Unit 6 - Django and Vue.js

Classwork: The classwork for this unit should be saved in a new folder: **your_repo/unit_6/**
Homework: Homework should be saved here: **your_repo/homework/unit_6/**

*** Day 01 ***

Django and Vue.js

We learn how to develop a backend application with Django. We learn how to do the frontend like a pro with Vue.js 🙂

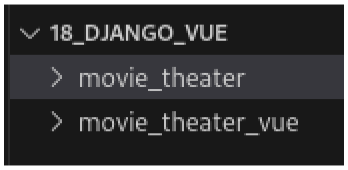
There are more features on the front-end that you can still learn, but you have learned enough to make a beautiful front-end already. If you want, you can learn more on your own, like using a router to have multiple pages in one vue application, using Pinia for state management, …

Well, now let's make them work together. And then you can tell that you are a full-stack developer!!!

Let's see how to make them work together. You know almost everything, we just need to link them together, which is just a matter of configuration (which can be a pain with JS because for me, nodejs is a mess) and then a matter of requests between them.

Configure Django and Vue.js

1. Copy the last django application version you have (authentication lesson). Remember, you need to have communication with the database. If you need to set up part, please set it up.
2. Create a new python env for this lesson. Make sure to have **nodeenv in the dev requirements** (now, we are going to need it as you are expert on front-end).
3. Include a new package on your **main requirements: django_vite**
4. Create a new env for nodejs.
5. Let's create a new Vue.js project. So what I like to do is to have a Django folder, and at the same level, the Vue.js folder.



```
18_DJANGO_VUE
  > movie_theater
  > movie_theater_vue
```

6. Check that your Vue.js app is working.

Insert Vue.js pages into Django templates

1. Let's create a folder apps in the src folder in the Vue.js folder. In this apps folder, let's create a movie_edit folder.
2. Let's create a js file movie_edit.js inside the movie_edit folder, that is going to have the Vue.js app and mount it in the html (but we still do not know which html, but that does not matter for now, it is going to be mounted somewhere).

```
import 'vite/modulepreload-polyfill';

import { createApp } from 'vue';
import App from './MovieEdit.vue'


createApp(App).mount("#app")
```

3. Let's create the file MovieEdit.vue in the same folder just to have something in the template.

```
<template>
     <div>
     This is the page where we are going to edit movie in
Vue.js
     </div>
</template>

<script>

export default {
  name: 'App',
  components: {
  },
  data: function() {
     return {

     }},
     methods: {

     },
}
</script>
```

4. Let's rewrite the vite.config.js. This file will define the multiple Vue.js app that we can write, it will also define where we want to build our .js file for production.

```
import { defineConfig } from "vite";
import vue from "@vitejs/plugin-vue";

const backendPath = '../movie_theater';

// https://vitejs.dev/config/
export default defineConfig({
 plugins: [vue()],
 base: '/static/vite/',
 server: {
   watch: {
     ignored: [],
   },
 },
 build: {
   manifest: true,
   emptyOutDir: true,
   outDir: backendPath + '/core/static/vite/',
   rollupOptions: {
     input: {
       vue_movie_edit: "./src/apps/movie_edit/movie_edit.js",
     },
   },
 },
});
```

5. What we are going to use right now, it is the rollupOptions, input: (https://vitejs.dev/config/build-options#build-rollupoptions, https://rollupjs.org/configuration-options/#input, https://github.com/MrBin99/django-vite)

   Let's add **'django_vite'** to the **INSTALLED_APPS** list in Django app setting.py

   Also, add the following lines at the end of settings.py:

```
DJANGO_VITE_ASSETS_PATH = os.path.join(BASE_DIR, "core", "static", "vite")
DJANGO_VITE_DEV_SERVER_PORT = get_secret("vite_dev_server_port")
DJANGO_VITE_STATIC_URL_PREFIX = "vite/"
DJANGO_VITE_DEV_MODE = True # This line has to be removed in production
```

