Recuperatorio del segundo parcial de Laboratorio III

Aclaración:
Las partes se corregirán de manera secuencial (ascendentemente). Si están bien todos los puntos de una parte, habilita la corrección de la parte posterior.
Se pueden usar templates o reutilizar código hecho, pero en ningún caso, se debe incluir código obsoleto o que no cumpla ninguna función dentro del parcial.
Toda comunicación con el backend se realizará con AJAX. Todo pasaje de datos se hará con JSON.
Las vistas (páginas .php o .html) deben respetar fielmente las formas, colores, iconos, etc. Utilizar Bootstrap 4.
Se deben respetar los nombres de los archivos, de las clases, de los métodos y de los parámetros de las peticiones.
Respetar la estructura de directorios (index.php → ./public; fotos → ./src/fotos; clases en ./src/poo; vistas en ./src/views;).
La Api Rest que se utilizará en el parcial la proveerá el alumno. Se sugiere utilizar como base, la ya realizada en el segundo parcial de programación III -> APi Rest (Slim 4) para la concesionaria de autos Scaloneta, que interactúe con la clase Auto, la clase Usuario y la base de datos concesionaria_bd (autos - usuarios).
Agregar a la Api los siguientes verbos GET (a nivel de ruta 'front-end'): /login, /registro y /principal, para acceder a login.html, registro.html y principal.php respectivamente.

PARTE 1 (hasta un 5)

login.html – login.ts – validacion_login.ts

Asociar al evento click del botón ***btnEnviar*** una función que recupere el correo y la clave para luego invocar al verbo POST de la ruta **/login** (de la Api Rest).

(POST) /login
*Se envía un JSON → **user** (correo y clave) y retorna un JSON (éxito: true/false; jwt: JWT (con todos los datos del usuario) / null; status: 200/403).*

Si el atributo éxito del json de retorno es false, se mostrará (en un alert de BOOTSTRAP - danger) un mensaje que indique lo acontecido.
Si es true, se guardará en el LocalStorage el JWT obtenido y se redireccionará hacia ***principal.php***.

El botón 'Quiero Registrarme!' llevará al usuario hacia la página ***registro.html***.

Colores e iconos: lightpink; lightgrey – fas fa-envelope; fas fa-key;

Crear en TypeScript una función para verificar que todos los campos de la página, **login.html**, sean requeridos (no vacíos). Para ello, asociar al evento click del botón Enviar (*btnEnviar*) una función que administre dicha tarea, mostrando mensajes de error (en un alert de BOOTSTRAP - danger) o permitiendo el "envio", según corresponda.

registro.html – registro.ts – validacion_registro.ts

Se registrarán los datos de un nuevo usuario.
Asociar al evento click del botón ***btnRegistrar*** una función que recupere el correo, la clave, el nombre, el apellido, el perfil y la foto. Invocar al verbo POST de la ruta **/usuarios** (de la Api Rest).

*(POST) Alta de usuarios. Se agregará un nuevo registro en la tabla usuarios *.*
*Se envía un JSON → **usuario** (correo, clave, nombre, apellido, perfil**) y **foto**.*
** ID auto-incremental. ** propietario, encargado y empleado.*
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418).

Si el atributo éxito del json de retorno es false, se mostrará (en un alert de BOOTSTRAP - danger) un mensaje que indique lo acontecido.
Si es true, se redirigirá hacia ***login.html***.

Colores e iconos: rgb(153, 167,184); lightgrey – fas fa-envelope; fas fa-key; fas fa-user; far fa-id-card; fas fa-camera;

Crear en TypeScript las funciones necesarias para verificar que todos los campos de la página, **registro.html**, sean enviados correctamente. Para ello, asociar al evento click del botón Registrar (*btnRegistrar*) una función que administre dicha tarea, mostrando mensajes de error (en un alert de BOOTSTRAP - danger) o permitiendo el "envio", según corresponda.

