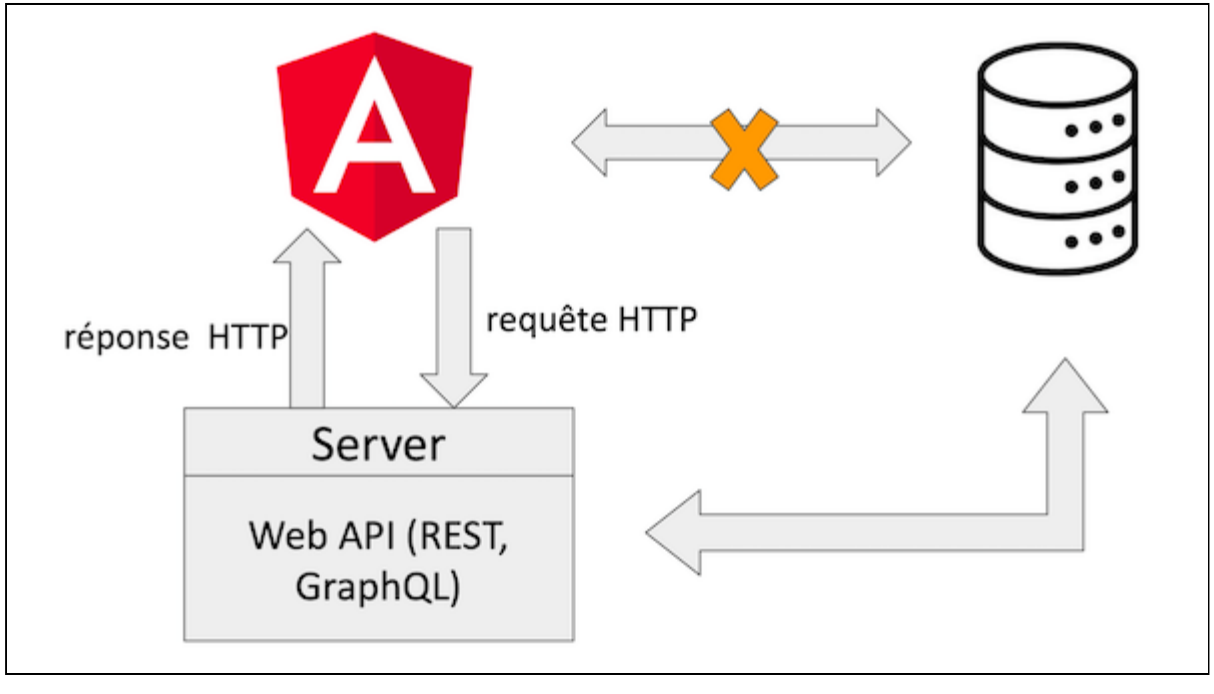
PARTIE 5: ENVOYER ET RECEVOIR DES DONNÉES PAR HTTP

ACCÈS AUX DONNÉES

- Une application avec Angular (*front-end*) ne se connecte pas à un SGDB pour accéder aux données comme peuvent le faire les **frameworks back-end** (ex: Laravel/Symfony).
- Côté client, ils peuvent utiliser les messages HTTP pour consulter et modifier des données stockées sur un serveur.



LE SERVICE `HttpClient`

- `HttpClient` est un service avec des méthodes correspondant aux différents verbes HTTP (GET, POST, DELETE...) pour effectuer tout type de requête HTTP. Chaque méthode retourne un **observable** `RxJS` pour gérer l'asynchrone.
- Par défaut `HttpClient` utilise `XMLHttpRequest` pour effectuer les requêtes. Pour utiliser l'API `fetch`, il faut le préciser dans la configuration.

ENREGISTRER/INJECTER LE SERVICE `HttpClient`

- Afin de pouvoir l'injecter où on en aura besoin, il faut l'enregistrer au niveau de l'application en ajoutant un *provider* au démarrage de l'application:

```
//app.config.ts
import { ApplicationConfig } from '@angular/core';
import { provideHttpClient } from '@angular/common/http';

export const appConfig: ApplicationConfig = {
  providers: [provideHttpClient(withFetch())]
};
```

- Comme les autres services, `HttpClient` peut être injecté.

```
import { HttpClient } from '@angular/common/http';
export class MyService {
  private readonly http = inject(HttpClient)
}
```

UTILISER `HttpClient`

- Par défaut, ce service considère que le format de données utilisé est le JSON. Il prend en charge la transformation entre JSON et objets javascript.
- Chaque méthode retourne un observable. Vous devrez vous abonner pour que la requête soit exécutée (`subscribe`, pipe `async`).
- On peut manipuler les réponses par le chaînage d'opérateurs `RxJS` (ex: map(), filter(), ...).

