

Logan Hammond

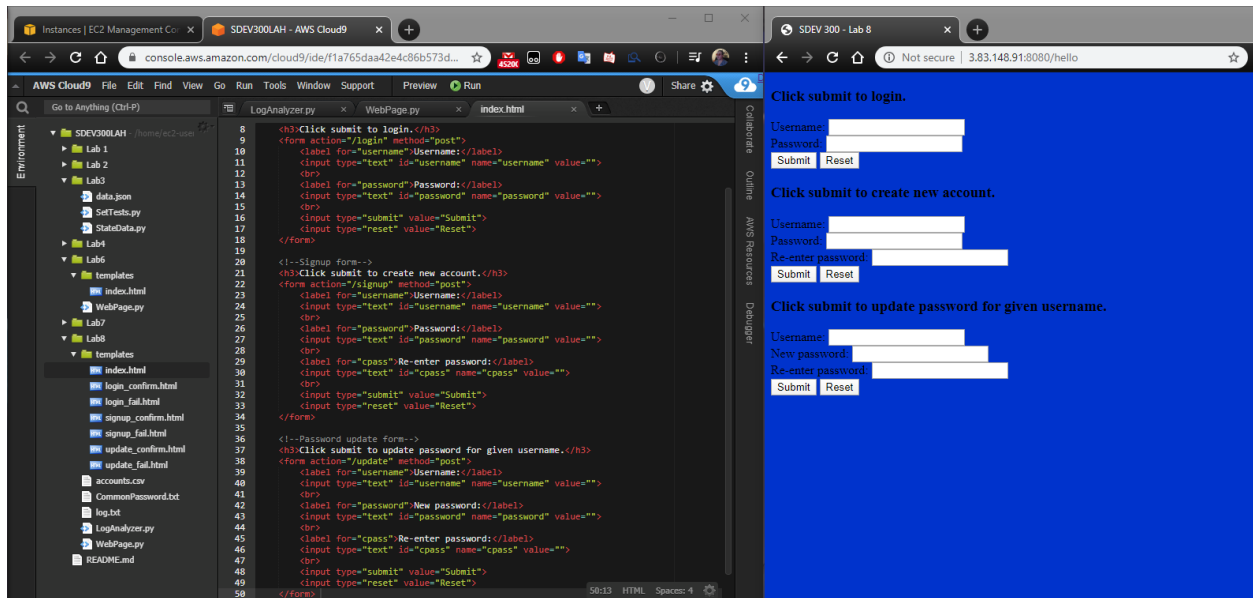
3 March 2020

SDEV 300, Fair

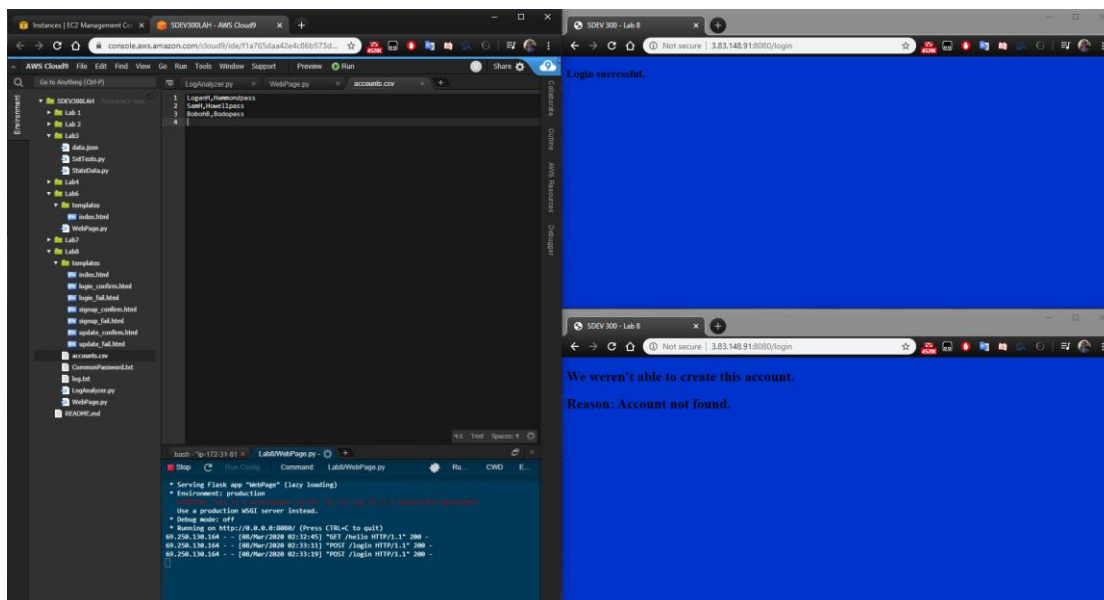
Lab 8 Results Document

Website

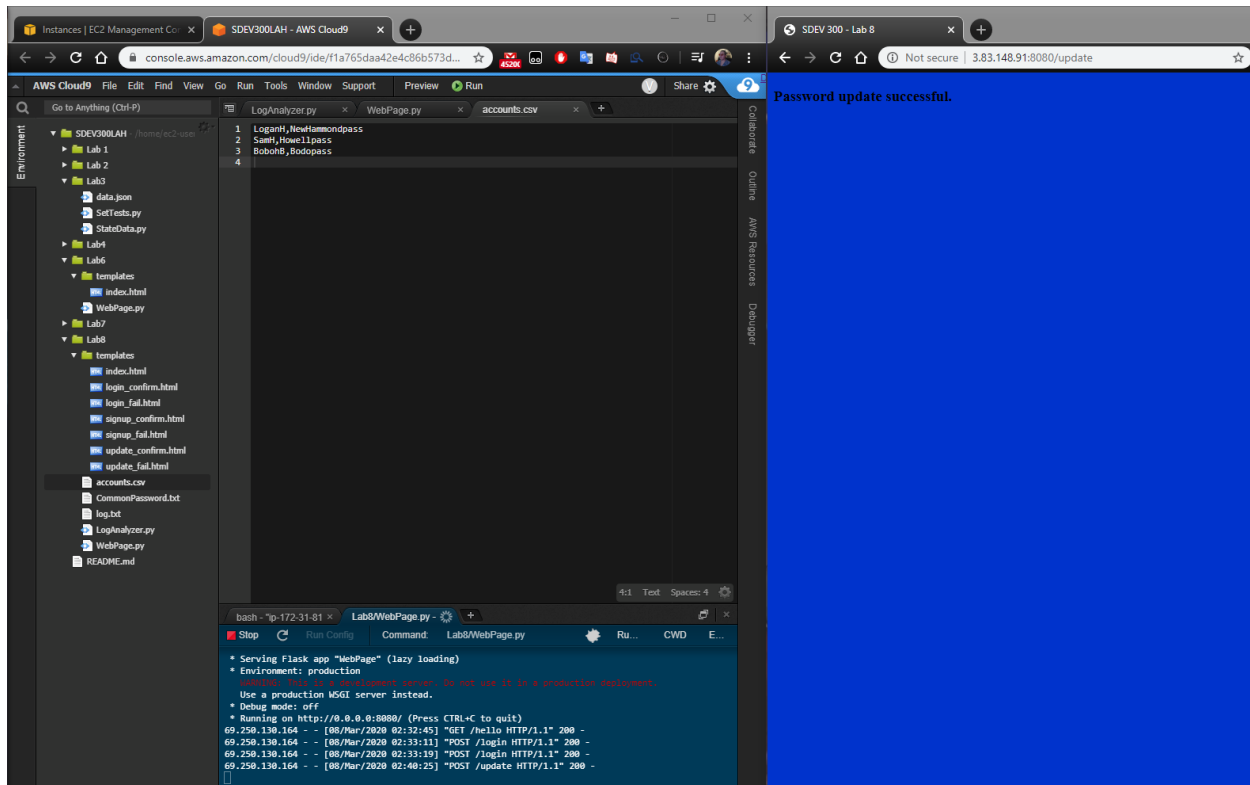
A – B



Main page of website (/hello) shows 3 forms to the user: a login form, a register form, and a password update form. Left image shows the `index.html` file centered on these forms.

