Formation **Angular : développement avancé et performance**

Cette formation de 3 jours s'adresse aux développeurs Angular expérimentés qui souhaitent passer au niveau supérieur. Vous explorerez en profondeur le nouveau paradigme réactif basé sur les Signals — writable, computed, inputs signalés — et le mode Zoneless qui révolutionne la détection de changements. Vous mettrez en place une gestion d'état robuste avec NgRx et ses API standalone, et vous implémenterez le rendu côté serveur avec hydration incrémentale. La formation couvre également les patterns RxJS avancés, le routage sécurisé et les stratégies de test avec Jest ou Vitest. Chaque chapitre s'appuie sur des cas pratiques concrets pour ancrer les concepts dans la réalité des projets Angular en production.

Durée
3 jours

Objectifs pédagogiques

- Architecturer une application Angular moderne avec standalone components et injection de dépendances avancée
- Implémenter le mode Zoneless et optimiser la détection de changements
- Maîtriser les Signals avancés (writable, computed, linkedSignal, inputs signalés) et leur interopérabilité avec RxJS
- Gérer l'état applicatif avec NgRx et les API standalone (provideStore, provideEffects)
- Mettre en œuvre le rendu côté serveur (SSR) avec hydration incrémentale
- Sécuriser et optimiser le routage avec guards fonctionnels, resolvers et lazy loading avancé

Public
Développeurs

Prérequis
- Expérience pratique de développement avec Angular (minimum 3-6 mois de projet)- Bonne maîtrise de TypeScript (génériques, types utilitaires)- Connaissance des bases de RxJS (Observables, pipe, subscribe)- Notions de Signals Angular (signal, computed, effect)- Pratique des tests unitaires (Jasmine ou équivalent)

Programme de formation

Architecture moderne et standalone components

- Architecture applicative Angular : organisation d'un projet standalone-first, suppression des NgModules et structuration par feature
- Injection de dépendances avancée : InjectionTokens personnalisés, providers hiérarchiques, stratégies d'encapsulation des services
- Workspace Nx : introduction au monorepo, génération de projets et gestion des dépendances entre librairies
- Composants avancés : projection de contenu (ng-content), vues dynamiques (NgComponentOutlet), décorateurs @ViewChild et @ContentChild

Exemples d'activités pratiques :
- Mise en place d'un projet Angular moderne avec architecture standalone et injection avancée
- Création d'un composant générique réutilisable avec projection de contenu

Zoneless et détection de changements

- Zone.js : rôle historique dans la détection de changements et limites de performance
- Mode Zoneless : architecture sans Zone.js pour un rendu plus rapide et un contrôle explicite des mises à jour
- Stratégies de détection : Default vs OnPush, utilisation de ChangeDetectorRef pour le contrôle manuel
- Optimisation des cycles de rendu : identification et suppression des re-rendus inutiles, impact sur les performances réelles

Exemples d'activités pratiques :
- Migration d'une application utilisant Zone.js vers le mode Zoneless
- Profiling de la détection de changements avec Angular DevTools et comparaison des stratégies

Signals avancés et interopérabilité

- Writable signals : gestion de l'état local des composants avec signal(), set(), update()
- Computed signals : réactivité fine avec computed(), dépendances automatiques et évaluation paresseuse
- Effects et linkedSignal : effect() pour les effets de bord, linkedSignal pour la synchronisation d'état dérivé
- Inputs signalés : communication parent/enfant via input signals comme alternative aux décorateurs @Input classiques
- Interopérabilité RxJS/Signals : toSignal(), toObservable(), stratégies de migration progressive depuis les Observables

Exemples d'activités pratiques :
- Réécriture d'un composant basé sur des Observables en utilisant exclusivement les Signals
- Implémentation d'une communication inter-composants complexe avec inputs et outputs signalés
- Migration progressive d'un service RxJS vers un pattern hybride Signals/RxJS

