Promesas

$q

- Interfaz para interactuar con un objeto que representa el resultado de una acción que se realiza de forma asíncrona

$q

- Esta acción puede o no ser terminada en cualquier momento dado en el tiempo

$q

- Es un servicio que nos ofrece dos APIs:
  - **Deferred**
  - **Promise**

La API Deferred

- Expone la instancia de la promesa asociada
- Así como la API para resolver su finalización y el estado de la tarea
- Nueva instancia: `var deferred = $q.defer()`

La API Deferred

- Métodos
  - **resolve**(valor): resuelve la promesa con dicho valor
  - **reject**(razón): rechaza la promesa debido a esa razón

La API Deferred

- Métodos
  - **notify**(valor): proporciona actualizaciones sobre el estado de la promesa en ejecución
  - Puede llamarse varias veces antes de que la promesa sea resuelta o rechazada

La API Deferred

```
function asyncGreet(name) {
  var deferred = $q.defer();
  setTimeout(function() {
    deferred.notify('About to greet ' + name + '.');
    if (okToGreet(name)) { deferred.resolve('Hi'+name +'!'); }
    else { deferred.reject('Greeting'+ name + 'not allowed.'); }
  }, 1000);
  return deferred.promise;
}
```

La API Promise

- Al llamar `var deferred = $q.defer()` se crea una promesa que puede obtenerse de la siguiente manera: `deferred.promise;`

La API Promise

- El propósito del objeto promesa es permitir tener acceso a los resultados de la tarea diferida cuando esta se complete

La API Promise

```
.then(successCallback, errorCallback, notifyCallback)
```

- llama de manera asíncrona a los **callbacks** de **error** o **success** tan pronto como el resultado esté disponible

La API Promise

- Los **callbacks** se llaman con un solo argumento: el resultado del **success** o la razón del **reject**

La API Promise

- Además, el **callback** de **notify** puede ser llamado cero o más veces para proporcionar información de progreso
- Notify no puede resolver o rechazar

La API Promise

```
.then(successCallback, errorCallback, notifyCallback)
```

- devuelve una nueva promesa que se resuelve o rechaza a través del valor de devuelto por el **callback success**, o el **callback error**

La API Promise

```
var greetingPromise = asyncGreet('Robin Hood');
greetingPromise.then(function(greeting) {
  alert('Success: ' + greeting);
}, function(reason) {
  alert('Failed: ' + reason);
}, function(update) {
  alert('Got notification: ' + update);
});
```

La API Promise

- Y si el callback devuelve una promesa, esta será resuelta con el valor devuelto por dicha promesa a través de **“Promise Chaining”**

Ejercicios

- Añade un módulo **promise**
  - Duplica el módulo API
  - Haz los cambios oportunos (index, router, navbar, renombra módulo API)

Ejercicios

- Ahora cambia la comunicación mediante eventos ($on y $broadcast) por promesas

Ejercicios

- Échale un vistazo al Back-end. Verás que hay 3 servicios que fuerzan un tipo de respuesta (**ok**, **error** y **delay**)
- También mira el nuevo módulo Angular **Messenger**, que muestra mensajes

Ejercicios

- Escribe el código necesario para poder efectuar llamadas a los 3 servicios del Back-end, haciendo uso del módulo messenger y las APIs deferred y promise

Ejercicios

- ¿Siguen funcionando los tests?