ANGULAR

Google Angular 15 LTS

Présentation d'Angular

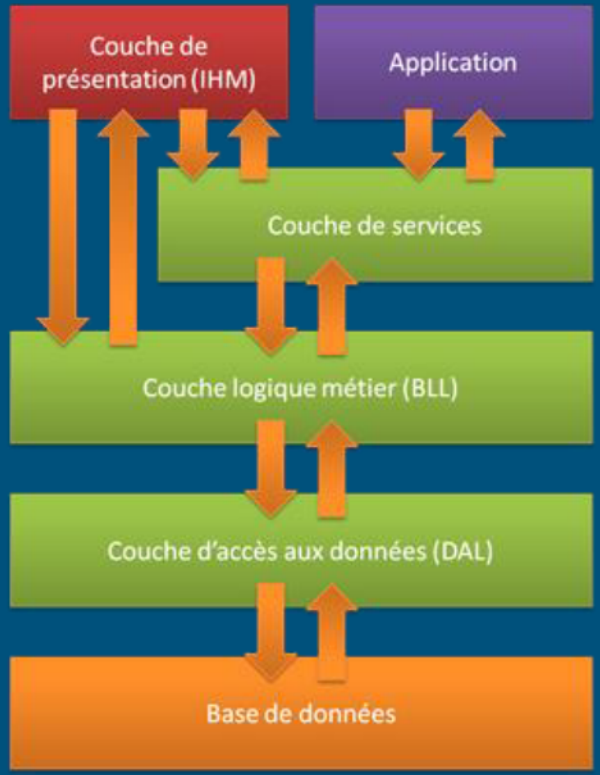
Créé par Google en 2009 sous le nom d'AngularJS

Angular depuis la version 2 en 2016

Autre frameworks : React, VueJS

Présentation d'Angular

Framework FrontEnd



Présentation d'Angular

Nécessite un serveur NodeJS pour le développement

Utilise le langage TypeScript :

- Créé en 2012 par Microsoft
- Implémente la version 6 d'EcmaScript
- Typage obligatoire, orienté POO
- Améliore et sécurise le code
- Transpilé en Javascript

Le projet Angular

Installation des outils

- Serveur : NodeJS
- Package Manager : npm ou yarn
- Client Angular : ng

Création du projet

- Commande d'initialisation d'un projet : `ng new nom-du-projet`

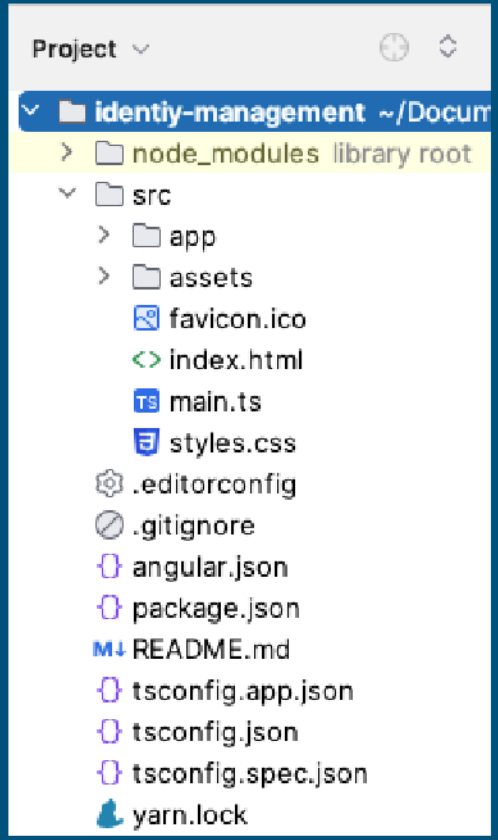
Structure d’un projet Angular

La structure de base contient :

- Versionning des sources (git)
- Dépendances aux modules externes
- Configuration du projet
- Sources de l’application
- Tests

Le projet Angular

Extrait de la structure



Structure d’un projet Angular

La configuration du projet :

- Fichier .editorconfig
  - Configuration de l’IDE
- Fichier angular.json
  - Configuration d’angular-cli (build, lint, tests, ...)
- Fichier tsconfig.json
  - Configuration de typescript pour le projet
- src/** contenant le code des tests fonctionnels :
  - Fichiers suffixés par .spec.ts décrivent les tests

