Assignment T5: Second Iteration

Team Name: Alchemist

Members:

Zihan Guo , Erica Chenchen Wei Jialiang Zhang Linxiao Wu

Part 1. User story

Revise your user stories, from the first iteration and/or project proposal, to reflect what is now fully implemented, tested and working. Do not include anything that is not actually evident in your github repository and/or you will not be able to show in your final demo.

Compare the user stories given in the second iteration report to those given in the first iteration report (part 2). If there are any differences you did not expect, you should discuss the discrepancies with the team. There is no grade for this part, except do not grade the rest of the report if the user stories are missing.

Between Release 1 and Release 2, we have made the following changes:

1. Jialiang Zhang has added **OAUTH 2.0** support.
2. Jialiang Zhang has improved our **admin panel** such that admin users are able to approve/disapprove new user registration requests. In addition, an admin user can make another user an admin.
   a.

List (10) Create With selected

	Username	Email	Image File	Password	Is Approved	User Ty
	zihan_v2	z@admin.com	default.jpg	$2b$12$7z.7m5b7Ab0LuyNNxnHideGLUNRQLOUQSvAKwaeUw8ZF4un/P5rpa	Yes	user
	linxiaow	linxiaow@umich.edu	default.jpg	$2b$12$rgcOOUPY8G03acBfwY28lOgbEhq3O./kDP.pTDQQAf1rKXXbVfsVG	Yes	user
	123	123@123.com	default.jpg	$2b$12$rXp8VmyNZntumhZwdicubuq2feF7bDFfvpT4jBS5cqpDRB2TGtiYK	Yes	user
	test	test@test.com	default.jpg	$2b$12$WIoQdzghokq5awBl0gDxY.Ew.v.C5bJTW6RA5EhkI55A1D/l39UNG	Yes	user
	maybe501	wulinxiao1997@gmail.com	default.jpg	$2b$12$S/cGiTSfNRM.KhQTKQl5JebyJqybyY6W/DXT/i8lXMjitruhupb2u	Yes	user
	testuser	testuser@testuser.com	default.jpg	$2b$12$kJOnlGZo4FXC.hEXPpR/H.BL78mi7N6.kmaPygNjAxVPZi.5x7ArO	Yes	user
	Erica	cw3137@columbia.edu	default.jpg	$2b$12$huMrQoB1/Jq.KLBbuLqLFOLZUBpicURGtoDUihJ/e4U7a50yiAii6	Yes	user
	zzz	zguocollege@gmail.com	default.jpg	$2b$12$qyPL.ztJ5bSLN/772iluyOTbT6alZwxohSjUgtpyK3sKPSQprVRxO	Yes	user
	admin	736693495@qq.com	default.jpg	$2b$12$FxLZigHZJXbc2M0k6GeDBO6K8pPqk1Du6A3q.qonCZrNWLxB2Hbxu	Yes	admin
	test_user007	test_user007@admin.com	default.jpg	$2b$12$Ray8.R5vE8ByVSJW9g.dlebg3G07p4OaX2DdzYgxMAO0R56Er0Tc2	Yes	user

3. Zihan Guo has set-up CI/CD using **AWS CodePipeline** and **AWS Elastic Beanstalk**
4. Erica Chenchen Wei improved integration of our flask application with dask application using **werkzeug.middleware.dispatcher**

5. Linxiao Wu improved strategy failure events using pop-up alerts; in addition, we improved our U.I. by displaying the error trace and style check failure reason on the same page.

Users: Investment Professionals

<**Strategy Upload**>**:** As an investment professional, I want to easily upload all trading strategies so that I can get results and compare them smoothly. My conditions of satisfaction are <

Common case:

1. Investment professionals should log in before they can upload their strategies.
2. Investment professionals upload the valid trading strategies(.py) file and then they have options to upload or not.
3. After uploading, they can choose to use which strategy to run backtesting.
4. After backtesting is done, then can view the result and save to compare with others.

Special cases:

1. Investment professionals upload an invalid strategy, which can't be compiled, a warning message will prompt up to let them know.
2. Investment professionals do not have an account

Users: Administrators

<**Access Control**>: As an <admin>, I want to <manage users' access to the system> so that this user can log in to the application and access relevant information. My conditions of satisfaction are <
common case:
Admin approves the user's login and adds the user to the User table. Admin disapproves a user's login by removing the user from the User table.

Special cases:

1. Admin tries to approve a user's login request when the email matches an existing record in the user database.

Part 2. Equivalence Partitions & Boundary Conditions

Write a test plan that explains the equivalence partitions and boundary conditions necessary to unit-test each of the major subroutines in your system (methods or functions, excluding constructors, getters/setters, helpers, etc.) and then implement your plan. Associate the names of your specific test case(s) with the corresponding equivalence partitions and boundaries (if applicable). Your test suite should include test cases from both valid and invalid equivalence partitions, and just below, at, and just above each equivalence class boundary (or inside vs. outside the equivalence class when boundary analysis does not apply). Note the same test case might apply to multiple equivalence classes. Say there is a method whose input should be an integer between 1 and 12. Then there is an equivalence class 1-12, an equivalence class <1 (or <=0), and an equivalence class >12 (or >= 13). A test case with input 0 would be just below the lower boundary of the equivalence class 1-12 and also at the high boundary of the equivalence class <1.

