Les services angular

La correction du TP en vidéo : la TodoList construite partie par partie (17 min 11)

La vidéo ne s'affiche pas ? La regarder sur YouTube.

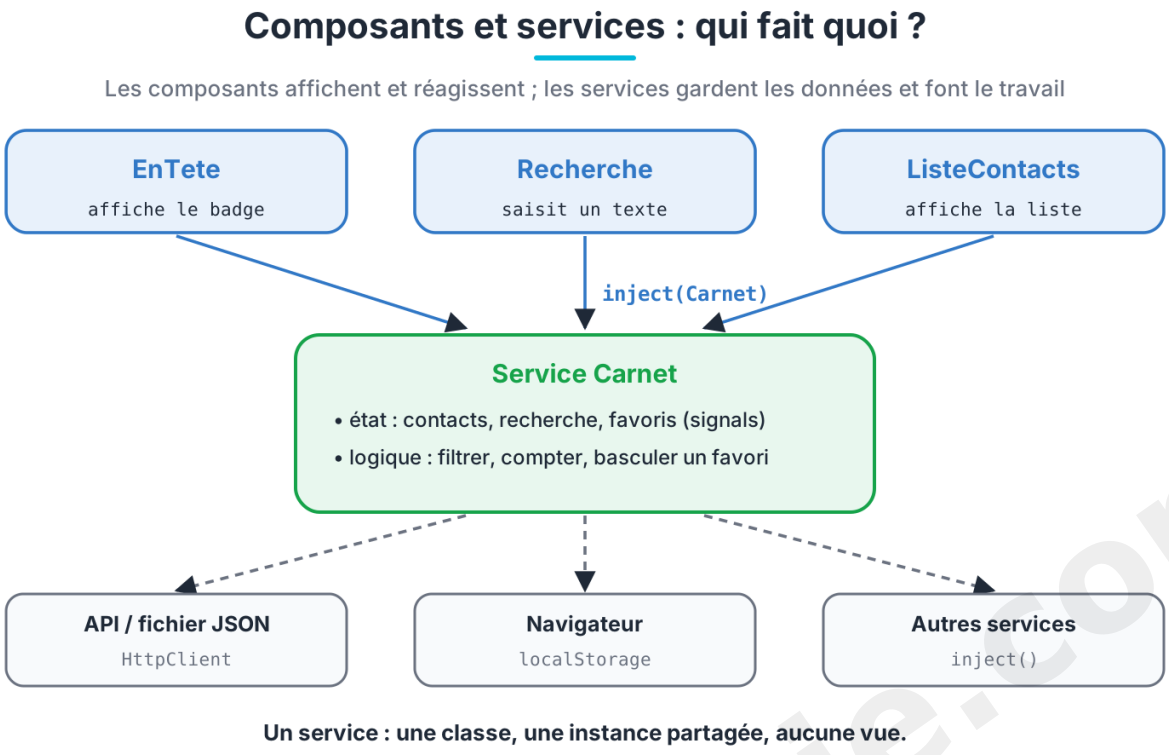
1. Objectifs

- Savoir ce qu'on met dans un **service** et ce qu'on laisse au composant
- Générer un service avec le CLI (`ng g s`) et lire le fichier produit
- Utiliser le décorateur `@Service()` d'Angular 22 et le comparer à `@Injectable()`
- Écrire un **service d'état** avec des signals : état privé, lecture seule, méthodes
- Charger des données avec `HttpClient` dans un service, et gérer le chargement et les erreurs
- Mémoriser un état dans `localStorage`, nettoyer un service avec `DestroyRef`
- Tester un service avec `TestBed.inject()`

1. Présentation

- Le chapitre précédent a présenté le **principe** de l'injection de dépendances : un composant ne crée pas ses outils, il les demande avec `inject()`. Ce chapitre passe à la **pratique**: comment écrire de bons services, ceux que l'on trouve dans toute application Angular.
- Un service est une classe sans vue, dont Angular crée une instance et la distribue aux composants. On y place ce qui n'appartient à aucun écran en particulier : les **données** partagées (un panier, un utilisateur connecté, des contacts), les **accès extérieurs** (une API, `localStorage`) et la **logique** réutilisable (un calcul, une conversion).
- Le composant reste simple : il affiche ce que le service lui donne et lui transmet les actions de l'utilisateur.

apcpedagogie.com



- Pour suivre ce chapitre, continuez dans le projet `premiers-pas-angular` et créez un dossier `src/app/services/`. Tous les exemples ont été testés avec Angular 22.2 (composants `OnPush` par défaut, sans Zone.js).

1. Générer un service avec le CLI

- La commande `ng generate service`, ou en abrégé `ng g s`, crée le service et son fichier de test :

```
$ ng g s carnet
CREATE src/app/carnet.spec.ts (320 bytes)
CREATE src/app/carnet.ts (79 bytes)
```

```
import { Service } from '@angular/core';

@Service()
export class Carnet {}
```

- Depuis Angular 20, le guide de style ne met plus de suffixe : le fichier s'appelle `carnet.ts` et la classe `Carnet`. Avec `--type=service`, on retrouve l'ancien nom : `carnet.service.ts` et

`CarnetService`.

- Angular 22 génère le nouveau décorateur `@Service()`. L'option `--injectable` génère à la place `@Injectable({ providedIn: 'root' })`, l'écriture des versions précédentes.
- Pour ranger les services dans un dossier : `ng g s services/carnet`. Pour ne pas créer le test : `--skip-tests`.

