Team logistics and evaluation

Team name

- 2Cool 2Cook

Team member names and roles

- Ruisi Jian (Expert coder/Full-stack)
- Yueting Liao (Team leader)
- Yuchen Wang (Front-end/UI design)
- Yifan Wang (Back-end/Database design)
- Xiaoyu Zhang (Back-end/Search algorithm)

Any changes to your team dynamics, individual roles, weekly team meetings schedule, how meetings are run, where to find meeting minutes and other artifacts, etc.

Weekly team meeting schedule in EDT

- Mon/Thu 9:00 AM - 10:50 AM

A summary of how you will run each meeting and where minutes will be recorded and stored (and searchable)

- Summarize and evaluate what we did in the last meeting and conclude our inadequacy.
- Each member will be reporting their progress of tasks and potential problems from the last meeting
- Discuss the progress made so far and evaluate for potential improvements.
- Provide some constructive thinking and set a list of tasks for the next meeting.
- Assign appropriate works for each team member.
- Use Google Doc to record the minutes from the meeting (One Doc for all meetings)
- Meeting Minutes: https://docs.google.com/document/d/1mJW8UWMUnrlh8FBEFf_GY9MLaYU6fGM3n-BjViskeRc/edit?usp=sharing (updated every meeting)

Nooks room we typically meet in

- K

Project's URL

- https://github.com/YuetingLiao/SDD-Team.git

Project status reports

As with previous Team Deliverable assignments, write a project status report that summarizes the current state of your project (shortly before the deliverable due date). The project status report describes an up-to-date list of issues encountered (since Team Deliverable 3). And for each issue, include a brief summary, the person responsible for resolving the issue, and how the problem is being solved or was solved.

The current state of the project:

- The overall progress of the project is approaching specific UI design

  The implementations of the essential features were completed, including search/filter information, display information on a different page, store data to the personal profile, and finalize the database tables. The detail of how each issue was being solved shows below. Improving UI design will be the next primary task for the team to promote the overall quality of the project.

- New databases were needed when implementing the Menu page

  During the implementation of the rating and comment requirement of the Menu page, Yuchen identified the need to create new databases to store the rating and comments for each recipe. He had an issue in connecting the database to the web page and designing the input bar for comment. To solve the issues, he uses the method used in the user page for connecting databases and input.

- All webpages in the project should have a consistent style

  This issue would serve as the starting point of the UI design of the project. Ruisi was responsible for solving this issue. She needed to set up the basic CSS style that all webpages of the project would employ. She also needed to find a way to conveniently link all the HTML files with this CSS file in node-red. She solved this issue by hosting the CSS file under a URL. And add a single line of link reference in each HTML file to get the CSS content so all the Webpages would get a consistent rendering style.

- An issue with implementing pagination on the node-red platform on the farmers market page

  The issue relates to finishing the implementation stage of the project. Adding pagination is essential for all the search pages, but lack of experience with the node-red platform results in members having an issue with interacting SQL with page parameters. The problem was solved by using POST requests on every search page, and the URL is changed based on the input parameter by the user.

Your project status report also must include a summary of how each team member contributed to the team deliverable 3. This helps ensure that each team member is taking an active role in each aspect of the project.

Xiaoyu Zhang: Responsible for correction of the search. Fix the errors of capital and the lowercase letter does not display the same results. Also, we add separate pages for amounts of results which means the results will not be dominated in a single page, we set the limited results in one page. Ex: The 100 results will display in 5 pages if each page set 20 is the number of maximum results.

Yuchen Wang: Responsible for comment and rating function on the recipe page. Creating new databases to store comments and ratings for recipes.

Yifan Wang: Responsible for the user profile page. Responsible for designing the dietary_preference database and connecting it to the user profile web page using node-red and SQL techniques.

Ruisi Jian: Responsible for the design and implementation of the Menu page. Responsible for setting up a single CSS file that would be used as the basis of UI design of web pages under a consistent style.

Yueting Liao: Responsible for managing the overall progress of the project, set the detailed schedule for each stage, and be supportive of the team. Also, implemented more core features for the farmers market page and improved partial UI design.

