

Explicación práctica 08/11

Creación de la app VueJS

Carpeta *portal* a la misma altura que *admin* porque es una aplicación totalmente diferente. Podría estar en un repositorio diferente.

En la terminal dentro de *portal*

```
#Elegir esta versión porque es la que se tiene en el servidor
asdf local nodejs 22.7.0
npm create vue@latest
```

Es interactivo, así que creamos

Si colocamos `.` se usa la carpeta en la que está parado sino crea otro.

Nombre del paquete: portal-web

TypeScript > NO

JSX support > NO

Vue Router SPA development > YES #librería auxiliar

Pinia para manejos de estado > YES

Vitest for Unit Testing > NO

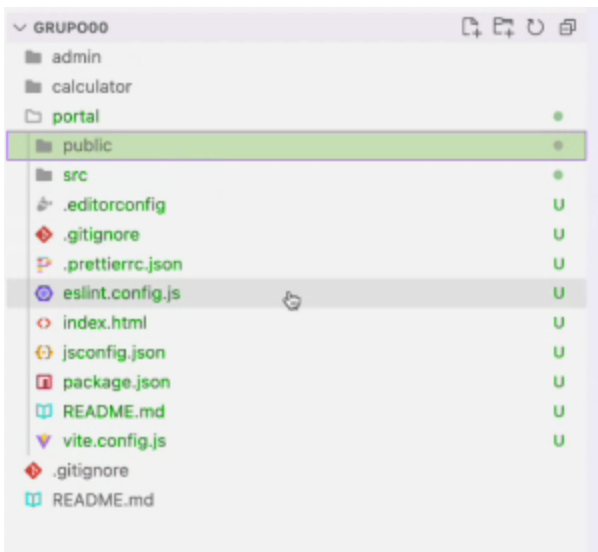
END TO END > NO

ESLint for Code quality > YES

Prettier for code formatting > YES

... Se crea la app Vue Js basic

Se nos ofrecen los comandos



En la carpeta *public* tendremos los assets: *src* donde estará el proyecto; el resto son archivos de configuración.

index.html es el HTML principal de la aplicación. Es donde se encuentra *main.js* . único contenedor de todo.

En el *package.json* tenemos la parte de scripts. Ahí tenemos los comandos que se ejecutan con `npm run <comando>` :

```

{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "lint": "eslint . --fix",
    "format": "prettier --write src/"
  },
}

```

El comando `npm run build` es el más importante, porque es el que compila la aplicación y dejarla lista para producción. Se tiene que ejecutar todo con `npm run build` . Los archivos que se generan con el comando se suben a producción. Dejan el pipeline listo.

El comando `npm run preview` permite ejecutar la aplicación que se generó con el build, es como hacerlo como en **modo produccion**

El comando `lint` y `format` ejecutan `eslint` y `format` correspondientemente.

En el *src* se tienen diferentes archivos.



`main.js` es el archivo que crea la aplicación. Instancia, configura y ejecuta. Con `app.mount(#app)` se carga en el `<div>` del `html index.html` que es donde se cargará el contenido dinámico.

Se carga el `html index.html` llama al contenido `main.js` que monta la aplicación en el tag con el id correspondiente.

`App.vue` posee un template con scripts, template, navbar y style.

Sobre estructura de carpeta

`assets` : Imágenes, css, relacionado con archivos estáticos.

`components` : todos los componentes de Vue.

`router` : indica como maneja las rutas de la aplicación. Indica la ruta y qué contenido debe mostrar.

Hay dos formas diferentes de cargar una ruta. 1. Importar el contenido arriba y lo asocia a la ruta o 2. un `Import lazy loaded`, cuando se visita la página recién lo carga.

```
1  import { createRouter, createWebHistory } from 'vue-router'
2  import HomeView from '../views/HomeView.vue'
3
4  const router = createRouter({
5    history: createWebHistory(import.meta.env.BASE_URL),
6    routes: [
7      {
8        path: '/',
9        name: 'home',
10       component: HomeView,
11     },
12     {
13       path: '/about',
14       name: 'about',
15       // route level code-splitting
16       // this generates a separate chunk (About.[hash].js) for this route
17       // which is lazy-loaded when the route is visited.
18       component: () => import('../views/AboutView.vue'),
19     },
20   ],
21 })
22
23 export default router
24
```