Générer un service avec le CLI

ng generate service (ou ng g s), Angular 22

```
$ ng g s carnet
CREATE src/app/carnet.spec.ts
CREATE src/app/carnet.ts

$ ng g s services/carnet --skip-tests
CREATE src/app/services/carnet.ts
```

```
// src/app/carnet.ts
import { Service } from '@angular/core';

@Service()
export class Carnet {}
```

Option	Effet
`--skip-tests`	pas de fichier de test .spec.ts
`--type=service`	fichier carnet.service.ts, classe CarnetService
`--injectable`	@Injectable({ providedIn: 'root' }) au lieu de @Service()
`--flat=false`	crée un dossier carnet/ pour le service

apcpedagogie.com

1. **Un premier service avec @Service()**
   - `@Service()` marque la classe comme service et la rend disponible dans **toute l'application**, avec une seule instance, créée à la première demande. Les composants la demandent avec `inject()`.
   - Ici, le service `Citations` garde la citation courante. Le bandeau l'affiche, et le parent la fait changer :

```
import { Component, Service, inject, signal } from '@angular/core';

// Un service Angular 22 : @Service() le rend disponible dans toute l'application
@Service()
export class Citations {
  private readonly liste = [
    'Le code le plus simple est celui que l’on n’écrit pas.',
    'Un composant affiche, un service travaille.',
```

```
    'Mieux vaut un petit service clair qu'un gros composant confus.',
  ];
  private readonly indice = signal(0);

  courante(): string {
    return this.liste[this.indice()];
  }

  suivante(): void {
    this.indice.update((i) => (i + 1) % this.liste.length);
  }
}

@Component({
  selector: 'app-bandeau-citation',
  template: `<p>« {{ citations.courante() }} »</p>`,
})
export class BandeauCitation {
  protected readonly citations = inject(Citations);
}

@Component({
  selector: 'app-premier-service-srv',
  imports: [BandeauCitation],
  template: `
    <app-bandeau-citation />
    <button (click)="citations.suivante()">Citation suivante</button>
  `,
})
export class PremierServiceSrv {
  protected readonly citations = inject(Citations); // la même instance que le bandeau
}
```

```
« Le code le plus simple est celui que l'on n'écrit pas. »
[Citation suivante]
(1 clic)
« Un composant affiche, un service travaille. »
```

- Le bandeau et son parent reçoivent **la même instance**: quand le parent appelle `suivante()`, le bandeau se met à jour, sans entrée ni sortie.
- `courante()` lit le signal `indice`: le template qui l'appelle se met donc à jour, même en `OnPush`.

apcpedagogie.com

1. ## @Service() ou @Injectable() ?

    - `@Service()` est apparu avec Angular 22. Il fait la même chose que `@Injectable({ providedIn: 'root' })`, en plus court, avec deux différences :
        - l'option `autoProvided: false` crée un service qui n'est **pas** disponible partout : il faut le fournir dans les `providers` d'un composant (c'est l'équivalent de `@Injectable()` sans `providedIn`) ;
        - un `@Service` reçoit ses dépendances **uniquement** avec `inject()`: l'injection par le constructeur est refusée à la compilation.

```
import { Component, Injectable, Service, inject, input, signal } from '@angular/core';

// Les deux écritures fournissent une instance unique pour toute l'application
@Service()
export class Horloge {
  heure(): string {
    return new Date().toLocaleTimeString('fr-FR', { hour: '2-digit', minute: '2-digit' });
  }
}

@Injectable({ providedIn: 'root' })
export class HorlogeAncienne {
  heure(): string {
    return new Date().toLocaleTimeString('fr-FR', { hour: '2-digit', minute: '2-digit' });
  }
}

// autoProvided: false : le service n'est disponible que là où on le fournit
@Service({ autoProvided: false })
export class Brouillon {
  readonly texte = signal('');
}

@Component({
  selector: 'app-editeur-brouillon',
  providers: [Brouillon], // une instance par éditeur
  template: `
    <p><strong>{{ titre() }}</strong>
      <input [value]="brouillon.texte()" (input)="brouillon.texte.set(champ.value)" #champ />
      {{ brouillon.texte().length }} caractère(s)</p>
  `,
})
```

apcpedagogie.com

```
export class EditeurBrouillon {
  readonly titre = input('Brouillon');
  protected readonly brouillon = inject(Brouillon);
}

@Component({
  selector: 'app-service-ou-injectable',
  imports: [EditeurBrouillon],
  template: `
    <p>@Service : {{ horloge.heure() }} · @Injectable : {{ ancienne.heure() }}</p>
    <app-editeur-brouillon titre="Message 1" />
    <app-editeur-brouillon titre="Message 2" />
  `,
})
export class ServiceOuInjectable {
  protected readonly horloge = inject(Horloge);
  protected readonly ancienne = inject(HorlogeAncienne);
}
```

```
@Service : 09:52 · @Injectable : 09:52
Message 1 [Bonjour]  7 caractère(s)
Message 2 [       ]  0 caractère(s)      ← chaque éditeur a son brouillon
```

- Un constructeur avec paramètre dans un `@Service` ne compile pas : `NG2028: @Service class cannot use constructor dependency injection. Use the `inject` function instead.`
- Un service `autoProvided: false` demandé hors d'un composant qui le fournit : `NG0201: No provider found for `Brouillon`.`
- `@Injectable` reste valable et très présent dans le code existant ; les deux écritures se mélangent sans problème dans un même projet.

apcpedagogie.com

@Service() ou @Injectable() ?