Structure d’un projet Angular

Les sources de l’application (dossier src)

- Dossier app
  - Code de l’application (Typescript / HTML / CSS)
- Dossier assets
  - Assets (images, fonts...)
- Fichier index.html
  - Entrée de l’application recevant toutes les requêtes
- Fichier main.ts
  - Bootstraping de l’application

Structure d’un projet Angular

Les dépendances :

- Dossier node_modules
  - Modules utilisés par le projet (de base presque 800)
- Fichier package.json
  - Décrit le nom du projet et sa version
  - Décrit les tâches npm disponibles
  - Contient également toutes les dépendances externes utilisées divisées en deux catégories :
    - Développement
    - Production

Material UI

Bibliothèque de composants UI

Règles Material Design de Google

Installation : `ng add @angular/material`

Importation des modules Material depuis le module principal

Les composants

Un composant est une classe permettant d'avoir un rendu HTML, il est constitué de 3 parties :

- Fichier TypeScript :
  - décrit le comportement du composant
- Fichier HTML :
  - contient le template
  - utilise des variables et/ou méthodes du composant par interpolation, ex: {{ var }}
- Fichier CSS (uniquement à ce composant)

Les composants

Exemple de composant :

```
@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    `
})
export class AppComponent {
  title = 'Tour of Heroes';
}
```

- @ définit une Directive
  - selector : nom de la balise
  - template :
    - code ou fichier HTML
    - interpolation : ex: `{{ title }}`
- Classe dépendant de la directive

Les composants - Echange de données

Echange de données Parent - Enfant avec les propriétés @Input et @Output

Le parent et/ou l'enfant peuvent être des "components"

- **@Input** permet d'envoyer des données du parent vers l'enfant
  - propriété en écriture, utilisation des **[ ]**
- **@Output** permet d'émettre un événement de l'enfant vers le parent
  - propriété en lecture de type événement observable, utilisation des **( )**

Les composants - @Input

Child	Parent

Composant :
`export class ItemDetailComponent {`
`  @Input() item: string;`
`}`
 	
Html :
`<p>`
`  Today's item: {{item}}`
`</p>`
	Composant :
	`export class AppComponent {`
	`  currentItem = 'Television';`
	`}`
	
	Html :
	`<app-item-detail [item]="currentItem"></app-item-detail>`

Les composants - @Output

Child	Parent

Composant :
`export class ItemOutputComponent {`
`  @Output() newItemEvent = new EventEmitter<string>();`
`  addNewItem(value: string) {`
`  this.newItemEvent.emit(value);`
`  }`
`}`
	Composant :
	`export class AppComponent {`
	`  items = ['item1', 'item2'];`
	`  addItem(newItem: string) {`
	`  this.items.push(newItem);`
	`  }`
	`}`

Html :
`<label>Add an item: <input #newItem></label>`
`<button (click)="addNewItem(newItem.value)">`
`Add to parent's list</button>`
	Html :
	`<app-item-output (newItemEvent)="addItem($event)">`
	`</app-item-output>`

Les composants - Template reference

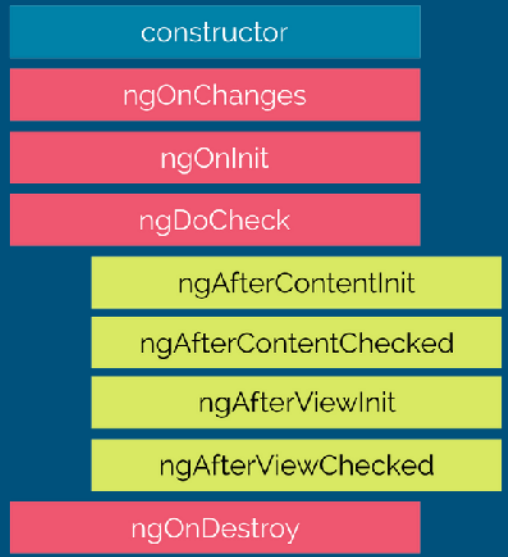
Une Template reference variable fait référence à un élément du DOM, la variable est préfixé du symbole #

Html :

```
<input #phone placeholder="phone number" />
<!-- phone refers to the input element; pass its `value` to an event handler -->
<button (click)="callPhone(phone.value)">Call</button>
```

Remarque : La variable n'existe que dans le html.