System test plan

Document in detail at least 32 test cases by describing the information below for each test. Use a spreadsheet similar to what we did in our November 9 lecture. Note that these are not unit tests; instead, these tests focus on functionality, behavior, performance requirements, ethical considerations, etc. Therefore, thoroughly review your requirements to find "good" test cases.

Test cases within the spreadsheet link:
https://docs.google.com/spreadsheets/d/1CDP5109MZNYJENNPTr1DOwuiu9ZCpnWdOanTFwsk2ks/edit?usp=sharing

In addition to the above, please indicate any necessary testing materials, e.g., a series of documents or files that you use for your tests. For this deliverable, you can simply describe these additional testing "accessories" (if any)

- Test database CSV file for user account info and menu info.

Implementation plan

Write a summary of your implementation plan by providing the following details:
1. A list of programming languages you will use; while you might use one language for your core functionality, you might also make use of other languages for peripheral tasks, e.g., data import/export, log file management, etc.
2. For each programming language, a description of what you will use this language for, as well as why you think this language is your best choice
3. For each programming language, your agreed upon coding standard, which should include both language-specific guidelines and language-independent guidelines (e.g., consistent naming and spacing conventions)
4. For each programming language, specific compilation/build requirements, if any (this ensures all team members are compiling and building the software component the same way)
Note that as you write code, you should expand and/or revise items 3 and 4 above. In other words, do not consider those to be static documents.

1. HTML, Javascript, SQL, Python
2. **HTML**: The team uses HTML to specify the display of the webpage. The team decides to use this to describe the webpage design because it is easy to use markup language that can be simply integrated with javascript and extensions such as CSS for web design.
   Javascript: The team uses Javascript to control the on-click event and dynamic display of the webpage. The team decides to use this language because Javascript can be easily integrated with HTML and it is an empirical programming language that is good at handling logic operations.
   SQL (for MySQL specifically): The team uses SQL to manage and call databases. The team decides to use this language because it is the only way to communicate with the MySQL database, which the team uses to store the data.
   Python: The team uses Python to generate CSV files for the database. The team can directly access information from the database or return the results that users expect. The team decides to use this language because it has strong modules for data processing such as pandas. The team also uses Python for unit test cases since Python has a convenient unit test case module.
3. The team uses the camel case for the consistent naming convention, the spacing conventions are based on the use of tabs. The comments in the Javascript follow Javadoc standard.
4. Installation and build requirements for team members
   OS and version of development
   Windows/ Mac:
   - MySQL Ver 8.0.21 for Win64 on x86_64 (MySQL Community Server - GPL)
   - Node-RED version: v1.1.3
   - Node.js Version: v12.18.4
   - Python version: 3.6.5

Unit tests

Unit testing is typically considered to be part of your implementation. For at least three of your methods or classes, provide a comprehensive set of unit tests.
Be sure to clearly describe your unit tests such that someone unfamiliar with your project could understand what representative test cases you are testing (e.g., from our November 9 lecture, one representative test would test valid inputs for a valid scalane obtuse triangle).
Show this via code by making sure your comments are completely clear; in other words, do not just include your test code and expect us to decipher it.

Unit Test Results:

```
test_add_menu (__main__.AddToMenuTest) ... ok
test_add_menu_invalid_cookie (__main__.AddToMenuTest) ... ok
test_add_menu_invalid_meal_type (__main__.AddToMenuTest) ... ok
test_add_menu_invalid_recipe (__main__.AddToMenuTest) ... ok
test_create_account (__main__.CreateAccountTest) ... ok
test_create_account_duplicate_username (__main__.CreateAccountTest) ... ok
test_create_account_empty (__main__.CreateAccountTest) ... ok
test_create_account_pwd_limit_16 (__main__.CreateAccountTest) ... ok
test_create_account_username_limit_25 (__main__.CreateAccountTest) ... ok
test_delete_menu (__main__.DeleteMenuTest) ... ok
test_delete_menu_invalid_cookie (__main__.DeleteMenuTest) ... ok
test_delete_menu_invalid_meal_type (__main__.DeleteMenuTest) ... ok
test_delete_menu_invalid_recipe (__main__.DeleteMenuTest) ... ok
test_get_macro (__main__.GetMacroTest) ... ok
test_get_macro_empty (__main__.GetMacroTest) ... ok
test_get_macro_invalid_cookies (__main__.GetMacroTest) ... ok
test_get_macro_invalid_meal_type (__main__.GetMacroTest) ... ok

----------------------------------------------------------------------
Ran 17 tests in 1.392s

OK
```

