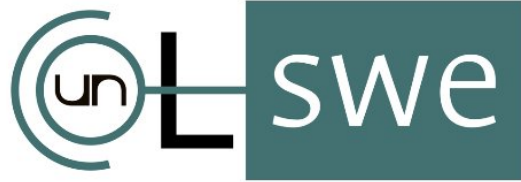


UNIVERSIDAD
NACIONAL
DE COLOMBIA



Laboratorio 2

Frameworks

Ingeniería de Software II.

El objetivo de este laboratorio es desarrollar tareas en una aplicación web sencilla, que permita evidenciar los conceptos de Back-end/Front-end y la comunicación entre ambos.

Prerrequisitos

1. [PostgreSQL 9.5 o mayor](#)
2. [Java JDK 8](#)
3. [Maven](#)
4. [Postman](#)
5. Npm y Node 8.11.0 o mayor
6. [Vue JS](#)

Ejercicio

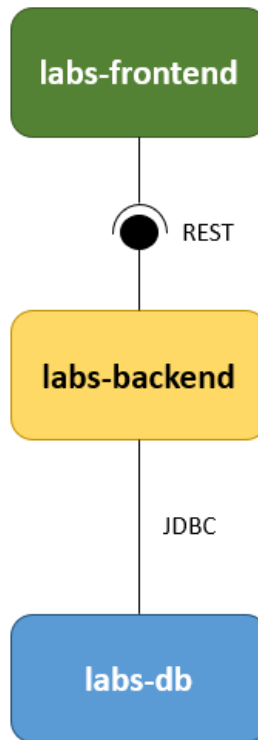
1. Clone el [repositorio](#) del backend del proyecto.
2. Clone el [repositorio](#) del frontend del proyecto.
3. Lea atentamente la introducción y explicación de los Frameworks (siguiente sección).
4. Implemente los siguientes requisitos:
 - a. En la vista de Login del proyecto de frontend, al iniciar sesión de forma exitosa aparece un alert que confirma cuándo se ha almacenado el token de acceso en el localStorage. Quite el alert y haga que al confirmar el inicio de sesión redirija a la página Home (a través del path “/principal”)
 - b. Desarrolle un servicio en Spring que retorne los cursos a los cuales está inscrito un usuario autenticado del sistema. Especificar si está asociado como profesor o

- como estudiante. Recuerde que se debe identificar el usuario por el token de sesión. (Todos los usuarios tienen contraseña del 1 al 5 - 12345).
- c. Desarrolle una vista en donde pueda visualizar los resultados proveídos por el punto b.
 - d. Desarrolle un servicio en Spring que retorne como respuesta la lista de roles que posee el usuario autenticado en el sistema. Debe identificarlo por el token de acceso.
 - e. La página principal (Home) renderiza el título general de la página (en el path “/principal”). También incluye un componente de forma jerárquica que permite realizar la adición de un rol nuevo para el usuario (en el path “/principal/nuevo-rol”). Adicione un nuevo componente como hijo de la vista Home que presente el resultado del servicio desarrollado en el punto anterior (**en el path “/principal/roles”**).
 - f. Desarrolle un servicio en backend que le permita a un docente crear un curso a través del path “/profesor/crear-curso”.
 - g. Adicione un componente hijo a la vista Home que muestre a los usuarios con rol de profesor, un formulario de creación del curso mediante el cual se consuma el servicio desarrollado en el punto f.
5. Grabe un video corto (**de máximo 10 minutos**) donde deberá mostrar su solución a cada punto funcionando correctamente.
 6. Suba el video a una plataforma que permita la reproducción (como YouTube or Google Drive) y compartirlo.

Introducción a Spring Boot y Vue JS

[Spring boot](#) es un framework que permite la construcción de aplicaciones web stand-alone en Java. Posee un servidor Tomcat embebido, y mediante [anotaciones](#) ahorra tiempo de configuración de [Beans](#) en la aplicación especialmente de los archivos XML.