Les composants - Cycle de vie



- **ngOnchanges** : après le constructeur et à chaque changement de valeur d'une propriété @Input
- **ngOnInit** : appelé une seule fois (initialisation pour les API par exemple)
- ngDoCheck : à chaque changement (ne pas utiliser avec ngOnchanges)
- ngAfterContentInit : initialisation des composants enfants (une seule fois)
- ngAfterContentChecked : après ngAfterContentInit et après chaque ngDoCheck
- ngAfterViewInit : initialisation des vues enfants (une seule fois)
- ngAfterViewChecked : après ngAfterViewInit et après chaque ngAfterContentChecked
- **ngOnDestroy** : avant la destruction du composant

Les Directives

Les directives permettent de changer l'apparence ou le comportement d'un éléments

3 types de directives :

- Composants : @Directive
- Structurel : modifie le DOM
- Attributs : modifie l'apparence ou le comportement d'un attribut

Les Directives pour un composant

Exemple (partiel), avec la gestion du Drag ’n’ Drop :

```
@Directive({  selector: '[appDragNDrop]'})
export class DragNDropDirective {
  @HostBinding('class.fileover') fileOver: boolean;
  @Output() fileDropped: EventEmitter<FileList> = new EventEmitter();

  // Dragover listener
  @HostListener('dragstart', ['$event']) onDragStart(evt) {
    evt.preventDefault();
    evt.stopPropagation();
    this.fileOver = true;
    console.log(evt);
    return true;
  }
  ...
```

Les Directives Structurelles

Elles sont préfixées du symbole *, exemples :

- ngIf
  - conditionne un bloc/une balise
  - `<h2 *ngIf="estImportant">Ceci est un message important</h2>`
- ngFor
  - itère une liste
  - ```
    <ul *ngFor="let item of [1,2,3]">
      <li>Elément n° {{ item }}</li>
    </ul>
    ```
- ngSwitch
    ```

Les Directives par attribut

- ngModel
  - échange de données d'un formulaire (voir chapitre plus loin)
- ngClass
  - conditionne les classes CSS
  - `<p [ngClass]="{'important': true}">Un texte important</p>`
  - Avec le CSS : `.important { font-weight: bold; color: red; }`
- ngStyle
  - applique un style à un objet DOM
  - `<div [ngStyle]="{background-color = getBGColor()}">Un autre texte</div>`
  - Avec `le code : getBGColor() { return this.bgColor; }`

Les Pipes

Permet de transformer une donnée pour l'afficher

Exemples :

- `<p>Aujourd'hui : {{ dateNow | date: "dd/MM/yyyy" </p>`
- `<p>Total : {{ prixTotal | currency: "EUR": "symbol":"4.2-2":"fr" }} </p>`
- `<p>{{ "Que des majuscules" | uppercase }}`

Les routes

La navigation se fait par “routing” :

- Avec le module Router
- D’après des chemins

```
const routes: Routes = [
  { path: 'first-component', component: FirstComponent },
  { path: 'second-component', component: SecondComponent },
  { path: '', redirectTo: '/first-component', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

- Avec le composant router-outlet (permet d’inclure un template)

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

Les formulaires

	Reactive forms	Template-driven forms
Initialisation	Explicite Dans la classe	Implicite Avec les directives
Données	Structurées Immuables (nouvelle instance de la donnée)	Non structurées Mutables (valeur de la données)
Echange	Synchrone	Asynchrone
Validation	Fonctions	Directives

Les formulaires - Template-driven forms

- Model to View : [attributeHML]=”value”
- View To Model : (event)=”function(...)”
- Bi-directionnel :
  - [(attributeHTML)]=”attributeComponent”
  - [(ngmodel)]=”modelName”

Template-drive forms

Les formulaires - Reactive forms

Nécessite un FormControl, simplifié avec un FormBuilder

```
export class ProfileEditorComponent {
  profileForm = this.fb.group({
    firstName: [''],
    lastName: [''],
    address: this.fb.group({
      street: [''],
      city: [''],
      state: [''],
      zip: ['']
    }),
  });
  constructor(private fb: FormBuilder) { }
}
```

```
<form [formGroup]="profileForm">
  <label>
    First Name:
    <input type="text" formControlName="firstName">
  </label>
  <label>
    Last Name:
    <input type="text" formControlName="lastName">
  </label>
</form>
```

