Coaching
Angular 11+

Sommaire

1. Communication entre composants
    1. Rappel du pattern observer (Observable)
    2. Exemple de cas d'usage / exercice sur les observables
2. Architecture / séparation des responsabilités
    1. Utiliser les principes de l'archi hexagonale
    2. Découpage et Factorisation des composants
3. TypeScript
    1. Nouveautés
    2. Rappel de l'utilisation des promesses
    3. Exercice pour traiter des promesses en parallèle
4. Javascript : Opérations
5. Test unitaire
    1. Exercice mise en place d'un TU sur un service
6. Thème

Communication entre composants

Communication entre composants

Découpage des composants

- Architecturer son application en composant
  - Utiliser le découpage Atomic Design

- Source : https://openclassrooms.com/fr/courses/5249021-initiez-vous-a-la-methode-atomic-design/5630171-decouvrez-l-atomic-design

Communication entre composants

Les différents mécanismes

- Angular propose plusieurs moyens de communication entre composants
- Les mécanismes doivent s'utiliser selon des cas bien précis
  - Communication synchrone entre composant Parent → c. enfant
    - A l'initialisation le composant Parent transmets des paramètres aux composants enfants
  - Communication entre composant Enfant → c. Parent
    - Suite à une action le composant enfant émet un paramètre de sortie (intercepté par le c. Parent)
  - Communication suite à un événement entre composants indépendants
    - Suite à un évènement asynchrone ou synchrone les composants abonnés observe la mise à jour du sujet (données dynamiques)
  - Communication suite à une action Parent → c. enfant
    - Suite à une action synchrone le composant parent utilise une référence du composant parent pour interagir avec les membres publiques du composant enfant

Communication entre composants

Communication synchrone entre composant Parent → c. enfant

- Utilisation des paramètres d'entrés
- L'annotation @Input() permet de récupérer la valeur des paramètres à l'initialisation du composant :

Composant parent

```
<div class="itemProduit" *ngFor="let item of list; let index = index;">
    <app-produit-item idProduit="{{index}} " description="{{item}}"></app-produit-item>
</div>
```

Passage des paramètres lors de la définition dans le composant parent

```
<div *ngFor="let p of produits" class="container-produit">
  <app-produit [produit]="p"></app-produit>
</div>
```

Dans le cas d'un objet en paramètre d'entrée

Composant enfant

```
export class ProduitItemComponent implements OnInit {

  @Input()
  public idProduit = '';

  @Input()
  public description = '';

  constructor() { }
```

Accès aux paramètres du composant enfant

Communication entre composants

Communication entre composant Enfant → c. Parent

- Utilisation du paramètre **Output**
- L'annotation @**Ouput**() permet de remonter une valeur sur le composant appelant (parent) :

Composant enfant

```
@Output()
public nombreArticle: EventEmitter<number> = new EventEmitter<number>();

public panierProduits: string[] = [];

private panierSubscription: Subscription | undefined;

constructor(private panierService: PanierService) { }

public nombreArcticleChanged() {
  const nb = 5;
  this.nombreArticle.emit(nb);
}
```

Déclaration du paramètre de sortie dans le composant app-produit-item.

Emettre la valeur via la paramètre de sortie.

Composant parent

```
<app-produit-item idProduit="{{index}} " description="{{item}}"
  (nombreArticle)="ajouterLesArticles($event)"></app-produit-item>
```

On récupère la valeur du paramètre de sortie du composant par une callback (<ajouterLesArticles>). Le paramètre **$event** contient la valeur typé.

```
public ajouterLesArticles(nombreArticle: number) {
    console.log(nombreArticle) → 5
}
```

Communication entre composants

Communication suite à un événement entre composants indépendants

- Partage de données asynchrones entre plusieurs composants
- Observer Pattern : RxJs
  - Cas d'utilisation :
    - un composant d'une page attend des données en asynchrone pour les affichés dans son template. Depuis un autre composant, un évènement déclenchera de façon différé la mise à jour des données.
    - A l'initialisation du composant, on sollicite le service pour récupérer des données asynchrones.
    - Le composant s'abonne à un sujet (observable) du service → sujet.subscribe((data)=>{})
    - Lorsqu'un événement différé veut partager une mise à jour des données, il utilise le service contenant le sujet pour émettre une diffusion aux composants abonnés → service.emitNewData(newData)
    - Quand les données sont récupérées par le service, celui-ci averti les abonnées → sujet.next(data).

