INFO3180 Lab 7 (20 marks)

VueJS and File Upload Forms
Due: ~~March 31, 2020~~ May 3, 2020 at 11:55pm

The purpose of this lab is for you to build an upload form with VueJS and submit it, sending the data via an AJAX request to be processed on the backend.

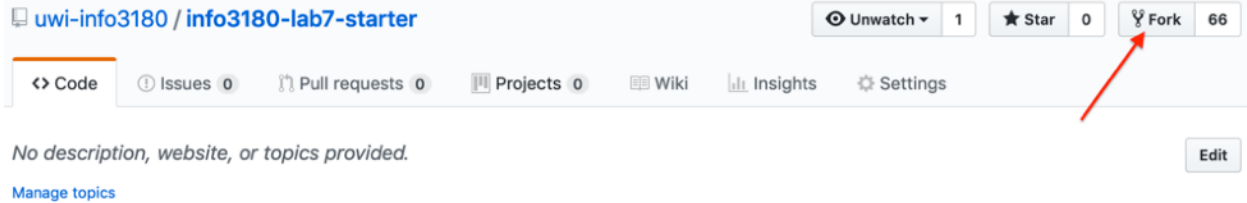| Learn VueJS |
| --- |
| Here are some helpful links to learn about VueJS.

Why VueJS? https://player.vimeo.com/video/247494684
VueJS Guide: https://vuejs.org/v2/guide/
Learn VueJS 2 Step by Step: https://laracasts.com/series/learn-vue-2-step-by-step |
| **Debugging VueJS** |
| It is recommended that you also install the VueJS Devtools browser extension to help with debugging your VueJS application. You may follow the instructions and download the extension at the following link:

https://github.com/vuejs/vue-devtools#vue-devtools |

Fork and Clone the Repository

Start with the example at: https://github.com/uwi-info3180/info3180-lab7-starter

Fork the repository to your own account

In github.com rename it by going to settings and changing the name to **`info3180-lab7.`**
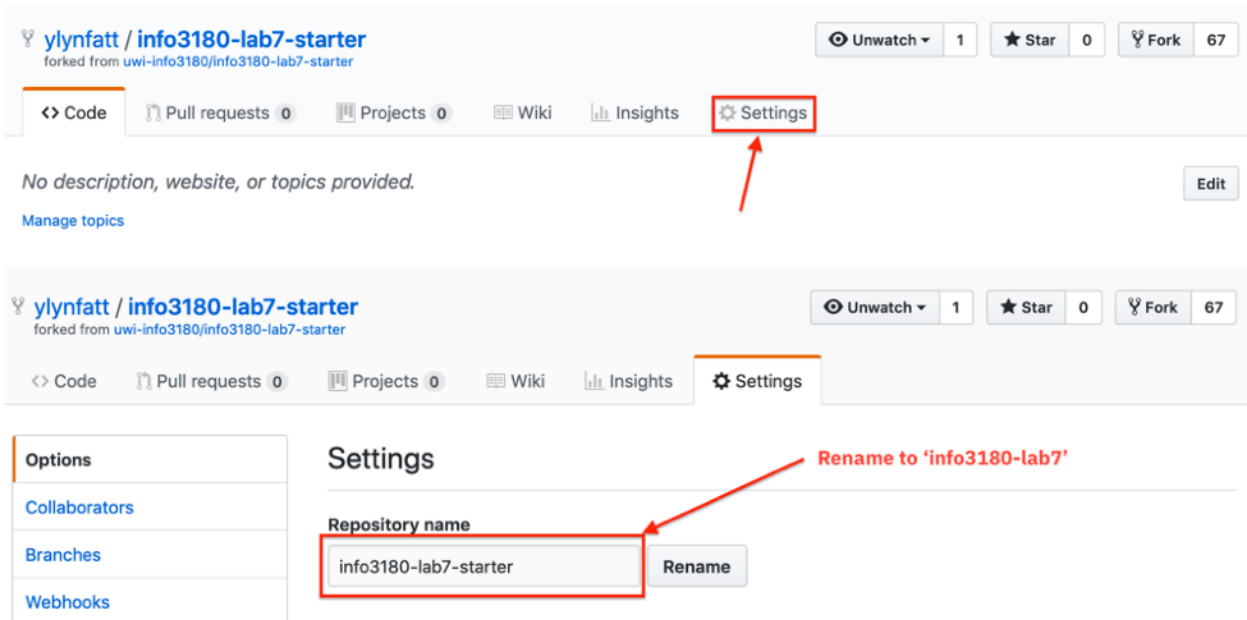
To start working on your code, clone it from your newly forked repository for example:

If you are working in Cloud9 or if you are working locally on your own computer (ie. NOT in Cloud9) :

```
git clone https://github.com/{yourusername}/info3180-lab7
```

Now, as always ensure you remember to activate your **virtual environment BEFORE** doing any package installation with pip or running python!

