58 - TypeScript y Angular: definición de una interfaz para controlar los tipos

En Angular los modelos de datos se los representa generalmente con una interfaz, veamos un ejemplo utilizando una interfaz para definir el modelo de datos.

Problema

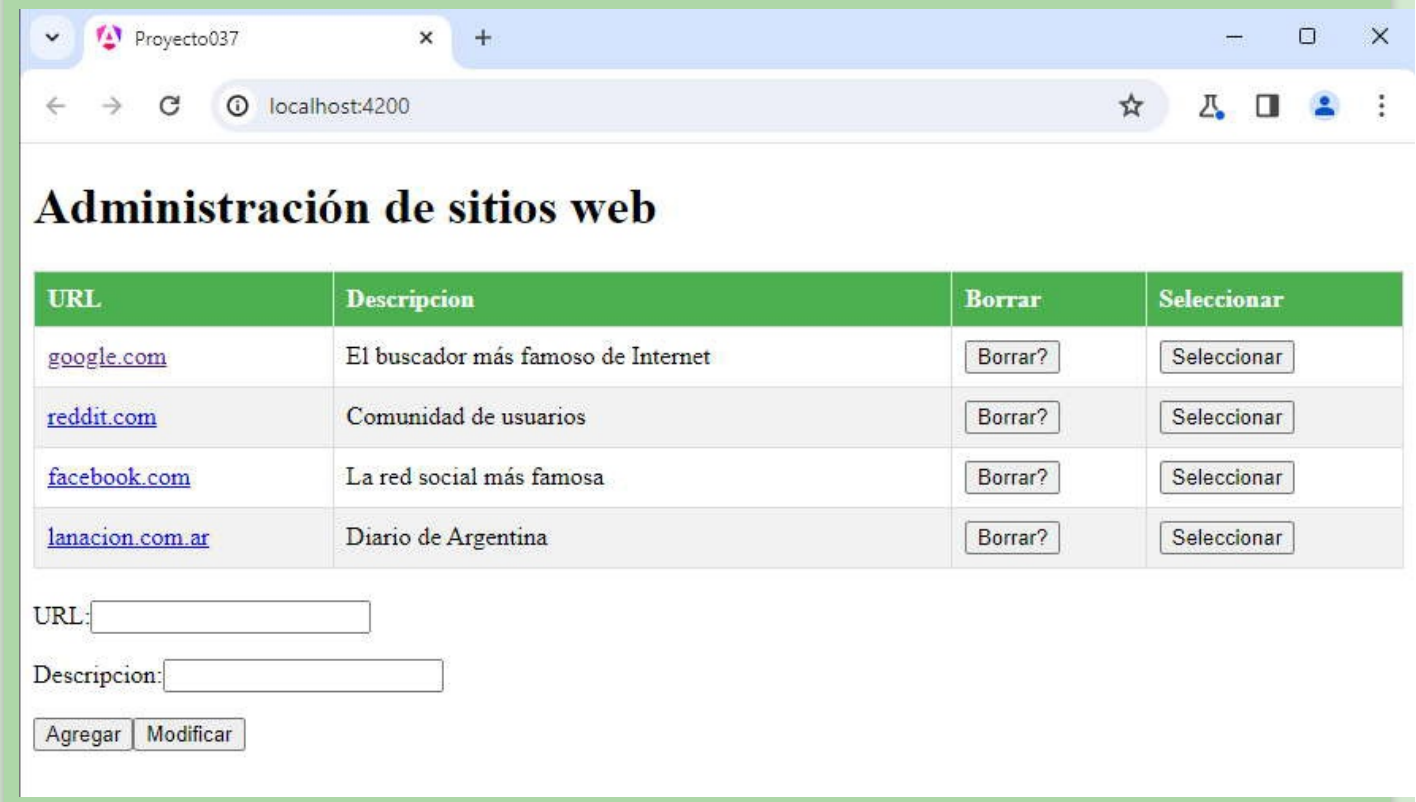
Implementar una aplicación que permita almacenar en un arreglo los sitios favoritos de internet y una descripción del sitio.

Para resolver el problema vamos a definir la siguiente interfaz:

```
export interface Sitio {
  url: string;
  descripcion: string
}
```

La aplicación debe almacenar en el localStorage del navegador todos los datos para evitar que se pierdan cuando cerremos la aplicación.

