Sprint 2 - Manual Testing Document

This document will outline the procedure and results of manually testing each story as specified in our sprint planning document.

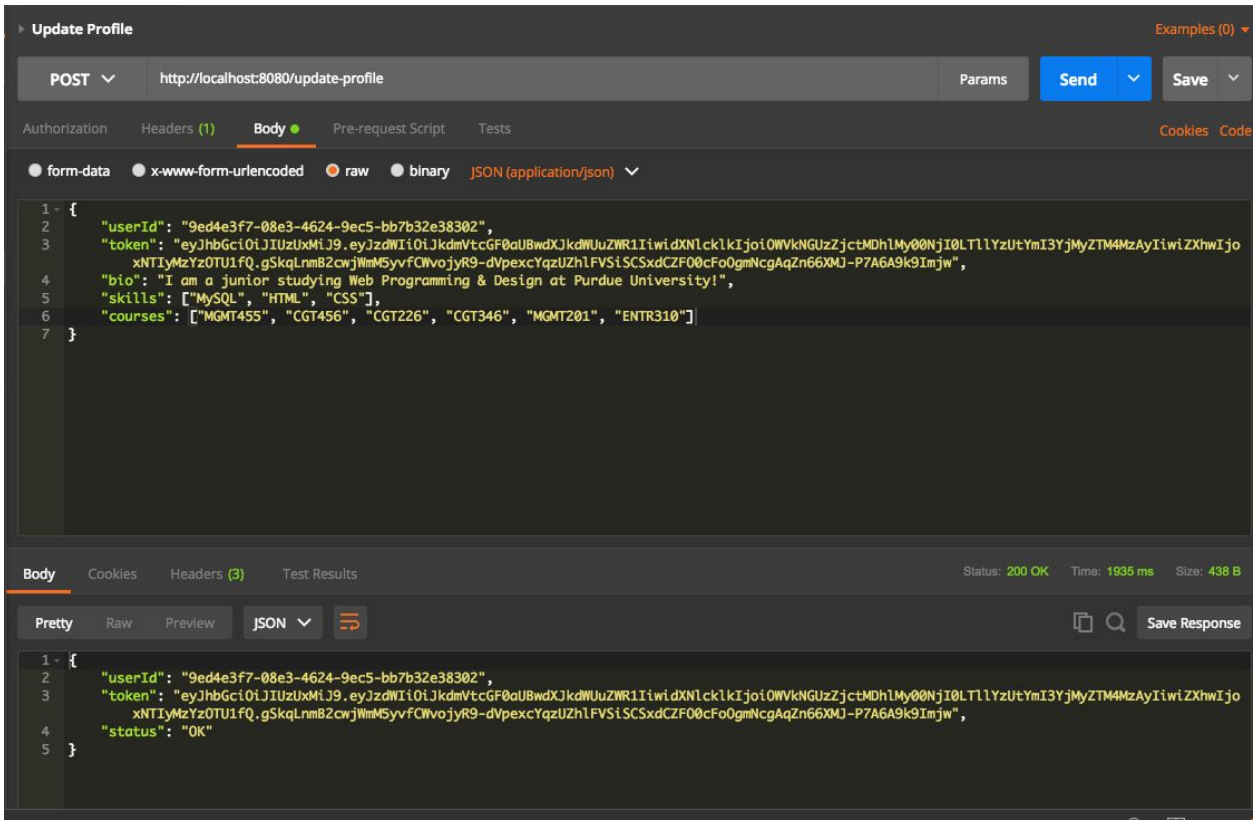
Testing Set One: Updating a user's profile by adding or removing skills and courses, as well as updating their personal biography.

Testing Procedure (Backend): This is the protocol used to test profile updating functionality from a backend perspective.

- Create a **POST** request on Postman that hits the endpoint /update-profile
- Using an existing user and token in our database, create a request body that contains fields for bio, skills, and courses.
- Send the request via Postman and observe the response body, which should be a 200 OK.
- With the same user and token, send another request with fields containing an updated bio, as well as the same skills and courses fields minus one or two objects. This signifies the removal of a skill or course.

Test results and follow up:

1. Requests made to add new skills, courses, and a biography were successful (can be observed via Postman and MySQL workbench). The following images will demonstrate changes to biography and courses, but the same results can be observed for skills as well.



This image shows the request body sent to the /update-profile endpoint with the necessary fields (a userId, token, bio, skills, and courses). The response body is also shown, showing a 200 OK status.

Result Grid | Filter Rows: Search | Export:

user_id	course_id
07852045-baa5-4794-86f5-8d56dcec93ed	c1443a3f-5317-4536-9551-ce206b6cdb90
07852045-baa5-4794-86f5-8d56dcec93ed	7d69cab0-0cef-4f79-a76b-aadfa94a5671
07852045-baa5-4794-86f5-8d56dcec93ed	418f783f-8a39-4cbc-8ecb-cdb1e5ab9b65
07852045-baa5-4794-86f5-8d56dcec93ed	52e53f3d-ccf4-4399-a87a-fb38b0732b3f
07852045-baa5-4794-86f5-8d56dcec93ed	30191a30-d192-4a84-816a-d6b584bb8903
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	35c5bce2-5edc-44ef-ad64-6784d5448cd3
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	a1fc8b22-8335-4441-829d-4758ffbe7dda
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	dbb17246-abc3-4195-bf88-9000abdf2855
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	5107df8f-cb13-4de1-95f3-a60e41275f98
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	52e53f3d-ccf4-4399-a87a-fb38b0732b3f
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	3d22f7aa-bea7-49e7-92c6-209b91926d62

We can observe that this request went through successfully by examining the user_courses database table. Notice that the user_id **<9ed4e3f7-08e3-4624-9ec5-bb7b32e38302>** associated with the user that is updating his/her profile now is linked with the course_ids. For reference, we added six courses in the POST request, as we can note 6 entries associated with the user_id in this table.

100% 1:36 1 error found

Result Grid | Filter Rows: Search | Edi

course_id	course_name
30191a30-d192-4a84-816a-d6b584bb8903	CS307
35c5bce2-5edc-44ef-ad64-6784d5448cd3	CGT456
3d22f7aa-bea7-49e7-92c6-209b91926d62	MGMT201
418f783f-8a39-4cbc-8ecb-cdb1e5ab9b65	COM217
5107df8f-cb13-4de1-95f3-a60e41275f98	ENTR310
52e53f3d-ccf4-4399-a87a-fb38b0732b3f	MGMT455
7d69cab0-0cef-4f79-a76b-aadfa94a5671	MA262
a1fc8b22-8335-4441-829d-4758ffbe7dda	CGT346
c1443a3f-5317-4536-9551-ce206b6cdb90	CS348
dbb17246-abc3-4195-bf88-9000abdf2855	CGT226
NULL	NULL

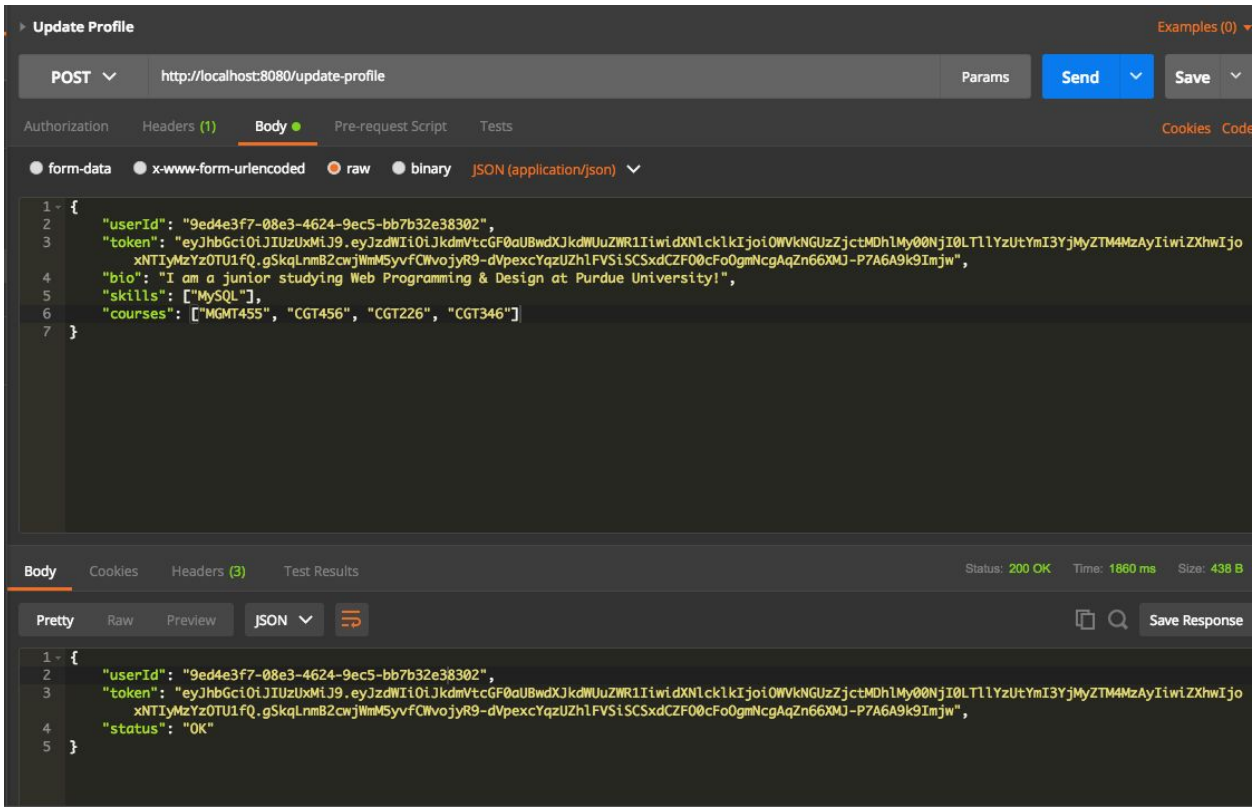
As a sanity check, here is the courses table that shows the courses that were added as well as their respective course_ids. This same result can also be observed for the skills and user_skills table (but is not shown because of redundancy).

Result Grid | Filter Rows: Search | Edit: | Export/Import:

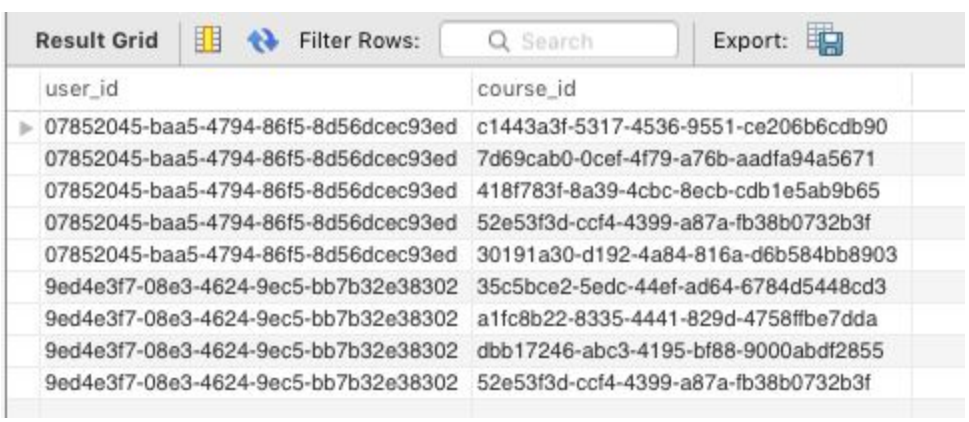
user_id	full_name	bio	major	grad_year
07852045-baa5-4794-86f5-8d56dcec93ed	Terry Lam	I am a junior studying CS at Purdue University!	Computer Science	2019
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	Divya Vempati	I am a junior studying Web Programming & Desi...	Web Programming and Design	2018
NULL	NULL	NULL	NULL	NULL

This image demonstrates that their biography was also updated in the profiles table.

2. Requests to remove skills and courses were also successful. Again, these images will demonstrate removal of courses, but the same results can be observed for skills too (left out due to redundancy)



Here is the same request, but notice that we have now taken out HTML and CSS from the skills, and removed a few enrolled courses. We still receive a 200 OK in the response body.

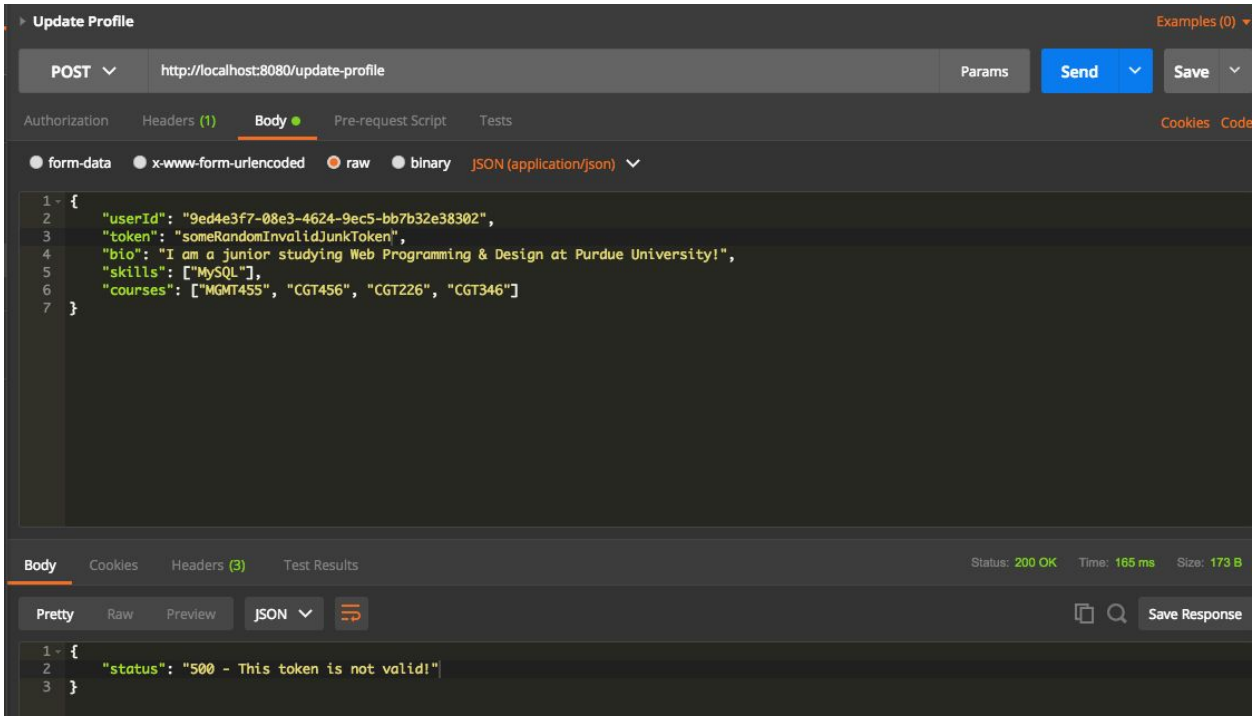


Result Grid | Filter Rows: Search | Export:

user_id	course_id
07852045-baa5-4794-86f5-8d56dcec93ed	c1443a3f-5317-4536-9551-ce206b6cdb90
07852045-baa5-4794-86f5-8d56dcec93ed	7d69cab0-0cef-4f79-a76b-aadfa94a5671
07852045-baa5-4794-86f5-8d56dcec93ed	418f783f-8a39-4cbc-8ecb-cdb1e5ab9b65
07852045-baa5-4794-86f5-8d56dcec93ed	52e53f3d-ccf4-4399-a87a-fb38b0732b3f
07852045-baa5-4794-86f5-8d56dcec93ed	30191a30-d192-4a84-816a-d6b584bb8903
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	35c5bce2-5edc-44ef-ad64-6784d5448cd3
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	a1fc8b22-8335-4441-829d-4758ffbe7dda
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	dbb17246-abc3-4195-bf88-9000abdf2855
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	52e53f3d-ccf4-4399-a87a-fb38b0732b3f

When we inspect the user_courses table again, we notice that there are now only FOUR entries for the associated user, since we removed two courses. As a sanity check, we can associate the course_ids back to the course table and see that they are correct.

3. As part of verification testing, requests with invalid tokens also were successfully denied.



As we can note from the image, API calls need to be verified with a correct user id and a valid associated JWT. This verification is done for every API call.

Testing Summary(Backend): All scenarios for updating a profile were tested. This includes adding and removing skills or courses, and changes made to a user's personal biography. Note that the bio, skills, and courses field in the request body are optional (can send one or more for successful requests). This is to account for users who only are updating parts of their profile at a time.

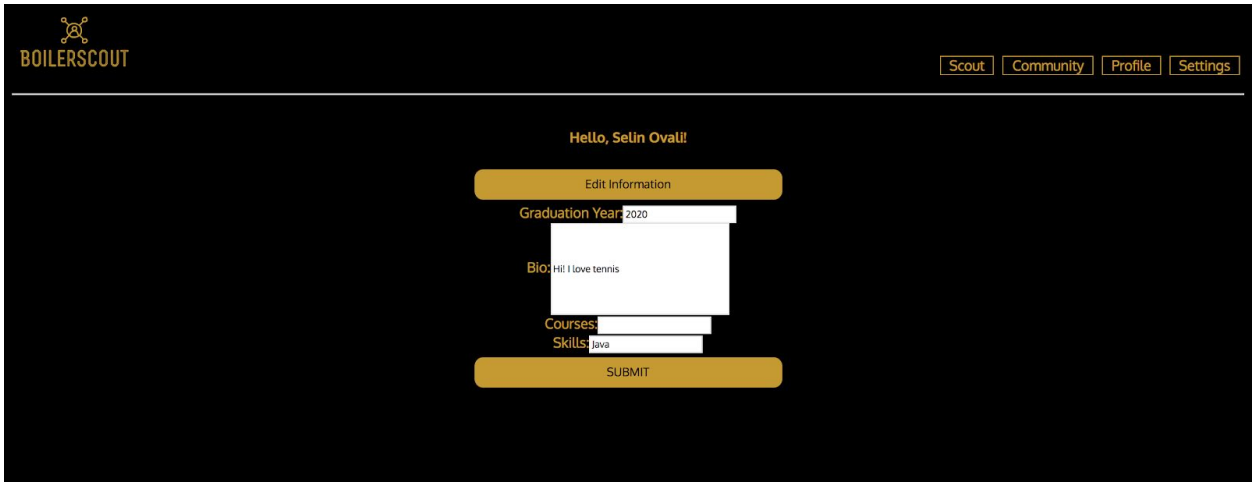
Testing Procedure (Frontend): This is the protocol used to test profile updating functionality from a frontend perspective.

- Logged in user's info should be displayed in Edit Profile.
- Changes to bio, courses, graduation year and skills are made on the Edit Profile page.
- Profile is checked to ensure changes to bio have been submitted successfully.
- Duplicate skills and courses are added in Edit Profile page, which should not be displayed on the Profile page.
- Profile is checked to make sure duplicates are not added to the profile information.

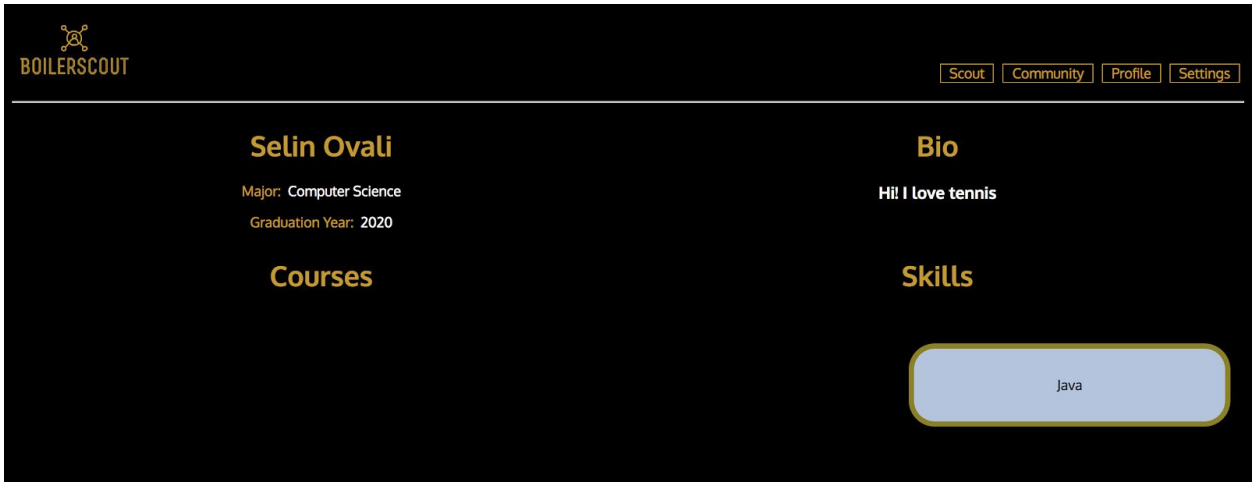
1. User should be displayed the already existing information they have on the Edit Profile page.

As the user I am logged in with only has filled their name and graduation year, those are displayed in Edit Profile.

