TP Angular - xkcd Reader (4H)

Sujet : Création d'une application web Angular liseuse des planches de la bande dessinée humouristique xkcd (https://xkcd.com/) à travers l'api json proposé par l'auteur.

Objectifs de ce TP :

- Utiliser Angular CLI
- Maîtriser les composants (directives, binding, pipe, ...)
- Maîtriser les services et les observables
- Utilisation du client http (appel API)
- Aborder les routes

Les documentations annexes sont fournies en lien à la fin du TP.

Pré-Requis : Installer votre poste

1. Téléchargez et installez nodejs - https://nodejs.org/, npm est fourni avec 😀.
2. Un IDE sera également nécessaire, de mon côté j'utilise Visual Studio Code (VSCode): https://code.visualstudio.com/.
3. Installez Angular CLI avec npm : `npm install -g @angular/cli`.

```
Annexes :
L'ensemble des commandes pour Angular CLI sont ici :
https://angular.io/cli.
Les commandes npm : https://docs.npmjs.com/cli/npm.
```

Votre première application Angular

1. Créez votre workspace (nouveau dossier) pour le tp et placez vous dans ce dernier

```
mkdir workspace
cd workspace
```

2. Initialisez votre application Angular avec l'angular CLI (Remplacez name par le nom de votre application) et complétez : Angular routing : yes, et sélectionnez CSS en stylesheet. La commande peut prendre quelques secondes / minutes.

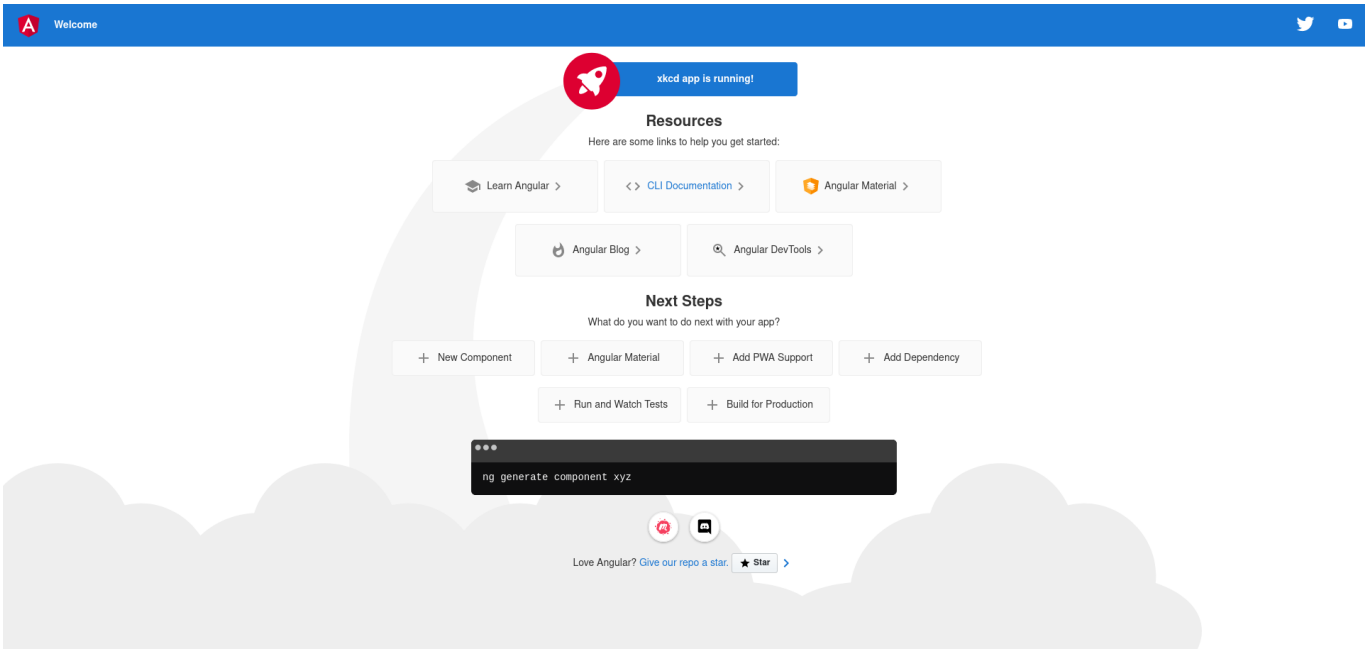
```
ng new <name>
```

3. Ouvrez le répertoire généré avec votre IDE. Avec VSCode, vous pouvez ouvrir un terminal intégré. Placez vous à l'intérieur du projet au niveau du package.json avec votre terminal. Lancez votre application.

```
npm start
```

4. Vous pouvez également la lancer avec l'angular CLI : `ng serve`
5. Ouvrez votre navigateur internet à l'adresse : http://localhost:4200.

Vous devez arriver sur la page d'accueil par défaut offerte par angular CLI.




```
/!\ Astuce : ng serve --open permet de lancer votre application et d'ouvrir votre navigateur directement.
Vous pouvez changer la commande `npm start` grâce au package.json.
```

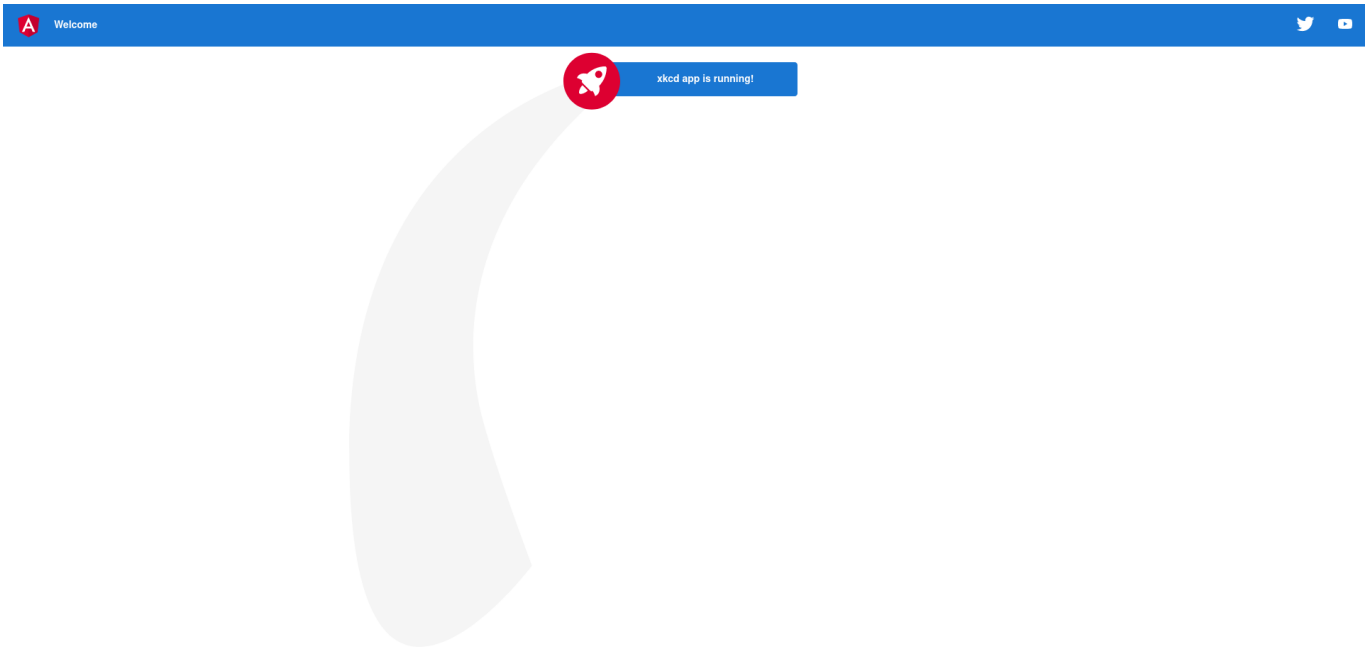
6. Ajoutez une nouvelle commande `"browse": "echo Run With Browser ; ng serve --open",` dans la section scripts du fichier package.json
7. Appelez la nouvelle commande browse avec npm.

```
Rappel du cours : le fichier package.json contient des commandes que pouvez ajouter / personnaliser à votre guise, mais attention seules certaines commandes peuvent être lancées sans le mot clé "run". La commande npm start est l'équivalent de la commande npm run start, si vous créez une nouvelle commande, vous devrez la lancer avec la commande npm run <ma commande>, et dans ce cas précis : npm run browse.
```

Vos premiers développements - Création de la liseuse

1. Modifiez la page d'accueil pour supprimer ce qui ne nous intéresse pas. Pour garder un peu de style, conservez les styles, le bandeau bleu et l'image en fond. Il suffit de supprimer tout ce qui suit `<!--`

`Resources -->` dans le fichier html du root component de l'application (indice : les sources de l'application se situent dans le dossier src/).

