- 10. Router
  - Introduction
  - Mise en oeuvre
    - Définition des routes
    - Définition du router dans la template.
    - Lazy loading
    - Navigation via le controleur
    - Navigation via la template HTML
    - Envoi et récupération de paramètres
    - Contrôle d'accès à une URL
  - Exercice 1
    - Présentation
    - Etapes de création
  - Exercice 2
    - Présentation
    - Etapes de création

10. Router

Introduction

Le Router Angular met à disposition les principes de navigation classiques entre différentes vues d'une application, à travers la modification dynamiques des URLs du navigateur Web :

- chargement de l'application dans un état donné par saisi d'une URL. L'URL construite est suffisante à la régénération de l'application dans l'état attendu. Exemple : `http://www.mondomaine.com/mon-app-mon-site/mon-etat/`
- navigation dans l'application par l'utilisation des boutons de navigation dans l'historique (précédent, suivant).

Exemple : `http://www.mondomaine.com/mon-app-mon-site/mon-etat-un/ > http://www.mondomaine.com/mon-app-mon-site/mon-etat-deux/`

Au dela de l'état de l'application défini statiquement dans l'URL, il est tout à fait possible de définir des variables dynamiques dans l'URL, permettant par exemple de charger des informations. Ces variables sont accessibles au runtime pour exécuter par exemple un service web permettant de récupérer les informations nécessaires à la remise en état de données dynamique.

Exemple : `http://www.mondomaine.com/mon-app-mon-site/utilisateur/details/120`

Il met également un certains nombre de mécanismes permettant de faire transiter des données entre vue, de contrôler l'accès à des URLs, etc.

Documentation officielle

Mise en oeuvre

Définition des routes

Pour pouvoir exploiter le Router Angular, vous devez importer les modules suivants :

```
import { RouterModule, Routes } from "@angular/router";
```

Il est nécessaire ensuite de définir la liste des `routes` disponibles pour l'application.

Pour une route, voici les paramètres disponibles :

- `path` : défini le chemin qui sera ajouté à l'URL de l'application,
- `component` : défini le composant associé et instancié lors de la navigation sur l'URL,
- `loadChildren` : défini le module associé et instancié lors de la navigation sur l'URL (chargé en Lazy Loading),
- `children` : liste des sous URLs du `path` initial,
- `canLoad, canActivate` : protège l'accès à une URL par un contrôle (si un path n'est disponible qu'après une authentification par exemple),
- `data` : liste de données pouvant être passées au composant associé au path,
- `outlet` : spécifie la route outlet dans laquelle sera chargée le composant (s'il y en a plusieurs),
- `pathMatch` :
  - `full` : L'url doit totalement matcher avec le `path` défini,
  - `prefix` : L'url doit partiellement matcher avec le `path` défini, notemment lorsque des paramètres sont passés dans l'URL.

```
const routes: Routes = [
  { path: 'comptes', component: CompteListComponent },
  { path: 'compte/:id', component: CompteDetailComponent },
```

Il est ensuite nécessaire d'injecter ces routes dans le `RouterModule` . Conventionnelement, on définira un module dédié au router, que l'on importera ensuite via `imports` dans le module principal.

```
@NgModule({
  imports: [RouterModule.forRoot(routes, { useHash: true })],
  exports: [RouterModule],
  providers: [],
})
export class AppRoutingModule {}
```

Vous noterez que la méthode `forRoot` prend en paramètre :

- `routes` : liste des routes définies dans la constante,
- paramètres compélentaires :
  - `useHash` : défini si le caractère `#` est utilisé pour séparer l'URL principale avec les URL du router,
  - `preloadingStrategy` : défini la stratégie de préchargement des modules chargé initialement en Lazy Loading,
  - `enableTracing` : active les logs du Router lors de la navigation

Définition du router dans la template.