RxJS avancé et patterns réactifs

- Subjects spécialisés : BehaviorSubject, ReplaySubject et AsyncSubject — cas d'usage et choix du bon type
- Opérateurs avancés : switchMap, mergeMap, concatMap, exhaustMap pour l'orchestration de flux asynchrones
- Combinaison de flux : combineLatest, forkJoin, withLatestFrom pour synchroniser des sources de données multiples
- Gestion des souscriptions : stratégies d'unsubscribe, takeUntilDestroyed et bonnes pratiques pour éviter les fuites mémoire
- Marble testing : tests d'Observables avec la syntaxe marble pour valider les comportements asynchrones

Exemples d'activités pratiques :
- Implémentation d'un service réactif avec orchestration de flux complexes (recherche avec debounce et annulation)
- Écriture de marble tests pour valider le comportement d'un opérateur personnalisé

Gestion d'état avec NgRx

- Architecture NgRx : Store, Actions, Reducers et le flux unidirectionnel de données
- Effects : gestion des effets de bord (appels HTTP, navigation) de manière déclarative
- Selectors avancés : composition de sélecteurs, mémoïsation et optimisation des performances
- API standalone : provideStore() et provideEffects() pour une intégration sans NgModule
- NgRx et Signals : utilisation conjointe de NgRx pour l'état global et des Signals pour l'état local des composants

Exemples d'activités pratiques :
- Mise en place d'un store NgRx complet pour gérer l'état d'une feature applicative
- Intégration de NgRx avec les Signals pour un pattern hybride global/local

Routage avancé et sécurisation

- Configuration avancée du Router : provideRouter() avec options, secondary routes et named outlets
- Guards fonctionnels : CanActivate, CanActivateChild, CanDeactivate sous forme de fonctions standalone
- Resolvers : pré-chargement de données avant la navigation pour une expérience utilisateur fluide
- Lazy loading avancé : loadComponent(), loadChildren(), stratégies de pré-chargement et redirections asynchrones

Exemples d'activités pratiques :
- Sécurisation d'une application multi-rôles avec guards fonctionnels et redirections conditionnelles
- Implémentation d'un resolver pour pré-charger les données d'une page de détail

Server-Side Rendering et hydration

- SSR Angular : rendu côté serveur activé par défaut dans les nouveaux projets, configuration et bénéfices
- Hydration incrémentale : réactivation progressive des composants côté client pour un temps d'interactivité optimal
- Event replay : capture et rejeu des interactions utilisateur pendant la phase d'hydration
- Prérendu statique : génération de pages HTML statiques pour les contenus peu dynamiques

Exemples d'activités pratiques :
- Mise en place d'une application Angular avec SSR et hydration incrémentale
- Configuration du prérendu statique pour les pages publiques d'une application

Tests avancés et optimisation des performances

- Jest ou Vitest : configuration comme runner de tests moderne en remplacement de Karma
- TestBed avancé : tests de composants standalone, mocking de services et de HttpClient, tests asynchrones
- Marble testing : validation du comportement des Observables et des flux NgRx avec la syntaxe marble
- Optimisation de production : tree-shaking, bundle analysis, directives @defer pour le chargement conditionnel
- Profiling : utilisation d'Angular DevTools pour identifier les goulots d'étranglement et mesurer l'impact des optimisations

Exemples d'activités pratiques :
- Écriture de tests avancés pour un composant intégrant NgRx et des Signals
- Audit de performance d'une application et mise en œuvre d'optimisations ciblées

Moyens et méthodes pédagogiques

- La formation alterne entre présentations des concepts théoriques et mises en application à travers d'ateliers et exercices pratiques (hors formation de type séminaire).
- Les participants bénéficient des retours d'expérience terrains du formateur ou de la formatrice
- Un support de cours numérique est fourni aux stagiaires

Modalités d'évaluation

- **En amont de la session de formation**, un questionnaire d'auto-positionnement est remis aux participants, afin qu'ils situent leurs connaissances et compétences déjà acquises par rapport au thème de la formation.

- **En cours de formation**, l'évaluation se fait sous forme d'ateliers, exercices et travaux pratiques de validation, de retour d'observation et/ou de partage d'expérience, en cohérence avec les objectifs pédagogiques visés.

- **En fin de session**, le formateur évalue les compétences et connaissances acquises par les apprenants grâce à un questionnaire reprenant les mêmes éléments que l'auto-positionnement, permettant ainsi une analyse détaillée de leur progression.