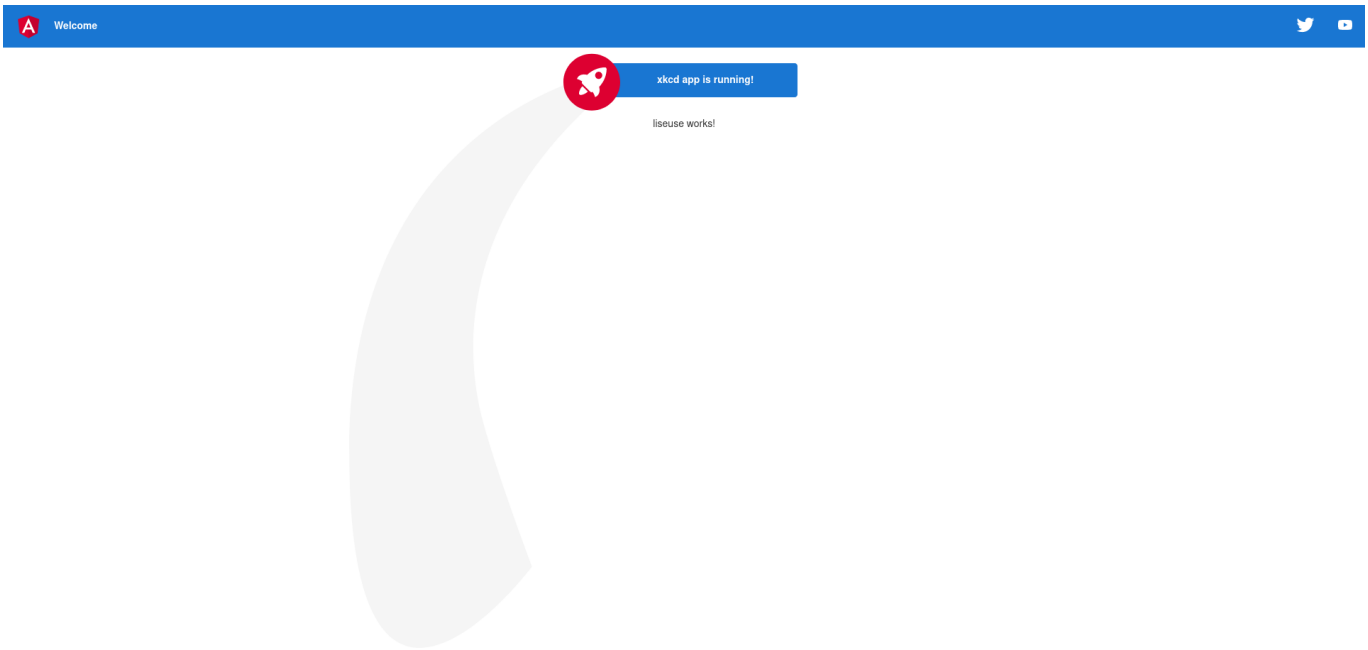


2. Créez un nouveau composant "components/liseuse" avec angular cli. Observez les 4 nouveaux fichiers créés et le fichier modifié (app.module.ts).

- Quel est le fichier template du composant ?
- Quel est le fichier class du composant ?
- Quel est le fichier de style du composant ?
- Quel est le fichier de tests du composant ?
- Pourquoi le fichier app.module.ts est modifié à la création d'un composant ?

```
/!\ Astuce : Rangez tout vos composants dans un répertoire "components",
générez vos composants en préfixant par le nom de dossier :
components/<name> (components/liseuse)
/!\ Astuce : Pour être plus rapide, ng generate component <name> est
équivalent à ng g c <name> ! (Idem pour les services : ng g s, ...)
```

3. Utilisez votre composant liseuse depuis la page d'accueil. Indice : Utilisez la directive (selecteur) du composant.



4. Créez un second composant "toolbar".
5. Déplacez le code html de la div toolbar de la vue app.component.html dans le template du nouveau composant **toolbar** (Code de la `<div>` se situant après `<!-- Toolbar -->`).
6. Appelez votre nouveau composant **toolbar** depuis la page d'accueil juste avant l'appel du composant liseuse.
7. Déplacez le code html de la div content de la vue app.component.html dans le template du composant **liseuse**, créez une propriété "title" dans la classe du composant **liseuse** avec pour valeur "xkcd".
8. Vérifiez le résultat. Les styles ne sont plus actifs, pourquoi ?

```
C'est dû au scope des composants. Un composant possède son propre scope et
n'est pas partagé, on parle d'encapsulation. (Possibilité de changer ces
contraintes grâce au ViewEncapsulation.) Il est possible de réunir les
styles principaux à un endroit stratégique pour éviter de redéfinir les
styles pour chaque composant.
```

9. Déplacez les styles de l'app.component.html dans le fichier styles.css (Sans les balises <style>), supprimez les balises styles devenues inutiles et le placeholder situé au dessus. Chaque composant possède ainsi ses surcharges css s'il souhaite personnalisé son affichage (fichier : `<name>.component.css`).

10. Revérifiez le résultat. Le code final de votre fichier app.component.html doit être réduit à deux lignes, celles appelant les deux composants.

```
<app-toolbar></app-toolbar>
<app-liseuse></app-liseuse>
```

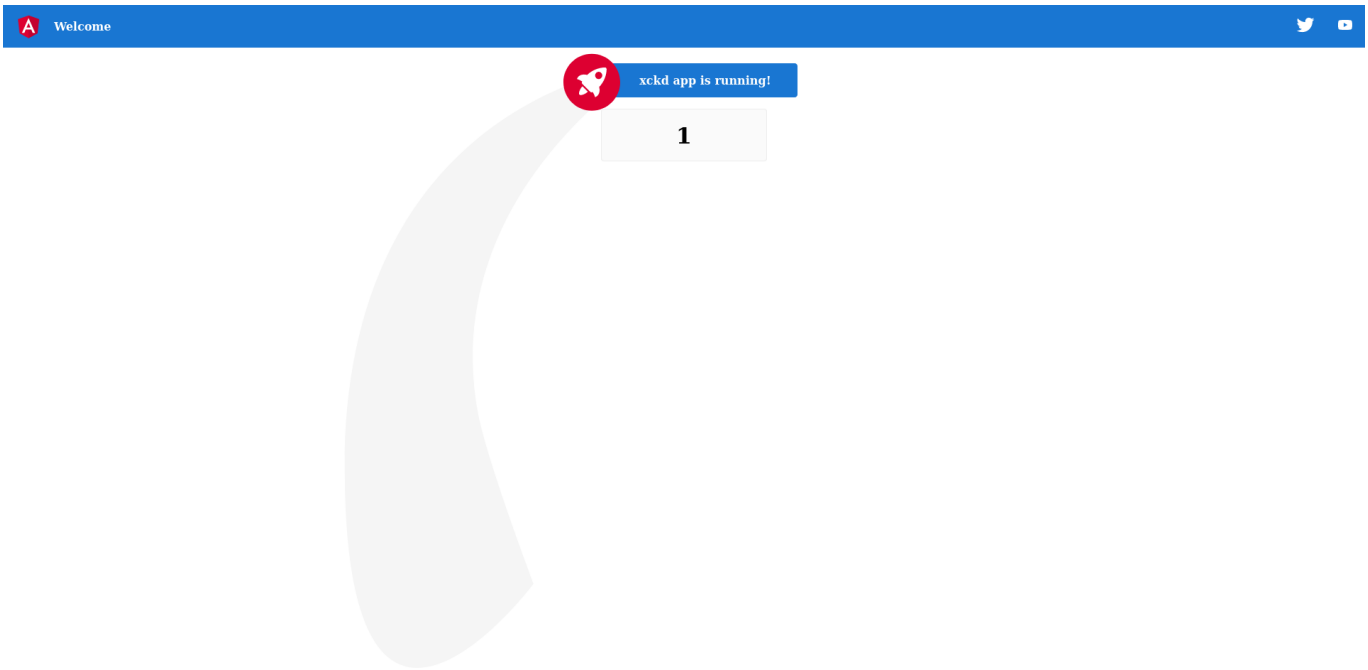
Création du composant Planche

La liseuse doit être capable de lire une planche, une par une, l'objectif ici est d'initialiser une liseuse avec une planche vide mais contenant un numéro de planche (1, 2, 3, ...).