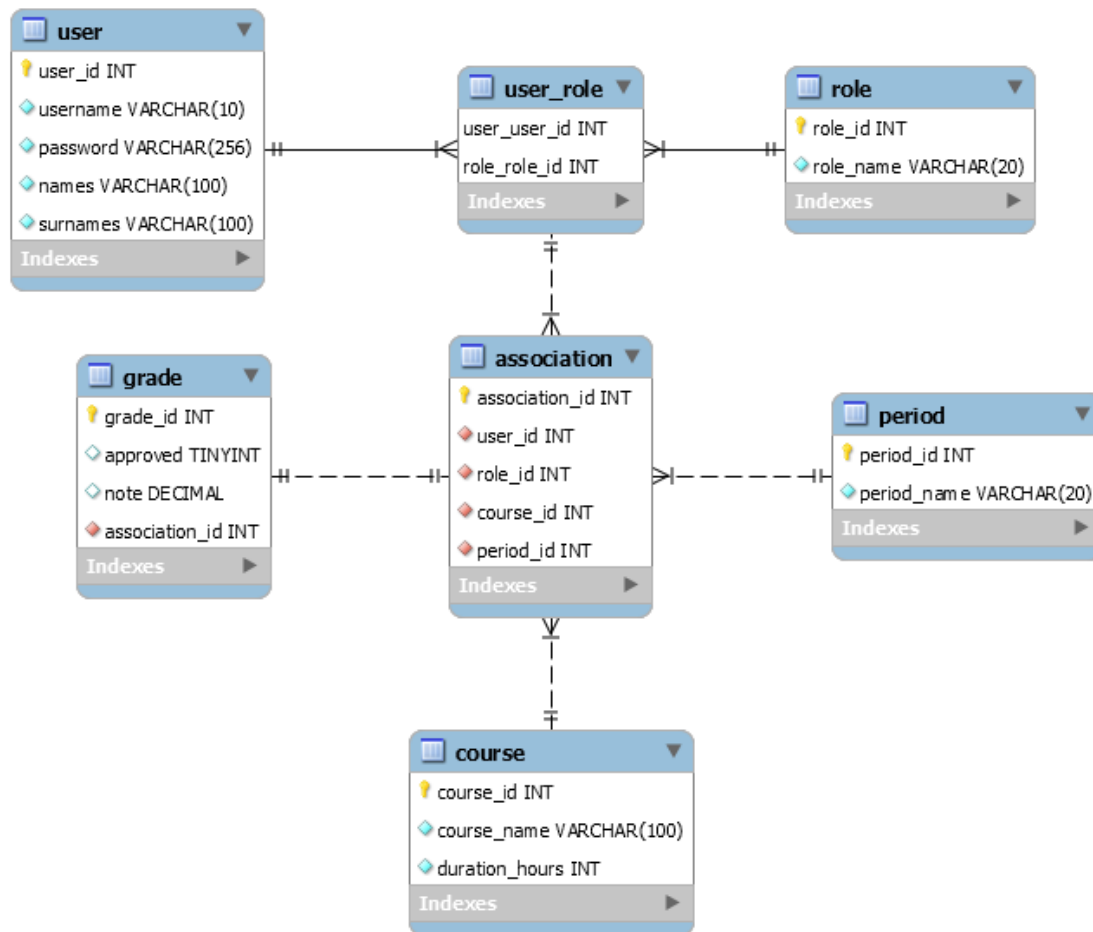
La aplicación web tendrá la siguiente arquitectura:



Por lo tanto tendremos 3 procesos en ejecución: PostgreSQL, Tomcat (Spring Boot 2), y npm serve (Vue).

Para comenzar, luego de la instalación de PostgreSQL, crear una base de datos llamada “labs”.

