INFO3180 Lab 5 (20 marks)

VueJS and Flask API

Due: April 10, 2023 at 11:55pm

In this lab, you will practice integrating a VueJS frontend with a Flask backend API. You will be required to create a basic application that allows you to add your favorite movies to a database and display them on a page.

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

 VueJS 3 Guide: https://vuejs.org/guide/
 Intro to Vue 3: https://www.vuemastery.com/courses/intro-to-vue-3/intro-to-vue3/
 VueRouter 4: https://router.vuejs.org/guide/

 Note: Remember we will be using the Composition API, so ensure in the VueJS documentation that you select that option. |

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://devtools.vuejs.org/   **Ensure you choose the option for either Google Chrome or Mozilla Firefox.**

Ensure that you download and install NodeJS (if you haven't previously done so) from https://nodejs.org.
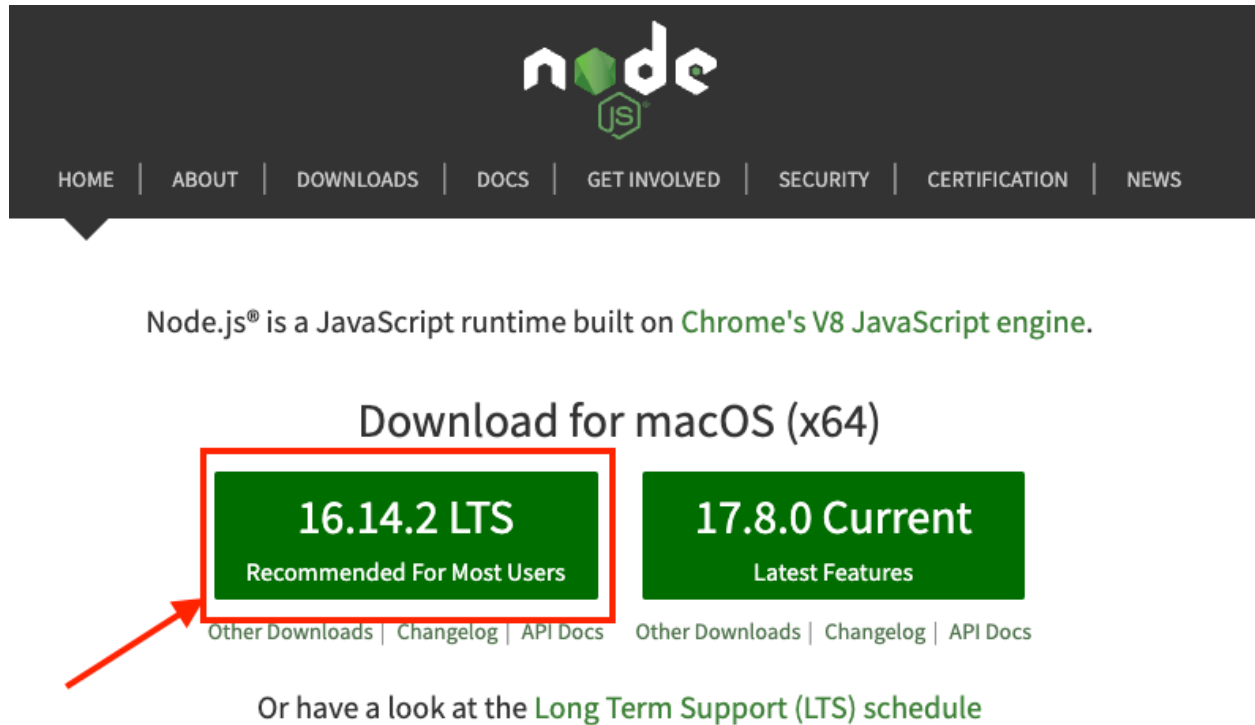
Once NodeJS is installed, you will need to fork and clone the starter code repository and install the dependencies for your VueJS project and start the development server.
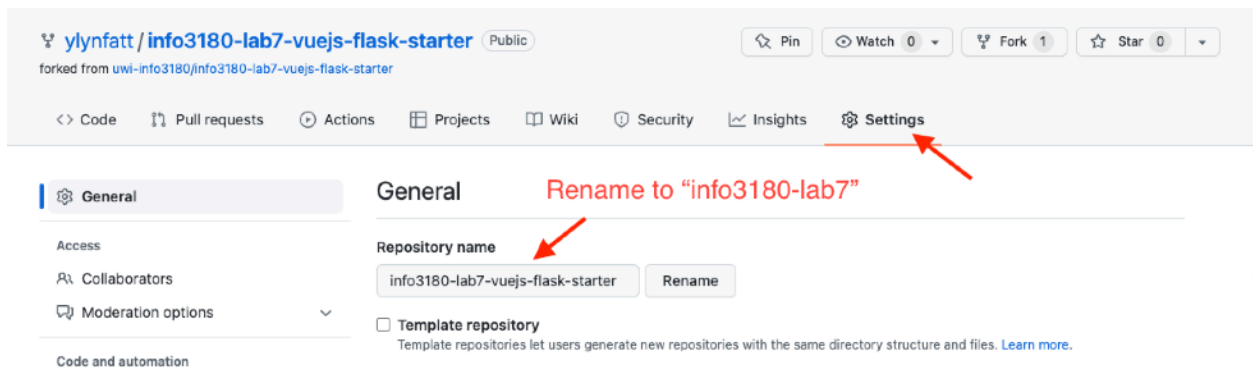
Fork and Clone the Repository

