DÉVELOPPEMENT FRONTEND

TypeScript pour les développeurs JavaScript

Ajouter le typage statique à vos projets JavaScript et React avec TypeScript. Réduisez les bugs, améliorez l'autocomplétion et écrivez du code plus robuste.

DURÉE	TARIF HT	NIVEAU	LANGUE	GROUPE	FORMAT
2j (14h)	**1200.00 €**	**Intermédiaire**	**Français**	**3-12**	**Formation**

1 PUBLIC VISÉ

- Développeurs JavaScript maîtrisant ES6+ souhaitant sécuriser leur code avec le typage statique.
- Développeurs React voulant typer leurs composants, props et hooks pour réduire les bugs en production.
- Développeurs autodidactes ou en reconversion souhaitant adopter les standards professionnels des équipes front-end modernes.
- Personnes ayant suivi la formation JavaScript fondamentaux ou disposant d'une expérience JavaScript équivalente.

2 PRÉREQUIS

- Bonne maîtrise de JavaScript ES6+ : fonctions fléchées, destructuring, modules, classes.
- La formation JavaScript fondamentaux ou équivalent est requise.
- Des notions de React sont appréciées pour les modules dédiés aux composants.
- Savoir utiliser le terminal et VS Code.

3 OBJECTIFS PÉDAGOGIQUES

- Comprendre l'intérêt du typage statique et ce que TypeScript apporte concrètement par rapport à JavaScript
- Maîtriser les types de base, les interfaces et les types littéraux
- Distinguer interface et type alias et savoir choisir entre les deux
- Utiliser les generics pour écrire du code réutilisable et fortement typé
- Typer les props, les états et les hooks dans un projet React
- Utiliser les utility types : Partial, Required, Omit, Pick, Record, ReturnType
- Configurer tsconfig.json selon le contexte du projet
- Migrer un projet JavaScript existant vers TypeScript de manière progressive

4 PROGRAMME DÉTAILLÉ

Jour 1 (7h) – Fondamentaux TypeScript

Module 1 – Découverte de TypeScript (3h)

Pourquoi TypeScript ? (1h)

- Les limites de JavaScript à grande échelle : bugs silencieux, refactoring risqué
- Ce que le typage statique apporte : autocomplétion, détection d'erreurs à la compilation
- TypeScript vs JavaScript : transpilation, compatibilité, courbe d'adoption
- TypeScript dans l'écosystème : React, Node.js, Next.js, Angular

Mise en place de l'environnement (1h)

- Installer TypeScript et ts-node
- Compiler avec tsc : options essentielles
- Configurer tsconfig.json : strict, target, module, paths
- Intégrer TypeScript dans un projet Vite existant
- Lire et interpréter les erreurs de compilation

Types de base (1h)

- string, number, boolean, null, undefined, any, unknown, never, void
- Inférence de type : quand TypeScript devine le type automatiquement
- Pourquoi éviter any et comment le remplacer par unknown
- Cas pratique : typer des variables et des valeurs de retour de fonctions simples

Module 2 – Fonctions, objets et tableaux (2h)

Typer les fonctions (1h)

- Paramètres typés, valeur de retour explicite
- Paramètres optionnels et valeurs par défaut
- Rest parameters et tuples
- Function overloading : définir plusieurs signatures pour une même fonction
- Cas pratique : bibliothèque utilitaire entièrement typée

Typer les objets et les tableaux (1h)

- Object types inline : définir la forme d'un objet directement
- Propriétés optionnelles et propriétés readonly
- Tableaux typés : Array et la syntaxe T[]
- Tuples : tableaux à longueur et types fixes
- Cas pratique : modéliser une entité utilisateur avec ses propriétés obligatoires et optionnelles

Module 3 – Interfaces et types alias (2h)

Interface vs type alias (45min)