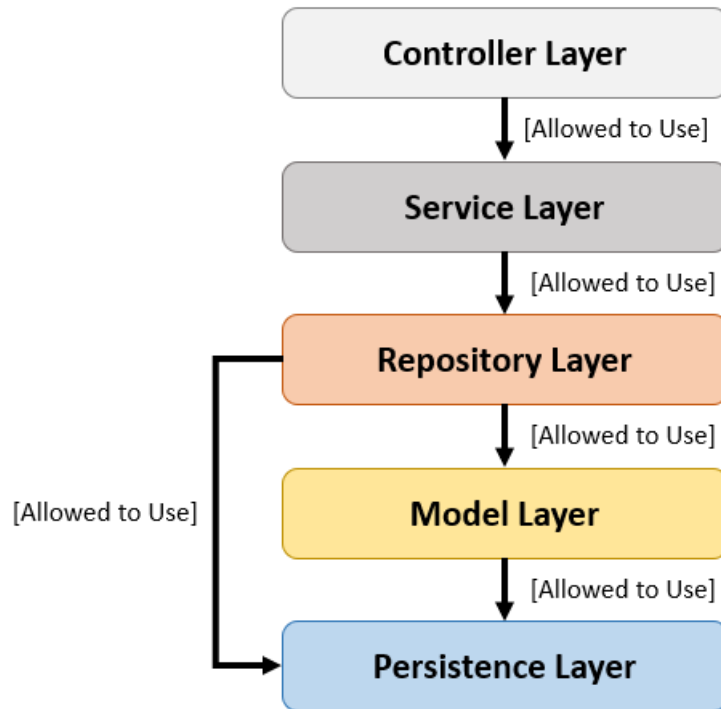
El laboratorio es una aplicación web basado en el siguiente modelo de datos.



La idea principal es llevar el registro de las asociaciones de las asignaturas en un periodo correspondiente por un rol específico. Para el sistema poseemos dos roles: Estudiante y Profesor. Por tanto, la asociación nos permite conocer si un usuario fue asistente al curso o si el usuario se refiere al docente.

Únicamente cuando el usuario sea estudiante deberá poseer una nota, sin embargo esta integridad no es llevada a cabo por el modelo de datos, debe ser implementada en una capa superior. Los docentes son las únicas personas que podrán crear cursos.

Para el backend, utilizaremos una arquitectura en capas de la siguiente forma:



Este patrón arquitectónico nos permite jerarquizar el flujo de información en la aplicación y no se debe permitir el uso de una capa superior en la arquitectura.

En el archivo `application.properties` (`src/main/resources/application.properties`) se configura la url, el nombre de usuario y la contraseña adecuada para la conexión con la base de datos.

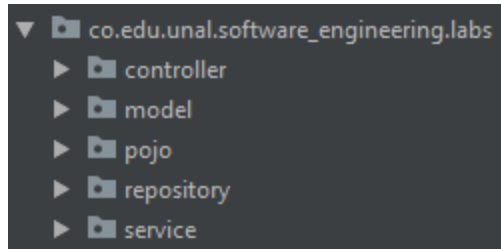
```
spring.datasource.url=jdbc:postgresql://localhost:5432/labs
spring.datasource.username=postgres
spring.datasource.password=admin

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.properties.hibernate.jdbc.time_zone=America/Bogota
```

Modifique el archivo para permitir la conexión de la aplicación con su base de datos.

En la aplicación encontramos un paquete correspondiente a cada capa de la arquitectura (Controller, Service, Repository, Model), más un package de [POJOs](#) (Plain Old Java Object).



Spring requiere de unos artefactos super importantes en las aplicaciones tipo JEE llamado [Beans](#). Para indicar el tipo de Bean, Spring nos provee las anotaciones necesarias para cada capa de la arquitectura.

```
@CrossOrigin
@RestController
public class UserController
```

La anotación “CrossOrigin” nos permite recibir peticiones de un origen distinto al entorno del Spring, resolviendo los problemas [CORS](#). La anotación “RestController” le indica al framework que será un controlador de comunicación REST. Dentro de este Bean podremos hacer uso de otros Beans (por la especificación de la arquitectura, deberían ser solo los Bean de servicio). Para este controlador se utiliza los servicios de Usuario, de Rol y un PasswordEncoder para las contraseñas.

```
@PostMapping( value = { "/registro/{roleId}" } )
public ResponseEntity register( @PathVariable Integer roleId,
                                @RequestBody RegisterUserPOJO userPOJO )
```

Las notaciones de “Get/Post/Delete/Put” Mapping nos indica el tipo de método HTTP que aceptará el método java del controlador. Como parámetros de entrada se recibirán las variables que vengan definidas por el path con la anotación de “PathVariable” y el cuerpo de la petición indicada con la anotación de “RequestBody”. Para recibir el cuerpo de la petición se utiliza generalmente los [POJOs](#) que son Objetos Java con [Getters y Setters](#), y tienen como atributos los parámetros del cuerpo de la petición. Ya dentro del controlador se debe implementar la funcionalidad básica para que funcione el sistema. Allí se puede distribuir la carga funcional con la capa de servicio. Generalmente se intenta que la capa controladora maneje todo lo relacionado con las peticiones (estados, respuestas, correctitud de la petición, etc), mientras que la capa de servicio se encargue de la lógica del negocio y tratamiento de la información.