Unit Test Code: (In Python)

import unit test, requests, PyMySQL, random

#Help Function: make a request to database with an sql query as input

```
def search_db(sql):
    try:
        conn = pymysql.connect("localhost", "root", "123456", "sdd")
        curs = conn.cursor()
        curs.execute(sql)
        conn.commit()
        result = curs.fetchall()
    except EOFError:
        conn.rollback()
    finally:
        conn.close()
        return result
```

#1 Unit Test for AddToMenu method

```
class AddToMenuTest(unittest.TestCase):
```

#Purpose: Test an appropriate addition to the menu
#Input: valid user id, valid meal type, valid recipe id
#Expected output: new recipe should be inserted in menu, menu_ins, and menu_ingr table in the database

```
    def test_add_menu(self):
        before = search_db("SELECT count(*) FROM menu")
        before1 = search_db("SELECT count(*) FROM menu_ins")
        before2 = search_db("SELECT count(*) FROM menu_ingr")
        cookies = {
            'id': 'jianr'
        }
        data = {
          'type': 'b',
          'name': '64d3724048'
        }
        response = requests.post('http://localhost:1880/addToMenu', cookies=cookies,
data=data).json()
        after = search_db("SELECT count(*) FROM menu")
        after1 = search_db("SELECT count(*) FROM menu_ins")
        after2 = search_db("SELECT count(*) FROM menu_ingr")
        diff1 = search_db("SELECT count(*) FROM (SELECT DISTINCT * FROM
instructions WHERE id = '{}') as x".format(data["name"]))
        diff2 = search_db("SELECT count(*) FROM (SELECT DISTINCT * FROM ingredient
WHERE id = '{}') as x".format(data["name"]))
        self.assertTrue(before[0][0]+1 == after[0][0])
        self.assertTrue(before1[0][0]+diff1[0][0] == after1[0][0])
```

```
        self.assertTrue(before2[0][0]+diff2[0][0] == after2[0][0])
        search_db("DELETE FROM menu WHERE recipe = '{}' and username = '{}' and type
= '{}'".format(data["name"], cookies["id"], data["type"]))
        search_db("DELETE FROM menu_ins WHERE recipe = '{}' and username = '{}' and
type = '{}'".format(data["name"], cookies["id"], data["type"]))
        search_db("DELETE FROM menu_ingr WHERE recipe = '{}' and username = '{}' and
type = '{}'".format(data["name"], cookies["id"], data["type"]))
```

#Purpose: Test an addition to the menu with an invalid user id
#Input: invalid user id, valid meal type, valid recipe id
#Expected output: menu, menu_ins, and menu_ingr table in the database should remain unchanged

```
    def test_add_menu_invalid_cookie(self):
        before = search_db("SELECT count(*) FROM menu")
        before1 = search_db("SELECT count(*) FROM menu_ins")
        before2 = search_db("SELECT count(*) FROM menu_ingr")
        cookies = {
            'id': 'rjian'
        }
        data = {
          'type': 'b',
          'name': '64d3724048'
        }
        response = requests.post('http://localhost:1880/addToMenu', cookies=cookies,
data=data).json()
        after = search_db("SELECT count(*) FROM menu")
        after1 = search_db("SELECT count(*) FROM menu_ins")
        after2 = search_db("SELECT count(*) FROM menu_ingr")
        self.assertTrue(before[0][0] == after[0][0])
        self.assertTrue(before1[0][0] == after1[0][0])
        self.assertTrue(before2[0][0] == after2[0][0])
```

