Second Iteration

Team name: Wall-E

Team members: Yao Fu (yf2470), Wufan Shangguan (ws2541), Qing Teng (qt2126), Wenqi Wang (ww2505)

Part 1: Use Cases

Title: Register for an account

Actor: Unregistered User

Description: An unregistered user wants to get started with LrnDeep and is redirected to the register page. They register for an account with their email address and set their username and password. If this is successfully completed, they will be redirected to the canvas page.

Basic Flow:

1. An unregistered user goes to /canvas and clicks “Get Started”.
2. They are redirected to /users/signup/?next=/canvas/workspace/.
3. They enter their username, email address and password and click “Register”.
4. They have successfully registered for an account and are redirected to /canvas/workspace where they can start constructing a graph.

Alternate Flow:

4. a. Password not meeting requirements:
    1. The user entered a password that does not satisfy requirements (too short, only letters, etc.)
    2. The system reruns step 2.

Title: Log into account

Actor: Registered User

Description: The registered user logs into their account, whether via their username and password or via their Google account. If they are successfully logged in, they will be redirected to the canvas page.

Basic Flow:

1. A registered user goes to /canvas and clicks “Get Started”.
2. They are redirected to /users/signup/?next=/canvas/workspace/.
3. They click “Log In” because they already have an account! :-)

4. They enter their username and password and click “Log In”.
5. They are redirected to /canvas/workspace where they can start constructing a graph.

Alternate Flow:

4. a. Login via Google:
    1. They click on “Sign in with Google”
    2. They enter their Google account credentials and sign in.
5. a. Login failed:
    1. The user entered the wrong username and/or password.
    2. The system reruns step 3.

Title: Create a graph

Actor: Logged-in User

Description: The logged-in user drags components from the palette to form a graph from a blank canvas. They can configure the input and output dimensions of the components or delete components.

Basic Flow:

1. The user clicks on a component in the palette.
2. A pop-up window appears and the user enters the input and output dimensions of the component.
3. The component is added to the graph.

Alternate Flow:

1. a. Deleting a component:
    1. The user clicks on a component on the graph.
    2. They hit the “Delete” key.
    3. The component is deleted.
3. a. Invalid dimensions:
    1. The input and/or output dimensions that the user entered were invalid.
    2. Another pop-up window appears alerting the user that the dimensions were invalid.
    3. No components are added to the graph.

Title: Compile graph into code

Actor: Logged-in User

Description: The user wants to compile the current graph into Keras code.

Basic Flow:

1. The user clicks on “Compile”.
2. The compiler processes the valid portion of the graph and compiles it into Keras code.
3. Corresponding code is displayed on the page.

Alternate Flow:

N/A

Part 2: Test Plan

We unit-test the following subroutines: register, log in, add component to graph and compile graph. For register and login, we utilize Django’s testing tools. For add component to graph, we use Selenium to simulate user actions. For compile graph, we test JavaScript code with jstest.

Register

- The equivalence partitions are:
  - register with new username and password
  - register with existent username and password
- The boundary conditions are:
  - Boundary conditions are not applicable in this case
- Link to test suite: https://github.com/CathyMouse96/lrn-deep/blob/master/users/tests.py

Log in

- The equivalence partitions are:
  - (existent username, correct password)
  - (existent username, incorrect password)
  - (inexistent username, existent password)
  - (inexistent username, inexistent password)
- The boundary conditions are:
  - Boundary conditions are not applicable in this case
- Link to test suite: https://github.com/CathyMouse96/lrn-deep/blob/master/users/tests.py

Add component to graph

- The equivalence partitions are:
  - (valid input dimension, valid output dimension)
  - (valid input dimension, invalid output dimension)
  - (invalid input dimension, valid output dimension)
  - (invalid input dimension, invalid output dimension)
- The boundary conditions are:
  - Boundary conditions are not applicable in this case
- Link to test suite: https://github.com/CathyMouse96/lrn-deep/blob/master/canvas/tests.py

Compile graph

- The equivalence partitions are:
  - valid input data structure (not NULL)
  - valid input data structure (NULL)
  - invalid input data structure (inconsistent length of adjacent modules)
- The boundary conditions are:
  - Boundary conditions are not applicable in this case
- Link to test suite: https://github.com/CathyMouse96/lrn-deep/tree/master/js_test

Part 3: Branch Coverage

We use Coverage.py to measure branch coverage. After installing Coverage.py, we run the following commands to collect execution data and generate reports:

```
python3 -m coverage run --branch manage.py test
python3 -m coverage report
```

The report can be found at https://github.com/CathyMouse96/lrn-deep/tree/master/reports.

Part 4: Links

GitHub repository: https://github.com/CathyMouse96/lrn-deep

CI configurations:

- Pre-commit: https://github.com/CathyMouse96/lrn-deep/blob/master/bin/git-hooks/pre-commit
- Post-commit: https://github.com/CathyMouse96/lrn-deep/blob/master/.travis.yml

Test suites:

- Register and log in: https://github.com/CathyMouse96/lrn-deep/blob/master/users/tests.py
- Add component to graph: https://github.com/CathyMouse96/lrn-deep/blob/master/canvas/tests.py
- Compile graph: https://github.com/CathyMouse96/lrn-deep/tree/master/js_test

Branch coverage reports: https://github.com/CathyMouse96/lrn-deep/tree/master/reports