En el sistema se encuentran implementados dos métodos para “UserController”. El primer método (**register**) permite realizar el registro de un nuevo usuario al sistema a un rol específico, indicado por el path. El segundo método (**registerRoleToUser**) permite asociar un usuario existente a un rol adicional.

Ejecute el servidor por línea de comandos o a través de un IDE.

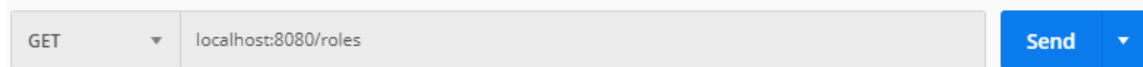
Para ejecutar el servidor Spring por línea de comandos, utilice el siguiente comando:

```
mvn clean install spring-boot:run
```

Para probar la correcta funcionalidad de estas implementaciones utilizaremos Postman. Descargar la [colección](#) e importarla en Postman. Allí encontraremos 5 peticiones: una para RoleController que retorna los roles existentes en el sistema, una inscripción de usuario de tipo profesor y estudiante, una para asociar el rol estudiante para el usuario de docente y una petición para realizar la autenticación mediante OAuth 2.0.

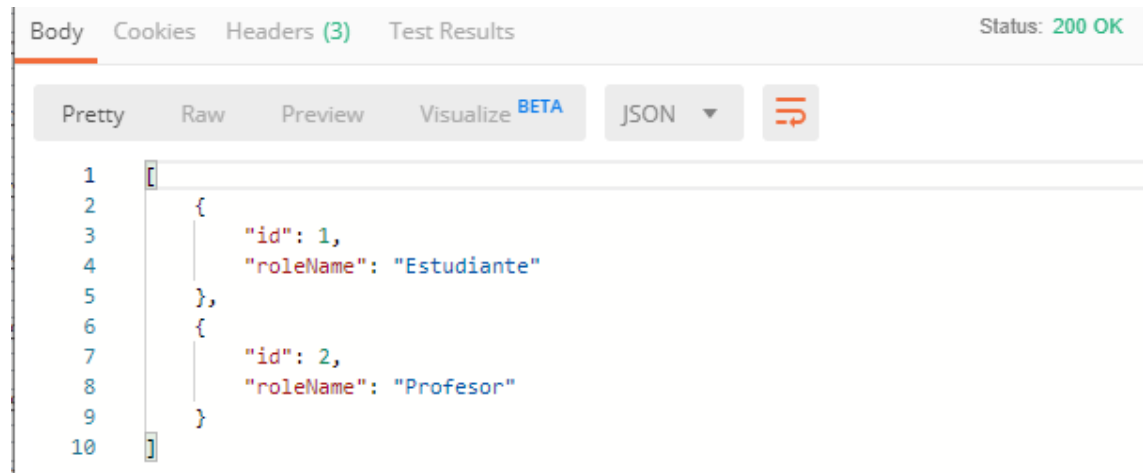
Primero realizaremos la petición de los roles:

En la barra URL configuramos el tipo de método http que utilizaremos y la url del servicio.