Dans la template de l'application ou d'un composant quelconque, il est nécessaire de définir le `router-outlet`, qui sera en quelque sorte le conteneur du router, où seront déposés les composants/modules instanciés lors de la navigation.

```
<div id="app-title">
  <p>Hello Application</p>
</div>
<hr />
<router-outlet></router-outlet>
```

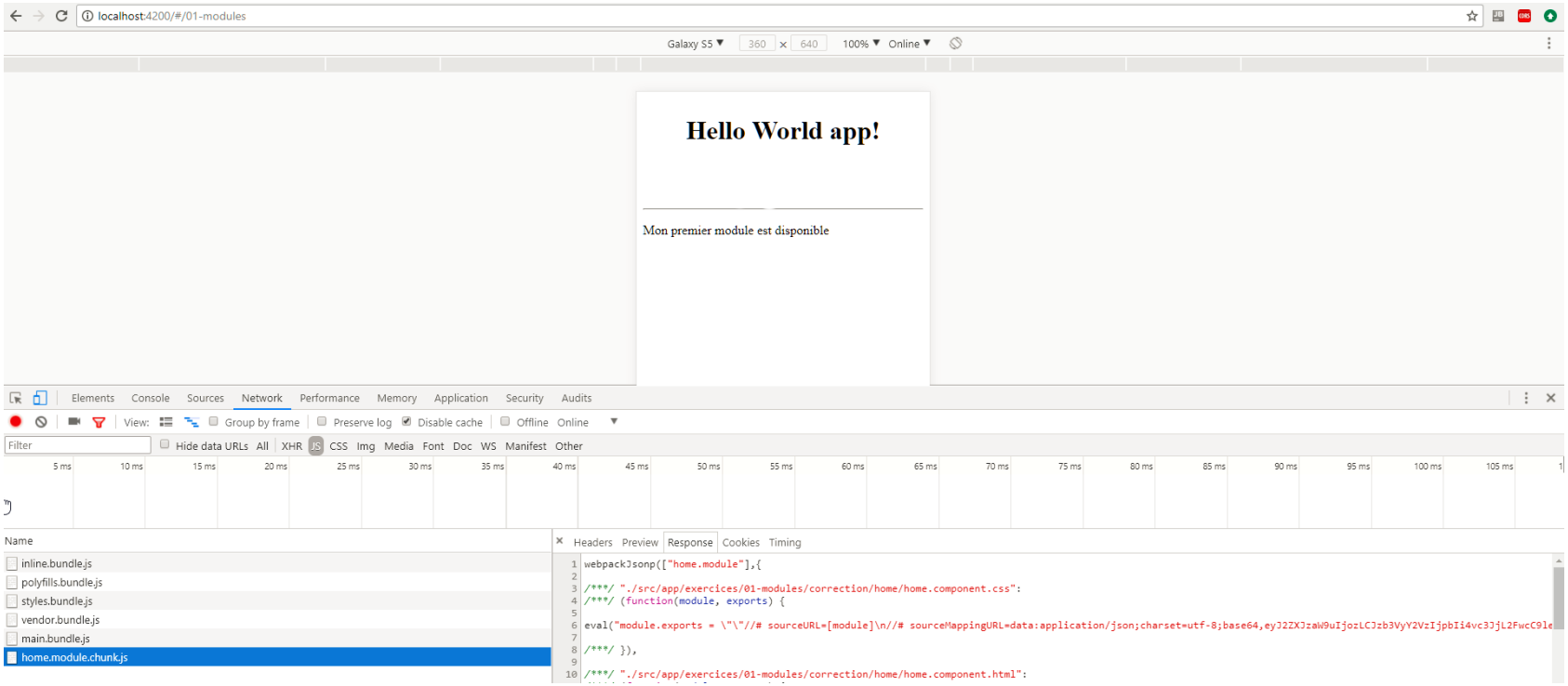
Lazy loading

Il est possible de charger à la volée des modules lors de la navigation via le router Angular. Cette technique permet de charger des 'parties' de l'application, uniquement quand celles-ci sont explicitement demandées par l'utilisateur. Cela peut être intéressant dans le cas où l'application est lourde, et qu'il est souhaitable de charger des briques applicatives optionnelles, uniquement à la demande.

