Assignment T5: Second Iteration

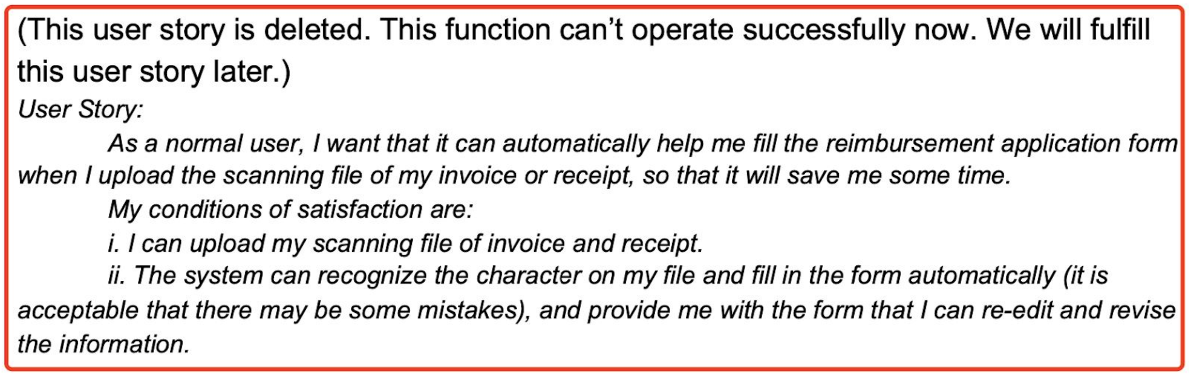
Team Iceberg

Chong Hu (ch3467)
Weijie Huang (wh2447)
Zeyang Chen (zc2483)
Wenjie Chen (wc2685)

Part 1:

Revised User Stories:

In first iteration's report, we have a user story that did not successfully implement:



(This user story is deleted. This function can't operate successfully now. We will fulfill this user story later.)

User Story:

As a normal user, I want that it can automatically help me fill the reimbursement application form when I upload the scanning file of my invoice or receipt, so that it will save me some time.

My conditions of satisfaction are:

i. I can upload my scanning file of invoice and receipt.

ii. The system can recognize the character on my file and fill in the form automatically (it is acceptable that there may be some mistakes), and provide me with the form that I can re-edit and revise the information.

In this iteration, this user story's function has been added successfully.

In general, we add:

1. Image upload and storage service;
2. OCR service which can extract information from bill/invoice image;
3. "Quick add" function which fulfills the reimbursement form for users according to the image information;
4. Github auto tools: CI, CodeQL, CheckStyle, CodeCov...

Based on the suggestions in the last demo, we update:

1. Turning the rei-request type to "ienumerable" type (such as Hotel, Transport, Life and Catering), and make a drop-down menu;
2. Group manager or administrator can make "comments" for the reimbursement request, and users can view the comments.
3. Content Check when submitting form;
4. Normal users now are able to change his password in the "Users Center".

Part 2:
Since we have nearly 100 methods to test, we try to explain our test plan by dividing our projects into six parts based on the packages: *controller, entity, external api, service, utils, configs*, and discuss equivalence partitions and boundary conditions based on classes.

***Controller*:**

1. ***UserInfo controller:***

   Major function 1: toLogin()

   - Description: login handler
   - Test plan: Because this method is mainly focused on handling the request to the login page, and we can divide into two scenarios that one is with session information and one is without session information.
   - Equivalence partition:
     Requests which have the session information of the user will be redirected to the welcome page.
     Requests which have no session information will go to the login page.
   - Valid input test case: testDirectToIndexPage()
   - Invalid input test case: testToLoginPage()

   Major function 2: getUserInfo()

   - Description: mainly used for login in the system, and check password.
   - Test plan: divide into two classes that one is with correct username and password, and one is without.
   - Equivalence partition:
     Requests with the right username and password will login successfully.

Requests with the wrong username and password will login failed.

- Valid input test case: testLogin()
- Invalid input test case: testWronglyLogin()

Major function 3: getUsersByWhere()

- Description: get users' information from the database.
- Test plan: Because we have set different levels of users, such as group manager, normal user and administrator. Therefore, the group manager cannot view administrator's information but can have access to normal user's information.
- Equivalence partition:
  Requests from the group manager can get all users' information in their group.
  Requests from the group manager can get none information from the administrator in the system.
- Valid input test case: testGetUserByWhere()
- Invalid input test case: testGetUserByWhereByGroupManager()

Major function 4: addUser()

- Description: add an user into the database.
- Test Plan: add an none-existed user and add an existed user.
- Equivalence partition:
  Requests with non-existent users will be successful, while requests with existing users will be failed.
- Valid input test case: testUserAdd().
- Invalid input test case: testUserAddFailed().