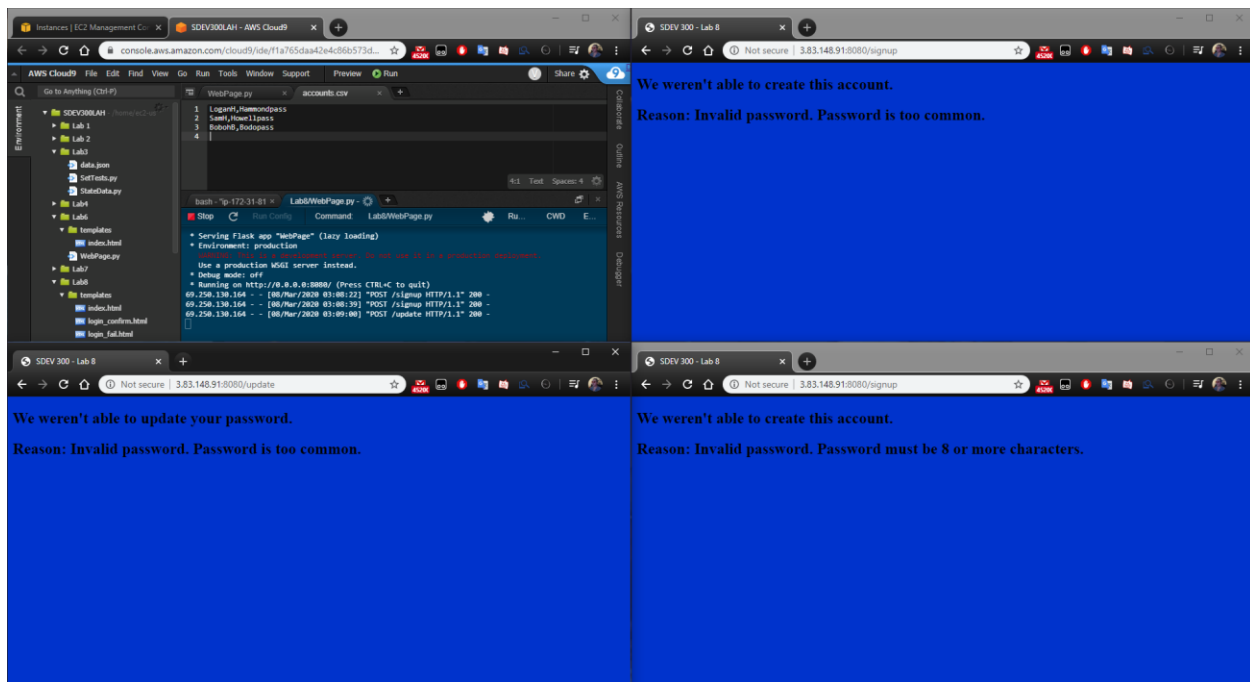


Submitting the first form will allow the user to login with a set of credentials. If the credentials are found in the master list of accounts (accounts.csv) then the user is redirected to a separate page confirming the login was successful. If the credentials were not found the user is submitted to a page alerting them that the credentials were not valid i.e. the account was not found in the master list of accounts. The credentials used were “LoganH” and “Hammondpass” for the username and password respectively.



From the homepage a user may submit their username and a new password for their account. When this is done the master list updates accordingly without changing the chronological ordering of the accounts.