`stores` : directorio donde se crean los diferentes stores de Pinia. Es para el manejo de estados. Lo que se busca es separar o agregar una capa entre el componente y el estado del componente y la información del mismo. Manipula la data que el componente termina mostrando.

```
portal > src > stores > counter.js > useCounterStore > defineStore('counter') callback
1 import { ref, computed } from 'vue'
2 import { defineStore } from 'pinia'
3
4 export const useCounterStore = defineStore('counter', () => {
5   const count = ref(0)
6   const doubleCount = computed(() => count.value * 2)
7   function increment() {
8     count.value++
9   }
10
11   return { count, doubleCount, increment }
12 })
13
```

views dentro están los componentes. Están los templates básicos que renderizan lo básico. No tiene lógica.

```
▼ HomeView.vue U x
portal > src > views > HomeView.vue > ...
1 <script setup>
2 import TheWelcome from '../components/TheWelcome.vue'
3 </script>
4
5 <template>
6   <main>
7     <TheWelcome />
8   </main>
9 </template>
10
```

Iniciar aplicacion (min 47)

```
asdf local nodejs 22.7.0 #Global, no hay un entorno como en py
npm install #Se genera el node_modules
npm run format #Ejecuta prettier
npm run dev #Corre la aplicación en localhost
```

Todo lo que sucede al clicar sobre elementos que poseen rutas no hace ningún request al backend; simula serlo (min 50)

Explicación del pipeline > min 56:10

El `mc mirror` es para minio, los archivos estáticos se mueven a minio al bucket correspondiente e indica la url del sitio.

Ejemplo (min 60:30)

- Crear componente de una tablita de issues.

¿Cómo obtener la información?

En `stores` conseguimos la información. Creamos un `issues.js`

```
import {defineStore} from 'pinia'

export const useIssuesStore = defineStore('issues', {
  //Definimos un json
  state: () => ({ //Se pueden almacenar variables auxiliares para la lógica de la app
    issues: [] //Lista de issues. Variable que aparecerá en el componente
    loading: false, //Variable que indica si está cargando o no
    error: null
  }),
  actions: {
    async fetchIssues(){ //Acción que hará obtener la información q se quiere guardar en el
      try{
        //Inicializamos algunas variables auxiliares
        this.loading = true
        this.error = null
        //Acá se haría el llamado a la API El prof lo harcodea (min 65)
        this.issues = [...]
      } catch(error){
        this.error = 'An error occurred' //Si pasa algo malo en la consulta a la API
        console.error(error)
      } finally{ //Se ejecuta siempre
        this.loading = false //Que finalizó la consulta a la api
      }
    }
  }
})
```

Dentro de `components` creamos un file `IssuesList.vue` :

```
issues.js U IssuesList.vue U X
portal > src > components > IssuesList.vue > {} template > div
1 <template>
2   <div>
3     <h2>Lista de Issues</h2>
4     <p v-if="loading">Cargando...</p>
5     <p v-if="error">{{ error }}</p>
6
7     <table v-if="!loading && issues.length">
8       <thead>
9         <tr>
10          <th>#</th>
11          <th>Título</th>
12          <th>Descripción</th>
13          <th>Usuario</th>
14        </tr>
15      </thead>
16      <tbody>
17        <tr v-for="issue in issues" :key="issue.id">
18          <td>{{ issue.id }}</td>
19          <td>{{ issue.title }}</td>
20          <td>{{ issue.description }}</td>
21          <td>{{ issue.user.email }}</td>
22        </tr>
23      </tbody>
24    </table>
25    <p v-if="!loading && !issues.length">No hay issues para mostrar.</p>
26  </div>
27 </template>
28
29 <script setup>
30   import { useIssuesStore } from '../stores/issues';
31   import { storeToRefs } from 'pinia';
```

v-if directivas de VueJS, si la variable es true muestra el texto (línea 4).

Repasar directivas mostradas en la teoría.

Aca Pinia hace la sincronización con lo definido previamente.

```

issues.js U  IssuesList.vue U X
portal > src > components > IssuesList.vue > {} script setup
1  <template>
2    <div>
25      <p v-if="!loading && !issues.length">No hay issues para mostrar.
26    </div>
27  </template>
28
29  <script setup>
30    import { useIssuesStore } from '../stores/issues';
31    import { storeToRefs } from 'pinia';
32    import { onMounted } from 'vue';
33
34    const store = useIssuesStore();
35    const { issues, loading, error } = storeToRefs(store);
36
37    const fetchIssues = async () => {
38      await store.fetchIssues();
39    };
40    onMounted(() => {
41      if (!issues.value.length) {
42        fetchIssues();
43      }
44    });
45  </script>
46
47  <style scoped>

```

Se ejecuta la función definida en el store. Es un pasamos de información.

```

const store = useIssuesStore();
const { issues, loading, error } = storeToRefs(store);

const fetchIssues = async () => {
  await store.fetchIssues();
};
onMounted(() => {
  if (!issues.value.length) {
    fetchIssues();
  }
});
</script>

```

Ahora es necesario crear una ruta para mostrar el componente. (min 74:50)

