Assignment 3

Student ID: AC3837

Student name: Thanaphon Sombunkaeo

Group ID: TTV19S1

First part A5:2017-Broken Access Control (10 pts)

1. [Other] Real Broken Access Control bugs (3 pts, 1 point each)
    1. **Facebook Business Pages**: facebook user can use HTTP request to assign admin permission to themselves without permission. The request simple as shown below

        ```
        POST /<page_id>/userpermissions HTTP/1.1
        Host : graph.facebook.com
        Content-Length: 245
        role=MANAGER&user=<target_user_id>&business=
        <associated_business_id>&access_token=<application_access_token>
        ```

        It was high serverity because facebook page is used widely. Since attacker can make them as an admin of page, they can remove other admins and completely hijack that page.

        The mitigation is to check to role, permission of who sent the request whether they can grant the admin permission to page or not.

    2. **Steam**: There is an improper access control in the steam that security reseacher can get the CD keys of any game. The step is to use endpoint */partnercdkeys/assignkeys/* on *partner.steamgames.com*, they can download previously generated CD keys that they should not have permission to do it.

        It was medium serverity because only security researcher can do this bad action (not any anonymous user).

        The mitigation is to always check the role of that endpoint whether use have a proper role on the action.

    3. **YouTrack**: Any user can read the issue description of other user that they should not have permission. The endpoint is not protected by role validation.

        It was medium serverity because although user can only read the issue description of another user, that issue description might contains sensitive content.

        The mitigation is simply validate the permission of user when access that endpoint.
2. [Issue report] **Target => Juice Shop**: View another user's shopping basket (3.5 pts)

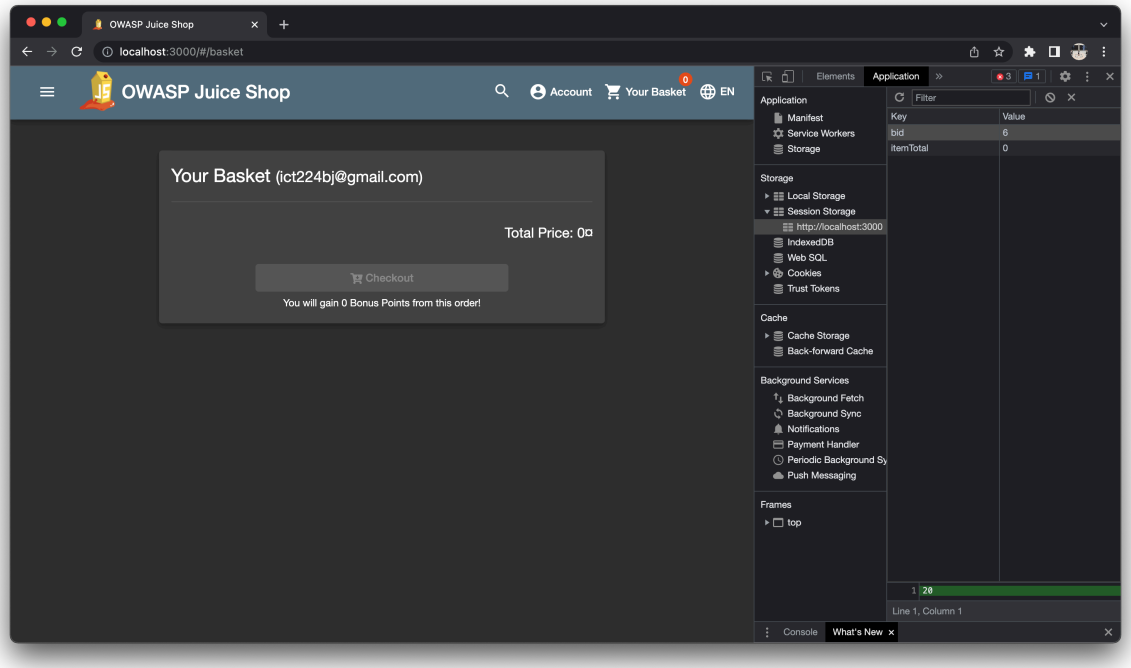
    Title: User can view another user's shopping basket without permission

Description: Juice shop store basket ID which is a running number (1,2,3,..) in the session storage. User can change the bid value and refresh the basket page to see basket of another user.