Include the link to the folder in your github repository that contains your automated test suite.

We have equivalence partitioning & boundary condition tests for our major module:

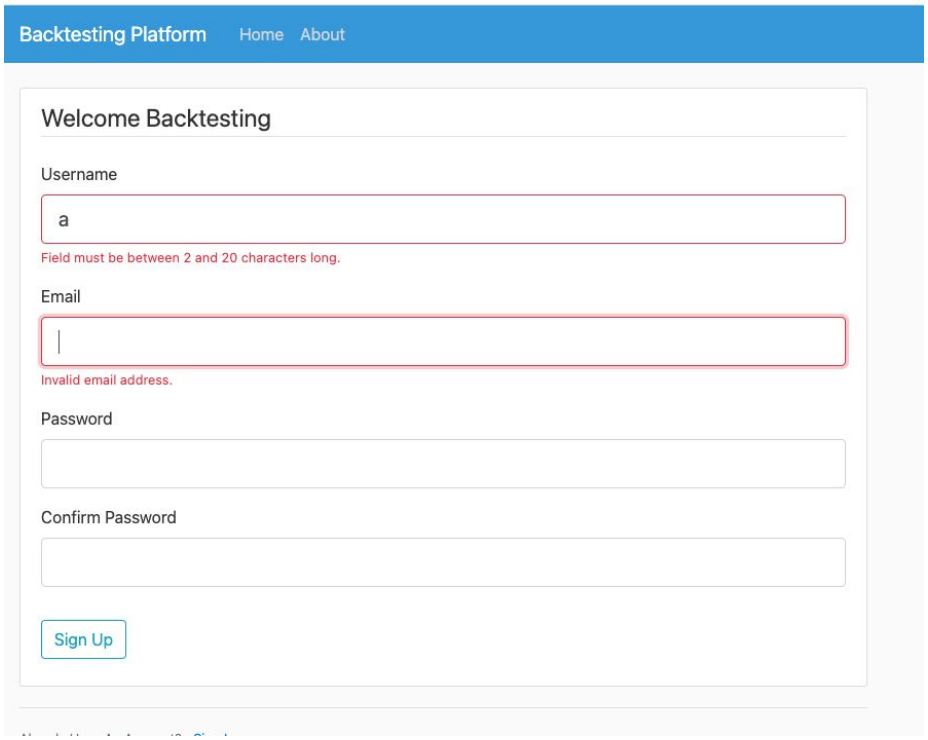
1. **Registration & login:** we have both username check and password check and test
2. **Upload**: we have file name, type, length check and test
3. **Backtesting:** Not applicable because this module does not accept user input
4. **Visualization**: strategy id check and test

Names of test case: Username length

We consider a user's username when a new user tries to register, the length of username should be in the range from 2 to 20. The equivalence class is [2, 20] inclusively. Here are the test conditions when the username length drops in [0, 2), [2, 20], (20, 50] :

1. Length of username less than 2 is considered invalid. Username cannot be a string that is less than two characters. [when username is in [0, 2)]
2. Length of username from 2 to 20 is considered valid. Users can choose a username that contains at least 2 characters and at most 20 characters. [when username is in [2, 20]]
3. Length of username greater than 20 is considered invalid. Users cannot choose a username that contains more than 20 characters. [when username length is in (20, 50]]

Case 1:




```
"""
username = StringField('Username', validators=[DataRequired(), Length(min=2, max=20)])
email = StringField('Email', validators=[DataRequired(), Email()])
```

Case 1 & Case 3:

```python
def test_register_short_username(self):
    """
    test register a user with username shorter than 2 characters
    """
    self.app.get(
        "/register"
    )
    response = self.app.post(
        "/register",
        data={
            "username": "1",
            "email": "1@1.com",
            "password": "1234567",
            "confirm_password": "1234567"
        },
    )
    self.assertEqual(response.status_code, 200,
                     "register a user with username shorter than 2 characters")

def test_register_long_username(self):
    """
    test register a user with username longer than 20 characters
    """
    self.app.get(
        "/register"
    )

    response = self.app.post(
        "/register",
        data={
            "username": "abcdefghijklmnopqrstuvwxyz",
            "email": "abcdefghijklmnopqrstuvwxyz@1.com",
            "password": "1234567",
            "confirm_password": "1234567"
        },
    )
    self.assertEqual(response.status_code, 200,
                     "register a user with username longer than 20 characters")
```