Now install the requirements and start the development server.

```
pip install -r requirements.txt
python run.py
```

Exercise 1 - Build the Form using Flask-WTF (5 marks)

You have been provided with starter code at the link above which has an empty file called **forms.py**. Create a form called **UploadForm** that has two fields. A TextArea field called '**description**' that requires a user to fill in some data and a FileField called '**photo**' that only allows images to be uploaded.

Commit your code to your Github Repository.

Exercise 2 - Flask API Route to process form (5 marks)

Use your work from Exercise 1 and integrate it with a flask route **/api/upload** and view function called **'upload'**. You should ensure that this route only allows **POST** requests (no GET requests are allowed) and should validate user input on submit. The route should save the file to an uploads folder and return the **filename**, **description** and a **message** in a JSON format similar to the example below:

```
{
    "message": "File Upload Successful"
    "filename": "you-uploaded-file.jpg"
    "description": "Some description for your image".
}
```

If the validation fails however, we should return a list of errors in JSON format similar to the example below:

```
{

    "errors": [
        {},
        {}
    ]
}
```

You have been given a function called **form_errors()** in your **views.py** file that you can use.

Commit your code to your Github Repository.

Exercise 3 - Create the front-end to display the form using VueJS (10 marks)

Create the front-end for your application using VueJS that will display the upload form. When the submit button is clicked it should make an AJAX request to the API route (endpoint) you created in Exercise 2.

Start by first creating a new VueJS component called '**upload-form**' in your **app.js** file. This component should have a '**template**' property that has the HTML code for the form. You will have to manually create the form fields. Remember to use the backticks (` `` `) to surround you HTML code so you can have it span multiple lines if necessary.

On the **<form>** tag, you will use the VueJS directive **@submit.prevent="uploadPhoto"**. This will ensure that when the submit button is clicked or if the user press the ENTER key on their keyboard that the function **uploadPhoto** (which we will define next) in the quotes will be called. The **.prevent** will ensure that the default action for the form will be prevented. This is similar to the **preventDefault()** function that you used in INFO2180.

Next, define the method called '**uploadPhoto**' in the **methods** property of your component. This will be responsible for making the AJAX request using the Fetch API (similar to what you used in Lab 6) to your API endpoint "**/api/upload**" that you created in your **views.py** file. Start with the following example for your fetch() function:

```
fetch("/api/upload", {
    method: 'POST'
})
    .then(function (response) {
        return response.json();
    })
    .then(function (jsonResponse) {
        // display a success message
        console.log(jsonResponse);
```

```
    })
    .catch(function (error) {
        console.log(error);
    });
```

Assuming you did everything correctly, try to submit the form without anything in your form fields and you should get a response that lists your form validation errors in the console of your web browser. Take note of what these errors are. Now try actually adding a description and uploading a file. Do you still get any errors? What error do you see?

We haven't actually sent any data along with our AJAX request. Since we are sending a file along with our request we will take advantage of the **FormData** interface that is available to us in JavaScript. The FormData interface provides a way to easily construct a set of key/value pairs representing form fields and their values, which can then be easily sent as part of our AJAX request. It uses the same format a form would use if the encoding type were set to "multipart/form-data". Which is what we need when sending a file. Update your **uploadForm** method in your component to have the following:

```
let uploadForm = document.getElementById('uploadForm');
let form_data = new FormData(uploadForm);

fetch("/api/upload", {
    method: 'POST',
    body: form_data
})
    .then(function (response) {
        return response.json();
```

```
        })
        .then(function (jsonResponse) {
            // display a success message
            console.log(jsonResponse);
        })
        .catch(function (error) {
            console.log(error);
        });
```

Note: You will also need to add an **id** attribute with a value of "**uploadForm**" to your **<form>** tag so that we can specifically reference that form in our FormData interface.

Now fill out your form again and submit the form, what do you see in your browser console? You should still see an error because we haven't supplied a CSRF token.

We could tell Flask-WTF to disable CSRF protection, but being the security conscious web developers that we are, we will leave it enabled. What we need to do is find a way to pass the generated CSRF token from Flask-WTF to our JavaScript code. If you look in your **views.py** file at the **index()** view function you may notice that instead of using **app.send_static_file()** as we did in Lab 6, that we are using **render_template()** again. This will allow us to print Jinja2 variables and thus render the CSRF token.

First, let us update our **__init__.py** file to tell it to allow CSRF Protection globally for our Flask app so it has the following code:

```
from flask_wtf.csrf import CSRFProtect
```

```python
app = Flask(__name__)
csrf = CSRFProtect(app)
```

Next, since the **index.html** Jinja2 template file extends base.html, let us open **base.html** and at the very bottom where we list our JavaScript files, let us add some JavaScript code to assign the token to a variable. Just before the line that loads our **app.js** file, place the following code:

```html
<script>const token = "{{ csrf_token() }}";</script>
```

Now open in your **app.js** file let us update the **fetch()** AJAX request to use this token:

```javascript
fetch("/api/upload", {
    method: 'POST',
    body: form_data,
    headers: {
        'X-CSRFToken': token
    },
    credentials: 'same-origin'
})
    .then(function (response) {
        return response.json();
    })
    .then(function (jsonResponse) {
        // display a success/error message
        console.log(jsonResponse);
    })
    .catch(function (error) {
```

```
        console.error(error);
    });
```

Here we are sending a Header along with our AJAX request. This header is the **X-CSRFToken** header and we have passed the token variable we created earlier. Also we have added a **credentials** property, which tells the browser to send cookies, auth tokens or headers such as our CSRF Token along with the request but only for requests on the same origin (or same website domain). Now try to submit the form again, but this time with an actual file and description. You should now get a successful JSON response and the file should be saved to your uploads folder.