Le Framework

Communication suite à un évènement entre composants indépendants

- Exemple d'implémentation

Composant abonné

```
export class JournalComponent implements OnInit {
  // Abonnement du journal
  private journalSubscription: Subscription;
...
  ngOnInit() {
    // On s'abonne au sujet (observable) pour rafraîchir automatiquement le composant
    this.journalSubscription = this.journalService.journalSubject.subscribe((journal: string[]) => {
      this.journal = journal;
    });
  }
...
  ngOnDestroy() {
    // A la destruction du composant on se désabonne pour éviter les fuites mémoires.
    this.journalSubscription.unsubscribe();
  }
```

Service contenant le sujet (Observer)

```
export class JournalService {

  // Observable du Journal (sujet)
  public journalSubject = new Subject<string[]>();

...
  /**
   * Diffusion du Journal aux subscribers (abonnés).
   */
  emitModificationSubject(dataWS: string[]) {
    this.journalSubject.next(dataWS);
  }
```

Composant émetteur

```
export class CmsComponent implements OnInit {
...
  updateJournal() {
    // Evénement asynchrone
    this.journalWebService.getJournal(idJournal).subscribe((journal: string[]) => {
      // Emet la mise à jour
      this.journalService.emitModificationSubject(journal);
    });
  }
```

Le Framework

Communication suite à un évènement entre composants indépendants

- Exemple d'implémentation avec désynchronisation des composants à l'instanciation.

Composant abonné : instancier en second

```
export class JournalComponent implements OnInit {
  // Abonnement du journal
  private journalSubscription: Subscription;

  constructeur(private journalService: JournalService) {}
  ...

  ngOnInit() {
    this.journal = this.journalService.journal;
    // On s'abonne au sujet (observable) pour rafraîchir automatiquement le composant
    this.journalSubscription = this.journalService.journalSubject.subscribe((journal: string[]) => {
      this.journal = journal;
    });
  }
  ...
  ngOnDestroy() {
    // A la destruction du composant on se désabonne pour éviter les fuites mémoires.
    this.journalSubscription.unsubscribe();
  }
```

Service contenant le sujet (Observer)

```
export class JournalService {

  // Observable du Journal (sujet)
  public journalSubject = new Subject<string[]>();
  private journal: string[] = [];

...
  /**
   * Diffusion du Journal aux subscribers (abonnés).
   */
  emitModificationSubject(dataWS: string[]) {
    this.journal = dataWS;
    this.journalSubject.next(dataWS);
  }
```

Composant émetteur : instancier en premier

```
export class CmsComponent implements OnInit {
...
  ngOnInit() {
      // Emet la mise à jour
      this.journalService.emitModificationSubject(journal);
  }
```

Communication entre composants

Communication suite à une action Parent → c. enfant

- L'annotation @ViewChild permet d'accéder aux membres et méthodes publiques d'un composant enfant :

Composant parent

```
<app-notification #notif1></app-notification>
```

Identification dans le template html

Composant parent

```
@ViewChild('notif1')
notification!: NotificationComponent;
```

Déclaration dans le composant parent

```
this.getStatus().then((status: string) => {
  this.status = status;
  this.notification.message = this.status;
});
```

