realize the laboratory task, You should have *Maven* project. To test if You have properly configured *Maven* tool on Your computer You can use sample application that You should build and run. Sample application can be downloaded from Laboratory 1 section.

First of all, account in the GitLab of GUT (https://git.pg.edu.pl) should be created (You can also log in by the MojaPG profile). GitLab is an environment for managing the team cooperation and includes among others: version control repository, bug tracker, continuous integration (build and test application on the server), code review tools, project documentation, statistics etc. After the registration one user should create new project.

After that You need to determine name of the project, visibility level set to private and… that's simple as that – You have already created a project!

Student, who created project, should import to repository latest version of the application. It can be done from the command line or use the integrated tools in the IDE

(Integrated Development Environment). In the first case the instructions from the GitLab could be used (with installed git on Your computer):

```
cd existing_folder
git init
git remote add origin https:// git.pg.edu.pl/login/project_name.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

After completing the operations, refresh the GitLab website in Your web browser and make sure that all project files are in the repository! Afterwards, student who created new project should invite other team members. Go to Your project website, and from the left side menu choose *Settings* and *Members*. Then, login of the invited team member should be filled and *Maintainer* role should be chosen. After completing the invitation, other members should see the project on their list of projects and can modify them.

<Optional> – additional 1 point per laboratory that is done with the use of CI/CD

Next, team members that were invited to the project, should configure the continuous integration mechanism. To do this, choose *Set up CI/CD* button that is located on the project main page and enter the content of the configuration file *.gitlab-ci.yml* that You can download from the eNauczanie platform (in the laboratory 1 section) and confirm it by clicking *Commit changes*.

Please analyze that file in that stage (don't only copy paste it ☺). The provided configuration defines the continuous integration (CI) pipeline that consists of 4 stages. Each stage is run only if previous stage was successfully completed. Stages are defined in the section *stages*:

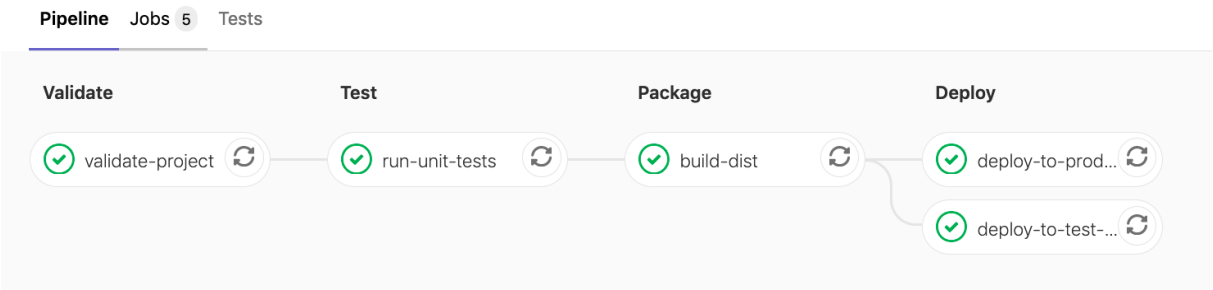
- validate – project validation, verification of syntax correctness, compilation of source files; if project does not compile correctly, the pipe will be interrupted in this stage;
- test – unit test runtime; if one of the tests will fail, pipeline will be interrupted in this stage;
- package – preparation of the distribution version of the software (JAR archive);
- deploy – implementation (deployment) application on the server; in that case we do not administrate such public server, so this feature is mocked for this stage.

Tasks to do in each CI pipeline are defined in the rest part of the file. The example of that configuration that was provided include 5 such task: validate-project, run-unit-tests, build-dist, deploy-to-test-server, deploy-to-production. Each task has a stage attribute specifying at what stage of the pipe it should be performed. In addition, tasks have script attributed that define commands to be invoked on the CI server. For a project built using Maven, the console

mvn command is used. You could also see that there is a part about unit testing, which will be used in the upcoming laboratories and will work in Your projects.

Artifacts attribute can allow to specify which effects of a given task should be saved, e.g. after completing the build-dist task, the JAR archive with the application is persisted. Artifacts are available when performing further tasks under the CI pipe and can be download from GitLab. In turn, the *only* attribute allows to specify for which branches of the repository data the task should be carried out. In the provided example, the *deploy-to-production* task is run only for the project located in the *master* repository branch. Section *variables* is defining the variables used in the further part of the configuration file.

After continuous integration mechanism configuration, on the left side menu go to *CI/CD* and *Pipelines* and choose the first option on the list to check details of CI pipeline. Pipe is run automatically after every change in the repository (commit/push). As the effect, each version of the project which goes to the repository is automatically tested, built and can be deployed on the server. After successfully completion of all defined stages, pipe should be represented similar as:



</Optional>

If project is correctly built in the server, current version of the *master* branch should be tagged as *v1.0*:

- from command line: `git tag`
- in Your IDE please check in documentation;
- in GitLab graphic interface: *Repository / Files / project_name / + / New tag*.

Team member that has been invited to the project should download repository to his workstation (local computer):

- from command line: `git clone`
- in Your IDE please check in documentation.

Project manager (member that created the project) should update his local copy:

- from command line: `git pull`
- in Your IDE please check in documentation.

In the next step, new branch *devel* should be created:

- from command line: `git tag`

- in Your IDE please check in documentation;
- in GitLab graphic interface: *Repository / Branches / New branch*.

Then, go to *Issues / List* and add issues that corresponds to new network functionalities that should be done as an application extension. To each task there should be one team member assigned (all team members need to be assigned to some development tasks) – field *Assignee*.

Every team member needs to create his own branch in repository based on the *devel* branch, in which the functionality will be developed. Please remember to change the branch in Your local workspace:

- from command line: `git checkout`
- in Your IDE please check in documentation.

All extensions should be committed to members separate branches. After all of the features done, all changes should be merged to *devel* branch. If new project version in *devel* branch will be successfully built on the CI server, merge it to *master* branch and tag the software as *v.1.1*.

Within the laboratory whole team's work is evaluated – every team member gets the same number of points:

- successfully configured continuous integration mechanism (1 points);
- correct assignment of team member to their tasks in *Issues* section (1 points);
- proper usage of git functionalities (1.5 points):
  - all branches should be visible;
  - every commit to repository should be described with proper comment;
- report (1.5 points).

Please prepare the report (as a PDF), whereas a prove of correct configurations please attach the proper screenshots and commands that You were using and Your comments to them. At the end add me as a team member (@p232812 or search me by name), so everything can be verified.

Please also use comments for a description of the classes/fields/methods (and their parameters). Code without the proper (can be basic) comments will not be accepted ☹ (they need to fulfill the javadoc specification). Also remember to highlight the code sections that in Your opinion are connected with the realization of the application requirement. Don't forget to upload project and report into the eNauczanie platform in proper section.