   In the #Static file section in the same settings.py file add STATIC_ROOT. Otherwise, django_vite is going to complain. This is a parameter only used in production to serve the static files, through a web server like nginx, so that's why you set a directory just for that, and when you ask django to build the static files with the command "python manage.py collectstatic", it will put all the static files in there.

```
STATIC_ROOT = os.path.join(os.path.dirname(BASE_DIR), "movie_theater_static")
```

6. Do not forget to set 'vite_dev_server_port' in the secrets.json to 5173 (could be different in your computer, double check). We add this port here in case we want to have the flexibility later to change the vite server port.

```
 "vite_dev_server_port": "5176"
```

7. Good, let's continue. We are going to work on a Django template now, the one to edit a movie (movie_form.html)

   Right now, we have something like this:

```
{% extends "base.html" %}

{% block content %}
<form method="post">{% csrf_token %}
   {{ form.as_p }}
   <input type="submit" value="Save">
</form>

{% endblock content %}
```

   On top of the template, we are going to add:

   {% load django_vite %}

   And at the end of the template:

```
{% block js %}
   {{ block.super }}

   {% vite_hmr_client %}
   {% vite_asset 'src/apps/movie_edit/movie_edit.js' %}
{% endblock js %}
```

   We should have something like the following code. We are going to keep the original form just in case for now.

```
{% extends "base.html" %}
{% load django_vite %}

{% block content %}
<form method="post">{% csrf_token %}
   {{ form.as_p }}
   <input type="submit" value="Save">
</form>

<br><br>

<div id="app">
   <app></app>
</div>

{% endblock content %}

{% block js %}
  {{ block.super }}

  {% vite_hmr_client %}
  {% vite_asset 'src/apps/movie_edit/movie_edit.js' %}
{% endblock js %}
```

8. At this point, if you refresh the website page on a new movie or update a movie, you should see the part coming from Vue.js.

   If it is working, you can even change something on the MovieEdit.vue and see the change almost immediately on the html page. (this is the magic of the vite server).

If you get it until there, that is great. We did a lot of configuration and now we are able to load a Vue.js app in a Django template. That is great!!!!!

*** Day 02 ***

Pass Django Data into Vue component

First we need to give the csrf token, as we will have to submit information to Django.

Q: Do you remember what a csrf is? When do you need to use it? In a GET request? In a POST request?

So here, as we need to submit a post request with the movie form, the Vue.js application will need to have this csrf token. We can do the following:

In our **movie_form.html**, in the block js, we can insert a script tag, and then just define our **csrf_token** there. We will configure our Vue.js to grab it.

```
{% block js %}
  {{ block.super }}
  <script>
       var ext_csrf_token = '{{ csrf_token }}'
  </script>
  {% vite_hmr_client %}
  {% vite_asset 'src/apps/movie_edit/movie_edit.js' %}
{% endblock js %}
```

Remember, that if you load some JS into your code that is not trusted, people will be able to hack your website, because they will have access to csrf_token from your html page.

In case just check your base.html (in core/templates/) an make sure you have:

```
…
</body>

   {% block js %}
   {% endblock js %}
…
```

Previously, I messed it up by including a <script> before the block js, and the js console was complaining about it. Please move `{% block js %} {% endblock js %}` outside the script tag.

****

Exercise 1:

Search: how you can retrieve this **`ext_csrf_token`** from your Vue.js app.
Web development is always a quest. You have to search for the information you need and try to find the best information quickly.

Hint: There is something you need to add in the MovieEdit.vue

****

Once you are done with Exercise 1, please verify that Vue.js app has access to the csrf token by using the Vue.js plugin (in firefox or chrome), and checking that the csrf_token has a value.

****

Exercise 2:

Now, find a way to display the same form from django, but from your Vue.js application.

Hint: You must modify the movie_form.html and the MovieEdit.vue

****

****

Exercise 3:

At some point, it would be great if we recreate the form field by field in Vue.js. But for the moment, let's define the movie object as a dictionary in Vue, as well as the actor list (for the multiple choice).