2. Changes to profile information are made.

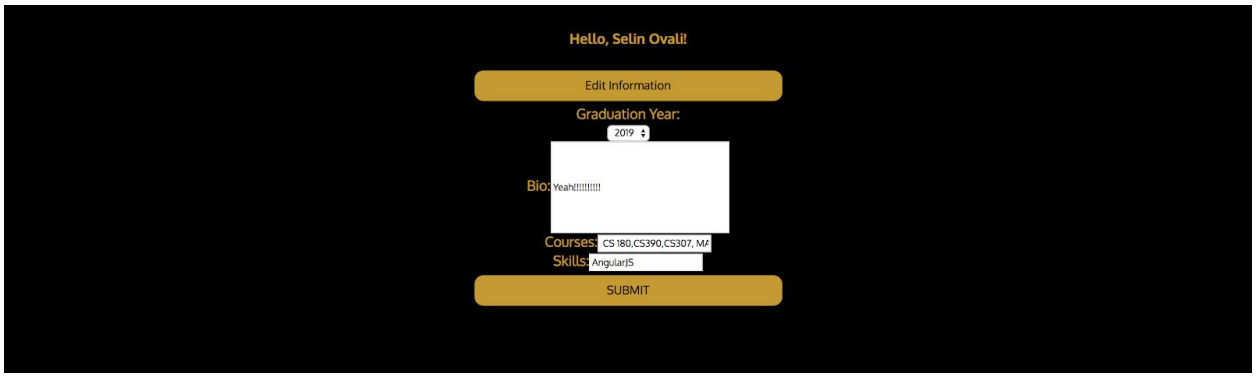


A bio and skill is added. SUBMIT is clicked. Direct to Profile to see if changes have been made.

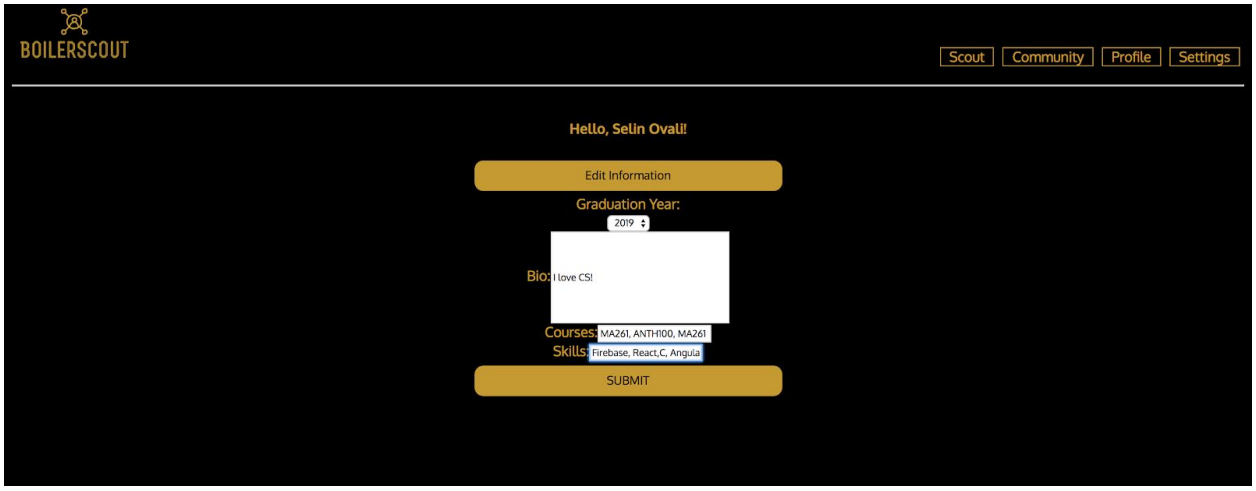


The user information we have added is successfully displayed in Profile.

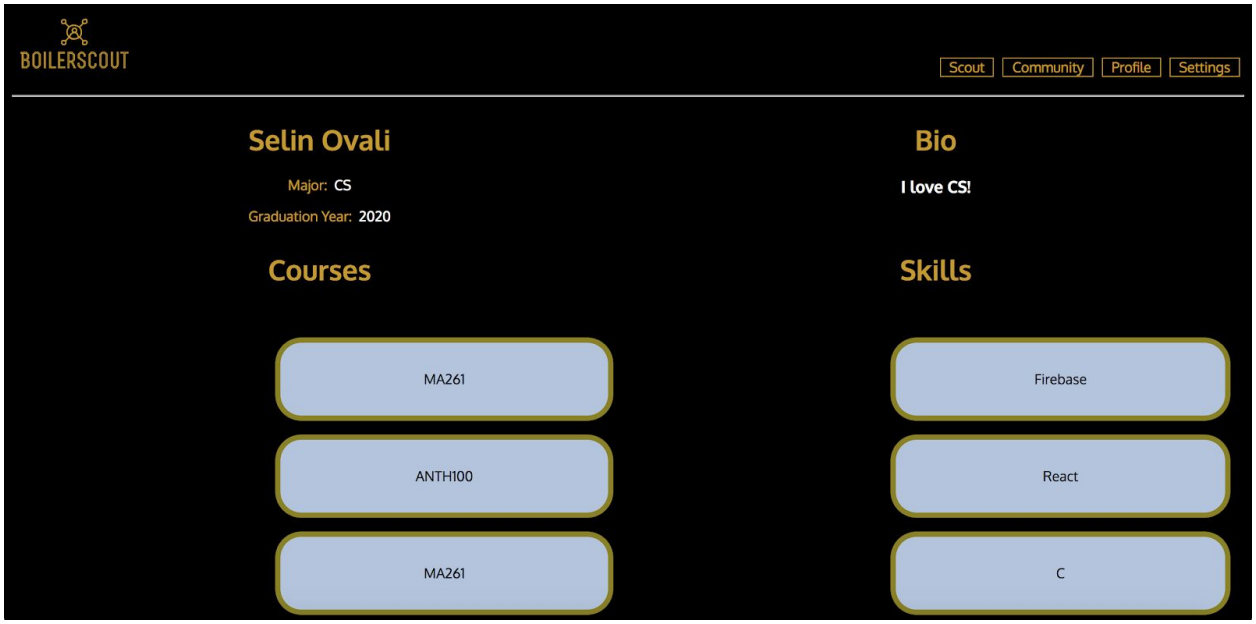
3. We can try this with adding courses, skills and bio.



We have the user's information and we change them.



We click SUBMIT and go to profile to make sure that profile is updated.



User information is updated correctly in Profile.

Testing Summary (Frontend): Each sign up field is updated accordingly to the information in Settings. All functionality works.

Testing Set Two: Searching for users based on three different criteria: name, an individual skill, or an enrolled course.

Testing Procedure (Backend): This is the protocol used to test basic search functionality from a backend perspective.

- Create a **GET** request on Postman that hits the endpoint /scout
- As request parameters, input an existing userId and token, as well as the type of query (name, skill, or course) and the query itself.
- Send three different requests via Postman to test each type of query and observe the response body, which should be a 200 OK and contain a list of associated users and their profile information.

Testing results and follow up

1. Requests made for “scouting” users based on a certain query were successful. The first test conducted was for queries based on name.

Scout

Examples (0)

GET http://localhost:8080/scout?userId=07852045-baa5-4794-86f5-8d56dcec93ed&token=eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsY... Params Send Save

Key	Value	Description
☑ userId	07852045-baa5-4794-86f5-8d56dcec93ed	
☑ token	eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsYW00NUBwdXJkdWUuZW...	
☑ type	name	
☑ query	e	
New key	Value	Description

Authorization Headers Body Pre-request Script Tests Cookies Code

The above image is an example URI with specified request parameters. In this case, we are searching for a name with the query just being the letter “e”.

Pretty Raw Preview JSON Save Response


```
{
    "query": [
        {
            "user_id": "07852045-baa5-4794-86f5-8d56dcec93ed",
            "full_name": "Terry Lam",
            "bio": "I am a junior studying CS at Purdue University!",
            "major": "Computer Science",
            "grad_year": 2019
        },
        {
            "user_id": "9ed4e3f7-08e3-4624-9ec5-bb7b32e38302",
            "full_name": "Divya Vempati",
            "bio": "I am a junior studying Web Programming & Design at Purdue University!",
            "major": "Web Programming and Design",
            "grad_year": 2018
        },
        {
            "user_id": "da57f4c8-dd55-4204-8be6-62b14c5c1b06",
            "full_name": "Andrew Lee",
            "bio": "Junior studying Supply Chain, looking for internship!",
            "major": "Supply Chain Management",
            "grad_year": 2019
        }
    ],
    "userId": "07852045-baa5-4794-86f5-8d56dcec93ed",
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsYW00NUBwdXJkdWUuZWR1IiwidXNlcklkIjoiMDc4NTIwNDUtYmFhNS00Nzk0LTg2ZjUtOGQ1NmRjZWM5M2VkIiwiZXhwIjoxNTIyMzYzMzc1fQ.VekS4OL7gpm6FrxPzlt3RNKpymDkoTq5FcqLYT-f38FVgXxrQYwFahgcthGtGzq1r1NxIvJdIt04Bu3-PqWWhA",
    "status": "OK"
}
```


As expected, we are returned all users containing the letter “e” in their name.

Key	Value
☑ userId	07852045-baa5-4794-86f5-8d56dcec93ed
☑ token	eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsYW00NUBwdXJkdWUuZW...
☑ type	name
☑ query	terry
New key	Value

Here, we have the same request, but the query is now “terry”

```
{
    "query": [
        {
            "user_id": "07852045-baa5-4794-86f5-8d56dcec93ed",
            "full_name": "Terry Lam",
            "bio": "I am a junior studying CS at Purdue University!",
            "major": "Computer Science",
            "grad_year": 2019
        }
    ],
    "userId": "07852045-baa5-4794-86f5-8d56dcec93ed",
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsYW00NUBwdXJkdWUuZWR1IiwidXNlcklkIjoiMDc4NTIwNDUtYmFhNS00Nzk0LTg2ZjUtOGQ1NmRjZWM5M2VkIiwiZXhwIjoxNT
        IyMzYzMzc1fQ.VekS4OL7gpm6FrxPzlt3RNKpymDkoTq5FcqLYT-f38FVgXxrQYwFahgcthGtGzq1r1NxIvJdItO4Bu3-PqWWhA",
    "status": "OK"
}
```


As expected, we are returned the only user whose name is “Terry”. Notice that queries are allowed to be as vague or specific as needed.

2. The second test was conducted for searching with users with specific skills.

Scout

GET http://localhost:8080/scout?userId=07852045-baa5-4794-86f5-8d56dcec93ed&token=eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsY... Params

Key	Value	Description
☑ userId	07852045-baa5-4794-86f5-8d56dcec93ed	
☑ token	eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsYW00NUBwdXJkdWUuZW...	
☑ type	skill	
☑ query	scala	

In this example, we are now querying for users by a skill, specifically Scala.


```
{
    "query": [
        {
            "user_id": "07852045-baa5-4794-86f5-8d56dcec93ed",
            "full_name": "Terry Lam",
            "bio": "I am a junior studying CS at Purdue University!",
            "major": "Computer Science",
            "grad_year": 2019
        },
        {
            "user_id": "f9cd8b1c-c7d2-4f03-a458-b305c39785e7",
            "full_name": "Simon Lam",
            "bio": "Aspiring software engineering, interested in FinTech!",
            "major": "Computer Engineering",
            "grad_year": 2016
        }
    ],
    "userId": "07852045-baa5-4794-86f5-8d56dcec93ed",
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsYW00NUBwdXJkdWUuZWR1IiwidXNlcklkIjoiMDc4NTIwNDUtYmFhNS00Nzk0LTg2ZjUtOGQ1NmRjZWM5M2VkIiwiZXhwIjoxNT
        IyMzYzMzc1fQ.VekS4OL7gpm6FrxPzlt3RNKpymDkoTq5FcqLYT-f38FVgXxrQYwFahgcthGtGzq1r1NxIvJdItO4Bu3-PqWWhA",
    "status": "OK"
}
```


The response body returns two users in our database who have listed Scala as one of their skills. The associated skill_id for Scala is **<9adb5ef6-ffc6-4c1f-bf90-bf20811a9ea8>**

Result Grid | Filter Rows: Search | Export:

user_id	skill_id
07852045-baa5-4794-86f5-8d56dcec93ed	51d7d2c4-ee74-498f-ab66-768a5347d997
07852045-baa5-4794-86f5-8d56dcec93ed	8faadadc-d3ac-4084-a9be-0aa34fd98898
07852045-baa5-4794-86f5-8d56dcec93ed	6c72780d-0759-4321-a968-ea9afd3d917c
07852045-baa5-4794-86f5-8d56dcec93ed	8cbacae9-f572-457b-8864-a7757523b8d2
07852045-baa5-4794-86f5-8d56dcec93ed	9adb5ef6-ffc6-4c1f-bf90-bf20811a9ea8
07852045-baa5-4794-86f5-8d56dcec93ed	c4e4e89e-6784-4278-a38b-f6eeb4309fea
07852045-baa5-4794-86f5-8d56dcec93ed	fa6b9394-8aa5-492d-b8ee-7e748c5dc502
07852045-baa5-4794-86f5-8d56dcec93ed	03cc7061-fed9-4104-b6b8-133ad27e3dc0
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	c4e4e89e-6784-4278-a38b-f6eeb4309fea
f9cd8b1c-c7d2-4f03-a458-b305c39785e7	51d7d2c4-ee74-498f-ab66-768a5347d997
f9cd8b1c-c7d2-4f03-a458-b305c39785e7	9adb5ef6-ffc6-4c1f-bf90-bf20811a9ea8

The user_skills table supports this and shows the two user_ids associated with the skill_id for Scala.

3. As a third test, we wanted to search with users enrolled in a specific course.

Scout

GET http://localhost:8080/scout?userId=07852045-baa5-4794-86f5-8d56dcec93ed&token=eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsY...

	Key	Value	Description
☑	userId	07852045-baa5-4794-86f5-8d56dcec93ed	
☑	token	eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsYW00NUBwdXJkdWUuZW...	
☑	type	course	
☑	query	cgt	

This image demonstrates a query for a course, specifically users enrolled in a CGT course.

Pretty Raw Preview JSON Save Response

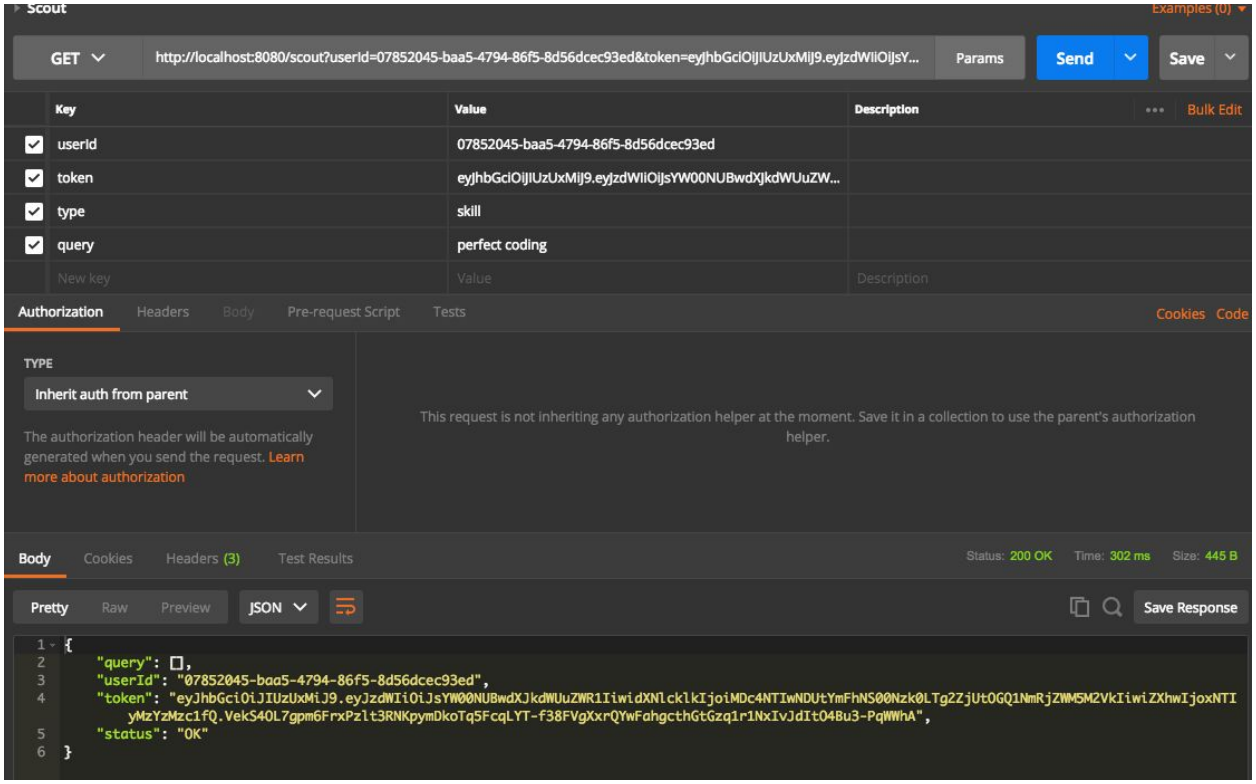

```
{
    "query": [
        {
            "user_id": "9ed4e3f7-08e3-4624-9ec5-bb7b32e38302",
            "full_name": "Divya Vempati",
            "bio": "I am a junior studying Web Programming & Design at Purdue University!",
            "major": "Web Programming and Design",
            "grad_year": 2018
        }
    ],
    "userId": "07852045-baa5-4794-86f5-8d56dcec93ed",
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsYW00NUBwdXJkdWUuZWR1IiwidXNlcklkIjoiMDc4NTIwNDUtYmFhNS00Nzk0LTg2ZjUtOGQ1NmRjZWM5M2VkIiwiZXhwIjoxNTIyMzYzMzc1fQ.VekS4OL7gpm6FrxPzlt3RNKpymDkoTq5FcqLYT-f38FVgXxrQYwFahgcthGtGzq1r1NxIvJdItO4Bu3-PqWWhA",
    "status": "OK"
}
```


We are returned the only user enrolled in any CGT courses.
Associated course_ids for CGT courses are
<35c5bce2-5edc-44ef-ad64-6784d5448cd3>
<a1fc8b22-8335-4441-829d-4758ffbe7dda>
<dbb17246-abc3-4195-bf88-9000abdf2855>

user_id	course_id
07852045-baa5-4794-86f5-8d56dcec93ed	c1443a3f-5317-4536-9551-ce206b6cdb90
07852045-baa5-4794-86f5-8d56dcec93ed	7d69cab0-0cef-4f79-a76b-aadfa94a5671
07852045-baa5-4794-86f5-8d56dcec93ed	418f783f-8a39-4cbc-8ecb-cdb1e5ab9b65
07852045-baa5-4794-86f5-8d56dcec93ed	52e53f3d-ccf4-4399-a87a-fb38b0732b3f
07852045-baa5-4794-86f5-8d56dcec93ed	30191a30-d192-4a84-816a-d6b584bb8903
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	35c5bce2-5edc-44ef-ad64-6784d5448cd3
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	a1fc8b22-8335-4441-829d-4758ffbe7dda
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	dbb17246-abc3-4195-bf88-9000abdf2855
9ed4e3f7-08e3-4624-9ec5-bb7b32e38302	52e53f3d-ccf4-4399-a87a-fb38b0732b3f
da57f4c8-dd55-4204-8be6-62b14c5c1b06	52e53f3d-ccf4-4399-a87a-fb38b0732b3f

The user_courses table supports this and shows that the user_id for “Divya Vempati” is associated with the course_ids for CGT classes.

4. As an additional test, we want to ensure no results or errors are thrown for a query that doesn't apply to any user.



This image demonstrates a query for a skill called "perfect coding" which doesn't exist in our database. Therefore, an empty query result set is returned (NOT an error).

