Scanner: Manual Tecnico

Instituto Tecnologico de Costa Rica
Joshua Molina Van Den Bergh
Esparza, Costa Rica
nostarckdev@gmail.com

Instituto Tecnologico de Costa Rica
Esteban Artavia Padilla
Heredia, Costa Rica
artaviap@outlook.com

Este manual tiene como objetivo, orientar a futuros desarrolladores del sistema, a ingresar nuevos jueces de programacion competitiva al sistema y a desplegar el sistema con Docker. Para esto ofrece una guia detallada paso a paso de lo que se debe realizar para ingresar nuevos jueces de manera que el nuevo desarrollador pueda ingresar un nuevo juez sin complicaciones.

Contents

1 Instalacion de un nuevo juez

Lo primero que hay que hacer, es ingresar el nuevo juez en la base de datos y esto es muy sencillo, simplemente hay que ingresar a la base de datos y correr la siguiente sentencia:

```
1  insert into tbl_judge(id, name, created)
2  values
3  (gen_random_uuid(),'[nombre del nuevo juez]', current_date)
       ;
```

1.1 El rol del juez en el sistema

En este proyecto el juez tiene el único rol de proveer problemas al sistema, esto se hace al registrar estudiantes dentro de la aplicación. Cuando se registra un estudiante hay que ingresar la informacion de este estudiante, dentro de esta informacion se piden los usuarios de todos los jueces del sistema. Si los usuarios se escribieron correctamente, el sistema será capaz de consultar los problemas resueltos por el estudiante en los jueces mediante una función que se conecta con la API del juez.

Para ver los problemas, hay que ingresar a la sección de problemas del sistema, marcado al lado izquierdo de la pantalla principal, cuando se entra en esta página, el sistema automáticamente hace una llamada a todas las apis de todos los jueces y consulta los problemas de todos los estudiantes ingresados en el sistema.También se puede actualizar los problemas al dar click en el siguiente icono de refrescar que esta debajo de la lista de problemas.

Con esto vemos que la creación de un estudiante tiene un rol importante cuando se quieren ingresar problemas al sistema. Entonces si queremos ingresar un nuevo juez, hay que trabajar en el modulo de creación de estudiantes.

1.2 Cambios en la interfaz

Primeramente, vamos a trabajar con los cambios de la interfaz para que el sistema sea capaz de ingresar usuarios del nuevo juez. Para esto hay seguir los siguientes pasos:

1. ingresar a la carpeta src/app/components/dialog-box
2. modificar el archivo dialog-box-component.html. En este archivo están todas todas las plantillas de todos los modales del sistema, estos modales se tienen que modificar para que se muestran los nuevos jueces que vayamos ingresando.
3. buscar el componente ng-template #studentTemplate, este es el modal para ingresar un nuevo estudiante al sistema o para editarlo
4. agregar un nuevo campo al formulario consultando el usuario del nuevo juez.

Si todo sale bien, cuando en el sistema se de click a agregar un nuevo estudiante, debería aparecer un nuevo campo consultando el usuario del nuevo juez del sistema.

Otro cambio de la interfaz que se debe realizar, está en el menú de estudiantes, este menu muestra toda la información de cada estudiante, entre esta información están los usuarios de los jueces del estudiante. Para modificar esta interfaz, hay que seguir los siguientes pasos:

1. Ir a la carpeta src/app/components/pages/students
2. Modificar el archivo student.component.ts
3. Buscar la lista llamada displayedColumns. Esta lista tiene el nombre de todas las columnas que se van a mostrar en la pantalla principal.
4. Agregar el nombre del nuevo juez a la lista.
5. Ingresar al archivo student.component.html, aquí se encuentra toda la interfaz de la página principal de los estudiantes.
6. Agregar una nueva columna en las columnas de los jueces, con el nombre del juez agregado en la lista de displayedColumns

Nota: Es importante que las variables que representen el nombre del juez, tengan de valor el mismo nombre del juez registrado en la base de datos.

1.3 Cambios en la logica

Seguidamente hay que trabajar en la parte lógica de este proceso.

1. Ir a la carpeta src/app/components/pages/students
2. Modificar el archivo student.component.ts, hay que ir específicamente a la función addRowData, esta función hace una llamada al service del estudiante nuevo para meter los nuevos datos al sistema, aquí hay que agregar el juez en la variables judges y this.allStudentsResult (tome como referencia los jueces que están ahí para saber cómo agregarlos)
3. Ingresar el juez en la función UpdateRowData (tome como referencia los jueces que están ahí para saber cómo agregarlos)

Una vez hecho esto, los datos del nuevo estudiante serán mandados a la clase controlador del estudiante en el sistema.