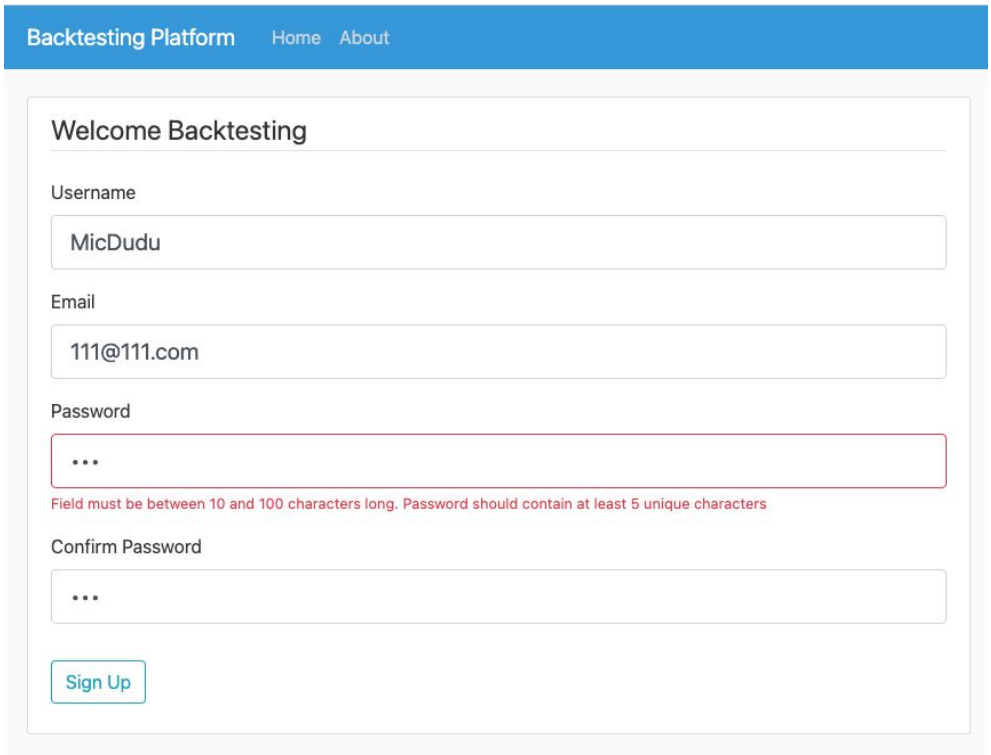
Case 2 (valid):

```python
def test_register(self):
    """test register a user with valid input info"""
    self.app.get(
        "/register"
    )

    response = self.app.post(
        "/register",
        data={
            "username": "12345",
            "email": "12345@0.com",
            "password": "abc.12345",
            "confirm_password": "abc.12345"
        },
    )

    self.assertEqual(response.status_code, 302,
                     "Unable to register for the test user")

    parsed_url = urlparse(response.location)
    path = parsed_url.path

    self.assertEqual(
        path, "/login",
        "Redirect location is not /login"
    )

    conn = rds.get_connection()
    cursor = conn.cursor()
    query = "delete from backtest.user where username = '12345';"
    cursor.execute(
```

Names of test case: Password length

We consider a user's password when a new user tries to register, the length of password should be in the range from 10 to 100. The equivalence class is [10, 100] inclusively. Here are the test conditions when the password length drops in [0, 10), [10, 100], (100, 500] :

1. Length of password less than 10 is considered invalid. Password cannot be a string that is less than ten characters. [when password is in [0, 10)]
2. Length of password from 10 to 100 is considered valid. Users can choose a password that contains at least 10 characters and at most 100 characters. [when password is in [10, 100]]
3. Length of password greater than 100 is considered invalid. Users cannot choose a password that contains more than 100 characters. [when password length is in (100, 500]]



Case 1:

```python
def test_register_short_password(self):
    """
    test register a user with password shorter than 10 characters
    """
    self.app.get(
        "/register"
    )
    response = self.app.post(
        "/register",
        data={
            "username": "testuser",
            "email": "testuser@testuser.com",
            "password": "123",
            "confirm_password": "123"
        },
    )
    self.assertEqual(response.status_code, 200,
                     "register a user with password shorter than 10 characters")
```

Case 2 (valid):

```python
def test_register_with_valide_password(self):
    """
    test register a user with valid password
    """
    self.app.get(
        "/register"
    )

    response = self.app.post(
        "/register",
        data={
            "username": "12345",
            "email": "12345@0.com",
            "password": "abc.1234567890",
            "confirm_password": "abc.1234567890"
        },
    )

    self.assertEqual(response.status_code, 302,
                     "Unable to register for the test user")

    parsed_url = urlparse(response.location)
    path = parsed_url.path

    self.assertEqual(
        path, "/login",
        "Redirect location is not /login"
    )

    conn = rds.get_connection()
    cursor = conn.cursor()
    query = "delete from backtest.user where username = '12345';"
    cursor.execute(
        query
    )
    conn.commit()
```