Accès aux membres du composant enfant

Le Framework

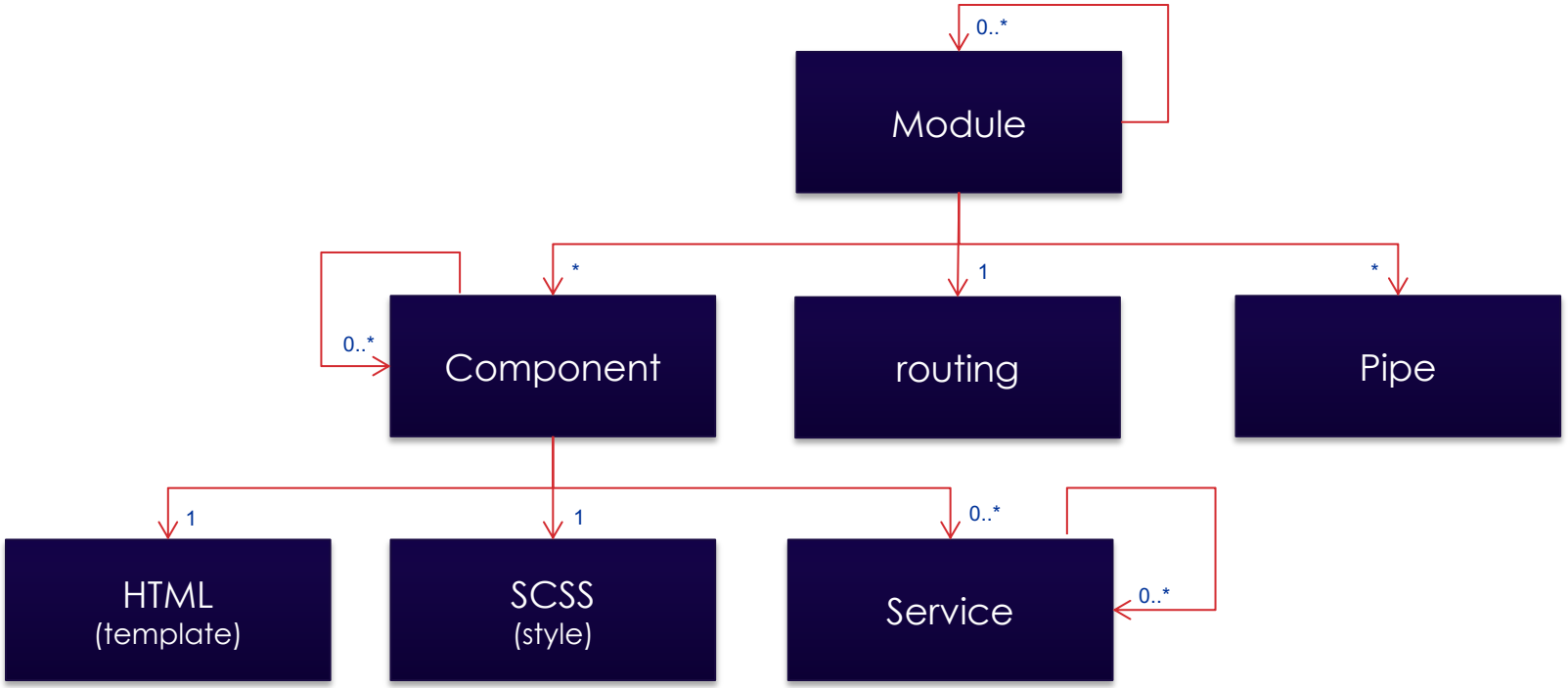
Module

- Générateur Angular :
  - ng g module path/monModule
    - Génération du module qui va encapsuler les composants de la fonctionnalité
  - ng g component path/moduleXXX/monComponent
    - Génération du composant (html, classe, style, test unitaire) + intégration dans le module
  - ng g service path/monService
    - Génération de la classe de service (singleton)
  - ng g module customers --route customers --module app.module
    - Génération d'un composant de type page avec le routing module et la déclaration de la route dasn app-routing.module
- Module Material.Angular (Composants UI) :
  - https://material.angular.io/components/categories