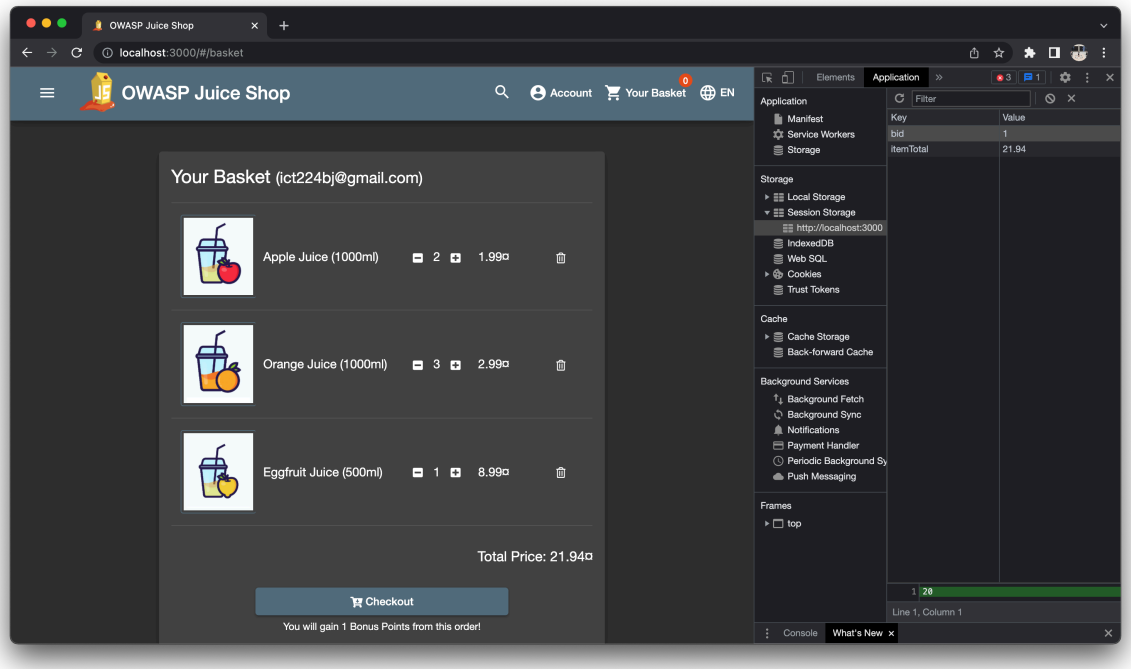
Step to produce:

- Login as any user you have registered.
- Go to basket page `/#/basket` and open session storage of `localhost:3000` in development tool
- Change the bid which might stand for basket ID to another value.
- Refresh the page and you will see the basket of another user.

Before change bid screenshot



After change bid (basket id of another user)



Impact Estimation: Medium serverity because attacker can only view and edit the basket of another user. But in order to complete the order, they need to use their own address and payment method.

Mitigation:

- Avoid storing basket ID as a running integer in the session storage because attacker can random another basket ID easily and view that basket.
- Avoid storing basket ID in session storage. They should store the basket ID of user in the database. When user go to basket page, they can only see their own basket.