Case 3:

```python
def test_register_long_password(self):
    """
    test register a user with password longer than 100 characters
    """
    self.app.get(
        "/register"
    )
    response = self.app.post(
        "/register",
        data={
            "username": "testuser",
            "email": "testuser@testuser.com",
            "password": "1234567"*100,
            "confirm_password": "1234567"*100
        },
    )
    self.assertEqual(response.status_code, 200,
                     "register a user with password longer than 100 characters")
```

Names of test case: Password Uniqueness

We also consider the password content when a new user tries to register, the password should contain 5 or more unique characters. Here are the test conditions when the number of unique characters in the password drops in [0, 5), [5, 100]:

1. The number of unique characters in the password less than 5 is considered invalid. Password cannot be a string that contains less than 5 unique characters. [when number of unique character is in [0, 5)]
2. The number of unique characters in the password at least 5 is considered valid. Password should be a string that contains 5 or more unique characters. [when number of unique character is in [5,100]]

Case 1:

```python
def test_register_duplicate_characters_password(self):
    """
    test register a user with password contains less than 5 unique characters
    """
    """
    test register a user with valid password
    """
    self.app.get(
        "/register"
    )

    response = self.app.post(
        "/register",
        data={
            "username": "12345",
            "email": "12345@0.com",
            "password": "123123123123",
            "confirm_password": "123123123123"
        },
    )
    self.assertEqual(response.status_code, 200,
                     "Unable to register for the test user")
```

Case 2 (valid):

```python
def test_register_unique_characters_password(self):
    """
    test register a user with password contains at least 5 unique characters
    """
    """
    test register a user with valid password
    """
    self.app.get(
        "/register"
    )

    response = self.app.post(
        "/register",
        data={
            "username": "12345",
            "email": "12345@0.com",
            "password": "abc.1234567890",
            "confirm_password": "abc.1234567890"
        },
    )

    self.assertEqual(response.status_code, 302,
                     "Unable to register for the test user")

    parsed_url = urlparse(response.location)
    path = parsed_url.path

    self.assertEqual(
        path, "/login",
        "Redirect location is not /login"
    )

    conn = rds.get_connection()
    cursor = conn.cursor()
    query = "delete from backtest.user where username = '12345';"
    cursor.execute(
        query
    )
    conn.commit()
```

Names of test case: Upload Filename length

We corporate this case in acceptance testing on the uploading files, where equivalence class for input filename length is (0, 50) exclusively. We consider three partitions of the filename length , [0], (0, 50), and [50, 100]

1. If the user doesn't give an empty filename, which the filename length is 0, it will give a warning and the user can't upload the file with an empty filename. We test with "" filename. This is also a boundary condition. We also tested boundaries by passing in filename lengths of 49, 50 and 51 respectively since 50 is our boundary for the upper bound.

2. If the user enters a filename length in [50, inf) , it will give a warning and the user can't enter a filename greater or equal to 50 characters. We test with a filename length of 100.
3. If the user enters a filename length in (0, 50), the upload should be successful. We test with length 30.

Case 1: without strategy name

```python
def test_upload_no_strategy_name(self):
    """
    The test strategy name is not there
    """
    self.login()
    data = {}  # no strategy name
    with open('tests/uploads/helpers.py', 'rb') as fh:
        buf = io.BytesIO(fh.read())
        data['user_file'] = (buf, '')
    response = self.app.post(
        "/upload",
        data=data,
    )

    self.assertIn(b"No strategy name specified", response.data,
                  "cannot check no strategy_name field")
```

Case2.1 (length is 100):

```python
def test_upload_very_long_strategy_name(self):
    """
    The test strategy with very long name
    """
    self.login()
    data = {'strategy_name': 'a' * 100}
    with open('tests/uploads/helpers.py', 'rb') as fh:
        buf = io.BytesIO(fh.read())
        data['user_file'] = (buf, '')
    response = self.app.post(
        "/upload",
        data=data,
    )

    self.assertIn(b"Strategy name should not be greater than 50 characters", response.data,
                  "cannot detect very long name")
```

Case 1 (length is 50):

```python
def test_upload_long_strategy_name(self):
    """
    The test strategy name is too long
    """
    self.login()
    data = {'strategy_name': 'a' * 50}
    with open('tests/uploads/helpers.py', 'rb') as fh:
        buf = io.BytesIO(fh.read())
        data['user_file'] = (buf, '')
    response = self.app.post(
        "/upload",
        data=data,
    )

    self.assertIn(b"Strategy name should not be greater than 50 characters", response.data,
                  "cannot detect long name")
```

