Second Iteration Demo Submission

GroupUs

Rui Ge rg3105

Ruimin Zhao rz2390

Hao Fang hf2345

Shenxiu Wu sw3196

December 4, 2018

1 Section One

- Finished Time: We demonstrated the changes we made compared to the previous iteration primarily following our test cases listed in the submitted report for second iteration assignment. The demonstration was completed on Monday evening at around 7:45 pm.
- Demo: We first demonstrated the pre-commit process. And for the test cases demonstration, we found it could be quite repetitive if we exactly follow the test cases and equivalent classes we wrote about in the report. As a result, we suggested that we demonstrate the sign up user case in detail as an example. This test case cover almost all the input check criteria we added for this second iteration. They are: input length should be within a certain range; input should only consist of letters and digits without any other non-ASCII characters or special characters; input should start with letters rather digits; input should have certain prefix or suffix. These input check code (regular expression) is used in multiple places in different use cases (especially for sign up and post where quite a lot user inputs are happening).

2 Section Two

Changes we made are mainly some added check criteria for user inputs happened in Sign up and Post. These inputs include user email, event subject etc. For every user inputs, we added more criteria like length check, special character check, and prefix/suffix check (like columbia email pattern). The test cases are listed below:

2.1 Sign in/up

2.1.1 Sign up

- Note: step 2 to 4 could change the order, but here we assume the order to be 2, 3, 4 for simplification. The steps that could contain wrong actions are step 2, 3, and 4. Theoretically there could be 8 possible alternate flows, but there we list 3 symbolic ones where each has one wrong step.
- Title: Sign up an account

- Actor: Unregistered User
- Description: Part of this group up system. The user registers an account with a unique Columbia email address as the username and a password with valid content.
- Precondition: The program is available on user's local laptop.
- Triggers: The user execute the program and get sign up/in window.
- Basic Flow:
  - 1. User starts the system and the system prompts out a window for the user to enter registration information.
  - 2. User enter the email address with correct format (with the suffix being "@columbia.edu", only consisting of numbers and normal letter characters, and with prefix pattern being either "2 letters + 4 digits" or "3 letters + 4 digits") which has never been used to register before.
  - 3. User enter the password with correct format (with the input name starting with letter (character), and with length no more than 20).
  - 4. User enter the name with correct format (with the input password starting with letter (character), and with the length between 6 to 18, and only consisting of characters/numbers/underlines).
  - 5. User click on the "Sign Up" button on the window, and complete the registration.
- Alternate Flow 1:
  - Note: Step 2 has error, the other steps are the same as basic flow (except the result for final step).
  - 2. User enter the email address with wrong format (with the suffix being "@gmail.edu"/with special characters other than letters and digits/with prefix pattern being neither "2 letters + 4 digits" nor "3 letters + 4 digits"), or which has already been used to register before.
  - 5. User click on "Sign up", then the registration fails and user is re-directed to step 1 user interface.
- Alternate Flow 2:
  - Note: Step 3 has error, the other steps are the same as basic flow (except the result for final step).

  - 3. User enter the password with wrong format (with the input name starting with number/with length more than 20).
  - 5. User click on "Sign up", then the registration fails and user is re-directed to step 1 user interface.

- Alternate Flow 3:
  - Note: Step 4 has error, the other steps are the same as basic flow (except the result for final step).
  - 4. User enter the name with wrong format (with the input password starting with number/with the length out of 6 to 18 range/containing special characters).
  - 5. User click on "Sign up", then the registration fails with an alert window prompted out, and the user is re-directed to step 1 user interface.

2.1.2 Sign in

- Title: Sign in to an account
- Actor: A registered user who has already signed up.
- Description: Part of this group up system. The user who has already registered an account logs in to the account with correct combination of username and password.
- Basic Flow:
  - 1. User starts the system and the system prompts out a window for the user to enter registration information.
  - 2. User enter the email and password he/she used for registration.
  - 3. User click on the "Sign In" button on the window, and log in.
- Alternate Flow:
  - Note: Step 2 has error, the other steps are the same as basic flow (except the result for final step).
  - 2. User enters wrong combination of password and email which do not fit any account stored in database/any wrong format input itself.
  - 3. User click on the "Sign In" button on the window, then the sign in fails with an alert window prompted out, and the user is re-directed to step 1 user interface.

