First Iteration Demo Submission

GroupUs

Rui Ge rg3105

Ruimin Zhao rz2390

Hao Fang hf2345

Shenxiu Wu sw3196

November 8, 2018

1 Question One: Challenges

- Completion time: November 5, 2018
- Challenges:
  - Java:
    * Maven: We are all new to java programming. At the beginning, it's hard for us to import some third party packages like mongodb without loading the java driver file from local. Every time we load some java driver file in our local laptop, our teammate needs to do the same thing. Too much time are wasted, and some wreid bugs always occur. To solve this, it's necessary to learn something about independence system, in this case, we use maven to add the libraries we need.
    * MongoDB: If java application wants to connect the mongodb, the java driver for mongodb should be added. The latest version is java-driver-mongodb 3.4.1. To be honest, the design is so bad for java driver after 3.0. Some methods defined in the library is really hard to use and is totally different from java driver 2.0. It takes a lot of time to do the basic database insert, search or update operations.
    * MVC design pattern: At the beginning, we split the back-end code into different layers. All the code are put in the same package, there are so many duplicate code works in the same way. Also, it's really hard to debug. Then when the file gets bigger and bigger, we have to try something else. We dive into the MVC design pattern and split the back-end code into two layers: service layer and data layer. Several interfaces, factory classes, value object classes are provided, so that the code becomes more readable and clear. Some of them could be moved to another project directly. It's the first time we know the advantages of this design pattern.
  - CI: We planned to use Azure CI tool initially, but we found it could not build code related to JavaFx packages. We could not find a proper way to solve that problem and did not want to spend money to upgrade our account. Therefore, we decided to explore with other tools and we chose Jenkins because of its flexibility to use - we could configure our stages of CI in a more customized manner.

- Git Version Control: Every time, we push our own code into the github. We need to reslove so many conflicts problems. We guys are uncomfortable with this. If we didn't pull or push the files well, we have to move back to the last version. To solve this problem, we add a .ignore plugin into our project. This plugin helps us ignore some unimportant conflicts in unimportant files, saving a lot of time for us.

2 Question Two: Demonstration

- Demonstrated user stories: We demonstrated all user stories and conditions of satisfaction listed in our proposal. And we added some conditions of satisfaction for all these features as well.

- Changes made compared to last week's weekly meeting: We changed the alert window behavior and added the location validation check based on TA's suggestion. We also added many detailed check conditions such as the invalid time range for post feature etc.

  - **Login**: As a Columbia University student, I want to use this group-up application so that I can find some after-class group events among Columbia students. My conditions of satisfaction are (1) If I use the email address correctly ending with a suffix of @*columbia.edu* for registration, I should be able to sign up properly. (2) If I use email address with wrong format, I will get an error message and be reminded to use my @*columbia.edu* email. (3) If I use Wrong combination of user ID and password, I will get an error when sign in.

  - **Search**: As a user who has logged in successfully, I want to have the place to search for events so that I can either join a group event or post one if I do not find one fits my need. My conditions of satisfaction is that (1) I could search for events by categories (study, eat, go home) and see their basic information (start and end time, event description). (2) The displayed events should be listed in the order of distance to the user's current location. (3) If I entered an invalid location, my search request will be rejected.

  - **Join**: As a user who has logged in successfully and has searched out a list of available events, I want to be able to join the event by clicking on the JOIN button. My condition of satisfaction are (1) I could see this

joined events listed in my profile. (2) User cannot join an event which has already been joined before. (3) User do not need to join the event that is posted by themselves.

  - **Post**: As a user who doesn't find suitable events, I want to post a new event so that I can find someone else shares the common interest with me. My conditions of satisfactions are (1) I could have a profile to save the events I create. (2) I could specify the start and end time of the event, and give detailed description, location of the event. (3) If I entered an invalid location, my post request will be rejected. (4) If I enter start time earlier than current time, my post will be rejected. (5) If I enter end time earlier than start time, my post will be rejected.

- Changes:
  - **We added location validation check and some other edge cases check:** In both search and post feature, we make it necessary for user to enter valid location for their search and post request. Also, we added more time setting check for post. And we added join edge cases test.
  - **We changed our plan of using external API**: We planned to store both geolocation information and plain text information for counting the distance for each event initially. However, when made more exploration in using Google API, we found a distanceMatrix API which could accept plain text location as inputs and computes the distance result directly without the step of converting plain test location into gelocation.
  - **We used NoSQL database rather than relational database**: AWS RDS service was our initial choice, but when we tried to learn how to use it, we found that this service is costly. Then we tried to look for other database services and found that mLab provides the cloud-hosted database service and we do not need to spend money on it.