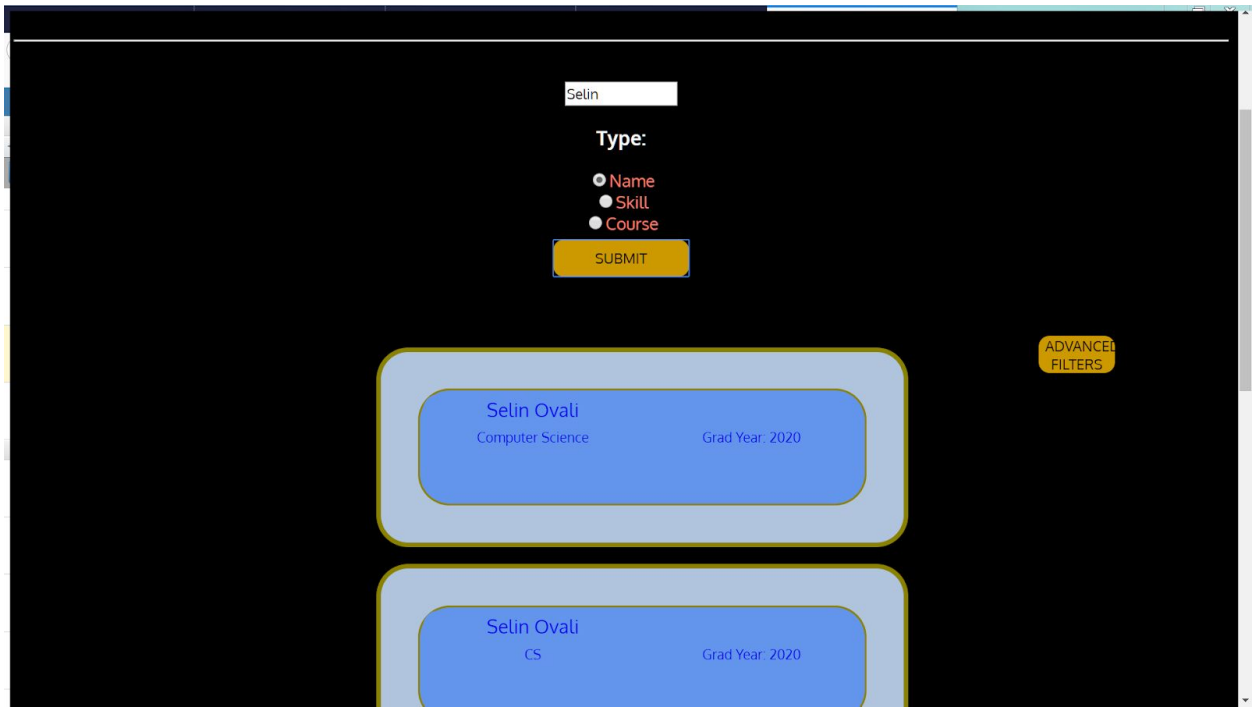
Testing Summary (Backend): Each search query type was tested for this module (search by name, skill, or course). It is also demonstrated that queries can be as specific or vague as possible (for example, just querying "cs" rather than a specific CS course will still return results). We also tested queries for empty result sets and error handling, which is also shown in the last test in set one.

Testing Procedure (Frontend): This is the protocol used to test basic search functionality from a frontend perspective.

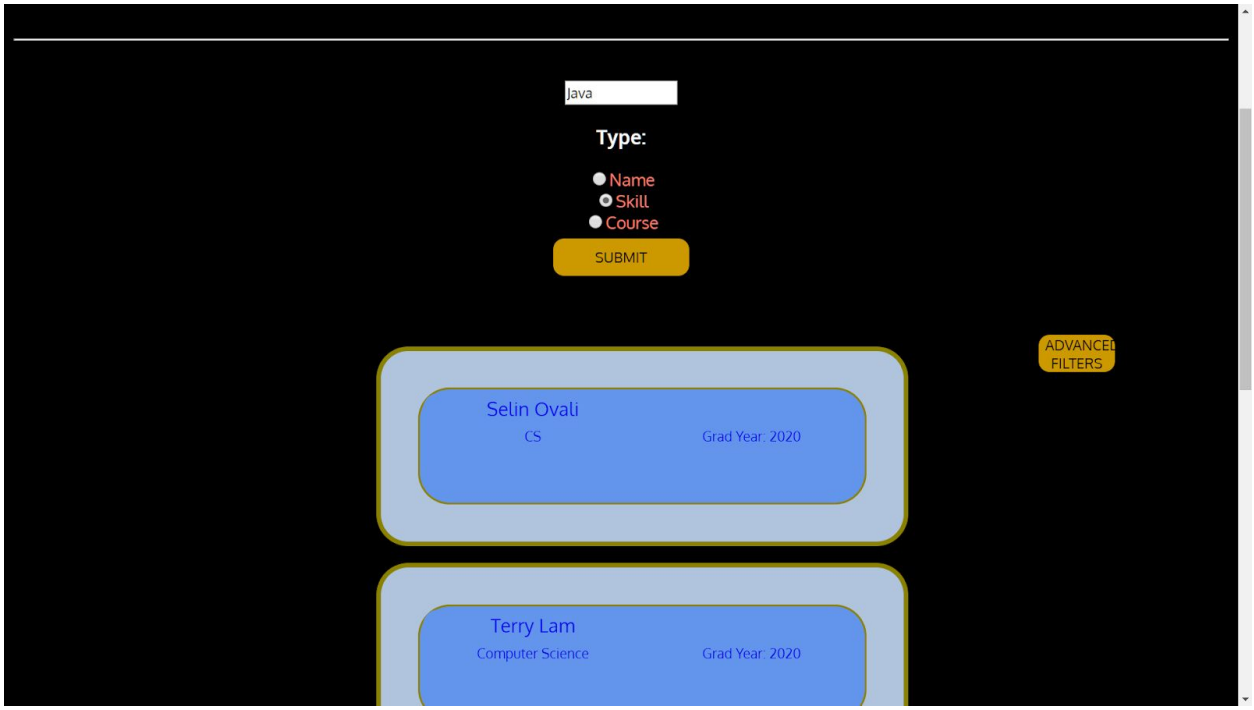
- Navigate to the /scout page after logging in
- Enter in a request that doesn't match any user. Enter in nothing into the search box. Enter in something that should match a user.
- The request that matches no users should display no users. When nothing is entered into the search box, nothing should be displayed and if results are currently displayed, they should be cleared. When a valid search is inputed that matches existing users, those users should be displayed.

Testing results and follow up

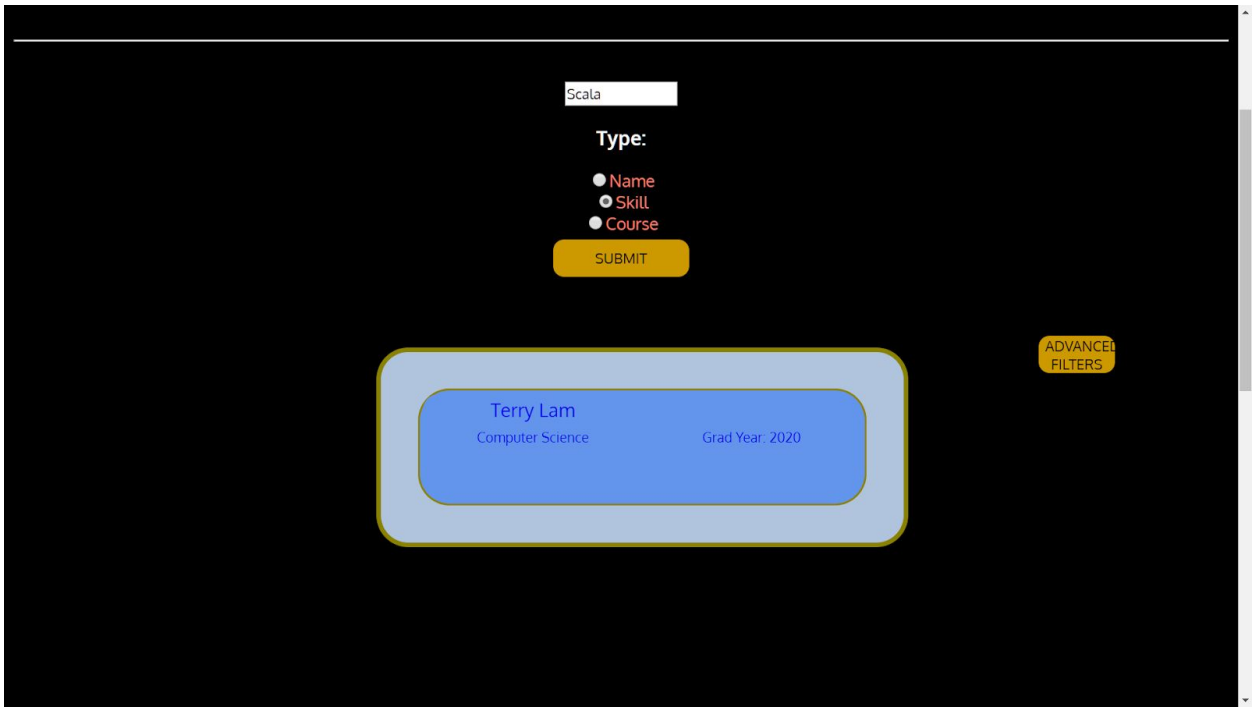
1. Requests made for “scouting” users based on a certain query were successful. The first test conducted was for queries based on name.



2. The second test was conducted for searching with users with specific skills.

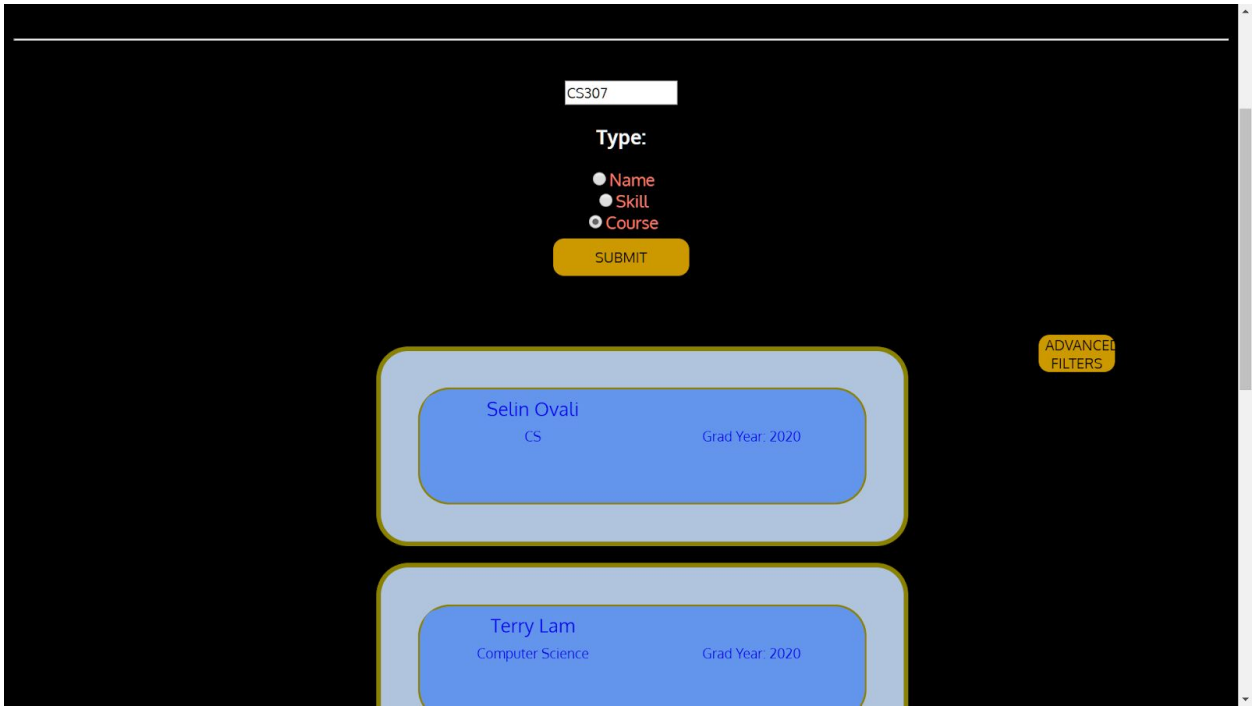


Two Users were returned when Java was searched



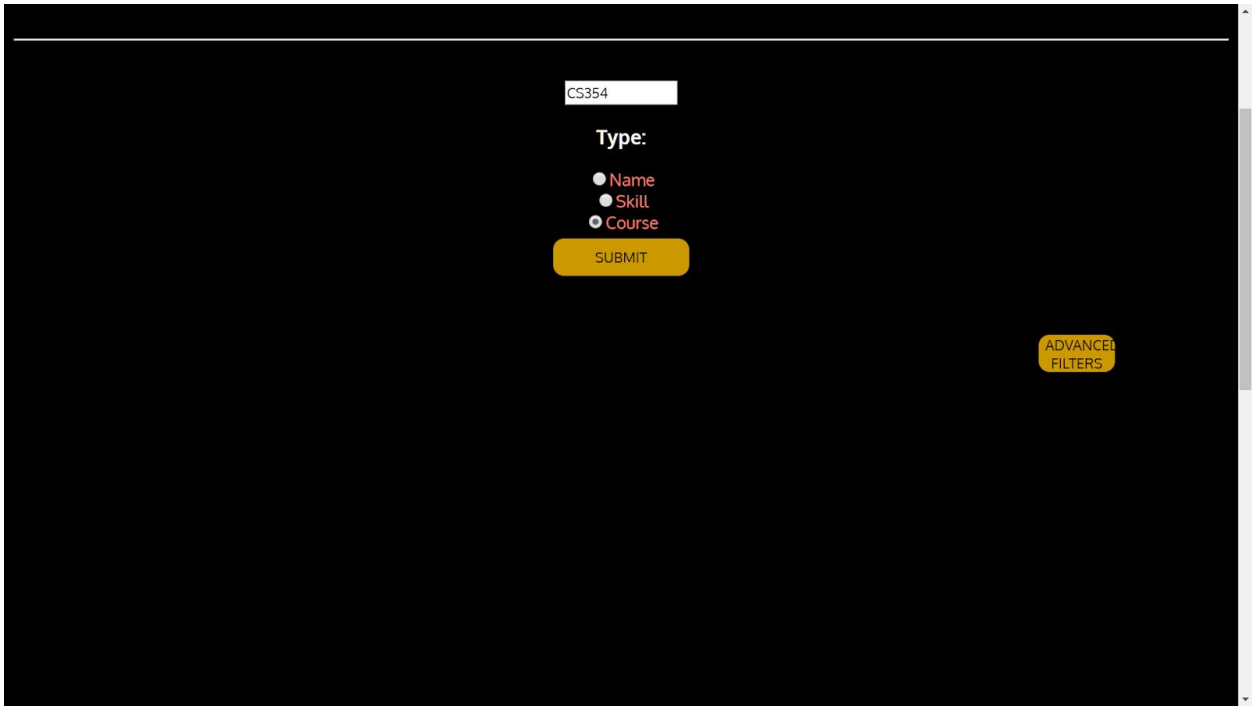
One user was returned when Scala was searched.

3. As a third test, we wanted to search with users enrolled in a specific course.

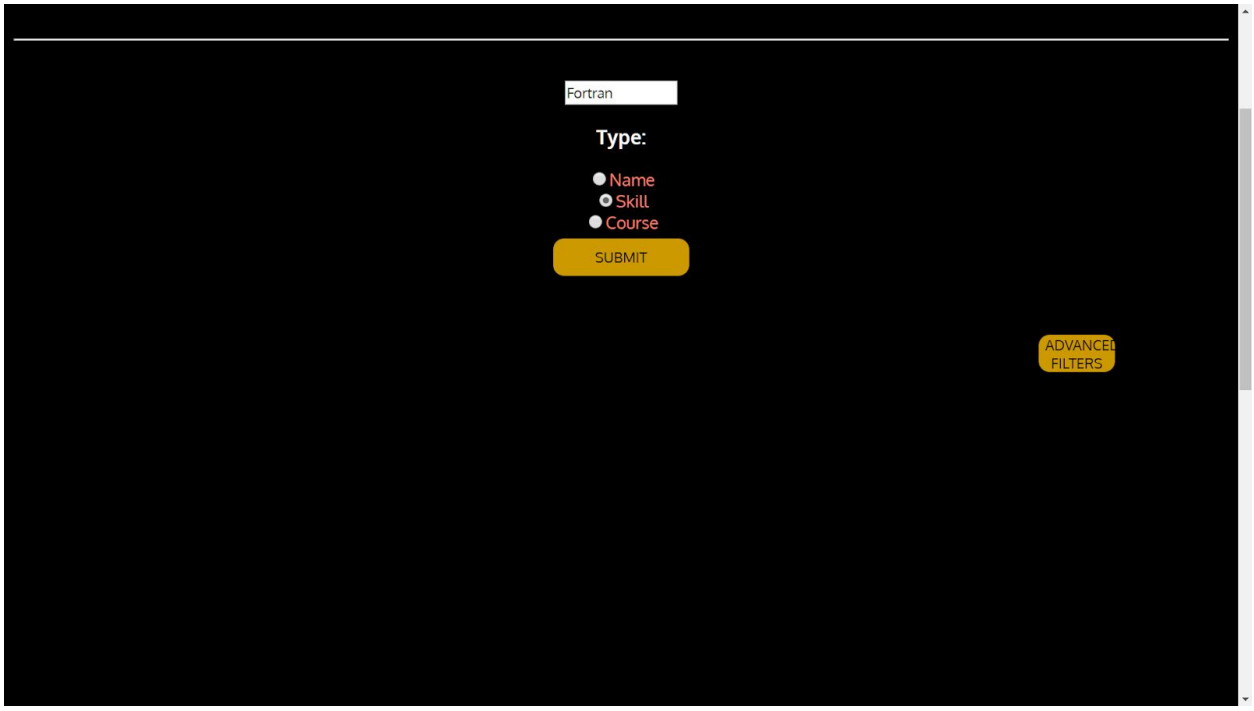


Two users are returned when CS307 is searched as a course.

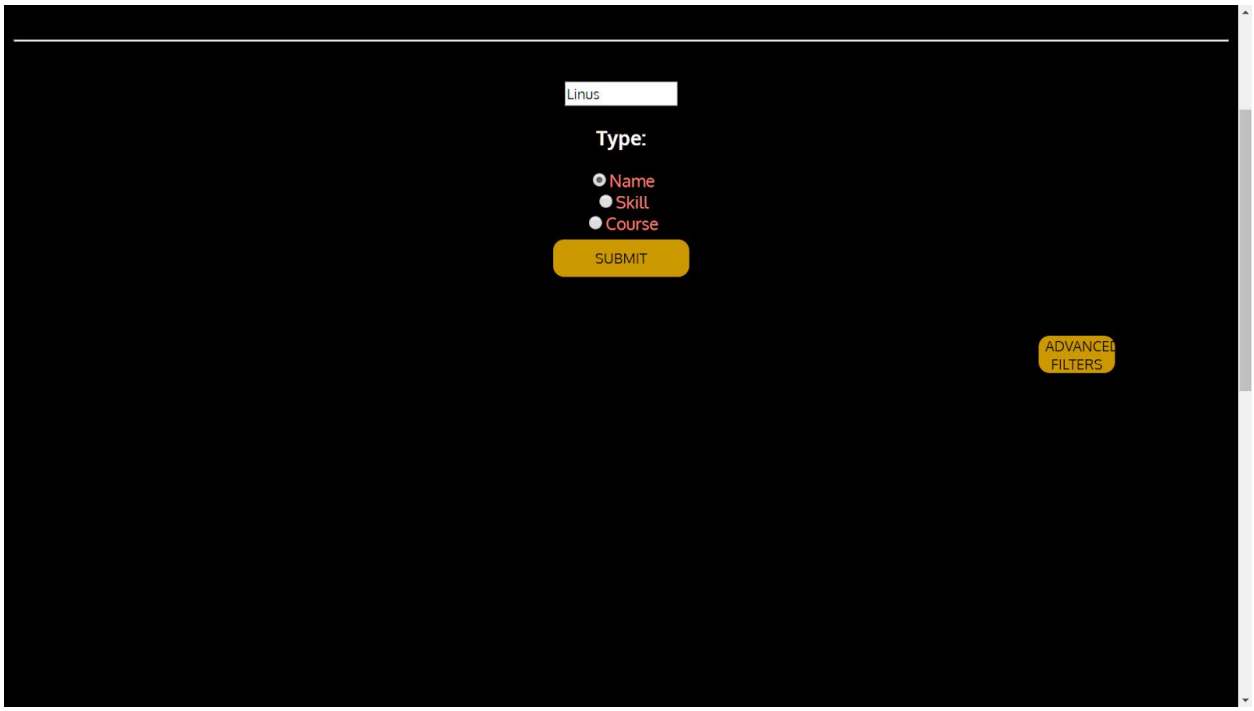
4. When a name that doesn't exist, or a skill that doesn't exist, or a course that doesn't exist is searched, we want nothing to be displayed.