Let's think about a way to do it and implement it.

Hint: Three files should be modified:

- movies/views.py => class MovieUpdateView(UpdateView) => Add method:
  def get_context_data(self, **kwargs) => In this method define a dictionary for movies and a list for movie actors.
- Movies_form.html => Pass the movies dictionary and actors list
- MovieEdit.vue => Receive the movies dictionary and actors list

****

*** Day 03 ***

Customize your form in Vue

Let's copy the form we have in Django to vue, so we can customize it.

```
<template>
 <div>
               This is the page where we are going to edit movie in Vue.js, this is cool
               This is the form coming from django, displayed in vue
 </div>
 <div>
               <form method="post" class="form">
                      <input type="hidden" name="csrfmiddlewaretoken" v-bind:value="csrf_token">
                      <p>
                             <label for="id_name">Name:</label>
                              <input type="text" name="name" value="Matrix" maxlength="100"
required="" id="id_name">
                      </p>
                      <p>
                             <label for="id_running_time">Running time:</label>
                             <input type="text" name="running_time" value="02:18:00" required=""
id="id_running_time">
                      </p>
                      <p>
                             <label for="id_actors">Actors:</label>
                             <select name="actors" required="" id="id_actors" multiple="">
                                    <option value="2">Carrie-Anne Moss</option>
                                    <option value="1" selected="">Keanu Reeves</option>
                             </select>
                      </p>
                      <p>
                             <label for="id_director">Director:</label>
                             <input type="text" name="director" value="The Wachowskis"
maxlength="200" required="" id="id_director">
                      </p>
                      <p>
                             <label for="id_release_date">Release date:</label>
                             <input type="text" name="release_date" value="1999-03-30" required=""
id="id_release_date">
                      </p>
                             <button type="submit" class="btn btn-primary">
                                           Submit
                             </button>
                  </form>
  </div>
  <br><br>

</template>
```

Let's use a Vue.js library for the date field: vue datepicker (https://vue3datepicker.com/).

In package.json (vue project), add a dependency: "@vuepic/vue-datepicker": "8.x"

```
"dependencies": {
   "vue": "^3.4.21",
   "@vuepic/vue-datepicker": "8.x"
 },
```

Then, install this new package: **`npm install`**

Now, you should have the new Vue.js package.

****

Exercise 4:

Read the documentation, and insert the component VueDatePicker to select the date, and when you click on submit, it should work.

Hints:

- In the template, you must insert the new component. Just by adding the component, it does not trigger a new entry in the post request, so you must change the text input where we had the date in string before to a hidden input and get the date in the format expected by Django.
- You must initialize the date variable in data (date: null).
- You might need a computed property to convert date to string. The computed method could call a method to do the job.
- You will need to make some changes on the date object, because JS is really a mess concerning the Date object. It is not as easy to manipulate as in Python. There are some libraries to manipulate Date objects better, but for now with 2 lines you can have the result we want. If you need to use Date objects more often on your vue app, it is better to install libraries to format or manipulate the date object way better than the js standard library.

****

****

Exercise 5:

Load the current release date on the date field in the form.

****

****

Exercise 6:

Now that you know how to create a Vue Datepicker for dates, let's do the same for the field "Running time".

You just need to insert an object for the time:

```
<VueDatePicker style="display:inline-block;width:
150px;padding-bottom:10px;padding-left:10px" v-model="time"
:time-picker="true" enable-seconds></VueDatePicker>
```

You have a time-picker property to indicate that the object only needs time (check the documentation).

We are using the enable-seconds property so we can see seconds too. Otherwise, it is only hours and minutes by default.

Make sure you load the current running time on the time field in the form (similar to exercise 5).

****

Multi select feature

What we have on the django form is a basic html multiselect. If you have a long list of actors, it will be painful to see all the names in the list and also check which ones have been selected already. So, let's use a nice multiselect built on vue.