Case 3 (length = 30) partition range: 0 < length < 50

```python
def test_upload_normal_length_strategy_name(self):
    """
    The test strategy with normal length
    """
    self.login()
    data = {'strategy_name': 'a' * 30}
    with open('tests/uploads/helpers.py', 'rb') as fh:
        buf = io.BytesIO(fh.read())
        data['user_file'] = (buf, '')
    response = self.app.post(
        "/upload",
        data=data,
    )
    self.assertEqual(response.status_code, 200, "uploaded valid strategy")
```

Upper Boundary Tests

```python
def test_upload_long_strategy_name(self):
    """
    The test strategy name is too long
    """
    self.login()
    data = {'strategy_name': 'a' * 50}
    with open('tests/uploads/helpers.py', 'rb') as fh:
        buf = io.BytesIO(fh.read())
        data['user_file'] = (buf, '')
    response = self.app.post(
        "/upload",
        data=data,
    )

    self.assertIn(b"Strategy name should not be greater than 50 characters", response.data,
                  "cannot detect long name")
```

```python
def test_upload_lower_boundary_strategy_name(self):
    """
    test strategy name has length 49, boundary is 50.
    """
    self.login()
    data = {'strategy_name': 'a' * 49}
    with open('tests/uploads/helpers.py', 'rb') as fh:
        buf = io.BytesIO(fh.read())
        data['user_file'] = (buf, '')
    response = self.app.post(
        "/upload",
        data=data,
    )
    self.assertEqual(response.status_code, 200, "uploaded valid strategy")
```

```python
def test_upload_upper_boundary_strategy_name(self):
    """
    test strategy name has length 51, boundary is 50.
    """
    self.login()
    data = {'strategy_name': 'a' * 51}
    with open('tests/uploads/helpers.py', 'rb') as fh:
        buf = io.BytesIO(fh.read())
        data['user_file'] = (buf, '')
    response = self.app.post(
        "/upload",
        data=data,
    )
    self.assertIn(b"Strategy name should not be greater than 50 characters", response.data,
                  "cannot detect long name")
```

Names of test case: Backtesting Visualization

For the unit testing on the results page, we consider two boundaries: valid number of strategy_ids = 0 and valid number of strategy_ids > 0. Since we don't have an upper limit for how many files a user can upload and get backtest results right now, we don't have an upper boundary condition.

1. If the user doesn't have any valid backtesting results, then we do nothing on the visualization page. len(strategy_ids) = 0 is a boundary condition here.
2. If the user has any valid backtesting results, then we update the options in the dropdown bar so that the user can select which result to visualize. We test with only one strategy id, which is a boundary condition.
3. If the user has any valid backtesting results, then we update the options in the dropdown bar so that the user can select which result to visualize. We test with a list of 3 strategy ids, which partition case in (0, inf)

Case 1 & Cas2:

```python
def test_get_plot():
    """
    Test get_plot with valid strategy id list, should return true
    to demonstrate we update global variables in application,
    which will update dropdown bar for visualization.
    :return:
    """
    _, strategy_id = before_test()

    # test with this strategy id
    result = app.get_plot([str(strategy_id)])
    assert result


def test_get_plot_invalid():
    """
    Test get_plot with empty list, should return false to demonstrate nothing changed.
    :return:
    """
    result = app.get_plot([])
    assert not result
```

Case 3:

```python
def test_get_plot_more_results():
    """
    Test get_plot with valid strategy id list, should return true
    to demonstrate we update global variables in application,
    which will update dropdown bar for visualization.
    :return:
    """
    _, strategy_id1 = before_test()
    _, strategy_id2 = before_test()
    _, strategy_id3 = before_test()
    test_ids = [str(strategy_id1), str(strategy_id2), str(strategy_id3)]

    # test with 3 strategy ids
    result = app.get_plot(test_ids)
    assert result
```

Part 3. Branch Coverage

Measure the branch coverage achieved by your automated test suite. This requires using a coverage tool appropriate for your programming language and platform. Branch coverage should strive to achieve 100%, but may not reach 100%. Add more test cases until you reach at least 90%. Each additional test case should try to force a particular branch that was not previously covered. Tell us what branch coverage you finally did achieve. If the coverage tool

reports less than 100% (that is, between 90% and 99%), discuss one example of a branch that your test cases did not cover and explain why it is difficult to test this branch. If you were unable to reach 90%, explain why not.

Include the link to the folder in your repository that contains the coverage test reports. Note this means you need to configure your coverage tool to produce reports that can be saved as files in your repository.

We have an overall coverage of 92%.

The reason why we cannot cover some branches/code snippets is because

1. Some branches requires authentication token that we cannot mock, for example, Google API
2. We separate the test branch and production branch for some use cases for a deterministic testing pattern.
3. For the forget password feature, we would need to set-up email access to retrieve the reset password token. Which is very convoluted so we assume Google API is robust in delivering the password reset token.