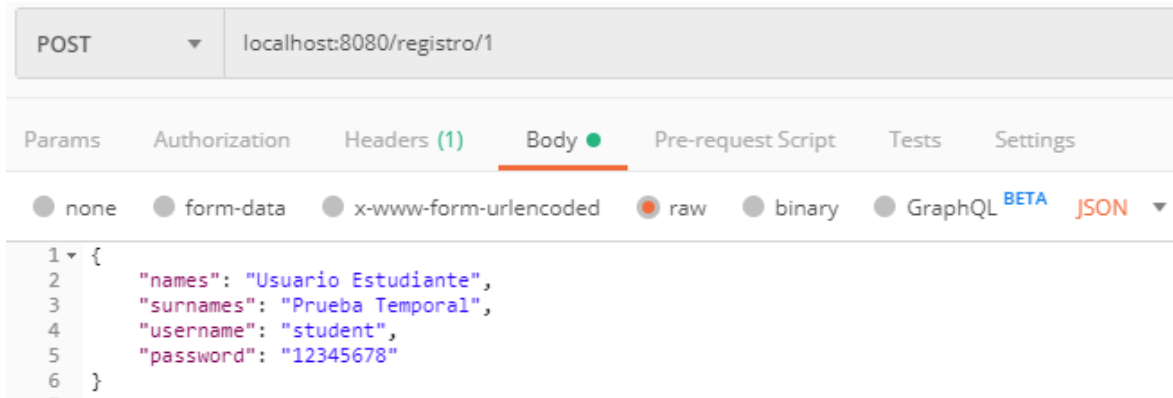


Al dar clic en el botón “Send” se realizará la petición al servidor de Spring.

Como respuesta Spring nos retornará una lista de roles en formato JSON, y un estado de 200.

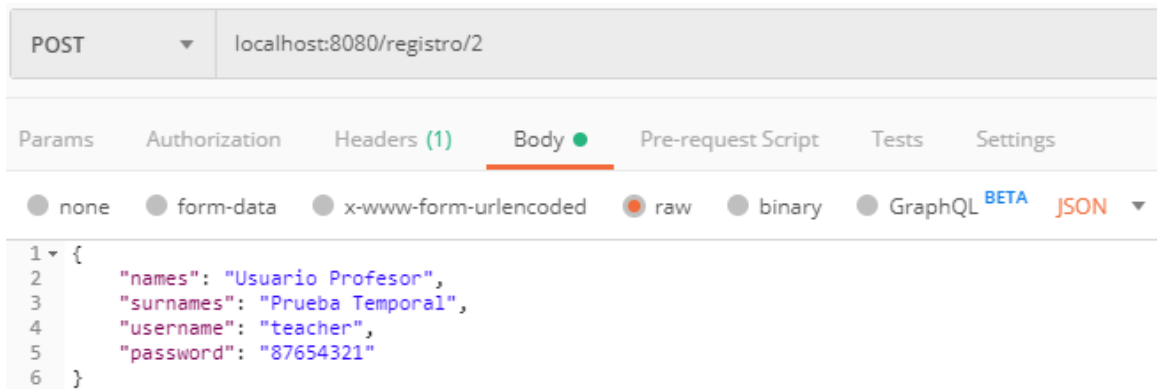


Segundo, realizaremos la petición para la inscripción de un usuario de tipo estudiante (rol con id 1).



Aquí podemos observar que se realiza mediante un método POST y tiene un cuerpo en formato JSON con los campos de RegisterUserPOJO.

Así mismo sucede con la petición para el registro del usuario de profesor.



Si revisamos la base de datos en Postgres, encontraremos que efectivamente se han registrado los usuarios. Una buena práctica es almacenar la contraseña en texto no legible, mediante un método hash (unidireccional) como los algoritmos SHA. Estos algoritmos permiten convertir la cadena de texto en una cadena nueva sin retorno, es decir que de ahora en adelante siempre tendremos que realizar el algoritmo hash a la contraseña para poder comparar si coincide con la almacenada, ya que no es posible devolverse el texto original.

Tabla “user”

	user_id	username	password	names	surnames
1	1	teacher	\$2a\$10\$n.hpF9FMhEt4J8lLfwNUQ.gcS...	USUARIO PROFESOR	PRUEBA TEMPORAL
2	2	student	\$2a\$10\$whrbEsXezIcJcaPBCbRfdOkp...	USUARIO ESTUDIA...	PRUEBA TEMPORAL

Tabla “user_role”

	user_id	role_id
1	1	2
2	2	1

Una vez los usuarios están registrados, podemos realizar la autenticación en el sistema a través de la petición “Authentication”.

