Formation Angular Avancée

Ihab ABADI / UTOPIOS

Programme

- **Architecture optimisée pour projet Angular**
- **Utilisation des tests et l'approche TDD**
- **Utilisation avancée du Routing et Optimisation**
- **Utilisation des événements personnalisés**
- **Formulaires**
- **Données et optimisation.**
- **Angular et programmation réactive**
- **Sécurité et Angular**
- **NodeJs FullStack(express et nestJs)**

Programme Jour 1

- **Architecture Angular**
  - **Angular Modules**
  - **Type de modules Angular**
  - **Modules de pages**
  - **Modules globaux**
  - **Modules réutilisables**
- **Test et TDD**
  - **Test avec Jasmine**
  - **Test unitaire**
  - **TDD**
  - **TestBed**
  - **Test d'un service**
  - **Test d'un composant**
  - **Test requête HTTP**
  - **Test end to End**
- **Utilisation avancée du Routing et Optimisation**
  - **Rappel notion de lazy loading**
  - **Utilisation des resolver.**

Architecture Angular

- **Pour réaliser une architecture optimisée d'un projet Angular, il faut garder comme objectif :**
  - **Avoir une architecture cohérente**
  - **Avoir une architecture réutilisable**
  - **Avoir une architecture facilement optimisable**
  - **Avoir une architecture testable**

Architecture Angular – Angular modules

- **Pour réaliser une architecture Angular avec les objectifs préalablement définies, on peut utiliser la notion de module.**
- **La création d'un module se fait à l'aide de l'annotation ngModule.**
- **Le but d'un module en Angular est de regrouper des composants, services, directives.**
- **Les éléments d'un module doivent rester cohérent.**
- **On peut distinguer trois types de modules.**
  - **Modules de pages**
  - **Modules globaux**
  - **Modules réutilisables**

Architecture Angular – Modules de pages

- **Les modules de pages contiennent les éléments qui séparent et organisent les différents partie de notre application.**
- **Les pages de notre application peuvent être charger en une ou plusieurs fois.**
- **Les modules de pages peuvent contenir**
  - **Des services et interfaces**
  - **Des composants routés**
  - **Des composants de présentation**

Architecture Angular – Modules de pages - Services et interfaces

- **Les services et interfaces sont des éléments qui peuvent être utiliser par un ou plusieurs composants.**
- **Les services et interfaces peuvent être dans un dossier shared.**
- **Dans le cadres d'un lazy loading les services et interfaces ne seront pas accessible dans un autre module.**
- **Démo**

Architecture Angular – Modules de pages - Composant routés

- **Les composants routés sont des pages de votre application.**
- **Chaque composant est associé à une route.**
- **Un composant page injecte et utilise les services pour charger des données.**
- **Un composant page ne doit pas afficher les données directement dans un template.**
- **Les données doivent être envoyées dans un input à un composant de rendu.**
- **On peut avoir un dossier pages.**
- **Démo**

Architecture Angular – Modules de pages - Composants de présentation

- **Un composant de présentation récupère les données à l'aide d'un input et les affiches dans un template.**
- **On peut avoir un dossier components.**
- **Démo**

Architecture Angular – Modules globaux

- **Les modules globaux contiennent des services accessibles partout dans l'application.**
- **Les modules globaux doivent être charger directement dans AppModule.**
- **Chaque module global doit avoir un point d'entrée.**
- **Les modules globaux peuvent être dans un dossier core.**
- **démo**