Deux décorateurs pour la même idée ; @Service() est l'écriture d'Angular 22

	@Service()	@Injectable()
Écriture	`@Service()`	`@Injectable({ providedIn: 'root' })`
Disponible partout	oui, par défaut	avec providedIn: 'root'
Seulement via providers	`@Service({ autoProvided: false })`	`@Injectable()`
Dépendances	inject() uniquement	inject() ou constructeur
constructor(x: X)	`erreur NG2028`	accepté
Généré par ng g s	oui (Angular 22)	avec --injectable
Code existant	—	toujours valable

1. **Un service d'état avec les signals**
   - Le cas le plus courant : un service qui **garde des données** partagées par plusieurs composants. On l'écrit toujours de la même façon :
     - un signal **privé** contient les données ;
     - il est exposé en **lecture seule** avec `asReadonly()`, et les valeurs dérivées avec `computed()` ;
     - des **méthodes** sont le seul moyen de modifier les données.

```
import { Component, Service, computed, inject, signal } from '@angular/core';

export interface Message {
  id: number;
  texte: string;
  lu: boolean;
}

// Un service d'état : les données privées, une lecture seule, des méthodes pour les modifier
@Service()
export class Messagerie {
  private readonly _messages = signal<Message[]>([]);
  readonly messages = this._messages.asReadonly();
  readonly nonLus = computed(() => this._messages().filter((m) => !m.lu).length);
  private suivant = 1;
```

apcpedagogie.com

```
  envoyer(texte: string): void {
    const t = texte.trim();
    if (t) this._messages.update((liste) => [...liste, { id: this.suivant++, texte: t, lu: false
}]);
  }

  toutLire(): void {
    this._messages.update((liste) => liste.map((m) => ({ ...m, lu: true })));
  }
}

@Component({
  selector: 'app-badge-messages',
  template: `<span class="badge">✉ {{ messagerie.nonLus() }} non lu(s)</span>`,
  styles: `.badge { background: #dd0031; color: #fff; border-radius: 12px; padding: 2px 10px;
font-size: 14px; }`,
})
export class BadgeMessages {
  protected readonly messagerie = inject(Messagerie);
}

@Component({
  selector: 'app-envoi-message',
  template: `
    <input #champ placeholder="Votre message" (keyup.enter)="envoyer(champ)" />
    <button (click)="envoyer(champ)">Envoyer</button>
  `,
})
export class EnvoiMessage {
  private readonly messagerie = inject(Messagerie);

  envoyer(champ: HTMLInputElement): void {
    this.messagerie.envoyer(champ.value);
    champ.value = '';
  }
}

@Component({
  selector: 'app-liste-messages',
  template: `
    <ul>
      @for (m of messagerie.messages(); track m.id) {
        <li [class.lu]="m.lu">{{ m.id }}. {{ m.texte }}</li>
      } @empty {
```

apcpedagogie.com

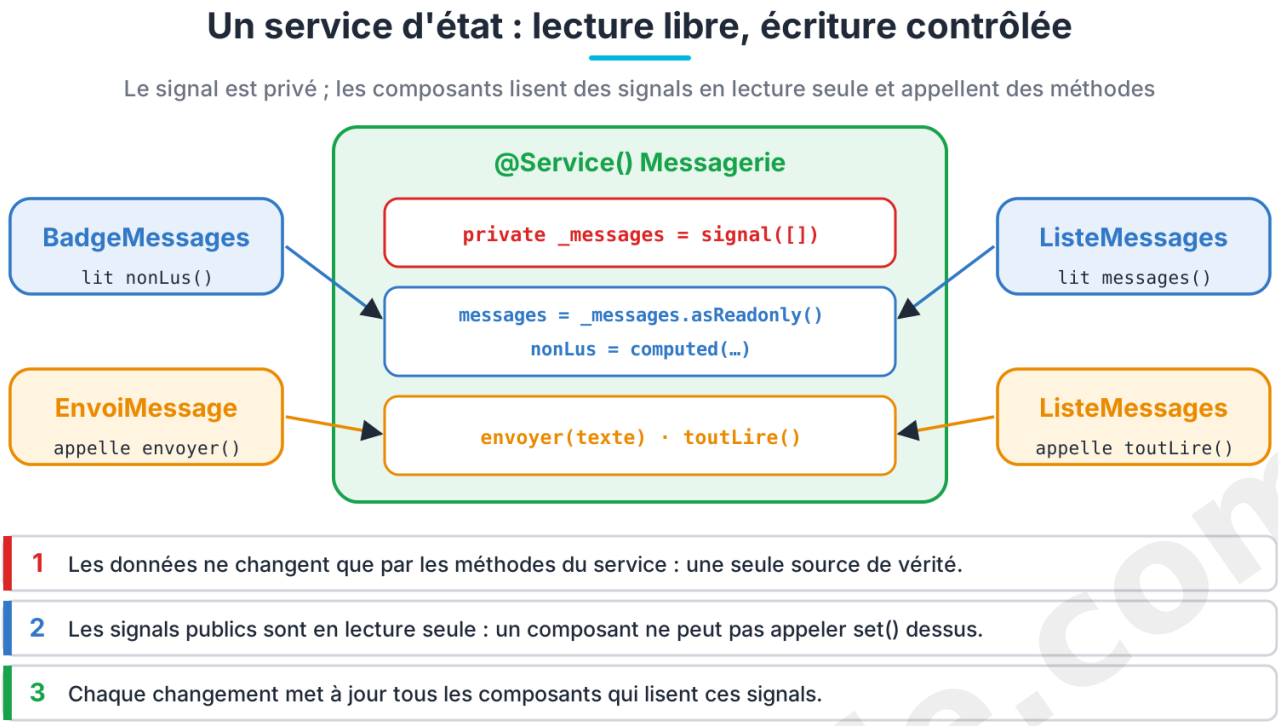
```
        <li>Aucun message.</li>
      }
    </ul>
    <button (click)="messagerie.toutLire()" [disabled]="!messagerie.nonLus()">Tout marquer comme
lu</button>
  `,
  styles: `.lu { color: #94a3b8; }`,
})
export class ListeMessages {
  protected readonly messagerie = inject(Messagerie);
}

@Component({
  selector: 'app-etat-partage',
  imports: [BadgeMessages, EnvoiMessage, ListeMessages],
  template: `
    <p>Messagerie <app-badge-messages /></p>
    <app-envoi-message />
    <app-liste-messages />
  `,
})
export class EtatPartage {}
```

```
Messagerie ✉ 0 non lu(s)       Aucun message.
(« Réunion à 10 h » + Envoyer, « Rendre le TP » + Entrée, « ␣␣␣ » + Envoyer)
Messagerie ✉ 2 non lu(s)       1. Réunion à 10 h   2. Rendre le TP
(Tout marquer comme lu)
Messagerie ✉ 0 non lu(s)       (les messages passent en gris)
```