C



D

```

34 reader = csv.reader(f, delimiter=",")
35 for line in reader:
36     accounts.append(line)
37
38 # Get data from form
39 username = request.form["username"]
40 password = request.form["password"]
41 user_combo = [username, password]
42
43 # Search for user in account list.
44 if user_combo not in accounts:
45     reason_str = "Account not found."
46
47 # Return failure page is reason_str is not empty. Otherwise return confirm.
48 if reason_str:
49     # If login failed append instance of failure to log file.
50     with open("Lab8/log.txt", "a") as lf:
51         log_line = "{}\t{}\n".format(datetime.now(), request.remote_addr)
52         lf.write(log_line)
53     if reason_str:
54         return render_template("signup_fail.html", reason = reason_str)
55     return render_template("login_confirm.html")

```

```

1 2020-03-07 23:25:17.140028 68.251.131.165
2 2020-03-07 23:26:17.140028 69.250.130.164
3 2020-03-07 23:27:17.140028 68.251.131.165
4 2020-03-07 23:28:17.140028 70.252.132.166
5 2020-03-07 23:29:17.140028 69.250.130.164
6 2020-03-07 23:30:17.140028 68.251.131.165
7 2020-03-07 23:31:17.140028 70.252.132.166
8 2020-03-07 23:32:17.140028 69.250.130.164
9 2020-03-07 23:33:17.140028 68.251.131.165
10 2020-03-07 23:33:17.140028 71.253.134.167
11 2020-03-07 23:33:47.140028 71.253.134.167
12 2020-03-07 23:34:17.140028 71.253.134.167
13 2020-03-07 23:34:47.140028 71.253.134.167
14 2020-03-07 23:35:17.140028 71.253.134.167
15 2020-03-07 23:35:47.140028 71.253.134.167
16 2020-03-07 23:36:17.140028 71.253.134.167
17 2020-03-07 23:36:47.140028 71.253.134.167
18 2020-03-07 23:37:17.140028 71.253.134.167
19 2020-03-07 23:37:47.140028 71.253.134.167
20 2020-03-07 23:38:17.140028 71.253.134.167
21 2020-03-07 23:38:47.140028 71.253.134.167
22

```

Upon a failed login attempt the program logs the date and time of the failed attempt as well as the IP address that the attempt originated from. Note: The log shown above is manufactured but based on real recorded logs in order too maintain anonymity and showcase functionally of the log analyzer in the next section.

E

```

1 # Author: Logan Kennedy, lskenned@student.umuc.edu
2 # Source File: LogAnalyzer.py
3 # Description: Script which analyzes log of website and creates file
4 # "analyzed_log.txt" which contains a human-readable analysis
5 # of the file.
6 # If:
7 # AWS Cloud9
8 from collections import Counter
9 from datetime import datetime, timedelta
10 from ipgeotools.databases.noncommercial import DbIpCity
11
12 log_list = [] # List of each individual log.
13 ip_list = [] # List of every unique ip address in log.
14 ip_count_delta_list = [] # List of each ip and how many times it was in log.
15 timestamp_delta_list = [] # List of each unique ip and the change in time
16 # between its first and last occurrence.
17 latlong_list = [] # List of latitudes and longitudes.
18 dt_format = "%Y-%m-%d %H:%M:%S.%f"
19
20 # Create a list of each individual log, and each unique ip address.
21 with open("Lab8/log.txt", "r") as lf:
22     for log in lf:
23         date_time, ip_addr = log.split("\t")
24         ip_addr = ip_addr.replace("\n", "")
25         log_list.append((date_time, ip_addr))
26         if ip_addr not in ip_list:
27             ip_list.append(ip_addr)
28
29 # Count the number of times and time delta between first and last occurrence of
30 # each unique ip address.
31 for unique_ip in ip_list:
32     unique_ip_count = 0
33     timestamp_list = []
34     for log in log_list:
35         if unique_ip == log[1]:
36             unique_ip_count += 1
37             timestamp_list.append(log[0])
38     dt_start = datetime.strptime(timestamp_list[0], dt_format)
39     dt_end = datetime.strptime(timestamp_list[-1], dt_format)
40     dt_delta_mins = int((dt_end - dt_start).total_seconds() / 60)
41
42 # Get lat/long of each unique ip address.
43 response = DbIpCity.get(unique_ip, api_key="free")
44 lat_long = "{}/{}".format(response.latitude, response.longitude)
45
46 # Create delta for time between first and last occurrence of unique ip
47 dt_delta_mins = dt_delta_mins
48
49 # Print out the results
50 print("IP count list")
51 print(ip_count_delta_list)
52 print(ip_list)
53 print(timestamp_delta_list)
54 print(latlong_list)
55 print(dt_delta_mins)
56 print("IP count list")
57 print(ip_count_delta_list)
58 print(ip_list)
59 print(timestamp_delta_list)
60 print(latlong_list)
61 print(dt_delta_mins)