1.4 Registrar los problemas del nuevo juez

Ya cuando tenemos los usuarios del nuevo juez dentro de los estudiantes, lo que tenemos que hacer, es consultar los problemas de esos usuarios en los jueces. Para hacer esto hay que ingresar a src/controllers y seguidamente abrir el archivo problemController.js.

En este archivo tenemos una función llamada syncProblems() esta función es la encargada de llamar a todas las APIS de todos los jueces para consultar los problemas de cada estudiante dentro del sistema. Antes de continuar es importante tener en cuenta que hay que crear una función que se conecte con la API del nuevo juez. Esta función tiene que tener las siguientes características:

1. Debe recibir como parámetro el usuario del juez.
2. Debe devolver una lista de strings con los nombres de los problemas que resolvió el usuario en el juez.

Cuando ya tenemos esta función, debemos escribir una función llamada [nombre del juez]ApiCall(), que recibe como parámetros el id del usuario logueado en el sistema, y una lista de los usuarios del juez correspondiente. se recomienda utilizar la siguiente plantilla

```
async function NombreJuezAPICall(userID,
    studentsJudgeNombreJuez){

   var judgeName = "Nombre del juez";
   for(let i = 0; i < studentsJudgeNombreJuez.length; i++){
       var userName = studentsJudgeNombreJuez[i]["
           studentUsername"];
       var problemsSolvedList = await //aqui se llama la
           funcion que recoger los problemas del juez ejemplo
           problemsSolvedByUserOmegaUP(userName);

       var problems = "";
       for(let j = 0; j < problemsSolvedList.length; j++){
           problems += problemsSolvedList[j] + ";";
       }
       problems = problems.slice(0,-1);

       pool.connect(function (err, client, done){
           if (err) {
               console.log("Not able to stablish connection:
                   " + err);
           } else {
               //-- esto es para sincronizar los problemas
                   resueltos por un estudiante, si el problema
                    ya existen la DB solo se asocia que el
                   estudiante lo resolvio
               // Execution of a query directly into the DB
                   with parameters
               client.query('SELECT * from
                   prc_update_student_problems($1, $2, $3, $4)
                   ', [userID, studentsJudgeNombreJuez[i]["
                   studentId"], judgeName, problems], function
                    (err, result) {
                   done();
                   if (err)
                       console.log(err);
               });
           }
       })
   }
}
```

Simplemente debe intercambiar los nombres del juez, por el nombre del juez que se este ingresando en el sistema. En resumen lo que hace es, recorrer la lista de usuarios registrados en el juez, llamar una funcion que se conecta a la API del juez y retorna una lista con los problemas del estudiante. y finalmente registra los problemas en la base de datos.

Ahora si, ya con esto listo podemos volver a la función syncProblems(), aqui tenemos que seguir los siguientes pasos:

1. Agregar la siguiente variable:

```
const studentsJudgesNombreJuezResult = await client.query
    ('SELECT * from prc_get_students_judge($1, $2)', [
    userID, "NombreJuez"]).catch(err => {
           if (err) {
               console.log("Not able to stablish
                   connection: " + err);
               // Return the error with BAD REQUEST (400)
                    status
               res.status(400).send(err);
           }
       });
```

   Esta variable tendrá una lista de todos los usuarios del respectivo juez.

2. Agregar un arreglo llamado studentsJudgeNombreJuez

```
studentsJudgesNombreJuez = []
```

3. Agregar en el for la siguiente linea:

```
studentsJudgeNombreJuez.push(flattenObject(
    studentsJudgeNombreJuezResult.rows[i]));
```

4. Agregamos el nombre del juez y la función de su API en esta linea

```
const [codeForcesResult, codeChefResult, uvaResult,
    SPOJResult, CsAcademyResult, OmegaUpResult] = await
    Promise.all([codeForcesAPICall(userID,
    studentsJudgeCodeForces),
       codeChefAPICall(userID, studentsJudgeCodeChef),
       uvaAPICall(userID, studentsJudgeUVA), SpojAPICall(
           userID, studentsJudgeSPOJ)], CsAcademyAPICall(
           userID, studentsJudgeCsAcademy));
```

Con esto listo, ya podemos ingresar/editar estudiantes en el sistema, escribir sus usuarios en los nuevos jueces y visualizar los problemas en el sistema.

2 Despliegue

El despliegue se puede hacer de cualquier manera conveniente, solamente se necesita node y postgres, pero cada parte del proyecto también tiene un dockerfile para su despliegue, y el proyecto tiene un docker-compose. Con docker y docker-compose instalados y encendidos, el despliegue es tan simple como correr

```
$ docker-compose -f docker-compose.yaml up
```

desde el directorio raíz del proyecto.
Importante notar que se debe ejecutar

```
$ docker-compose -f docker-compose.yaml build
```

para que los cambios tomen efecto. También se puede construir una única parte del proyecto utilizando su dockerfile si fuera deseado.