2.2 Post

- Title: Post an event
- Actor: A registered user who has already logged in
- Description: A user who wants to post an event, he/she at the post screen with specific properties including category, subject, time, location, memo and description.
- Basic Flow: The user is currently in the post screen now.
  - 1. The user presses the "Post" button at the status screen and the system will be redirected to the post screen for posting events.
  - 2. The user clicks on the drop down box to select the event category among "study", "eat out" and "go home". The default category setting is "study".
  - 3. The user enters a subject of event with the correct format(length between 1 to 20)
  - 4. The user enters a valid location for the event(length between 1 to 200 and can pass the check of external geology application)
  - 5. The user clicks on the drop down box to choose proper start date and end date of the event.There are two drop down boxes for each. Select a certain day and specify the accuracy of time to one minute.
  - 6. The user enters content into the field of "memo" and "description" with correct format(length between 1 to 200)
- Alternate Flow1: The user comes to the post page. Because the default category is "study", this field will not cause any conflict even the user don't select it.
  - 1. After choosing any category among those three categories, he or she type a subject which exceed 20 bytes, which obviously is not a correct format in our standard. Another condition is that user input some special character will are not allowed. This will trigger a same alert as over-long length. Thus, here, we combine the 2 situations to have a discussion.
  - 2. The user enters a valid location for the event, here we assume this location can pass the check of external geology API.

  - 3. The user clicks on the drop down box to choose proper start date and end date of the event.There are two drop down boxes for each. Select a certain day and specify the accuracy of time to one minute.
  - 4. The user enters content into the field of “memo” and “description” with correct format(length between 1 to 200)
  - 5. Once the user click the “post” button, the system will check the input information and finally find out the subject is invalid. The system will pop out an alert window indicating the user some error information and tell them how the subject input violated our rules and how to fix it. Finally, once the user click the “OK” button on the alert window, our system will return the user to step 1.

- Alternate Flow2:
  - 1. Similar with the assumption of alternate flow 1, but this time, the user input a invalid location while other required information is valid.
  - 2. System will check the inputs and call the geology API to check and later find out the location input is invalid. The system will pop out an alert window indicating the user that the location input should be revised and then return the user to alternate flow step 1.

- Alternate Flow3:
  - 1. Similar with the assumption of alternate flow 1, but this time, the user input a invalid description and/or memo while other required information is valid.
  - 2. Once the user click the “post” button, the system will check the input information and finally find out the description and/or memo is invalid. The system will pop out an alert window indicating the user some error information and tell them how the input violated our rules and how to fix them. Finally, once the user click the “OK” button on the alert window, our system will return the user to alternate flow step 1.

- Alternate Flow3:
  - 1. Quite Similar with the analysis of aforementioned flow 1, but this time, the user input both invalid location and description memo while other required information is valid.

  - 2. Once the user click the "post" button, the system will follow the following order to check the input information and return some error info to ask user to revise through alter window if exists. The check order is:

$$\begin{aligned} &\textit{subject text} \Rightarrow \textit{location text} \Rightarrow \textit{location validation} \\ &\qquad \Rightarrow \textit{memo text} \Rightarrow \textit{description text}. \end{aligned} \tag{1}$$

    Once any one or some check failed, the system will pop out an alert window indicating the user what error happened and tell them how the input violated our rules and how to fix them. Finally, once the user click the "OK" button on the alert window, our system will return the user to alternate flow step 1.

2.3 Search

- Title: category based search
- Actor: A user who wants to see the events appealing to him/her.
- Description: A user who wants to see the existent events, he/she signed up already.
- Basic Flow: The user is in the search screen now. He/She needs to select search by category.
  - 1. The user selects one category in the all three possible categories: Study , Eat Out, Go Home.
  - 2. The user also enters his/her current location, then clicks the search button.
  - 3. The system will redirected to a new screen showing the events in that category. The results will be in the order of the distance from user's current location to the events' locations.
  - 4. The user could see other critical information like start time, end time about this event and make his/her favourite choice.
  - 5. The expired events will not be displayed.
- Alternate Flow1:

  - Note: 1.In step 2, if the user doesn't enter his/her current location, there will be no error.
  - 2.The system will still redirect to a new screen. The user can still see the events in one specific category.
  - 3.The other information about this event is still available, but the events are not in the order of the distance.

- Alternate Flow2:
  - Note: 1.In the step 2, if the entered location is invalid, there will be no result.
  - 2.The system won't redirect to a new screen, instead a error message alert will be shown to the user. The user will know the entered location is invalid and he needs to enter a valid location or just skip this blank.
  - 3.The user can't see the events in the corresponding category absolutely.