```

```

1 2020-03-07 23:25:17.140028 68.251.131.165
2 2020-03-07 23:26:17.140028 69.250.130.164
3 2020-03-07 23:27:17.140028 68.251.131.165
4 2020-03-07 23:28:17.140028 70.252.132.166
5 2020-03-07 23:29:17.140028 69.250.130.164
6 2020-03-07 23:30:17.140028 68.251.131.165
7 2020-03-07 23:31:17.140028 70.252.132.166
8 2020-03-07 23:32:17.140028 69.250.130.164
9 2020-03-07 23:33:17.140028 68.251.131.165
10 2020-03-07 23:33:17.140028 71.253.134.167
11 2020-03-07 23:33:47.140028 71.253.134.167
12 2020-03-07 23:34:17.140028 71.253.134.167
13 2020-03-07 23:34:47.140028 71.253.134.167
14 2020-03-07 23:35:17.140028 71.253.134.167
15 2020-03-07 23:35:47.140028 71.253.134.167
16 2020-03-07 23:36:17.140028 71.253.134.167
17 2020-03-07 23:36:47.140028 71.253.134.167
18 2020-03-07 23:37:17.140028 71.253.134.167
19 2020-03-07 23:37:47.140028 71.253.134.167
20 2020-03-07 23:38:17.140028 71.253.134.167
21 2020-03-07 23:38:47.140028 71.253.134.167
22

```

```

1 71.253.134.167 had 12 failed login attempts in a 5 minute period.
2 71.253.134.167 has a lat/long of 40.7127281/-74.0060152.
3
4

```

```

IP count list
[["68.251.131.165", 4, 8, "43.6355829/-84.2472317"],
["69.250.130.164", 3, 6, "39.3819338/-76.4573884"],
["70.252.132.166", 2, 3, "39.1803695/-84.5781416"],
["71.253.134.167", 12, 5, "40.7127281/-74.0060152"]]

```

LogAnalyzer.py is a script which parses the log file and outputs noteworthy data in a human-readable format. If no noteworthy data is found that is reported to the user. Noteworthy for this script is defined as an IP address appearing more than 10 times within a period of 5 minutes. Shown at the bottom is the complete list of each unique IP address and its relevant data which includes the number of times it appears, the time delta (time between first and last appearance) and the latitude and longitude from which the IP originates.

Decryption

a) Dots, dashes, and slashes; spaces => Morse code.

Given: - / .. - . - . - / .. - - - - - / - . - . - . - / .. - - - - . / .. - . - . - . - . - . - . - . - - -

Solution: THIS SDEV 300 CLASS HAS SOME STRANGE REQUESTS.

b) Contains a-z, A-Z, 0-9; no spaces => (probably) Base64.

Solution: So this is base64. Now I know.

c) Formatted like plain language; apparent repetition; (probably) Simple cipher/cryptogram.

Solution: BEGIN KEY I AM SO CLEVER NO ONE COULD POSSIBLY FIGURE THIS OUT END KEY