Start with the example at: https://github.com/uwi-info3180/info3180-vuejs-flask-starter.git

Fork the repository to your own account
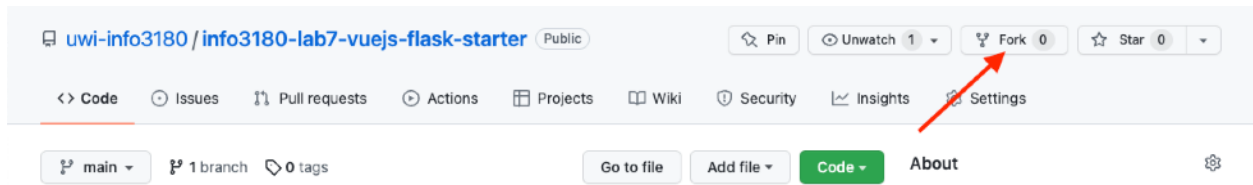
In github.com rename it by going to Settings and changing the name to **info3180-lab5.**

To start working on your code, clone it from your newly forked repository for example:

```
git clone https://github.com/{yourusername}/info3180-lab5
```

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
flask --app app --debug run
```

Now in a separate terminal window, install the dependencies for your VueJS frontend by doing the following:

```
npm install
npm run dev
```

You should now see something like the following in your terminal:

```
vite v2.8.6 dev server running at:

> Local: http://localhost:3000/
> Network: use `--host` to expose

ready in 190ms.
```

Your VueJS dev server should be running at http://localhost:5173 while your Flask dev server will be running on http://localhost:8080 .

Exercise 1 - Build the Form using Flask-WTF (2 marks)

You have been provided with starter code at the link above which has an empty file called **forms.py** . Create a form class called **MovieForm** that has three (3) fields and appropriate validation rules. A String field called '**title**' for the Movie Title, TextArea field called '**description**' that requires a user to fill in a brief description or summary of the movie and a FileField called '**poster**' that only allows images of a movie poster to be uploaded.

Commit your code to your Github Repository.

Exercise 2 - Movies Database table (3 marks)

Next we will create a PostgreSQL database called "**lab5**" and a table called "**movies**" that will store your favorite movies along with a reference to the filename of the uploaded movie poster. You will use the empty `models.py` file that was provided with your starter code and create a Movie class with the following columns in your database:

- id (integer)
- title (string)
- description (text)
- poster (string)
- created_at (date time)

Note: You will need to ensure that you initialize Flask Migrate and Flask SQL Alchemy in your code and that you create a migration file as well. You can refer to your previous labs and lectures on how to do this.

Exercise 3 - Flask API Route to process form (5 marks)

Use the MovieForm class and Movie Model from Exercise 1 and 2 and integrate it with a flask route `/api/v1/movies` and view function called `'movies'`. You should ensure that this route only allows **POST** requests and should validate user input on submit. The route should save a movie to your database and also save the file to an uploads folder and return the movie title, **poster**, **description** and a **message** in a JSON format similar to the example below:

