



Présentation projet Olympique pour Télésport

CHRISTOPHE PIERRÈS
MERCREDI 28 MAI 2025, DURÉE : 17 MN

ÉVALUATEUR : GEOFFREY HERNANDEZ

COMMENT AI-JE ABORDÉ LE PROJET ?

- Spécifications de départ font apparaître :
 - Nécessité d'une conception évolutive et modulaire
 - Responsive (mobile, tablette, ordinateur) et réactive
 - Identifier les composants réutilisables
 - Avoir la capacité de présenter une maquette (sans les calculs liés au backend)
- Autonomie sur le choix de la librairie graphique
 - Encapsuler les graphes dans ses propres composants pour répondre aux besoins et simplifier l'utilisation de la librairie pour les consommateurs
- Les composants devront s'intégrer dans une application existante chez le client
 - Standalone, documentation complète (README.md + compodoc déployée sur github)
- Starter code, défini selon les pratiques de respect des couches (architecture MVVM)
 - 1^{ère} version fonctionnelle mais rigide (j'apprenais ...) ➔ Refactorisations pour respecter SOLID

COMMENT AI-JE ABORDÉ LE PROJET ?

- Spécifications de départ font apparaître :
 - Nécessité d'une conception évolutive et modulaire
 - Responsive (mobile, tablette, ordinateur) et réactive
 - Identifier les composants réutilisables
 - Avoir la capacité de présenter une maquette (sans les calculs liés au backend)
- Autonomie sur le choix de la librairie graphique
 - Encapsuler les graphes dans ses propres composants pour répondre aux besoins et simplifier l'utilisation de la librairie pour les consommateurs
- Les composants devront s'intégrer dans une application existante chez le client
 - Standalone, documentation complète (README.md + compodoc déployée sur github)
- Starter code, défini selon les pratiques de respect des couches (architecture MVVM)
 - 1^{ère} version fonctionnelle mais rigide (j'apprenais ...) ➔ Refactorisations pour respecter SOLID

COMMENT AI-JE ABORDÉ LE PROJET ?

- Spécifications de départ font apparaître :
 - Nécessité d'une conception évolutive et modulaire
 - Responsive (mobile, tablette, ordinateur) et réactive
 - Identifier les composants réutilisables
 - Avoir la capacité de présenter une maquette (sans les calculs liés au backend)
- Autonomie sur le choix de la librairie graphique
 - Encapsuler les graphes dans ses propres composants pour répondre aux besoins et simplifier l'utilisation de la librairie pour les consommateurs
- Les composants devront s'intégrer dans une application existante chez le client
 - Standalone, documentation complète (README.md + compodoc déployée sur github)
- Starter code, défini selon les pratiques de respect des couches (architecture MVVM)
 - 1^{ère} version fonctionnelle mais rigide (j'apprenais ...) ➔ Refactorisations pour respecter SOLID

COMMENT AI-JE ABORDÉ LE PROJET ?

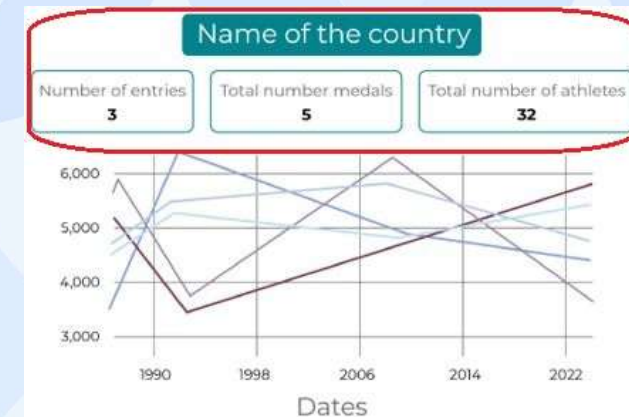
- Spécifications de départ font apparaître :
 - Nécessité d'une conception évolutive et modulaire
 - Responsive (mobile, tablette, ordinateur) et réactive
 - Identifier les composants réutilisables
 - Avoir la capacité de présenter une maquette (sans les calculs liés au backend)
- Autonomie sur le choix de la librairie graphique
 - Encapsuler les graphes dans ses propres composants pour répondre aux besoins et simplifier l'utilisation de la librairie pour les consommateurs
- Les composants devront s'intégrer dans une application existante chez le client
 - Standalone, documentation complète (README.md + compodoc déployée sur github)
- Starter code, défini selon les pratiques de respect des couches (architecture MVVM)
 - 1^{ère} version fonctionnelle mais rigide (j'apprenais ...) ➔ Refactorisations pour respecter SOLID

COMMENT AI-JE ABORDÉ LE PROJET ?

- Spécifications de départ font apparaître :
 - Nécessité d'une conception évolutive et modulaire
 - Responsive (mobile, tablette, ordinateur) et réactive
 - Identifier les composants réutilisables
 - Avoir la capacité de présenter une maquette (sans les calculs liés au backend)
- Autonomie sur le choix de la librairie graphique
 - Encapsuler les graphes dans ses propres composants pour répondre aux besoins et simplifier l'utilisation de la librairie pour les consommateurs
- Les composants devront s'intégrer dans une application existante chez le client
 - Standalone, documentation complète (README.md + compodoc déployée sur github)
- Starter code, défini selon les pratiques de respect des couches (architecture MVVM)
 - 1^{ère} version fonctionnelle mais rigide (j'apprenais ...) ➔ Refactorisations pour respecter SOLID

DÉMARCHE D'ANALYSE – IDENTIFIER LES COMPOSANTS

- Identifier les éléments susceptibles de devenir des composants :
 - En entête, similitudes sur les « box de statistiques »
 - Composant : « **box-stats** »
 - Page d'accueil contient un graphe « Pie » avec drill-down vers une page détail Composant : « **global-graph** »
 - Page détail contient un graphe linéaire avec une seule série
 - Composant : « **detail-graph** »