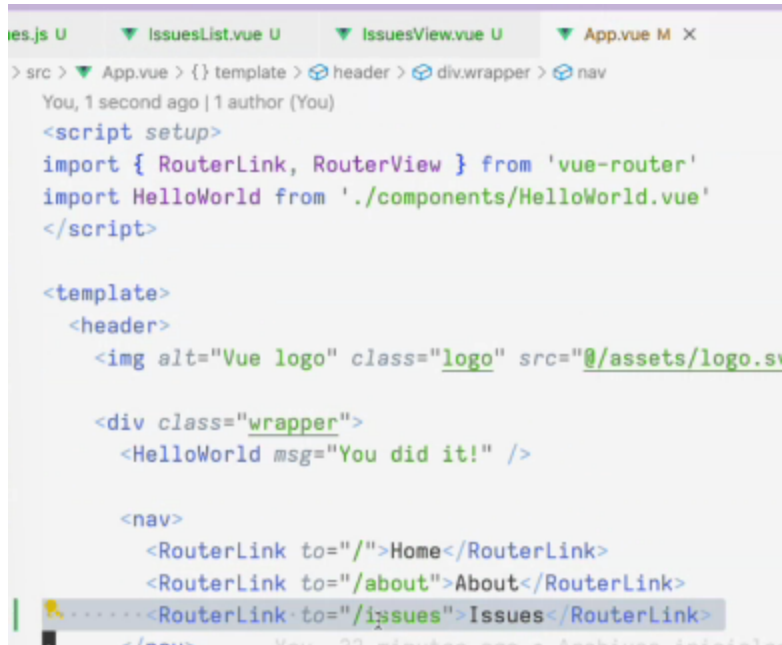
En views creamos la vista e importamos el componente:

```

issues.js U  IssuesList.vue U  counter.js  IssuesView.vue U X
portal > src > views > IssuesView.vue > ...
1  <script setup>
2    import IssuesList from '../components/IssuesList.vue'
3  </script>
4
5  <template>
6    <main>
7      <IssuesList />
8    </main>
9  </template>

```

Generamos el enlace



```
ies.js U IssuesList.vue U IssuesView.vue U App.vue M X
> src > App.vue > {} template > header > div.wrapper > nav
You, 1 second ago | 1 author (You)
<script setup>
import { RouterLink, RouterView } from 'vue-router'
import HelloWorld from './components/HelloWorld.vue'
</script>

<template>
  <header>
    
      <HelloWorld msg="You did it!" />

      <nav>
        <RouterLink to="/">Home</RouterLink>
        <RouterLink to="/about">About</RouterLink>
        <RouterLink to="/issues">Issues</RouterLink>
```

en index.js , en el router le cargamos la vista asociada a la ruta

En el minuto 77:40 explica el camino inverso, qué es lo que genera una acción en la vista.

min 82:10

Usaremos **Axios**. Librería que actuará como cliente.

Se instala `npm install axios` y realizamos un `import {axios} from 'axios'` (se puede usar el fetch nativo).


```
Issues.js M x IssuesList.vue IssuesView.vue
portal > src > stores > Issues.js > useIssuesStore > actions > fetchIssues
You, 1 second ago | 1 author (You)
1 import { defineStore } from 'pinia'
2 import axios from 'axios'
3
4 export const useIssuesStore = defineStore('issues', {
5   state: () => ({
6     issues: [],
7     loading: false,
8     error: null,
9   }),
10  actions: {
11    async fetchIssues() {
12      try {
13        this.loading = true
14        this.error = null
15        const response = await axios.get('http://localhost:5000/api/issues')
16        this.issues = response.data
17      } catch {
18        this.error = 'Error al obtener los issues'
19      } finally {
20        this.loading = false
21      }
22    },
23  },
24 })
25
```

Hay que evitar hardcodear la declaración del base_url -> se podrían usar variables de entorno.

(min 90:40) Explicación CORS: restringe las consultas entre servidores que tienen dominios diferentes. Como front corre en 5173 y el back en 5000, como es un método de seguridad no lo permite. Solución: configurar el back.

```
#En la app py
poetry add flask-cors@latest
```

Investigar CORS para producción y development

```
from flask_cors import CORS

def create_app(...):
    app = Flask(...)
    ...
    CORS(app)

    return app
```

Documentación CORS

Resource specific CORS

Alternatively, you can specify CORS options on a resource and origin level of granularity by passing a dictionary as the resources option, mapping paths to a set of options. See the full list of options in the [documentation](#).

```
app = Flask(__name__)
cors = CORS(app, resources={r"/api/*": {"origins": "http://localhost:3000"}})

@app.route("/api/v1/users")
def list_users():
    return "user example"
```

min (105.40) explicación para subir a prod la aplicación de Vue.