```
{
    "message": "Movie Successfully added",
     "title": "Some Movie Title"
    "poster": "your-uploaded-movie-poster.jpg",
    "description": "Summary of the movie".
}
```

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 4 - Create the front-end to display the form using VueJS (5 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 '**MovieForm**' in a file called '**MovieForm.vue**' in your **src/components** folder. This component should have a '**<template>**' block that has the HTML code for the form. You will have to manually create the form fields.

On the **<form>** tag, you will use the VueJS directive **@submit.prevent="saveMovie"**. This will ensure that when the submit button is clicked or if the user press the ENTER key on their keyboard that the function **saveMovie** (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.

Within your <form></form> tags you will need to create your form label and input tags for each field. For example:

```
<div class="form-group mb-3">
  <label for="title" class="form-label">Movie Title</label>
  <input type="text" name="title" class="form-control" />
</div>
```

Next, in the **<script setup>** block of your component define the method called '**saveMovie**'. This will be responsible for making the AJAX request using the Fetch API to your API endpoint "**/api/v1/movies**" that you created in your **views.py** file. Start with the following example for your fetch() function:

```
fetch("/api/v1/movies", {
    method: 'POST'
})
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        // display a success message
        console.log(data);
    })
    .catch(function (error) {
        console.log(error);
    });
```

We will now need to create a route and a page to display this component in our Frontend and we will also need to make a few changes to our Flask Backend so that VueJS and Flask will work well together.

Create another component called **AddMovieFormView.vue** in your **src/views** folder and import your **MovieForm** component that you previously created and use it in the **<template>** block for **AddMovieFormView.vue**. You also need to ensure you update your **src/router/index.js** file to add the route '**/movies/create**' to the VueRouter.

The last thing we need to do is to let VueJS proxy requests for **/api*** routes to your Flask API. If you don't do this you will receive Cross Origin Resource Sharing (CORS) errors. One way to fix this is to open your

`vite.config.js` file and add the following after the **`resolve{}`** property:

```
resolve: {
…
},
server: {
  proxy: {
    '^/api*': {
      target: 'http://localhost:8080/'
    }
  }
}
```

You will need to stop and restart your VueJS dev server for the changes o take effect.

Now visit your newly created VueJS route, http://localhost:5173/movies/create . 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 **saveMovie** method in your component to have the following:

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

fetch("/api/v1/movies", {
    method: 'POST',
    body: form_data
})
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        // display a success message
        console.log(data);
    })
    .catch(function (error) {
        console.log(error);
    });
```

Note: You will also need to add an **id** attribute with a value of "**movieForm**" 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.

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

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

Next, let us create an api route in your **views.py** file that we can make a call to so that we can generate the CSRF token. That API route should look like the following:

```
@app.route('/api/v1/csrf-token', methods=['GET'])
def get_csrf():
    return jsonify({'csrf_token': generate_csrf()})
```

Also ensure you import the **generate_csrf()** function from **flask_wtf**.

```
from flask_wtf.csrf import generate_csrf
```

Now in your VueJS frontend, open your **MovieForm.vue** file and let us add a reactive property called "**csrf_token**" and add a new method to the called **getCsrfToken()** which will look something like this:

```
import { ref } from "vue";
let csrf_token = ref("");

  function getCsrfToken() {

    fetch('/api/v1/csrf-token')
      .then((response) => response.json())
      .then((data) => {
        console.log(data);
        csrf_token.value = data.csrf_token;
      })
  }
```

And now update your import statement and add a '**onMounted**' lifecycle hook and call the **getCsrfToken()** method. This will ensure that the CSRF Token is fetched and stored as soon as the component is mounted:

```
import { ref, onMounted } from "vue";

onMounted(() => {
    getCsrfToken();
});
```

