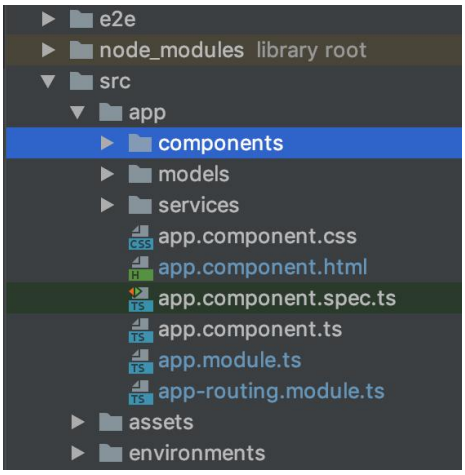
Introducción al consumo de servicios REST y Websockets en Angular

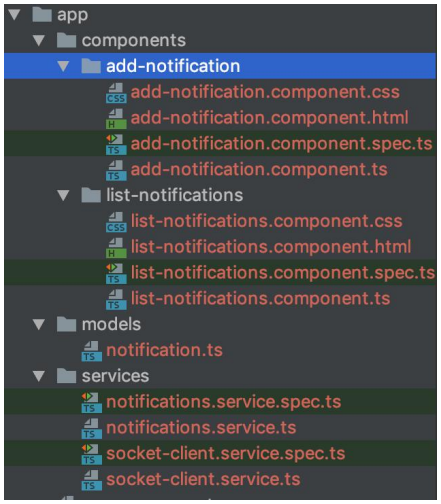
En la línea de comandos dentro del directorio app, creamos 3 carpetas: components, models, services



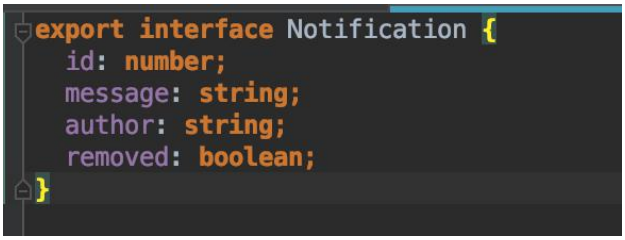
Desde la línea de comandos nos colamos dentro de la carpeta components y con el siguiente comando creamos 2 componentes: ng g c nombre-componente. Automáticamente se generaran todos los archivos correspondientes a los componentes.

En la línea de comandos dentro de la carpeta models, generamos la interfaz con el siguiente comando: ng g interface nombre-interface.

Dentro de la carpeta services crearemos 2 servicios con el siguiente comando: ng g s nombre-servicio.
Los directorios deberan quedar similar al presente ejemplo:



Dentro del archivo notification.ts, declaramos los atributos que contendrá el modelo:



Dentro del archivo notifications.service.ts, declaramos los datos principales para la conexión a los servicios, así como las funciones que se encargaran del listado y el guardado de datos.

```
import { Injectable } from '@angular/core';
import {HttpClient, HttpHeaders} from '@angular/common/http';
import {Observable} from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class NotificationsService {
  private REST_API_SERVER = 'http://localhost:8081';
  private headers = new HttpHeaders( headers: {
    'Content-Type': 'application/json'
  });
  constructor(public http: HttpClient) {}

  public getListNotifications(): Observable<any> {
      const url = this.REST_API_SERVER + '/demo/getDataListNotifications';
      return this.http.get(url, options: {headers: this.headers});
  }

  public saveDataNotification(body: any): Observable<any> {
    const url = this.REST_API_SERVER + '/demo/saveDataNotification';
    return this.http.post(url, body, options: {headers: this.headers});
  }
}
```