We are going to use this Vue-multiselect: https://vue-multiselect.js.org/

The document is for Vue multiselect version 2, but it is working also for version 3. So, we are going to install the version 3-bet, because the version 3 is for Vue.js3, which we are using.

Add this dependency on **package.json:**

```
"vue-multiselect": "~3.0.0-beta.3",
```

What should you do after you add a dependency?

Run the command npm list, to see if the new packages have been added.

Now that we have experimented with the Vue Datepicker this should be easier. Let's review the Multiselect documentation and make it work.

****

Exercise 7:

Please add a Vue multiselect to display the actors list.

Hints:

- **READ THE DOCUMENTATION**
- You must import and add the multiselect component. Where?
- Add this css at the end of MovieEdit.vue
  ```
  <style src="vue-multiselect/dist/vue-multiselect.css"></style>
  ```
- Check the file I posted with the solution for previous exercises, it already includes the variables you might need: **actor_list** and **actor_list_source.**
- Replace the block of the actors field using the Vue multiselect.
- Remember to keep the select field hidden in the form because this is the element needed in the POST request. Then, add the Vue multiselect component.
  ```
  <select hidden name="actors" required="" id="id_actors" multiple="">
          <option v-for="actor in actor_list" :value="actor.id" selected=""></option>
  </select>
  ```
- In the multiselect component: link the v-model with actor_list, the options available should be linked with actor_list_source
- Add some styles if you would like

****

*** Day 04 ***

We are going to send the form data to Django, but without having an html form.

- Let's duplicate the last form we did (with the parent div).
- Remove the form tag. The submit is not working anymore, try it.

****
Exercise 8:

Send the data from Vue to Django without the html form.

Hints:

- To simulate the post request done by submitting the button in the form, you need to recreate it as a form object in JS, and add the child elements needed like in the html form.
- Create a new method for this feature, you may name this new method `submit_form.In the Submit button you may add:`
  ```
      @click.prevent="submit_form"
  ```
- To create a form element in the JS, you may do this:
  ```
      var form = document.createElement('form');
      form.setAttribute('method', 'post');
  ```
- I'll let you find out how to create some input element in JS and add it to the form object. It is something like this:
  ```
      var html_field = document.createElement('input')
      form.appendChild(html_field)
  ```
- At the end, you want to have your form object in JS, which should be a mirror of what you have with the html form.
- To submit your form, you do it like this:
  ```
      document.body.appendChild(form);
      form.submit()
  ```
- The browser will submit the form as you would have done it with the form tag by clicking on the submit button.

****

*** Day 05 ***

You have learned that using an HTML form tag is the easier option to send data from Vue to Django in a post request. This option is useful when you have a simple form, like the one we are using in our movie theater project. You can create the form from Django first, duplicate the HTML, and replicate it in Vue if you need to modify it with more intractable fields. This is a simple solution that helps us achieve a nice result.

You have also learned how to build the form element using JS, which was not as simple as using an html form. You probably are wondering why we learned that? It is because for complex forms, or when you need more flexibility on the forms (select something that will make another input field appear, for example), creating a JS form where you have more control of the fields is useful in this case, even if it is tedious to create a JS form.

There is one more way to send a POST request from Vue to Django when you have a complex form. **On the Vue.js side**, we are going to create a form object but this time we will not build it like an html element. Just a form object, we will put our data there, and **send it with a fetch api.** Do you remember the fetch api?

On the django side, we are going to **process the data and then return a json message**. Vue will receive the answer and will execute what you indicate in the code: if it is successful we can redirect the user to the view page, or stay in the same page if you want it.

It is like an api call from vue, but it is not exactly an api call: We are still using our cookie session (something we do not do with an api), we are still using our crsf for the POST request. That is it is not 100% api, but it is close enough, and we are going to limit the requests from a web browser.

If you want to use something 100% api in your project (that means that you will be able to use it from code directly), you can use the Django rest framework.