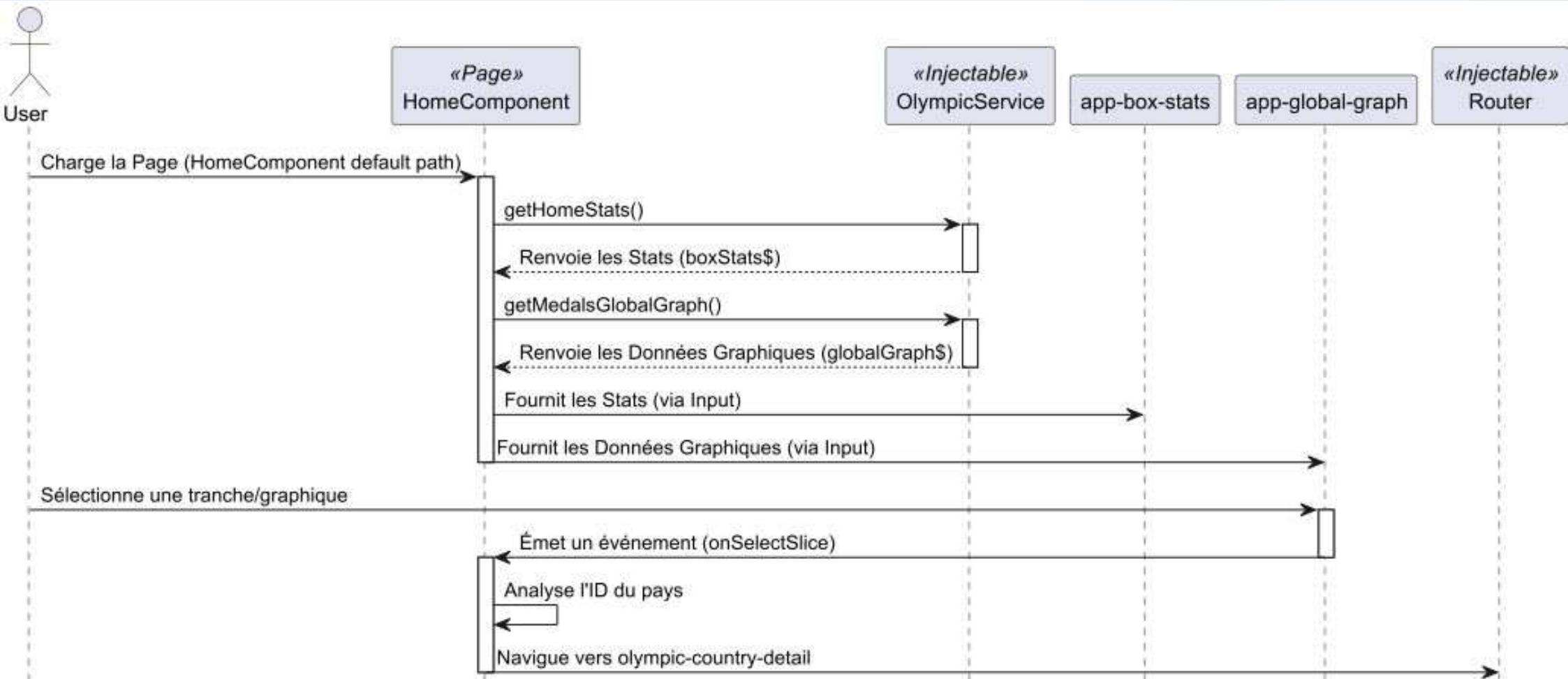
ARCHITECTURE ET BONNES PRATIQUES - SOMMAIRE

- Première démonstration IHM : uses case nominaux (responsive)
- Architecture en couches (MVVM)
 - Revue de l'organisation générale du projet
 - A/R entre code et démonstration IHM pour démontrer SOLID
 - Souplesse, simplicité et maintenabilité des composants
 - Intégration de la librairie ngx-charts
 - Adaptation des composants (ajout de statistiques, changement de graphiques)
 - Démo des cas limites d'IHM (affichages des Messages erreurs, loading)

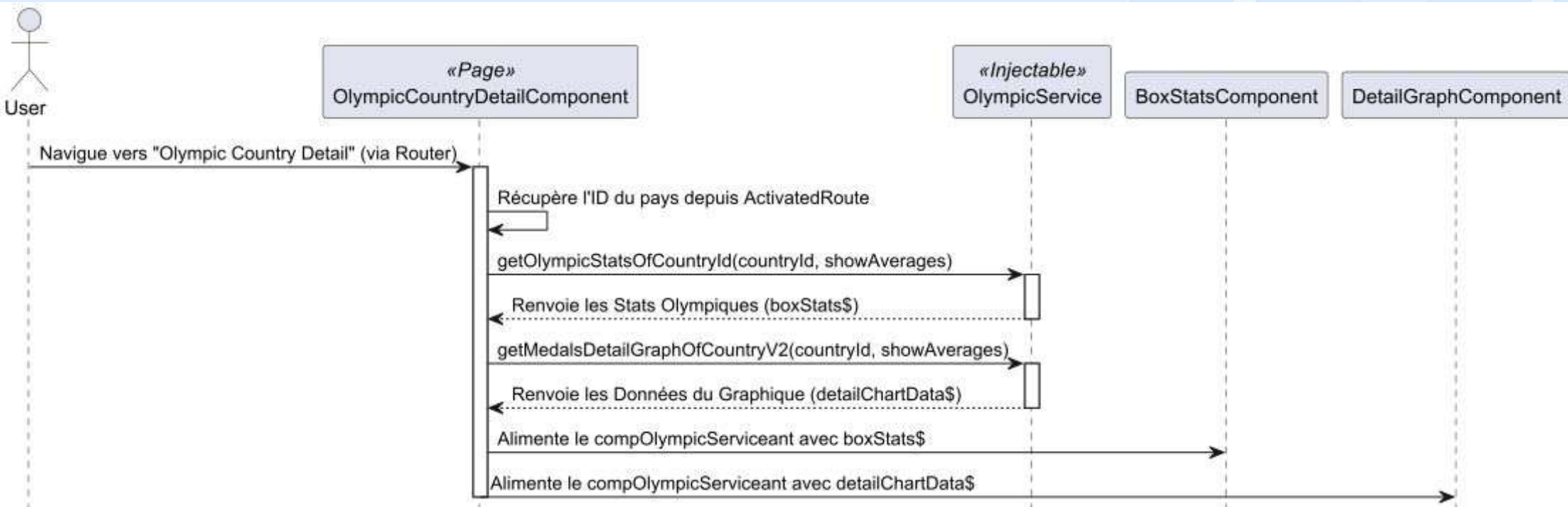
ARCHITECTURE ET BONNES PRATIQUES - SOMMAIRE

- Première démonstration IHM : uses case nominaux (responsive)
- Architecture en couches (MVVM)
 - Revue de l'organisation générale du projet
 - A/R entre code et démonstration IHM pour démontrer SOLID
 - Souplesse, simplicité et maintenabilité des composants
 - Démo des cas limites d'IHM (affichages des Messages erreurs, loading)