Then, update your **saveMovie()** method and add a header to your fetch request that has the CSRF Token:

```
function saveMovie() {
    let movieForm =
document.getElementById('movieForm');
```

```
    let form_data = new FormData(uploadForm);

    fetch("/api/v1/movies", {
        method: 'POST',
        body: form_data,
        headers: {
            'X-CSRFToken': csrf_token.value
        }
    })
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        // display a success message
        console.log(data);
    })
    .catch(function (error) {
        console.log(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. Now try to submit the form again, but this time with an actual title, poster and description. You should now get a successful JSON response in your web browser console and the file should be saved to your uploads folder.

Next, ensure that you add a router link to your **AppHeader** component for the new route you created for the add movie form page in VueJS.
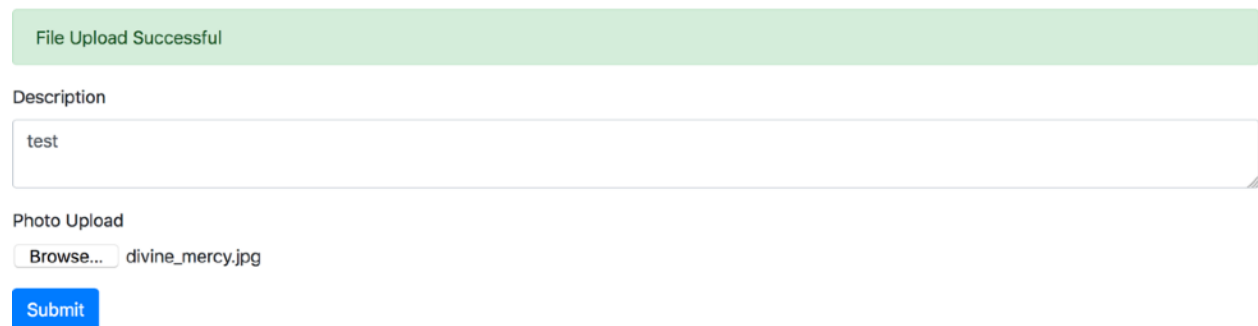
Commit your code and push to your Github Repository.

Bonus: Now see if you can figure out how to give the user feedback by displaying the success or error message on the same movie 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 and push to your Github Repository.

Upload Form

File Upload Successful

Description

test

Photo Upload

Browse...   divine_mercy.jpg

Submit

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.

Exercise 5 - Create an API endpoint and a front-end page to display your movies (5 marks)

Now that we are able to add movies to our database via our VueJS Frontend and our Flask API, next we want to display our movies on a page. But first you will need to create a new API endpoint called "**/api/v1/ movies**" but this time when a **GET** request is made you will query your database and return a list of movies in JSON format that looks similar to the following:

```
{
  "movies": [
    {
      "id": 1,
      "title": "The movie title"
      "description": "The summary for the movie",
      "poster": "/api/v1/posters/movie-poster.jpg",
    }
```

```
    ]
}
```

For the poster image url you will also need to have an endpoint "**/api/v1/posters/<filename>**" that can fetch the poster image from your uploads folder. This should be very similar to what you had done in Lab 4.

Next we will create a new VueJS page called **MoviesView.vue** in your **src/views** folder. In your **<script setup></script>** tags, import your **ref** and **onMounted** functions from **vue** and create a reactive property called "**movies**" which is an empty array.

```
<script setup>
import { ref, onMounted } from "vue";


let movies = ref([]);