Si ejecutamos la aplicación tenemos una pantalla similar a:



Podemos probar la aplicación en la web aquí.

Como primer paso pasemos a crear nuestro proyecto:

- Crearemos primero el proyecto

  ```
  ng new proyecto037
  ```

- El segundo paso es crear un archivo en la carpeta 'app' con el nombre 'sitio.ts' y en su interior codificamos la interfaz 'Sitio', la idea de separarla en un archivo independiente es porque la vamos a utilizar a la interfaz 'Sitio' en más de un archivo:

  sitio.ts

```
export interface Sitio {
  url: string;
  descripcion: string
}
```

Disponemos la palabra clave 'export' para poder importarla en otros archivos.

- Crearemos el servicio que se va a encargar de almacenar y recuperar los datos del localStorage.

  Para crear el servicio desde Angular CLI ejecutamos:

  ```
  ng generate service sitios
  ```

  Se nos generan 2 archivos y el que debemos modificar es el archivo sitios.service.ts.

  sitios.service.ts

```
import { Injectable } from '@angular/core';
import { Sitio } from './sitio';

@Injectable({
  providedIn: 'root',
})
export class SitioService {
  private localStorageNombre = 'sitios';

  obtenerSitios(): Sitio[] {
    const sitiosStr = localStorage.getItem(this.localStorageNombre)
;
    return sitiosStr ? JSON.parse(sitiosStr) : [];
  }

  agregarSitio(sitio: Sitio): void {
    const sitios = this.obtenerSitios();
    sitios.push(sitio);
    localStorage.setItem(this.localStorageNombre, JSON.stringify(si
tios));
  }

  borrarSitio(url: string): void {
    const sitios = this.obtenerSitios();
    const index = sitios.findIndex((s) => s.url === url);
    if (index !== -1) {
      sitios.splice(index, 1);
      localStorage.setItem(this.localStorageNombre, JSON.stringify(
sitios));
    }
  }

  modificarSitio(sitio: Sitio, urlBuscar:string): void {
    const sitios = this.obtenerSitios();
    const index = sitios.findIndex((s) => s.url === urlBuscar);
    if (index !== -1) {
      sitios[index] = sitio;
      localStorage.setItem(this.localStorageNombre, JSON.stringify(
sitios));
    }
  }
}
```

```
  obtenerSitios(): Sitio[] {
    const sitiosStr = localStorage.getItem(this.localStorageNombre);
    return sitiosStr ? JSON.parse(sitiosStr) : [];
  }
```

El método 'obtenerSitios' retorna un arreglo de la interfaz 'Sitio'. Lo primero que hacemos es con el método getItem recuperamos los datos del localStorage o nos retorna null si todavía no hay nada almacenado.

Finalmente retornamos un arreglo vacío [] si la variable 'sitioStr' almacena null o retornamos un objeto JSON a partir del string almacenado en el localStorage.

```
  agregarSitio(sitio: Sitio): void {
    const sitios = this.obtenerSitios();
    sitios.push(sitio);
    localStorage.setItem(this.localStorageNombre, JSON.stringify(sitios));
  }
```

El método agregarSitio recibe como parámetro un objeto de tipo 'Sitio'. Lo primero llama al método obtenerSitios que retorna como vimos un arreglo vacío [] o un arreglo con todos los sitios almacenados actualmente en el localStorage.

Agregamos al arreglo 'sitios' el sitio que llega como parámetro y finalmente volvemos a almacenar el arreglo en el localStorage. Recordar que en el localStorage debemos convertir el arreglo 'sitios' a string mediante el método JSON.stringify.

```
  borrarSitio(url: string): void {
    const sitios = this.obtenerSitios();
    const index = sitios.findIndex((s) => s.url === url);
    if (index !== -1) {
      sitios.splice(index, 1);
      localStorage.setItem(this.localStorageNombre, JSON.stringify(sitios));
    }
  }
```

Primero recuperamos todos los sitios almacenados en el localStorage, buscamos la 'url' del sitio que queremos borrar llamando al método 'findIndex' de la clase Array. Si lo encontramos procedemos a eliminar un elemento del arreglo llamando al método 'splice' y finalmente volvemos a guardar los datos del arreglo en el localStorage.

```
  modificarSitio(sitio: Sitio, urlBuscar:string): void {
    const sitios = this.obtenerSitios();
    const index = sitios.findIndex((s) => s.url === urlBuscar);
    if (index !== -1) {
      sitios[index] = sitio;
      localStorage.setItem(this.localStorageNombre, JSON.stringify(sitios));
    }
  }
```

Buscamos la url del sitio que queremos modificar, y en caso de encontrarlo procedemos a asignar a dicha componente los datos modificados que llegan en el parámetro 'sitio'.