- Trois composants indépendants partagent le même état : le badge, la saisie et la liste ne se connaissent pas.
- La règle « un message vide est ignoré » est écrite **une seule fois**, dans `envoyer()`. Aucun composant ne peut la contourner : `messages` n'a pas de méthode `set()`.
- `EnvoiMessage` injecte le service en `private`: il ne s'en sert que dans sa classe. Les autres l'injectent en `protected`, car leur template l'utilise.

apcpedagogie.com



1. Un service de données

- Un service est l'endroit idéal pour charger des données : le composant n'a pas à savoir **d'où** elles viennent (un fichier, une API, un cache). Le service utilise `HttpClient`, qu'il obtient lui aussi avec `inject()`.
- Dans Angular 22, `HttpClient` est disponible sans configuration. `provideHttpClient()`, dans `app.config.ts`, ne sert qu'à le configurer (par exemple `withFetch()` ou des intercepteurs).
- Le fichier `stagiaires.json` est placé dans le dossier `public/`: il est servi à l'adresse `/stagiaires.json`.

```
import { Component, Service, computed, inject, signal } from '@angular/core';
import { DecimalPipe } from '@angular/common';
import { HttpClient } from '@angular/common/http';

export interface Stagiaire {
  id: number;
  nom: string;
  ville: string;
  moyenne: number;
}

// Un service de données : il sait OÙ et COMMENT charger, le composant se contente d'afficher
```

apcpedagogie.com

```
@Service()
export class Stagiaires {
  private readonly http = inject(HttpClient);
  private readonly _liste = signal<Stagiaire[]>([]);
  private readonly _etat = signal<'chargement' | 'pret' | 'erreur'>('chargement');
  readonly liste = this._liste.asReadonly();
  readonly etat = this._etat.asReadonly();
  readonly admis = computed(() => this._liste().filter((s) => s.moyenne >= 10).length);

  charger(url = 'stagiaires.json'): void {
    this._etat.set('chargement');
    this.http.get<Stagiaire[]>(url).subscribe({
      next: (liste) => {
        this._liste.set(liste);
        this._etat.set('pret');
      },
      error: () => this._etat.set('erreur'),
    });
  }
}

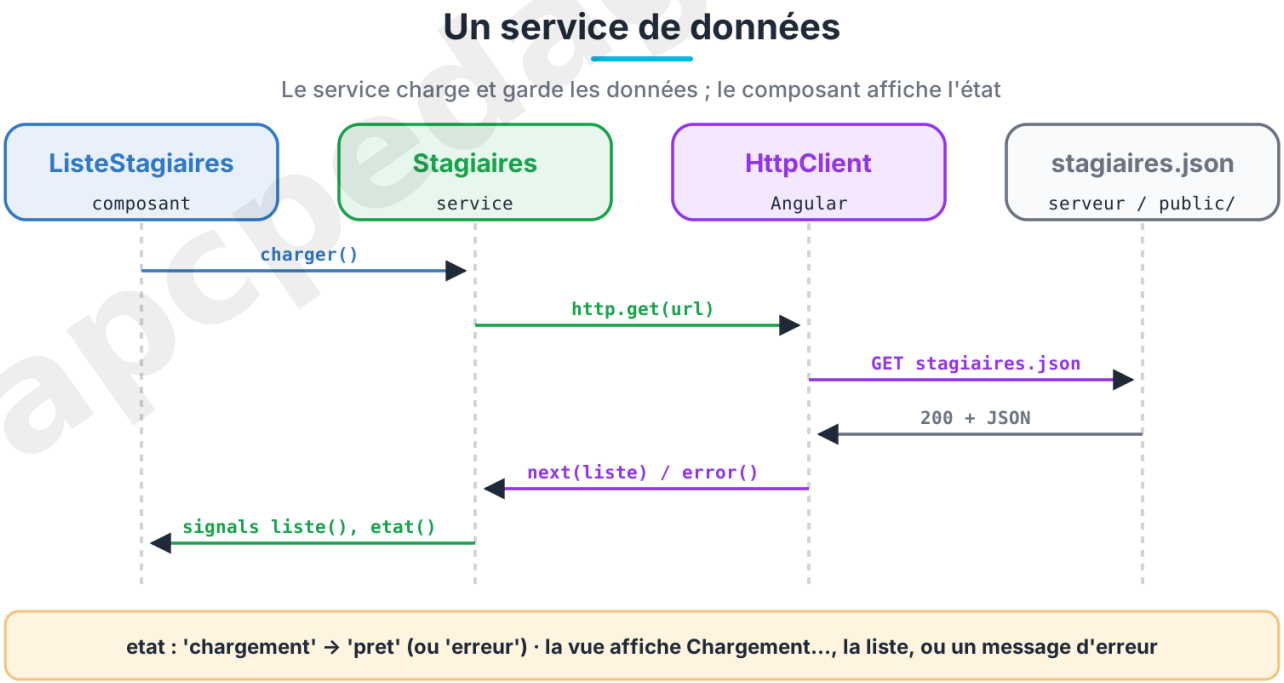
@Component({
  selector: 'app-liste-stagiaires',
  imports: [DecimalPipe],
  template: `
    @switch (stagiaires.etat()) {
      @case ('chargement') { <p>Chargement…</p> }
      @case ('erreur') { <p class="erreur">Impossible de charger la liste.</p> }
      @case ('pret') {
        <ul>
          @for (s of stagiaires.liste(); track s.id) {
            <li>{{ s.nom }} ({{ s.ville }}) : {{ s.moyenne | number: '1.2-2' }}</li>
          }
        </ul>
        <p>{{ stagiaires.admis() }} admis sur {{ stagiaires.liste().length }}</p>
      }
    }
    <button (click)="stagiaires.charger()">Recharger</button>
    <button (click)="stagiaires.charger('inconnu.json')">Tester une erreur</button>
  `,
  styles: `.erreur { color: #dc2626; }`,
})
export class ListeStagiaires {
  protected readonly stagiaires = inject(Stagiaires);
```