#Purpose: Test an addition to the menu with an invalid recipe id
#Input: valid user id, valid meal type, invalid recipe id
#Expected output: menu, menu_ins, and menu_ingr table in the database should remain unchanged

```
    def test_add_menu_invalid_recipe(self):
        before = search_db("SELECT count(*) FROM menu")
        before1 = search_db("SELECT count(*) FROM menu_ins")
        before2 = search_db("SELECT count(*) FROM menu_ingr")
        cookies = {
            'id': 'jianr'
        }
        data = {
```

```
        'type': 'b',
        'name': '666666666'
      }
      response = requests.post('http://localhost:1880/addToMenu', cookies=cookies,
data=data).json()
      after = search_db("SELECT count(*) FROM menu")
      after1 = search_db("SELECT count(*) FROM menu_ins")
      after2 = search_db("SELECT count(*) FROM menu_ingr")
      self.assertTrue(before[0][0] == after[0][0])
      self.assertTrue(before1[0][0] == after1[0][0])
      self.assertTrue(before2[0][0] == after2[0][0])

  #Purpose: Test an addition to the menu with an invalid meal type
  #Input: valid user id, invalid meal type, valid recipe id
  #Expected output: menu, menu_ins, and menu_ingr table in the database should
remain unchanged
  def test_add_menu_invalid_meal_type(self):
      before = search_db("SELECT count(*) FROM menu")
      before1 = search_db("SELECT count(*) FROM menu_ins")
      before2 = search_db("SELECT count(*) FROM menu_ingr")
      cookies = {
          'id': 'jianr'
      }
      data = {
        'type': 'a',
        'name': '64d3724048'
      }
      response = requests.post('http://localhost:1880/addToMenu', cookies=cookies,
data=data).json()
      after = search_db("SELECT count(*) FROM menu")
      after1 = search_db("SELECT count(*) FROM menu_ins")
      after2 = search_db("SELECT count(*) FROM menu_ingr")
      #print(after)
      self.assertTrue(before[0][0] == after[0][0])
      self.assertTrue(before1[0][0] == after1[0][0])
      self.assertTrue(before2[0][0] == after2[0][0])
```

#2 Unit Tests for GetMacro method

```
class GetMacroTest(unittest.TestCase):

    #Purpose: Test for getting all values of 5 macronutrients for a valid user id and meal type
    #Input: valid user id, valid meal type
    #Expected output: if menu is empty for given type, return a list of null values. Otherwise, the values are not null
    def test_get_macro(self):
        cookies = {
            'id': 'jianr'
        }
        data = {
          'type': 'b'
        }
        response = requests.post('http://localhost:1880/get_macro', cookies=cookies, data=data).json()
        self.assertTrue(len(response["nutrients"]) == 5)
        count = search_db("SELECT count(*) FROM menu WHERE type = '{}'".format(data["type"]))
        if count[0][0] == 0:
            for i in response["key"]:
                self.assertTrue(response[i] == None)
        else:
            for i in response["key"]:
                self.assertTrue(response[i] != None)

    #Purpose: Test for getting all values of 5 macronutrients for empty inputs
    #Input: empty user id, empty meal type
    #Expected output: get a list of null values
    def test_get_macro_empty(self):
        cookies = {
            'id': ''
        }
        data = {
          'type': ''
        }
        response = requests.post('http://localhost:1880/get_macro', cookies=cookies, data=data).json()
        self.assertTrue(list(response[0].keys())[0] == "username")

    #Purpose: Test for getting all values of 5 macronutrients for invalid user id
    #Input: invalid user id, valid meal type
    #Expected output: get a list of null values
```

```
    def test_get_macro_invalid_cookies(self):
        count = search_db("SELECT count(*) FROM users")
        cookies = {
            'id': 'rjian'
        }
        data = {
          'type': 'b'
        }
        response = requests.post('http://localhost:1880/get_macro', cookies=cookies,
data=data).json()
        self.assertTrue(list(response[0].keys())[0] == "username")

    #Purpose: Test for getting all values of 5 macronutrients for invalid meal type
    #Input: valid user id, invalid meal type
    #Expected output: get a list of null values
    def test_get_macro_invalid_meal_type(self):
        count = search_db("SELECT count(*) FROM users")
        cookies = {
            'id': 'jianr'
        }
        data = {
          'type': 'a'
        }
        response = requests.post('http://localhost:1880/get_macro', cookies=cookies,
data=data).json()
        self.assertTrue(len(response["nutrients"]) == 5)
        for i in response["key"]:
            self.assertTrue(response[i] == None)
```