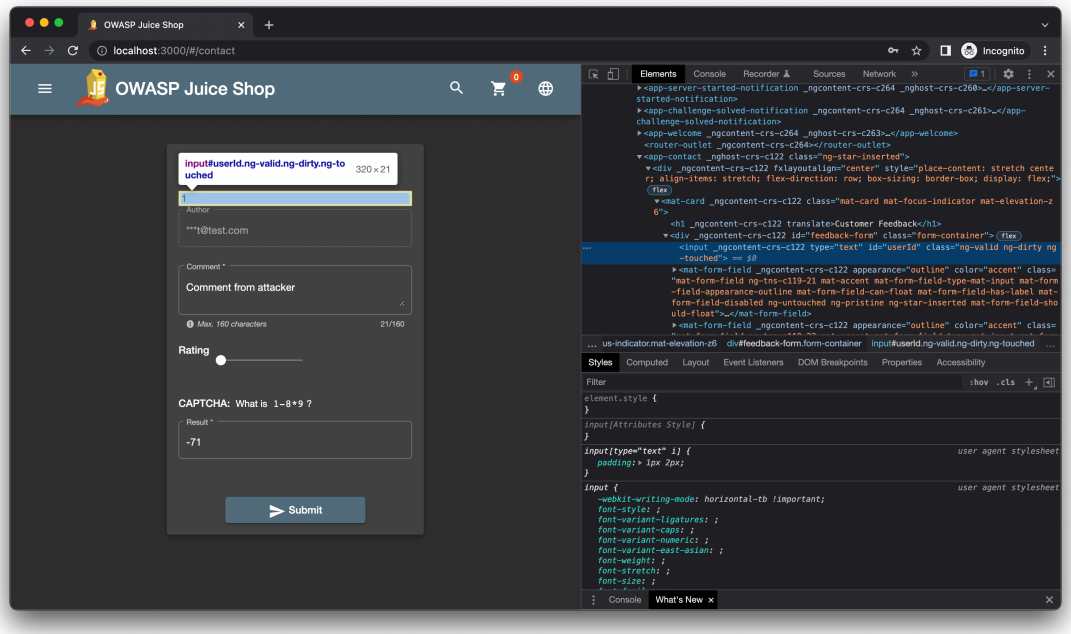
3. [Issue report] **Target => Juice Shop**: Feedback in another users name (3.5 pts)

   Title: User can post feedback as another user name due to the broken access control.

   Description: Logged in user can post the feedback in another user name by inspecting the form input and change the value of user ID to be another user ID. The system considers the owner of feedback from the user ID field in the payload of request. That means there is the broken access control in the system.

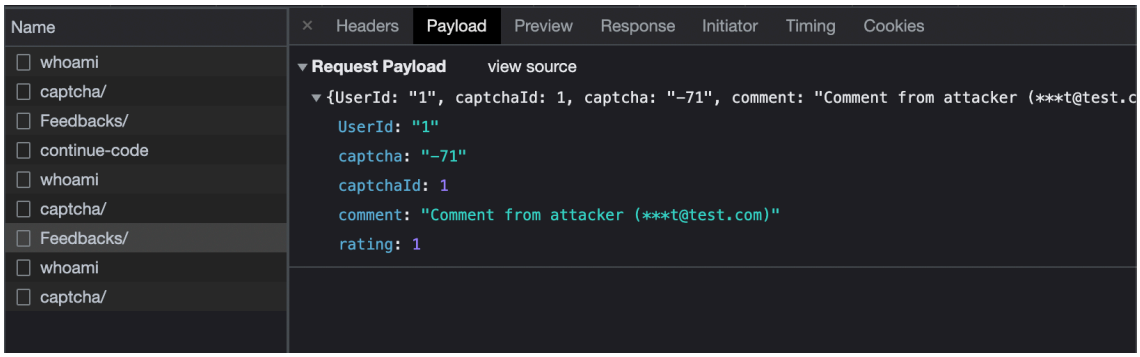
   Step to produce:

   - Log in as any user that you registered
   - Go to customer feedback page `/#/contact` and inspecting form element. Then delete the hidden keyword as shown below



   You can edit that input field to be another value. In this case, I will change it to be 1

   - Write comment and add rating. Now the owner of this feedback will be the user that has ID match to the id that we fill in.



Impact Estimation: Medium serverity because attacker can only write the feedback as an another user, they cannot control the whole accound.

Mitigation:

- Avoid sending user id in the payload of request especially user id that is a running number, attacker can change them and that feedback will be consider as another user.
- Use user ID from JWT as an authorized signature.