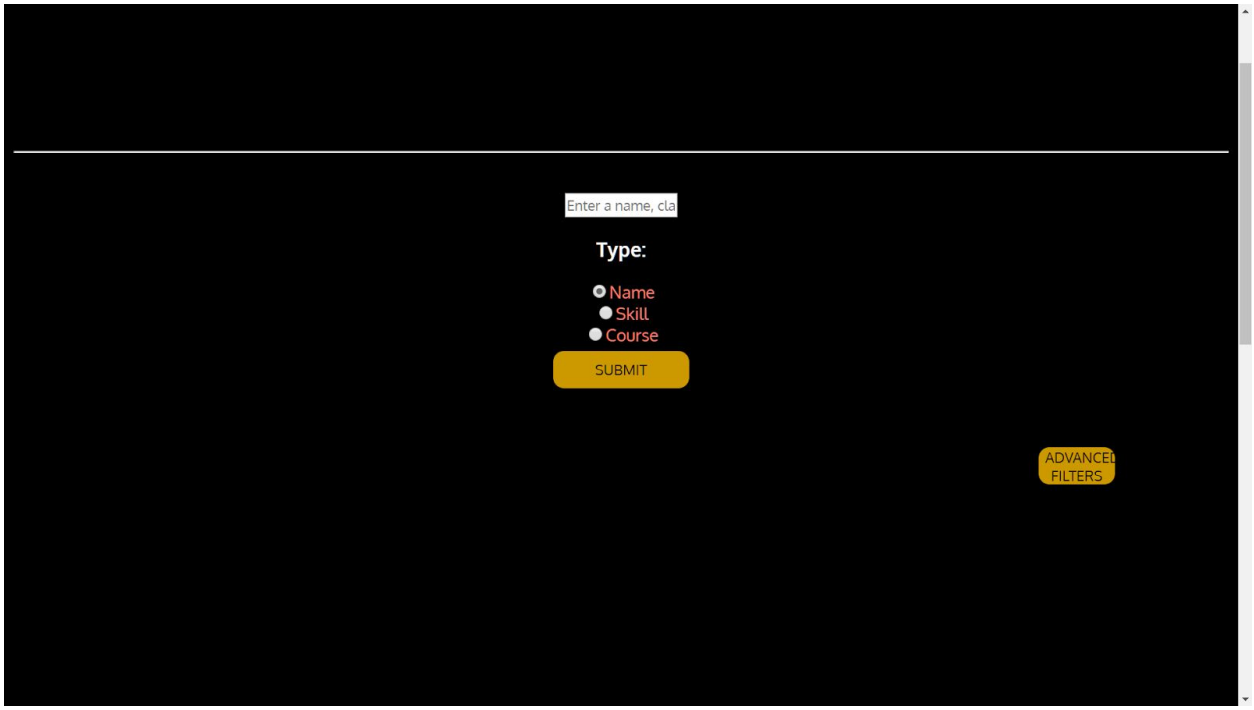
No users exist that have taken CS354 before, or are currently in it.



No users exist that have Fortran as a skill



No users with the name Linus exists



A blank search should display nothing, and clear the last results, if any are present.

Testing Summary (Frontend): Each search query type was tested for this module (search by name, skill, or course). If a name that doesn't exist is searched, nothing is displayed. The same goes for courses and skills that no users possess. If no users have those skills or courses, nothing is displayed. A blank search clears the results, if any are already present and displays nothing otherwise.

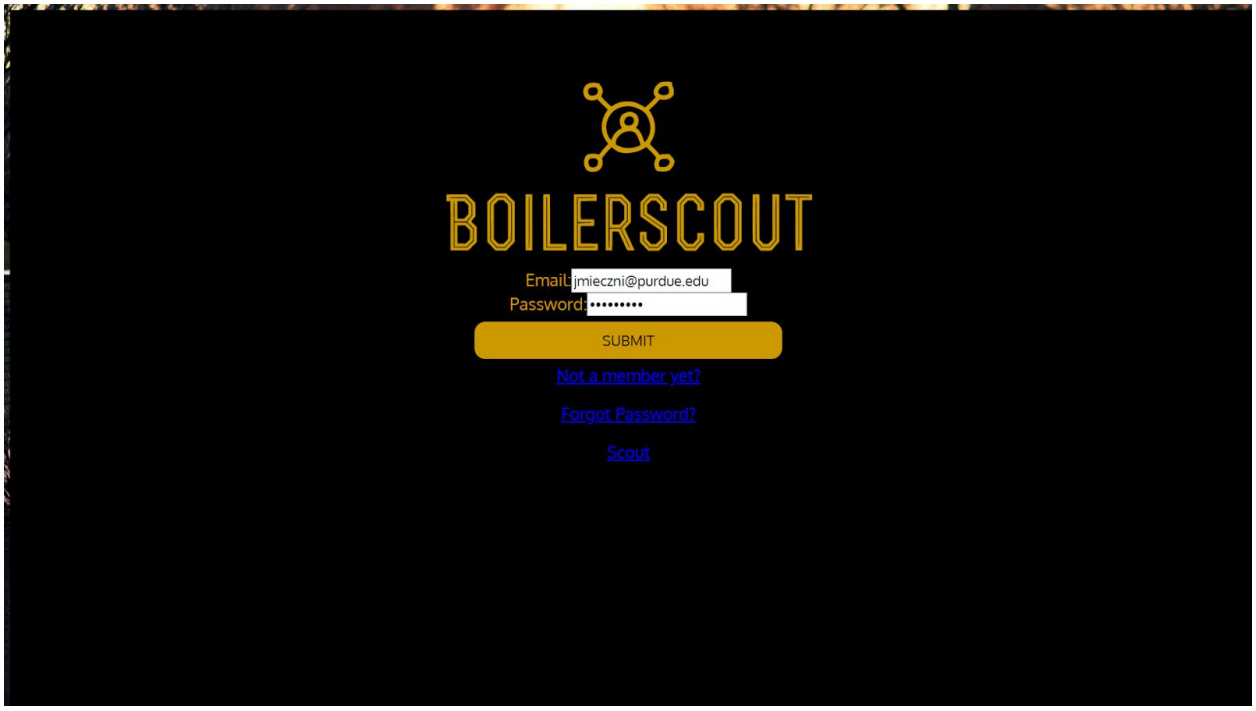
Testing Set Four: Logining in with an existing username and password

Testing Procedure: This is the protocol used to test basic search functionality from a frontend perspective.

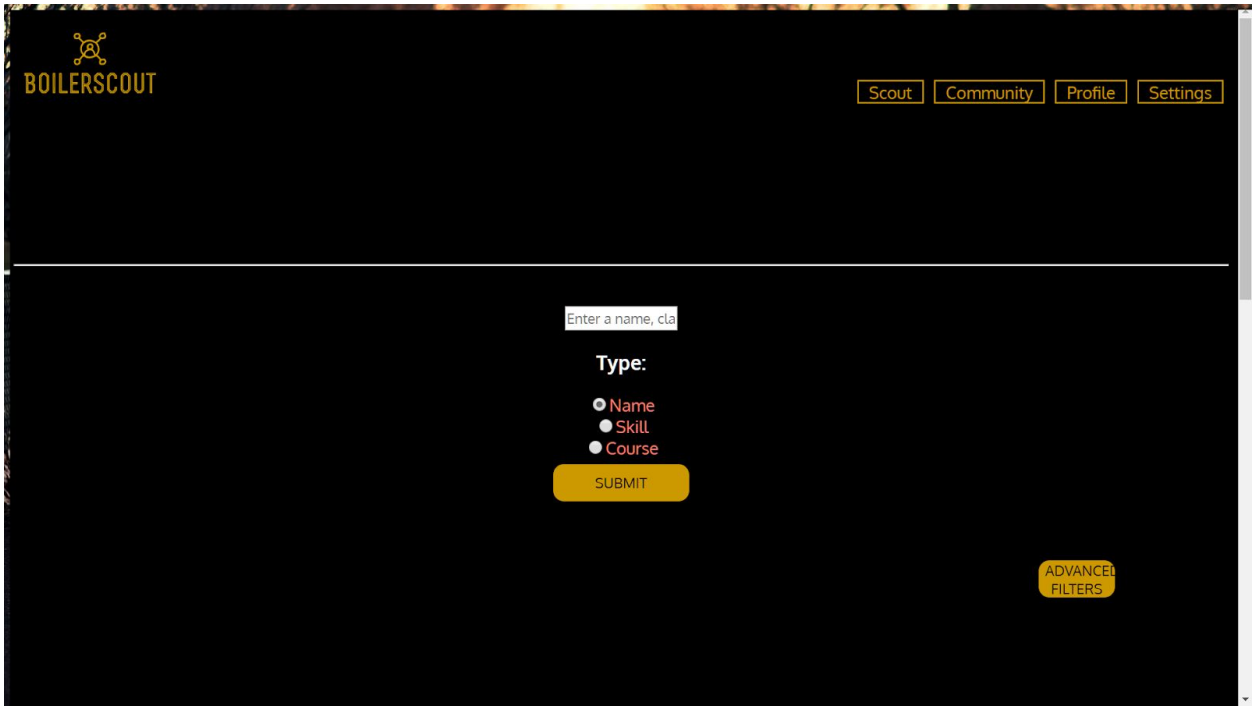
- Enter in an existing email into the application with the correct password. This should result in letting the user know that he/she was successfully logged in by redirecting to home (Scout view)
- A non existent email entered should result in an error message.
- A valid email entered with an invalid password should result in an error message as well.
- On a successful login, the user id and token should be stored in Local Storage

Testing results and follow up

1. An existing email and correct password redirect to /Scout

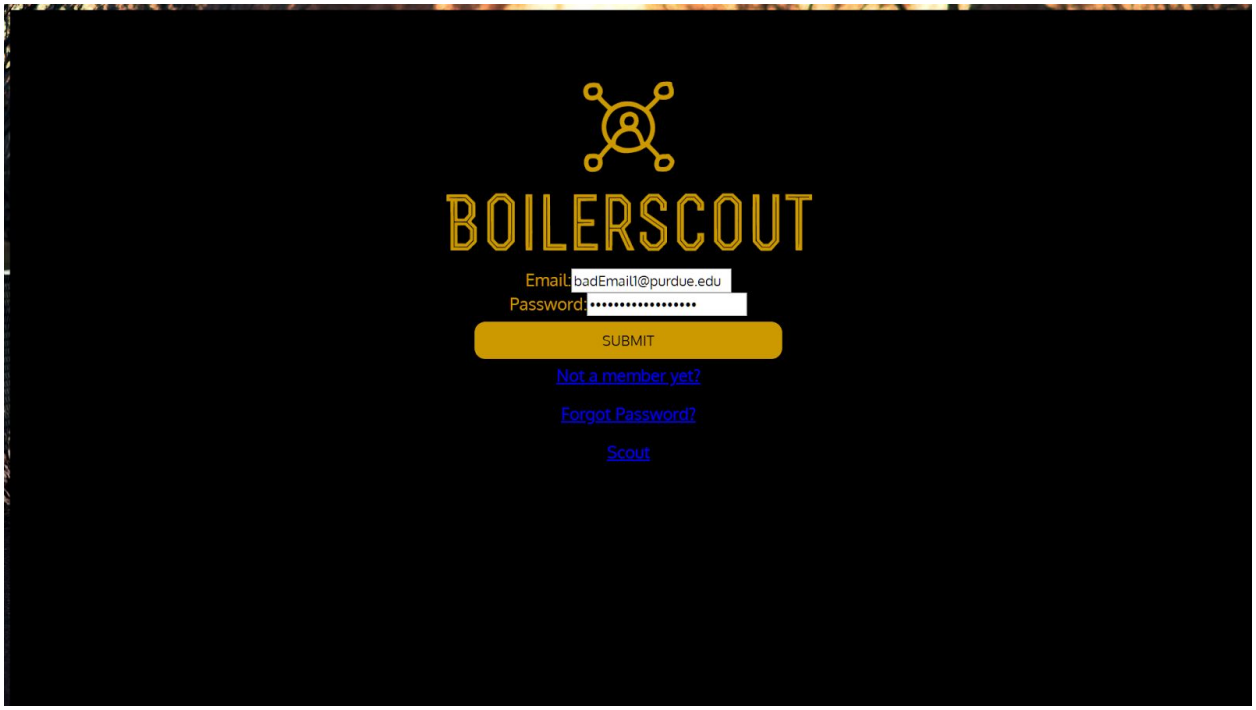


A correct email and correct password

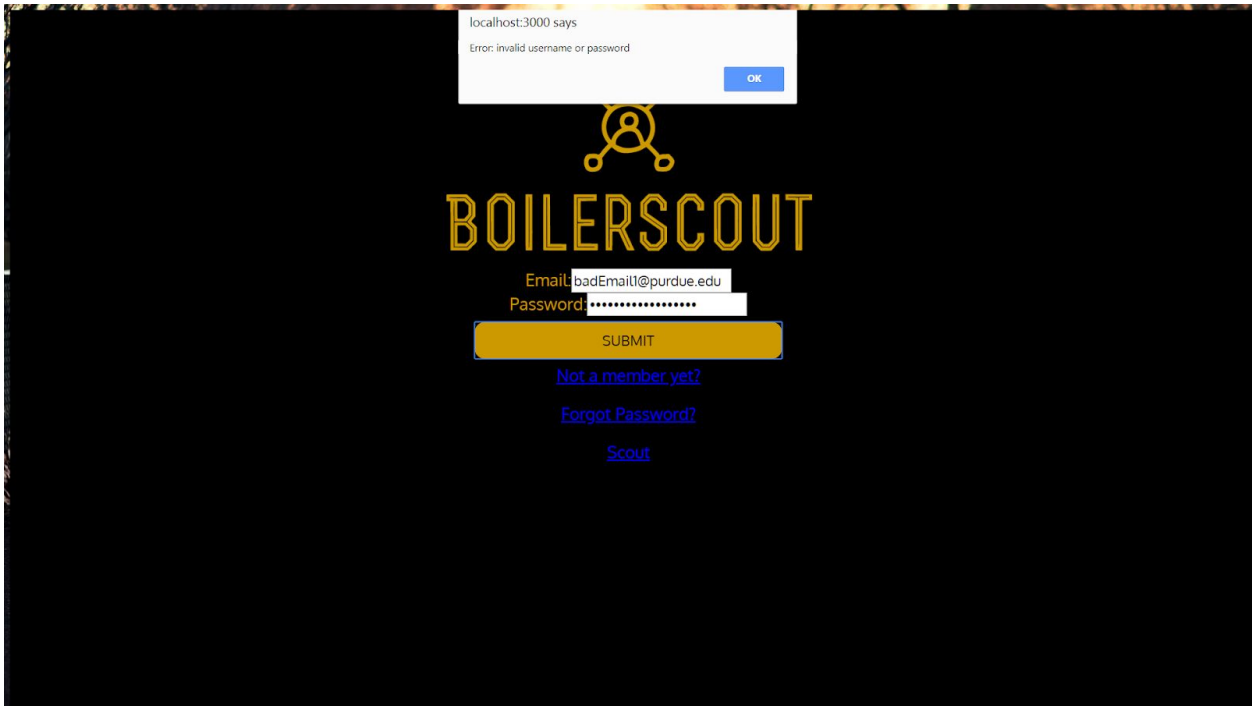


SUBMIT button is clicked. User is redirected to Scout and is now logged in.

2. A non existent email entered should result in an error message.

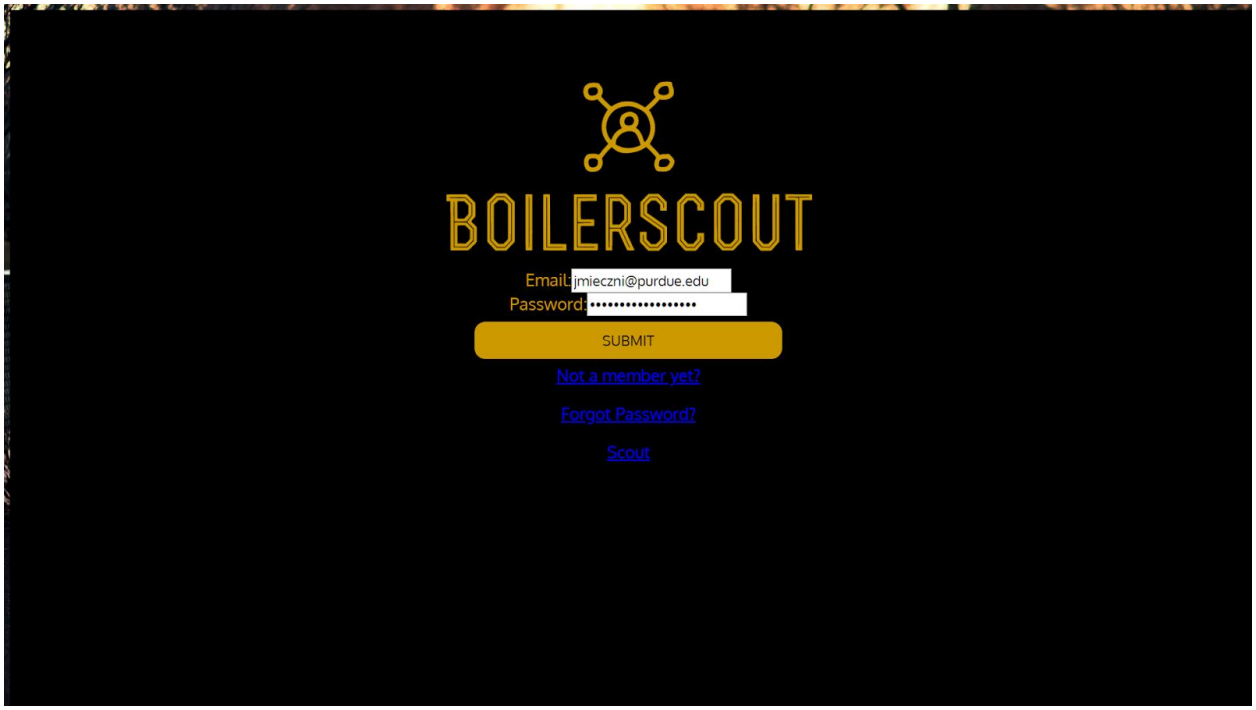


A nonexistent email is entered.

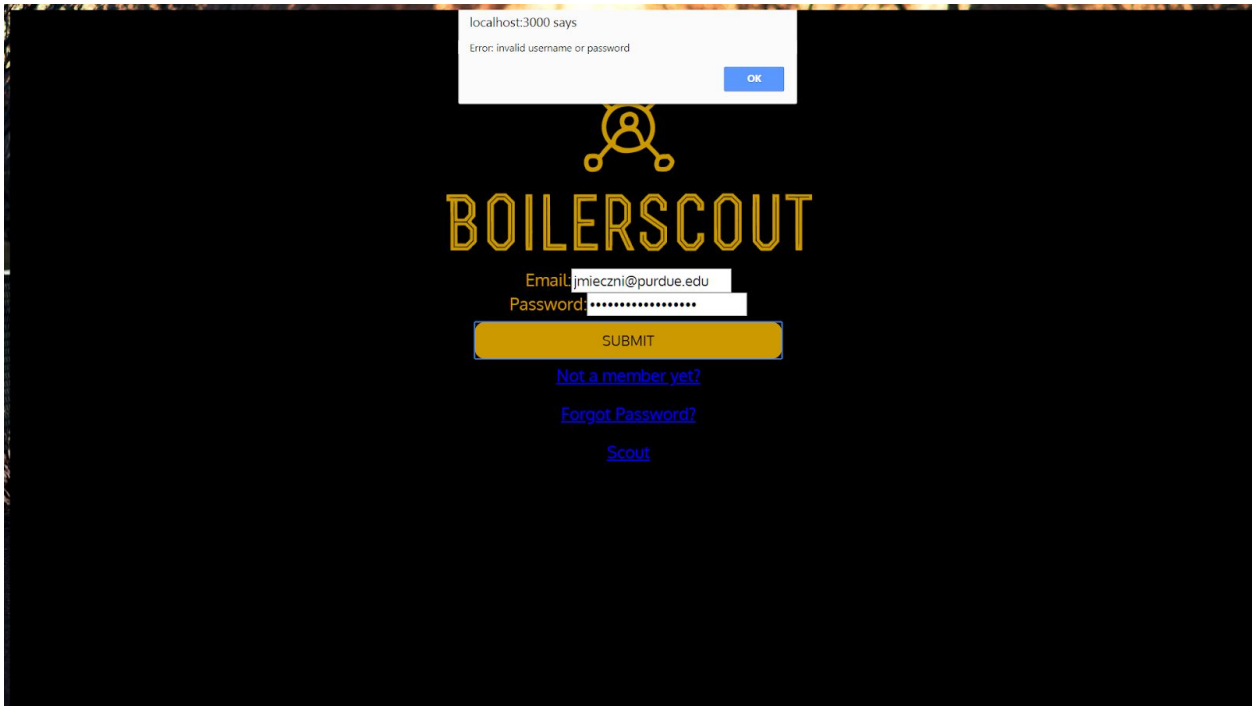


SUBMIT is clicked, and the user is informed that there is an invalid username or password.