1. Créez un nouveau composant "planche", ajoutez une propriété "numero" avec une valeur "1" dans la classe du composant et utilisez l'interpolation (indice: double moustache) dans le template du composant pour afficher la valeur de la propriété "numero" à l'intérieur d'un élément html ci-dessous.

```
<div class="card">
    <h1> CODE HERE </h1>
</div>
```

2. Utilisez le composant **planche** depuis le composant **liseuse**.



Création de la navigation entre planches

1. Créez une propriété "numeroEnCours" à la classe du composant **liseuse** représentant le numéro de planche en cours de lecture.

2. Créez 5 boutons html au template du composant **liseuse** : un pour revenir à la première planche, un précédent, un random, un suivant et un pour aller à la dernière planche.

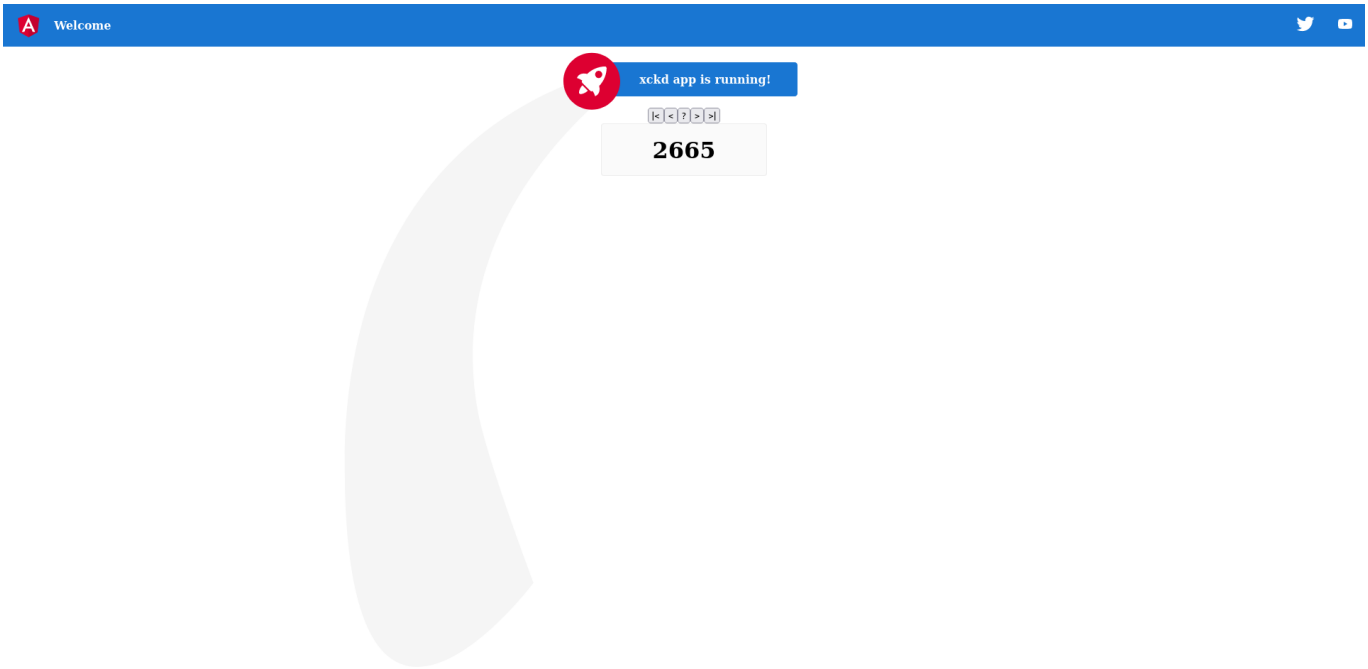
```
Astuce : La classe inline permet d'afficher vos boutons sur une seule
ligne.
<div class="inline">
  <button>1</button>
  <button>2</button>
  ...
</div>
```

3. Créez les 5 fonctions associés aux boutons respectifs :

```
first() // La fonction réinitialise la propriété "numeroEnCours" à 1.
back() // La fonction décrémente de 1 de la propriété "numeroEnCours"
seulement si cette dernière est supérieure à 1.
random() // La fonction associe une valeur aléatoire comprise entre
[1;2677] à la propriété "numeroEnCours".
next() // La fonction incrémente de 1 la propriété "numeroEnCours"
seulement si cette dernière est inférieure à 2677.
last() // La fonction réinitialise la propriété "numeroEnCours" à 2677.
```

4. Modifiez la propriété "numero" du composant **planche** afin que celle-ci soit passée par la valeur de la propriété "numeroEnCours" du composant **liseuse** (indice: Property Binding Parent -> Child).

5. Associez les fonctions de la classe aux événements "click" des boutons du template (indice: Event Binding). L'objectif étant que le numéro affiché par la planche doit être modifié en fonction des clics effectués sur les boutons.



Utilisons un modèle !

Imaginez un composant avec plus d'une cinquantaine d'attributs, les templates souhaitant utiliser ce composant devront binder chaque propriété ! Cela risque d'être lourd et de complexifié le code produit. Conseil : favorisez plutôt l'utilisation d'un modèle.

1. Créez un **modèle** "planche" et son constructeur avec les attributs numero de type number, image de type string, date de type date et title de type string.

```
/!\ Astuce : La création du modèle peut être réalisé avec angular cli : ng
g class models/planche
```

```
export class Planche {
    constructor(public numero: number = 1, public image: string =
'undefined', public date: Date = new Date(), public title: string =
'undefined') {}
}
```

2. Modifiez le composant **planche**: supprimez la propriété numero et créez la propriété "planche" de type "Planche" `@Input() planche: Planche = new Planche();`. Modifiez l'interpolation au niveau du template pour afficher l'attribut numero de l'objet planche. Modifiez le composant **liseuse**

(template et classe) afin de passer également par le nouveau modèle `Planche` en ajoutant une nouvelle propriété "plancheEnCours".

Contrainte:

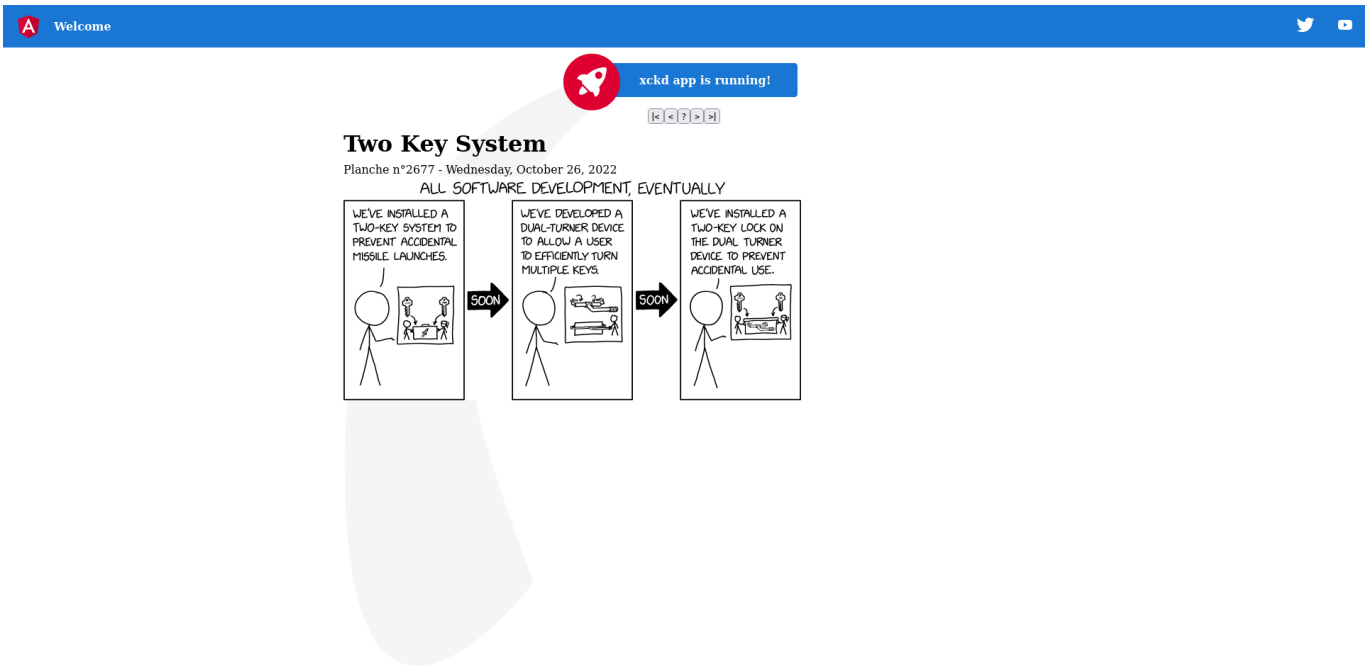
- Les fonctions de la classe du composant **liseuse** doivent faire appelle à une nouvelle fonction `getPlancheNumero(num: number): void` permettant de réinitialiser la propriété "plancheEnCours" (utilisation de `new Planche()`), elles ne doivent pas se contenter de modifier l'attribut de l'objet (ne pas faire : `plancheEnCours.numero = this.numeroEnCours`).

Améliorations du composant Planche

1. Modifiez le template du composant Planche afin d'afficher : le titre, la date, le numéro et et l'image (html via url) d'une planche. Contrainte : la date doit être affichée avec le jour de la semaine (Lundi, Mardi, ...), le mois et l'année (indice: pipe).

2. Initialisez les valeurs des propriétés du composant **liseuse** comme suit :

```
  numeroEnCours = 2677;
  plancheEnCours = new Planche(this.numeroEnCours,
"https://imgs.xkcd.com/comics/two_key_system.png", new Date(2022, 9, 26),
"Two Key System");
```



Votre premier Service (stateful !) avec l'utilisation du client HTTP.

L'objectif de cette étape est maintenant d'aller récupérer les données depuis l'API publiée par l'auteur d'xkcd afin d'afficher correctement les planches en fonction de la navigation de l'utilisateur. Les Observables seront manipulés dans cette partie.

1. Créez votre premier service avec angular cli : `ng g s services/xkcd`.

2. Configuration spécifique, l'api étant de la forme `https://xkcd.com/614/info.0.json`, 614 étant le numéro de planche :

- Créez un fichier `proxy.conf.json` sous le dossier `src/`,
- Copiez le contenu suivant dans ce fichier, cette configuration permettra de rediriger tous les appels local de l'application angular qui sont préfixés par "/api/" vers la cible :

```
{
    "/api/*": {
        "target": "https://xkcd.com/",
        "secure": false,
        "changeOrigin": true,
        "logLevel": "debug",
        "pathRewrite": {
            "^/api": ""
        }
    }
}
```

- Rajoutez la ligne de configuration suivante au niveau du fichier `angular.json` (~ ligne 62, voir cours slide 46 en cas de doute).

```
"proxyConfig": "src/proxy.conf.json"
```

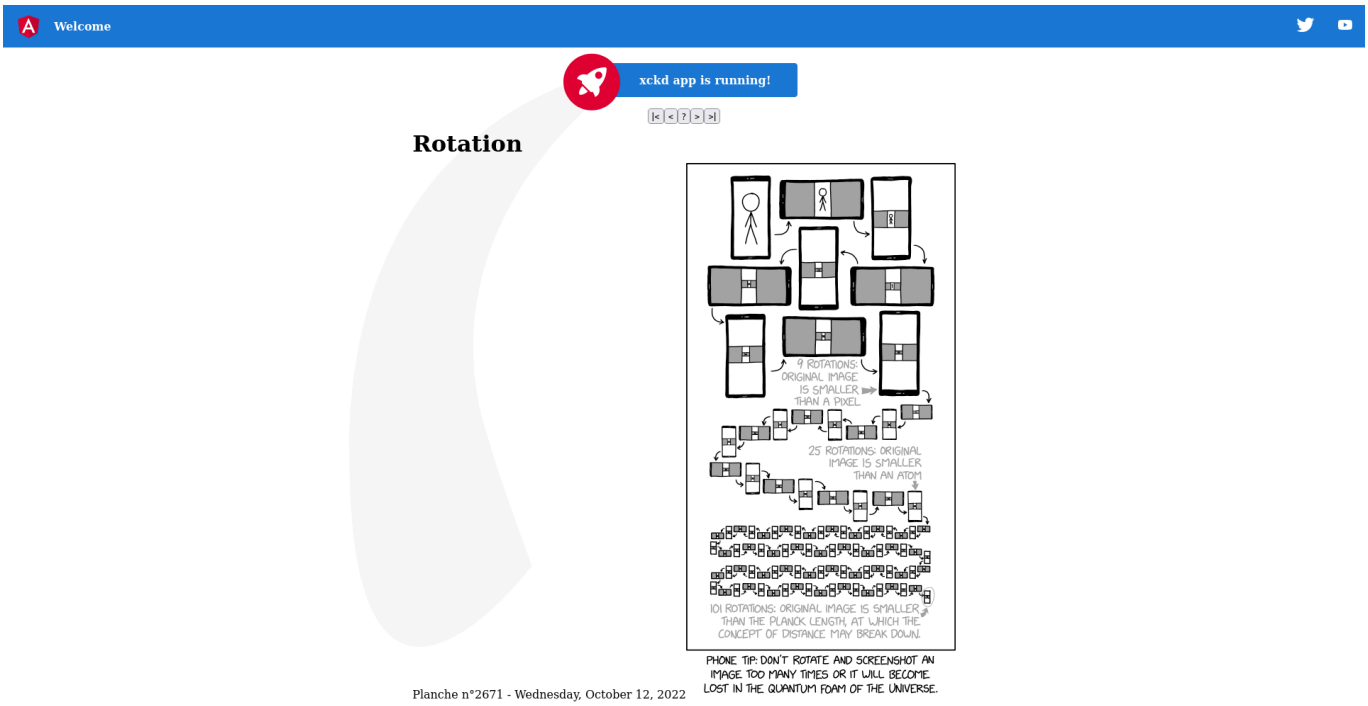
3. Relancez l'application avec `npm start` (ou browse) afin de prendre en compte cette nouvelle configuration qui n'est pas prise en compte par le live reload (CTRL+C pour stopper).

4. Créez une fonction "getPlancheNumero" au nouveau service permettant de retourner les données récupérées depuis l'url : `/api/<numero>/info.0.json` sour la forme d'un Observable de Planche. Analysez bien le retour de l'api pour compléter correctement l'objet **Planche** (new Planche(<...>)).

```
getPlancheNumero(numero: number): Observable<Planche>
```

```
Indices (Cours), ne pas oublier :
- l'import des modules dans l'app.module (HttpModule)
- l'injection de dépendance par constructeur pour l'http client
- la manipulation d'observable : utilisez l'opérateur pipe :
this.http.get(<url here>).pipe(map((planche: any) => new Planche(<code
here>)))
```