Example for division of test branch and production

```
    conn = rds.get_connection()
    cursor = conn.cursor()
    if test_id is not None:
        test_id = int(test_id)
        logger.info("uploading testing file...")
        cnt_loc = test_id
    else:
        logger.info("uploading user file...")
        cursor.execute(
            "SELECT MAX(strategy_id) as max_strategy FROM backtest.strategies"
        )
        first = cursor.fetchone()
        cnt_loc = first['max_strategy']
        cnt_loc += 1
        logger.info("max + 1 is - %s", cnt_loc)
```

If branch leads to test part, which can be tested, but else goes to production, so it is not testable

Example for security token:

```python
@application.route("/OAuth_login/callback")
def callback():
    """
    OAuth login callback function from google auth page
    :return:
    """
    code = request.args.get("code")
    google_provider_cfg = requests.get(application.config["GOOGLE_DISCOVERY_URL"]).json()
    token_endpoint = google_provider_cfg["token_endpoint"]
    token_url, headers, body = client.prepare_token_request(
        token_endpoint,
        authorization_response=request.url,
        redirect_url=request.base_url,
        code=code
    )
    token_response = requests.post(
        token_url,
        headers=headers,
        data=body,
        auth=(application.config["GOOGLE_CLIENT_ID"], application.config["GOOGLE_CLIENT_SECRET"])
    )
    client.parse_request_body_response(json.dumps(token_response.json()))
    userinfo_endpoint = google_provider_cfg["userinfo_endpoint"]
    url, headers, body = client.add_token(userinfo_endpoint)
    userinfo_response = requests.get(url, headers=headers, data=body)
    if userinfo_response.json().get("email_verified"):
        unique_id = userinfo_response.json()["sub"]
        user_email = userinfo_response.json()["email"]
        picture = userinfo_response.json()["picture"]
        user_name = userinfo_response.json()["given_name"]
        current_user.id = int(unique_id)
    else:
        return "User email not available or not verified by Google", 400
    user = OAuthUser(
        id_=unique_id, username=user_name, email=user_email, image_file=picture
    )
    if not OAuthUser.get(unique_id):
        OAuthUser.create(unique_id, user_name, user_email, picture)
    login_user(user)
    return redirect(url_for("home"))
```

To reach the red (uncovered) part, we need to have the authentication token, which is not possible

The image to our coverage report is :

```
└ $ ▶coverage report
Name                                          Stmts   Miss  Cover
-------------------------------------------------------------------
application.py                                  668    101    85%
config.py                                        24      0   100%
errors/__init__.py                                0      0   100%
errors/handlers.py                               15      4    73%
strategies/user_id_11/__init__.py                 0      0   100%
strategies/user_id_11/current_strategy.py        21      0   100%
tests/__init__.py                                 0      0   100%
tests/acceptance/__init__.py                      0      0   100%
tests/acceptance/test_account.py                 19      0   100%
tests/acceptance/test_admin.py                   18      0   100%
tests/acceptance/test_baseclass.py               12      0   100%
tests/acceptance/test_errors.py                   5      0   100%
tests/acceptance/test_home_welcome.py            13      0   100%
tests/acceptance/test_login_logout.py            56      0   100%
tests/acceptance/test_oauth_login.py              9      0   100%
tests/acceptance/test_register.py                23      0   100%
tests/acceptance/test_reset_password.py          15      0   100%
tests/acceptance/test_show_results.py            10      0   100%
tests/acceptance/test_show_strategies.py         18      0   100%
tests/acceptance/test_upload.py                  84      2    98%
tests/acceptance/test_welcome.py                  9      0   100%
tests/unit/__init__.py                            0      0   100%
tests/unit/test_dash.py                          99      0   100%
tests/unit/test_helper.py                        51      0   100%
tests/unit/test_user.py                          32      0   100%
tests/unit/test_utils.py                         72      0   100%
user.py                                          28      0   100%
utils/__init__.py                                 0      0   100%
utils/mock_historical_data.py                    28      0   100%
utils/rds.py                                     21      0   100%
utils/s3_util.py                                 23      0   100%
-------------------------------------------------------------------
TOTAL                                          1373    107    92%
```

The link to our detail reports is:
https://github.com/gzhami/research_lab/blob/dev/submissions/coverage_report.txt

Part 4. Continuous Integration

As part of the release effort, you need to institute continuous integration for your codebase. The idea is to integrate automated build and test with your version control repository using Travis CI or a similar tool, so that build and test is automatically initiated whenever there is a new commit to the main branch of your github repository. Make sure to ask for help far in advance of the assignment deadline if you have trouble getting CI to work.