Architecture / séparation des responsabilités

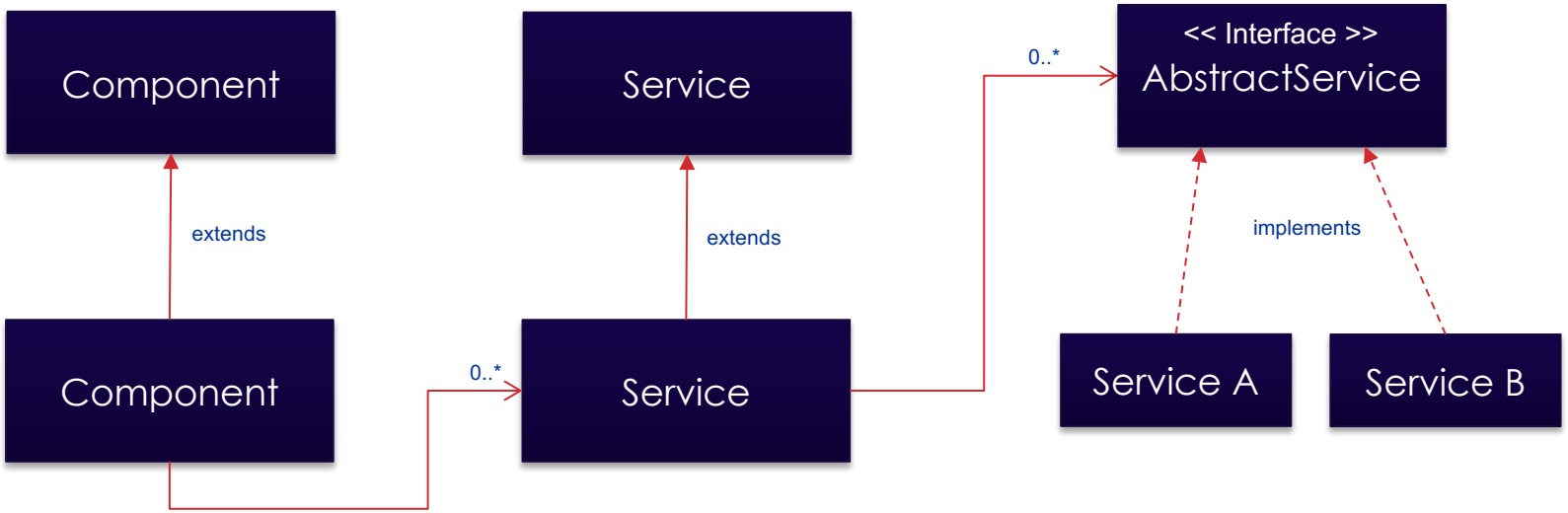
Le Framework

Composition



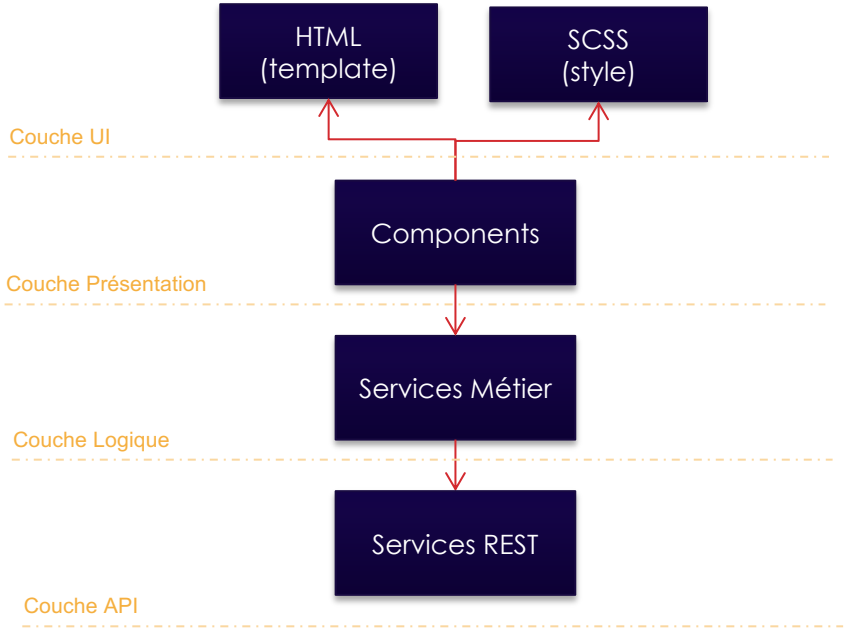
Le Framework

Composition, héritage, interface



Le Framework

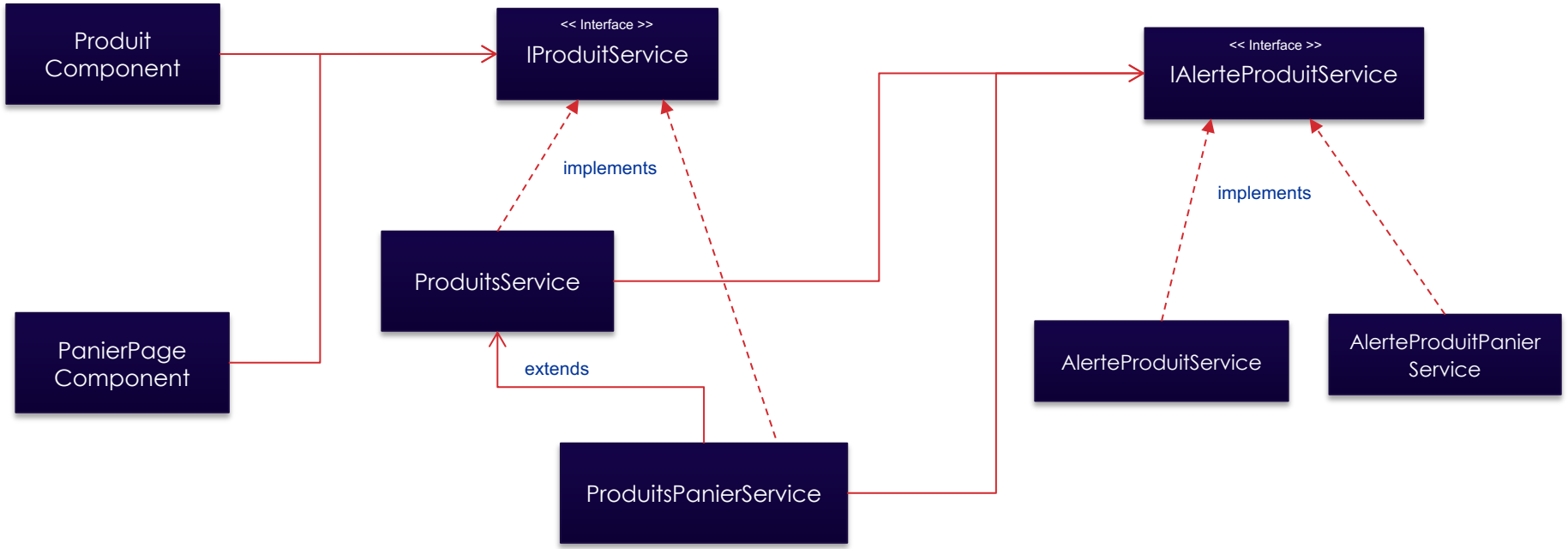
Les principes de l'architecture en couche