Architecture Angular – Modules réutilisables

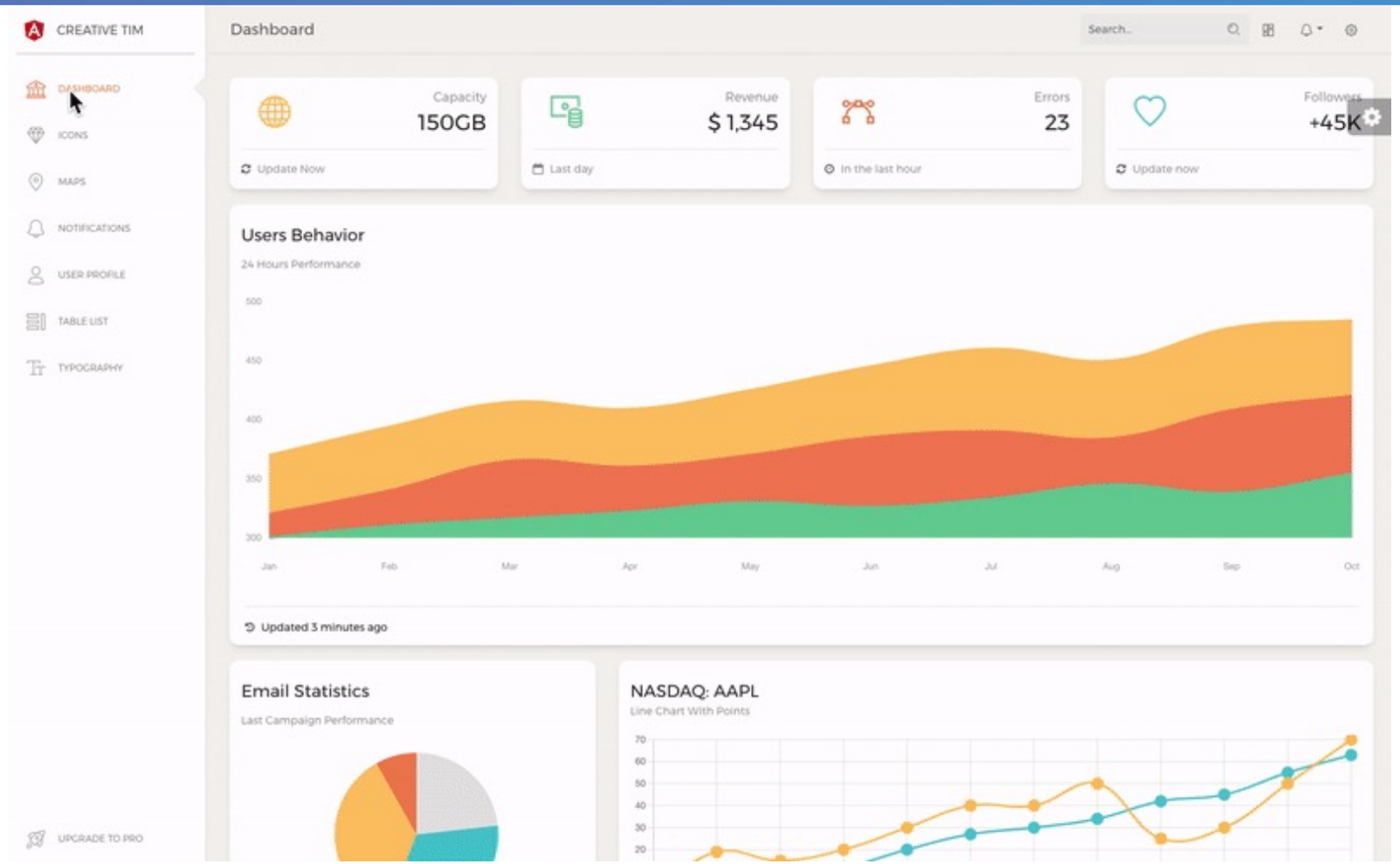
- **Les modules de composants réutilisables sont des modules de rendu réutilisables dans plusieurs projets.**
- **Ces composants ont une portée locale.**
- **Ces composants doivent être déclarés dans chaque autre module qui utilises ces composants.**
- **Les modules réutilisables peuvent contenir des directives et des pipes.**
- **Les modules réutilisables peuvent contenir des services.**

Architecture Angular – Modules réutilisables – Public et privés

- **Les modules réutilisables peuvent avoir des composants public ou privés.**
- **Les composants public doivent être exporter.**
- **On peut avoir des services privés dans un composant réutilisable pour manipuler les données.**
- **Un service privé doit être injecté directement dans le composant.**
- **Dans le cadres d'un service public, il faut utiliser une Factory pour exporter le service pour éviter de charger le service plusieurs fois.**
- **Démo**

Architecture Angular - Exercice

- **Proposer une architecture pour une application angular comme:**



Test et TDD – Test avec Jasmine - Karma

- **http://karma-runner.github.io/**
- **Karma = ex-Testacular**
- **Développé par la team Angular**
- **Permet de :**
  - **lancer les tests dans un ou plusieurs navigateurs**
  - **lancer les TU à chaque modification de fichiers**
  - **Est entièrement configuré à travers angular.json**
  - **Se lance avec la commande : npm run ng test**
  - **Peut se lancer en parallèle d'un npm run ng serve**

Test et TDD – Test avec Jasmine

- **Bibliothèque de rédaction de TU : Jasmine (https://jasmine.github.io/)**
- **En utilisant angular-cli pour générer des éléments, une structure de test est générée en parallèle**
- **Respectent la structure : Given / When / Then**
- **Il inclut tout le nécessaire pour :**
  - **définir des suites de tests (fonctions describe et it),**
  - **implémenter des assertions de toute sorte (fonction expect),**
  - **implémenter rapidement des "Spies" (alias mocks) (fonctions createSpy, createSpyObj et spyOn).**

Test et TDD – Test avec Jasmine – test unitaire

- **Pour créer un test unitaire, il suffit de créer un fichier avec l'extension .spec.ts**
- **Chaque test doit définir un groupe de test à l'aide.**
- **Une spécification du test.**
- **Une assertion du test.**
- **Chaque groupe de test permet de définir une logique de setup et de teardown.**
- **démo**

Test et TDD – Test avec Jasmine – TDD

- **Le test driven developement consiste à développer les tests avant l'implémentation de la logique métier en suivant le paradigme RGR (Red, Green, Refactor)**
- **Le développement du test en se concentrant sur la fonctionnalité.**
- **Faire échoué le test pour éviter les faux positifs.**
- **Implémenter la fonctionnalité.**
- **Vérifier que le test a réussi.**
- **Démo**