ÉCRAN D'ACCUEIL – SEQUENCES ENTRE PAGE-SERVICE-COMPOSANT



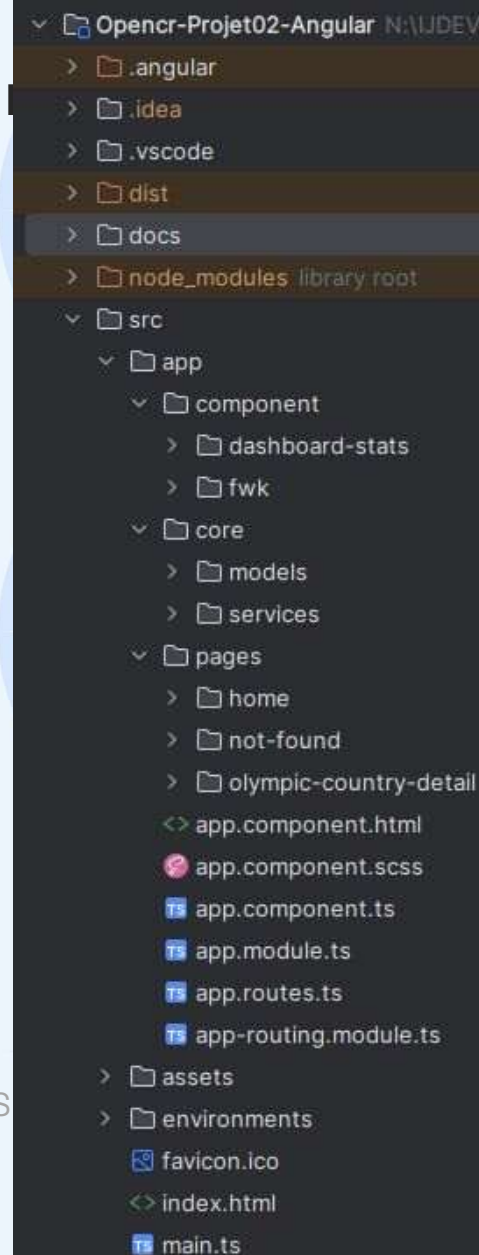
PAGE DÉTAIL



STRUCTURE DE L'APPLICATION ANGULAR / RESPONSABILITÉS

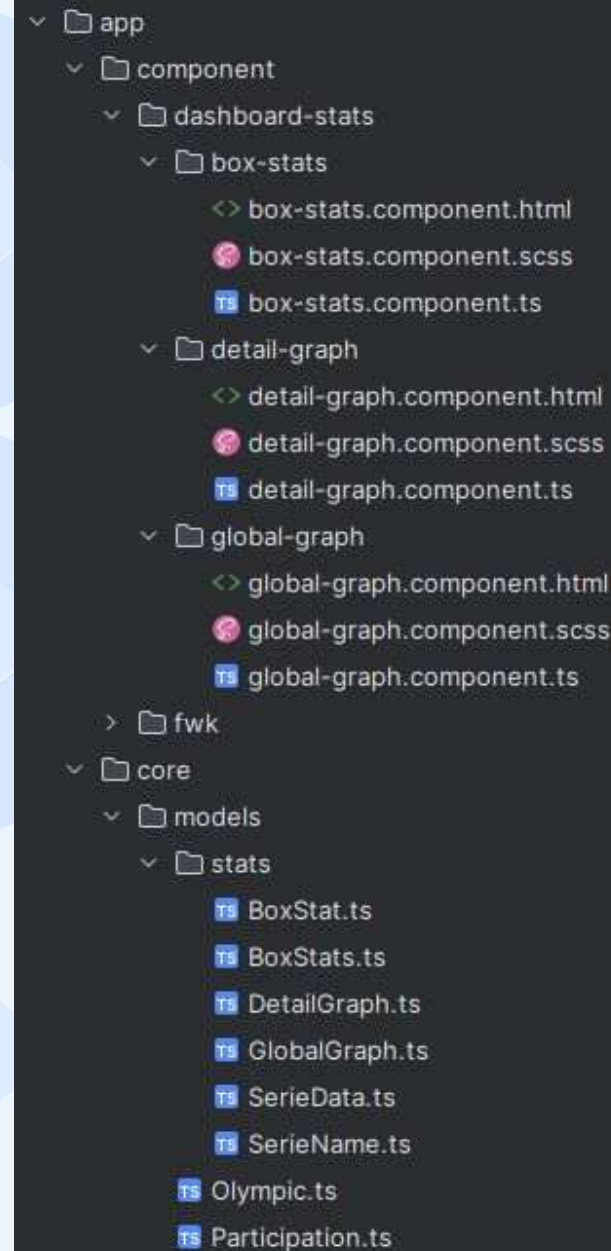
- Racine : configurations, README.md, docs (partie compodoc dans branche dédiée)
- **src** : Points d'entrées
- **src/app** : module racine et routes principales
- **src/app/components** : composants standalone
 - Définir les inputs et outputs pour les rendre faciles à tester
 - Permet de réaliser rapidement une maquette avec des valeurs par défaut
- **src/app/pages** : responsable de la composition et de la logique liée à l'affichage :
 - Instanciation des composants en initialisant les inputs et outputs (événements)
 - Base pour la navigation / routes
- **src/app/core** (partie Model de MVVM) :
 - **services** : pour fournir les données et les logiques de transformation associées
 - **models** : Structures d'échanges entre Angular et le backend

Christophe Pierrès



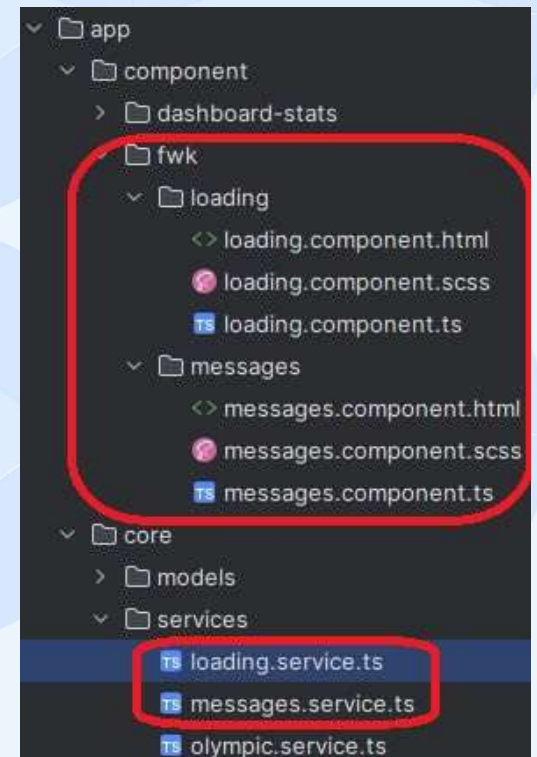
COMPOSANTS

- Composants paramétrables liés au dashboard
 - box-stats (entêtes avec titre et n boites statistiques)
 - `@input() boxstats: BoxStats`
 - global-graph (analyse comparative des pays)
 - `@input() globalGraph: GlobalGraph`
 - `@output() sliceSelected = new EventEmitter<SerieData>()`
 - detail-graph (statistique détaillée d'un pays)
 - `@Input() chartData: DetailGraph`