Le Framework

Composition, héritage, interface

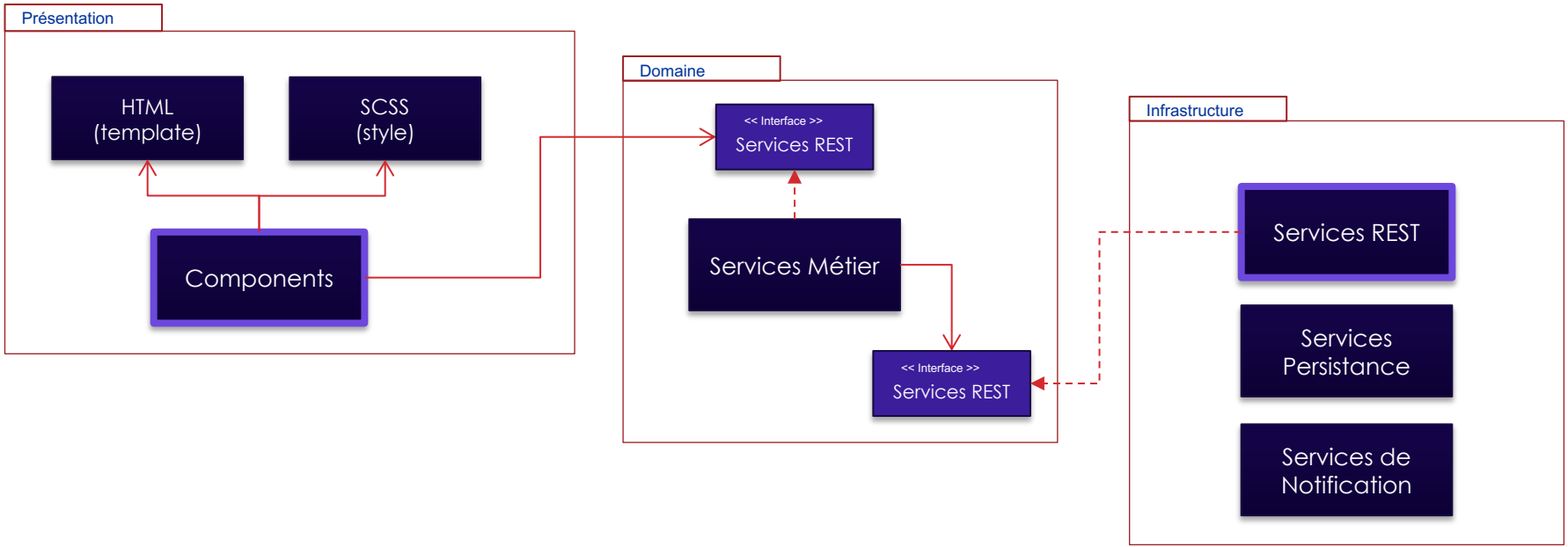
Diagramme du TP Partie 3



Le Framework

Les principes de l'architecture hexagonale

- Séparation en logique (applicative, métier et infrastructure)
- Couplage faible (communication au travers des ports et adapters)