Test et TDD – TDD

- **Le test driven developement consiste à développer les tests avant l'implémentation de la logique métier en suivant le paradigme RGR (Red, Green, Refactor)**
- **Le développement du test en se concentrant sur la fonctionnalité.**
- **Faire échoué le test pour éviter les faux positifs.**
- **Implémenter la fonctionnalité.**
- **Vérifier que le test a réussi.**
- **Démo**

Test et TDD – TestBed

- **TestBed permet d'émuler le fonctionnement d'un module.**
- **La configuration du module se fait à l'aide de la méthode static** configureTestingModule
- **TestBed permet de déclarer les composants, les services à tester.**
- **testBed permet d'importer les dépendances nécessaires pour notre module.**

Test et TDD – Tester un service

- **Pour tester un service, on peut utiliser la méthode get de TestBed.**
- **Dans le cadre de service Asynchrone, on peut utiliser:**
  - **La fonction done.**
  - **Utiliser une promise.**
  - **Utiliser async await.**
  - **Utiliser la fonction async d'angular.**
- **Démo.**

Test et TDD – Tester un composant

- **La méthode TestBed.createComponent instancie le composant dont le type est transmis en paramètre puis retourne un objet ComponentFixture permettant de contrôler et inspecter le composant.**
- **Les principales propriétés et méthodes de cette classe sont les suivantes :**
  - **componentInstance**
  - **Fixture**
  - **DetectChanges**
  - **Démo**

Test et TDD – HttpClientTestingModule

- **HttpClientModule est un module qui fournit un mock pour les différents services nécessaires pour les requêtes HTTP**
- **Démo**

Test et TDD – Exercices

- **Test Basic**
- **Test Composant**
- **Test d'un service**

Test et TDD – End-to-End

- **Même framework et même fonctionnement qu'en AngularJS (1.X)**
- **Lancent l'application dans un navigateur et simulent une utilisation**
- **Permet de tester l'application dans son ensemble**
- **Ils sont plus lents que les TU (plusieurs secondes par test)**
- **Souvent difficile de tester les cas limites**
- **Se mettent en place en complément des TU**
- **Se lance avec npm run e2e**
- **Démo**

Test et TDD – End-to-End - Exercice

- **Exercice simulation action utilisateur**

Routing - Rappel Lazy Loading

- **Permet de ne pas charger toute l'application directement mais d'uniquement charger les modules au moment de leur utilisation**
- **Le lazy loading permet de réduire le temps de chargement d'une application composée de plusieurs modules**
- **Le chargement des modules en lazy peut être définit à l'aide de stratégie de loading pour optimiser du temps de réponse et la navigation.**
- **On distingue deux types de stratégie:**
- **Build-in préloading : NoPreloading ou PreloadAllModules**
- **Custom preloading strategies**
- **Pré chargement en fonction de la qualité de connexion**
- **Démo**

Routing - Resolver

- **Mécanisme permettant de charger des données avant d'afficher la page**
- **Évite les incohérences lors du chargement d'une route**

```
// contact.resolver.ts
import {ActivatedRouteSnapshot, Router, Resolve} from '@angular/router';

@Injectable({
  providedIn: 'root'
})
export class ContactResolver implements Resolve<Contact> {

  constructor(private contactService: ContactService,
              private router: Router) {
  }

  resolve(route: ActivatedRouteSnapshot): Observable<any> {

    const contactId = route.params['contactId'];
    return this.contactService.getContact(contactId).pipe(
      catchError(() => this.router.navigate(['/error']))
    });
  }
}
```

Routing - Resolver

- **Le resolver doit être**
- **configuré dans la route**
- **Dans le composant utilisant le resolve,**
- **la propriété data du ActivatedRoute**
- **contient le contenu des resolves**
- **ici, data.contact correspond à la propriété contact définie dans la route**

```
// app/app.routes.ts
export const AppRoutes: Routes = [
   {
      path: 'contact/:contactId',
      component: ContactComponent,
      resolve: {
         contact: ContactResolver
      }
   }
];
```

```
// contact.component.ts
constructor(private route: ActivatedRoute) {
   this.route.data.subscribe((data: Data) => {
      this.contact = data.contact;
   });
}
```

Programme Jour 2

- **Utilisation des événements personnalisés**
  - **Rappel du mécanisme des events dans Angular**
  - **Optimisation des events à l'aide des directives**
  - **Optimisation des events à l'aide d'Event Plugin Manager**
- **Formulaires**
  - **Rappel Angular template Driven Froms,**
  - **Rappel Reactive Forms et FromBuilder API**
  - **Comparaison**
  - **Scénario d'utilisation**
  - **Validation personnalisée**
  - **FormConrol personnalisée**
- **Données et Optimisation**
  - **Best practice pour le chargement des données.**