Pour cela on utilisera cette synthaxe :

```
const routes: Routes = [
  { path: "**", redirectTo: excercicePath, pathMatch: "full" },
  {
    path: "home",
    loadChildren: import("../home/home.module").then(m => m.HomeModule)
  }
```

Cette déclaration va permettre de ne pas embarquer `HomeModule` dans la compilation du projet, mais de le charger au runtime.



Navigation via le controleur

Il est possible d'exploiter le `Router` pour naviguer depuis le contrôleur. Pour cela, il est nécessaire d'injecter dans le constructeur du contrôleur le `Router`

le constructeur du contrôleur le `Router` :

```
constructor(private router: Router, private activateRoute: ActivatedRoute) {}
```

Puis dans une méthode de naviguer à l'aide de l'instance avec la méthode `navigate` .

```
this.router.navigate(["page-xxx"], {});
```

Navigation via la template HTML

Il est possible d'exploiter directement le `Router` dans la template HTML. Pour cela on utilisera l'annotation suivante :

```
<a routerLink="/contact" routerLinkActive="active">Aide</a>
```

Envoi et récupération de paramètres

Il est possible de faire transiter des paramètres dans le router, et de les récupérer dans le contrôleur instancié.

Pour l'envoie, on utilisera cette synthaxe :

```
this.router.navigate(["/compte/:id", { id: compteId }]);
```

La récupération du paramètre se fera par injection du router `Router` .

```
constructor(private router: Router, private activateRoute: ActivatedRoute) {}

ngOnInit() {
    this.route.queryParams.subscribe((params) => {
```

```
    })
}
```

Contrôle d'accès à une URL

Il est possible de limiter un accès à une URL, par un contrôle en amont dans le router. Cela peut permettre par exemple de limiter l'accès uniquement aux utilisateurs authentifiés, et le cas échéant de les rediriger vers la page d'authentification.

Pour cela, on utilisera la propriété `CanLoad` pour un module chargé en Lazy Loading, ou `CanActivate` pour un composant, dans la définition de la route. On spéfifiera alors l' `AuthGuard` en charge du contrôle de l'accès, en implémetant la méthode `canActivate` . Cette méthode doit renvoyer un booléen définissant si l'accès est authorisé. Si le contrôle est asynchrone (demande via un web service par exemple), la méthode doit renvoyer un `Observable` .

```
 {
    path: "mes-comptes",
    component: MesComptesComponent,
    canActivate: [AuthGuard]
  },
```

```
import { CanActivate } from "@angular/router";

export class AuthGuard implements CanActivate {
  canActivate() {
    console.log("AuthGuard");
    return true;
  }
}
```

Exercice 1

Présentation

Application Météo.

L'objectif de cet excercice est de créer une vue qui propose la géolocalisation avant d'accéder à la recherche, afin de pré remplir le champs de recherche sur la ville où laquelle est située l'utilisateur. Une fois la géolocalisation validée, l'utilisateur accède au moteur de recherche, et la recherche est lancée automatiquement. L'utilisateur peut également annuler, à ce moment là le moteur de recherche n'est pas pré rempli, et la recherche n'est pas automatiquement lancée.

Etapes de création

NB: si vous avez initialisé l'application de base avec le router intégré, vous pouvez sauter les premières étapes d'initialisation du routeur (1,2,3).

1. Ajouter le router à l'application Angular :

> npm install @angular/router --save

2. Créer un modulle `AppRoutingModule`

> ng generate module AppRoutingModule

```
import { NgModule } from "@angular/core";
import { Routes, RouterModule } from "@angular/router";
const routes: Routes = [];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

3. Ajouter la `router-outlet` dans le template HTML principale de `app.component`

3. Ajouter le `router-outlet` dans la template HTML principale de `app.component` .