2.4 Join

- Title: Join a event
- Actor: A user who wants to join an event
- Description: The user wants to join a group study event, he/she signed up already and never used our desktop app before.
- Basic Flow: The user could see the searched results now.
  - 1. The user would like to select a suitable event to join. He/She could just select a event and click the join button.
  - 2. System will redirect to the user's profile page after joining into the event. The user could manage other events he posted or joined here.
- Alternate Flow1:
  - Note: 1.In step 1, if the user is reviewing a joined event and wrongly click the join button.
  - 2.The system will not redirect to a new screen. Instead, an error message will be shown to the user, which is "you joined this event already!".
- Alternate Flow2:

- Note: 1.In step 1, if the event is posted by the user, the similar error will be shown.
- 2.when the user searched the events by category, he/she can also see the events posted by himself/herself.
- 3.If the user wrongly click join button to his/her posted event. The same error occurred, because when someone posted a event he/she is assumed to join this event.

3 Section Three

When we first time came to the CI part, the first challenge we need to conquer is to figure out how to connect our code version to the CI part. Because each of us four people is responsible for different part of our project, which means each of us will develop brand-new and different code and how to merge them into one final version and how to check the combined version would work is the first challenge we need to handle.

The second challenge we need to handle is how to run our CI tools appropriately. Do we need to set the Jenkins server on cloud or can we set it on our local machine? What's more, because there is a pre-commit step we need to consider, so do all of the teammates need to set up Jenkins in their own machines?

Actually, we originally planned to use Azure where we can build a pipeline to manage the whole process of building and testing our project. But finally we drop this scheme due to some core functions on Azure would be charged and some weird incompatible error happened between javaFx and Azure. We finally came up with Jenkins whose server running on one team member's local machine for the CI part. Jenkins is an open source automation server written in Java. It helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat. One **technology** we use is **Git** which is a version-control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source-code management in software development, but it can be used to keep track of changes in any set of files.

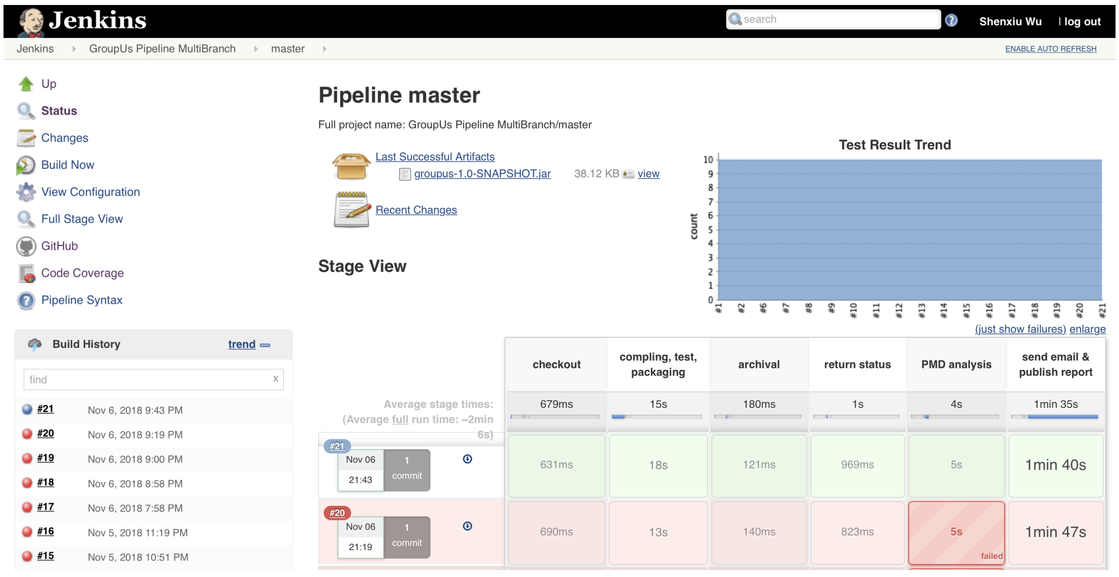
Our **solution** for the aforementioned difficulties is that each of us individually creates one branch from the master, and the master will adopt a protected policy which means it will refuse force pushing, prevent branches from being deleted, and require status checks before merging. Thus, if we want to merge our new version code to the master, we have to first push this new version to his or her own branch, and the push action will trigger the Jenkins(running on one team member's local machine) to build, compile, test and package. If all stages finished successfully, the Jenkins will return a success status to GitHub and add the return status to the newest commit in the updated branch. Finally, if we set a new pull request on GitHub to merge the new commit from our own branch to the master, only the commit with successful check status can pass through, otherwise, the master will refuse the merge request. After successfully merge, the master will trigger Jenkins to run again(because there is an update on master) to double check the newest merged version on master works well.