apcpedagogie.com

```
  constructor() {
    this.stagiaires.charger();
  }
}
```

```
Amira Ben Ali (Sousse) : 16,25
Sami Trabelsi (Tunis) : 9,50
…
Karim Mansour (Tunis) : 12,25
5 admis sur 6
(Tester une erreur)
Impossible de charger la liste.            ← la requête vers inconnu.json renvoie 404
```

- Le service garde aussi l'**état du chargement** (`'chargement'`, `'pret'`, `'erreur'`) : le template choisit l'affichage avec `@switch`.
- `http.get()` renvoie un `Observable`: la requête ne part qu'au `subscribe()`. Oublier `subscribe()` est une erreur classique : aucune requête, et la liste reste vide.
- Le service fournit les données et leur logique (`admis`) ; le composant ne fait qu'afficher et déclencher `charger()`.



- Angular 22 propose aussi `httpResource()`, qui gère lui-même l'état du chargement. Le service devient très court ; le chapitre consacré à HTTP le détaillera.

apcpedagogie.com

```
import { Component, Service, computed, inject } from '@angular/core';
import { httpResource } from '@angular/common/http';
import { Stagiaire } from './donnees';

// La même chose avec httpResource : l'état de chargement est fourni par Angular
@Service()
export class StagiairesResource {
  readonly liste = httpResource<Stagiaire[]>(() => 'stagiaires.json', { defaultValue: [] });
  readonly admis = computed(() => this.liste.value().filter((s) => s.moyenne >= 10).length);
}

@Component({
  selector: 'app-resume-stagiaires',
  template: `
    @if (stagiaires.liste.isLoading()) {
      <p>Chargement…</p>
    } @else if (stagiaires.liste.error()) {
      <p>Impossible de charger la liste.</p>
    } @else {
      <p>{{ stagiaires.liste.value().length }} stagiaires, {{ stagiaires.admis() }} admis</p>
    }
    <button (click)="stagiaires.liste.reload()">Recharger</button>
  `,
})
export class ResumeStagiaires {
  protected readonly stagiaires = inject(StagiairesResource);
}
```

```
6 stagiaires, 5 admis
```

1. **Un service qui se souvient**
   - Un service peut aussi se charger de la **persistance**: il relit `localStorage` à sa création, et un `effect()` enregistre la valeur à chaque changement. Les composants n'en savent rien.

```
import { Component, Service, effect, inject, signal } from '@angular/core';

export type Taille = 'petite' | 'normale' | 'grande';

// Un service qui se souvient : il relit localStorage à sa création et l'écrit à chaque
changement
```

apcpedagogie.com

```
@Service()
export class Preferences {
  private readonly cle = 'preferences-cours';
  private readonly _taille = signal<Taille>(this.lire());
  readonly taille = this._taille.asReadonly();

  constructor() {
    effect(() => localStorage.setItem(this.cle, this._taille()));
  }

  choisir(taille: Taille): void {
    this._taille.set(taille);
  }

  private lire(): Taille {
    const valeur = localStorage.getItem(this.cle);
    return valeur === 'petite' || valeur === 'grande' ? valeur : 'normale';
  }
}

@Component({
  selector: 'app-choix-taille',
  template: `
    @for (t of tailles; track t) {
      <button [class.choisi]="preferences.taille() === t" (click)="preferences.choisir(t)">{{ t
}}</button>
    }
  `,
  styles: `.choisi { background: #16a34a; border-color: #16a34a; }`,
})
export class ChoixTaille {
  protected readonly preferences = inject(Preferences);
  protected readonly tailles: Taille[] = ['petite', 'normale', 'grande'];
}

@Component({
  selector: 'app-article-lisible',
  template: `<p [class]="preferences.taille()">Ce texte garde sa taille, même après un
rechargement de la page.</p>`,
  styles: `.petite { font-size: 13px; } .normale { font-size: 16px; } .grande { font-size: 21px;
}`,
})
export class ArticleLisible {
  protected readonly preferences = inject(Preferences);
```

apcpedagogie.com

```
}

@Component({
  selector: 'app-persistance',
  imports: [ChoixTaille, ArticleLisible],
  template: `<app-choix-taille /><app-article-lisible />`,
})
export class Persistance {}
```

```
[petite] [normale] [grande]        Ce texte garde sa taille…   (normale)
(clic sur « grande »)             localStorage : preferences-cours = "grande"
(rechargement de la page)         la taille « grande » est conservée
```

- `effect()` s'appelle dans le constructeur du service : c'est un contexte d'injection, comme pour `inject()`.
- La méthode privée `lire()` vérifie la valeur enregistrée : une valeur inattendue (modifiée à la main, ancienne version) donne la valeur par défaut.