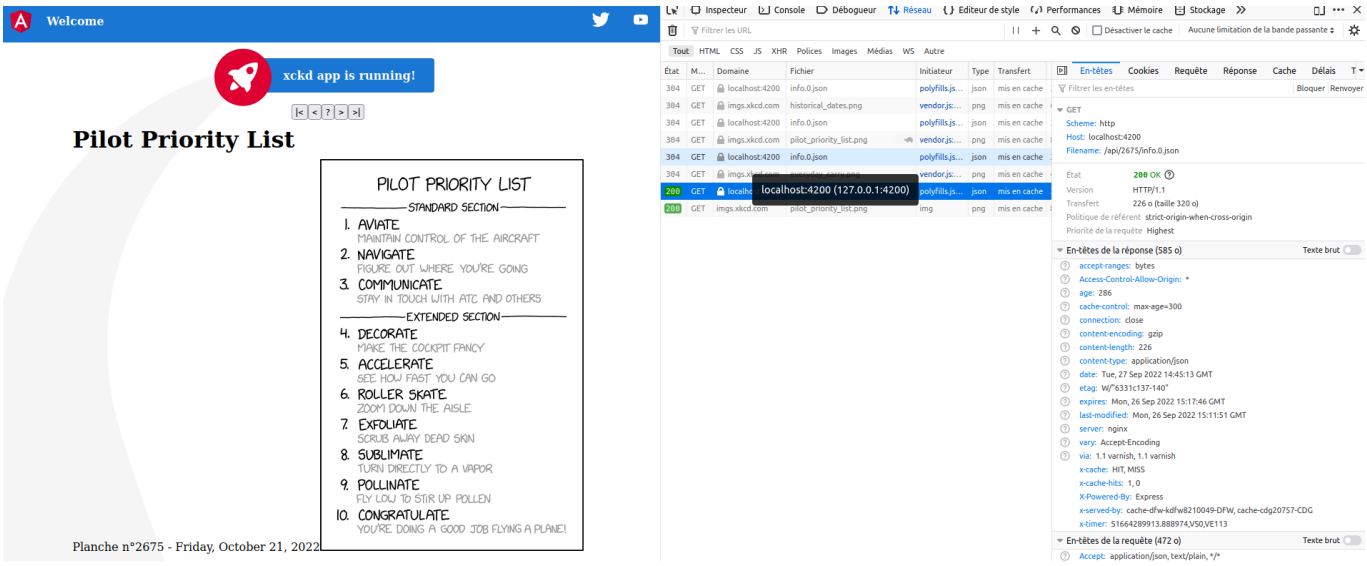
6. Appelez la fonction du service depuis le composant **liseuse** (modification de la fonction **getPlancheNumero** du composant) pour aller récupérer la planche en fonction de la navigation (indice: injection de dépendance et souscription d'observable).




```
Rappel du cours

Angular est un framework SPA (Single Page Application), c'est-à-dire qu'on
récupère l'application sur le navigateur une seule fois, on ne retélécharge
pas les éléments statiques (html/css/js/ressources/...) à chaque
navigation.
Et même si Angular est SPA, il est tout de même intelligent pour détruire
et construire les composants au bon moment (gestion du cycle de vie). Cela
permet de garder un bon niveau de performance. Une sorte de Garbage
Collector :-).

Pour conserver les états de certaines données, il faut alors utiliser un
service stateful, attention cet état n'est conservé qu'uniquement pendant
la navigation courante de l'utilisateur, si vous rafraîchissez votre onglet
ou fermez votre navigateur, l'état est perdu. Il existe plusieurs
techniques afin de conserver cet état dans le temps et habituellement il
faudrait créer une application backend afin de l'état persiste dans une
base de données par exemple.

Pour information, nous allons ici créer notre premier service stateful pour
conserver les données (en cache client côté navigateur) même si
généralement, on utilise une libraire supplémentaire ajoutée à Angular pour
gérer l'état global de l'application (librairie les plus connues : NGRX,
NGXS, Akita).
On préfère garder ainsi les services de manière sans état ("stateless").
```

7. Essayons de rendre "stateful" le service xkcd que nous venons de créer : ajoutez un cache au niveau du service pour éviter d'appeler une nouvelle fois l'api http json si l'appel a déjà été fait. Aidez-vous des outils développeurs de vos navigateurs pour vérifier que les appels ne sont pas fait à chaque fois dans l'onglet "Network/Réseau" (on parle ici de l'appel api json, pas de la resource image).

- Ajoutez une propriété planches : `planches: Map<number, Planche> = new Map<number, Planche>();` au niveau du service.
- Modifiez la fonction du service pour utiliser le cache si la planche a déjà été récupérée (indice pour le retour de l'observable au niveau de la fonction: utilisez l'opérateur `of` de la librairie rxjs), sinon appelez l'api http pour récupérer la planche et la mettre dans le cache.
- Votre navigateur étant (trop) intelligent, il utilise son propre cache grâce au retour HTTP 304 - Not Modified. L'objectif de cette étape étant qu'il ne doit **PAS** y avoir deux appels avec un retour HTTP pour une même planche comme ci-dessous (surligné en bleu), la planche 2675 à été récupéré deux fois.




```
Astuce : Vous pouvez utiliser l'opérateur `!` (prononcé "bang") pour
indiquer au compilateur statique qu'il y a forcément une donnée à
l'intérieur du cache pour éviter un problème de typage. À utiliser avec
attention, après une vérification que la donnée est bien présente par
exemple.

Exemple :
  this.planches.get(numero) est une donnée de type Planche | Undefined
  this.planches.get(numero)! est une donnée de type Planche
```

Votre première groute

1. Créez un composant représentant une page 404 simple (appelez-le page-not-found)

```
<div class="content" role="main">
  <h1>404 Not found.</h1>
</div>
```

2. Modifiez le template de votre RootComponent (`app.component.html`) pour qu'il ressemble au code ci dessous : (il suffit de supprimer l'appel au composant album et d'ajouter le router-outlet).

```
<app-toolbar></app-toolbar>
<router-outlet></router-outlet>
```

```
Rappel du cours :

Le router-outlet est un élément qu'Angular change dynamiquement en fonction
de l'état du routeur ("Router"), ce dernier étant un service de navigation
entre les templates et proposant une manipulation via URL (ou non).

Rappelez-vous, en intialisant l'application, nous avons choisi "yes" à la
question de l'activation d'"Angular routing".  Angular CLI a donc créer et
importer un fichier `app-routing.module.ts` déjà tout pré-configuré :-).
```

3. Créez deux routes (voir cours):

- une route par défaut redirigeant vers votre nouvelle page 404 (indice: wildcards)
- une route /liseuse/ redirigeant vers votre composant liseuse

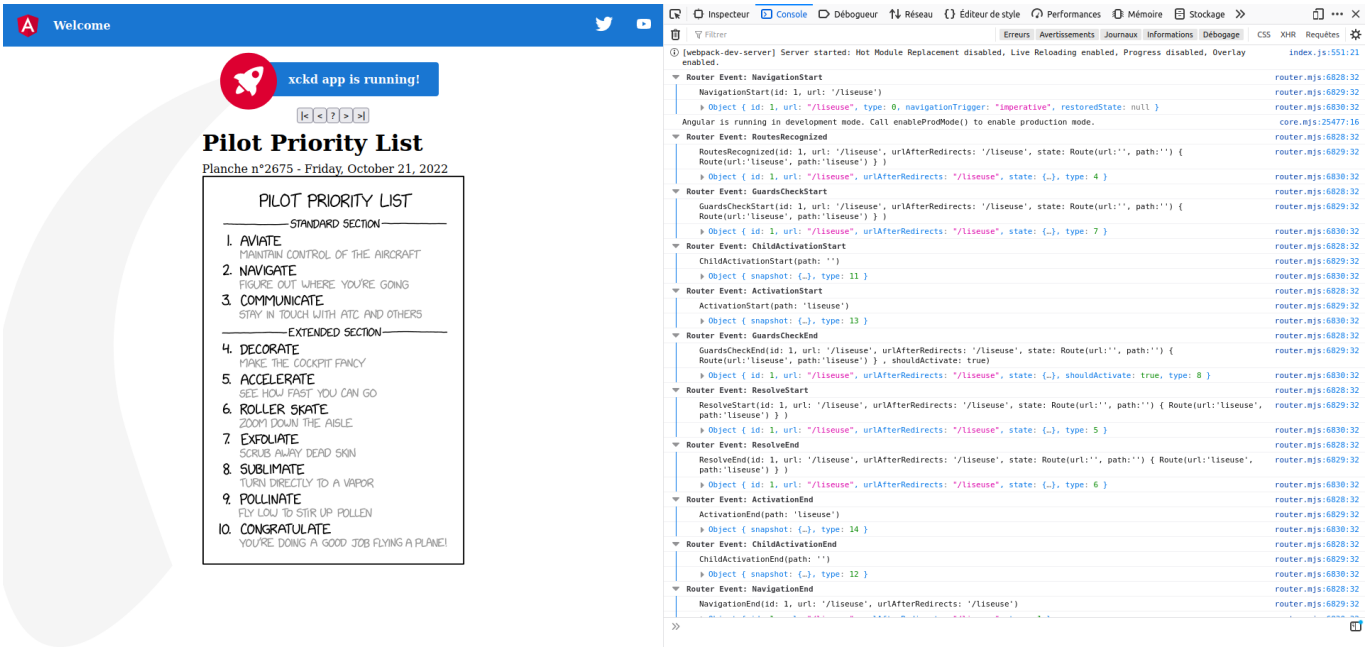
```
/!\ Indice : celà se passe au niveau du app-routing.module.ts
```

4. Rajoutez deux raccourcis (élément html <a>) au niveau du composant **toolbar** pour rediriger un raccourci vers l'album et l'autre vers votre page 404 en utilisant la fonctionnalité de routing Angular (`<a routerLink="...">...</a>`).