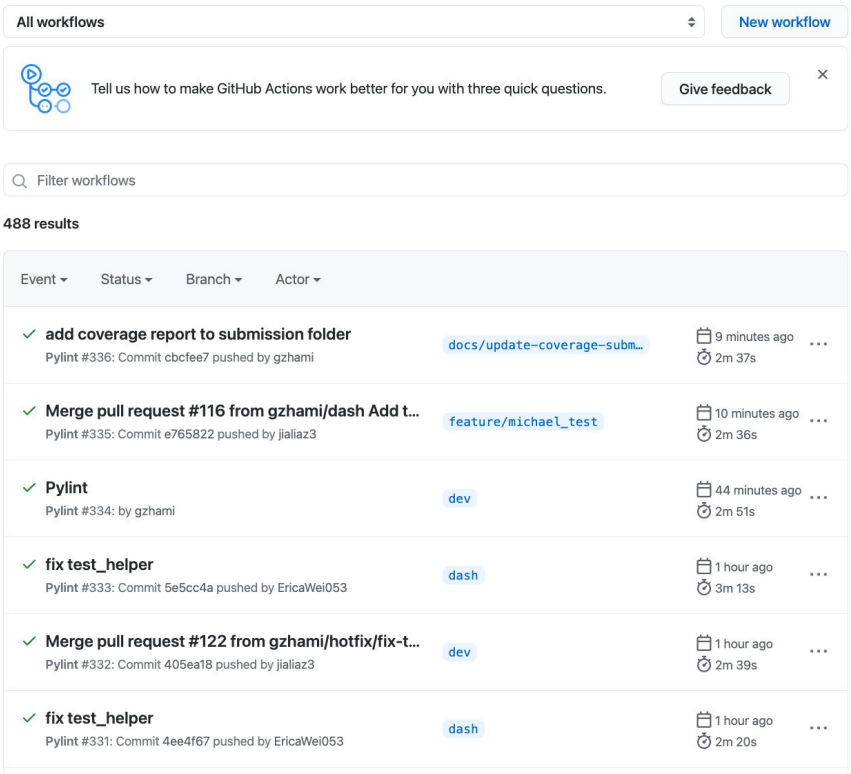
Include links to the files in your repository that configure the CI tool(s) and a folder in your repository that includes the CI reports. Note this means you need to configure your CI tool to produce reports that can be saved as files in your repository.

5 points for CI configuration (e.g., .travis.yml file), 0 points if not found.

5 points for a folder in the github report containing CI reports, or any other mechanism allowing you (the grader) to access the CI reports, that demonstrates the CI indeed works and is being used. This should be 0 if CI does not appear to be working.

We use Github Action instead of Travis CI because Github Action is Github-native and easy to use.

We use Git Action to test if everything compiles on Linux. In addition, we deploy our code using AWS CodePipeline and AWS Elastic Beanstalk. It gets automatically deployed every time we push to dev/master. For details, please see screenshot below.



gzhami / research_lab

Unwatch 1 Star

Code | Issues 7 | Pull requests 1 | Actions | Projects | Wiki | Security

add coverage report to submission folder

docs/update-covera… Zihan Guo cbcfee7

Pylint
on: push

1

build-linux

build-linux
succeeded 4 minutes ago in 2m 25s

Search logs

- Set up job
- Run actions/checkout@v2
- Set up Python 3.8
- update conda to latest version
- Install dependencies
- Pylint
- Post Run actions/checkout@v2
- Complete job

aws Services ▼ zihan.guo @ 2260-8223-1735 ▼ Ohio ▼ Suppo

Elastic Beanstalk

- Environments
- Applications

▼ alchemist-dev
- Application versions
- Saved configurations

▼ AlchemistDev-env
- Go to environment
- Configuration
- Logs
- Health
- Monitoring
- Alarms
- Managed updates
- Events
- Tags

▼ Recent environments
- AlchemistProd-env
- AlchemistDev-env
- SignalDevV20-env
- SignalDevV10-env

Elastic Beanstalk > Environments > AlchemistDev-env

Shared Application Load Balancer: Save on load balancer costs. Create an ALB once, and use it when you create multiple web server environments. Learn more

In September 2020, Elastic Beanstalk introduced the `EnhancedHealthAuthEnabled` option. It enables you to require authorization of instances that report enhanced health information. If you're using an Elastic Beanstalk managed policy for your environment's instance profile (the default when using Elastic Beanstalk console or EB CLI), you can safely enable this option.

On November 30, 2020, we plan on enabling this option by default for all new environments (no impact on existing environments). On May 31, 2021, we plan to start enforcing enhanced health authorization; it will be enabled for all new and existing environments, with no option to disable it.

If you're using a custom instance profile, your environment might be impacted and might need a configuration update. To learn more, see Enhanced health authorization in the *AWS Elastic Beanstalk Developer Guide*.