POST localhost:8080/oauth/token

Params Authorization Headers (10) Body Pre-request Script Tests Settings

TYPE
Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Username soft-eng-ii

Password secret

☒ Show Password

POST localhost:8080/oauth/token

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE
<input checked="" type="checkbox"/>	username	teacher
<input checked="" type="checkbox"/>	password	87654321
<input checked="" type="checkbox"/>	grant_type	password

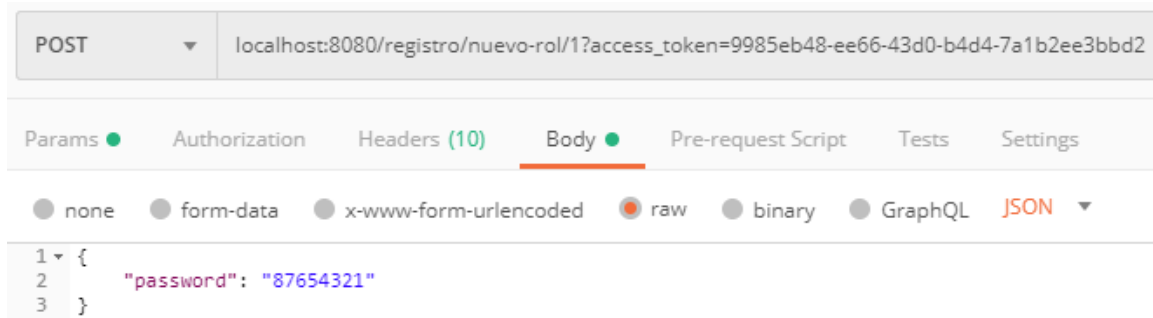
Si es exitosa la petición, deberá obtener la siguiente respuesta:

Body Cookies Headers (13) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "access_token": "9985eb48-ee66-43d0-b4d4-7a1b2ee3bbd2",
3   "token_type": "bearer",
4   "expires_in": 1599,
5   "scope": "read write trust"
6 }
```

Para realizar la asignación de un nuevo rol para el usuario de docente utilizamos la siguiente petición



Esta petición realizará la asignación del rol de estudiante al usuario de profesor, preservando el rol de docente. Es importante recalcar que para realizar la petición obligatoriamente el usuario debe estar autenticado (mire el `access_token` como un query) y requiere de la contraseña. **El sistema identifica al usuario a través del token de acceso.** Como respuesta obtendrá 400 si el rol no existe o el usuario ya posee el rol indicado. Si la contraseña no coincide o el token es inválido, devolverá el estado de 401. Si revisamos nuevamente la base de datos, encontramos el nuevo registro del usuario de profesor con el rol de estudiante.

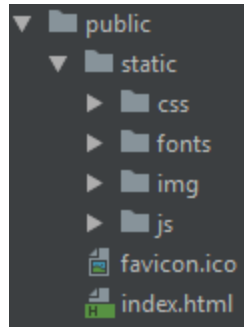
Tabla “user_role”

	user_id	role_id
1	2	1
2	1	2
3	1	1

En este punto, el backend está listo y expone una interfaz REST para realizar la conexión con el frontend.

[Vue](#) es un framework reactivo utilizado para la construcción de interfaces de usuario y su base es JavaScript, soportado por Node y npm. Su objetivo principal es dividir las vistas de usuario en componentes de código reutilizable que permitan una separación lógica de la vista.

En un proyecto Vue encontraremos una carpeta **public** que nos permite incorporar elementos estáticos como css, scripts JavaScripts, templates, etc. En este caso incorporamos Bootstrap 4 en los directorios.



En el archivo index.html, encontramos las referencias a los archivos estático con la notación de Vue para la ruta dinámica. Adicionalmente, en este archivo se requiere especificar en qué sección de la interfaz se renderiza el framework. Para ello se crea un div con id app. Dentro de este div, es donde ocurre la magia de Vue.

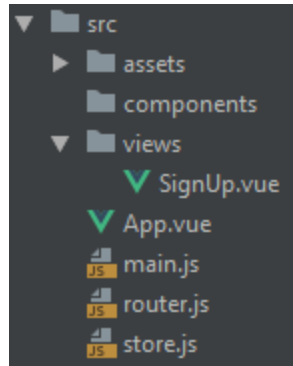
```
<link rel="icon"
      href="{%= BASE_URL %}>favicon.ico">

<link rel="stylesheet"
      href="{%= BASE_URL %}>static/css/bootstrap.min.css">