Second part A8:2013-Cross-Site Request Forgery (5 pts)

1. [Reading Report] RWBH Chapter 4: Cross-Site Request Forgery (pp. 29-40) (2 pts)
    - Not all authentication methods could be used with CSRF attacks (like non-cookie JWTs), but the two presented in the book can. What are those? (1 point)
        - Basic authentication protocol
        - Cookie
    - Describe briefly how you can mitigate CSRF attacks (1 point)
        - Require CSRF token for every request that are submitted. CSRF token is a string that is generated for user and use it as a signature of user. CSRF token can store on client cookie, so attacker have no way to get the that token and use it to generated malicious code. We can embed the token in the form input and hidden it as shown below.

```
<form action="http://some-site/some-path" method="POST">
  <input type="text" name="someName" value="someValue">
  <input type='hidden' name='csrf'
value='lHt7DDDyUNKoHCC66BsPB8aN4p24hxNu6ZuJA+8l+YA='>
  <input type="submit" value="submit">
</form>
```

2. [Issue report] **Target => Juice Shop**: CSRF username (3 pts)

    Title: Attacker can perform CSRF to change the username of juice shop account.

    Description: Attcker can send a link of their website to some user which has a form of changing username of juice shop account. When user press button to submit the form, POST request will be called and username will be change to the desired value that attacker provide.

    Step to produce:

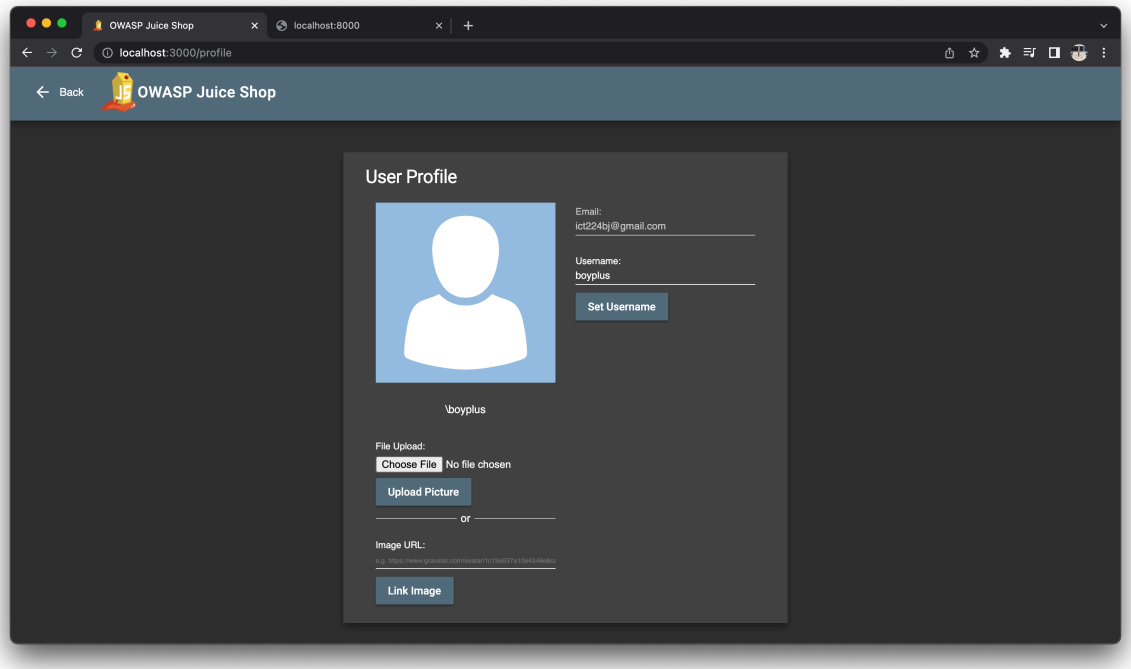
- Write HTML to perform CSRF username modifying as shown below (the desired username will be in value of input field)

```
<!DOCTYPE html>
<html>
<body>
  <form action="http://localhost:3000/profile" method="POST">
    <input type="hidden" name="username" value="attacker">
    <input type="submit" name="Set Username">
  </form>
</body>
</html>
```