```
<router-outlet></router-outlet>
```

4. Dans `AppRoutingModule` , définir vos routes.
    i. une route pour le module de géolocalisation (nous référencerons le module à l'étape 8),
    ii. une route pour le module météo (nous référencerons le module à l'étape 9)
5. Référencez `AppRoutingModule` dans le module principal,
6. Supprimez le module `MeteoModule` de `AppModule` , qui sera désormais chargé dynamiquement (sinon vous aller le charger dans l'application au chargement, puis au changement de route par Lazy Loading).
7. Créer un module `geoloc` , et un composant d'affichage pour la géolocalisation `meteo-geoloc` (utiliser le CLI),
8. Dans `AppRoutingModule` , référencer le module `GeolocModule` dans vos routes principales, c'est celui qui sera chargé par défaut (utiliser la déclaration pour chargement en lazy loading, voir déclaration dans les exemples plus haut).
9. Dans `AppRoutingModule` , référencer le module `MeteoModule` dans vos routes principales, il sera chargé plus tard (utiliser la déclaration pour chargement en lazy loading, voir déclaration dans les exemples plus haut).
10. Rajoutez les 'sous' routes dans les modules `GeolocModule` et `MeteoModule` pour que les composants par défaut soient chargés,

```
// geoloc module
//déclaration de la route par défaut
export const geolocRouteList: Routes = [
  {
    path: "**",
    component: GeolocComponent,
  },
];
// dans les imports
```

```
RouterModule.forChild(geolocRouteList);

// meteo module
//déclaration de la route par défaut
export const meteoRouteList: Routes = [
  {
    path: "**",
    component: MeteoViewComponent,
  },
];
// dans les imports
RouterModule.forChild(meteoRouteList);
```

11. Dans la template HTML du composant `meteo-geoloc` , indiquez à l'utilisateur qu'il doit accepter la localisation, puis ajouter un bouton "Recherche sur `nom ville` " qui sera affiché uniquement si la géolocalisation est active et si une ville peut être trouvée (nous allons développé le service dans l'étape suivante). Utiliser la directive nglf pour cela.

Bienvenue sur le moteur de recherche de la météo!

Veuillez activer la localisation pour rechercher automatiquement la météo.

Rechercher sur Plougastel-Daoulas ? | Choisir depuis une carte

12. Créer un service `geoloc-service` permettant de récupérer la géolocalisation de l'utilisateur (latitude et longitude), et de la renvoyer vers le composant `meteo-geoloc` .

    Pour le service, utiliser l'API Native de géolocalisation du navigateur

    Vous n'oublirez pas de référencer ce service dans le module `geoloc` , et de l'injecter dans le constructeur du composant. Vous l'exploiterez dans la méthode `ngInit` pour lancer de manière automatique le service. Au lancement de l'application, pour des contraintes de sécurité, le navigateur vous demandera si vous accepter la géolocalisation depuis l'application.

13. Créer un service `geoloc-to-city` , qui va permettre, à partir d'une latitude et longitude récupérée par `geoloc-service` , de la transformer en ville.

    Pour cela, utilisez une API Open Data, pour transformer la position GPS en ville (ex : https://opencagedata.com/api).

    Vous n'oublirez pas de référencer ce service dans le module `geoloc` , et de l'injecter dans le constructeur du composant. Vous l'exploiterez dans la méthode `ngInit` en chaînant l'appel avec `geoloc-service` .

14. Indiquez un texte "Localisation en cours..." sur le composant `geoloc` indiquant à l'utilisateur que la géolocalisation est en cours (uniquement pendant le chargement via ngIf).

15. Une fois la ville récupérée, afficher le bouton permettant de lancer la recherche directement sur la ville récupérée. Le libellé du bouton est la concaténation suivante :

```
Rechercher sur {{ geolocInfo.ville }} ?
```

16. Brancher le bouton "Recherche sur `nom ville` ". Le bouton doit, à partir du routeur, envoyer vers la page météo, en passant en paramètre les informations de la ville récupérée (si une ville a été récupérée).

Ex d'url généré pour le routeur : http://localhost:4200/#/meteo/result?ville=Brest

Vous utiliserez pour cela le routeur directement dans la template HTML via le `routerLink` et le `queryParams` contenant la ville qui sera contenue dans l'URL.

17. Dans le composant `meteo-view`, recupérer la ville potentiellement passée en paramètre de l'url dans la méthode `ngInit`. (voir paragraphe sur la récupération de paramètre)
18. Si une ville est trouvée en paramètre, lancez le chargement de la météo automatiquement.
19. Dans le champ texte du composant `search-meteo`, si une ville est passée dans l'URL, affichez là dans le champ à l'aide de `@Input` et `FormsModule`.