Dentro del archivo socket-client.service.ts, declaramos la url del servidor donde se encuantran los servicios, instanceamos igual las variables necesarias para crear la conexión al socket así como el observable para regresar la información a todos los que se suscriban a este socket.

```
  private serverUrl = 'http://localhost:8081';
  private stompClient;
  private updateData$: Subject<any> = new Subject<any>();
  constructor() {
    this.intializeWebSocketConnection();
  }

  intializeWebSocketConnection() {
    console.log('ejecuto el codigo');
    const ws = new SockJS( url: this.serverUrl + '/ws');
    this.stompClient = Stomp.over(ws);
    this.stompClient.__proto__.debug = () => {};

    this.stompClient.connect({}, () => {
      this.stompClient.subscribe('/topic/updateListNotifications', (response) => {
        console.log(JSON.parse(response.body));
        this.setDataUpdated(JSON.parse(response.body));
      });
    });
  }

  setDataUpdated(data) {
    this.updateData$.next(data);
  }
  getDataUpdated$(): Observable<any> {
    return this.updateData$.asObservable();
  }
```

Desde el archivo list-notifications.component.ts, que le pertenece al componente List Notifications, en el cual incializamos la variable que contendrá el listado que nos regrese el servicio, instanciamos las clases de los servicios para poder conectarnos al socket y poder mandar a llamar el método que hace la petición al servicio de listado.

```
export class ListNotificationsComponent implements OnInit,  OnDestroy{
  public listNotifications: Notification[] = [];
  destroy$: Subject<boolean> = new Subject<boolean>();
  constructor(public notificationService: NotificationsService, private socketClient: SocketClientService) {
    this.getDataTable();
    this.socketClient.getDataUpdated$().pipe(takeUntil(this.destroy$)).subscribe( next: (response: any) => {
      if (response && response.length > 0) {
        this.listNotifications = response;
      }
    });
  }

  ngOnInit() {
  }
  public getDataTable() {
    this.notificationService.getListNotifications().pipe(take( count: 1)).subscribe( next: (response: any) => {
      if (response.status == 200) {
        this.listNotifications = response.data;
      }
    });
  }
  ngOnDestroy(): void {
    this.destroy$.next( value: true);
    this.destroy$.unsubscribe();
  }
}
```

En el html correspondiente al componente de List Notifications, creamos un maquetado básico para mostrar los datos que nos regrese el servicio con la diretiva ngFor.

```
<div class="container" style="margin-top: 20px">
  <table class="table table-striped">
    <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">Message</th>
      <th scope="col">Author</th>
      <th scope="col">Removed</th>
    </tr>
    </thead>
    <tbody>
    <tr *ngFor="let data of listNotifications">
      <th scope="row">{{ data.id }}</th>
      <td>
        {{ data.message }}
      </td>
      <td>{{ data.author }}</td>
      <td>{{ data.removed }}</td>
    </tr>
    </tbody>
  </table>
</div>
```

Desde el archivo add-notifications.component.ts, que le pertenece al componente Add Notifications, en el cual incializamos un objeto de tipo Notification, el servicio de notificacones para mandar a guardar los nuevos datos. El objeto de tipo notificación se inicializa como objeto vacío para relacionarlo con el formulario de la vista. Se crea iun método el envío de los datos

```
export class AddNotificationComponent implements OnInit {
  public newRecord: Notification = {} as Notification;
  constructor(public notificationService: NotificationsService) {
  }

  ngOnInit() {
  }

  public sendData() {
    this.newRecord.removed = false;
    this.notificationService.saveDataNotification(this.newRecord).subscribe( next: (response: any) => {

    });
  }

}
```

Se crea un formulario básico para el envío de datos, a los inputs correspondiente lo relacionamos con el objeto Notification con el atributo correspondiente y el botón esta al pendiente del click para ejecutar el método que manda la información.

```
<div class="container" style="margin-top: 10px">
  <form class="form">
    <div class="row">
        <label style="margin-right: 15px">Mensaje</label>
        <input type="text" class="form-control" id="mensaje" placeholder="Mensaje" [(ngModel)]="newRecord.message" [ngModelOptions]="{standalone: true}">
    </div>
    <div class="row">
        <label style="margin-right: 30px">Autor</label>
        <input type="text" class="form-control" id="autor" placeholder="Autor" [(ngModel)]="newRecord.author" [ngModelOptions]="{standalone: true}">
    </div>
    <div class="row">
      <button (click)="sendData()">Enviar</button>
    </div>
  </form>
</div>
```