Major function 5: updateUser()

- Description: update an user's information in the database.
- Test Plan: update an existing user and update an none-existed user.
- Equivalence partition:
  Requests with existing users will be successful, while requests with non-existing users will be failed.

- Valid input test case: testUserUpdate().
- Invalid input test case: testUserUpdateFailed().

Major function 6: updateUser()

- Description: update an user's information in the database.
- Test Plan: update an existing user and update an none-existed user.
- Equivalence partition:
  Requests with existing users will be successful, while requests with non-existing users will be failed.
- Valid input test case: testUserUpdate().
- Invalid input test case: testUserUpdateFailed().

Major function 7: deleteUser()

- Description: delete user.
- Test Plan: delete an existing user and delete an none-existed user.
- Equivalence partition:
  Requests with existing users will be successful, while requests with non-existing users will be failed.
- Valid input test case: testUserDel().
- Invalid input test case: testUserDelFailed().

Major function 8: logout()

- Description: logout from the system.
- Test Plan: check the session and the redirected page after logout.
- Test case: testLogOut().

Major function 9: getAllRoles()

- Description: get all roles information from the database.
- Test Plan: test whether we can get all roles from the database.
- Test case: testGetAllRoles().

Major function 10: addRole()

- Description: add a role in the database.

- Test Plan: add an existing role and add a none-existed role.
- Equivalence partition:
  Requests with existing roles will be unsuccessful, while requests with non-existing roles will be successful.
- Valid input test case: testRoleAdd().
- Invalid input test case: testRoleAddFailed().

Major function 11: updateRole()

- Description: update a role information in the database.
- Test Plan: update an existing role and update a none-existed role.
- Equivalence partition:
  Requests with existing roles will be successful, while requests with non-existing roles will be failed.
- Valid input test case: testRoleUpdate().
- Invalid input test case: testRoleUpdateFailed().

Major function 12: deleteRole()

- Description: delete the role information from the database.
- Test Plan: delete an existing role and update a none-existed role.
- Equivalence partition:
  Requests with existing roles will be successful, while requests with non-existing roles will be failed.
- Valid input test case: testRoleDel().
- Invalid input test case: testRoleDelFailed().

Major function 13: getRoleById()

- Description: get role information by id.
- Test Plan: test getting an existing role and an non-existing role.
- Equivalence partition: the existing role will be returned and the non-existing role will not be shown.
- Valid input test case: testGetRoleById().
- Invalid input test case: testGetRoleByIdFailed().

2. ***ReiRequest controller:***

Major function 1: add()

- Description: Add a new reimbursement request into the database.
- Test Plan: When adding a new reimbursement request, there are many inputs like title,id, userid, money and so on. For the data type restriction, it’s done when setting those parameters. You can’t set a string to user id. For other limitations, money input should be 0 to 1000. Payway id should be 0 to 5. When payway id is illegal, we will set it to default values 1. Therefore, besides valid input test cases, some invalid inputs including illegal paywayid, illegal money amount or missing values will be given to test this function.
- Corresponding test cases: com.iceberg.controller.ReiRequestControllerTest.shouldAddReiRequest() & shouldAddReiForIllegalPayway() & shouldAddReiForMissingPayway() & shouldNotAddReiForErrorServiceCode() & shouldnotAddReiForServiceException

Major function 2: update()

- Description: update an existing reimbursement.
- Test Plan: Similar to what has been talked about in add function’s test plan. Those limitations also apply to update functions. What’s more, the reimbursement to be updated must exist in the database. If it can not be found, the update operation should fail. Also, a normal user has no right to update other normal user’s requests. Finally, if the request has been approved by the manager, it can not be changed anymore. We will test above invalid inputs.
- Corresponding test cases:

com.iceberg.controller.ReiRequestControllerTest.shouldUpdateReiRequest() & shouldNotUpdateReiForNoRequestExist() & shouldNotUpdateReiForOtherNormalUser() & shouldNotUpdateReiForApprovedType()

Major function 3: delete()

- Description: Delete an existing reimbursement by id.
- Test Plan: The reimbursement to be deleted is found by id. Therefore for this test, we are trying to delete a reimbursement that does exist and a reimbursement that does not exist.
- Corresponding test cases: com.iceberg.controller.ReiRequestControllerTest.shouldDelReiRequest() & shouldNotDelReiForErrorServiceCode() & shouldNotDelReiRequestForDelException()

