CSC 648/848 Software Engineering – Spring 2022
Milestone 4 Beta Launch, QA and Usability Testing and Final Commitment for Product Features (P1 list)

1) Product summary

- Name: LinkedSF
- Registered students will be able to search jobs based on tech area, job position, and skills.
- Registered students will be able to apply for jobs.
- Registered students will be able to view company information.
- Registered students will be able to view their inbox and compose messages.
- Registered students will be able to log in and log out of the website.
- Registered employers will be able to create job posts.
- Registered employers will be able to view inbox and compose messages.
- Registered employers will be able to filter their own posts.
- Registered employers will be able to log in and log out of the website.
- Unique Feature: Creating alerts to notify job seekers/companies.
- URL:

2) Usability test plan

Test Objectives

Ensure the search engine returns appropriate job posts based on the filter selection and keywords. Any job posts that show up in the search result should be in the field of expertise that the user specifies. The job posts should also include any keyword from the search bar. The user should not see any job posts that are not in the field of their speciality. Any job posts that do not meet the user's requirements will not interest the user and will make it more difficult for the user to find a job of their choice.

Display job posts within an acceptable time frame. The response time between clicking the search button to displaying the posts should be minimal. If users are waiting for results, they are not looking at job posts. The cumulative wait time may deter users away from the service and search on other sites. If there are less applicants on the site, companies may not post their openings on the site. Users tend to lean towards using responsive and snappy services.

Test Background and Setup

System Setup:

Use the application to measure the number of clicks, number of pages visited, and time of completion. Use screen recording to manually count the number of errors the test user made.

Starting Point:

The starting point will be the student home page. This will be the same page a student will see after they are logged into their student account.

Intended Users:

The intended users are students who are in their last semester of completing their Bachelor degree.

URL:

Root URL subject to change every time the server restarts/StudentHomepage.html (student homepage url). Count the completion rate among the users. Measure the average clicks and pages visited per user.

Usability Task Description

Search for a IoT job with and without the use of the keyword “blockchain”. The benchmark time frame should be under 5 minutes.

Effectiveness:

Evaluate how many of them can complete a search within a set amount of time. Determine the rate of success for the intended users. Test how many errors the average user makes while completing the task.

Efficiency:

Measure the average time it takes for users to complete a search. Count the number of clicks and the number of screens the average user would have to go through to complete the task.

Lickert Subjective Test

Question 1:

It was easy to search for a job posting.

☐ ☐ ☐ ☐ ☐

Very Difficult Average Very Easy

Comments:

Question 2:

The process of the search was very efficient.

☐ ☐ ☐ ☐ ☐

Very inefficient Average Very efficient

Comments:

Question 3:

In a real application, you would use the search function.

☐ ☐ ☐ ☐ ☐

Strongly Disagree Neutral Strongly Agree

Comments:

3) QA test plan

- Test objectives:
  Ensure the search engine returns correct job posts according to the user's specification. Also, job posts should be displayed correctly within a human acceptable short latency.
- HW and SW setup:
  - HW:
    - server: Amazon AWS t2.micro
  - SW:
    - OS: Windows 10
    - browser: Chrome, Firefox
    - web server: Apache2 on Ubuntu 18.04
    - URL: (*changes every time the server restarts*)
- Feature to be tested:
  The search function in the student home page.
- QA Test plan:

Chrome	Title	Desciption	Input	Expected output	Result
#1	No keywords	Search under single field with no keywords provided in search box	field: IoT keyword:	All job posts under IoT field	PASS
#2	With keywords	Search under single field with keywords provided in search box	filed: IoT keyword: blockchain	All job posts that contain blockchain in the title under IoT field	PASS
#3	SQL injection	Search under single field with keywords that contains typical SQL injection provided in search box	field: IoT keyword: blockchain OR 1=1	No results since no matches.	PASS

Firefox	TItle	Desciption	Input	Expected Output	Result

#1	No keywords	Search under single field with no keywords provided in search box	field: IoT keyword	All job posts under IoT field	PASS
#2	With keywords	Search under single field with keywords provided in search box	filed: IoT keyword: blockchain	All job posts that contain blockchain in the title under IoT field	PASS
#3	SQL injection	Search under single field with keywords that contains typical SQL injection provided in search box	field: IoT keyword: blockchain OR 1=1	No results since no matches.	PASS

4) Code Review

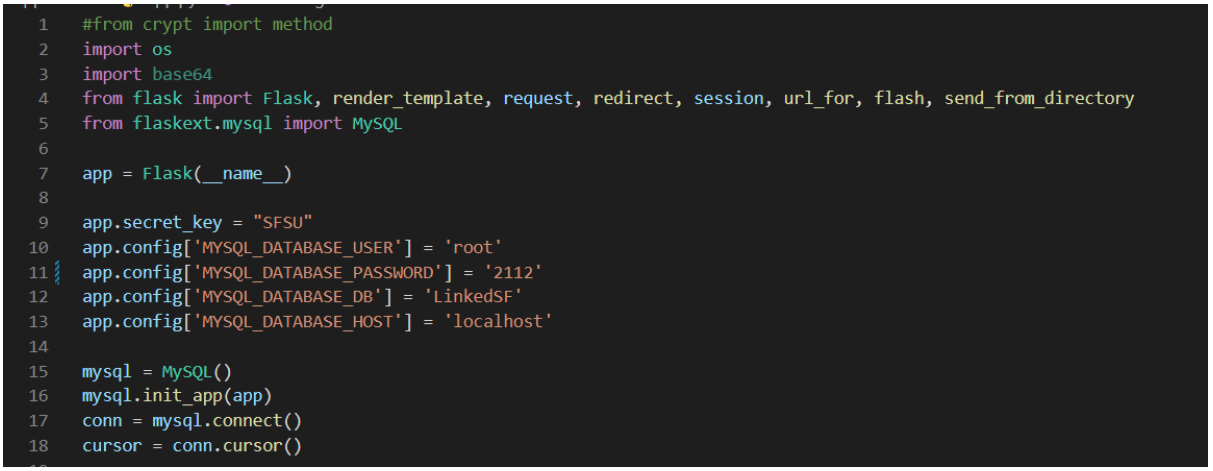
There is not much of a coding style we have implemented because we are working in Python. Our coding style involves indentation in this case. Our html pages are aligned with any other html styled page. As for the variable names, we have a combination of camelcase where instead of the first letter being lowercase, it is uppercase. Each variable is spelled thoroughly with no shorthand.