COMPOSANTS

- Composants transversaux réactifs (+services partagés associés)
 - loading : avec une méthode de service non intrusive sur le flux
 - Messages : pour messages erreurs affichés à l'utilisateur



CONCLUSION

- Respect des principes SOLID
 - SRP : une seule responsabilité ou tâche
 - OCP : Ouvert à l'extension mais fermé à la modification
 - LSP : classes dérivées doivent pouvoir se substituer à leurs classes parentes sans altérer le comportement
 - ISP : limiter l'interfaçage à ce qui est strictement nécessaire, ne pas imposer des interfaces inutiles
 - DIP : Dépendre des abstractions plutôt que des implémentations spécifiques
- A votre disposition pour justifier plus en détail ces principes par rapport au développement

CONCLUSION

- Respect des principes SOLID
 - SRP : une seule responsabilité ou tâche
 - OCP : Ouvert à l'extension mais fermé à la modification
 - LSP : classes dérivées doivent pouvoir se substituer à leurs classes parentes sans altérer le comportement
 - ISP : limiter l'interfaçage à ce qui est strictement nécessaire, ne pas imposer des interfaces inutiles
 - DIP : Dépendre des abstractions plutôt que des implémentations spécifiques
- A votre disposition si vous souhaitez justifier plus en détail ces principes par rapport au développement



Merci à vous




CHRISTOPHE PIERRÈS

CPIERRES@HOTMAIL.COM

07 81 42 54 06

DÉMARCHE POUR LE CHOIX DE LA LIBRAIRIE GRAPHIQUE

- Au préalable, vérifier s'il existe chez le client une charte de développement incluant une librairie graphique préconisée par leur service R&D
- Si non, établir un tableau comparatif puis décider d'un classement en fonction des besoins actuels du projet.

Caractéristique	ngx-charts 	ng2-charts 	ng-chartist 
Basé sur	D3.js	Chart.js	Chartist.js
Facilité d'utilisation	Facile	Facile	Facile
Types de graphiques	Grande variété	Très variée (cartes, chartes en radar, etc.)	Essentiels (ligne, barres, etc.)
Documentation	Complète	Complète et conviviale	Bonne
Performances	Bonnes pour des jeux de données moyens à grands	Excellentes sur tous les types de données	Bonnes pour les petits jeux de données
Thèmes/Styles	Personnalisation via CSS et propriétés Angular	Support des thèmes via Chart.js	Hautement personnalisable avec CSS
Réactivité	Responsive par défaut	Responsive par défaut	Responsive par défaut
Dernière mise à jour	Activement maintenue	Activement maintenue	Moins souvent mise à jour
Communauté	Active	Très active	Plus petite
Intégration	Très facile avec Angular	Facile avec Angular	Facile avec Angular
Licence	MIT	MIT	MIT

DÉMARCHE POUR LE CHOIX DE LA LIBRAIRIE GRAPHIQUE

- critères classiques du tableau comparatif liés à la pérennité (communauté active), à la documentation et au droit (licence MIT)
- meilleure intégration à Angular donne ngx-charts gagnant
 - grâce aux propriétés de style évitant le recours à CSS (réduit le besoin de personnalisation manuelle)
 - s'intègre naturellement dans le concept de liaison de données unidirectionnelle d'Angular (réduit le risque d'erreur et affichage direct via angular à chaque modification)
 - Gestion d'événement selon le paradigme Angular
 - s'intègre bien à la notion de module angular
- Du fait de la conception de nos propres composants qui encapsulent les graphes, on limite les impacts en cas de changement de librairie si souhaité par le client ultérieurement
- Créer une branche par rapport à ma branche de dev afin d'effectuer un essai avec ce premier choix
 - Essai avec ngx-charts rapidement concluant
 - Du coup, dans un souci de timing, pas de raison de tester les autres lib car dans le contexte présent, je suis développeur et ne fais pas partie de la R&D
 - Normalement, seulement après validation du responsable projet, on merge la branche

RÔLE DE CHAQUE COUCHE

- Model (dossier **core** du projet) :
 - Services (sous dossiers core/**services** et core/**models** du projet)
 - Métier(s) :
 - communique avec le Backend (API → JSon)
 - alimente les objets TypeScript (**DTO** / structures de données)
 - Transversaux (partagés/découplés) : loading, message
- View :
 - template HTML
- ViewModel :
 - Classe du component
 - Relie les données (modèles/données) à la Vue (HTML/Template)
 - Interagit avec les services

DÉMARCHE DE DÉVELOPPEMENT

- Créer les composants d'IHM
- Les mettre en place dans les pages (accès aux pages directement via url)
- Gérer la cinématique des écrans
 - mise en place des Routes (chemins les plus spécifiques en premier)
 - Créer une constante AppRoutes pour ne pas référencer en dur les urls si le client souhaite les changer
 - Gérer les IHM de navigation (drill down sur graphique, bouton Retour)
- Mettre en place les méthodes du service pour alimenter nos données statistiques via des Observables. Un BehaviorSubject se comporte comme un cache avec une valeur initiale,
- Finaliser/peaufiner en créant les services partagés pour améliorer la gestion des erreurs et des attentes potentielles (services decouplés de l'IHM)

DÉMARCHE DE DÉVELOPPEMENT

- Code bien documenté (y compris pour des fins pédagogiques)
- Quel exemple de code voulez-vous voir ?
 - Code du service pour mettre en cache, le retour du `http.get`, et d'une manière simple et sécurisée ?
 - Code d'une méthode de service pour, à partir de `olympic$`, transformer/calculer (via Streams) vers la structure attendue par chaque composant, en flux Observable ?
 - Et exploitation dans la page pour transmission au composant via le pipe `async` ?
 - Code des services partagés réactifs ? (découplé de l'IHM => pas de dépendance parent-enfant)
 - `messagesService` (+composant associé) : pour la gestion des messages erreur
 - `loadingService` (+composant associé) : pour message d'attente, lorsqu'il y a attente sur le chargement d'un flux. Ceci d'une manière la moins intrusive possible.
- Ou autres ... (gestion des routes, ...)