```
@Input("ville") ville: string;
```

Exercice 2

Présentation

Application Météo.

L'objectif de cet excercice est de rajouter une route 'map' permettant d'accéder à une carte si l'on a annulé (ou pas autorisé) la localisation. Pour cela nous allons utiliser une dépendance externe ESRI, qui est un éditeur mettant à disposition une API cartographique en JS, et exploitant des serveurs cartographiques Arcgis.

Etapes de création

1. installer le loader ESRI Map

> npm install esri-loader --save

2. Référencer les CSS nécessaire à l'utilisation de l'API ESRI dans votre fichier `index.html`

```
<link rel="stylesheet" href="https://js.arcgis.com/4.7/esri/css/main.css" />
```

3. Intégrer la map ESRI en vous appuyant sur cette exemple : https://stackblitz.com/edit/angular6-arcgis-api

   I. Créer un composant `meteo-map` , et intégrer le composant `esri-map` dans la template HTML.

```
<div id="mapContainer">
  <esri-map id="map" basemap="topo"></esri-map>
</div>
```

II. Référencer le nouveau composant dans le module `MeteoModule`

III. Créer un service `arcgis-api.service` (comme dans l'exemple) dont l'objectif est de charger les dépendances JS et d'initialiser la map

IV. Ajouter les CSS nécessaire au bon fonctionnement :

```
#mapContainer {
  width: 100%;
  height: calc(100vh - 8em);
  background-color: brown;
}

#map {
  width: 100%;
  height: calc(100vh - 8em);
}
```

4. Modifier les routes de `MeteoModule` pour créer une route vers la Map.

```
export const meteoRouteList: Routes = [
  {
    path: "result",
    component: MeteoViewComponent,
  },
  {
    path: "map",
    component: MeteoMapComponent,
  },
];
```

5. Dans le composant `meteo-geoloc` , rajouter un bouton "annuler". Ce bouton "annuler" annulera la géolocalisation, et enverra vers la vue Map pour que l'utilisateur puisse choisir une position sur la carte. Exploiter pour cela le routeur pour mener à la Map.

Ex : http://localhost:4200/#/meteo/map

6. Tester le bouton "annuler" pour valider que le routeur fonctionne, et que le composant map s'affiche, et que carte se charge coorectement.

Ma météo !

Bienvenue sur le moteur de recherche de la météo!

7. Exploiter l'output MapClick dans le contrôleur `esri-map.component` pour récupérer les infos de coordonnées. Cette méthode sera appelée lorsque vous cliquerez sur la carte, et contiendra des informations de localisation dans le paramètre de cette méthode callback.

```
this.sceneView.on('click', (event) => {
                  this.mapClickHandler(event);
                });