- Déclarer une interface : syntaxe et conventions de nommage
- Déclarer un type alias : cas d'usage spécifiques
- Les différences clés : extension, fusion de déclarations, unions
- La règle pratique : quand choisir l'un ou l'autre
- Cas pratique : modéliser les entités d'une application e-commerce

Héritage et composition de types (45min)

Étendre une interface avec extends

- Intersection types : combiner plusieurs types avec &
- Union types : accepter plusieurs types possibles avec |
- Discriminated unions : pattern pour distinguer des variantes d'un type
- Cas pratique : système de notifications avec types de message distincts

Enums et types littéraux (30min)

- Enums numériques et string enums
- Les limites des enums et les alternatives modernes
- Types littéraux : restreindre une valeur à un ensemble fini
- Const assertions : as const pour inférer des types précis
- Cas pratique : statuts d'une commande avec union de littéraux

Jour 2 (7h) – Generics et utility types

Module 3 suite – Consolidation interfaces et types (1h)

- Exercices de modélisation : concevoir les types d'une application de A à Z
- Correction collective et analyse des erreurs fréquentes
- Questions et approfondissement selon les besoins du groupe

Module 4 – Generics (3h)

Comprendre les generics (1h30)

- Le problème que les generics résolvent : code réutilisable sans perdre le typage
- Syntaxe de base : fonctions génériques, types génériques
- Contraindre un generic avec extends
- Plusieurs paramètres de type : T, U, K, V
- Cas pratique : fonction de tri générique et wrapper de réponse API

Generics avancés (1h30)

- Generics avec interfaces et classes
- Valeurs par défaut pour les paramètres de type
- keyof et types indexés : accéder aux clés d'un objet de manière typée
- Conditional types : T extends U ? X : Y
- Cas pratique : hook générique useFetch avec typage de la réponse

Module 5 – Utility types (3h)

Les utility types essentiels (1h30)

- Partial : rendre toutes les propriétés optionnelles
- Required : rendre toutes les propriétés obligatoires
- Readonly : interdire la modification des propriétés
- Pick : extraire un sous-ensemble de propriétés
- Omit : exclure des propriétés d'un type existant
- Cas pratique : formulaire de mise à jour partielle d'un utilisateur

Les utility types avancés (1h30)

- Record : créer un dictionnaire typé clé/valeur
- Exclude et Extract : filtrer des unions de types

NonNullable : retirer null et undefined d'un type

- ReturnType et Parameters : inférer les types depuis une fonction existante
- Cas pratique : typer un store Redux ou une configuration d'API sans duplication

Jour 3 (7h) – TypeScript avec React et projet de synthèse

Module 6 – TypeScript avec React (4h)

Typer les composants React (1h30)

- Typer les props avec une interface : syntaxe et conventions
- React.FC vs typage direct : pourquoi préférer le typage direct
- La prop children : React.ReactNode vs React.PropsWithChildren
- Les événements React : MouseEvent, ChangeEvent, FormEvent
- Cas pratique : composant Button générique avec variants typés

Typer les hooks (1h30)

- useState avec un type explicite : useState<T>
- useRef : HTMLElement, RefObject vs MutableRefObject
- useReducer : typer l'état et les actions avec discriminated unions
- useContext : créer un contexte fortement typé
- Cas pratique : hook personnalisé useForm avec typage générique

Migration et bonnes pratiques (1h)

- Migrer un projet React JS vers TypeScript : stratégie progressive fichier par fichier
- Intégrer TypeScript dans un projet Next.js : next.config.ts, types Next.js
- Les fichiers de déclaration .d.ts : à quoi ils servent et quand en créer
- Installer les types manquants avec @types/...
- Les règles tsconfig strict les plus importantes : noImplicitAny, strictNullChecks
- Cas pratique : migrer un composant React existant et corriger les erreurs de type

Module 7 – Projet de synthèse (3h)