1. La durée de vie d'un service

- Un service vit aussi longtemps que l'**injecteur** qui l'a créé. Un service racine vit jusqu'à la fermeture de l'application. Un service fourni par un composant est créé avec ce composant et **détruit avec lui**.
- S'il a lancé quelque chose (un intervalle, un abonnement, un écouteur), il doit l'arrêter en partant : `inject(DestroyRef).onDestroy()` enregistre ce nettoyage.

```
import { Component, DestroyRef, Service, inject, signal } from '@angular/core';

// Un service fourni par un composant naît et meurt avec lui
@Service({ autoProvided: false })
export class Minuteur {
  private static suivant = 1;
  readonly numero = Minuteur.suivant++;
  readonly secondes = signal(0);

  constructor() {
    const id = setInterval(() => this.secondes.update((s) => s + 1), 1000);
    console.log(`Minuteur n° ${this.numero} créé`);
    inject(DestroyRef).onDestroy(() => {
      clearInterval(id); // sinon l'intervalle continuerait de tourner
```

apcpedagogie.com

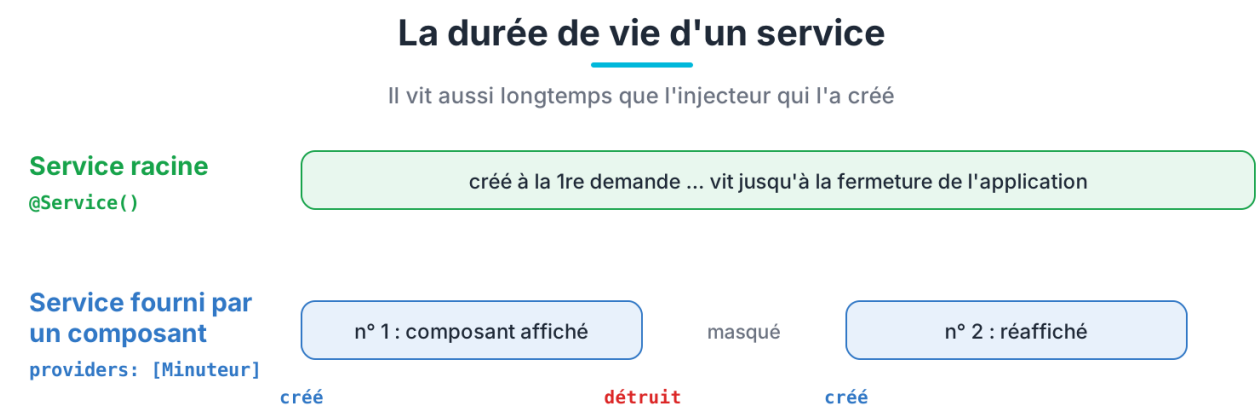
```
      console.log(`Minuteur n° ${this.numero} détruit`);
    });
  }
}

@Component({
  selector: 'app-panneau-minuteur',
  providers: [Minuteur],
  template: `<p>Minuteur n° {{ minuteur.numero }} : {{ minuteur.secondes() }} s</p>`,
})
export class PanneauMinuteur {
  protected readonly minuteur = inject(Minuteur);
}

@Component({
  selector: 'app-duree-de-vie',
  imports: [PanneauMinuteur],
  template: `
    <button (click)="visible.set(!visible())">{{ visible() ? 'Masquer' : 'Afficher' }} le
minuteur</button>
    @if (visible()) {
      <app-panneau-minuteur />
    }
  `,
})
export class DureeDeVie {
  protected readonly visible = signal(true);
}
```

```
Minuteur n° 1 : 6 s
console : Minuteur n° 1 créé
(Masquer)     console : Minuteur n° 1 détruit
(Afficher)    console : Minuteur n° 2 créé       Minuteur n° 2 : 0 s
```

apcpedagogie.com




```
const id = setInterval(() => …, 1000);
inject(DestroyRef).onDestroy(() => clearInterval(id));   // nettoyer en partant

// Sans ce nettoyage, l'intervalle continuerait de tourner après la destruction.
```

- Sans `clearInterval`, le minuteur n° 1 continuerait de compter en mémoire après sa destruction, et chaque affichage en ajouterait un nouveau.
- Un service peut aussi implémenter `ngOnDestroy()`, avec le même effet ; `DestroyRef` permet de placer le nettoyage juste à côté de ce qu'il arrête.

1. Tester un service

- Un service n'a pas de vue : on le teste directement, sans composant. `TestBed.inject()` demande l'instance à l'injecteur de test, comme `inject()` dans l'application.
- C'est un grand avantage de placer la logique dans les services : elle se teste en quelques lignes.

```
import { TestBed } from '@angular/core/testing';
import { Messagerie } from './etat-partage';

describe('Messagerie', () => {
  let messagerie: Messagerie;

  beforeEach(() => {
    messagerie = TestBed.inject(Messagerie); // le service, sans aucun composant
  });

  it('commence sans message', () => {
    expect(messagerie.messages()).toEqual([]);
```

apcpedagogie.com

```
    expect(messagerie.nonLus()).toBe(0);
  });

  it('ignore un message vide', () => {
    messagerie.envoyer('   ');
    expect(messagerie.messages().length).toBe(0);
  });

  it('compte puis lit les messages', () => {
    messagerie.envoyer('Bonjour');
    messagerie.envoyer('Rendre le TP');
    expect(messagerie.nonLus()).toBe(2);
    messagerie.toutLire();
    expect(messagerie.nonLus()).toBe(0);
  });
});
```

```
$ ng test --include src/app/services/etat-partage.spec.ts
 ✓ Messagerie > commence sans message
 ✓ Messagerie > ignore un message vide
 ✓ Messagerie > compte puis lit les messages

 Test Files  1 passed (1)
      Tests  3 passed (3)
```

1. Que mettre dans un service ?

- Une règle simple : le composant s'occupe de l'**affichage** et des **interactions**; tout le reste peut aller dans un service.