TypeScript

Le langage TypeScript

Template littéral

- Simplification de la concaténation de chaine dynamique

```
const myFunction = (name: string, date: string, temperature: number): string => {
  const message = `Bonjour ${name} !
  Nous sommes le ${date}
  La température est de ${temperature}°C`;


  return message;
}
```

Le langage TypeScript

Opérateur « spread »

- Avant ES 6

```
var params = [ "hello", true, 7 ];
var other = [ 1, 2 ].concat(params);
```

- A partir ES 6

```
const params = [ "hello", true, 7 ]
const other = [ 1, 2, ...params ]
```

Fusionner des objets

```
const obj1 = {
            name: 'table',
            id: 1
};

const obj2 = {
            id: 1,
            description: 'rectangulaire'
}

const objTable = { ...obj1, ...obj2 };
console.log(objTable);
=> { description: "rectangulaire", id: 1, name: "table" }
```

Le langage TypeScript

Déstructuration

- Avant ES 6

```
var myObject = {
prop1: 1,
prop2: { a: 1, b: 2 }, prop3: 3
};

var prop1 = myObject.prop1;
var a = myObject.prop2;
var myProp3 = myObject.prop3;
```

- A partir ES 6

```
const myObject = {
            prop1: 1,
            prop2: { a: 1, b: 2 },
            prop3: 3
};

const {
            prop1,
            prop2 : { a, b },
            prop3: myProp3,
} = myObject;
```

Les promesses

Implémentation

- Promise
  - Une promesse désigne une donnée qui est fabriquée de manière asynchrone et sera disponible dans le futur : « résolue » si traitement OK, « rejetée » si traitement KO.
  - Alternative élégante aux callbacks d'avant ES6
  - Possibilité d'enchaîner ou de combiner des promesses « Promise.all ».

```
// implementation
public authenticate (user, password) {
  return new Promise((resolve, reject) => {
    …
    resolve(…)
    …
    reject(…)
    …
  })
}
```

```
const promise1 = Promise.resolve(3);
const promise2 = functionPromise(); // 42
const promise3 = new Promise((resolve, reject) =>
{  setTimeout(resolve, 100, 'foo');});

Promise.all([promise1, promise2, promise3])
  .then((values) => {
    console.log(values);
  });
// output: Array [3, 42, "foo"]
```

```
// utilisation
authenticate('login', 'password')
  .then((authInfos) => {
    console.log(`Authenticated with token ${authInfos.token}`)
    return api.getLastOrders()
 })
  .then((lastOrders) => {
    console.log('Do something with the last orders');
  })
  .catch((err) => {
    console.log(`Authentication failed with error ${err.message}`);
  });
```

Les promesses

Implémentation

- Promise (Async / Await)
  - Alternative aux promesses introduite dans ES2016
  - Permet d'écrire du code asynchrone comme du code synchrone
    - Mot clé « async » pour désigner une méthode asynchrone
    - Mot clé « await » pour attendre l'exécution d'une méthode asynchrone

```
async function authenticate (user, password) {
  return await api.autenticate (user, password)
}

try {
  const authInfos = await this.authenticate("login", "password")
  console.log(`Authenticated with token ${authInfos.token}`)

  const lastOrders = await this.api.getLastOrders()
  console.log('Do something with the last orders')
})
catch((err) => {
  console.log(`Authentication failed with error ${err.message}`)
})
```

Javascript :
Opérations sur Tableau

Javascript

Opérations sur Tableau

`Array.prototype.concat()`
Renvoie un nouveau tableau qui est le tableau courant, joint avec d'autres tableaux ou valeurs.

`Array.prototype.entries()`
Renvoie un nouvel itérateur de tableau qui contient les paires de clés/valeurs pour chaque indice dans le tableau.

`Array.prototype.every()`
Renvoie true si chaque élément du tableau vérifie la condition fixée par la fonction passée en argument.

`Array.prototype.fill()`
Remplit tous les éléments d'un tableau à partir d'un indice de début jusqu'à un indice de fin avec une valeur statique.

`Array.prototype.filter()`
Renvoie un nouveau tableau qui contient tous les éléments du tableau courant pour lesquels la fonction de filtre passée en argument a renvoyé true.

`Array.prototype.find()`
Renvoie l'élément trouvé dans le tableau si un des éléments satisfait la condition fixée par la fonction passée en paramètre. Renvoie undefined si aucun élément correspondant n'est trouvé.