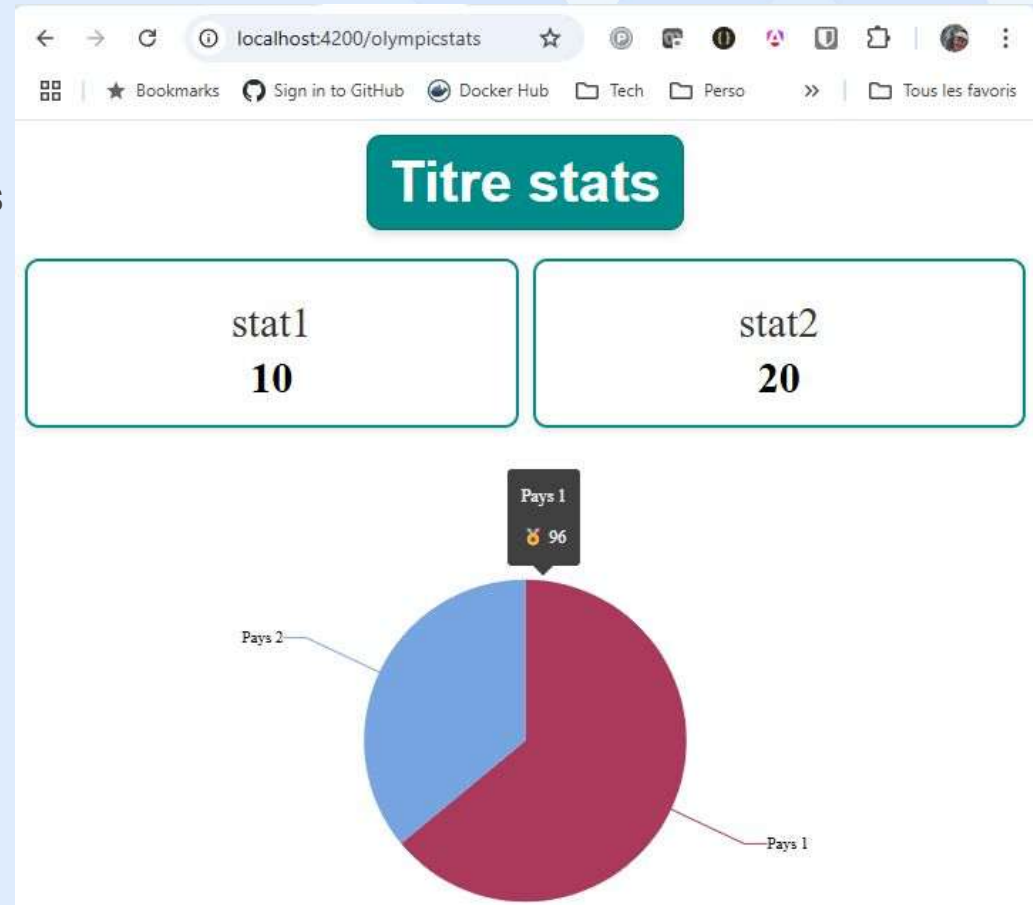
CHOIX DE CONCEPTION

- Produire une maquette fonctionnelle, sans créer le service pour les données
- Faire une abstraction de ce que l'on voit sur les écrans
 - Raisonner « composants » et « structures de données » réutilisables
 - Déterminer pour chaque composant la structure des valeurs nécessaires en input et output
 - Permettra de bien définir ce que devra produire le service fournisseur des données
 - Définir des valeurs par défaut « exemples » pour permettre l'instanciation du composant
- Réaliser une maquette fonctionnelle, avant même d'avoir implémenter les services qui fournissent les données
 - Permet de faire valider par un manager : les besoins, l'ergonomie, l'évolutivité pour répondre à l'actualité olympique ... et la librairie graphique !

CHOIX DE CONCEPTION

- Dans ma page Home, j'appelle mes composants sans paramétrage et ils fonctionnent déjà :

```
<> home.component.html x
1 <!-- h2>Olympic games dashboard (page Home)</h2 -->
2 <app-box-stats/>
3 <app-global-graph/>
```

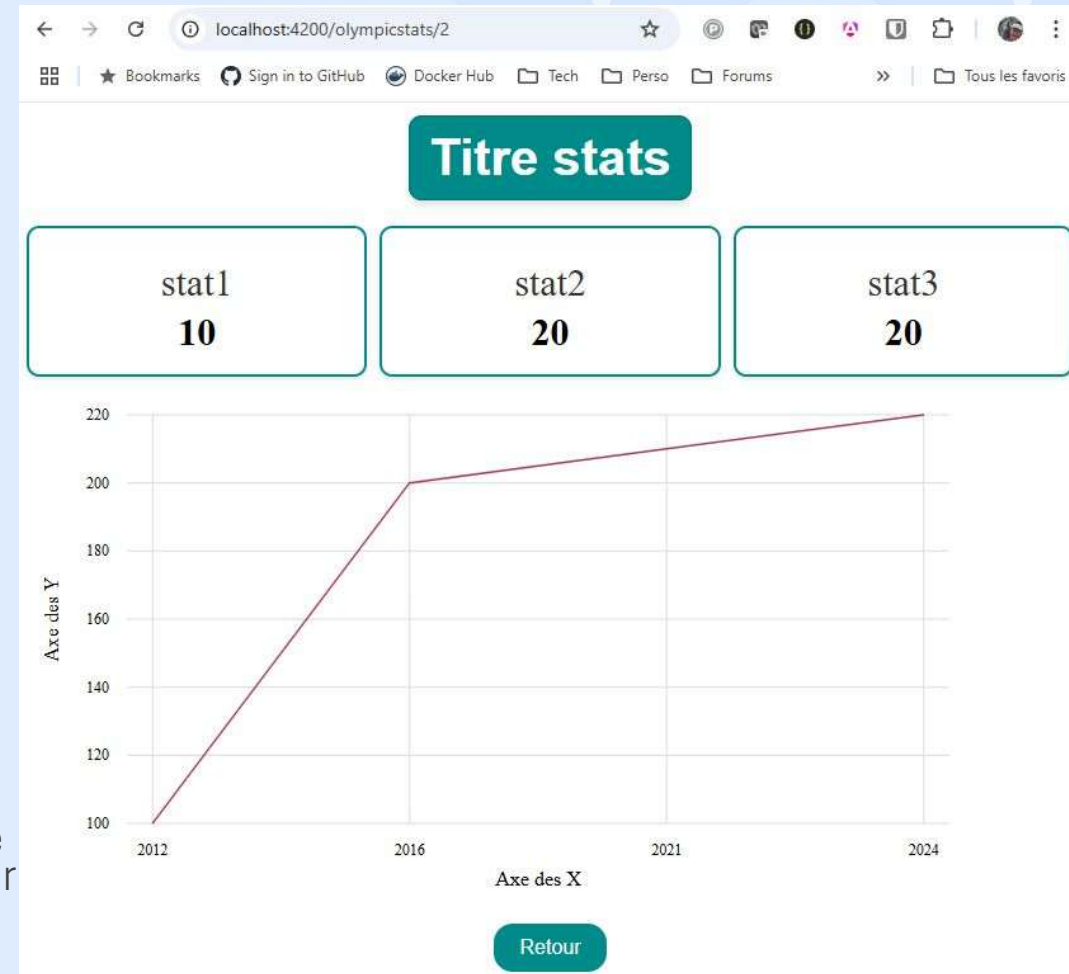


CHOIX DE CONCEPTION

- Dans ma page olympic-country-detail, mes composants sans paramétrage fonctionnent également :