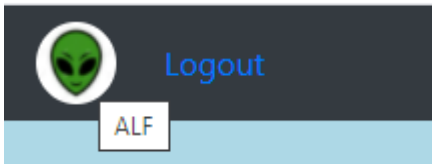
La función se llamará ***AdministradoraDeValidaciones*** y será la encargada de invocar a otras funciones que verifiquen:

- Campos no vacíos. (aplicarlo a todos los campos de la página)
  - ***ValidarCamposVacios(string): boolean***. Recibe como parámetro el valor del atributo id del campo a ser validado. Retorna true si no está vacío o false caso contrario.
- Cantidad máxima y mínima de caracteres. (aplicarlo al campo clave, nombre y apellido. Clave: mínimo 4, máximo 8, nombre: mínimo 4, máximo 10, apellido: mínimo 2, máximo 15)
  - ***ValidarCantidadCaracteres(number, number, number): boolean***. Recibe como parámetro el valor a ser validado y los valores mínimos y máximos del rango. Retorna true si el valor pertenece al rango o false caso contrario.
- Selección del perfil.
  - ***ValidarCombo(string, string): boolean***. Recibe como parámetro el valor del atributo id del combo a ser validado y el valor que no debe de tener. Retorna true si no coincide o false caso contrario.

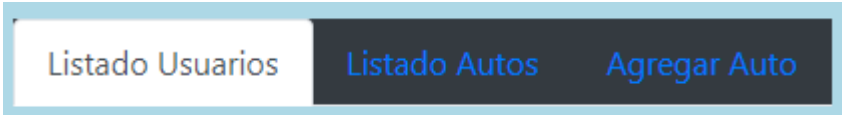
Según corresponda, se mostrarán mensajes (por un alert de BOOTSTRAP - danger) informando el error encontrado.

principal.php – principal.ts

Debe tener la foto del usuario logueado (al pasar el mouse por sobre la mísma, se debe mostrar el nombre del usuario). La misma debe ser redonda. La única opción de menú será el Logout, que al pulsarlo redigirá hacia login.html (limpiando el jwt).



En el cuerpo, se tendrá un tab con las siguientes solapas.



Generar por TypeScript la carga del listado de usuarios (al terminar de cargarse la página) con la solapa *Listado Usuarios* 'activa'.
Dentro del cuerpo del tab, se mostrará el listado de los usuarios en una tabla de BOOTSRTAP (elegir un estilo). Las fotos tendrán un ancho y un largo de 50px.

Invocar al verbo GET (nivel de aplicación, de la Api Rest).

(GET) Listado de usuarios. Mostrará el listado completo de los usuarios (array JSON).
Retorna un JSON (éxito: true/false; mensaje: string; dato: stringJSON; status: 200/424).

Si el atributo éxito del json de retorno es false, se mostrará (en un alert de BOOTSTRAP - danger) el mensaje recibido.
Si es true, se armará (en el frontend) el listado que proviene del atributo **dato** del json de retorno.

CORREO	NOMBRE	APELLIDO	PERFIL	FOTO
a@a.com	a	a	propietario	
b@b.com	b	b	empleado	
c@c.com	c	c	supervisor	
d@d.com	d	d	empleado	
e@e.com	e	e	empleado	

***NOTA**: en cada interacción, se debe verificar la autenticidad del usuario, para ello, se tiene que invocar al verbo GET (ruta **/login**, de la ApiRest).*

*(GET) Se envía el JWT → **token** (en el header) y se verifica. En caso exitoso, retorna un JSON (éxito: true/false; mensaje: string; status: 200/403).*
*Si el **JWT** no es válido, redirigir al **login.html**.*

Al pulsar la solapa *Listado Autos* del tab (dejarla como activa), se mostrará el listado de los autos en una tabla de BOOTSRTAP (elegir otro estilo).
Invocar al verbo GET (ruta **/autos**, de la Api Rest).

(GET) Listado de autos. Mostrará el listado completo de los autos (array JSON).
Retorna un JSON (éxito: true/false; mensaje: string; dato: stringJSON; status: 200/424)

Si el atributo éxito del json de retorno es false, se mostrará (en un alert de BOOTSTRAP - danger) el mensaje recibido.
Si es true, se armará (en el frontend) el listado que proviene del atributo **dato** del json de retorno.