Sample #1 Review:

```python
@app.route('/StudentHomePage.html', methods=['GET', 'POST'])
def SearchJob():
    if request.method == "POST":
        if request.form['submit'] == "submit_search":
            Job_Field = request.form['Job_Field']
            Search_Value = request.form['Search']
            if Search_Value == '':
                cursor.execute("SELECT * FROM JobPost")
                conn.commit()
                data = cursor.fetchall()
            else:
                cursor.execute('SELECT * FROM JobPost WHERE Job_Title LIKE %s AND Job_Field = %s', ('%' + Search_Value + '%', Job_Field,))
                conn.commit()
                data = cursor.fetchall()
            # all in the search box will return all the tuples
            return render_template('StudentHomePage.html', data = data)
        if request.form['submit'] == "submit_apply":
                buttonID = request.form['buttonID']
                cursor.execute('INSERT IGNORE INTO applied (FK_Postid, FK_JobSeekerid) Values (%s, %s)', (int(buttonID), int(session['id'])))
                conn.commit()
                flash("You Have Successfully Applied")
    return render_template("StudentHomePage.html")
```

Code follows the syntax required for Python. Code is spaced at the appropriate locations to ensure readability. SQL statements appear to follow their appropriate syntax as well. Any variable related to the database follows the same naming scheme, however, for any program declared variables, such as “buttonID” follows a different naming scheme. In the code provided, there is only one comment. There is not a header above the function to indicate what is going on either.

Sample #2 Review:



```python
#from crypt import method
import os
import base64
from flask import Flask, render_template, request, redirect, session, url_for, flash, send_from_directory
from flaskext.mysql import MySQL

app = Flask(__name__)

app.secret_key = "SFSU"
app.config['MYSQL_DATABASE_USER'] = 'root'
app.config['MYSQL_DATABASE_PASSWORD'] = '2112'
app.config['MYSQL_DATABASE_DB'] = 'LinkedSF'
app.config['MYSQL_DATABASE_HOST'] = 'localhost'

mysql = MySQL()
mysql.init_app(app)
conn = mysql.connect()
cursor = conn.cursor()
```

For the beginning of the file, there are no headers to indicate what kind of file this is. It would be helpful to add a header indicating the file name, what it is about, and who has worked on it.

Sample #3 Review:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CompanyHomePage</title>
    <link rel="stylesheet" href="static/style.css"/>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kW

</head>
```

Each HTML file contains the same header. Perhaps it would be useful to refactor the HTML files into parts such as navbar, homepage, login, and have one main file with the header statement.

5) Self-check on best practices for security – ½ page

1. **Login Validations:** Jobseekers/Companies are not allowed to access their prospective pages without a valid login. Checks the information typed into the fields with the users in the database, if their entry matches, it will allow access and store their session.
2. **Form Validation:** When users create an account, they are required to meet certain criteria. An example would be using a valid email address, creating a username that contains 8 or more characters, and not allowing the creation of an account if these requirements are not met.
3. **SQL Injection:** Authorized/Unauthorized users cannot type SQL commands into the login field and be granted access to sensitive information stored in the database. When trying to type these commands in the login, it will not display any information along with giving the error message of an invalid login. Our login validation is also responsible for preventing injection.
4. **Securing App.py Routes:** When deploying a website, we are making the FLASK server publicly accessible. To avoid people from calling our routes, we use basic authentication to avoid any attacks or altering of code.

Self-check: Adherence to original Non-functional specs:

1. Applications shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0 (some may be provided in the class, some may be chosen by the student team but all tools and servers have to be approved by class CTO). **(DONE)**

2. Application shall be optimized for standard desktop/laptop browsers e.g., must render correctly on the two latest versions of two major browsers **(DONE)**

3. Selected application functions must render well on mobile devices **(DONE)**

4. Data shall be stored in the team's chosen database technology on the team's deployment server. **(DONE)**

5. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users. **(DONE)**

6. The language used shall be English. **(DONE)**

7. Application shall be very easy to use and intuitive. **(DONE)**

8. Google maps and analytics shall be added **(N/A, DONE)**

9. No email clients shall be allowed. You shall use webmail. **(N/A, DONE)**

10. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in the UI. **(DONE)**

11. Site security: basic best practices shall be applied (as covered in the class) **(DONE)**

12. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development **(DONE)**

13. The website shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Spring 2022. For Demonstration Only" at the top of the WWW page. (Important so not to confuse this with a real application). **(DONE)**