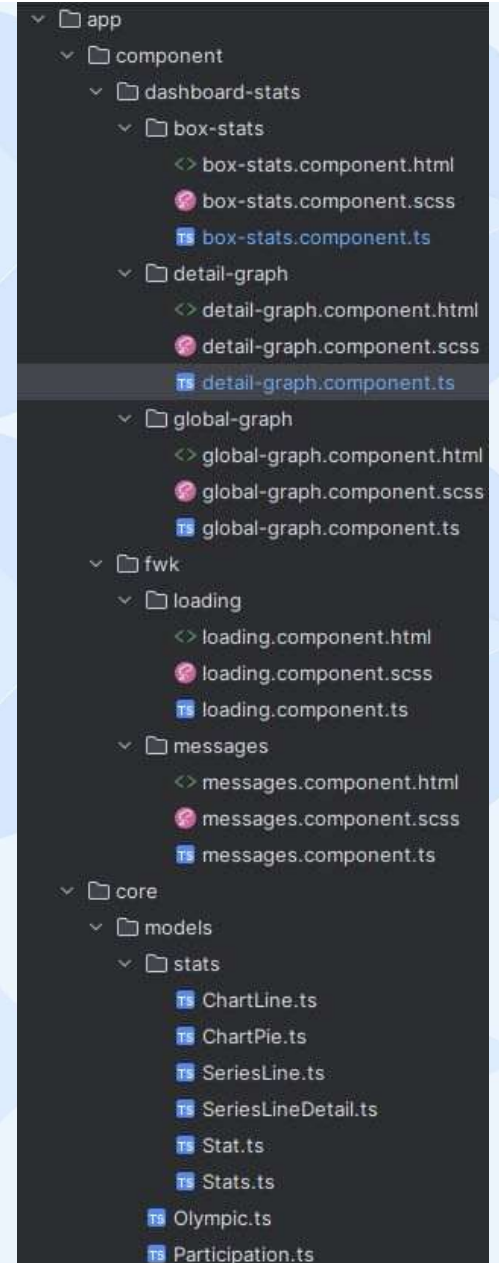
```
<> olympic-country-detail.component.html x
1  <!-- p>olympic-country-detail (page) works!</p -->
2  <app-box-stats/>
3  <app-detail-graph />
4
5  <!-- app-box-stats [boxStats]="boxStats$ | async" ></app-box-stats>
6  <app-detail-graph [chartLineData]="lineChartData$ | async" / -->
7
8  <div class="centered-button">
9    <button class="rounded-button" (click)="goToHome()">Retour</button>
10 </div>
```

- Je peux les tester manuellement en ajoutant des statistiques, en modifiant les titres et libellés, en ajoutant un second graphe, ou en ajoutant une série de ligne au graphe existant (par exemple ligne montrant la moyenne du nombre de médailles pour les autres pays)
- Créer éventuellement des dummy datas



CHOIX DE CONCEPTION

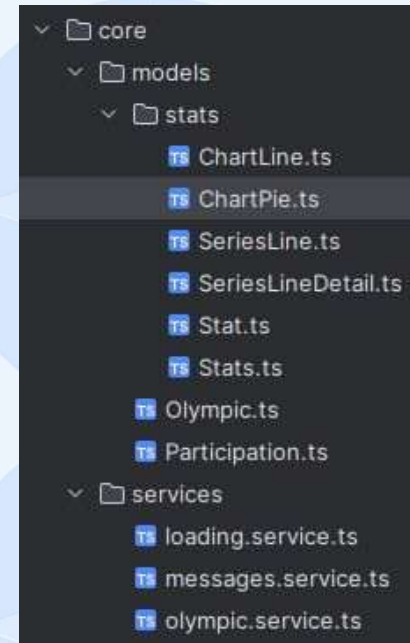
- Composants réutilisables afin d'accélérer la prise en compte de nouveaux besoins liés à l'actualité et assurer une bonne ergonomie
- Composants paramétrables liés au dashboard
 - box-stats (entêtes avec titre et n boites statistiques)
 - `@input() boxstats: Stats`
 - global-graph (analyse comparative des pays)
 - `@input() medalPieData: ChartPie[]`
 - `@output() sliceSelected = new EventEmitter<ChartPieData>()`
 - detail-graph (statistique détaillée d'un pays)
 - `@Input() chartLineData: ChartLine`
- Composants transversaux réactifs (+services partagés associés)
 - loading : avec une méthode de service non intrusive sur le flux
 - Messages : pour messages erreurs affichés à l'utilisateur



CHOIX DE CONCEPTION

- Les noms des interfaces de stats sont génériques et placés dans un répertoire dédié :
- Reflet de ce que demande le composant :
- Et aussi parfois complété d'éléments permettant un meilleur paramétrage :

```
TS ChartPie.ts x
1 export interface ChartPie {
2   name: string;
3   value: number;
4   extra: { id: number };
5 }
```



```
TS ChartLine.ts x TS SeriesLine.ts TS SeriesLineDetail.ts
1 import {SeriesLine} from "../SeriesLine";
2
3 export interface ChartLine {
4   xAxisLabel: string;
5   yAxisLabel: string;
6   seriesLines: SeriesLine[];
7 }

TS ChartLine.ts TS SeriesLine.ts x TS SeriesLineDetail.ts
1 import {SeriesLineDetail} from "../SeriesLineDetail";
2
3 export interface SeriesLine {
4   name: string;
5   series: SeriesLineDetail[];
6 }

TS ChartLine.ts TS SeriesLine.ts TS SeriesLineDetail.ts
1 export interface SeriesLineDetail {
2   name: string;
3   value: number;
4 }
```

CHOIX DE CONCEPTION

- Programmation fonctionnelle et réactive à base d'Observable et BehaviorSubject
 - Pas de blocage de l'IHM, rapidité
 - Permet une actualisation instantannée des composants lors de l'actualisation des flux
 - Mise en place possible de polling, Websockets ou Server-Sent Events pour automatiser
 - Synthétique et facile à comprendre en terme de code
 - Permet un découplage entre le flux et les composants qui observent de leur côté
- Protéger les gestions sensibles de flux au sein du service de fourniture des données
 - Certains observables tels que `BehaviorSubject` ne se terminent pas automatiquement
 - Bien penser à faire un `take (1)` pour clôturer automatiquement l'abonnement une fois la valeur reçue
 - évite d'avoir à gérer manuellement la désinscription (`unsubscribe`) et minimise les risques de fuites de mémoire
 - Rendre privé le `BehaviorSubject`, seul l'Observable correspondant est accessible
 - Note : le pipe `Async` gère le `unsubscribe` automatiquement à la fermeture du composant pour les paramètres Observables de nos composants
- Le `BehaviorSubject` sur la source de données principale sert de cache en conservant toujours la dernière valeur (stateful léger et facile)