Lastly, ensure that you create a route with the path "**/upload**" for the upload form component in the Vue Router. This is similar to the Home component that was given to you in your starter code. Also ensure that you add a router link to your **app-header** component for the new route you created for the upload form component in VueJS.
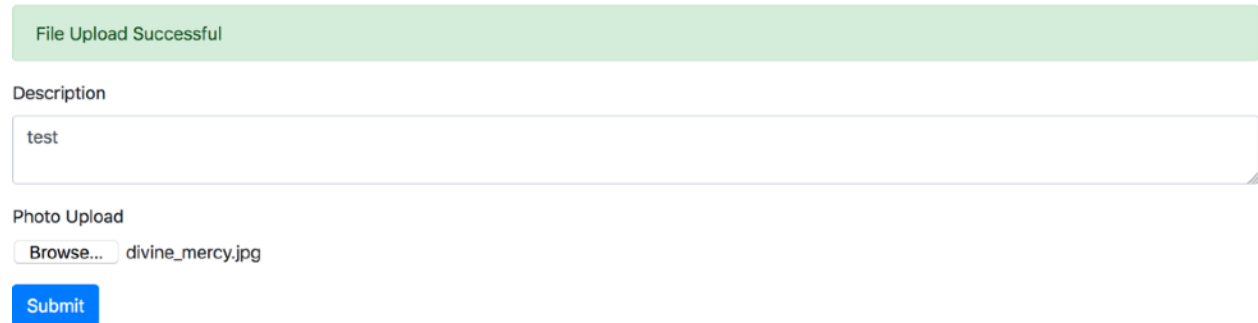
Commit your code to your Github Repository.

Bonus: Now see if you can figure out how to give the user feedback by displaying the success or error message on the same upload form page (instead of in the console) when you have a successful upload or the validation fails. See *Figure 1* and *Figure 2* below.

Hint: *You will need to define some data properties within your component and possibly use the **v-if** and **v-for** VueJS directives in your template to hide/show the message. You will also need to tweak one of the **then()** functions in your **fetch()** AJAX request to determine whether to display the success message or error messages.*

Commit your code to your Github Repository.

Upload Form

> File Upload Successful

Description

> test

Photo Upload

[Browse...] divine_mercy.jpg

[Submit]

FIGURE 1: SUCCESS MESSAGE DISPLAYED ABOVE FORM.

Upload Form

> • Error in the Photo field – This field is required.
> • Error in the Description field – This field is required.

Description

Photo Upload

[Browse...] No file selected.

[Submit]

FIGURE 2: FORM VALIDATION ERRORS DISPLAYED ABOVE FORM.

Submission

Submit your code via the "Lab 7 Submission" link on OurVLE. You should submit the following link:

1. Your Github repository URL for your Flask app e.g. https://github.com/{yourusername}/info3180-lab7

Grading

1. (5 marks) Build the Form using Flask-WTF
   a. (1 mark) UploadForm class created
   b. (2 marks) TextArea field with DataRequired() validator should be defined.
   c. (2 marks) FileField with validator that has FileRequired and FilesAllowed which only allows Images should be defined
2. (5 marks) Flask API Route to process form
   a. (1 mark) Route for /api/upload and view function called 'upload'.
   b. (1 mark) Route should only allow 'POST' method
   c. (1 mark) Route should save file to uploads directory
   d. (1 mark) Return JSON output as shown in lab document using jsonify method.
   e. (1 mark) If validation fails then JSON output should look similar to what is in lab document.
3. (10 marks) Create the front-end to display the form using VueJS
   a. (1 mark) VueJS component called 'upload-form' should be created.
   b. (1 mark) 'template' property in component should have an HTML form
   c. (1 mark) form tag should have @submit.prevent="uploadPhoto" on it.
   d. (1 mark) 'methods' property in component should have a function called uploadPhoto
   e. (1 mark) There should be an AJAX request made using the fetch() method to the route "/api/upload"
   f. (1 mark) __init__.py should have `csrf = CSRFProtect(app)`

g. (1 mark) In base.html template file there should be a
   <script>const token = "{{ csrf_token() }}";</script> just before
   the line that loads the app.js file
h. (1 mark) In the AJAX request, ensure there is a header property
   for X-CSRFToken with the token variable. e.g. 'X-CSRFToken':
   token
i. (1 mark) File successfully uploads.
j. (1 mark) create a route with the path "/upload" for the upload
   form component in the Vue Router.

4. Bonus
   a. (2 mark) If you are able to give user feedback by displaying
      success or error messages for the file upload.