Présentation de l'environnement de développement NodeJS

Node.js est un environnement d'exécution JavaScript côté serveur, open-source et multi-plateforme. Il permet d'exécuter du code JavaScript en dehors d'un navigateur web.

Histoire

Node.js a été créé par Ryan Dahl en 2009. Son développement et sa maintenance sont effectués par l'entreprise Joyent. Dahl a eu l'idée de créer Node.js après avoir observé la barre de progression d'un chargement de fichier sous Flickr : le navigateur ne savait pas quel pourcentage du fichier était chargé et devait adresser une requête au serveur web. Dahl voulait développer une méthode plus simple. Le serveur web Mongrel de Ruby a été l'autre source d'inspiration pour Dahl.

Source: Node.js — Wikipédia

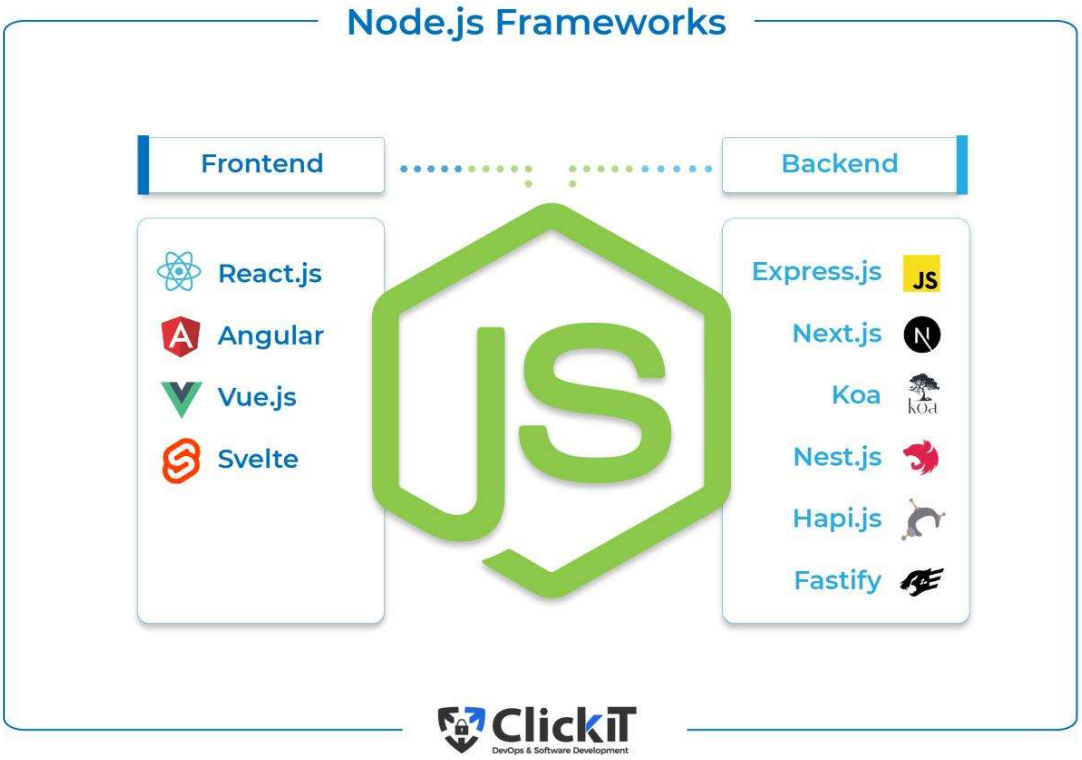
Caractéristiques

- Basé sur le moteur V8 de Chrome : performant et rapide
- Non-bloquant et asynchrone : gère efficacement les opérations d'entrée/sortie
- Écosystème riche : npm (Node Package Manager) donne accès à plus d'un million de packages
- Léger et évolutif : idéal pour les applications en temps réel et à forte charge

Cas d'utilisation courants

- Applications web et API REST
- Applications en temps réel (chat, jeux)
- Outils de ligne de commande
- Microservices

Frameworks



Présentation de la CLI Angular

Qu'est-ce qu'Angular ?