3 Question Three: CI

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

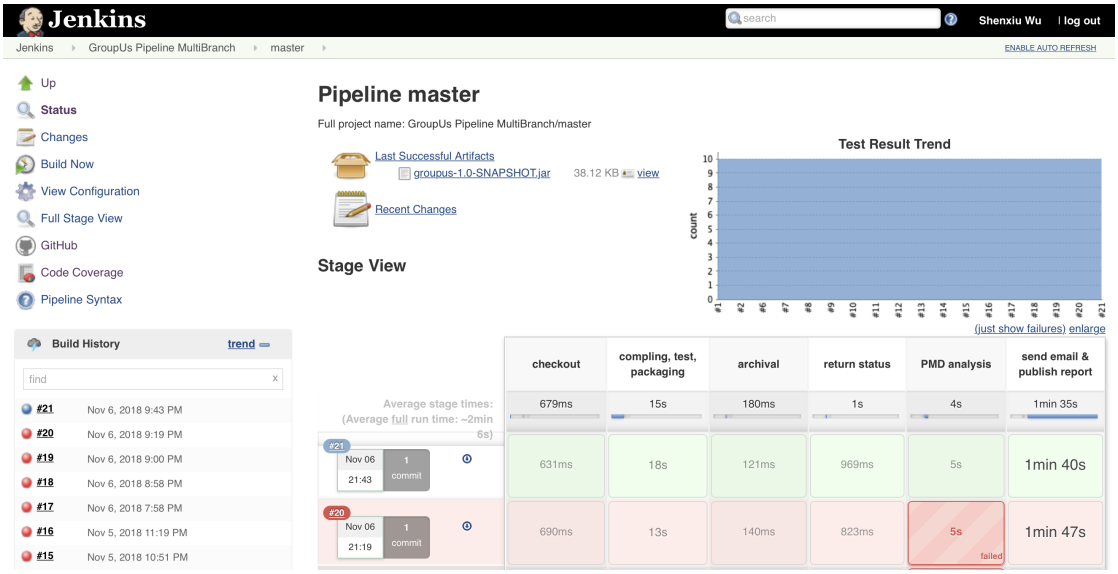

Figure 1: The CI work flow

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.

4 Question Four: Repo

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

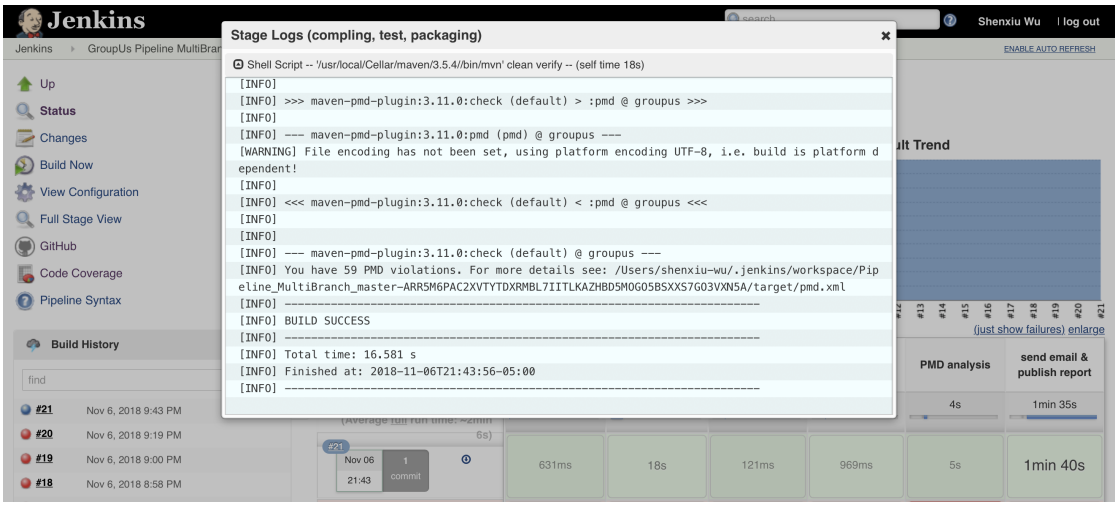

Figure 2: Stage logs

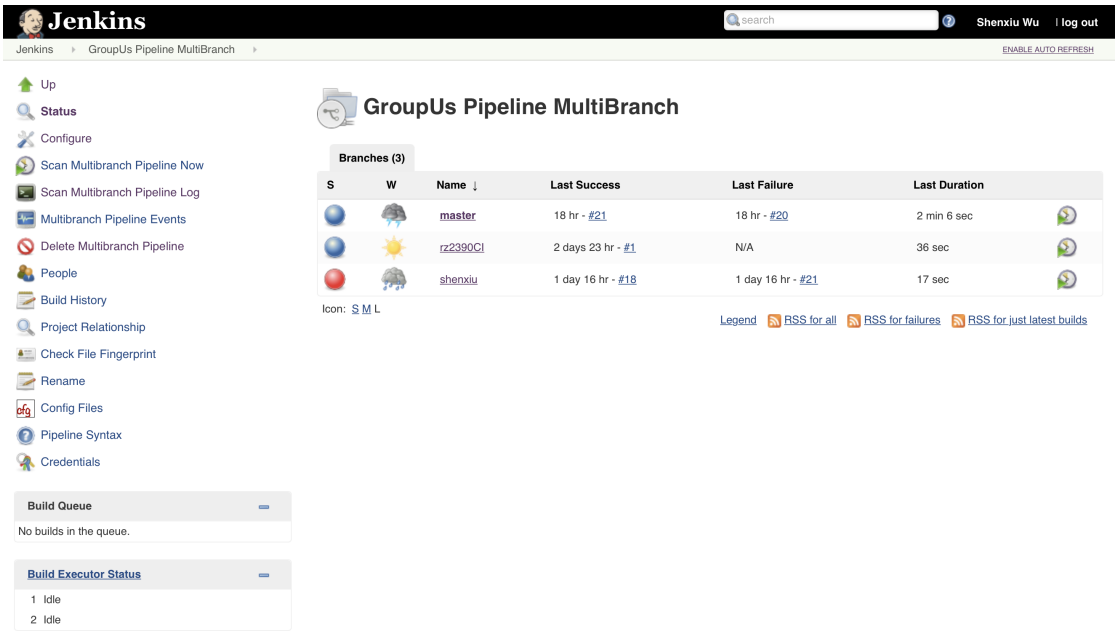

Figure 3: MultiBranch graphical interface