- Pasemos a codificar la lógica de la componente, la cual se apoya en el servicio para recuperar, agregar y modificar datos.

  app.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { Sitio } from './sitio';
import { SitioService } from './sitios.service';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet, FormsModule],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
  sitio: Sitio = { url: '', descripcion: '' }; // datos cargados en
 el formulario
  sitios: Sitio[] = []; // arreglo con todos los sitios
  sitioSeleccionado: string = ''; // se almacena el sitio seleccion
ado con el boton "seleccionar"

  constructor(private sitioService: SitioService) {
    this.sitios = sitioService.obtenerSitios();
  }

  borrar(url: string): void {
    this.sitioService.borrarSitio(url);
    this.sitios = this.sitioService.obtenerSitios();
  }

  agregar(): void {
    if (this.sitio.url.trim() === '') {
      alert('Debe ingresar una URL de sitio');
      return;
    }

    if (this.sitios.some((s) => s.url === this.sitio.url)) {
      alert('Ya existe un sitio con dicha URL');
      return;
    }

    this.sitioService.agregarSitio(this.sitio);
    this.sitio = { url: '', descripcion: '' };
    this.sitios = this.sitioService.obtenerSitios();
  }

  seleccionar(sitio: Sitio): void {
    this.sitioSeleccionado = sitio.url; // Guardar la URL seleccion
ada
    this.sitio.url = sitio.url;
    this.sitio.descripcion = sitio.descripcion;
  }

  modificar(): void {
    if (this.sitio.url.trim() === '') {
      alert('Debe ingresar una URL de sitio');
      return;
    }

    const sitioExistente = this.sitios.find((s) => s.url === this.s
itio.url && s.url !== this.sitioSeleccionado);
```

```
    if (sitioExistente) {
      alert('Ya existe un sitio con dicha URL');
      return;
    }

    if (this.sitioSeleccionado) {

      const index = this.sitios.findIndex((s) => s.url === this.sit
ioSeleccionado);
      if (index !== -1) {
        this.sitios[index] = { ...this.sitio };
        this.sitioService.modificarSitio(this.sitio,this.sitioSelec
cionado);
        this.sitio = { url: '', descripcion: '' };
        this.sitios = this.sitioService.obtenerSitios();
        this.sitioSeleccionado = '';
      }
    }
  }
}
```

```
sitio: Sitio = { url: '', descripcion: '' }; // datos cargados en el formulario
```

La propiedad 'sitio' es de la interfaz 'Sitio' y lo inicializamos con string vacíos tanto para la url como la descripción. Estos dos atributos están asociados a los controles input del formulario.

```
sitios: Sitio[] = []; // arreglo con todos los sitios
```

Corresponde al arreglo con todos los sitios. En el constructor se inicializa con los sitios almacenados hasta este momento en el localStorage.

```
sitioSeleccionado: string = ''; // se almacena el sitio seleccionado con el boton "seleccionar"
```

Esta propiedad se requiere para el proceso de modificación de datos.

```
constructor(private sitioService: SitioService) {
  this.sitios = sitioService.obtenerSitios();
}
```

El constructor se le inyecta el servicio y se crea en el mismo parámetro la propiedad 'sitioService'. Además llamamos al método obtenerSitios para recuperar todos los datos almacenados en el localStorage. Como sabemos el método retorna un arreglo de tipo Sitio.

```
borrar(url: string): void {
  this.sitioService.borrarSitio(url);
  this.sitios = this.sitioService.obtenerSitios();
}
```

El método borrar se lo llama desde la plantilla cuando se presiona el botón 'borrar' y llega como dato la 'url' del sitio a borrar. Llamamos al método borrarSitio del servicio sitioService que actualiza el localStorage eliminando dicho sitio. Además volvemos a llamar al método obtenerSitios del servicio para que nos retorne actualizado todos los sitios almacenados en el localStorage.