Major function 4: review()

- Description: Group manager can review and change the status of his group members' reimbursement requests.
- Test Plan: First of all, a group manager cannot review a request of other group members' because he has no permission. Then, a request cannot be approved if it has no money information. But it doesn't matter if it has no receiver's email or account. We will test all above situations and try to change a valid request from PROCESSING type to APPROVED, DENIED and MISSINGINFO.
- Corresponding test cases: com.iceberg.controller.ReiRequestControllerTest.shouldApproveReview() & shouldNotApproveReviewForPermissionDenied() & shouldApproveReviewForMissingEmail() & shouldApproveReviewForMissingReceiver() & shouldNotApproveReviewForMissingMoney() & shouldNotApproveReviewForProcessingType() & shouldNotApproveReviewForDeniedType() & shouldNotApproveReviewForMissingInfoType()

Major function 5: getReiRequestById() & getReiRequestByUserId()

- Description: Find the reimbursement requests by their ID or by users' ID.
- Test Plan: The input id must exist in the database, otherwise the operation will fail.
- Corresponding test cases: com.iceberg.controller.ReiRequestControllerTest.shouldGetReiRequestById() & shouldGetReiRequestByUserId() & shouldGetReiRequestForNormalUser() & shouldGetReiRequestForGroupManager() & shouldGetReiRequestForAdmin()

Major function 6: getImage

- Description: return image to front-end & support functions
- Equivalence partition: For getImageByImageId, the valid input should be imageId which is a string. The valid equivalence partition is all string with existent image id, while the invalid equivalence partition is all other types and string with non-existent image id. For getImageByRequestId, this function will extract the image id from the corresponding request id. Then the function will call getImageByImageId. Thus the valid equivalence partition is all string with existent request id. Even requests that don't have image value fulfilled can be input in the function and the function will return a blank image (null). The invalid equivalence partition is all other types and strings with non-existent request id.
- Test Plan: Our test cases will test both circumstances above--check if given valid input, corresponding response is returned; if given invalid input, corresponding exception is returned.
- Corresponding test cases: com.iceberg.controller.ReiReqeustControllerTest.shouldGetImageByRequestId() & shouldNotGetImageByRequestId() & shouldGetImageByImageId() &

shouldNotGetImageByImageId() &
shouldGetImageByRequestIdForOtherType()

Major function 7: analysisImage

- Description: extract information from uploaded image
- Test Plan: analyze a none-existed image and analyze an existed one.
- Equivalence partition: the valid equivalence partition is request parameter with existent multipart image file, the invalid one is all other types and non-existent image file.
- Corresponding test cases: com.iceberg.controller.ReiReqeustControllerTest.shouldAnlaysisImage() & shouldNotAnalysisImageForPutImageError() & shouldNotAnalysisImageForPutImageException() & shouldNotAnalysisImageForFileError()

Major function 8: uploadImage

- Description: upload image to image storage
- Test Plan: upload valid image files with correct session, and invalid image files or with incorrect code/permission
- Equivalence partition: the valid equivalence partition is valid image file and corresponding request id. The invalid equivalence partition is invalid image file and wrong id.
- Corresponding test cases: com.iceberg.controller.ReiReqeustControllerTest.shouldUploadImage & shouldUploadImageForUpdateException() & shouldUploadImageForUpdateErrorCode() & shouldNotUploadImageForFileException() & shouldNotUploadImageForFileError() & shouldNotUploadImageForPermissionDenied() & shouldNotUploadImageForRequestsNotFound()

3. *OAuth controller:*

Major function 1: githubLogin()

- Description: use OAuth protocol to implement github-login.
- **Reason why we can't test all the branches:** we find it is very difficult to test the callback function from the github. Because we don't know the code provided by github when it calls back to our endpoint. We need to use this code to get credentials of users and get user information from github accounts. The code is provided by the Github, so we don't know the code and we cannot simulate a test.

Major function 2: parseAccessTokenResponse()

- Description: parse the access token from the github.
- Test Plan: test whether the function can parse an access token.
- Test case: testParseGithubObject().

***Service*:**

1. *UserInfo service:*

Major function 1: add()

- Description: add users into the database
- Test plan: Because in our system, there are only three roles, Administrator, Group manager and normal users. Role ID could only be 1, 2 and 3. For invalid inputs, we will set the roleid of a user to 4 and 0 to test.
- Corresponding test cases: com.iceberg.service.UserInfoServiceImplTest.testAdd() & add2Test()

Major function 2: update()

- Description: find a user by id and update the user's information
- Test plan: Similar to test plan in add function. When the new role id of the user is not in the period of 1 to 3. The request is regarded as an invalid input. What's more, when the user id input does not exist, this input can also be regarded invalid.