Migration complète d'un projet React JavaScript vers TypeScript

- Cahier des charges : migrer une application React existante avec composants, hooks personnalisés, contexte et appels API
- Étape 1 : mise en place de tsconfig.json et renommage des fichiers
- Étape 2 : typage des entités métier avec interfaces et types
- Étape 3 : typage des composants, props et événements
- Étape 4 : typage des hooks et du contexte global
- Revue de code collective : choix de types, lisibilité, absence de any
- Retour formateur individualisé sur le projet rendu

5 COMPÉTENCES VISÉES

- Écrire du TypeScript idiomatique dans un projet React ou Node.js
- Concevoir des types et interfaces adaptés à la structure des données métier
- Exploiter l'autocomplétion et les erreurs de compilation pour produire un code plus fiable
- Utiliser les utility types pour transformer et composer des types existants
- Configurer et intégrer TypeScript dans un projet Vite ou Next.js
- Migrer progressivement une base de code JavaScript vers TypeScript sans bloquer la production

6 MODALITÉS PÉDAGOGIQUES

Formation délivrée en présentiel ou distanciel (visioconférence). Le formateur alterne entre méthode démonstrative (live coding commenté avec retour immédiat du compilateur), méthode interrogative (analyse d'erreurs TypeScript réelles et discussion des choix de typage) et méthode active (exercices de typage progressifs sur des cas concrets). L'accent est mis sur la compréhension du système de types plutôt que sur la mémorisation de la syntaxe.

7 MOYENS PÉDAGOGIQUES ET TECHNIQUES

- Support de cours numérique mis à disposition des apprenants
- Dépôt GitHub de démonstration avec exercices et corrections par module
- Environnement de développement : VS Code + TypeScript + Vite
- Pour le distanciel : visioconférence (Zoom ou équivalent), partage d'écran, chat en direct
- Accès à la plateforme pédagogique LaPolaris (supports, ressources, émargement)

8 MODALITÉS D'ÉVALUATION

- En cours de formation : exercices de typage progressifs corrigés à chaque module
- En fin de formation : migration d'un composant React JavaScript vers TypeScript avec correction de toutes les erreurs de compilation
- Questionnaire d'auto-évaluation des acquis en fin de parcours

9 CRITÈRES D'ÉVALUATION

- Choix pertinent entre interface et type alias selon le contexte
- Utilisation correcte des generics pour éviter la duplication de types
- Typage complet des props et hooks React sans recours à any
- Application adaptée des utility types pour transformer des types existants
- Configuration tsconfig.json cohérente avec le niveau de rigueur attendu en production

10 MODALITÉS DE VALIDATION

Attestation de fin de formation délivrée à l'issue du parcours, conditionnée à une assiduité d'au moins 80 % et à la réalisation de l'exercice de migration. L'attestation précise les objectifs atteints et les compétences acquises.

11 SUIVI ET ACCOMPAGNEMENT

- Feuilles d'émargement signées par demi-journée (présentiel) ou émargement numérique (distanciel)
- Traçabilité des activités pédagogiques réalisées
- Attestation d'assiduité délivrée en fin de formation
- Suivi individuel via les exercices corrigés et le projet de migration

12 CONDITIONS D'ACCÈS

Formation accessible sur inscription directe, sans prérequis administratif particulier. Le financement peut être pris en charge par l'employeur dans le cadre d'un plan de développement des compétences, ou en autofinancement avec possibilité de paiement en plusieurs fois.

13 DÉLAIS D'ACCÈS

Inscription possible jusqu'à **5 jours ouvrés** avant le début de la session. Pour toute demande urgente, nous contacter directement.

ACCESSIBILITÉ · HANDICAP

Nos formations sont accessibles aux personnes en situation de handicap. Pour toute situation nécessitant un aménagement (matériel, temporel ou pédagogique), nous vous invitons à nous contacter avant l'inscription afin d'étudier les adaptations possibles. Référent handicap : contact@lapolaris.fr

LaPolaris

TÉL. +33762584798

EMAIL contact@lapolaris.fr

WEB lapolaris.fr