Dans le composant	Dans un service
Le template, les styles	Les données partagées entre plusieurs composants
L'état propre à l'affichage (un menu ouvert, un onglet choisi)	Les appels HTTP et l'accès à `localStorage`
Les réactions aux clics et aux saisies	Les règles métier (valider, calculer, filtrer)
Les entrées et les sorties	Les traitements réutilisables (conversion, formatage)

- Un service ne manipule pas le DOM et ne connaît pas les composants qui l'utilisent.
- Un service = une responsabilité : `Carnet`, `Session`, `Theme` plutôt qu'un énorme `DonneesService`.
- Un service peut injecter d'autres services : `Carnet` peut utiliser `HttpClient` et `Session`.

apcpedagogie.com

1. **Aide-mémoire**

Aide-mémoire : les services Angular

Générer un service	`ng g s carnet (--skip-tests, --type=service)`
Service partagé (Angular 22)	`@Service() export class Carnet {}`
Écriture classique	`@Injectable({ providedIn: 'root' })`
Un service par composant	`@Service({ autoProvided: false }) + providers`
Utiliser un service	`private readonly carnet = inject(Carnet);`
État privé	`private readonly _liste = signal<T[]>([]);`
Exposer en lecture	`readonly liste = this._liste.asReadonly();`
Valeur dérivée	`readonly nb = computed(() => this._liste().length);`
Charger des données	`provideHttpClient() + inject(HttpClient).get<T>(url)`
Mémoriser	`effect(() => localStorage.setItem(cle, valeur()))`
Nettoyer	`inject(DestroyRef).onDestroy(() => …)`
Tester	`TestBed.inject(Carnet)`
Constructeur avec @Service	`NG2028 : utiliser inject()`

1. **Activité 1 : un thème clair ou sombre**

   - Créer le service `Theme` (`@Service()`) : un signal privé `mode` (`'clair'` ou `'sombre'`) exposé en lecture seule, un `computed` `estSombre` et une méthode `basculer()`.
   - Le mode est relu dans `localStorage` (clé `theme`) à la création du service, et enregistré à chaque changement avec un `effect()`.
   - Trois composants l'injectent : `BoutonTheme` (« ◫ Mode sombre » ou « ☀ Mode clair »), `CarteLecon` (fond foncé en mode sombre) et `PiedTheme` (« Thème actuel : … »). La page change aussi de fond.

apcpedagogie.com



- Vérifier : passer en mode sombre, recharger la page : le mode sombre est conservé.
- Pourquoi le bouton n'a-t-il besoin d'aucune sortie pour prévenir la carte et le pied de page ?

1. **Activité 2 : un carnet de contacts**
    - Placer `contacts.json` (6 contacts : `id`, `nom`, `ville`, `telephone`) dans `public/`.
    - Créer le service `Carnet`: il charge les contacts avec `HttpClient` dans son constructeur et garde l'état du chargement, le texte de recherche et la liste des favoris (des identifiants). Il expose `resultats` (les contacts dont le nom ou la ville contient le texte, sans tenir compte des majuscules), `nbFavoris`, et les méthodes `chercher()`, `estFavori()` et `basculerFavori()`.
    - Trois composants l'injectent : `BarreRecherche` (un champ), `ListeContacts` (« Chargement des contacts… », « Impossible de charger les contacts. », le nombre de résultats, une étoile ☆/★ par contact, « Aucun contact ne correspond. ») et `CompteurFavoris` (« ★ 2 favori(s) »).



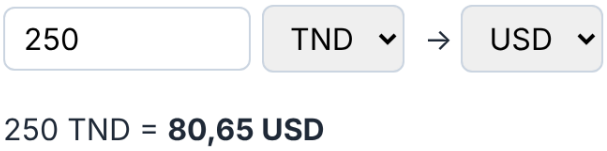
- Vérifier : « sou » → 3 contacts (deux à Sousse, et Karim Mansour) ; « tun » → 2 ; « zz » → « Aucun contact ne correspond. ». Les favoris sont conservés pendant la recherche.

apcpedagogie.com

- Changer l'adresse en `contact.json`: quel message s'affiche ?
- Cette activité est corrigée pas à pas dans la vidéo de la leçon.

1. **Activité 3 : un convertisseur testé**

    - Créer le service `Convertisseur`, sans état : la valeur de chaque devise en dinars (TND 1, EUR 3,4, USD 3,1 ; des taux fictifs) et une méthode `convertir(montant, de, vers)` qui arrondit au centime et renvoie `null` pour un montant négatif ou invalide.
    - Le composant `ConvertisseurDevises` affiche un champ, deux listes de devises et le résultat (pipe `number`), ou « Montant invalide. ».
    - Écrire `convertisseur.spec.ts` avec trois tests : 100 EUR = 340 TND, 10 TND = 3,23 USD, et un montant négatif donne `null`. Les lancer avec `ng test`.

