ANGULAR

FRAMEWORK JS
B3 WEB DEVELOPPEUR

FORMATION 2022-2023

MODULE 42H
MYDIGITALSCHOOL

Formateur : Yohann CORDIER
Mail : nemorigolo@hotmail.com
Tel : 06 81 47 12 77

OBJECTIFS PÉDAGOGIQUES

- A l'issue de ce module, l'apprenant sera à même d'utiliser une technologie Javascript pour réaliser une application côté front-end.
  - Connaître les fondamentaux théoriques des différents frameworks du marché
  - Connaître les fondamentaux théoriques MVC, MVVM, FLUX …
  - Pratiquer et utiliser au moins l'un des frameworks js suivants : Angular, Backbone, React ou Vue
  - Savoir appeler avec le framework js une api extérieure
  - Réaliser une application avec le framework js pratiqué

EVALUATIONS

- 1 évaluation sous forme de QCM, à mi module (21h)
  - Noté sur 40 questions (42 questions, 2 points bonus)

- 1 évaluation sur le rendu du projet final en fin de module (42h)
  - (5 points) A l'analyse du besoin
  - (5 points) Au Clean Code
  - (5 points) L'architecture du projet
  - (5 points) Gestion technique de Angular
  - (5 points) Gestion fine de Angular
  - (5 points) L'avancée du projet
  - 2 points bonus : Design graphique

PRÉSENTATION DE ANGULAR

- AngularJS 1.x : Créé en 2010 par 2 employés de Google
  - FullJS
  - Faciliter le développement de projets interne à Google
- Angular 2+ : En 2014, Réécriture complète suite à des limitations
  - Proposer un Framework qui simplifie la création d'application Web lourdes
  - Utilisation de Typescript
  - Non compatible avec AngularJS
  - Sortie finale en 2016, gros retards
  - Mécontentement de la communauté qui boude Angular au profit de React
- Depuis : 1 mise à jour majeur tous les 6 mois
- Version actuelle (en Mars 2023 : Angular 15)

- https://www.sitedetout.com/histoire-angular/

INSTALLATION DE ANGULAR

- Pré requis :
  - Npm (version 18+ pour Angular 15)
- Installation de la CLI (outil permettant de créer des projets simplement)
  - npm install –g @angular/cli
- Création d'un projet vierge :
  - ng new <NOM_DU_PROJET>

- ng : Commande de base pour la gestion des snippets Angular

QUELQUES SOURCES POUR BIEN DÉMARRER

- https://angular.fr/
- https://angular.io/
- https://google.fr/ (c'est Cadeau)

STRUCTURE D'UN PROJET ANGULAR

- https://formationjavascript.com/architecturer-un-projet-angular/

MON PREMIER MODULE

- ng g m <NOM_DU_MODULE>

- Le module peut servir en tant que :
  - Module de Page : Structure un “package” fonctionnel autonome et modulaire
  - Module de Composant : Créer une bibliothèque de composants partagés
  - Module de Routage : Défini les routes à utiliser pour un module référent

STRUCTURE D'UN MODULE

```
@NgModule({
declarations: [],
imports: [],
exports: [],
providers: [],
bootstrap: []
})
export class MyModule {}
```

Declarations : Liste des composants accessible par ce module
Imports : Liste des dépendances (autres modules) nécessaire pour ce module
Exports : Liste des composants exposé et exploitable pour d'autres modules
Providers : Liste des services ayant un scope limité au module
(accessible seulement par les composants gérés par ce module)
Bootstrap : Liste des composants initialisé au démarrage de l'application
(utilisé généralement dans le app.module pour référencer le composant de démarrage)

MON PREMIER COMPOSANT

- ng g m <NOM_DU_COMPOSANT>
- ng g c <NOM_DU_COMPOSANT> -m <NOM_DU_MODULE>

- Doit être déclaré dans un et un seul module (jusqu'à Angular 15)
- Se compose de 3 fichiers :
  - <mon_component>.component.html => La View
  - <mon_component>.component.scss => Le style CSS
  - <mon_component>.component.ts => Le ViewModel
- Le .ts référence les 2 autres fichiers

STRUCTURE D'UN COMPOSANT

```
@Component({
selector: 'app-consumer',
templateUrl: './consumer.component.html',
styleUrls: ['./consumer.component.scss'],
providers: []
})
export class ConsumerComponent implements OnInit, OnDestroy {
  constructor() {}

  ngOnInit(): void {}

  ngOnDestroy(): void {}
}
```

Selector : Identifiant du composant, accessible dans le HTML
templateUrl : Chemin du fichier HTML attaché
StyleUrls: Chemin du ou des fichiers CSS attachés
Providers: Liste des fournisseurs et services ayant un scope limité à ce composant

LE ROUTAGE SUR ANGULAR

- Gère la navigation dans l'application SPA (Single Page Application)
- Gère le chargement des modules / composants en fonction de l'Url
- Gère la gestion des accès aux Urls

- https://blog.codewise.fr/angular-routing
- https://angular.io/guide/router
- https://angular.io/guide/lazy-loading-ngmodules

LES ROUTER-OUTLET

- Associé au Router, mais côté HTML
- Gère l'affichage des composants en fonction du résultat du Router