In Jenkins, we used the Multibranch Pipeline, which creates a set of Pipeline projects according to detected branches in one SCM repository. Due to this, any update on our own feature branches and master will be detected by Jenkins which will trigger the preset CI process. Another **technology** we used is that we put our Pipeline script which is written in Groovy into version control so we can version it like every other file that's a part of our project. One of the benefits we get by doing this is we give back control to the teams that are developing software to conveniently define what this Pipeline looks like. We give each team member the **autonomy** to do what they need to do to deploy their newer code version. This file gives us highly freedom to re-define the CI process or any stage with our project's needs going on. Also, every member in our team can see who made changes, when they made the changes and why they made the changes, etc. It's transparent and everyone can even correlate the changes to changes in their own application code.

Another **technology** we adopt is taking a good use of graphical interface in CI by setting different stages to quickly find out which stage meets error and easily check its log. As you can see in figure 1, you can easily understand the CI process and figure out which stage the Jenkins server is running on. Briefly speaking, our logic in CI is firstly using Git to check whether there is a new commit or any update on a specific branch, if there is, one node will be triggered to assign some work-space and create a new pipeline to run the pre-defined execution flow in our Jenkinsfile. Our first three stages are using maven to compile, test, package, archive, test and verify, etc. These are some basic steps we need to do in a CI flow. Then, in the 4th stage, if we verify(including all previous steps) our new version code successfully and all Junit tests are passed, we will return a success status to Github and Github will indicate this returned status in the newest commit, meanwhile a code coverage report will also be published from the Junit test result. Then Jenkins will run PMD and publish the PMD report. Thus, the final stage is to publish and upload the generated reports to our Dropbox where every one can access and check it. If all thing have done, Jenkins will send an email to us which notify the final status of specific build job. In figure 2, the execution logs of each stage can be easily checked. From this figure, we could know that the build 21 in compiling stage is successful.

What's more, like we said before, due to the project in Jenkins is a MultiBranch one, thus, just as shown in figure 3, we could easily manage and control every separate feature branched and check what's their build status and what error happened in their own branch.

In a nutshell, our CI part is tremendous flexible and can greatly meets the needs and requirements of CI in our software develop process.



Pipeline master

Full project name: GroupUs Pipeline MultiBranch/master

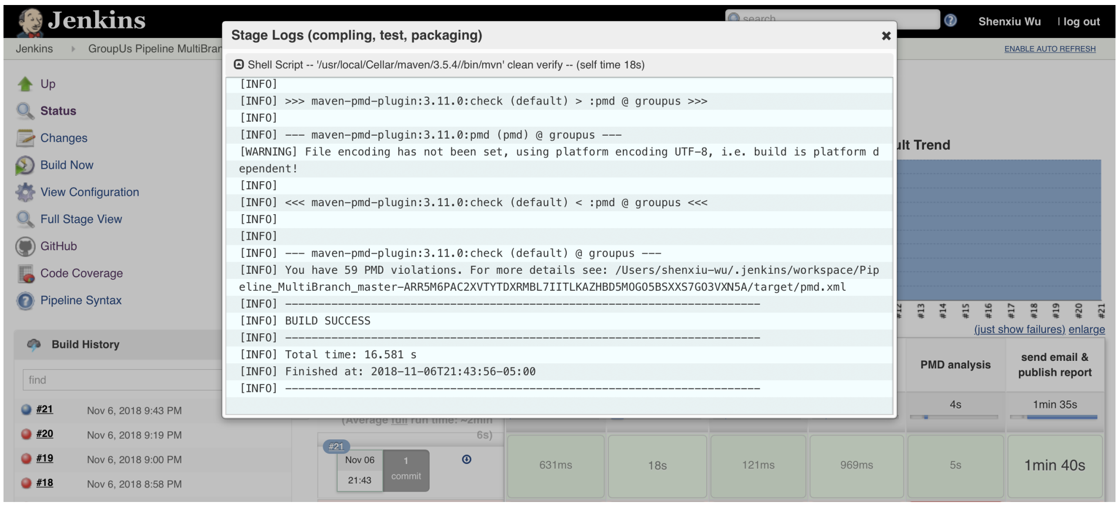
groupus-1.0-SNAPSHOT.jar 38.12 KB

Stage View

	checkout	compling, test, packaging	archival	return status	PMD analysis	send email & publish report
Average stage times: (Average full run time: ~2min 6s)	679ms	15s	180ms	1s	4s	1min 35s
#21 Nov 06 21:43 1 commit	631ms	18s	121ms	969ms	5s	1min 40s
#20 Nov 06 21:19 1 commit	690ms	13s	140ms	823ms	5s failed	1min 47s