#3 Unit Tests for CreateAccount method

```
class CreateAccountTest(unittest.TestCase):

    #Purpose: Test an appropriate registration to the website
    #Input: valid user id, valid password
    #Expected output: users table in database should be updated with the new user info
    def test_create_account(self):
        before = search_db("SELECT count(*) FROM users")
        data = {
          'firstname': str(int(random.random()*100)),
          'lastname': '123456'
        }
        response = requests.post('http://localhost:1880/create_account', data=data)
        after = search_db("SELECT count(*) FROM users")
        self.assertTrue(before[0][0]+1 == after[0][0])
        search_db("DELETE FROM users WHERE username = '{}'".format(data["firstname"]))

    #Purpose: Test a registration with empty user id and empty password
    #Input: empty user id, empty password
    #Expected output: users table in database is unchanged
    def test_create_account_empty(self):
        before = search_db("SELECT count(*) FROM users")
        data = {
          'firstname': '',
          'lastname': ''
        }
        response = requests.post('http://localhost:1880/create_account', data=data)
        after = search_db("SELECT count(*) FROM users")
        self.assertTrue(before[0][0] == after[0][0])

    #Purpose: Test a registration with user id already registered and valid password
    #Input: duplicate user id, valid password
    #Expected output: users table in database is unchanged
    def test_create_account_duplicate_username(self):
        before = search_db("SELECT count(*) FROM users")
        data = {
          'firstname': 'jianr',
          'lastname': '132435'
        }
        response = requests.post('http://localhost:1880/create_account', data=data)
        after = search_db("SELECT count(*) FROM users")
        self.assertTrue(before[0][0] == after[0][0])

    #Purpose: Test a registration with user id with length 25, 26 and valid password
```

#Input: long user id, valid password

#Expected output: if user id length is below 25, users table updated with new account. Otherwise, table is unchaged

```
    def test_create_account_username_limit_25(self):
        before = search_db("SELECT count(*) FROM users")
        data = {
          'firstname': 'jianrjianrjianrjianrjianr1',
          'lastname': '132435'
        }
        response = requests.post('http://localhost:1880/create_account', data=data)
        after = search_db("SELECT count(*) FROM users")
        self.assertTrue(before[0][0] == after[0][0])
        before = search_db("SELECT count(*) FROM users")
        data = {
          'firstname': 'jianrjianrjianrjianrjianr',
          'lastname': '132435'
        }
        response = requests.post('http://localhost:1880/create_account', data=data)
        after = search_db("SELECT count(*) FROM users")
        self.assertTrue(before[0][0]+1 == after[0][0])
        search_db("DELETE FROM users WHERE username = '{}'".format(data["firstname"]))
```

#Purpose: Test a registration with valid user id and pasword with length 16, 17

#Input: valid user id, long password

#Expected output: if password length is below 16, users table updated with new account. Otherwise, table is unchanged

```
    def test_create_account_pwd_limit_16(self):
        before = search_db("SELECT count(*) FROM users")
        data = {
          'firstname': 'new',
          'lastname': '12345678901234567'
        }
        response = requests.post('http://localhost:1880/create_account', data=data)
        after = search_db("SELECT count(*) FROM users")
        self.assertTrue(before[0][0] == after[0][0])

        before = search_db("SELECT count(*) FROM users")
        data = {
          'firstname': 'new',
          'lastname': '1234567890123456'
        }
        response = requests.post('http://localhost:1880/create_account', data=data)
        after = search_db("SELECT count(*) FROM users")
        self.assertTrue(before[0][0]+1 == after[0][0])
```

```
search_db("DELETE FROM users WHERE username = '{}'".format(data["firstname"]))
```