- Corresponding test cases: com.iceberg.service.UserInfoServiceImplTest.testUpdate()

Major function 3: delete()

- Description: delete a user in the database
- Test Plan: The user to be deleted is found by user id. Therefore for this test, we are trying to delete a user that does exist and a user that does not exist.
- Corresponding test case: com.iceberg.service.UserInfoServiceImplTest.testDelete()

Major function 4: getUsersByWhere

- Description: Get users' information from the database. Because in our system, there are only three roles, Administrator, Group manager and normal users. Therefore, the group manager cannot view administrator's information but can have access to normal user's information.
- Test plan: Requests from the group manager can get all users' information in their group but cannot get administrator's.
- Corresponding test cases: com.iceberg.service.UserInfoServiceImplTest.testGetUsersByWhere()

Major function 5: getUserInfo() & getUserInfoById()

- Description: Get users' information by the given information or by only id.
- Test Plan: For valid input, give a user that does exist and get the result. For invalid input, input user information or user id that do not exist in the database.
- Corresponding test case: com.iceberg.service.UserInfoServiceImplTest.testGetUserInfo() & testGetUserInfoWithOnlyId()

2. ***ReiRequest service:***

Major function 1: add()

- Description: Add a new reimbursement request into the database.
- Test plan: When adding a new reimbursement request, there are many inputs like title, id, userid, money and so on. For the data type restriction, it's done when set those parameters. You can't set a string to user id. Money input limitation is 0 to 1000. So for the equivalence partition, there is an equivalence 1-1000, an equivalence class < 0, and an equivalence class > 1000.
- Corresponding test cases: com.iceberg.service.ReiRequestServiceImplTest.addReiRequestTest()

Major function 2: update()

- Description: Update an existing reimbursement.
- Test plan: Similar to the test plan in add function (major function 1). There is an equivalence 1-1000, an equivalence class < 0, and an equivalence class > 1000.
- Corresponding test cases: com.iceberg.service.ReiRequestServiceImplTest.updateTest()

Major function 3: delete()

- Description: Delete an existing reimbursement.
- Test plan: The reimbursement to be deleted is found by id. Therefore for this test, we are trying to delete a reimbursement that does exist and a reimbursement that does not exist.
- Corresponding test cases: com.iceberg.service.ReiRequestServiceImplTest.delTest()

Major function 4: findByWhere()

- Description: Find reimbursements.
- Test plan: The reimbursement to be deleted is found by id. Therefore for this test, we are trying to delete a reimbursement that does exist and a reimbursement that does not exist.

- Corresponding test cases: com.iceberg.service.ReiRequestServiceImplTest.findByWhereTest()

3. ***Privilege Service:***
    - Description: Privilege service manages the website permissions given to different roles. When a new role is added, default 6 privileges are given to that role. You can also check a specific role's privileges and delete a role along with his privileges.
    - Test plan: All of the above methods have an input "roleid", which should be an integer. Therefore, there is an equivalence class (valid) with an integer input and an equivalence class (invalid) with a non-integer input.
    - Corresponding test cases: com.iceberg.service.PrivilegeServiceImplTest

Utils:

1. ***HttpClient utils:***
    Major function 1: doGet()
    - Description: execute doGet function.
    - Test case: doGetTest().

    Major function 2: doPost()
    - Description: execute doPost function.
    - Test case: doPostTest().

2. ***Mail utils:***
    Major function 1: sendMail()
    - Description: send mails to an email address.
    - Test Plan: test whether we can send an email successfully.
    - Test case: testSendEmail().

External api:

1. ***Paypal api:*** Since we have more types of input than functions, we change forms of our test plan. For the tests (create, get, cancel...) in this function, the input should be valid money value, currency, receiver account, and item id.

   --money value: in our scenario, the money will be limited from 0 to 1000, and the type is float. Thus:

      The valid equivalence partitions are all float value in the range, and the boundary conditions will be 0.00 and 1000.00.

      The invalid equivalence partitions are all other values, such as float 1100.00 or string “300”.

   In our tests, all requests with valid input will return a corresponding response, while those with invalid input will return an exception.

   --currency: in our scenario, since paypal only supports USD transferring money service right now, only string “USD” is considered valid. Thus:

   1. The valid equivalence partition is string “USD”, and there is no boundary condition.
   2. The invalid equivalence partition is all other values, such as other types or string “usd”.