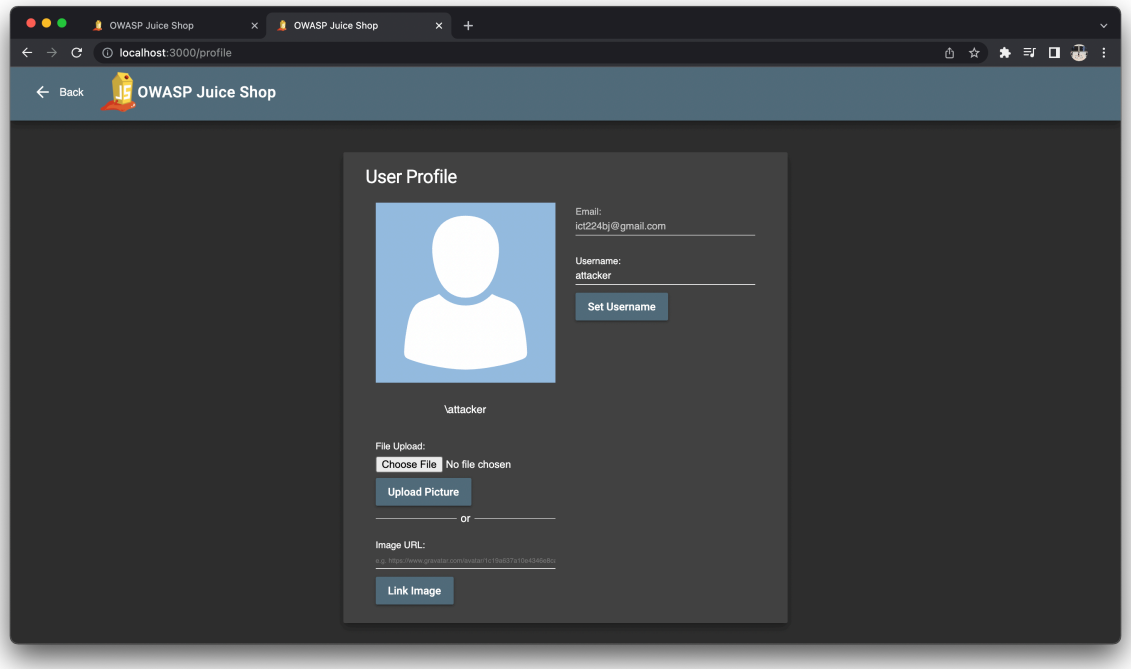
  Run this file on port 8000 by using command `python3 -mhttp.server`

- After press submit button, it will redirect to juice shop profile page. You will see that the username is changed.

  Before change username screenshot



After change username screenshot



Impact Estimation: Medium serverity because attacker can only change the username of victim and the victim can change it back anytime. In addition, if user did not click on the link that attacker sent to the, the action will not happen.

Mitigation:

- Add checking value of the Origin or Referer to ensure that it conatins the expected value. This approach works because attacker cannot change this value remotely.
- Use strict samesite cookie, the browser will not send the cookie with the HTTP request that does not created from the original.

Third part A6:2017-Security Misconfigurations (5 pts)