3. A valid email entered with an invalid password should result in an error message.

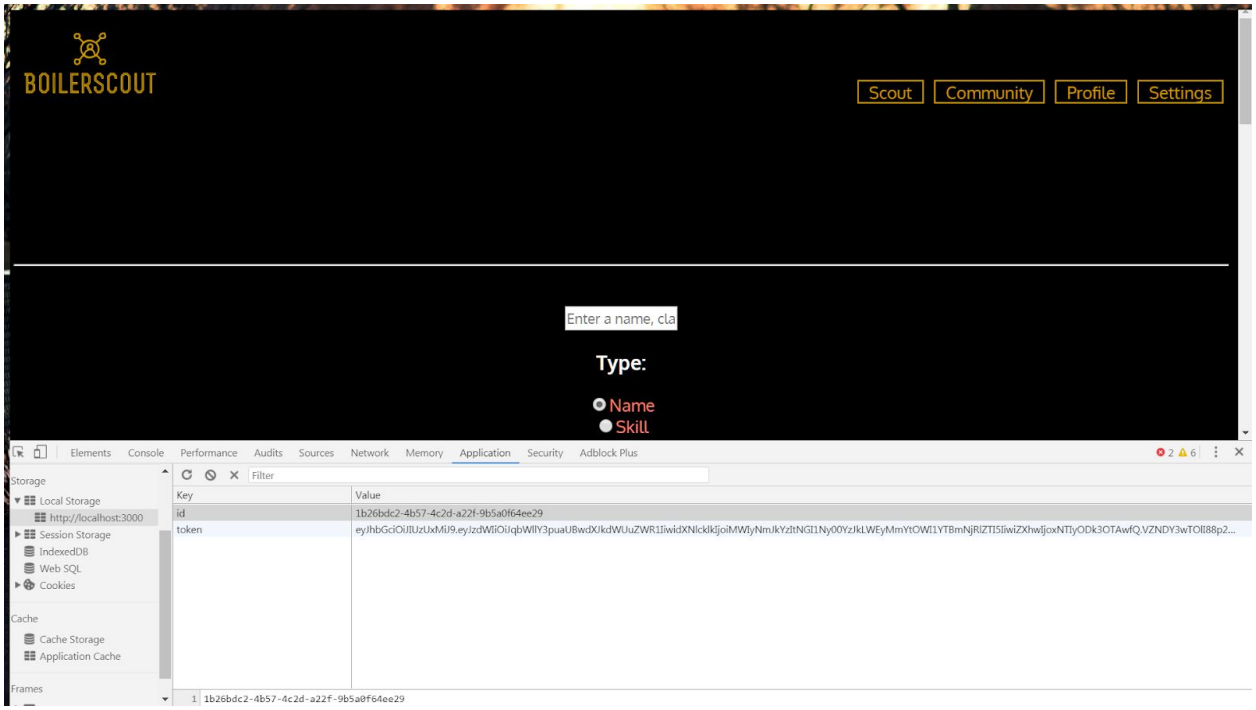


An existing email is entered, but there is an incorrect password.



SUBMIT is clicked, and the user is informed that the password is incorrect.

4. User id and token should be stored in Local Storage on successful login.



After a user is redirected to /Scout, the user_id and token of that user is now stored in LocalStorage.

Testing Summary(Frontend):

A nonexistent email is met with an error. A correct email and password combination result in a successful log, a redirect to Scout, and storage of the id and token in local storage. If the password is wrong, the user is informed. This component is fully functional.

Testing Set Six: Displaying a user when a result is clicked on from some other area of the application.

Testing Procedure (Backend): This is the protocol used to test the methods that return a user's profile information when requested, from a backend perspective.

- Create a **GET** request on Postman that hits the endpoint /profile/get
- As request parameters, input the active userId and its corresponding token, and the userId of the user whose profile we want to return.
- Send three different requests via Postman to test different users, each possessing a different amount of information in their profile. This should return all of their information available, as well as the (active) userId and token passed to the endpoint.

Testing results and follow up

1. First, we will pass as query, the id of a newly created user, who only has the basic info which all users have (Name, Major, Graduation Year, and Email), and nothing else.

user_id	full_name	bio	major	grad_year
29883e6f-b475-4539-84b6-7a8f60d2fc85	Hardy Montoya	NULL	cs	2019
NULL	NULL	NULL	NULL	NULL

This is the user represented in the 'profiles' table of the database.

GET localhost:8080/profile/get?id=29883e6f-b475-4539-84b6-7a8f60d2fc85&token=eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJobW9udG95Y... Params Send

Pretty Raw Preview JSON


```
{
    "Major": "cs",
    "Graduation": 2019,
    "Email": "hmontoya@purdue.edu",
    "userId": "29883e6f-b475-4539-84b6-7a8f60d2fc85",
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJobW9udG95YUBwdXJkdWUuZWR1IiwidXNlck1kIjoiMjk4ODNlNmYtYjQ3NS00NTM5LTg0YjYtN2E4ZjYwZDJmYzg1IiwiZXhwIjoxNTIyOTA
        .6W18gjVOrM74cIqO7swYtK1z3FPHCVjhN_rDasemCQ8OHOLsJ9R_lp9tJTYTBG2zypNR2tIfNqRoyb0bDxzVfg",
    "Name": "Hardy Montoya"
}
```


This image shows the return of running the /profile/get endpoint for a newly created user with only basic info.

2. Now we will add a bio to the user, utilizing the update profile method tested previously in the document, and then run the same method exactly.

POST localhost:8080/update-profile Params Send


```
        .6W18gjVOrM74cIqO7swYtK1z3FPHCVjhN_rDasemCQ8OHOLsJ9R_lp9tJTYTBG2zypNR2tIfNqRoyb0bDxzVfg",
    "bio":"New Bio!"
}
```


user_id	full_name	bio	major	grad_year
29883e6f-b475-4539-84b6-7a8f60d2fc85	Hardy Montoya	New Bio!	cs	2019
NULL	NULL	NULL	NULL	NULL

Adding a bio (New Bio!)

GET localhost:8080/profile/get?id=29883e6f-b475-4539-84b6-7a8f60d2fc85&token=eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJobW9udG95Y... Params Send Save

Pretty Raw Preview JSON


```
{
    "Major": "cs",
    "Graduation": 2019,
    "Email": "hmontoya@purdue.edu",
    "Bio": "New Bio!",
    "userId": "29883e6f-b475-4539-84b6-7a8f60d2fc85",
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJobW9udG95YUBwdXJkdWUuZWR1IiwidXNlck1kIjoiMjk4ODNlNmYtYjQ3NS00NTM5LTg0YjYtN2E4ZjYwZDJmYzg1IiwiZXhwIjoxNTIyOTA4MDc4fQ
        .6W18gjVOrM74cIqO7swYtK1z3FPHCVjhN_rDasemCQ8OHOLsJ9R_lp9tJTYTBG2zypNR2tIfNqRoyb0bDxzVfg",
    "Name": "Hardy Montoya"
}
```


We can notice that the reply this time is different, as it now includes the new bio added.

3. Now we will also add skills and courses to the user, following the method tested earlier.

POST localhost:8080/update-profile Params Send


```
{
    "userId":"29883e6f-b475-4539-84b6-7a8f60d2fc85",
    "token":"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJobW9udG95YUBwdXJkdWUuZWR1IiwidXNlcklkIjoiMjk4ODN1NmYtYjQ3NS00NTM5LTg0YjYtN2E4ZjYwZDJmYzg1IiwiZXhwIjoxNTI
        .6W18gjVOrM74cIqO7swYtK1z3FPHCVjhN_rDasemCQ8OHOLsJ9R_lp9tJTYTBG2zypNR2tIfNqRoyb0bDxzVfg",
    "skills":["MySQL","Java","Creative Writing"],
    "courses":["ANTH 205","CS 307"]
}
```


Adding courses and skills for the previous user.

GET localhost:8080/profile/get?id=29883e6f-b475-4539-84b6-7a8f60d2fc85&token=eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJobW9udG95Y... Params Send

etty Raw Preview JSON


```
{
    "Email": "hmontoya@purdue.edu",
    "Major": "cs",
    "Graduation": 2019,
    "Skills": [
        "Java",
        "Creative Writing",
        "MySQL"
    ],
    "Bio": "New Bio!",
    "Courses": [
        "ANTH 205",
        "CS 307"
    ],
    "userId": "29883e6f-b475-4539-84b6-7a8f60d2fc85",
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJobW9udG95YUBwdXJkdWUuZWR1IiwidXNlcklkIjoiMjk4ODN1NmYtYjQ3NS00NTM5LTg0YjYtN2E4ZjYwZDJmYzg1IiwiZXhwIj
        .6W18gjVOrM74cIqO7swYtK1z3FPHCVjhN_rDasemCQ8OHOLsJ9R_lp9tJTYTBG2zypNR2tIfNqRoyb0bDxzVfg",
    "Name": "Hardy Montoya"
}
```


As expected, the returned map has now changed and includes the list of skills and courses.

Testing Summary(Backend): All variations of users has been tested. While other permutations such as no skills but courses exist, these follow similar patterns to the ones tested. More testing for this method is provided by the frontend as well. Additional checks such as providing an id that does not exist, or an invalid token, are handled by the same function as other endpoints, therefore we assume it works as it has already been tested above.

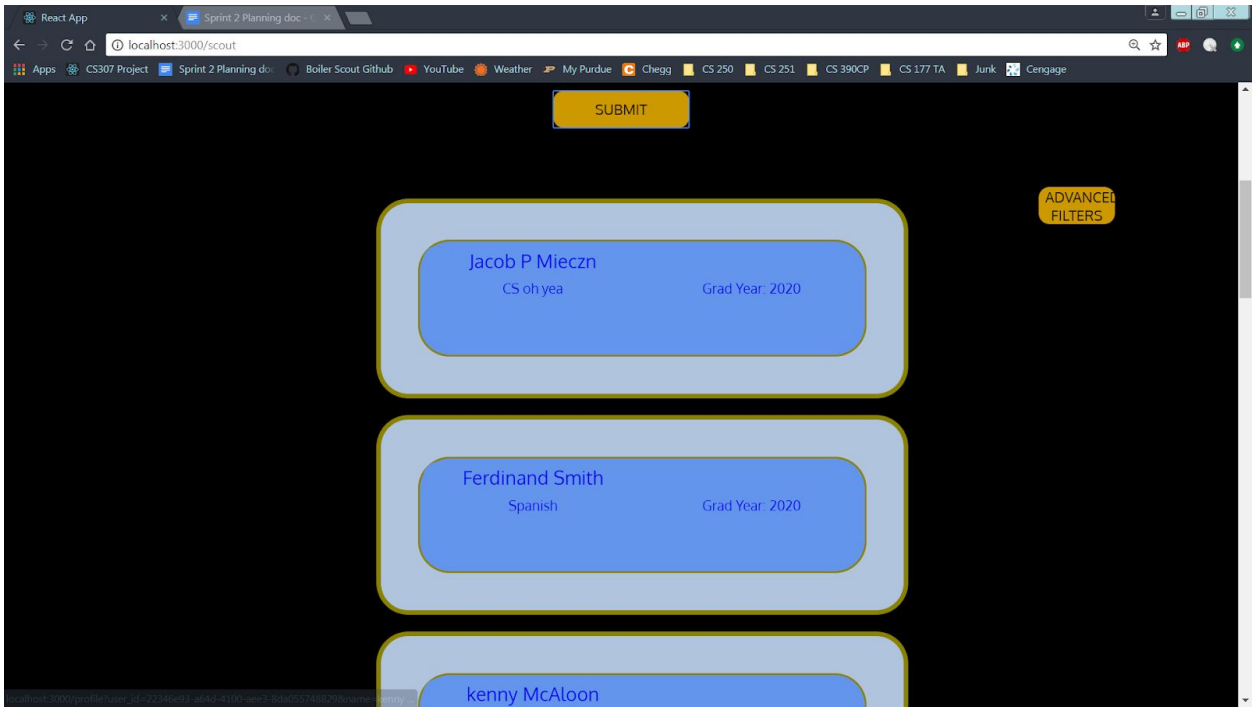
Testing Procedure (Frontend): This is the protocol used to test displaying a profile a frontend perspective.

- Many users are displayed in the Scout page. When a result is clicked on, you are taken to that user's profile page.
- Any data that doesn't exist for a user in the backend is handled gracefully and nothing is displayed in that section. For example, a user that hasn't entered a bio yet will be blank.

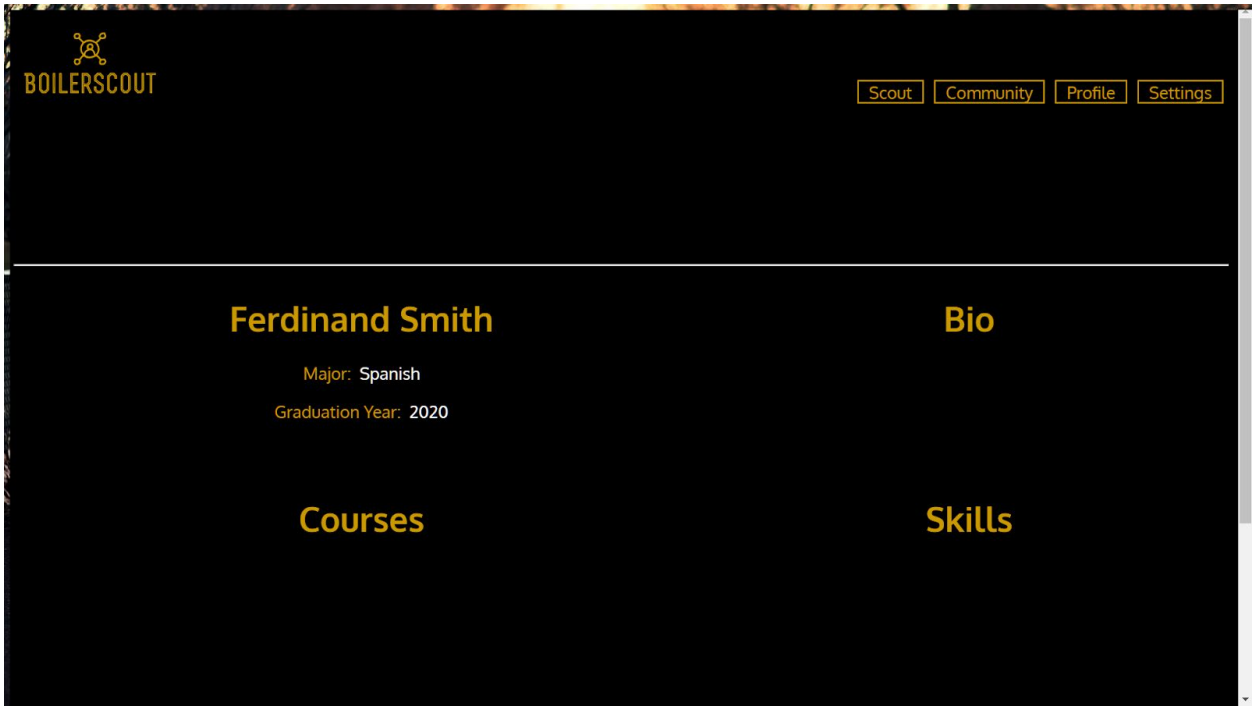
- For users that have entered in all information about themselves, all information will be displayed.
- If a user that is logged in clicks on profile link the navigation bar, he/she will be taken to view his/her own profile

Testing results and follow up

1. A user clicked on in Scout has the corresponding profile displayed.

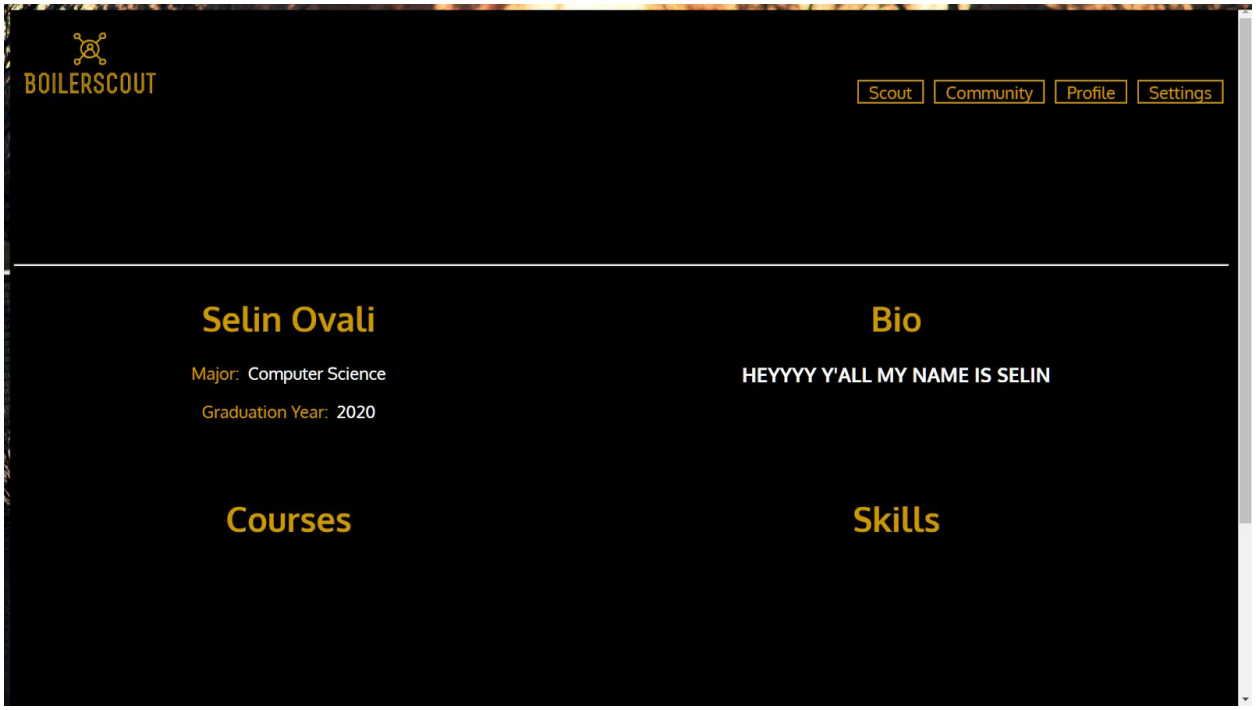


Searches shown in scout. Ferdinand Smith is clicked on.



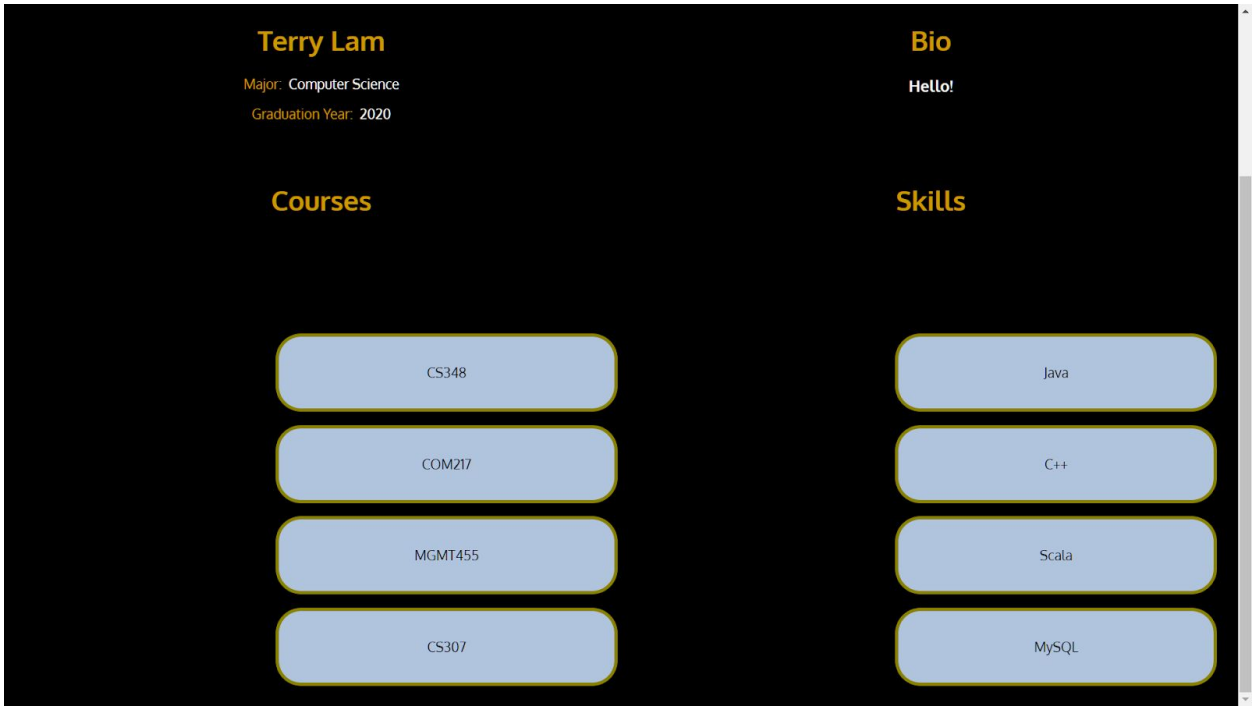
The user is redirected to Ferdinand's profile. Note that he has not filled in information about skills, courses and bio.

2. Any data that doesn't exist for a user in the backend is handled gracefully and nothing is displayed in that section. For example, a user that hasn't entered a bio yet will be blank.