Let's implement this last option in our movie theater project:

Create a new update view in Django (we are going to keep testing with update movie):

```
class MovieUpdatebisView(View)
```

Do not forget to import View:

```
from django.views import View
```

We use a view class because we are going to return a json. No template this time.

We are going to use the MovieForm we already built from the model Movie.

```
class MovieUpdatebisView(View):
   def post(self, request, *args, **kwargs):
       form = MovieForm(request.POST)
```

We can set up a breakpoint in your code by importing pdb: **import pdb;pdb.set_trace()** to see what is going on while we are developing. When you are at the breakpoint your can check the variable values, press the letter **c** to continue with the execution (in the terminal where Django server is running):

```
class MovieUpdatebisView(View):
   def post(self, request, *args, **kwargs):
       form = MovieForm(request.POST)
       import pdb;pdb.set_trace()
```

Add the view to the urls.py.:

```
   path(
     "movies/update_bis/<int:pk>",
     views.MovieUpdatebisView.as_view(),
     name="movie_update_bis",
     ),
```

Let's add a JS variable to have the url available from vue in the django template:

```
var ext_update_bis_url = '{% url 'movies:movie_update_bis' object.id %}'
```

In MovieEdit.vue get this url and store it in a variable in the data section.

Let's create a new div that will contain the data fields, and copy the code you have for the previous option (JS form). We are going to modify the submit button, so we need a new method, please update that in the submit button:

```
   <button type="submit" class="btn btn-primary"
           @click.prevent="submit_form_fetch"
           :disabled="submitting_form">
           Submit
    </button>
```

In the submit method, we are are going to use the form object:

```
let formData = new FormData();
```

You can add data to the formData like this:

```
let form_data = {
    'name': this.movie_dico.name,
    'running_time': this.get_time_string,
     …
     …
}
for (var key in form_data) {
    formData.append(key, form_data[key])
}
```

When you are going to send the request use the fetch api this way:

```
fetch(this.update_bis_url,
                {
                     method: "post",
                     body: formData,
                     headers: {'X-CSRFToken': this.csrf_token},
                     credentials: 'same-origin'
                }
           )
```

****

Exercise 9:

Please follow the steps from this lesson to use fetch api to send the POST request. In addition to what was explained in this lesson you will need to make some changes in the view MovieUpdatebisView to make your POST request work.

```
class MovieUpdatebisView(View):
     def post(self, request, *args, **kwargs):
          print("request", request)
          movie = get_object_or_404(Movie, pk=self.kwargs["pk"])
          form = MovieForm(request.POST)
          import pdb
          pdb.set_trace()
```

This line **`form = MovieForm(request.POST)`** is not going to work. Something is missing. Figure out what is missing to make it work (if you see the error, it says that the name of this movie already exists, so the form is in mode new object, not update, so check how to make it work for updating an object).

Good luck! Let's see how much you can make it work to update the movie object and return a json saying that the update was successful, or if there was an error updating the movie.

****

*** Day 06 ***

You know how to implement a post request with Vue.js. Now, let's learn how to implement a GET request to ask for information from vue to django.

Let's create a new folder and new files in your Vue.js project:

```
movie_show/movie_show.js
           MovieShow.vue
```

Code: movie_show.js

```
import 'vite/modulepreload-polyfill';

import { createApp } from 'vue';
import App from './MovieShow.vue'

createApp(App).mount("#app")
```

Let's just initialize the `MovieShow.vue` file:

```
<template>
        Hi, is this working?
</template>

<script>
        export default {
                name: 'App',
                components: {
                },
                data: function() {
                        return {
                        }
                },
                methods: {
                },
                computed: {
                }
        }
</script>
```

On Django side, let's create a templateview:

```
class MovieDetailbisView(TemplateView):
   template_name = "movies/movie_detailbis.html"
```

Add the url:

```
path(
       "movies/bis/<int:pk>",
       views.MovieDetailbisView.as_view(),
       name="movie_detail_bis",
   ),
```

And this is the template movie_detailbis.html:

```
{% extends "base.html" %}
{% load django_vite %}

{% block content %}
     <div id="app">
        <app></app>
     </div>
{% endblock content %}

{% block js %}
     {{ block.super }}
     {% vite_asset 'src/apps/movie_show/movie_show.js' %}
{% endblock js %}
```

Modify the list template (movie_list.html) to add the new link for the Vue.js detail link (with a different text) right next to the Django movie detail link.