<script type="application/javascript"
        src="{%= BASE_URL %}>static/js/jquery.js"></script>
<script type="application/javascript"
        src="{%= BASE_URL %}>static/js/popper.js"></script>
<script type="application/javascript"
        src="{%= BASE_URL %}>static/js/bootstrap.min.js"></script>

<body>
  <noscript>
    <strong>We're sorry but the app doesn't
      work properly without JavaScript enabled.
      Please enable it to continue.</strong>
  </noscript>
  <div id="app"></div>
</body>
```

La carpeta principal de Vue es **src**, allí localizamos toda la funcionalidad de la aplicación. Vue tiene un archivo principal llamado main.js, el cual hace referencia al componente App.vue. Para este proyecto hemos decidido implementar Vuex y Router, que nos ayuda a estructurar la aplicación.



En el componente App.vue indicamos que se va a utilizar el router, y en esta sección se renderiza según la url, un componente de vue

```
<template>
  <div id="app">
    <router-view/>
  </div>
</template>
```

El router deberá contener las rutas de nuestra aplicación, y un componente asociado. Las buenas prácticas dicen que cada ruta debe relacionar un componente dentro de la carpeta **views**. Es decir, que cada componente principal o view, tendrá un router distinto. Las descomposiciones y componentes reutilizables se ubicaran en la carpeta **components**.

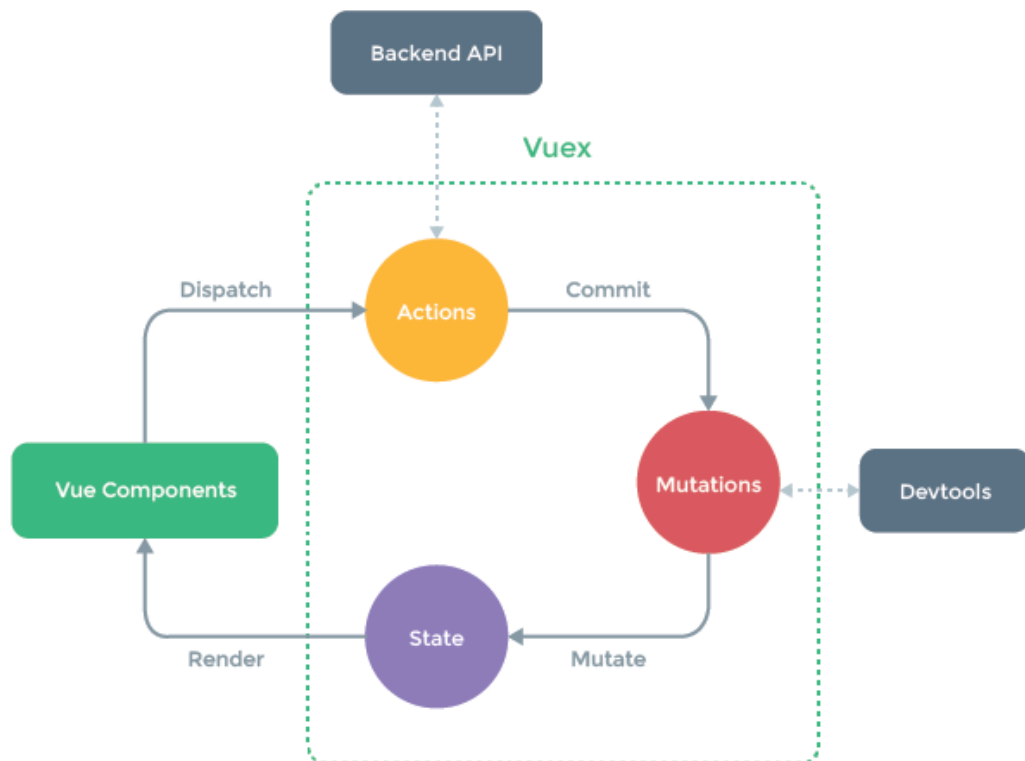
```
export default new Router({
  mode: 'history',
  base: process.env.BASE_URL,
  routes: [
    {
      path: "/registro",
      name: "signup",
      component: SignUp
    }
  ]
})
```

El Vuex nos permite definir variables globales para toda la aplicación (o states), mutaciones para modificar los states y acciones para responder ante las actividades externas.