</script>
```

Now add a function called **fetchMovies()** in your **<script setup></script>** tags that will make an AJAX request to your Flask API endpoint "**/api/v1/movies**" as a GET request this time. You will run that function in your **onMounted** life cycle hook. And ensure that you update your **movies** reactive property with the data returned from the API call.

Next, you will update your **<template></template>** for your *MoviesView* page component with the necessary HTML and and VueJS directives to loop over the list of movies and display the image, movie title and movie description in a card.

Add a route "/movies" to your vue-router that will load your MoviesView page and add a router link to your AppHeader component for the new route you created for the page to view all the movies in VueJS. Your final page should look like the following:
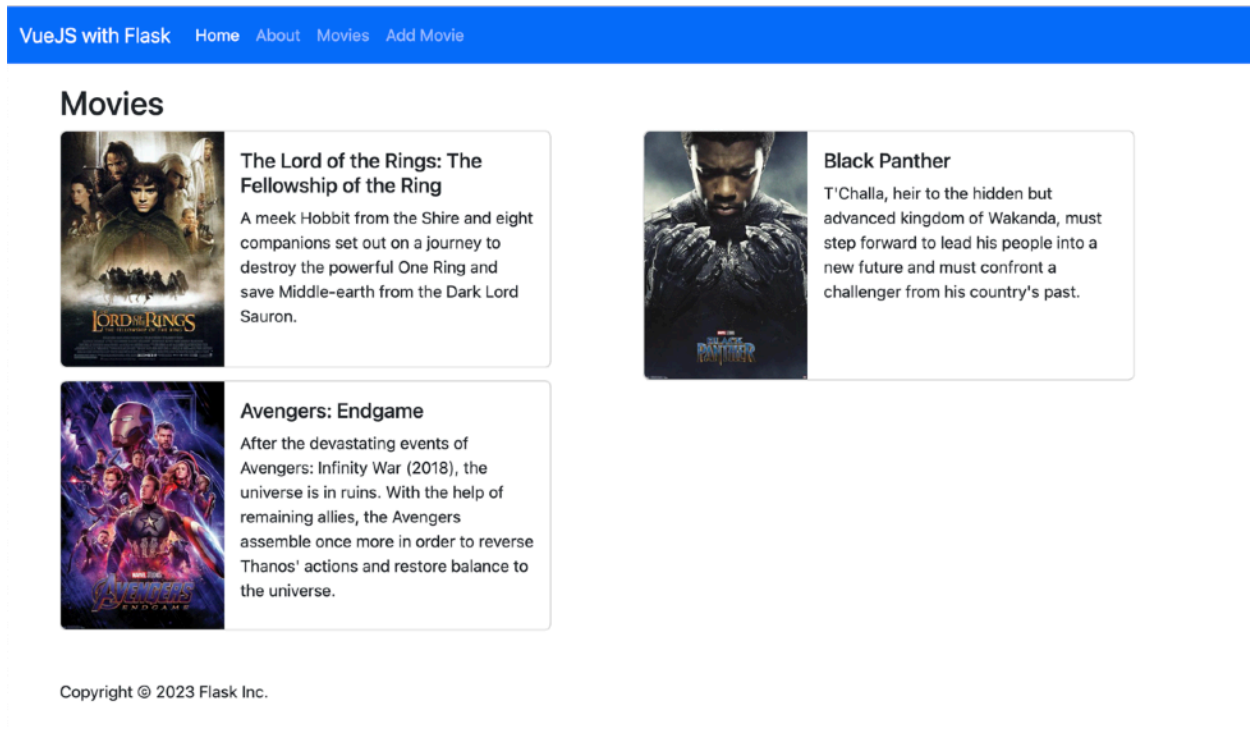
Note: Displaying the image might be a little tricky. Instead of just using the '`src`' attribute by itself on an **`<img />`** tag, try using '`:src`'. This is shorthand for '`v-bind:src`' which is another VueJS directive that allows us to bind a string to the attribute.

If we were to deploy this VueJS/Flask API to a production server there would be a few extra changes to make but we will leave it like this for now.

Commit your code and push to your Github Repository.

Submission

Submit your code via the "Lab 5 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-lab5

Grading

1. (2 marks) Build a Form class using Flask-WTF with the appropriate fields and validators.
2. (3 marks) Create model for your Movies table and create migration file(s). You also need to ensure you properly connect to your database using SQLAlchemy.
3. (5 marks) Create a Flask API Route to process form
   a. (1 mark) Route for /api/v1/movies and view function called 'movies'.
   b. (1 mark) Route should only allow 'POST' method
   c. (1 mark) Route should save data to the database and file to the 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 error messages should look similar to what is in lab document.
4. (5 marks) Create the front-end to display and submit the form using VueJS
   a. (1 mark) VueJS page called "AddMovieFormView" and component called 'MovieForm' should be created.

b. (1 mark) A function called saveMovie() should be created that makes an AJAX request using the fetch method to route */api/v1/movies* as a POST request.

c. (1 mark) __init__.py in your Flask backend should have `csrf = CSRFProtect(app)` and there should be a function called getCsrfToken() created in VueJS that will get the generated CSRF token from your Flask API and this method should be called in the onMounted lifecycle hook.

d. (1 mark) In the AJAX request to save the movie, ensure there is a header property for X-CSRFToken with the token variable. e.g. 'X-CSRFToken': token

e. (1 mark) create a route with the path "/movies/create" in the Vue Router that displays the movie form and the form should successfully submit.

5. (5 marks) Create an API endpoint and a front-end page to display your movies

a. (2 mark) Create another route for /api/v1/movies and view function called 'add_movies' that queries your database and returns the list of movies from your database in JSON format.

b. (1 mark) A VueJS page called "MoviesView" should be created.

c. (1 mark) A function called fetchMovies should be created that makes an AJAX request using the fetch method to route */api/v1/movies* as a GET request in the onMounted lifecycle hook.

d. (1 marks) create a route with the path "/movies" in the Vue Router that displays the list of movies as cards.

6. Bonus

a. (2 marks) If you are able to give user feedback by displaying success or error messages when attempting to save the movie.