MARCA	COLOR	MODELO	PRECIO
citroen	blanco	c4	123456
renault	gris	r9	50500
ford	rojo	fiesta	95300

Al pulsar la solapa *Algrear Autos*, se mostrará el formulario de alta de auto (cómo muestra la imagen) en el cuerpo del tab.

Formulario alta / modificación

Colores e iconos: darkcyan – fas fa-trademark; fas fa-palette; fas fa-car; fas fa-dollar-sign;

Asociar al evento click del botón ***btnAgregar*** una función que recupere el color, la marca, el precio y el modelo. Generar las validaciones correspondientes (campos NO vacíos) para TODOS los campos del formulario.
Invocar al verbo POST (nivel de aplicación de la Api Rest).

*(POST) Alta de autos. Se agregará un nuevo registro en la tabla autos *.*
*Se envía un JSON → **auto** (color, marca, precio y modelo).*
** ID auto-incremental.*
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418).

Si el atributo éxito del json de retorno es false, se mostrará (en un alert de BOOTSTRAP - danger) el mensaje recibido.

Si es true, se mostrará un mensaje (alert de BOOTSTRAP – success) indicando lo sucedido.

***NOTA**: en cada interacción, se debe verificar la autenticidad del usuario, para ello, se tiene que invocar al verbo GET (ruta **/login**, de la ApiRest).*

*(GET) Se envía el JWT → **token** (en el header) y se verifica. En caso exitoso, retorna un JSON (éxito: true/false; mensaje: string; status: 200/403).*
*Si el **JWT** no es válido, redirigir al **login.html**.*

PARTE 2

Agregar al listado de autos, dos columnas extras, cada una con un botón.
El primero con un botón (btn-danger) que permitirá el borrado del auto, previa confirmación del usuario, preguntando si el auto con tal marca, color y modelo se quiere borrar de la base de datos.
Si se confirma, se eliminará el auto, invocando al verbo DELETE (de la Api Rest).

(DELETE) Borrado de autos por ID.
*Recibe el ID del auto a ser borrado (**id_auto**, en el raw) más el JWT → **token** (en el header).*
*Si el perfil es '**propietario**' se borrará de la base de datos. Caso contrario, se mostrará el mensaje correspondiente (indicando que usuario intentó realizar la acción).*
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418).

Si el atributo éxito del json de retorno es false, se mostrará (en un alert de BOOTSTRAP - warning) el mensaje recibido.
Si es true, refrescar el listado de autos.

El segundo con botón (btn-info) que permitirá la modificación del auto seleccionado. Para ello, se cargarán todos los datos del auto en el formulario (formulario alta / modificación). Mostrarlo en una **ventana modal** (de tipo pop-up).
Al pulsar el botón ***btnModificar*** (cambiarlo en el formulario) se invocará al verbo PUT (de la Api Rest).

(PUT) Modificar los autos por ID.
*Recibe el JSON con los valores del auto a ser modificado → **auto** (color, marca, precio, modelo), el ID → **id_auto** (id del auto a ser modificado) (en el raw) y el JWT → **token** (en el header).*
*Si el perfil es '**encargado**' se modificará de la base de datos. Caso contrario, se mostrará el mensaje correspondiente (indicando que usuario intentó realizar la acción).*
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

Si el atributo éxito del json de retorno es false, se mostrará (en un alert de BOOTSTRAP - warning) el mensaje recibido.
Si es true, refrescar el listado de autos.

***NOTA**: en cada interacción, se debe verificar la autenticidad del usuario, para ello, se tiene que invocar al verbo GET (ruta **/login**, de la ApiRest).*

*(GET) Se envía el JWT → **token** (en el header) y se verifica. En caso exitoso, retorna un JSON (éxito: true/false; mensaje: string; status: 200/403).*
*Si el **JWT** no es válido, redirigir al **login.html**.*

NOTA 2: Validar los campos en la modificación.