   --receiver account and item id: in our scenario, these two values are string type. The function works whether the receiver account and id exist, the response will tell you adding successfully, cancelling successfully or not. In system level, they all work. What only matters is the length of these two string (for storage in our system). Thus:

   1. The valid equivalence partitions are all string with length from 0 to 40, and the boundary conditions will be string with length 0 and that with 40.
   2. The invalid equivalence partitions are all other values, such as other type or string with length 41.

   In our tests, all requests with valid input will return a corresponding response, while those with invalid input will return an exception.

2. ***OCR api:*** For the tests in this function, the input should be a valid string url. Then the photo will be processed and return corresponding information. Even the photo which is not a receipt or bill will return information (not error or exception).

   --Document Base 64 and document URL: in our scenario, the document url's type is string, and the file exists. Thus:
   1. The valid equivalence partitions are all real photos (jpeg, jpg, png) url strings whether online or in local.
   2. The invalid equivalence partitions are all other values, such as other types (float, int, boolean) or string not pointing to a photo (pdf, word, csv).

   In our tests, all requests with valid input will return a corresponding response, while those with invalid input will return an exception.

3. ***Image storage api:***
   Major function 1: getFileBytes()
   - Description: get image file data in bytes
   - Test plan: this is a basic class for supporting high-level class. We just need to test if given the valid file path, the corresponding byte array is returned. The valid equivalence is all true file path string, no boundary. The invalid equivalence is all other input types or strings not a valid file path which will trigger an io exception. Our test cases will test both circumstances.
   - Corresponding test cases: com.iceberg.externalapi.ImageStorageServiceTest.shouldGetFileBytesForvalidPath() & shouldNotGetFileBytesForInvalidPath()

   Major function 2: writeToFile()
   - Description: write image bytes to files
   - Test plan: this is also a basic class for supporting high-level class. We just need to test if given data and file path, the

corresponding data appears in the corresponding file by using class loader. The valid equivalence for data is bill photo byte array, and the valid equivalence for file path is the same as the above “get” function. The invalid equivalence is all other input types or same type but with non-existent value. Our test cases will test both circumstances.
- Corresponding test cases: com.iceberg.externalapi.ImageStorageServiceImplTest.shouldWriteToFile() & shouldNotWriteToFile()& shouldWriteToFileExists()

Major function 3: getImageBytes()
- Description: get image bytes from s3 bucket
- Test plan: In our system, we save the bill photos in a s3 bucket. If given true keyname, the function will return corresponding file bytes. The valid equivalence is all true file key names stored in s3 bucket. The invalid equivalence is all other input types or strings not a valid file path which will trigger an s3 exception. Our test cases will test both circumstances.
- Corresponding test cases: com.iceberg.externalapi.ImageStorageServiceImplTest.shouldGetImageBytes() & shouldNotGetImageBytes()

Major function 4: putImage() *2
- Description: put image bytes from s3 bucket
- Test plan: this is also a basic function for interacting with s3. We take object key and file source as input, then put the data into s3, with a response string returned back. The valid equivalence for object key is all string. We have two putimage methods--”from file” or “from bytes”. The valid equivalence for file source from file is existent file path string, and that from bytes is true file byte array. The invalid equivalence for object key is all other input types. The invalid equivalence for file source from file is non-existent file path string or other input types. The invalid equivalence for file source from bytes is

invalid byte arrays or other input types. Both invalid equivalence for data sources will trigger an s3 exception. Our test cases will test both circumstances--check if given valid input, corresponding response is returned; if given invalid input, corresponding exception is returned.

- Corresponding test cases: com.iceberg.externalapi.ImageStorageServiceImplTest.shouldputImageFromFile() & shouldNotputImageFromFile()& shouldputImageFromBytes() & shouldNotputImageFromBytes()

Entity:

Entities are the entity class in our system which just contains some constructor functions and some getter and setter methods. And they have already been covered in other tests.

Configs:

Configs are some basic configuration classes of our spring boot framework. They have already been covered in other tests.

Part 3:
Coverage: 93%
https://github.com/JackSnowWolf/Iceberg/tree/master/report/jacoco
Please check the index.html. Download files or use external website for preview: http://htmlpreview.github.io/

Part 4:
Continuous integration:
https://github.com/JackSnowWolf/Iceberg/tree/master/.github
https://github.com/JackSnowWolf/Iceberg/blob/master/.circleci/config.yml

Java CI with Maven passing CodeQL passing CheckStyle passing

We institute continuous integration for our github repo. When there is a new commit to our main branch, automated build and test is initiated.