`Array.prototype.findIndex()`
Renvoie l'indice de l'élément trouvé dans le tableau si un élément du tableau satisfait la condition fixée par la fonction passée en argument ou -1 si aucun élément n'est trouvé.

`Array.prototype.flat()`
Renvoie un nouveau tableau avec l'ensemble des sous-éléments concaténés récursivement dans le tableau jusqu'à une profondeur indiquée.

`Array.prototype.flatMap()`
Renvoie un nouveau tableau formé en appliquant une fonction de rappel donnée à chaque élément du tableau puis en « aplatissant » le tableau d'un niveau.

`Array.prototype.forEach()`
Appelle une fonction pour chaque élément du tableau.

`Array.prototype.includes()`
Détermine si le tableau contient une valeur et renvoie true ou false selon le cas de figure.

`Array.prototype.indexOf()`
Renvoie l'indice le plus petit d'un élément du tableau égal à la valeur passée en argument ou -1 si aucun élément n'est trouvé.

`Array.prototype.join()`
Fusionne tous les éléments du tableau en une chaîne de caractères.

`Array.prototype.keys()`
Renvoie un nouvel itérateur de tableau qui contient les clés de chaque indice du tableau.

`Array.prototype.lastIndexOf()`
Renvoie le plus grand indice d'un élément du tableau égal à la valeur passée en argument ou -1 si aucun élément n'est trouvé.

`Array.prototype.map()`
Renvoie un nouveau tableau contenant les résultats de l'appel de la fonction passée en argument sur chaque élément du tableau.

`Array.prototype.pop()`
Retire le dernier élément du tableau et renvoie cet élément.

`Array.prototype.push()`
Ajoute un ou plusieurs éléments à la fin du tableau et renvoie la nouvelle longueur (length) du tableau.

`Array.prototype.reduce()`
Applique une fonction sur un accumulateur et chaque valeur du tableau (de gauche à droite) afin de réduire le tableau à une seule valeur.

Javascript

Opérations sur Tableau

```
Array.prototype.reduceRight()
Applique une fonction sur un accumulateur et chaque valeur du tableau (de droite à gauche) afin de réduire le
tableau à une seule valeur.

Array.prototype.reverse()
Inverse l'ordre des éléments du tableau à même le tableau (le premier élément devient le dernier, le dernier
devient le premier, etc.).

Array.prototype.shift()
Retire le premier élément du tableau et renvoie cet élément.

Array.prototype.slice()
Extrait une section du tableau courant et renvoie un nouveau tableau.

Array.prototype.some()
Renvoie true si au moins un des éléments du tableau satisfait la condition fournie par la fonction passée en
paramètre.

Array.prototype.sort()
Trie les éléments du tableau à même le tableau et renvoie le tableau.

Array.prototype.splice()
Ajoute et/ou retire des éléments du tableau.

Array.prototype.toLocaleString()
Renvoie une chaîne de caractères localisée qui représente le tableau et ses éléments. Cette méthode surcharge
la méthode Object.prototype.toLocaleString().

Array.prototype.toString()
Renvoie une chaîne de caractères qui représente le tableau et ses éléments. Cette méthode surcharge la méthode
Object.prototype.toString().

Array.prototype.unshift()
Ajoute un ou plusieurs éléments à l'avant du tableau et renvoie la nouvelle longueur du tableau.

Array.prototype.values()
Renvoie un nouvel itérateur de tableau qui contient les valeurs pour chaque indice du tableau.
```

Javascript

Template littéral

- Simplification de la concaténation de chaine dynamique

```
const myFunction = (name: string, date: string, temperature: number): string => {
  const message = `Bonjour ${name} !
  Nous sommes le ${date}
  La température est de ${temperature}°C`;


  return message;
}
```

Test unitaire

Test Unitaire

Jest plus rapide que Jasmine

- Les fichiers de **spec** sont générés automatiquement avec **ng generate**
- Le premier TU est automatiquement écrit
- Il reste à ajouter les **providers** (dépendances du composant ou service) et les méthodes mockées.
- L'écriture des **mocks** est simplifiés pour isoler le test sur le composant ou service.
- Démarrer les Tests Unitaires :

  npm run test ou ng test

```
describe('ProduitsService', () => {
  let service: ProduitsService;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule, MatDialogModule],
      providers: [
        ProduitsRestService, IAlerteProduitService
      ]
    });
    service = TestBed.inject(ProduitsService);
    jest.spyOn(service.alerteProduitService, 'alerteRupture').mockReturnValue();
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });

  it('RG testRupture', () => {
    const p = {
      nom: 'string',
      description: 'string',
      quantite: 5
    };

    //@ts-ignore
    service.testRupture(p);

    p.quantite = 4;
    //@ts-ignore
    service.testRupture(p);
    expect(service.alerteProduitService.alerteRupture).toHaveBeenCalledTimes(1);
  });
});
```