Angular est un framework front-end open-source développé par Google pour créer des applications web dynamiques et des Single Page Applications (SPA). Il utilise TypeScript comme langage principal et suit le modèle d'architecture MVC (Modèle-Vue-Contrôleur).

Caractéristiques principales d'Angular

- Basé sur TypeScript : typage statique et fonctionnalités avancées
- Architecture par composants : structure modulaire et réutilisable
- Binding bidirectionnel : synchronisation automatique entre le modèle et la vue
- Injection de dépendances : gestion simplifiée des services
- SPA (single page application) et routeur intégré : navigation entre les vues sans rechargement de page
- Testabilité : outils de test unitaire et end-to-end

Qu'est-ce qu'Angular CLI ?

Angular CLI (Command Line Interface) est un outil en ligne de commande qui facilite la création, le développement, les tests et le déploiement d'applications Angular.

Fonctionnalités d'Angular CLI

- Création de projets : génération de la structure de base
- Génération de code : composants, services, modules, etc.
- Serveur de développement : avec rechargement à chaud
- Tests unitaires et end-to-end : intégration avec Karma et Protractor
- Build optimisé : préparation pour la production
- Mise à jour du framework : facilite les migrations

Commandes essentielles d'Angular CLI

```
# Installation globale d'Angular CLI
npm install -g @angular/cli

# Installation d'une version spécifique globale d'Angular CLI
npm install -g @angular/cli@version

# Vérifier la version
ng version

# Créer un nouveau projet
ng new mon-projet-angular

# Démarrer le serveur de développement
ng serve

# Générer un nouveau composant
ng generate component mon-composant

# ou en abrégé
ng g c mon-composant

# Générer un service
ng g service mon-service

# Construire pour la production
ng build --prod
```

En résumé

Angular et Angular CLI forment ensemble une solution complète pour développer des applications web modernes, maintenables et évolutives, avec un excellent outillage pour les développeurs.

Rappel des normes EcmaScript

Qu'est-ce qu'ECMAScript ?

ECMAScript est la spécification standardisée du langage JavaScript, maintenue par l'organisation ECMA International. JavaScript est l'implémentation la plus connue de cette norme.

Évolution des versions principales

ES5 (ECMAScript 2009)

La version de référence largement supportée par tous les navigateurs.

- Méthodes d'array : forEach, map, filter, reduce
- Mode strict : "use strict";
- Accesseurs de propriétés (getters/setters)
- JSON natif

ES6/ES2015

Révolution majeure avec de nombreuses fonctionnalités.

- Classes et modules
- Constantes et variables à portée de bloc (const, let)
- Fonctions fléchées (=>)
- Template literals (`texte ${variable}`)
- Destructuration (const { prop } = obj)
- Paramètres par défaut et rest (...args)
- Spread operator (...array)
- Promesses
- Nouveaux types (Map, Set, WeakMap, WeakSet)

ES2016

- Opérateur d'exponentiation (**)
- Méthode Array.prototype.includes

ES2017

- Async/await
- Object.values, Object.entries
- String padding (padStart, padEnd)

ES2018

- Rest/spread pour les objets
- Promesses améliorées (finally)
- Regex améliorées
- Asynchronous iteration

ES2019

- Array.prototype.flat, flatMap
- Object.fromEntries
- String.prototype.trimStart, trimEnd
- Améliorations pour catch sans paramètre

ES2020

- Opérateur de coalescence des nuls (??)
- Chaînage optionnel (?.)
- BigInt
- Promise.allSettled
- globalThis

ES2021

- String.prototype.replaceAll
- Opérateurs d'affectation logique (&&=, ||=, ??=)
- Promise.any
- Séparateurs numériques (1_000_000)

ES2022

- Top-level await
- Méthodes privées et champs privés dans les classes
- Array.prototype.at
- RegExp Match Indices