```
agregar(): void {
  if (this.sitio.url.trim() === '') {
    alert('Debe ingresar una URL de sitio');
    return;
  }

  if (this.sitios.some((s) => s.url === this.sitio.url)) {
    alert('Ya existe un sitio con dicha URL');
    return;
  }

  this.sitioService.agregarSitio(this.sitio);
  this.sitio = { url: '', descripcion: '' };
  this.sitios = this.sitioService.obtenerSitios();
}
```

El método agregar hace algunas validaciones como que hayamos ingresado una 'url' en el formulario o que dicha 'url' ya si haya cargado en el arreglo.

Si pasa las validaciones llamamos al método agregarSitio del servicio sitioService y volvemos a actualizar el arreglo llamando a obtenerSitios.

```
seleccionar(sitio: Sitio): void {
  this.sitioSeleccionado = sitio.url; // Guardar la URL seleccionada
  this.sitio.url = sitio.url;
  this.sitio.descripcion = sitio.descripcion;
}
```

El método seleccionar se llama cuando se presiona en la plantilla el botón 'seleccionar', donde procedemos a almacenar en la propiedad sitioSeleccionado la url del sitio seleccionado. También actualizamos el formulario asignando valores al atributo 'sitio'.

```
modificar(): void {
  if (this.sitio.url.trim() === '') {
    alert('Debe ingresar una URL de sitio');
    return;
  }

  const sitioExistente = this.sitios.find((s) => s.url === this.sitio.url && s.url !== this.sitioSeleccionado);
  if (sitioExistente) {
    alert('Ya existe un sitio con dicha URL');
    return;
  }

  if (this.sitioSeleccionado) {

    const index = this.sitios.findIndex((s) => s.url === this.sitioSeleccionado);
    if (index !== -1) {
      this.sitios[index] = { ...this.sitio };
      this.sitioService.modificarSitio(this.sitio,this.sitioSeleccionado);
      this.sitio = { url: '', descripcion: '' };
      this.sitios = this.sitioService.obtenerSitios();
      this.sitioSeleccionado = '';
    }
  }
}
```

Finalmente el método modificar luego de hacer algunas validaciones procedemos a buscar el sitio que seleccionamos anteriormente y proceder a llamar al método modificarSitio del objeto sitioService pasando como parámetos los datos del formulario que se encuentran en el atributo 'sitio' y la url del sitio que anteriormente seleccionamos y almacenamos en la propiedad sitioSeleccionado.

- La plantilla muestra la tabla HTML con los sitios web y un formulario para la carga de sitios.

  app.component.html

```
<div>
  <h1>Administración de sitios web</h1>
  <table>
    <thead>
      <tr>
        <th>URL</th>
        <th>Descripcion</th>
        <th>Borrar</th>
        <th>Seleccionar</th>
      </tr>
    </thead>
    <tbody>
      @for(sitio of sitios; track sitio.url) {
      <tr>
        <td><a href="https://www.{{sitio.url}}" target="_blank">{{s
itio.url}}</a></td>
        <td>{{sitio.descripcion}}</td>
        <td><button (click)="borrar(sitio.url)">Borrar?</button></t
d>
        <td><button (click)="seleccionar(sitio)">Seleccionar</butto
n></td>
      </tr>
      } @empty {
        <tr>
          <td colspan="4">
            No hay sitios.
          </td>
        </tr>
      }
    </tbody>
  </table>
  <div>
    <p>
      URL:<input type="text" [(ngModel)]="sitio.url" />
    </p>
    <p>
      Descripcion:<input type="text" [(ngModel)]="sitio.descripcion
" />
    </p>
    <p>
      <button (click)="agregar()">Agregar</button>
      <button (click)="modificar()">Modificar</button>
    </p>
  </div>
</div>
<router-outlet />
```