RÉCUPÉRER DES DONNÉES À PARTIR D'UNE API

- Utilisez la méthode `get` du service `HttpClient` pour récupérer des données structurées depuis une API.
- La méthode `get` retourne par défaut un observable typé des données transmises dans le *body* de la réponse HTTP.

```
this.http.get<type>(URL)
```

- Les méthodes de `HttpClient` peuvent toutes prendre en dernier paramètre un objet littéral pour ajouter des options (ex: ajout de paramètres à la requête)

MODIFIER DES DONNÉES À PARTIR D'UNE API

- les méthodes telles que `post` (création d'une nouvelle ressource) ou `put` (modification d'une ressource) nécessitent en deuxième paramètre les données à transmettre dans le *body* de la requête.

```
// data est une nouvelle donnée
// l'API utilisée retourne ici un observable vers
// une valeur du type de la nouvelle ressource créée
this.http.post<type>(URL, data)
```

MISE EN PLACE D'UNE "API"

Faire l'exercice correspondant sur Moodle

CRÉATION DE EXERCISESSERVICE (1/2)

- créer le service ExercisesService

```
ng g s exercises-service
```

```ts
export class ExercisesService {
  private readonly http = inject(HttpClient);
  private apiUrl = 'https://xxx.mockapi.io';

  // signal pour l'état global
  private readonly exercises = signal<ExerciseModel[]>([]);
  // signal en lecture seule pour les composants
  readonly tabExercises = this.exercises.asReadonly();
```

Pensez à enregistrer HttpClient au niveau de l'application en lui ajoutant le *provider*.

CRÉATION DE EXERCISESSERVICE (2/2)

```
constructor() {
  this.init();
}

private init(): void {
  this.http
    .get<ExerciseModel[]>(`${this.apiUrl}/exercises`)
    .subscribe(t => {
      this.exercises.set(t);
    });
}
```

- Le premier composant (ou service) qui fera appel à `ExercisesService` crée l'instance de celui-ci qui exécute la méthode `init`.
- Modifiez votre composant `Exercises` pour utiliser ce service au lieu du tableau actuellement déclaré en dur dans le composant.

GESTION DES ERREURS HTTP (1/2)

- Il est possible de gérer les erreurs HTTP avec l'opérateur `catchError` de `RxJS`
- Provoquez une erreur HTTP en créant une faute dans le point de terminaison utilisé pour l'API et regardez le résultat dans la console du navigateur

```
this.http
  .get<ExerciseModel[]>(`${this.apiUrl}/exercises`)
  .pipe(
    catchError(err => {
      return of([]);
    })
  )
  .subscribe...
```

- Provoquez de nouveau l'erreur HTTP et regardez le résultat dans la console du navigateur

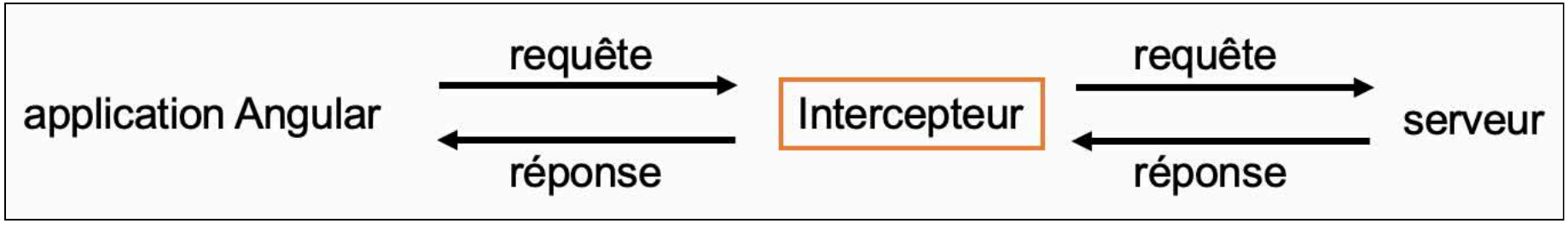
GESTION DES ERREURS HTTP (2/2)