- https://angular.io/api/router/RouterOutlet

LES GUARDS

- Associé au Router, gère l'accès aux routes

https://guide-angular.wishtack.io/angular/routing/route-guards

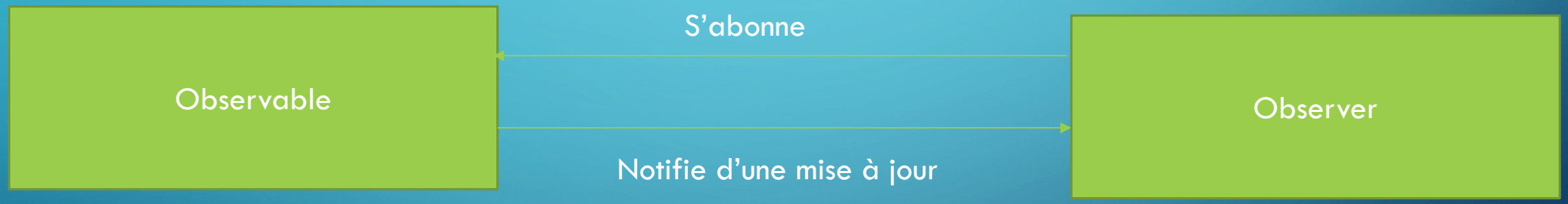
LES RESOLVERS

- Associé au Router, permet d'exécuter des traitements avant le chargement d'un module / composant

- https://makina-corpus.com/front-end/routing-angular-optimisez-le-rendu-au-changement-de-page

LES OBSERVABLES

- Design pattern Observer/Observable



- On appelle cela du "Push" : Le porteur de la donnée notifie les personnes qui se sont abonnées pour recevoir la donnée, si elle devait changer

LES OBSERVABLES

- Utilisation de la librairie RxJs
- Plusieurs types d'Observables, parmi eux, les Subject et BehaviorSubjects

- https://rxjs.dev/

EXEMPLE AVEC LE BEHAVIORSUBJECT

Définition d'un service qui expose un Observable

```
@Injectable()
export class AppService {
  // On crée un Observable de type BehaviorSubject<string>
  // Il est private pour éviter qu'un composant externe déclenche les notifications
  private mySubject: BehaviorSubject<string> = new BehaviorSubject<string>('');

  // On expose, lorsque cela est nécessaire, la version "bridée" de l'Observable
  // Ce dernier étant une référence au BehaviorSubject, il relaira la notification automatiquement
  public myObservable: Observable<string> = this.mySubject.asObservable();

  onClick(myValue: string): void {
    // On déclenche la notification de notre BehaviorSuject
    this.mySubject.next(myValue);
  }
}
```

EXEMPLE AVEC LE BEHAVIORSUBJECT

Consommation d'un service qui expose un Observable

```
@Component({
  selector: 'app-consumer',
  templateUrl: './consumer.component.html',
  styleUrls: ['./consumer.component.scss'],
})
export class ConsumerComponent implements OnDestroy {
  // RxJs nous fourni une classe pour gérer les subscriptions
  private subscription: Subscription = new Subscription();

  constructor(private readonly appService: AppService) {
    this.subscription.add( // On récupère la référence à l'abonnement pour pouvoir se désabonner
      appService.myObservable.subscribe((myValue: string) => {
        // le .subscribe permet de s'abonner à l'Observable
        // On attend la notification de l'Observable pour déclencher un traitement ici.
      })
    );
    // L'abonnement à l'observable est non bloquant.
    // On continue à executer les opérations suivantes
  }

  // Déclenché avant la destruction du composant
  ngOnDestroy(): void {
    // On demande à se désabonner de tous les Observables
    this.subscription.unsubscribe();
  }
}
```

LES REQUÊTES API

- Utilisation de HttpClientModule et HttpClient
- Fournit la liste des fonctions pour effectuer des requêtes Http

- HttpClientModule à importer une fois dans le app.module
- HttpClient à importer à chaque besoin

- Les fonctions (get, poste, put…) retournent un Observable.

- https://angular.io/guide/http

LES INTERCEPTORS

- Middleware pouvant agir avant et/ou après une requête
- Peut être utilisé pour :
  - Ajouter des header automatiquement avant une requête HTTP
  - Récupérer une erreur après réception d'une requête HTTP
  - Récupérer une erreur non contrôlée dans l'application

    …

- http://blog.ezoqc.com/les-intercepteurs-angular/

L'INTERNATIONALISATION (OU I18N)

- Permet de gérer plusieurs langues dans Angular
- 2 solutions possibles :
  - Utiliser le système natif de Angular
  - Utiliser une librairie externe

- https://whiteduck.de/how-to-approach-angular-internationalization-i18n-in-2022/

LE I18N ANGULAR

- Simple à mettre en oeuvre et utiliser
- Maintenu par Angular (car natif au Framework)
- Se base sur des attributs supplémentaires (i18n dans les balises HTML, $localize dans le TS)
- Génère des fichiers xliff par défaut pour les traductions
- Par défaut, génère un Bundle par langue

- https://angular.io/guide/i18n-overview
- Possibilité de charger “dynamiquement” les langues au démarrage de l’application
  - On s’affranchi de la generation de plusieurs Bundle
  - On doit gérer des fichiers xliff, puis les convertir en json
    https://whiteduck.de/how-to-approach-angular-internationalization-i18n-in-2022/

ANALYSEUR DE BUNDLE

- Très utile pour optimiser la taille et le chargement des Bundle
- Utilise une librairie externe : `webpack-bundle-analyzer`
  - npm install –g webpack-bundle-analyzer
- Génère un fichier json + interface Web avec graphe
- Ajouter dans le package.json :
  - `"analyze": "ng build --stats-json && webpack-bundle-analyzer ./dist/stats.json"`

ANALYSEUR DE BUNDLE