5. Ajoutez une route vide (home) '' pour rediriger vers la page liseuse (utilisez `redirectTo`).

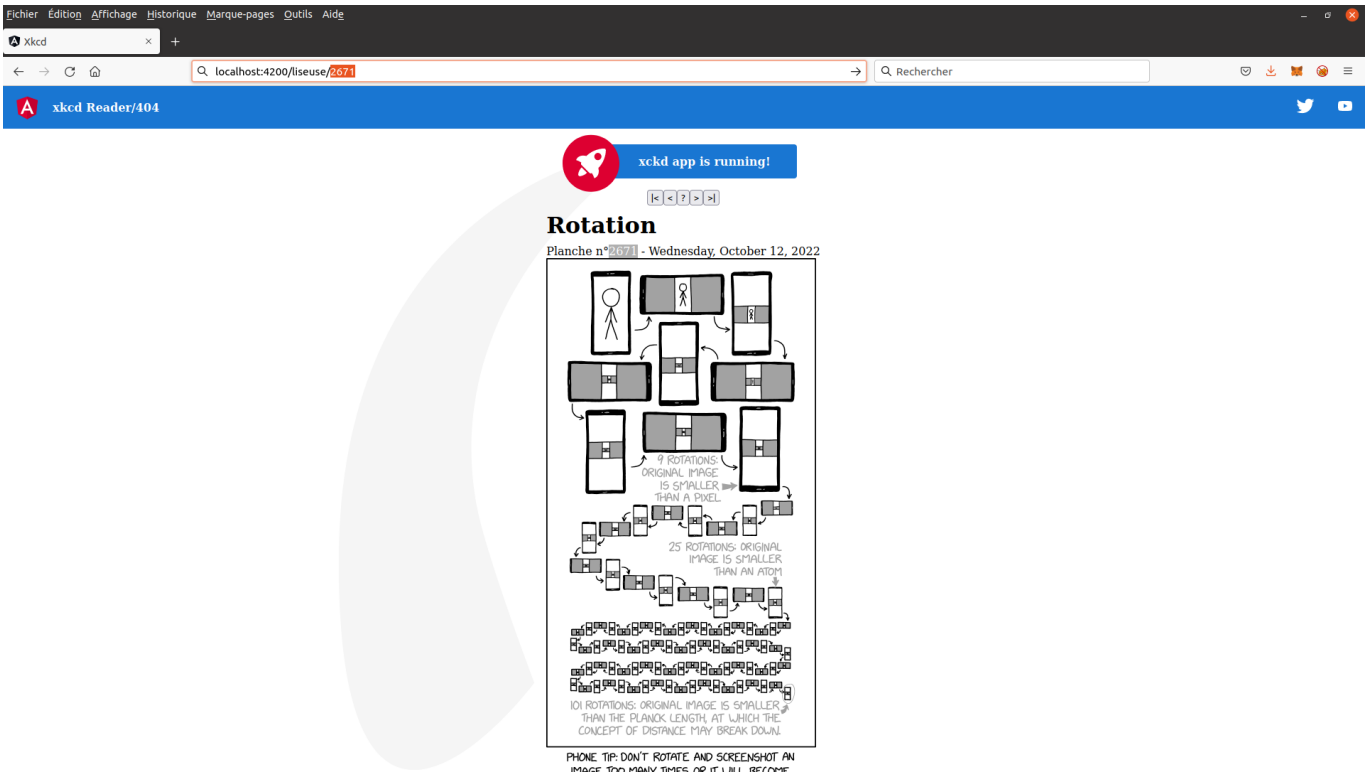
```
/!\ Astuce : en développement vous pouvez ajouter la variable enableTracing
au niveau de l'import des routes pour débuguer vos routes avec les outils
développeurs de votre navigateur préféré, attention à ne pas l'activer en
production !
Utiliser la variable d'environnement pour savoir si vous êtes en production
;-) !
```

```
@NgModule({
  imports: [
    RouterModule.forRoot(routes, { enableTracing: !environment.production }),
  ],
  exports: [RouterModule],
})
```



6. Modifiez la route 'liseuse' pour qu'elle puisse prendre en compte un paramètre : le numéro de planche et charger la planche correspondante en modifiant le composant **liseuse**, exemple : http://localhost:4200/liseuse/2671

```
Indices :
  - Injectez le service ActivatedRoute au constructeur du composant liseuse pour récupérer la valeur du paramètre.
  - Utilisez le cycle de vie et les fonctions proposées du composant liseuse pour charger la planche au bon moment (à l'initialisation).
```



7. Utilisez les fonctions de navigation (précédente, suivante, first, last, random), que constatez-vous au niveau de l'url ? Pourquoi ce comportement à lieu ?

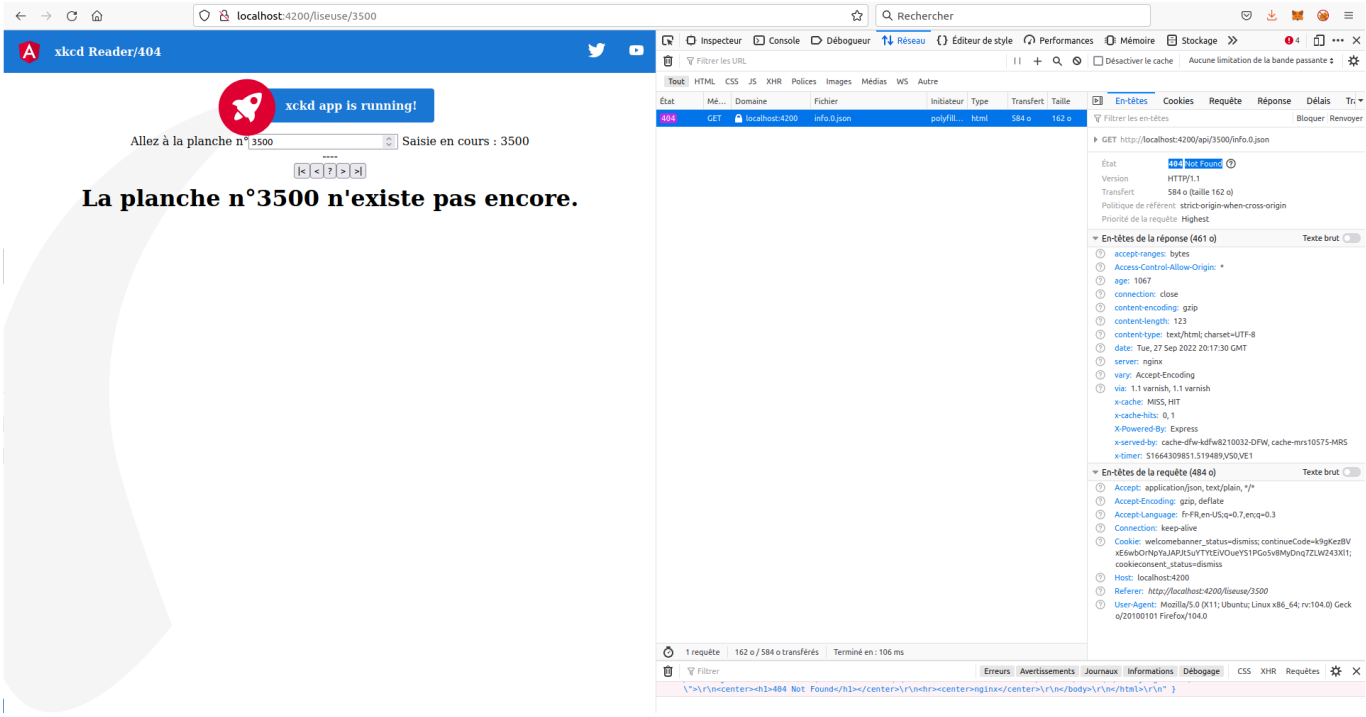
8. Modifiez les fonctions de navigation du composant **liseuse** pour corriger ce comportement (indice: injectez le service Router au constructeur du composant). Attention, il est possible que vous ayez affaire à une promesse plutôt qu'un observable 😃.