SOMMAIRE

- Présentation des livrables et démarche (15 à 20 mn max)
 - Démonstration
 - La démarche d'analyse
 - Les choix de conception
 - La démarche pour le choix de la librairie graphique ←
 - La démarche de développement
- Discussion (10 mn)
 - Questions de Jeannette
- Debrief (5 mn)

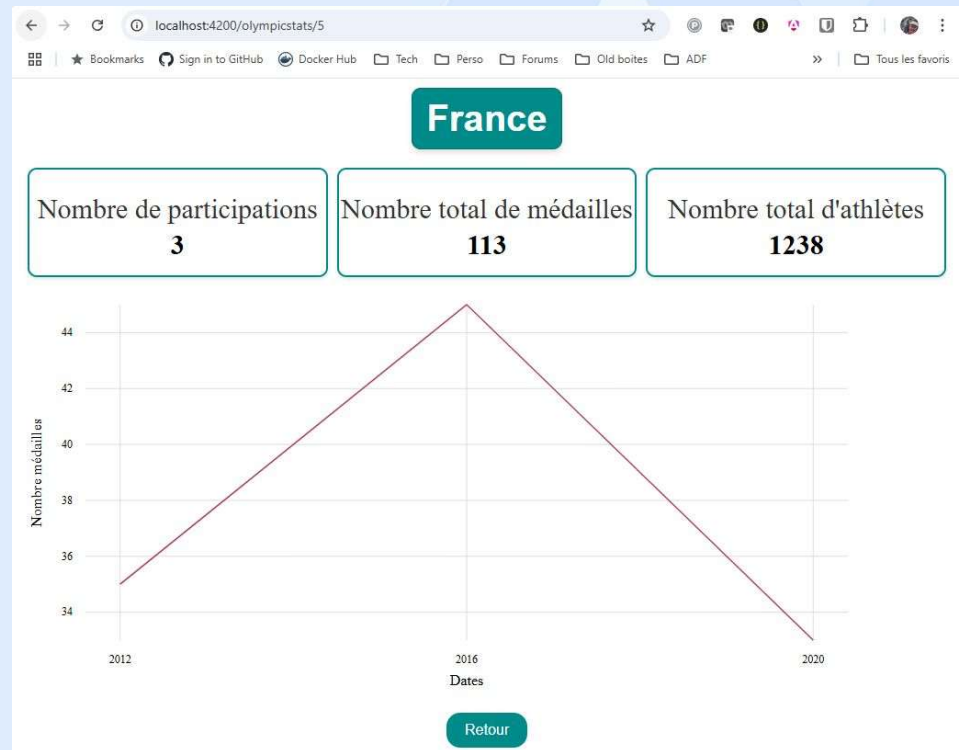
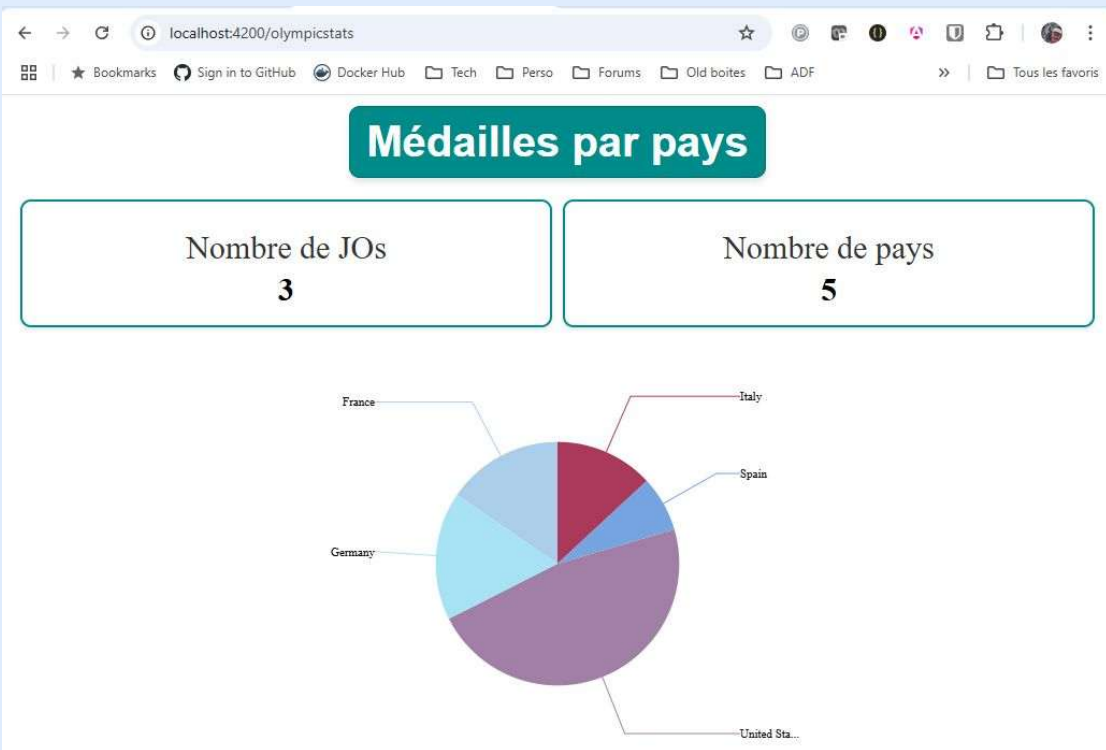
DÉMONSTRATION (FONCTIONNELLE ET TECHNIQUE)

1 - SCÉNARIO NOMINAL DE LA CINEMATIQUE

- ~~Ralentissement volontaire (délai de 2s) de la requête HTTP get initiale~~
 - ~~Affichage automatique d'un composant standalone **loading** (découplé, réactif via BehaviorSubject lié au flux observé, accessible via service partagé)~~
 - ~~Mise en cache du flux http.get Observable via un BehaviorSubject (plus d'accès réseau par la suite ... sauf si demande explicite de Refresh)~~
- Accès à la racine de l'application => route par défaut : home component (=dashboard)
- Clic sur slice de Pie pour aller vers Détail du pays
- Bouton Retour pour revenir au dashboard Home
- Faire plusieurs allers / retours (plus aucun accès réseau)
- Faire une Actualisation F5 du browser (le loading réapparaît)
 - Sur la page Home,
 - Sur la page Detail