Bonnes pratiques

- Préférer const à let quand possible, éviter var
- Utiliser les fonctions fléchées pour préserver le contexte this
- Exploiter la destructuration pour un code plus lisible
- Privilégier async/await plutôt que les callbacks
- Utiliser les modules ES pour organiser le code
- Créer un projet Angular à l'aide d'angular CLI

Compatibilité

Pour assurer la compatibilité avec les navigateurs plus anciens :

- Utiliser des transpileurs comme Babel
- Ajouter des polyfills pour les fonctionnalités manquantes
- Vérifier les tables de compatibilité (comme celles de MDN)

Ces normes ont considérablement amélioré JavaScript, le transformant d'un simple langage de script à un langage robuste pour le développement d'applications complexes.

Compatibilité des navigateurs avec Angular moderne

Comment Angular gère la transpilation ?

1. **TypeScript Compiler (tsc)** :
   (typesrcipt est à la fois un langage de programmation et un compilateur)
   - Angular utilise principalement TypeScript comme transpileur
   - Le compilateur TypeScript convertit votre code TS en JavaScript
   - TypeScript est capable de cibler différentes versions de JavaScript (ES5, ES2015, etc.) en fonction de la configuration **tsconfig.json**
2. **Angular Compiler (ngc)** :
   - Extension du compilateur TypeScript
   - Gère les templates HTML et les fonctionnalités spécifiques à Angular
   - Permet l'AOT (Ahead-of-Time) compilation
3. **Terser** :
   - Pour la minification du code en production

En résumé

- Angular utilise principalement **TypeScript** pour la transpilation,
- La configuration est automatique et ne nécessite généralement pas d'intervention
- Le système de build d'Angular gère intelligemment la compatibilité des navigateurs
- **Nous pouvons nous concentrer sur l'écriture de code TypeScript moderne sans nous soucier des détails de transpilation.** Sans configuration supplémentaire, en travaillant sur angular, nous bénéficions d'une excellente compatibilité avec les navigateurs

TypeScript

Présentation de TypeScript

TypeScript est à la fois :

Un langage de programmation :

- Sur-ensemble de JavaScript
- Ajoute le typage statique et d'autres fonctionnalités
- Permet d'écrire du code plus robuste avec vérification des types

Un compilateur/transpileur (tsc) :

- Convertit le code TypeScript (.ts) en JavaScript (.js)
- Vérifie les types pendant la compilation
- Transforme les fonctionnalités modernes en JavaScript compatible

Les types de variables de TS

Types simples / primitifs

boolean	True / false
number*	Entiers, décimaux, etc.
string	Texte
null	Absence intentionnelle de valeur
undefined	Variable non initialisée
symbol	Valeur unique et immuable
bigInt*	Grands entiers

Type number et bigInt *

Type number

- Basé sur le standard IEEE 754 (nombres à virgule flottante double précision)
- Plage sécurisée : -9,007,199,254,740,991 à 9,007,199,254,740,991
- Soit environ ±9 quadrillions (±2^53 - 1)
- Au-delà de cette plage, la précision peut être perdue

Type bigint

- À utiliser quand vous avez besoin de nombres supérieurs à 9,007,199,254,740,991
- Ou inférieurs à -9,007,199,254,740,991
- Permet de manipuler des entiers de taille arbitraire sans perte de précision
- Indiqué par le suffixe n (ex: 123456789012345678901234 5n)
- Cas d'usage typiques pour bigint : cryptographie, timestamps de précision extrême, identifiants très grands, calculs financiers de haute précision …

Types spéciaux

any	N'importe quel type **(éviter si possible)**
unknown	*any* plus sécurisé. **Force la vérification** : on ne peut pas accéder à des propriétés ou appeler des méthodes sans vérifier le type
void	absence de valeur de retour
never	Jamais de retour (erreurs, boucles infinies …) **Utilisé pour les fonctions qui :** ○ Ne terminent jamais leur exécution (boucle infinie) ○ Lancent toujours une exception ○ Ont des chemins d'exécution impossibles

Types complexes

array	Tableaux
Tuple	arrays avec **types fixes**
enum	Ensembles de constantes
object	Objets javascript

Types composés

union	plusieurs types possibles
intersection	Combinaison de types

Types personnalisés

interface	contrats pour objets
type	alias pour types existants
generics	types paramétrables <T>

Le typage fort

Qu'est-ce que le typage fort ?