Avez-vous toujours la Form ?

1. Ajoutez une barre de saisie de numéro de planche à la navigation pour l'utilisateur. La barre de saisie ne doit autoriser que des chiffres. Redigirez l'utilisateur vers la planche lorsque celui-ci appuie sur la touche "Entrée" de son clavier (indice: element html input text et event binding). Contraintes :

- ajoutez une nouvelle propriété "numeroSaisi" au composant **liseuse**,
- utilisez le two-way binding avec la directive ngModel pour la valeur de l'élément input (indice: Le module FormsModule de la librairie @angular/forms sera très certainement utile 😃)
- affichez la saisie en cours juste à droite de la barre de saisie en utilisant l'interpolation, la donnée doit être mise à jour automatiquement.

2. Affichez une erreur à l'utilisateur s'il saisi une planche qui n'existe pas encore (exemple: 3500). Contrainte : le service **liseuse** doit faire appel à l'api json (et donc recevoir un retour HTTP 404), votre code doit réagir en conséquence (idée : utiliser la directive ngIf au niveau du template du composant **liseuse** ?).



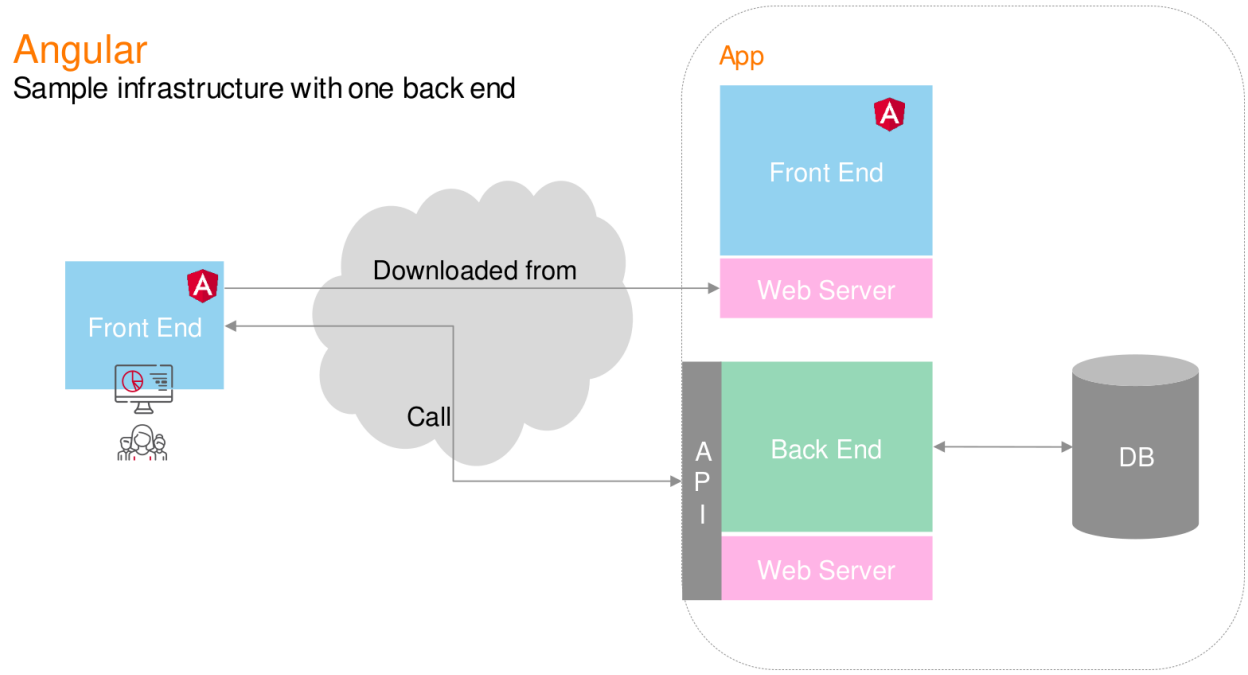
/!\ Note : Rappelez-vous que le code du front (l'application web Angular) est exécuté par le navigateur de votre PC. On aurait pu bloquer l'utilisateur à ne pas pouvoir saisir un numéro supérieur à 2677 au niveau de l'application Angular elle-même afin d'éviter un appel vers l'api json.

Mais il est très facile de bypasser ces protections en modifiant, par exemple, le DOM directement depuis votre navigateur grâce à la console développeur.

D'un point de vue sécurité, rappelez-vous qu'il faut toujours protéger les données au niveau du serveur, le code du backend étant exécuté sur ce dernier!

Dans notre cas, l'impact était nul car l'api nous a renvoyé un 404. Imaginez-vous développer une application bancaire qui permet de voir le montant et d'exécuter des opérations bancaires d'un compte, le compte possède un numéro et appartient à une personne, si les contrôles de propriété du compte ne sont fait qu'au niveau de l'application web et qu'aucun contrôle n'est fait au niveau serveur, alors il sera très aisé pour une personne débrouillarde de voir n'importe quel compte et d'effectuer n'importe quelle opération de son choix. Par ici la monnaie :-) !

Protégez vos apis backend en controlant les inputs utilisateurs ;-) ! Les contrôles au niveau du frontend ne sont utiles que pour le confort de l'utilisateur et l'ergonomie de l'application (Ceintures et Bretelles).