- Ajoutez un effet de bord qui ajoute un message à l'application en cas d'erreur HTTP.

```
this.http
    .get<ExerciseModel[]>(`${this.apiUrl}/exercises`)
    .pipe(
      tap({
        error: () =>
         this.messagesService.add('Problème réseau ou serveur', 'warning')
      }),
      catchError(() => {
        return of([]);
      })
    )
    .subscribe...
```

LES INTERCEPTEURS

- Ce sont des fonctions qui permettent d'intercepter les requêtes HTTP pour effectuer des traitements centralisés pour toutes celles-ci:
  - soit avant que la requête soit envoyée au serveur (exemple: ajout d'un *token* d'authentification dans l'en-tête des requêtes)
  - soit lorsque la réponse arrive, avant la transmission du résultat au reste de l'application angular (exemple: gestion centralisée des erreurs)



CRÉATION D'UN INTERCEPTEUR

```
ng g interceptor httpError
```

```
export const httpErrorInterceptor: HttpInterceptorFn =
  (req: HttpRequest<unknown>, next: HttpHandlerFn):
    Observable<HttpEvent<unknown>> => {
      return next(req);
    };
```

- l'intercepteur ne fait que laisser passer la reqûete/réponse sans la modifier ici.
- Il la passe au prochain intercepteur (éventuel), au serveur (requête) ou à l'application Angular (réponse).

CRÉATION D'UN INTERCEPTEUR

```
export const httpErrorInterceptor: HttpInterceptorFn = (req, next) => {
  const messagesService = inject(MessagesService);
  return next(req).pipe(
    tap({
      error: (error: HttpErrorResponse) => {
        messagesService.add('Problème réseau ou côté serveur', 'warning');
        throw error;
      }
    })
  );
};
```

`error.status` peut permettre de différencier les messages d'erreur.

UTILISATION D'UN INTERCEPTEUR

- Il faut préciser à `HttpClient` que l'on souhaite utiliser un ou plusieurs intercepteurs.

```
//app.config.ts
provideHttpClient(withInterceptors([httpErrorInterceptor]), withFetch())
```

- Vous pouvez enlever la partie ajout d'un message d'alerte en cas d'erreur HTTP de la méthode `init` de votre service `ExercisesService`.

MÉTHODE `create` DANS LE SERVICE PAS À PAS (1/3)

Un nouvel exercice (pas encore enregistré côté API) n'a pas d'identifiant.

- A partir du fichier `ExerciseModel.ts`, exportez une interface `NewExerciseModel` qui a les mêmes propriétés que l'interface existante `ExerciseModel` à l'exception de l'`id`. `ExerciceModel` devient une interface dérivée de `NewExerciseModel`.

MÉTHODE `create` DANS LE SERVICE PAS À PAS (2/3)