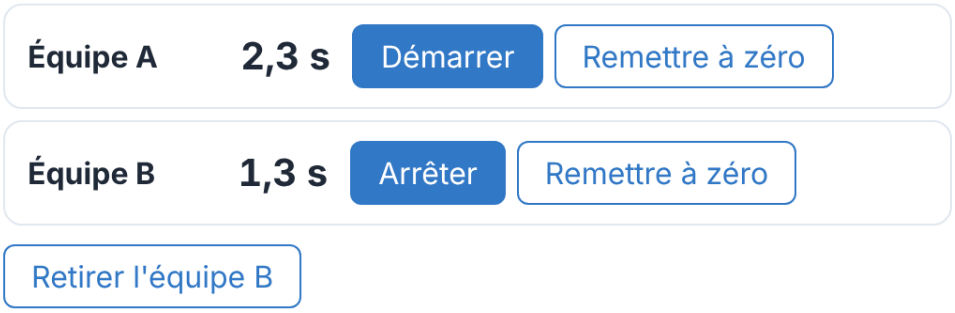


- Vérifier : 250 TND = 80,65 USD ; −3 → « Montant invalide. » ; `ng test` affiche 3 tests réussis.
- Pourquoi est-il plus simple de tester `Convertisseur` que le composant ?

1. **Activité 4 : des chronomètres indépendants**

    - Créer le service `Chronometre` avec `@Service({ autoProvided: false })`: un signal de dixièmes de seconde, `enMarche`, un `computed affichage` (« 2,3 s ») et les méthodes `demarrer()`, `arreter()` et `remettreAZero()`.
    - À sa destruction, le service arrête son intervalle et écrit « Chronomètre détruit » dans la console (`DestroyRef`).
    - `PanneauChrono` fournit le service et affiche le temps ; son enfant `CommandesChrono` l'injecte sans le fournir et affiche « Démarrer » ou « Arrêter » et « Remettre à zéro ». La page affiche deux panneaux (Équipe A, Équipe B) et un bouton pour retirer ou rajouter l'équipe B.

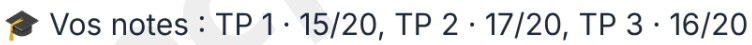
apcpedagogie.com



- Vérifier : les deux chronomètres comptent séparément ; retirer l'équipe B affiche « Chronomètre détruit » ; la rajouter la fait repartir de 0,0 s.
- Que se passe-t-il si l'on écrit `@Service()` et que l'on retire les `providers`?

1. **Activité 5 : une session simulée**
    - Créer le service `Session`: un signal privé `utilisateur` (`string | null`) en lecture seule, un `computed estConnecte`, `connecter(nom)` (refuse un prénom de moins de 2 caractères et renvoie `false`) et `deconnecter()`.
    - Trois composants : `EnteteSession` (« Bonjour Amira » et « Se déconnecter », ou « Non connecté »), `FormulaireConnexion` (champ et bouton, visible seulement hors connexion, message d'erreur) et `EspacePerso` (les notes, ou « ◫ Connectez-vous pour voir vos notes. »).





- Vérifier : « A » → message d'erreur ; « Amira » + Entrée → en-tête et notes ; « Se déconnecter » → retour au formulaire.
- Dans une vraie application, que faudrait-il changer dans `Session` pour vérifier un mot de passe ? Les composants devraient-ils changer ?

- Les corrigés des cinq activités, testés avec Angular 22, sont fournis dans l'archive de la leçon (dossier `corriges/`).

apcpedagogie.com

1. Erreurs fréquentes

Message ou symptôme	Cause	Solution
`NG2028: @Service class cannot use constructor dependency injection. Use the `inject` function instead.`	`constructor(private http: HttpClient)` dans un `@Service`	`private readonly http = inject(HttpClient);`
`NG0201: No provider found for `Brouillon`.`	Service `autoProvided: false` sans `providers` sur le chemin	`providers: [Brouillon]` sur le composant, ou `@Service()`
`NG0203: … `inject()` function must be called from an injection context …`	`inject()` dans une méthode du service	Injecter dans un champ
La liste reste vide, aucune requête dans l'onglet Réseau	`http.get()` sans `subscribe()`	S'abonner, ou utiliser `httpResource()`
« Impossible de charger » (erreur 404)	Fichier absent de `public/` ou mauvaise adresse	Vérifier le nom et l'emplacement du fichier
Deux composants n'affichent pas les mêmes données	`new Carnet()`, ou service aussi déclaré dans des `providers`	`inject(Carnet)`, retirer le `providers`
La vue ne se met pas à jour	Données du service dans une propriété ordinaire	Utiliser `signal` et `computed`
Un composant modifie les données sans passer par le service	Signal public modifiable	Signal privé + `asReadonly()` + méthodes
Le minuteur continue après la disparition du composant	Intervalle jamais arrêté	`inject(DestroyRef).onDestroy(() => clearInterval(id))`
Valeur incohérente après rechargement	Valeur de `localStorage` utilisée sans vérification	Valider la valeur relue, prévoir une valeur par défaut

1. À retenir

- Un service est une classe sans vue qui garde les données partagées, les accès extérieurs et la logique réutilisable ; le composant affiche et réagit.
- `ng g s carnet` crée `carnet.ts` avec `@Service()`, le décorateur d'Angular 22, équivalent de `@Injectable({ providedIn: 'root' })`.
- `@Service({ autoProvided: false })` + `providers`: une instance par composant, détruite avec lui.

apcpedagogie.com

- Un `@Service` reçoit ses dépendances avec `inject()`, jamais par le constructeur (NG2028).
- Le modèle d'un service d'état : signal privé, `asReadonly()`, `computed()` et méthodes.
- Un service de données gère aussi l'état du chargement et les erreurs ; `http.get()` ne part qu'au `subscribe()`.
- Un service qui lance quelque chose le nettoie avec `DestroyRef` ; il se teste simplement avec `TestBed.inject()`.

> *Des composants légers et des services bien découpés : c'est la base d'une application Angular facile à lire, à tester et à faire évoluer.*

Sommaire du cours Angular

Source : https://apcpedagogie.com/les-services-angular · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com