```
export default new Vuex.Store({ options: {
  state: {
    backURL: 'http://localhost:8080'
  },
  mutations: {

  },
  actions: {

  }
})
```



Un componente de Vue posee 3 elementos: **template**, donde se ingresa el código html con algunas notaciones de vue; **script**, que incluye toda la lógica de la vista a nivel funcional; y **style** que permite dar estilo a las páginas, y se puede acotar el alcance del estilo mediante la palabra reservada *scoped* (aplica al componente y subcomponentes), o extenderlo a todo el sistema (sin la palabra *scoped*). Para un fácil mantenimiento, los estilos generales se ubican en archivos css aparte, o se organizan en el componente de principal App.vue

```
<template...>

<script...>

<style scoped...>
```

Para realizar peticiones HTTP, se suele usar la librería axios. Por ejemplo, en el componente SignUp.vue, en la sección de script se encuentra un método del [ciclo de vida](#) de Vue llamado **beforeCreate**. Este método realiza la petición al backend para traer los roles del sistema y mostrarlos en el select como opciones. La respuesta la almacena en una variable local (para el componente) llamada roles.

```
beforeCreate() {
  const rolesPath = '/roles';
  axios
    .get( this.$store.state.backURL + rolesPath )
    .then( response => {
      if( response.status !== 200 ){
        alert( "Error en la petición. Intente nuevamente" )
      }else{
        this.roles = response.data;
        console.log( this.roles );
      }
    }).catch( response => {
      alert( "No es posible conectar con el backend en este momento" );
      console.log( response );
    });
},
```

Finalmente, mediante la ayuda de Vue, le indicamos que muestre el resultado como opción.

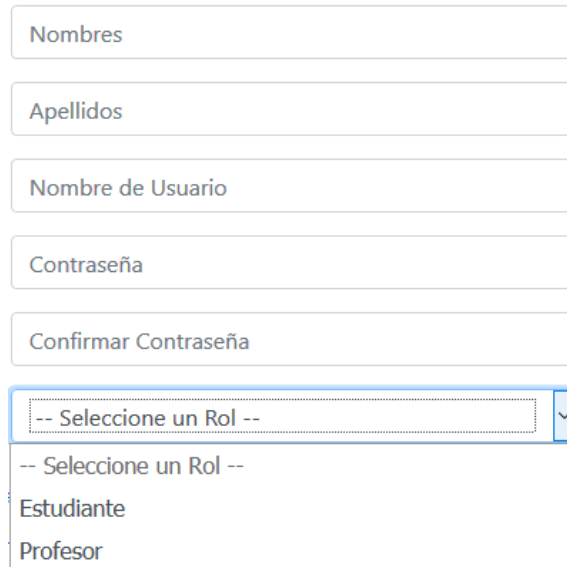
```
<select id="rol" class="form-control col-12
col-sm-10 col-md-7 offset-sm-1" v-model="role" required>
  <option value="" disabled selected>-- Seleccione un Rol --</option>
  <option v-for="role in roles" :value="role.id">{{role.roleName}}</option>
</select>
```

Inicie la aplicación utilizando el siguiente comando **en otra terminal** y mientras ejecuta el servidor de Spring Boot:

npm run serve

Inicie la aplicación de Vue mediante el servidor npm e ingrese a la dirección indicada desde un navegador (<http://127.0.0.1:<<PUERTO>>/registro>). El resultado de la vista es el siguiente:

Registro



Nombres

Apellidos

Nombre de Usuario

Contraseña

Confirmar Contraseña

-- Seleccione un Rol --

Estudiante

Profesor

Entregables

- Un documento **pdf** donde comparta el link de la plataforma donde subió el video.
- Añada el código **de su autoría** y explique cómo este permite la solución a los ejercicios propuestos en este enunciado.

Reglas:

- El laboratorio debe ser desarrollado por grupos de proyecto.
- Fecha de entrega: por acordar hora 23:59 máximo
- El documento debe nombrarse de la siguiente forma: ***Proyecto_L2.pdf*** (Cambie ***Proyecto*** por el nombre de su proyecto).