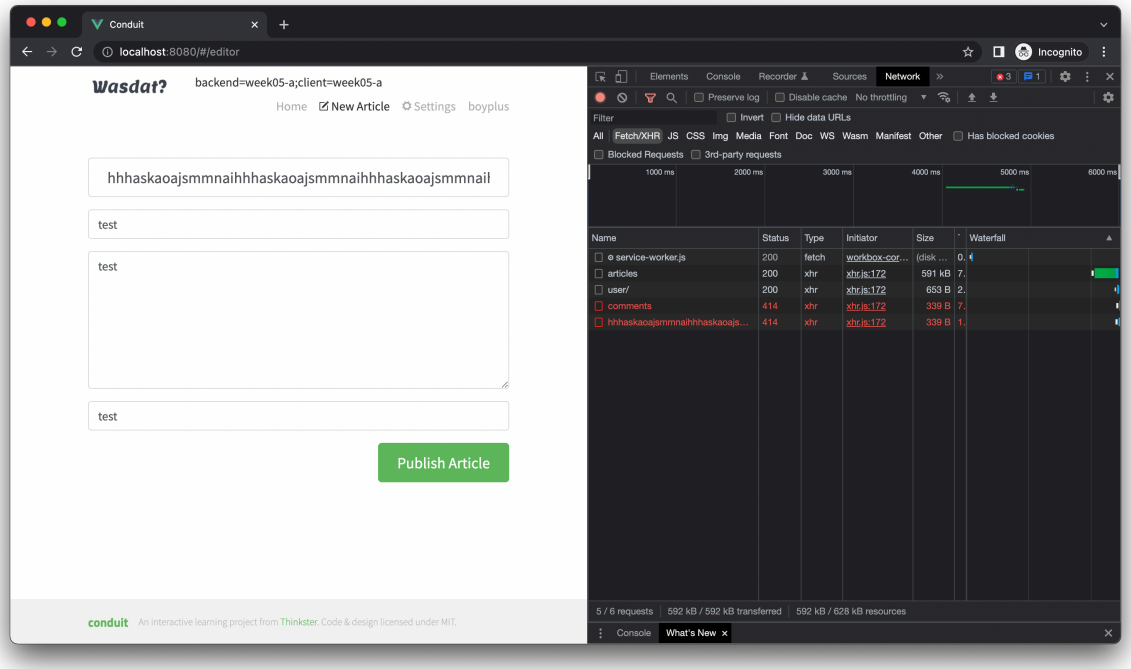
1.

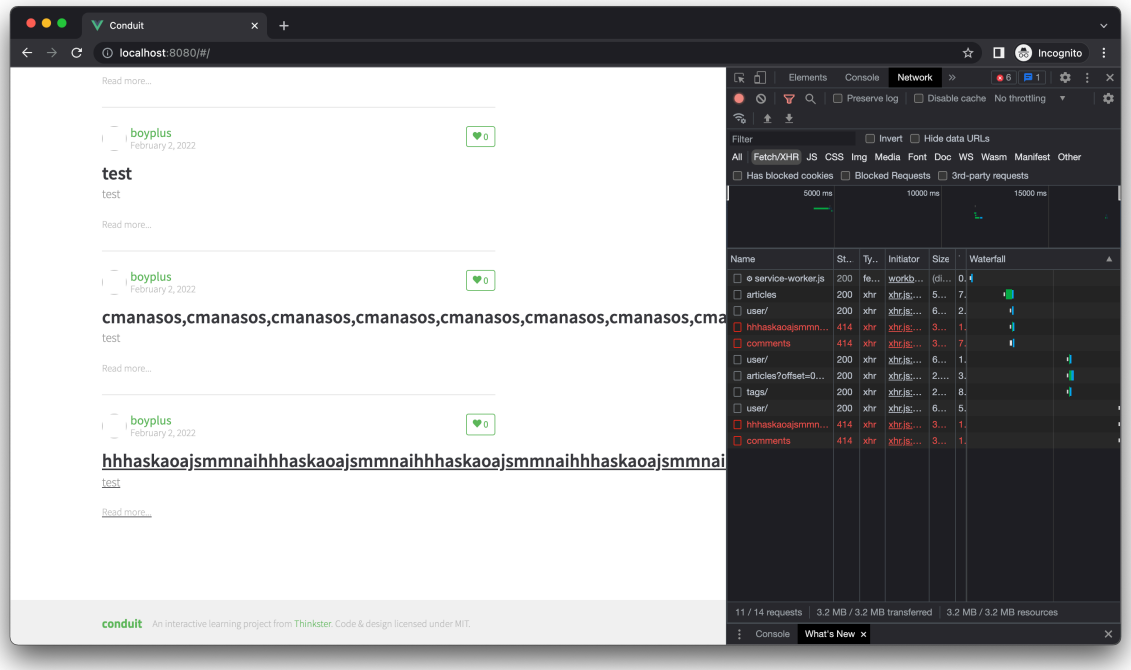
Title: User is allowed to publish a article with very long title article which lead to UI broken.