#21	Nov 6, 2018 9:43 PM
#20	Nov 6, 2018 9:19 PM
#19	Nov 6, 2018 9:00 PM
#18	Nov 6, 2018 8:58 PM
#17	Nov 6, 2018 7:58 PM
#16	Nov 5, 2018 11:19 PM
#15	Nov 5, 2018 10:51 PM

Figure 1: The CI work flow



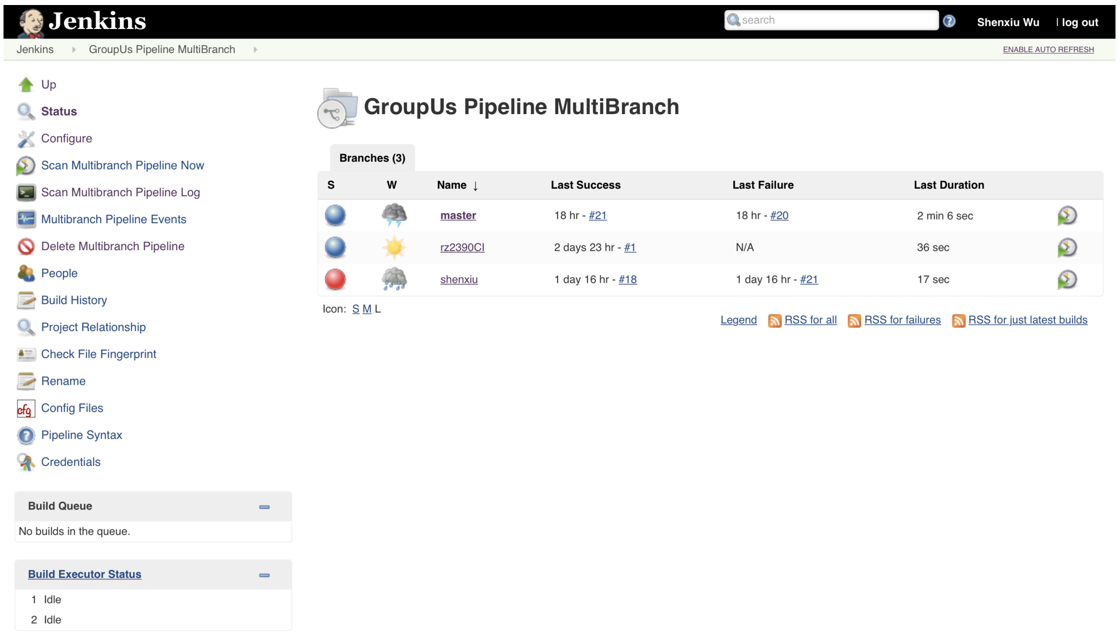
Stage Logs (compling, test, packaging)

Shell Script -- '/usr/local/Cellar/maven/3.5.4//bin/mvn' clean verify -- (self time 18s)

```
[INFO]
[INFO] >>> maven-pmd-plugin:3.11.0:check (default) > :pmd @ groupus >>>
[INFO]
[INFO] --- maven-pmd-plugin:3.11.0:pmd (pmd) @ groupus ---
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform d
ependent!
[INFO]
[INFO] <<< maven-pmd-plugin:3.11.0:check (default) < :pmd @ groupus <<<
[INFO]
[INFO]
[INFO] --- maven-pmd-plugin:3.11.0:check (default) @ groupus ---
[INFO] You have 59 PMD violations. For more details see: /Users/shenxiu-wu/.jenkins/workspace/Pip
eline_MultiBranch_master-ARR5M6PAC2XVTYTDXRMBL7IITLKAZHBD5MOGO5BSXXS7GO3VXN5A/target/pmd.xml
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 16.581 s
[INFO] Finished at: 2018-11-06T21:43:56-05:00
[INFO] ------------------------------------------------------------------------
```

#21	Nov 6, 2018 9:43 PM	
#20	Nov 6, 2018 9:19 PM	
#19	Nov 6, 2018 9:00 PM	
#18	Nov 6, 2018 8:58 PM	

Figure 2: Stage logs



GroupUs Pipeline MultiBranch

Branches (3)

S	W	Name ↓	Last Success	Last Failure	Last Duration
●		master	18 hr - #21	18 hr - #20	2 min 6 sec
●		rz2390CI	2 days 23 hr - #1	N/A	36 sec
🔴		shenxiu	1 day 16 hr - #18	1 day 16 hr - #21	17 sec

Figure 3: MultiBranch graphical interface

4 Section Four

- `https://github.com/sw3196/GroupUs.git`