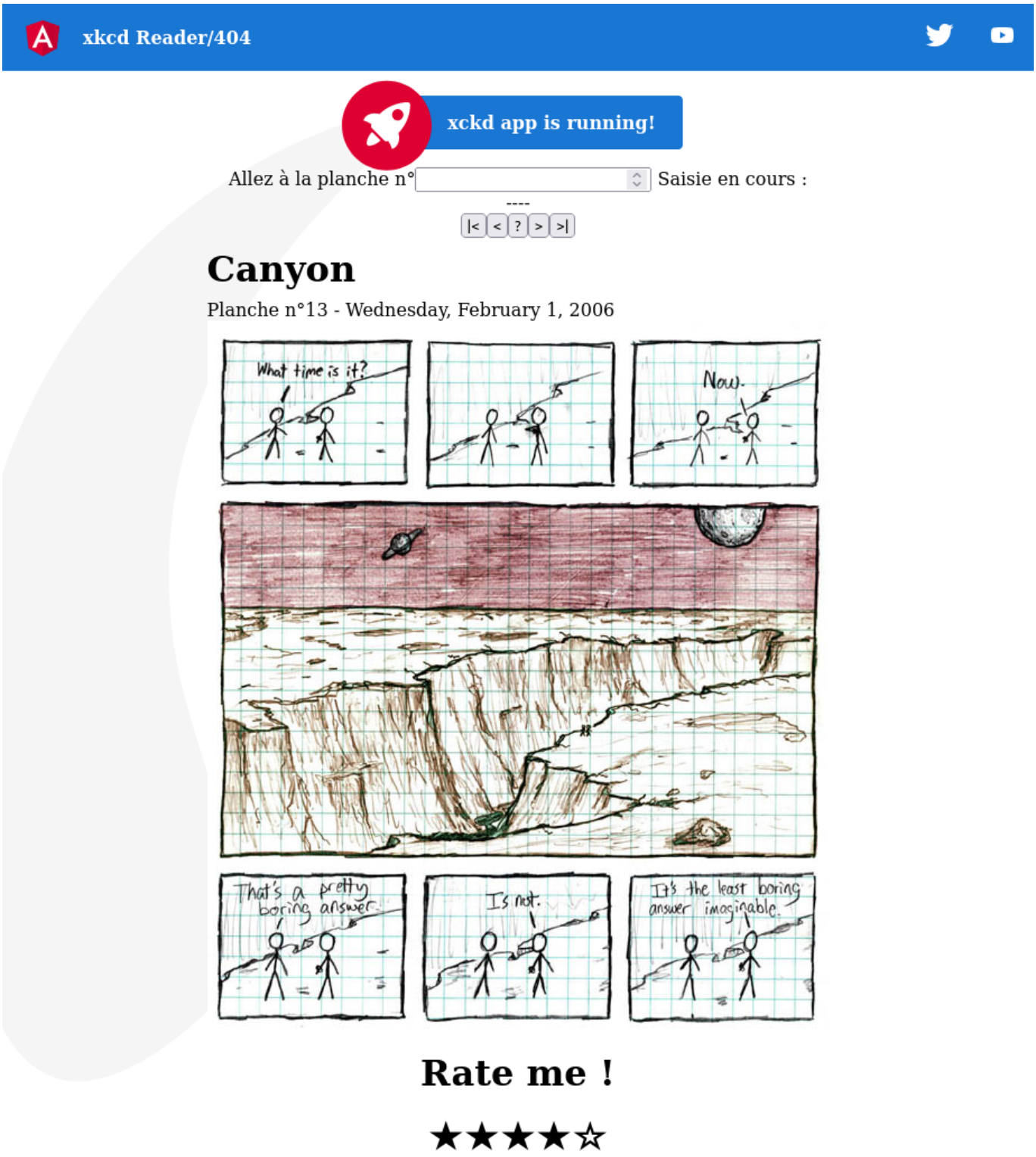


Endless ~~loop~~ TP

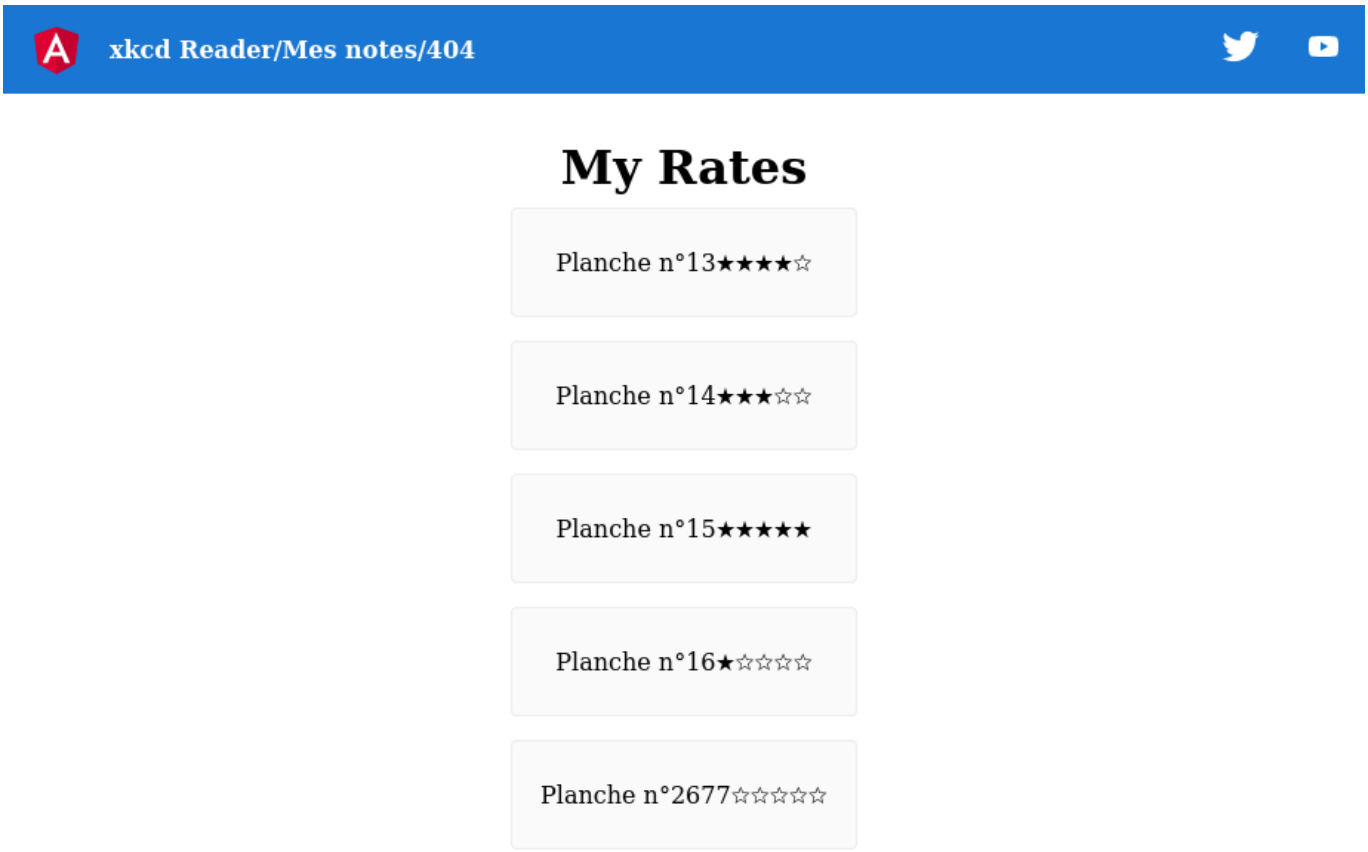
```
J'ai tenté de calibrer le sujet de TP pour une durée de 4 heures jusqu'à
cette partie. Cependant, c'est toujours difficile d'estimer une première
fois la durée du sujet, faites-moi un petit signe quand vous êtes arrivés
ici afin que je me rende compte du temps pour m'améliorer la prochaine
fois.

Et n'hésitez pas avant d'entamer cette dernière partie de me montrer vos
réalisations et d'échanger vos avis et remarques sur le contenu du TP.
Merci :-) !
```

```
L'objectif de cette dernière partie est de proposer un système de "rating"
de 1 à 5 étoiles à l'utilisateur des planches qui ont été lues.

Cette partie est volontairement moins détaillée dans le but de libérer
votre imagination.
```

1. Modifiez l'interface pour proposer à l'utilisateur de noter la planche de 1 à 5 étoiles.



2. Créez un mécanisme sur l'application Angular permettant de conserver les notes de l'utilisateur.
3. Créez une nouvelle page (nouveau composant + nouvelle route) permettant à l'utilisateur de consulter les notes des planches qu'il a lues (indice: ngFor).



4. Mettez quelques notes, consultez la page des notes, puis rafraîchissez l'application (CTRL+R ou CTRL+F5 ou via un nouvel onglet), que constatez-vous ? Pourquoi ? Quelle serait la solution pour remédier à ce problème (indice: c'est expliqué dans le TP 😃 !) ?

```
Merci, j'espère que vous avez apprécié le TP pour une première initiation à
la technologie Angular.
Pour aller plus loin, consultez les guides et documentation officielles, il
existe de super docs et d'autres tutoriels ! Elles seront vos meilleures
amies et seront à jour.
```

Documentations utiles :

- https://angular.io/docs, Angular,
- https://angular.io/cli, Angular CLI,
- https://angular.io/guide/observables, Angular & les Observables,
- https://rxjs.dev, RxJS librairie pour les Observables,
- https://rxjs.dev/guide/operators, Opérateurs pour les Observables,
- https://docs.npmjs.com/cli/v8, npm CLI,
- https://www.typescriptlang.org/, TypeScript
- https://www.ecma-international.org/publications-and-standards/standards/ecma-262/, Standard ECMAScript,
- https://www.w3schools.com/js/, W3 Schools JavaScript Tutorial,
- https://www.w3schools.com/html/, W3 Schools HTML Tutorial
- https://www.w3schools.com/typescript/, W3 Schools TypeScript Tutorial

Et Merci à https://xkcd.com/ !

The End.