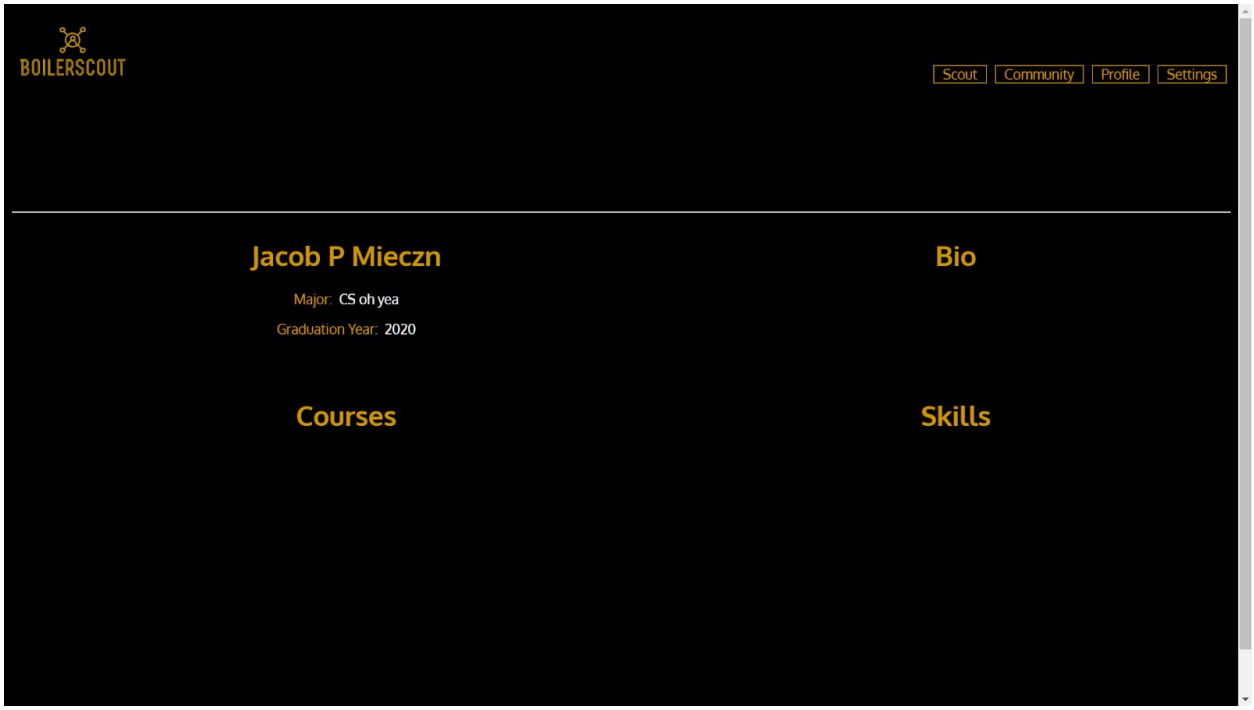
Selin is clicked on from results. She has only filled out a bio.

3. For users that have entered in all information about themselves, all information will be displayed.



Terry is clicked on. He has filled out all his information, and it is all displayed.

4. For whoever is logged in, when he/she clicks on profile in the navigation bar, his/her personal profile will be shown.



User named “Jacob P Mieczn” is logged. Profile is clicked on navbar and his own details are presented.

Testing Summary(Frontend):

Every part of information for every user displayed as it should be. For those who have not completed their information, it is not displayed. When the user clicks on Profile in the navigation bar, he/she is shown his/her own profile. This component is completely functional.

Testing Set Seven: Resending the email confirmation email.

Testing Procedure (Backend): This is the protocol to test the functionality of email verification from a backend perspective.

- An email is sent to an unverified email when /send/verification is hit. The value ‘email_verified’ in the ‘users’ table of the database changes to a random (not 1, not 0) value.
- If a new email is requested, the value changes again, never remaining equal (not even by chance).
- Verification can only be achieved by hitting the /verify endpoint with a query corresponding to the last received/requested email.

Testing results and follow up (backend)

1. We request a verification email for an unverified user.

email	email_verified
hmontova@purdue.edu	0

User's email in database before requesting email.

Today, 1:57 AM
Hardy Gustavo Montoya

Hi,
You are now one step closer to your BoilerScout account.

Please verifiy your account with the following link:

localhost:8080/verify?id=29883e6f-b475-4539-84b6-7a8f60d2fc85&query=1998566891

This is the email received, the value after "query=" is the one that the 'email_verified' should now have.

email	email_verified	authent
hmontova@purdue.edu	1998566891	evJhbGc

And here we see it is.

2. Before actually trying to verify, we request a second email, to see if the value changes again.

email	email_verified	
hmontova@purdue.edu	1857666428	e

The email is similar so we skip showing it, but this is the new value set for the field.

3. Now we try using the verification link we received first and see that it should fail.

localhost:8080/verify?id=29883e6f-b475-4539-84b6-7a8f60d2fc85&query=1998566891

{"Status":"This verification email has either expired, or is not valid. Please check your inbo
one.","email_verified":1857666428,"test":1857666428,"verification code":1998566891,"Exit":"0"}

This is a response indicating that we have attempted to verify with an expired email. Now lets try with the last email received (most current one, and therefore valid).

localhost:8080/verify?id=29883e6f-b475-4539-84b6-7a8f60d2fc85&query=1857666428

{"Status":"Verified","Exit":"1"}

This is a correct response. Meaning the user has now been verified.

4. Now lets try verifying the user again.

localhost:8080/verify?id=29883e6f-b475-4539-84b6-7a8f60d2fc85&query=1857666428

{"Status":"Email previously verified.","Exit":"0"}

Testing Summary(Backend):

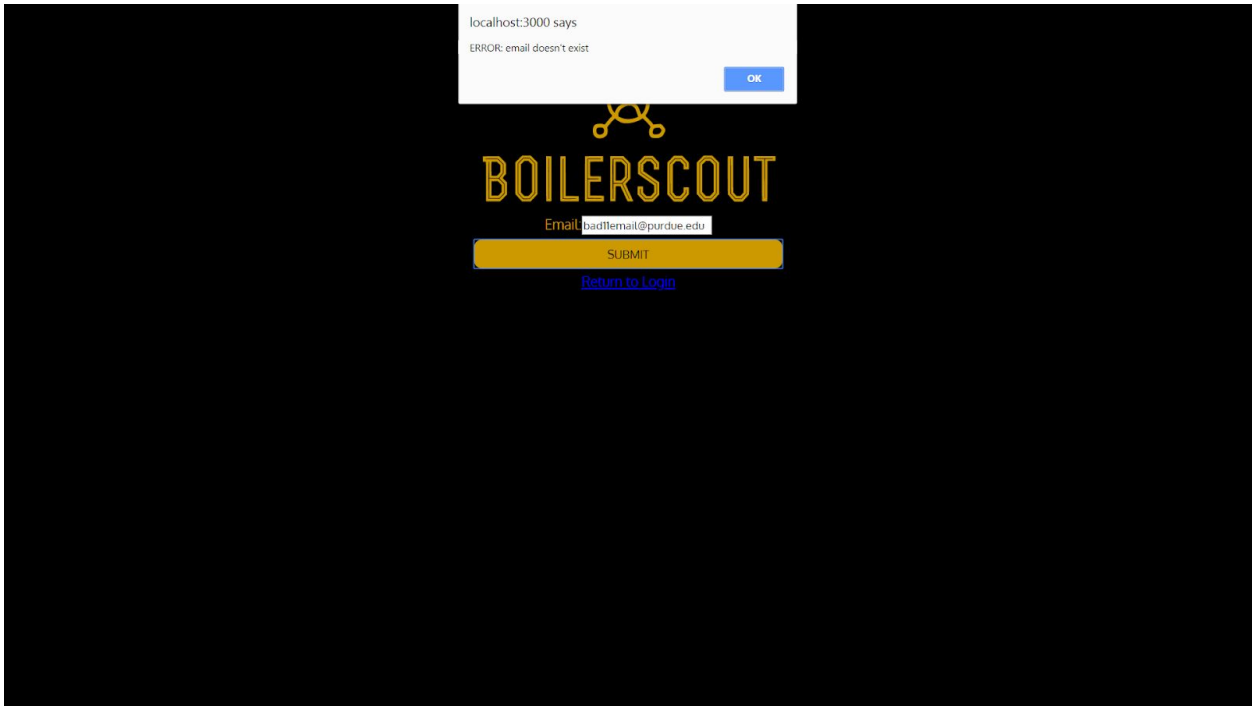
Every step of the verification procedure has been tested. Emails can be sent to any existing email, and the value never repeats itself in the immediately following iteration.

Testing Procedure (Frontend): This is the protocol used to test functionality of a user entering an email and if the confirmation email is resent.

- An email that does not exist in the database should result in the user being notified
- An email that does exist, should let the user known that a confirmation email was sent. That email should then show up in the user's email inbox.

Testing results and follow up (frontend)

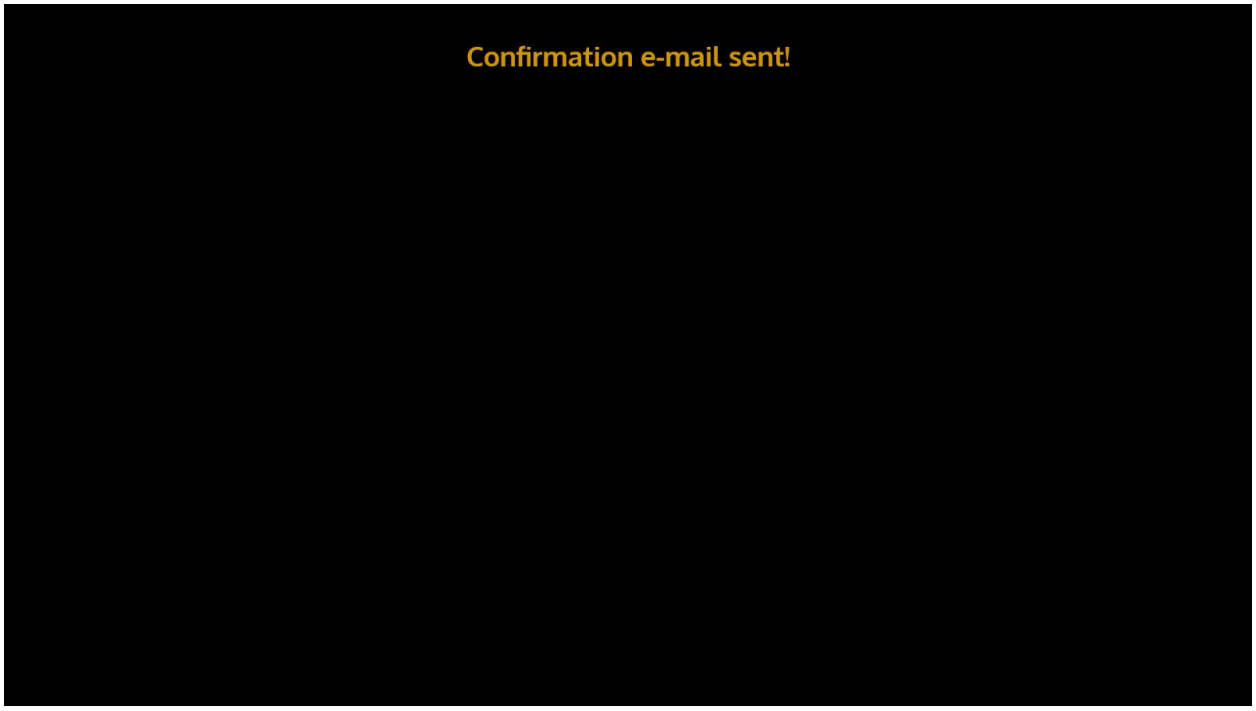
1. An nonexistent email entered lets the user know that is doesn't exist.



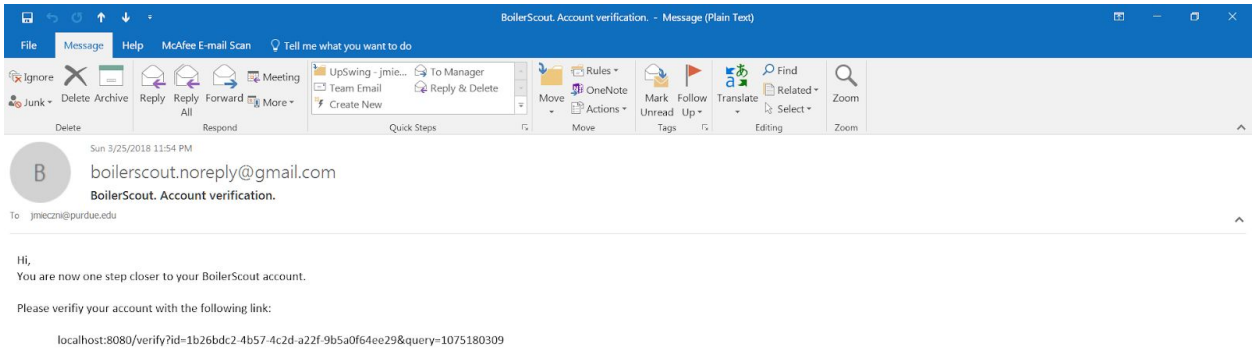
This email is not in the database, so an error is shown.

2. A Valid and existing email is entered. The user is informed that the email is sent. The email then is in the mailbox of the user's email.

This email exists. Submit is clicked.



The user is informed that the email was sent.



The email shows up in the inbox of what was entered.

Testing Summary(Frontend):

An incorrect email meets the user with an error, and a correct one sends the email to the user. This component is working perfectly.

Testing Set Eight: Resetting a password

Testing Procedure (backend): This is the protocol used to test resetting the password, and logging in to an account with the new one, from the backend perspective.

- When provided with a valid email, /send/forgot-pass sends an email with the new password, and changes the database to the corresponding hash of this new password. It also guarantees that the new one and the old one won't match.
- After this, the new password received replaces the old one for any use and becomes the only valid one. However, a new reset can always be requested.

Test results and follow up (backend):

1. First let's try to login to the account for hmontoya@purdue.edu.

POST localhost:8080/login

uthorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)


```
{
    "email":"hmontoya@purdue.edu",
    "password":"rYSukQICcu"
}
```


ody Cookies Headers (3) Test Results

Pretty Raw Preview JSON


```
{
    "user_id": "29883e6f-b475-4539-84b6-7a8f60d2fc85",
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJobW9udG95YUBwdXJkdWUuZWR1IiwidXN1ck1kIjoiMjk4ODN1NmYtYjQ3
        .NNVVMFj-X5-vO8pu6PCCbYSLXNQpMZ8XfLq2ssQrx1T3aEPmND1E4KZzxh9DyC3ejgjh--Y9GTs51_8kMYiw0g"
}
```


This represents a succesul login.

2. Now let's request a new password.

Hardy Gustavo Montoya

Hi,

You have requested a new password for your account.
Note that if you have recently requested more than one new password, the one contained in this email might be outdated. Please ensure this is the most recent email of this kind you have recieved.

Please login with the following password, and change it as soon as possible:

pt9ZQpHpsx

This is the email received, now let's try attempting to login in the same manner as step 1 (including the old password).

```
{
    "email":"hmontoya@purdue.edu",
    "password":"rYSukQICcu"
}
```

Cookies Headers (4) Test Results

tty Raw Preview JSON

```
{
    "timestamp": "2018-03-26T07:51:46.494+0000",
    "status": 500,
    "error": "Internal Server Error",
    "message": "[BadRequest] - Incorrect password provided!",
    "path": "/login"
}
```

We can see that the old password is now considered invalid.

3. Finally, attempt to use the new password received.

POST localhost:8080/login

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {
2     "email":"hmontoya@purdue.edu",
3     "password":"pt9ZQpHpsx"
4 }
```

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON


```
1 {
2     "user_id": "29883e6f-b475-4539-84b6-7a8f60d2fc85",
3     "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJobW9udG95YUBwdXJkdWUuZWR1IiwidXNlcklkIjoiMjk4ODNlNmYt
          .CRTu1zvyNNvT5VQRX5_IcAk7Bq9jiMRJ6OQr3Lin5W1132WTjZKeX5YapxWhPAcGJ3oQaonDmG48M_3Gqhhtrg"
4 }
```

The new password worked as intended, and allowed the user to login without any issues.

Testing Summary (Backend): Each time a user requests a new reset, he will receive a new password. This procedure proves that the information in database is correctly changed, including the hashing, as login method always hashes the password received before comparing it to the value in database.

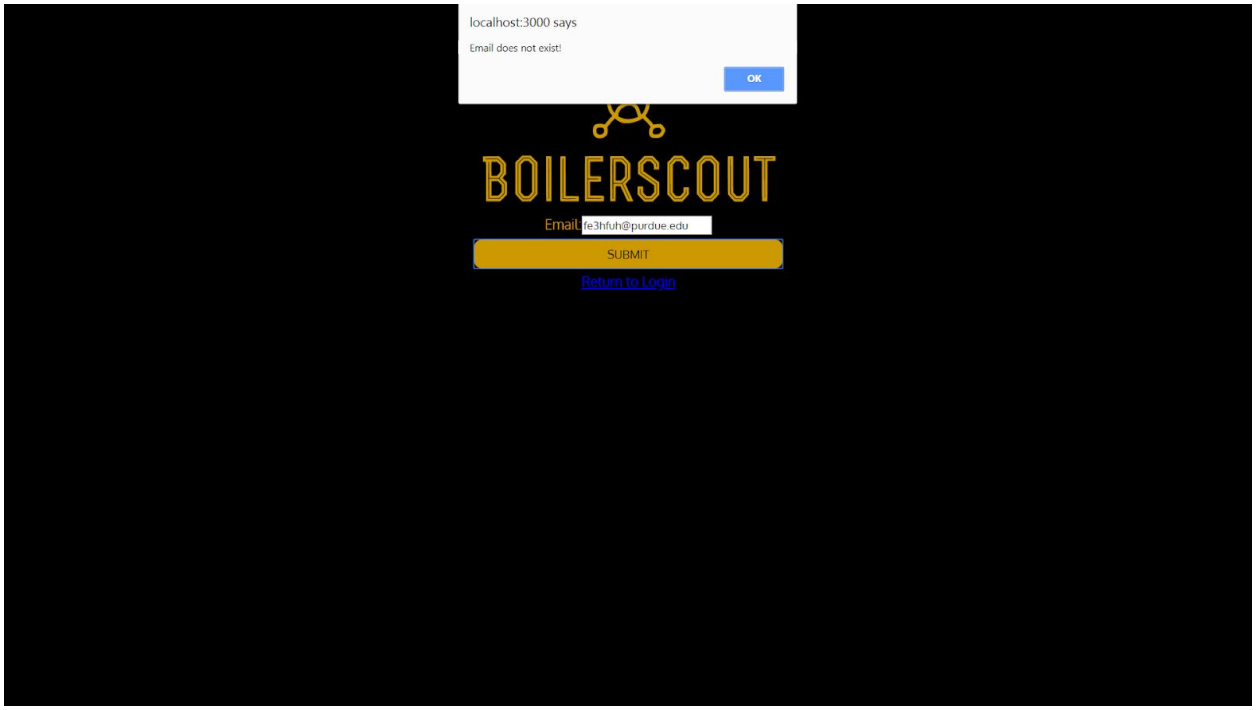
Testing Procedure (frontend): This is the protocol used to test resetting a password

- An invalid email is entered into the box. When SUBMIT is clicked, the user in notified that it is an invalid email.
- An existing email is entered into the box. When SUBMIT is clicked, the user is informed that the email was sent, and it shows up in his/her email inbox. Then the user can login with that new password.

Testing results and follow up (frontend)

1. An invalid email is entered into the box. When SUBMIT is clicked, the user in notified that it is an invalid email.

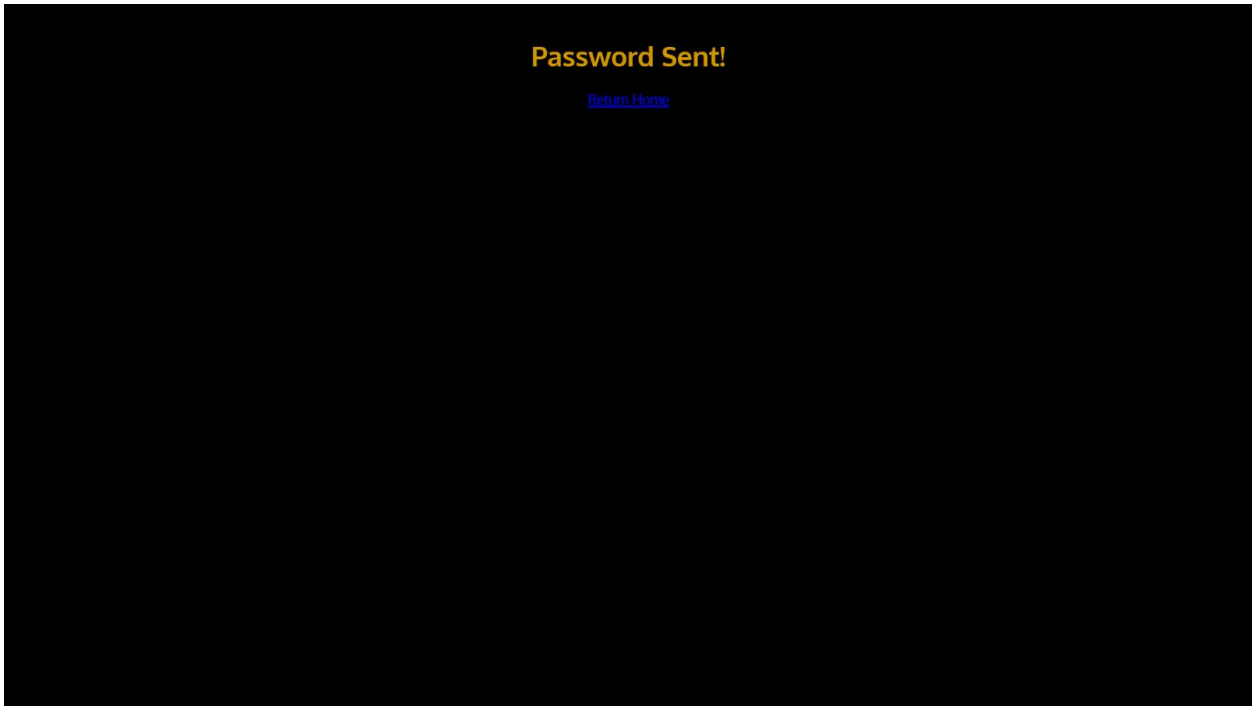
A bad email is entered.



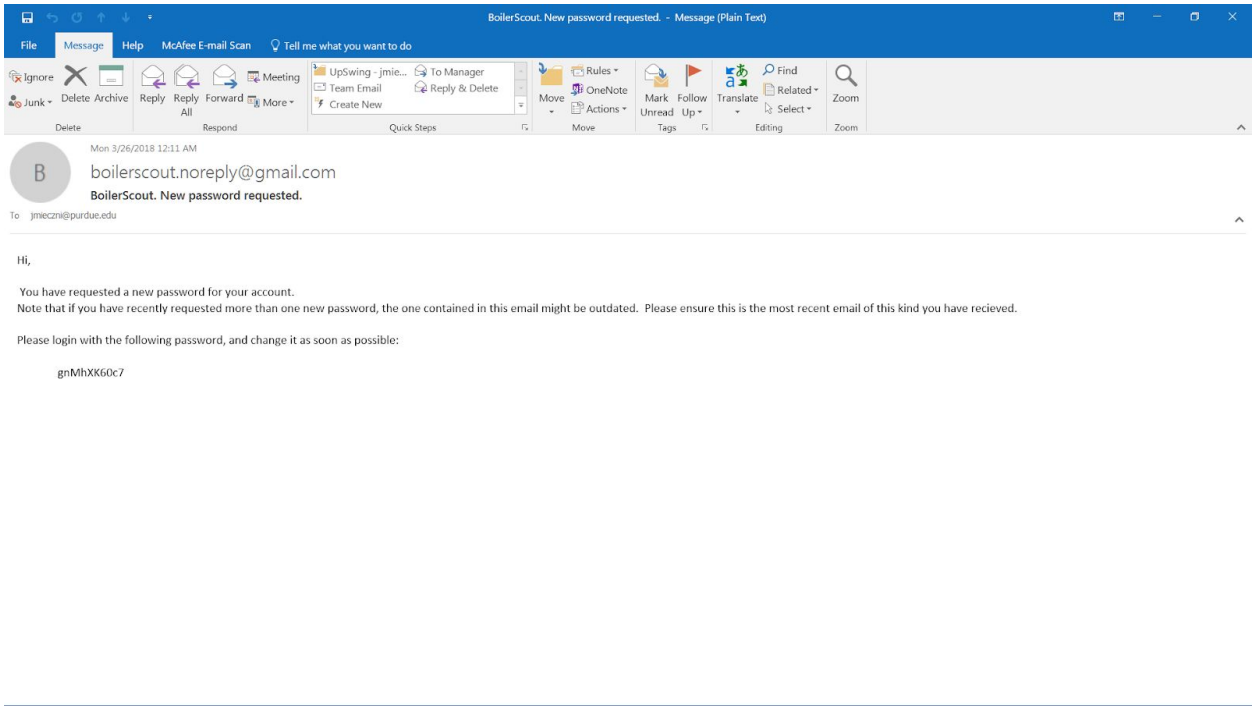
SUBMIT is clicked. The user is notified that the email is incorrect. No email is sent.

2. An existing email is entered into the box. When SUBMIT is clicked, the user is informed that the email was sent, and it shows up in his/her email inbox. Then the user can login with that new password.

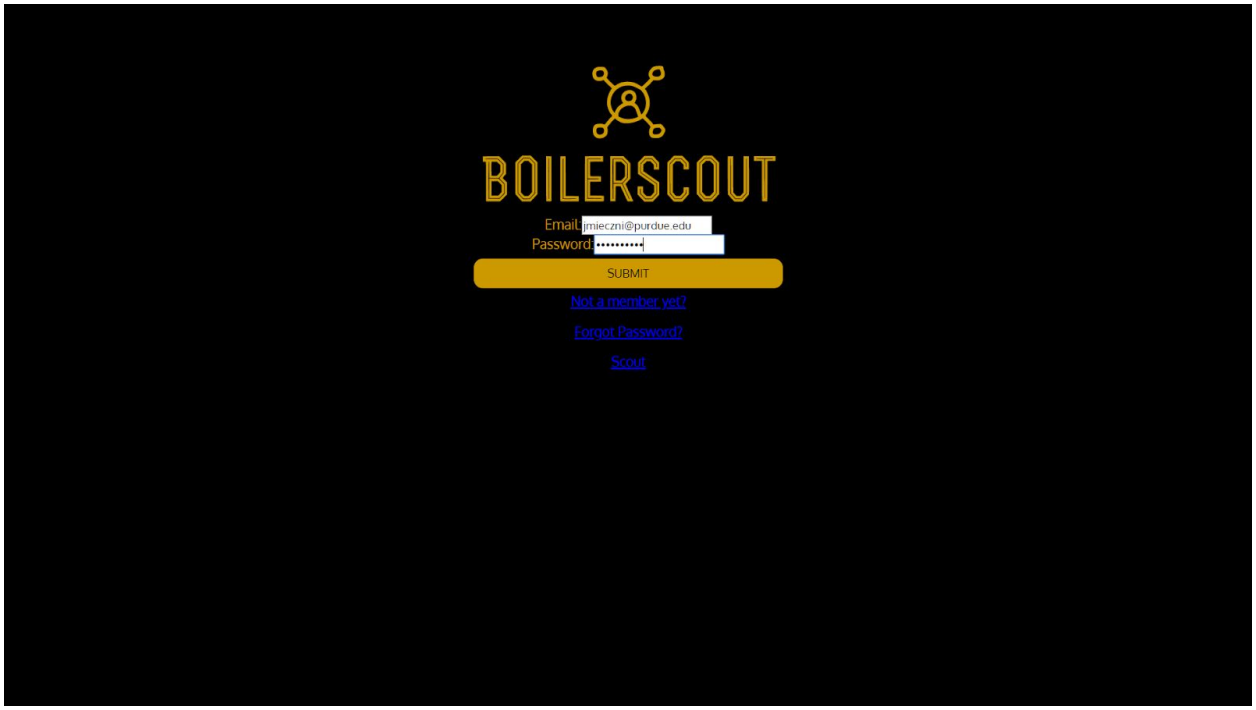
A correct, existing email is entered.



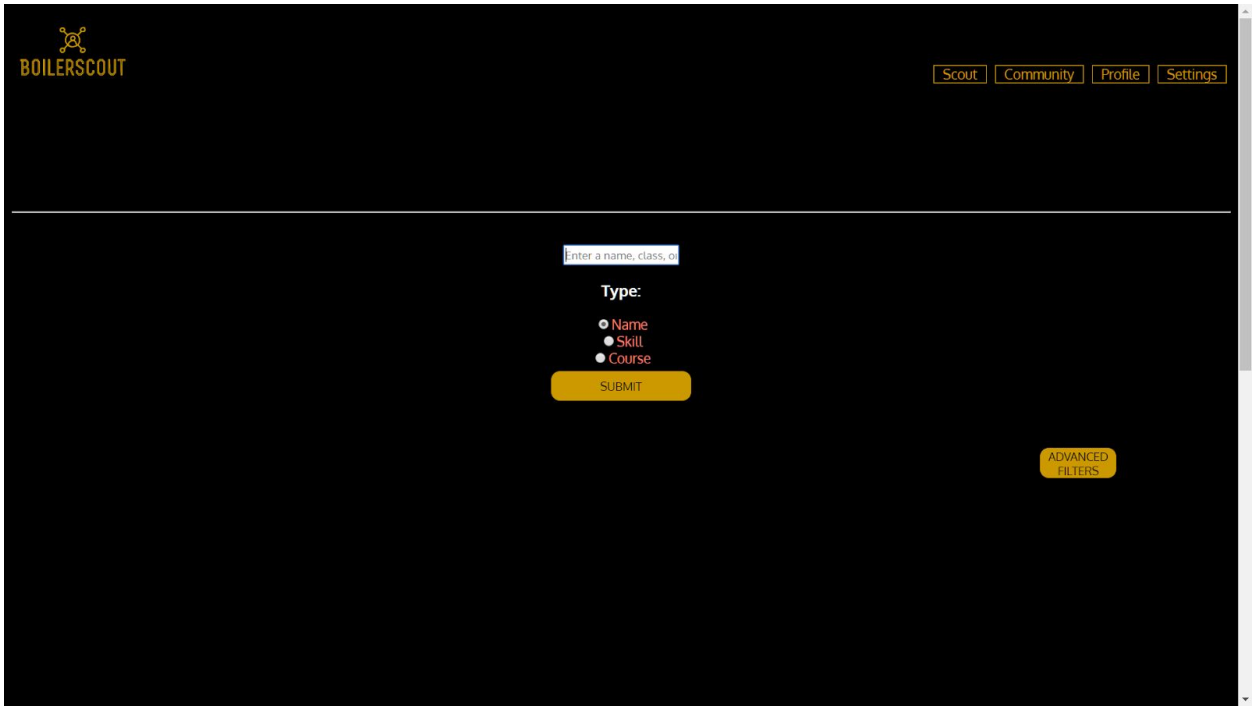
The user is informed that an email is sent.



The reset password email is shown in the inbox



The correct email and new password from the reset email are entered into login.



SUBMIT is clicked, and login is successful with the new password as the user was redirected to /Scout.

Testing Summary(Frontend): All functionality of this part works fully. Errors are shown correctly.

Testing Set: Signing up with an existing username

Testing Procedure: This is the protocol used to test sign up from a front end perspective.

- Enter in an existing email into the application with a valid password. This should result in letting the user know that he/she was not able to sign up.
- A non purdue email entered should not be submitted, as the form is disabled.
- A valid email entered with an invalid password should result in the user not being able to click the SUBMIT button, as the form is disabled.
- On a successful sign up, user should be displayed a page indicating so.
- After signing up, user should be able to login with their credentials.

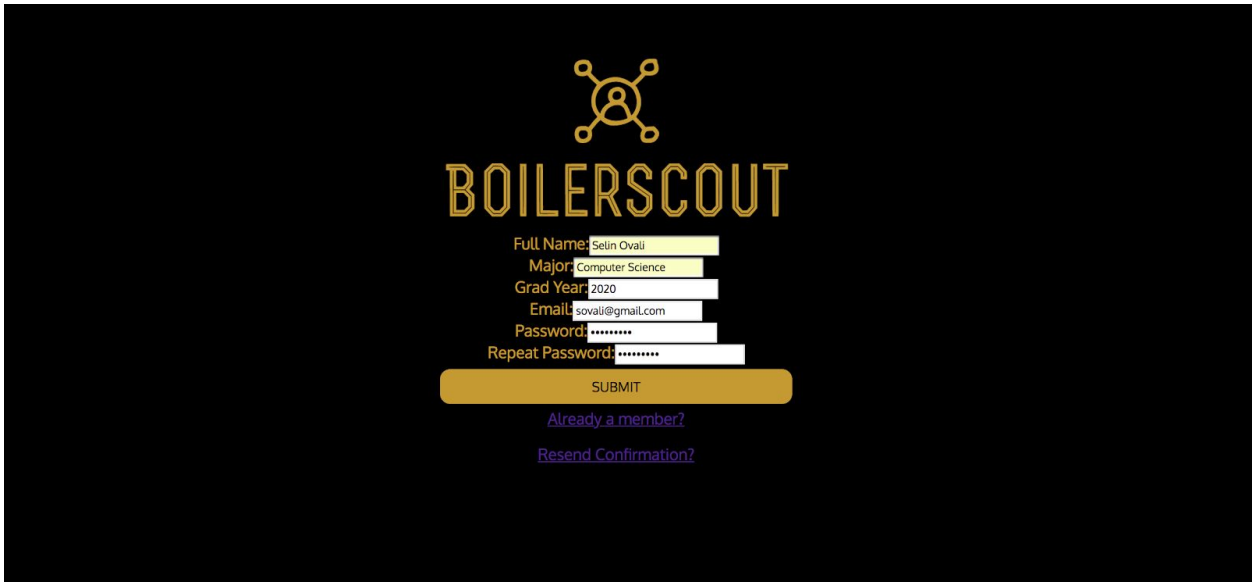
Testing results and follow up (frontend)

1. An existent email is entered into the sign up form.

SUBMIT button is clicked.

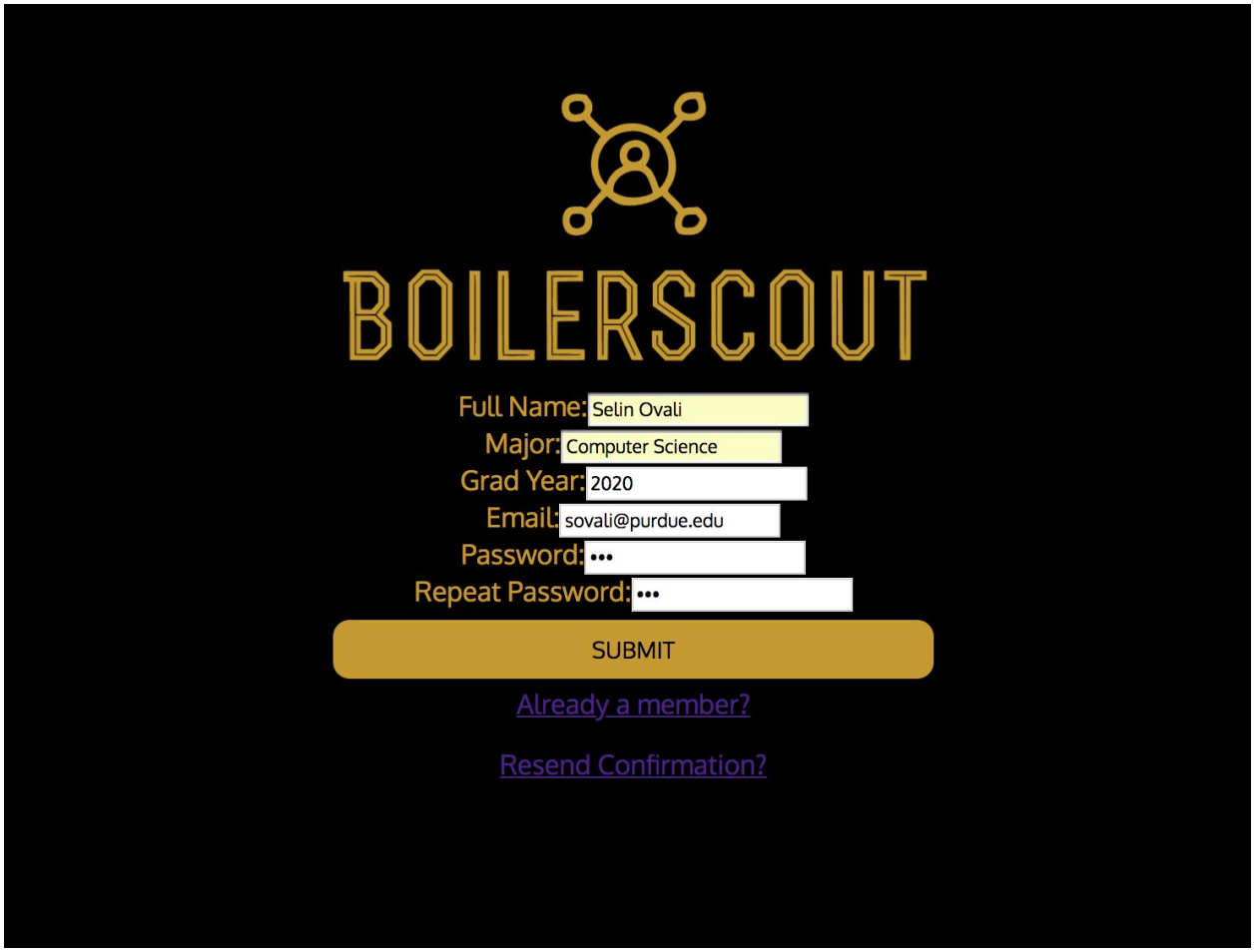
Appropriate error stating username already exists is shown. User is not signed up.

2. A non-Purdue email should have the form disabled.



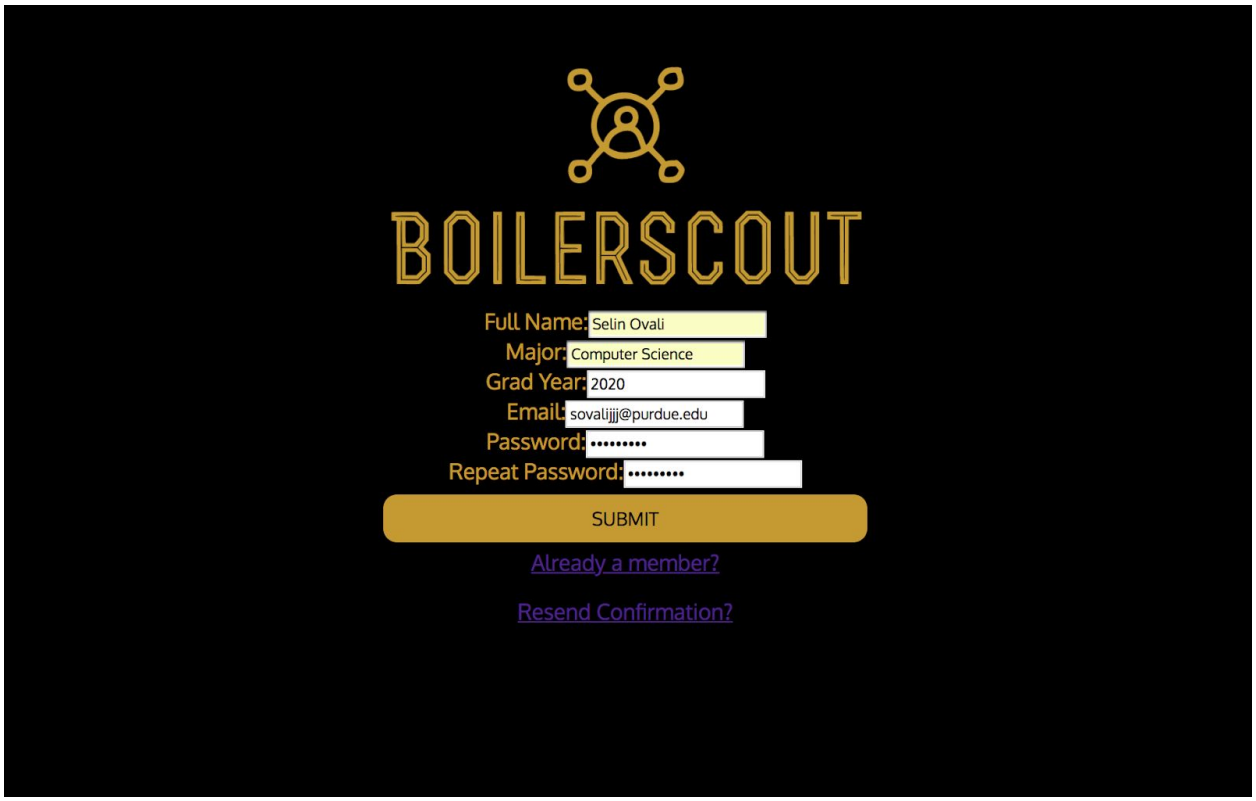
A non Purdue email is entered. User cannot click SUBMIT as there is no valid e-mail.

3. A valid e-mail paired with an invalid password should disable the form.



A valid e-mail and invalid password is entered. SUBMIT is clicked. As form is disabled, nothing happens.

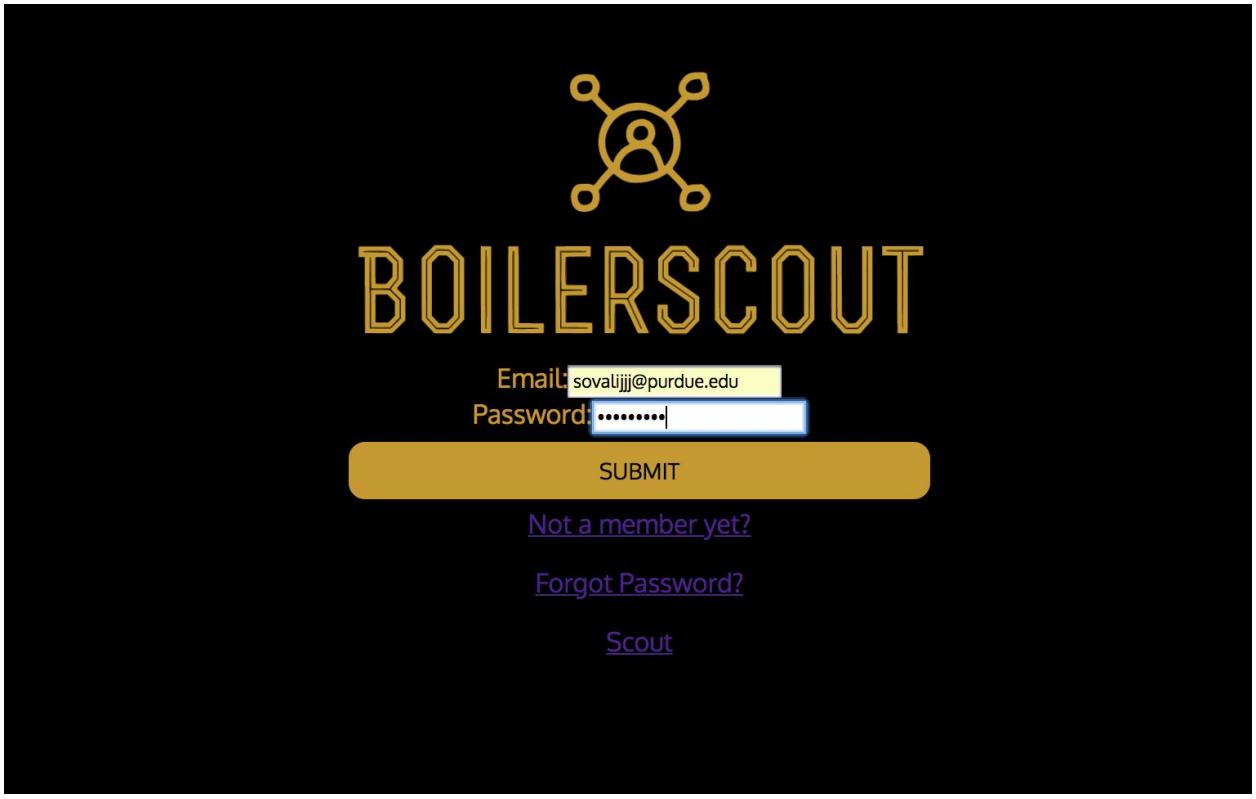
4. User enters the right credentials and can sign up successfully



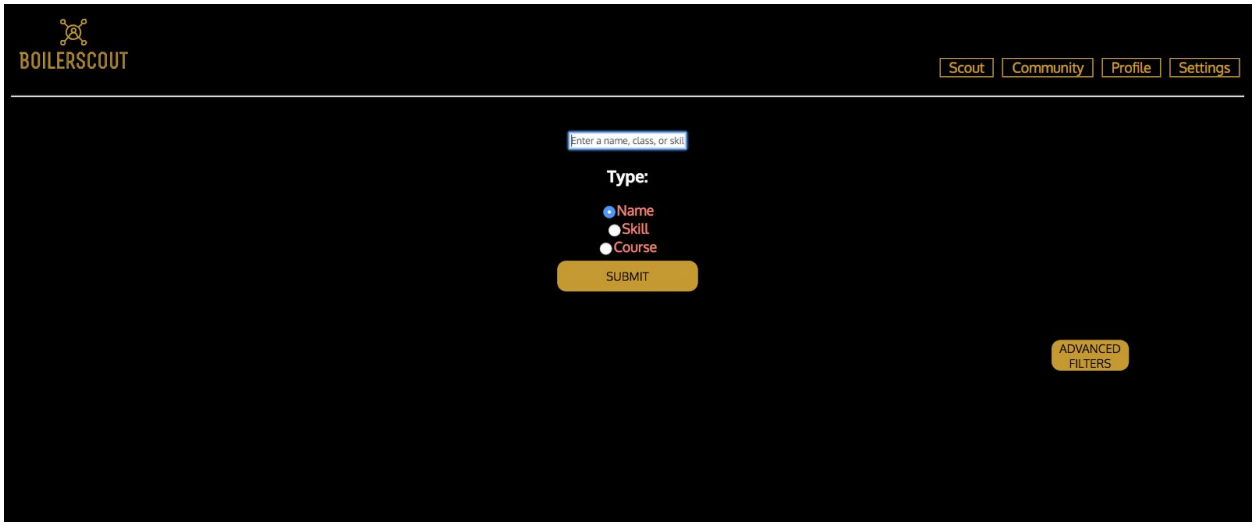
We enter a valid, non-existent e-mail and a valid password. SUBMIT is clicked.