Test Unitaire

Jest et Testing Library

- **Pour les composants UI, le test unitraire est simplifié avec Testing-Library**
- Doc : https://testing-library.com/docs/angular-testing-library/intro
- https://testing-library.com/docs/angular-testing-library/examples

```
describe('ProduitComponent 2', () => {
  test('should render product', async () => {
    await render(ProduitComponent, {
      componentProperties: { produit : {
                nom: 'chaise',
                quantite: 100,
                description: 'toujours',
      }},
    })

    expect(screen.getByText('chaise')).toBeDefined();
    expect(screen.getByText('100')).toBeDefined();
    expect(screen.getByText('toujours')).toBeDefined();
  })
})
```

Prise en main : TP

Prise en main

Exercices – Partie 1

- TP 1 : Lister des produits en utilisant les composants `Produits` `Produit`
  - Git clone https://gitlab.com/jmasson-bzh/formation/archi-hexagonal.git
  - Check out branch **formation-tp1-liste-produits**
- TP 2 : Dans le composant Produit utiliser la méthode addProduit pour ajouter un produit dans le composant panier.
- TP 3 : Utiliser le composant `PanierInfoBulleComponent` pour afficher le nombre d'article en temps réel du panier.
  - Réutiliser le composant dans le menu pour afficher l'information sur *Accueil*

Prise en main

Exercices – Partie 2

- TP 1 : Remplacer l'utilisation du service `AlerteProduitService` par une interface `IAlerteProduitService.`
  - Check out branch **formation-tp2-alerte-rupture**
  - `ng g interface <name> <type>`
    - <type> défini dans le nom du fichier le type d'interface → "name.type.ts"
- TP 2 : Factoriser l'ouverture de la popup (Dialog) dans un service hérité `AlerteDialogService`.
- TP 3 : Afficher l'alerte si la quantité est inférieur à 5.
- TP 4 : Créer une nouvelle implémentation d'alerte-produit (`AlerteProduitPanierService`).
  - Message d'alerte : « Produit bientôt en rupture, Produit : xxx, Quantité restante: zzz »

Prise en main

Exercices – Partie 3

- TP 1 : A l'aide du service `ProduitsService` afficher dans la page panier la nouvelle alerte (`AlerteProduitPanierService`).
  - Check out branch **formation-tp3-alerte-rupture2**
  - **Utiliser l'héritage sur** `ProduitsService` **pour utiliser** `AlerteProduitPanierService.`
- TP 2 : Remplacer l'usage de `ProduitsService` et `ProduitsPanierService` par une interface `IProduitService`
  - Utiliser les providers pour spécifier la classe d'implémentation

```
providers: [
   {provide: IProduitService, useClass: ProduitsService}
 ]
```

Prise en main

Exercices – Partie 4

- Formulaire TP 1
  - 4 champs : Nom, Prénom, email, téléphone
  - Gestion des erreurs et validateurs
  - Créer un service compte pour récupérer les valeurs de l'API. (les données seront mockées avec la méthode ci-dessous). Puis initialiser les valeurs de votre formulaire à l'aide des données du service.

```
getCompte() {
   return new Promise<Compte>((resolve, reject) => {
     console.log('# Request get current Compte !');

     resolve({
       firstName: 'Jean-Pierre',
       lastName:  'DUPONT',
       email: 'jp.dupont@ertdp.fr',
       phone: '0699887744'
     });
   })
 }
```

Prise en main

Exercices – Partie 4

- Formulaire TP 2
  - Ajouter dans la page Compte la gestion des formulaires de modification et de création.
    - Utiliser MatRadioModule

```
<mat-radio-group aria-label="Selectionner un compte"
                 [value]="compteSelected"
                 (change)="onChangeRadioGroup($event)">
    <mat-radio-button value="2">Mon Compte personnel</mat-radio-button>
    <mat-radio-button value="1">Nouveau Compte</mat-radio-button>
</mat-radio-group>
```

Prise en main

TP

- Ajouter des test unitaires avec Testing-Library

Prise en main

Exercices – Partie 4

- TP 1 : Tester la règle de gestion du services `ProduitsService` (testRupture)