RÉFÉRENTIEL : Conventions de Nommage Angular

Ce document définit les standards de nommage à respecter sur l'ensemble de nos projets Angular afin de garantir la cohérence du code, faciliter la maintenance et améliorer la lisibilité pour toute l'équipe.

---

1. Structure des Fichiers (kebab-case)

La règle d'or est le **“Pattern à deux points”** : `nom.type.extension`. Tous les noms de fichiers doivent être en **kebab-case** (minuscules et tirets).

Type de fichier	Convention	Exemple
Composant	`nom.component.ts`	`user-profile.component.ts`
Service	`nom.service.ts`	`auth.service.ts`
Module	`nom.module.ts`	`app-routing.module.ts`
Interface / Modèle	`nom.model.ts`	`hero.model.ts`
Guard	`nom.guard.ts`	`auth.guard.ts`
Pipe	`nom.pipe.ts`	`file-size.pipe.ts`
Directive	`nom.directive.ts`	`highlight.directive.ts`
Store / Facade	`nom.store.ts`	`user.store.ts`

---

2. Classes et Types (PascalCase)

Les noms de classes utilisent le **PascalCase** et doivent inclure le suffixe correspondant à leur rôle.

- **Composants :**

```
export class UserProfileComponent { }
```

- **Services :**

```
export class AuthService { }
```

- **Pipes :**

```
export class FileSizePipe { }
```

- **Directives :**

```
export class HighlightDirective { }
```

- **Guards :**

```
export class AuthGuard { }
```

- **Stores / Facades :**

```
export class UserStore { }
```

---

3. Sélecteurs de composants et directives

Les sélecteurs doivent :

- utiliser **kebab-case**
- avoir un **préfixe spécifique au projet**

Exemple :

- `<app-user-list></app-user-list>`
- `<app-dashboard></app-dashboard>`

Incorrect :

- `<user-list></user-list>`

Le préfixe par défaut Angular est **app-**, mais il peut être remplacé par un préfixe métier (ex : `logeas-`).

Exemple :

- `<logeas-grid></logeas-grid>`

---

4. Tests Unitaires (.spec.ts)

Les fichiers de tests portent le même nom que le fichier testé.

Exemple :

- `auth.service.ts`
- `auth.service.spec.ts`

Structure recommandée :

```
describe('AuthService', () => {

  it('should return true when user is authenticated', () => {
  });
```

```
});
```

Bonnes pratiques :

- décrire **le comportement attendu**
- utiliser la forme **should …**

---

5. Assets et Ressources Statiques

Tous les fichiers statiques utilisent **kebab-case**.

Exemples :

- `logo-entreprise.svg`
- `user-avatar-placeholder.png`

Organisation recommandée :

- `assets/images/`
- `assets/icons/`
- `assets/i18n/`
- `assets/config/`

---

6. Logique Interne (camelCase)

Les variables et méthodes utilisent le **camelCase**.

Exemples :

- variables

```
isValid: boolean = false
```

- méthodes

```
updateUserData()
```

- inputs

```
@Input() userRole: string
```

- outputs

```
@Output() userChanged = new EventEmitter<User>()
```

---

7. Observables

Les **Observables doivent être suffixés par `$`**.

Exemples :

```
users$ : Observable<User[]>;
isLoading$ : Observable<boolean>;
currentUser$ : Observable<User>;
```

Exemple dans un template :

```
<div *ngIf="users$ | async as users">
```

---

8. Constantes et Énumérations

Constantes globales :

UPPER_SNAKE_CASE

```
export const MAX_FILE_SIZE = 5000;
```

Enums :

- nom en **PascalCase**
- valeurs en **PascalCase** ou **UPPER_SNAKE_CASE**

```
export enum UserRole {
  Admin,
  Member,
  Guest
}
```

---

9. Organisation des dossiers

Structure recommandée :

```
src/app/

core/
services globaux
guards
```

```
interceptors

shared/
composants réutilisables
pipes
directives

features/
modules métier

features/user/
user.component.ts
user.service.ts
user.store.ts
user.model.ts
```

Principes :

- **core** → singleton global
- **shared** → réutilisable
- **features** → logique métier

---

10. Recommandations additionnelles

Interfaces :

- **ne pas utiliser le préfixe “I”**

Correct :

```
User
Account
Invoice
```

Incorrect :

```
IUser
IAccount
```

Services :

Nommer selon **la responsabilité métier**

Correct :

```
AuthService
StorageService
NotificationService
```

Incorrect :

```
DataService
HelperService
UtilsService
```

---

11. Bonnes pratiques générales

- un fichier = **une classe principale**
- éviter les fichiers trop longs (>500 lignes)
- privilégier les **composants petits et spécialisés**
- éviter la logique métier dans les composants → préférer les **services ou stores**