```
    @for(sitio of sitios; track sitio.url) {
    <tr>
      <td><a href="https://www.{{sitio.url}}" target="_blank">{{sitio.url}}</a></td>
      <td>{{sitio.descripcion}}</td>
      <td><button (click)="borrar(sitio.url)">Borrar?</button></td>
      <td><button (click)="seleccionar(sitio)">Seleccionar</button></td>
    </tr>
    } @empty {
      <tr>
        <td colspan="4">
          No hay sitios.
        </td>
      </tr>
    }
```

Utilizamos la estructura repetitiva '@for' en la plantilla para mostrar el nombre del sitio, su descripción y dos botones que nos permiten borrar o seleccionar el sitio.

Si el arreglo 'sitios' está vacío se ejecuta el bloque @empty.

- La hoja de estilo de la componente.

  app.component.css

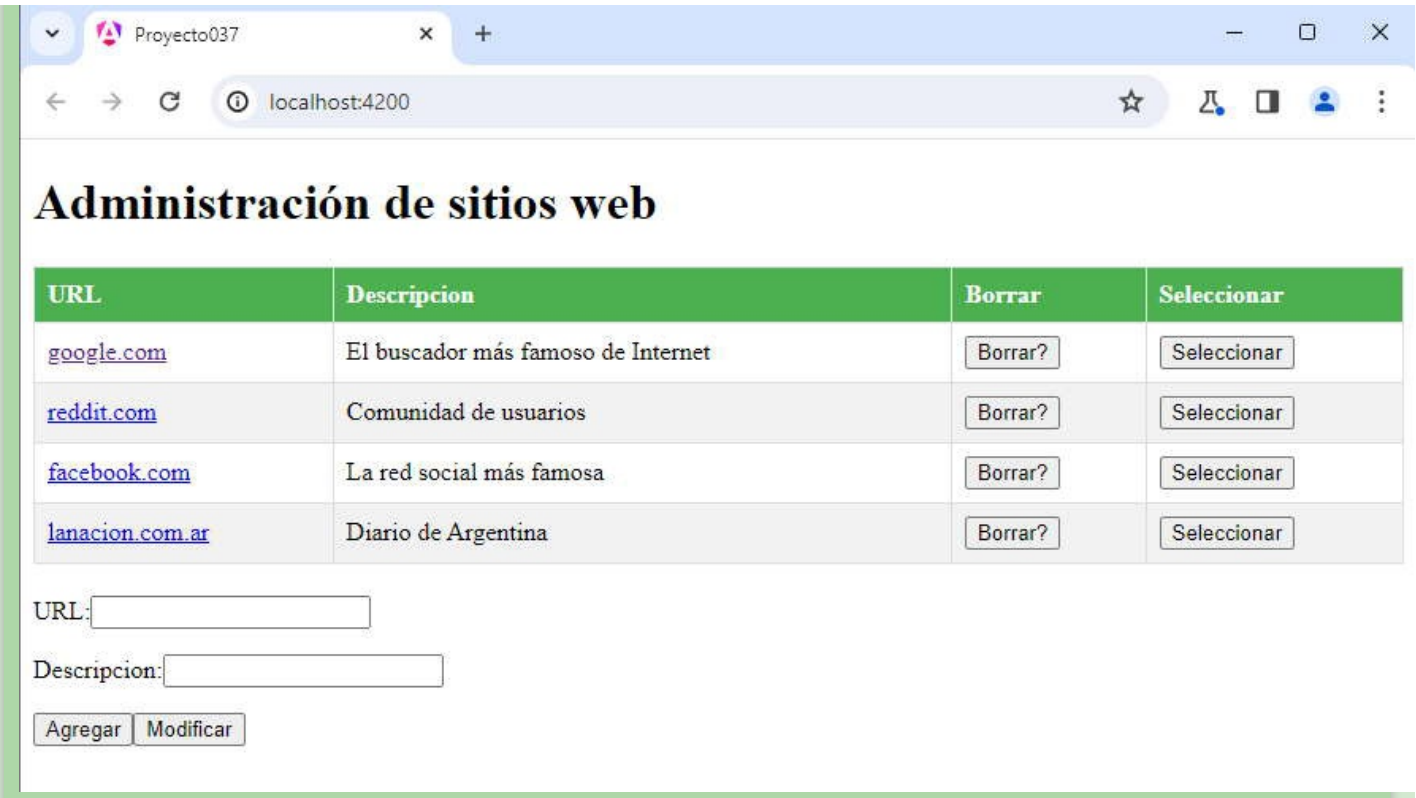
```
table {
  width: 100%;
  border-collapse: collapse;
  margin-bottom: 20px;
}

th, td {
  border: 1px solid #dddddd;
  padding: 8px;
  text-align: left;
}

tr:nth-child(even) {
  background-color: #f2f2f2;
}

th {
  background-color: #4CAF50;
  color: white;
}
```

Si ejecutamos la aplicación tenemos una interfaz similar a:



Podemos probar esta aplicación en la web aquí.

Retornar