- Système où chaque variable a un type bien défini
- Les types sont vérifiés à la compilation
- Empêche les opérations entre types incompatibles

- Réduit les erreurs de programmation

Avantages du typage fort

- Détection des erreurs à la compilation (avant l'exécution)
- Autocomplétion et aide au développement
- Documentation implicite du code
- Refactoring plus sûr

Les interfaces de classe

Définition d'une interface javascript

- Contrat que les classes doivent respecter
- Définit la structure sans l'implémentation
- Permet de garantir qu'une classe possède certaines propriétés et méthodes

Exemple d'une interface de classe :

```
interface Véhicule {
    // Propriétés
    brand: string;
    year: number;
    isElectric?: boolean; // Propriété optionnelle

    // Méthodes
    start(): void;
    stop(): void;
    getInfo(): string;
}
```

Typer avec des interfaces ou avec des class ?

En typescript nous pouvons créer des types personnalisés en utilisant des interfaces OU des *class*. Les classes et interfaces ont des usages différents :

Avantages des classes

- **Instanciables** : Vous pouvez créer des instances avec new MaClasse()
- **Comportement** : Peuvent contenir des méthodes avec implémentation
- **Constructeurs** : Permettent d'initialiser des objets avec des valeurs par défaut

- **Héritage** : Support complet de l'héritage (extends)
- **Implémentation runtime** : Existent dans le code JavaScript final

Avantages des interfaces

- **Vérification de type** : Uniquement à la compilation, pas de code généré
- **Légèreté** : Ne produisent aucun JavaScript (optimisation de bundle)
- **Extension multiple** : Peuvent étendre plusieurs interfaces (extends A, B, C)
- **Déclaration de forme** : Idéales pour définir des contrats

Présentation des décorateurs

Qu'est-ce qu'un décorateur ?

Un décorateur est une fonction spéciale qui permet de modifier le comportement de classes, méthodes, propriétés ou paramètres de fonction. C'est un moyen élégant d'ajouter des métadonnées ou d'altérer le comportement du code sans modifier sa structure principale.

Comment fonctionnent les décorateurs ?

- **Syntaxe** : Ils s'appliquent avec le symbole @ suivi du nom du décorateur
- **Moment d'exécution** : Ils sont exécutés lors de la définition de la classe/méthode, pas à l'instanciation
- **Fonction** : Un décorateur est essentiellement une fonction qui reçoit des informations sur l'élément décoré
- **Métaprogrammation** : Ils permettent d'écrire du code qui manipule d'autre code

Exemple

```
// Décorateur de classe
function Logger(target: Function) {
    console.log(`Classe créée: ${target.name}`);
}

// Application du décorateur
@Logger
class Person {
    name: string;

    constructor(name: string) {
        this.name = name;
```

```
    }
}
```

Types de décorateurs principaux

- Décorateurs de classe - modifient le comportement d'une classe entière
- Décorateurs de méthode - modifient le comportement d'une méthode
- Décorateurs de propriété - associent des métadonnées à une propriété
- Décorateurs de paramètre - associent des métadonnées à un paramètre de fonction

Utilisation pratique

Les décorateurs sont particulièrement utiles pour :

- Journalisation automatique
- Mesure de performances
- Gestion des autorisations
- Validation de données
- Injection de dépendances
- …

Les décorateurs dans les frameworks modernes

Les frameworks JavaScript modernes, en particulier Angular, exploitent massivement les décorateurs comme un mécanisme élégant **permettant d'injecter des fonctionnalités complexes et des métadonnées dans le code sans surcharger visuellement les fichiers**, offrant ainsi une syntaxe concise qui masque la complexité tout en enrichissant les classes et méthodes avec des comportements sophistiqués gérés par le framework.