Les formulaires - Reactive forms

Get/Set value

- this.*form*.get(*'nomDuControle'*).setValue(aValue)
  - Exemple : this.profileForm.get('firstName').setValue('Franck');
  - Exemple : this.profileForm.get('address.city').setValue('Montpellier');
- let value = this.*form*.get(*'nomDuControle'*).value;
  - let codepostal = this.profileForm.get('address.zip').value;

Etat du contrôle:

- this.*form*.get(*'nomDuControle'*).valueChanges.subscribe(...) {...}

Les formulaires - Reactive forms

Validation

```
ngOnInit(): void {
  this.heroForm = new FormGroup({
    name: new FormControl(this.hero.name, [
      Validators.required,
      Validators.minLength(4),
      forbiddenNameValidator(/bob/i)
    ]),
    alterEgo: new FormControl(this.hero.alterEgo),
    power: new FormControl(this.hero.power,
      Validators.required)
  });
}
```

```
export function forbiddenNameValidator(nameRe: RegExp)
  : ValidatorFn {
  return (control: AbstractControl):
  {[key: string]: any} | null => {
    const forbidden = nameRe.test(control.value);
    return forbidden ?
      {forbiddenName: {value: control.value}} : null;
  };
}
```

Les services

- Composant :
  - présentation des données
- Service :
  - classe “simple”
  - accessible dans toute l’application
    - Dependency Injection
  - manipulation des données (API)
    - Classe HttpClient avec des méthodes de type Observable

```
@Injectable({ providedIn: 'root' })
export class HeroService {
  private heroesUrl = 'api/heroes';  // URL to web api
  httpOptions = {
    headers: new HttpHeaders({ 'Content-Type':
                               'application/json' })
  };
  constructor(
    private http: HttpClient,
    private messageService: MessageService) { }
  /** GET heroes from the server */
  getHeroes(): Observable<Hero[]> {
    return this.http.get<Hero[]>(this.heroesUrl);
  }
}
```

Les services - Observable

Observable : passage de valeurs similaire au Design Pattern “Observer”

Observable	Promise
Valeurs multiples	Valeur unique
Traitement déclenché lors d’un résultat	Traitement immédiat de la classe (hors `then(...)`)
Plusieurs opérateurs RxJS	then / catch uniquement
Annulable	Non annulable

Les services - Observable

Exemple :

```
const counter :Observable<number>  = of( values: 1,  values: 2,  values: 3,  values: 4);
counter.subscribe( observerOrNext: {
  next (value :number ) :void  {
    console.log(`${value} count`);
  },
  error (err) :void  {
    console.error(err);
  },
  complete() :void  {
    console.log('end of observable');
  }
});
```

Voir aussi : Subject

Les services - RxJS

RxJS (Reactive Extensions for JavaScript) :

- Ensemble d'opérateurs
- Agit sur les collections
- Exemple :

```
const squareOdd = of(1, 2, 3, 4, 5)
  .pipe(
    filter(n => n % 2 !== 0),
    map(n => n * n)
  );
// Subscribe to get values
squareOdd.subscribe(x => console.log(x));
```

La sécurité

L’authentification selon Angular : mise en place d’un “Guard”.

“Guard” : est une classe

- permettant de contrôler la navigation
- **héritant d’interfaces** modifiant le comportement du routage

La sécurité

Création d'une classe pouvant hérité des interfaces :

- CanActivate : naviguer vers une route
- CanActivateChild : naviguer vers une route enfant
- CanDeactivate : quitter la navigation et aller vers une autre
- Resolve : récupérer des données de route avant l'activation de la route
- CanLoad : charger un module asynchrone afin de continuer la navigation

La sécurité

Ajout du Guard au Routing

```
const adminRoutes: Routes = [
  {
    path: 'admin',
    component: AdminComponent,
    canActivate: [AuthGuard],
    children: [
      {
        path: '',
        children: [
          { path: 'crises', component: ManageCrisesComponent },
          { path: 'heroes', component: ManageHeroesComponent },
          { path: '', component: AdminDashboardComponent }
        ],
      }
    ]
  }
];
```