#4 Unit Test for DeleteMenu method

```
class DeleteMenuTest(unittest.TestCase):

    #Purpose: Test a appropriate deletion from the menu
    #Input: valid user id, valid meal type, valid recipe id
    #Expected output: specified recipe should be removed in menu, menu_ins, and menu_ingr table in the database
    def test_delete_menu(self):
        cookies = {
            'id': 'jianr'
        }
        data = {
          'type': 'b',
          'name': '64d3724048'
        }
        response = requests.post('http://localhost:1880/addToMenu', cookies=cookies, data=data).json()
        before = search_db("SELECT count(*) FROM menu")
        before1 = search_db("SELECT count(*) FROM menu_ins")
        before2 = search_db("SELECT count(*) FROM menu_ingr")
        response = requests.post('http://localhost:1880/delete_menu', cookies=cookies, data=data)
        after = search_db("SELECT count(*) FROM menu")
        after1 = search_db("SELECT count(*) FROM menu_ins")
        after2 = search_db("SELECT count(*) FROM menu_ingr")
        diff1 = search_db("SELECT count(*) FROM (SELECT DISTINCT * FROM instructions WHERE id = '{}') as x".format(data["name"]))
        diff2 = search_db("SELECT count(*) FROM (SELECT DISTINCT * FROM ingredient WHERE id = '{}') as x".format(data["name"]))
        self.assertTrue(before[0][0]-1 == after[0][0])
        self.assertTrue(before1[0][0]-diff1[0][0] == after1[0][0])
        self.assertTrue(before2[0][0]-diff2[0][0] == after2[0][0])

    #Purpose: Test a deletion to the menu with an invalid user id
    #Input: invalid user id, valid meal type, valid recipe id
    #Expected output: menu, menu_ins, and menu_ingr table in the database should remain unchanged
    def test_delete_menu_invalid_cookie(self):
        cookies = {
            'id': 'rjian'
        }
        data = {
          'type': 'b',
          'name': '64d3724048'
```

```
        }
        response = requests.post('http://localhost:1880/addToMenu', cookies=cookies,
data=data).json()
        before = search_db("SELECT count(*) FROM menu")
        before1 = search_db("SELECT count(*) FROM menu_ins")
        before2 = search_db("SELECT count(*) FROM menu_ingr")
        response = requests.post('http://localhost:1880/delete_menu', cookies=cookies,
data=data)
        after = search_db("SELECT count(*) FROM menu")
        after1 = search_db("SELECT count(*) FROM menu_ins")
        after2 = search_db("SELECT count(*) FROM menu_ingr")
        self.assertTrue(before[0][0] == after[0][0])
```

#Purpose: Test a deletion to the menu with an invalid recipe id
#Input: valid user id, valid meal type, invalid recipe id
#Expected output: menu, menu_ins, and menu_ingr table in the database should remain unchanged

```
    def test_delete_menu_invalid_recipe(self):
        cookies = {
            'id': 'jianr'
        }
        data = {
          'type': 'b',
          'name': '666666666'
        }
        response = requests.post('http://localhost:1880/addToMenu', cookies=cookies,
data=data).json()
        before = search_db("SELECT count(*) FROM menu")
        before1 = search_db("SELECT count(*) FROM menu_ins")
        before2 = search_db("SELECT count(*) FROM menu_ingr")
        response = requests.post('http://localhost:1880/delete_menu', cookies=cookies,
data=data)
        after = search_db("SELECT count(*) FROM menu")
        after1 = search_db("SELECT count(*) FROM menu_ins")
        after2 = search_db("SELECT count(*) FROM menu_ingr")
        self.assertTrue(before[0][0] == after[0][0])
```

#Purpose: Test a deletion to the menu with an invalid meal type
#Input: valid user id, invalid meal type, valid recipe id
#Expected output: menu, menu_ins, and menu_ingr table in the database should remain unchanged

```
    def test_delete_menu_invalid_meal_type(self):
        cookies = {
            'id': 'jianr'
```

```
        }
        data = {
          'type': 'a',
          'name': '64d3724048'
        }
        response = requests.post('http://localhost:1880/addToMenu', cookies=cookies,
data=data).json()
        before = search_db("SELECT count(*) FROM menu")
        before1 = search_db("SELECT count(*) FROM menu_ins")
        before2 = search_db("SELECT count(*) FROM menu_ingr")
        response = requests.post('http://localhost:1880/delete_menu', cookies=cookies,
data=data)
        after = search_db("SELECT count(*) FROM menu")
        after1 = search_db("SELECT count(*) FROM menu_ins")
        after2 = search_db("SELECT count(*) FROM menu_ingr")
        self.assertTrue(before[0][0] == after[0][0])

if __name__ == '__main__':
    unittest.main(verbosity=2, exit=False)
```