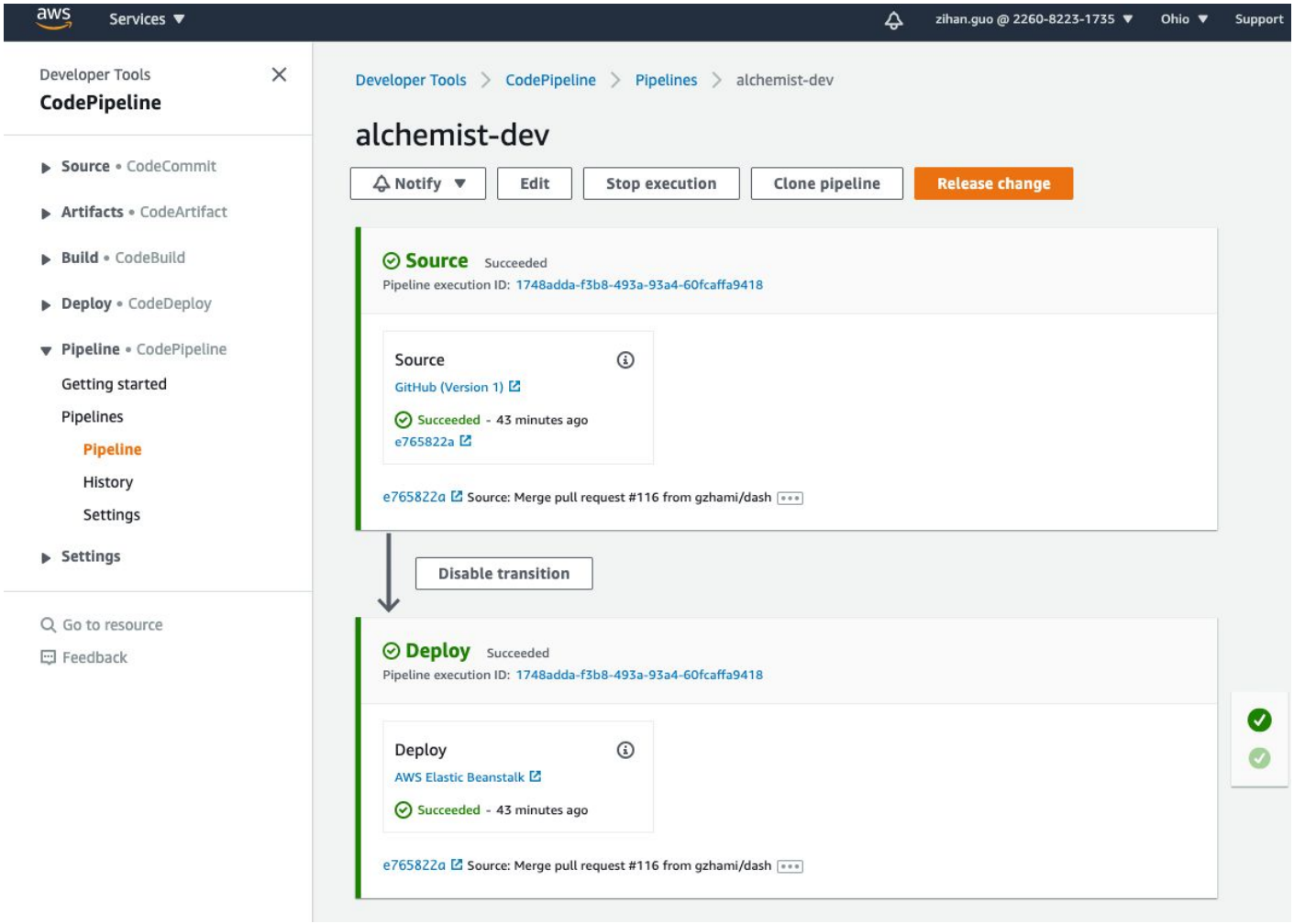
AlchemistDev-env

AlchemistDev-env.eba-upatp3uh.us-east-2.elasticbeanstalk.com (e-un5593ejs7)

Application name: alchemist-dev

Refresh | Actions ▼

Health	Running version	Platform
Ok Causes	code-pipeline-1607313751022-e765822aa8c46cc479f4b19b7e4d586402e649cc Upload and deploy	Python 3.7 running on 64bit Amazon Linux 2/3.1.3 Change


aws
Services
zihan.guo @ 2260-8223-1735
Ohio
Support
Developer Tools
CodePipeline
Source • CodeCommit
Artifacts • CodeArtifact
Build • CodeBuild
Deploy • CodeDeploy
Pipeline • CodePipeline
Getting started
Pipelines
Pipeline
History
Settings
Settings
Go to resource
Feedback
Developer Tools > CodePipeline > Pipelines > alchemist-dev
alchemist-dev
Notify
Edit
Stop execution
Clone pipeline
Release change
Source Succeeded
Pipeline execution ID: 1748adda-f3b8-493a-93a4-60fcaffa9418
Source
GitHub (Version 1)
Succeeded - 43 minutes ago
e765822a
e765822a Source: Merge pull request #116 from gzhami/dash
Disable transition
Deploy Succeeded
Pipeline execution ID: 1748adda-f3b8-493a-93a4-60fcaffa9418
Deploy
AWS Elastic Beanstalk
Succeeded - 43 minutes ago
e765822a Source: Merge pull request #116 from gzhami/dash