Description: There is no form validation to validate the length of article title. So when user publish the article successfully, that post will be shown on global feed tab. The result look broken as shown in step to produce. In addition, user cannot see the detail of article since wasdat use title as a URL path (it's too long and will get 414 code response).

Step to produce:

- Go to `http://localhost:8080/#/editor`
- Fill in article with very long title e.g. 400,000 characters and all other input field. Then publish article.
- It will not redirect us to the article page because it get 414 response, but article was created successfully. We can go to main page and clcik on the article that we have just create manually.





Impact estimation: Medium serverity because it can broke the UI of application and user cannot click to see the detail of article which means it cannot edit or delete by owner.

Mitigation:

- Add form validation to validate the length of article title in both front-end and back-end. In addition, frontend should display the long title properly e.g. display only first 20 characters.

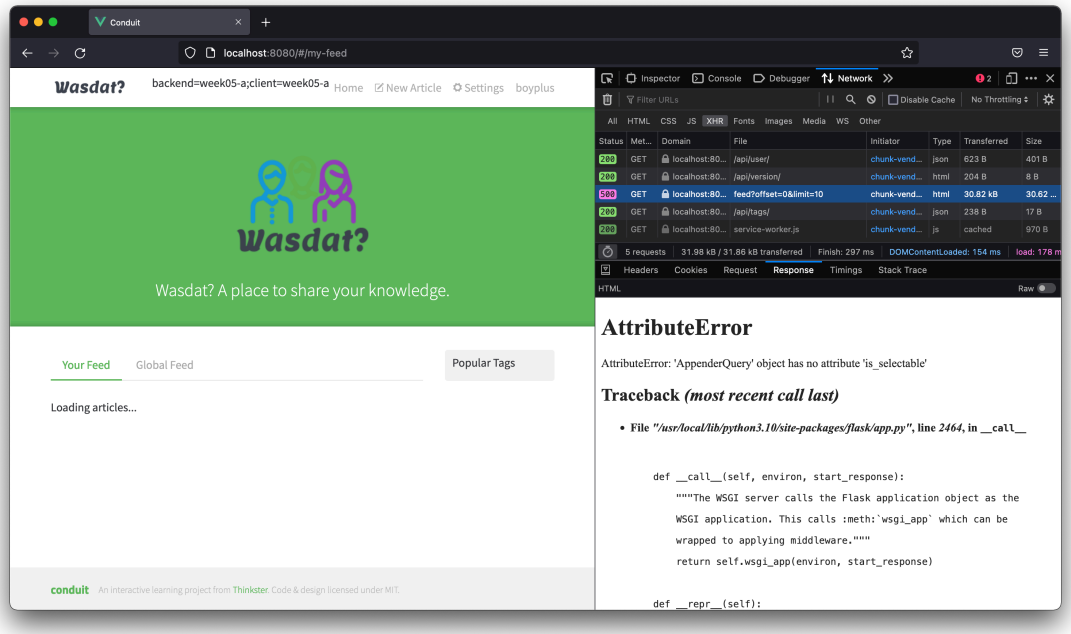
2.

Title: Wasdat did not throw the error in `/#/my-feed` properly.

Description: If user have their own article and go to `/#/my-feed`, they can see the 500 Internal server error and UI display loading article... that means both fron-end and back-end did not handle this kind of situation properly. In addition, User can see the traceback in the API response.

Step to produce:

- If you have not publish any article, just publish one article.
- Go to `/#/my-feed` and open development tools to see the error on GET `/api/articles/feed` as show below



Impact estimation: Medium serverity because it allows user to see the traceback of error that means user can see the detail of error and system e.g. attribute in database (is_selectable in this case), back-end framework (python with flask)

Mitigation:

- Avoid response error with traceback in API, use another format instead e.g. error message to tell user that there is an error to fetchs your feed.