 public async mapClickHandler(event: any) {

 }
```

8. A partir des informations récupérées dans l'event, exploiter le service géolocalisation `geoloc-to-city` pour retrouver la ville associée aux coordonnées. Vous injecterez donc ce service dans le composant `meteo-map` . Ca tombe bien, ce service prend en entrée des données lat et lon que vous avez récupéré depuis l'event.

9. Une fois la ville récupérée par le service, naviguer vers la page de resultat via le routeur (`Router` à injecter également dans le constructeur de `meteo-map`), en lui passant la ville trouvée via `queryParams` . Cette fois, vous ne naviguez donc pas via la template HTML, mais par le biais du contrôleur.

   De la même façon, l'url généré doit être de ce type : http://localhost:4200/#/meteo/result?ville=Brest

10. Une fois la navigation opérationnelle, la magie devrait opérer, puisque vous avez déjà codé la récupération du paramètre et le chargement automatique dans la vue `meteo-view` (étape 17,18 et 19 du précédent exercice). Lorsque vous cliquez sur la carte, les coordonnées sont récupérées, le service de localisation récupère la ville associé, et le routeur navigue vers la page de recherche, qui charge automatiquement les résultats.

Ma météo !

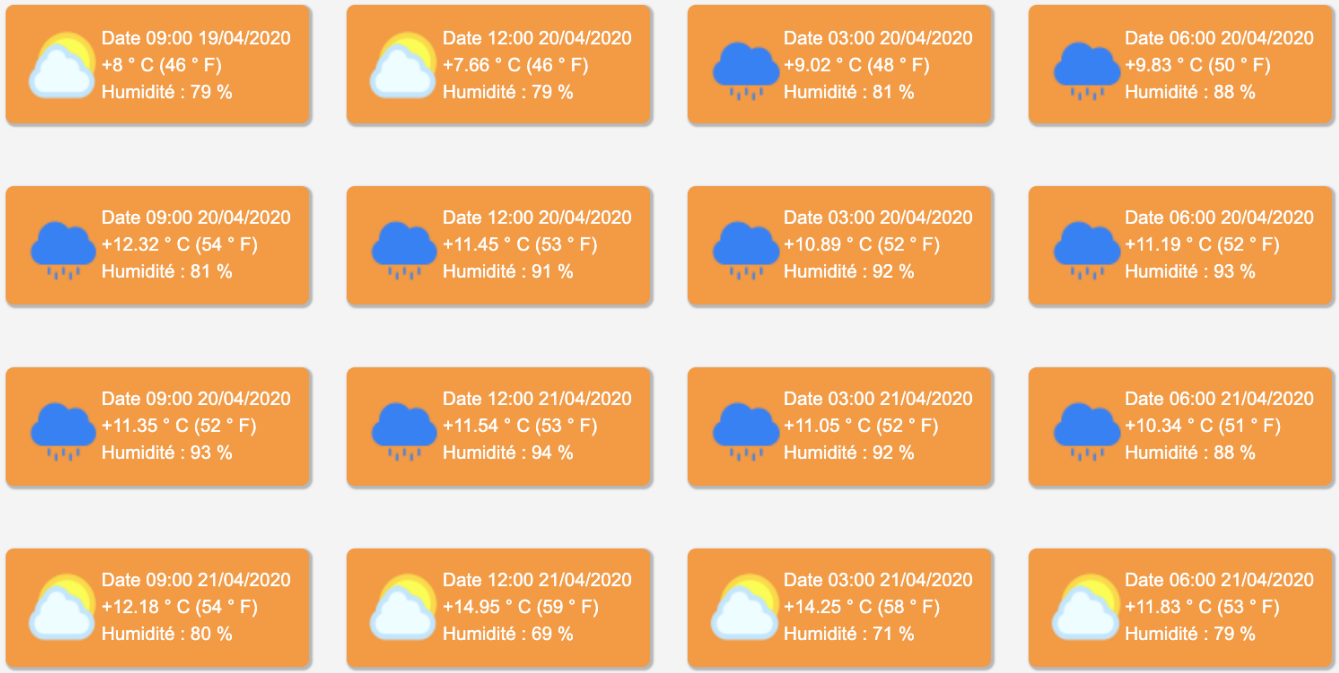
Bienvenue sur le moteur de recherche de la météo!

rechercher une ville... *

Brest

Lancer

Brest - 48.4|-4.4833 - 144899 habitants



Vous venez de développer une application complète de météo, permettant à partir d'une géolocalisation, ou d'une carte, d'afficher des prévisions météo ! Votre application est optimisée par le routeur, grâce au Lazy Loading, vous avez chargé les modules uniquement quand cela était nécessaire.

Correction disponible ici