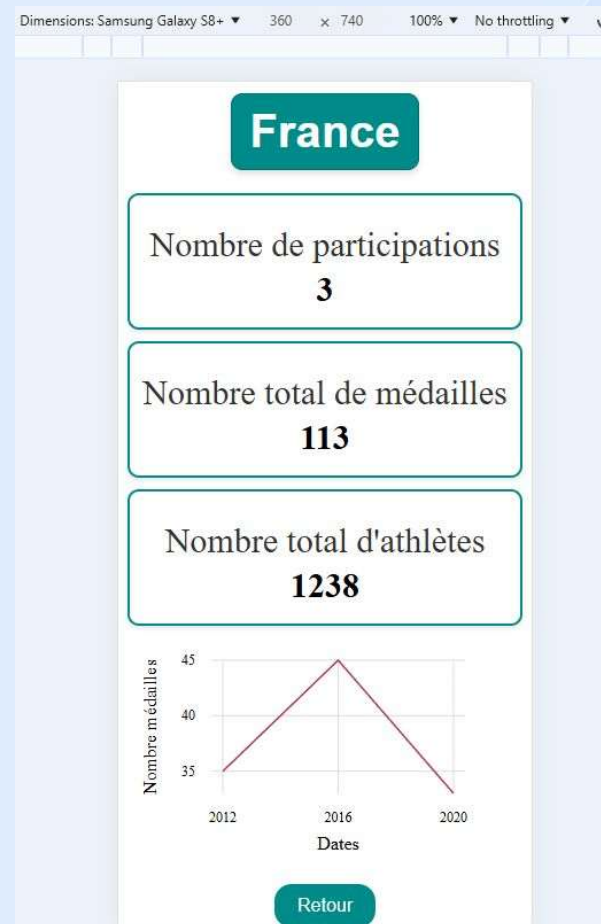
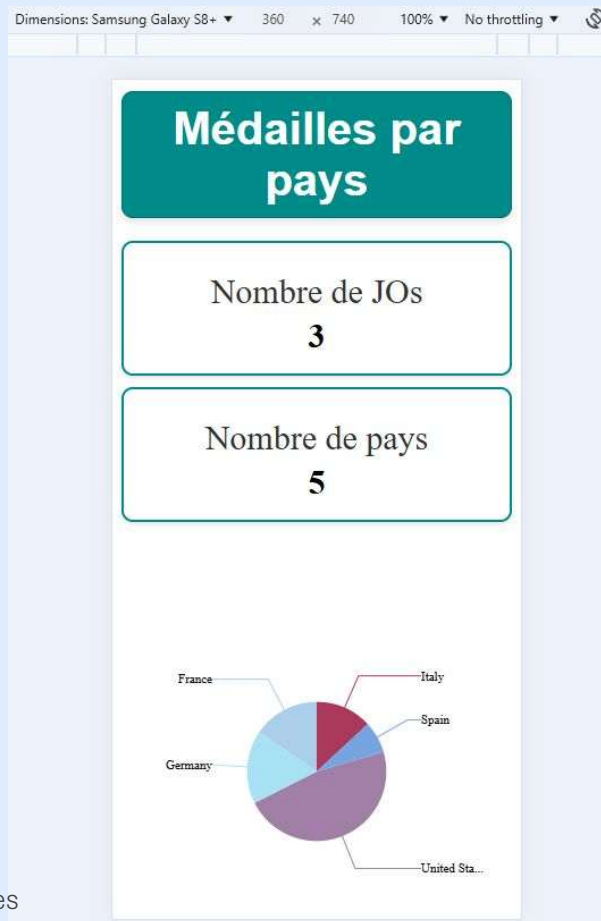
DÉMONSTRATION (FONCTIONNELLE ET TECHNIQUE) 2 – RESPONSIVE SUR ORDINATEUR ET TABLETTE

- Modifier la taille de l'écran pour montrer l'adaptation automatique de la page Home et Détail



DÉMONSTRATION (FONCTIONNELLE ET TECHNIQUE) 2 – RESPONSIVE SUR MOBILE

- Depuis débogueur, choisir affichage Samsung S8 puis S20



DÉMONSTRATION (FONCTIONNELLE ET TECHNIQUE) 3 – SCENARIOS

- SCENARIOS VIA URL
 - url racine : ok (re-direction vers /olympicstats)
 - url olympicstats : ok
 - url olympicstats/id : (avec id existant ok)
 - url : olympicstats/id (avec id non existant) => affichage adéquat



- url avec path inexistant : routage vers NotFound ok

DÉMONSTRATION (FONCTIONNELLE ET TECHNIQUE)

4 – SCENARIOS

- SCENARIOS AVEC ERREUR
 - Introduire une erreur dans le service pour provoquer une erreur 404
 - Messages gérés par un composant messages et un MessagesService partagé (réactif/découplé)

```
68 loadInitialData(): Observable<Olympic[]> { Show usages  cpmminipc
69   console.log('OlympicService.loadInitialData() : appel backend')
70   this.loadingService.loadingOn();
71   //return this.http.get<Olympic[]>('FichierInexistant.json')//pour simuler 404
72   return this.http.get<Olympic[]>(this.olympicUrl)
73   .pipe(
74     //filter(data => data.length > 0),
75     delay(3000), // délai de 3 secondes pour test loading
76     catchError(err: any => {
77       const message = "Impossible de charger les données Olympiques";
78       this.messagesService.showErrors(message);
79       console.error(message, err); //à remplacer par un log serveur (elastik stack, sentry) dans la vraie vie
80       return throwError(err);
81     }),
82     tap(olympics: Olympic[] => {
83       console.log('OlympicService.loadInitialData() tap : data (mise en cache dans le BehaviorSubject)', olympics);
84       this.subject.next(olympics) //dès lors qu'on inscrit le flux Observable de http.get vers le cache
85       //BehaviorSubject, l'observable olympics$ en variable membre est associé à ce sujet
86       //en tant que simple Observable (sans possibilité de le modifier)
87       //Toutes nos méthodes de service se reposent sur olympics$ (non modifiable)
88     }),
89     finalize(): void => this.loadingService.loadingOff()
90   );
91 }
```



- Ajouter un delai pour afficher le composant loading

SOMMAIRE

- Présentation des livrables et démarche (15 mn max)
 - Démonstration
 - La démarche d'analyse
 - Responsabilités des couches / structuration du code
 - Les choix de conception
 - (La démarche pour le choix de la librairie graphique)
- Discussion (10 mn)
 - Questions de Jeannette (jeu de rôles)
- Debrief (5 mn)