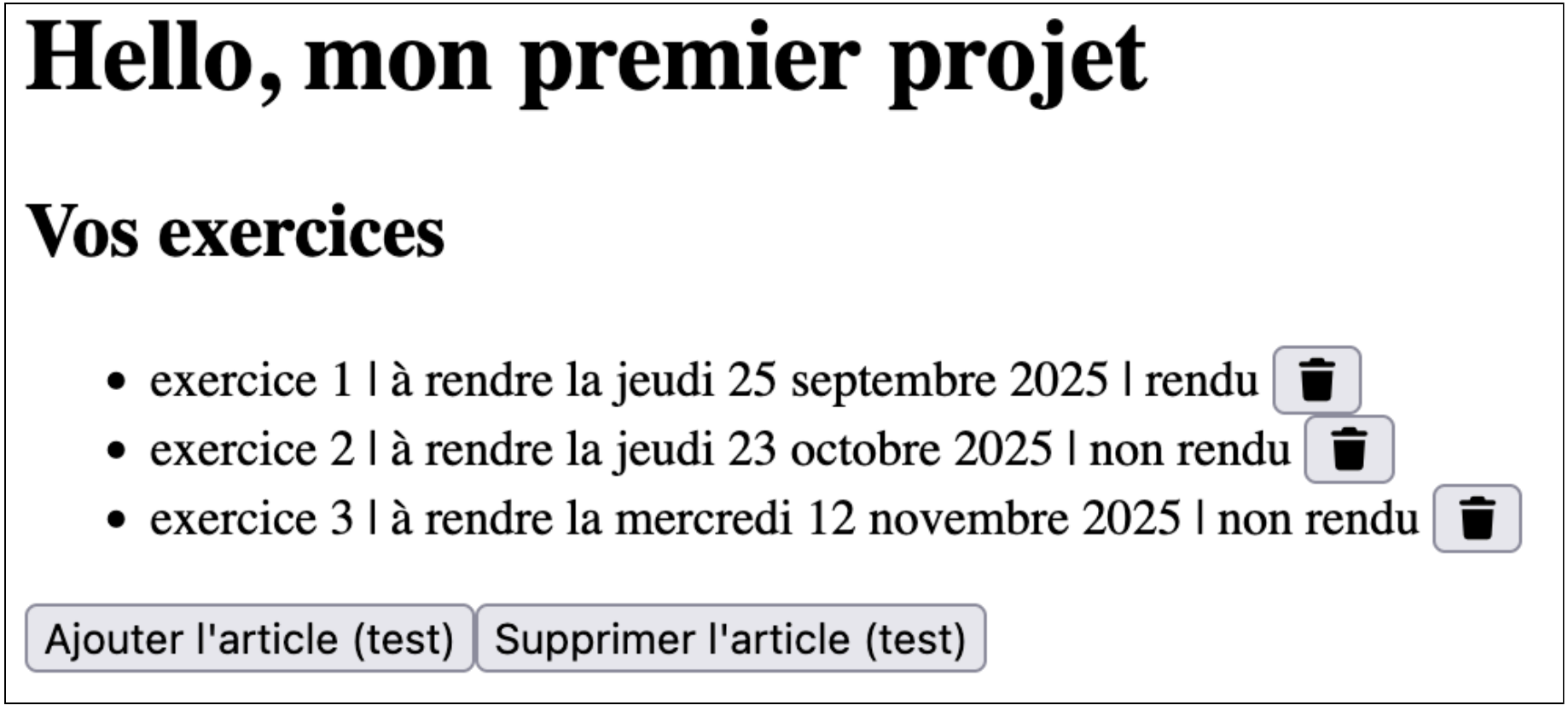
- Ecrivez la méthode `create` dans `ExercisesService` qui:
  - a en paramètre un `NewExerciceModel` (sans `id`).
  - fait une requête pour créer la ressource dans l'API.
  - change le tableau associé au signal `exercises` (même contenu + nouvel exercice avec `id` tel que créé par l'API).
  - ajoute un message de validation pour l'application qui dit que l'exercice a bien été ajouté.
  - gère les erreurs.

MÉTHODE `create` DANS LE SERVICE PAS À PAS (3/3)

- Testez avec un bouton de test dans le composant `Exercises` qui fait appel au service pour ajouter:

```
{
  title: 'écriture du service qui gère les requêtes HTTP',
  dueDate: new Date(),
  submitted: false
}
```

ENTRAINEMENT (1/3)



ENTRAINEMENT (2/3)

- Ajouter une méthode `delete` à votre service pour:
  - supprimer un exercice, à partir de son identifiant, et la tester avec :
    - un bouton de suppression au niveau de chaque item d'exercice,
    - un bouton de test comme vous l'avez fait pour `create` afin de tester la suppression d'un exercice inexistant.
- Faire en sorte que l'erreur dans le cas d'un code d'erreur 404 ajoute un message de *warning* pour dire que l'exercice n'existe plus (au lieu du problème réseau ou serveur).

ENTRAINEMENT (3/3)

Pour afficher des icônes dans notre application, on peut utiliser une collection comme Font Awesome.

```
npm install --save-exact @fortawesome/fontawesome-free@7.0.0
```

```
/* dans app/style.css */
@import '@fortawesome/fontawesome-free/css/all.css';
```

```
<!-- dans exercises.html -->
<button aria-label="Supprimer l'exercice">
  <i class="fa fa-trash" aria-hidden="true" title="Supprimer l'exercice"></i>
</button>
```

ALLER PLUS LOIN: EXERCICE

- Faire l'exercice disponible sur Moodle pour l'ajout d'un indicateur de chargement (*spinner*).