Click on the new link, and we should see your Vue.js template on your webpage:

Movie Theater App
Hi, Is this working?

Easy!!! Now, let's just verify that the id requested exists in the database. Also, add the id in the context to give it to vue.

```
class MovieDetailbisView(TemplateView):
   template_name = "movies/movie_detailbis.html"

   def get(self, request, *args, **kwargs):
       movie = get_object_or_404(Movie, pk=self.kwargs["pk"])
       return super().get(request, *args, **kwargs)

   def get_context_data(self, **kwargs):
       context = super().get_context_data(**kwargs)
       context['movie_id'] = self.kwargs["pk"]
       return context
```

And let's add it on the template:

```
{% block js %}
   {{ block.super }}
   <script>
       var ext_movie_id = {{ movie_id }}
   </script>
   {% vite_asset 'src/apps/movie_show/movie_show.js' %}
{% endblock js %}
```

Now, let's create the view that will be used by Vue.js to ask Django about the movie information.

```
class MovieDetailJsView(View):
   def get(self, request, *args, **kwargs):
       movie = get_object_or_404(Movie, pk=self.kwargs["pk"])
       movie_js = model_to_dict(movie)
       movie_js["actors"] = []
       for actor in movie.actors.values():
           movie_js["actors"].append(actor)
       return JsonResponse({"movie": movie_js})
```

Create the url for this view:

```
path(
       "movies/js/<int:pk>",
       views.MovieDetailJsView.as_view(),
       name="movie_detail_js",
   ),
```

Update the template:

```
{% block js %}
   {{ block.super }}
   <script>
       var ext_movie_id = {{ movie_id }}
       var ext_movie_detail_js_url = '{% url 'movies:movie_detail_js' movie_id %}'
   </script>
   {% vite_asset 'src/apps/movie_show/movie_show.js' %}
{% endblock js %}
```

See the code to make the get request and display the information (MovieShow.vue).

This is an example of how to implement the GET request and then do anything you would like with the data in Vue.js.

So, now you know how to:

- Implement a template using pure django.

- Add some functionalities in Vue.js.

- Implement the entire front-end in Vue if you would like to do that. For this one, you know how to get initial information from django to vue (via the script tag) and then make all the requests you need from Vue.js: at the beginning with a beforeMount (the component has finished setting up its reactive state, but no DOM nodes have been created yet) for example, or later when the user performs some actions. Depending on what you want to do, what you want to see, how important is each page and how quick you need to have some pages ready to show to someone a prototype, you will decide to use only django, or only vue for the front-end or a combination of the two.
  Check Vue.js lifecycle: https://Vue.js.org/guide/essentials/lifecycle

- Implement forms in Django and Vue.js. Forms in Django are easy and quick to implement and show what are the errors when the form entries are incorrect, but can be a little 'ugly', while on Vue.js you need a lot of more work is required (especially to show exactly what is the error, something we did not do it in class because of time) but can be really beautiful if you have time to work on it or if you have to because it absolutely necessary to attract the user.

It is time for a mini project. Django on the server side (back-end), and Vue.js for the frontend. You can pass some init data from Django to Vue.js, but all the front-end needs to be in Vue.js. You can do a SPA (single page application) if you know some advanced Vue.js feature that we did not learn in class (particularly to navigate between pages), or you can do one Vue.js app for each page and have the menu done in django with a template.
Do not work on something too complicated, leave the complex applications for the end of year project when you will have more time to develop your application.