User is presented with a page indicating that sign up was successful.

4. User should login with sign up credentials.



The e-mail/password used to sign up successfully is entered in the login page. SUBMIT is clicked.



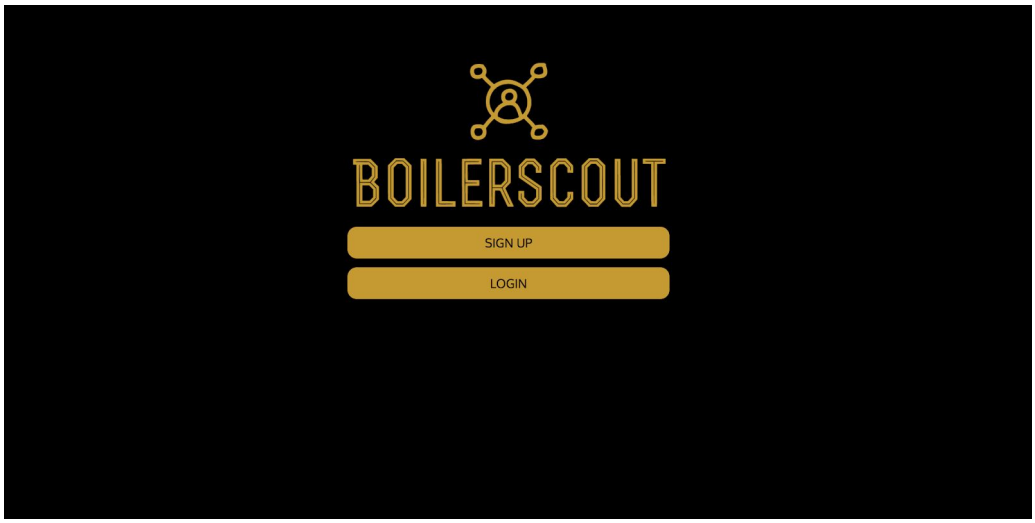
User has successfully logged in and is redirected to Home (Scout) page.
Testing Summary(Frontend): All functionality of signing up part works fully.

Testing Set: Top navigation bar and logo hierarchy

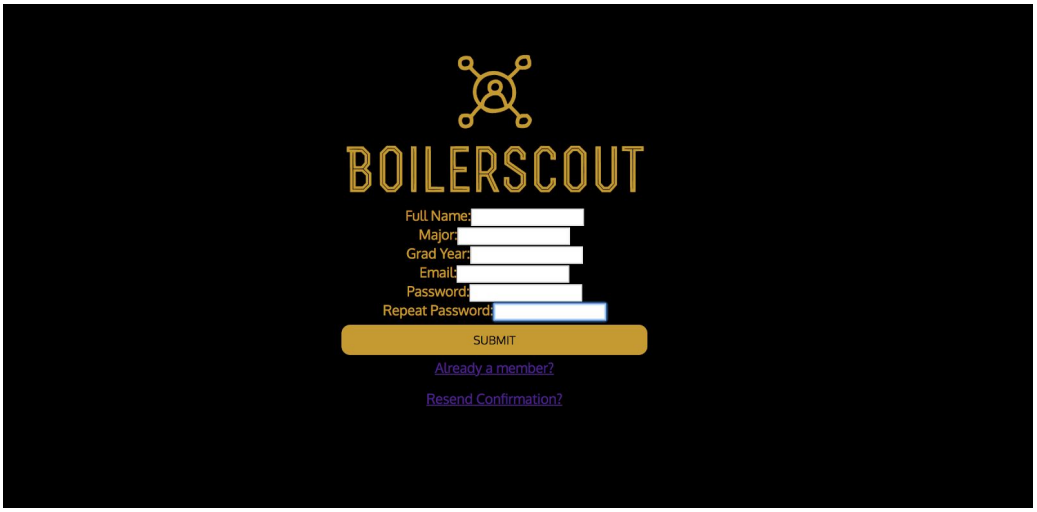
Testing Procedure: This procedure shows that top navigation bar and logo hierarchy has been fixed and appears at the correct pages.

- Go to pages before user has logged in such as Login, SignUp, Resend Confirmation & Forgot Password to see if logo is visible and navbar hidden.
- Go to pages after user has logged in such as Scout, Profile, Settings to see if navbar is visible and logo is hidden.

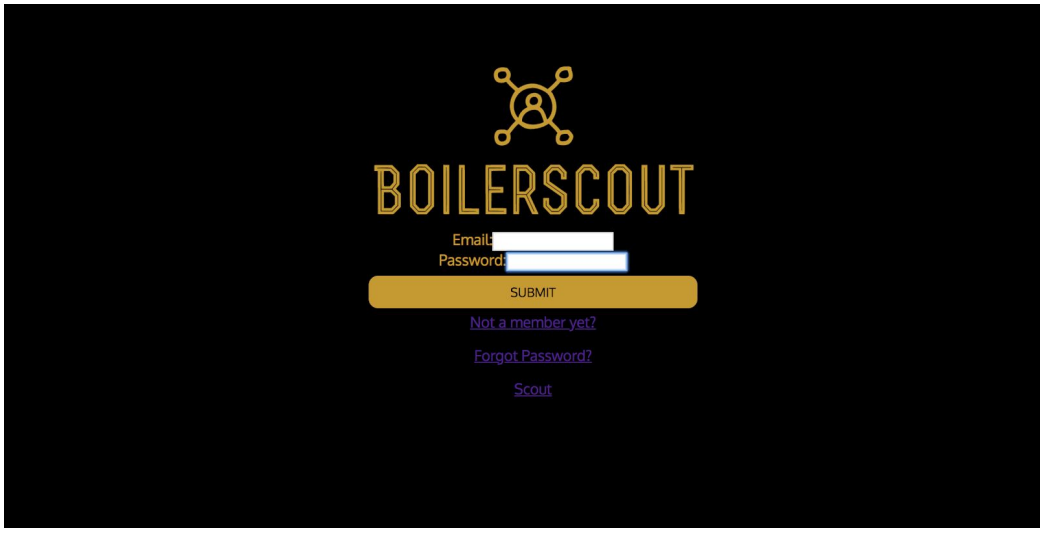
1. Pages before user has logged in are inspected.



The home page has logo as it should.

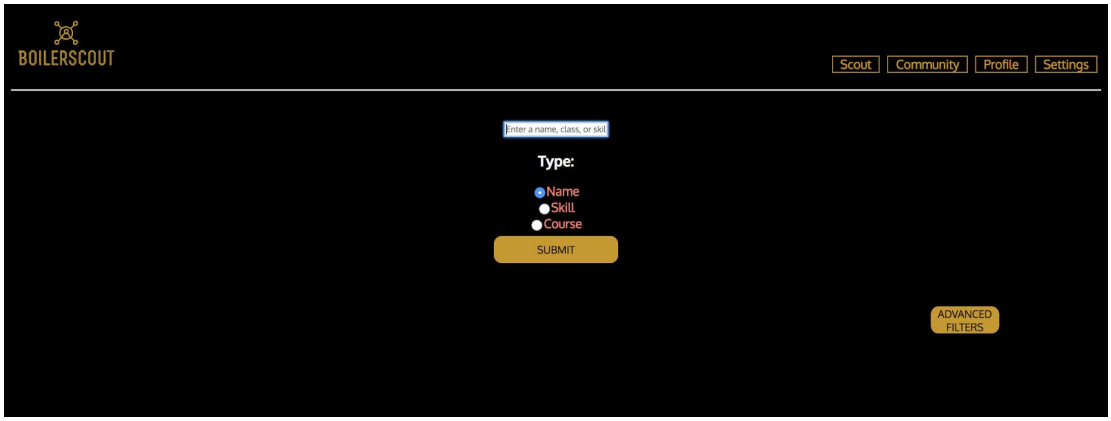


The sign up page has logo as it should.

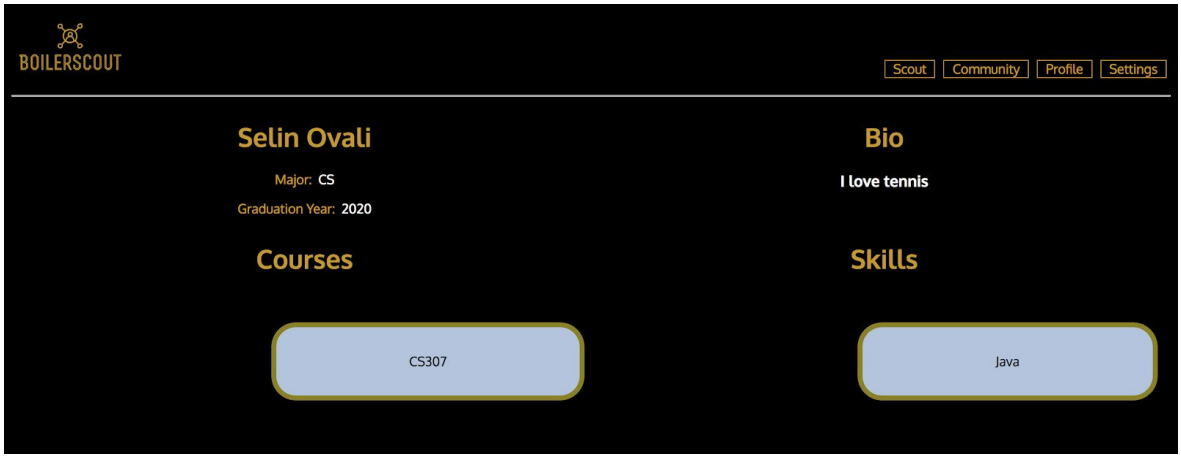


Login page has logo as it should.

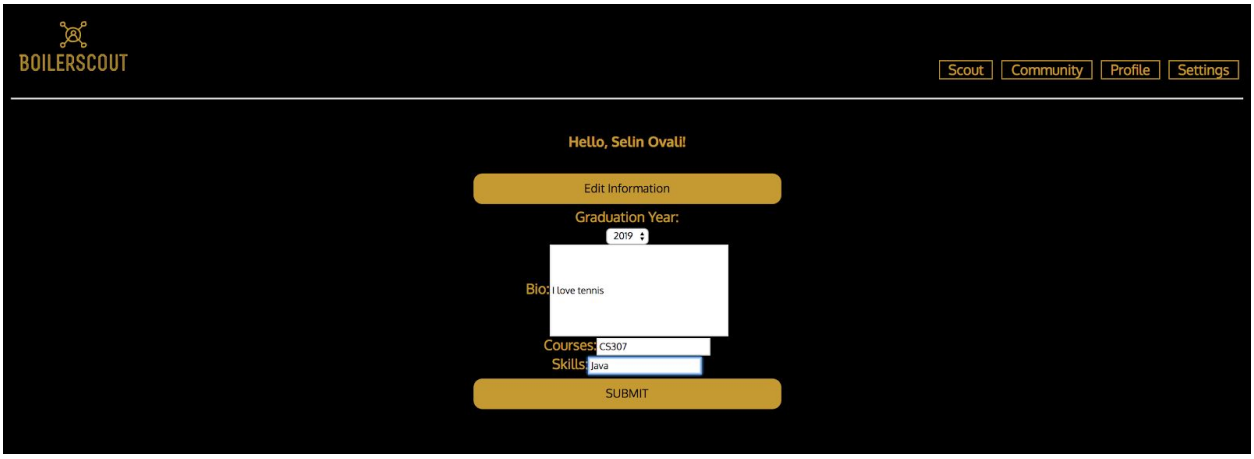
2. Pages after user has logged in are inspected.



Scout page has navbar as it should.



Profile page has navbar as it should.



Settings page has navbar as it should.

Testing Summary(Frontend): All functionality of navbar and logo works fully.

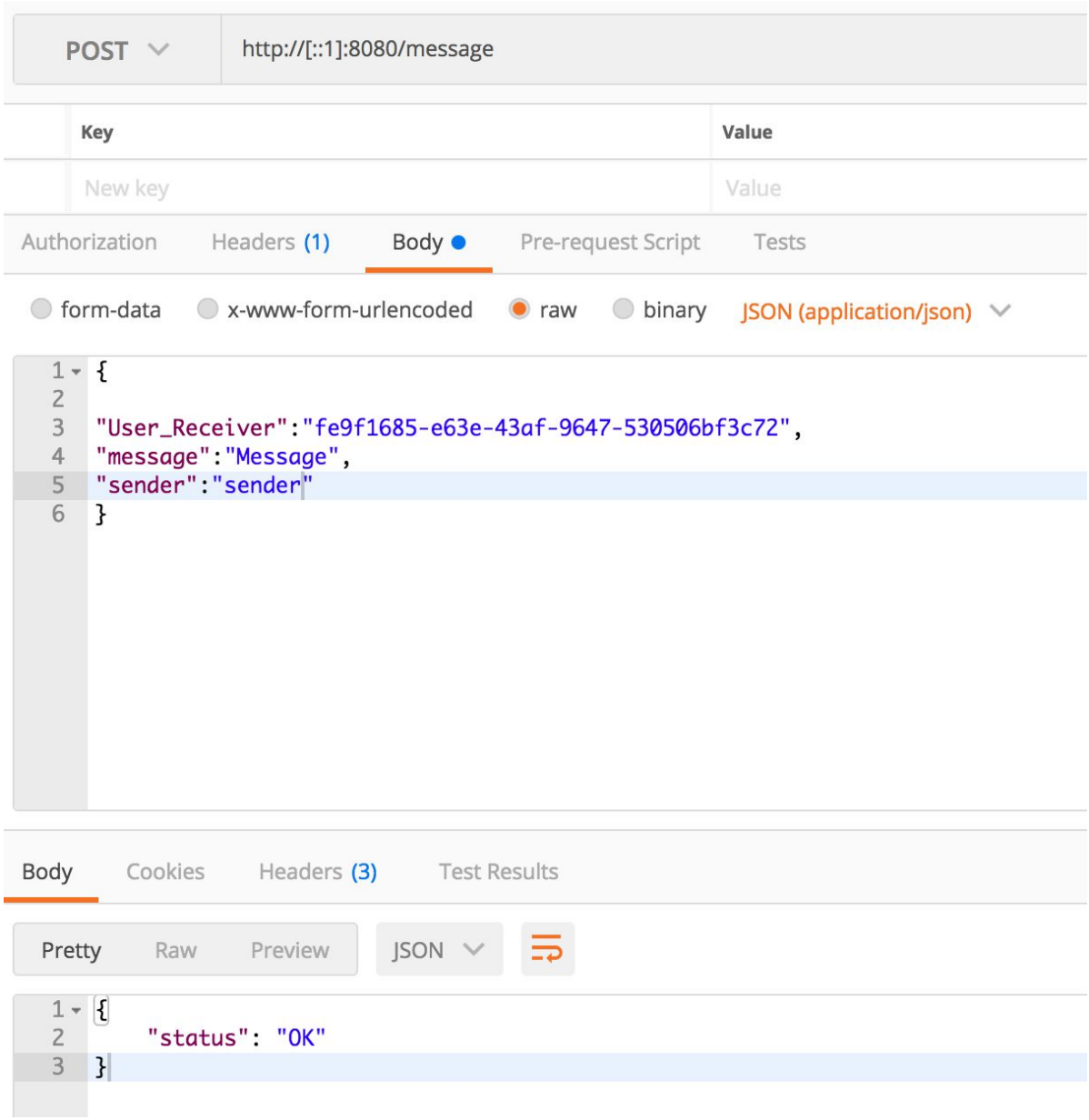
Testing Set Nine: Sending a private message to other users.

Testing Procedure (Backend): This is the protocol used to test getting messages and saving them in the database.

- Create a **POST** request on Postman that hits the endpoint /message
- Using an existing user in our database, create a request body that contains fields for User_Receiver, message, and the sender.
- Send the request via Postman and observe the response body, which should be a OK.
- If the user receiver don't exist get an error message.

Test results and follow up:

1. Requests made to get the ,User_receiver, messages, and the sender. They were all successful.(can be observed via Postman and MySQL workbench) The following images will demonstrate getting messages.



This image shows the request body sent to the /message endpoint with the necessary fields. The response body is also shown, showing a OK statues.

User_Receiver	message	sender
f	One two three test	Baris
fe9f1685-e63e-43af-9647-530506bf3c72	One two three test	Baris
fe9f1685-e63e-43af-9647-530506bf3c72	blbllbllblb	bd
fe9f1685-e63e-43af-9647-530506bf3c72	Message	sender

From this image we can observe that our entries are saved to the database.

2. If the user don’t exist, statues should return a error message by checking the existing user ids. This was successful when I tried to send a message to not existing user.

POST http://[::1]:8080/message

Key	Value
New key	Value

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{

"User_Receiver":"fe9f1685-e63e-43af-9647-530506bf3c72sd",
"message":"Message",
"sender":"sender"
}
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview JSON

```
{
    "timestamp": "2018-03-26T14:53:30.942+0000",
    "status": 500,
    "error": "Internal Server Error",
    "message": "User don't exist!",
    "path": "/message"
}
```

From the picture above, we can see that when we enter a not existing user as the User_receiver, we get a error message “User don’t exist!”.

Testing Summary(Backend): All possibilities were tested for sending messages, which there are two. If the user don’t exist, we get an error message. If the user exists, message is sendible and the data is saved to the database.