2.3 Outils et Processus de Développement

• Installation et Configuration de Cypress : Procédure d'installation, paramétrage, intégration avec le pipeline CI/CD

\* Mise en place d'un environnement de test de logiciels avec Cypress

L’exécution de tests automatisés via Cypress nécessite un environnement sollicitant 3 principaux outils qui sont :

Node.js

Node.js est un environnement d'exécution JavaScript côté serveur qui permet aux développeurs d'utiliser JavaScript pour créer des applications backend. Notamment réputé pour être rapide et efficace dans la gestion des applications en temps réel, des requêtes réseau et des systèmes d'entrées/sorties intensives.

### Principales caractéristiques de Node.js

1. \*\*Architecture non bloquante et asynchrone\*\* : Node.js utilise un modèle d'entrées/sorties non bloquant, ce qui signifie que les opérations peuvent s'exécuter sans attendre la fin des précédentes. Cela le rend particulièrement performant pour les applications nécessitant un grand nombre de requêtes simultanées, comme les applications de chat en temps réel ou les systèmes de streaming.

2. \*\*Évolutivité\*\* : Grâce à son modèle basé sur des événements, Node.js est capable de gérer des milliers de connexions simultanées, ce qui le rend particulièrement adapté pour les applications nécessitant une grande scalabilité.

3. \*\*Écosystème riche de modules\*\* : Node.js dispose d'un gestionnaire de paquets appelé \*\*npm\*\* (Node Package Manager), qui offre des milliers de bibliothèques et de modules permettant d’étendre ses fonctionnalités sans avoir à écrire du code depuis zéro.

### Pourquoi choisir Node.js ?

Node.js est souvent préféré pour ses performances élevées, sa capacité à gérer de grandes charges de trafic, et pour sa flexibilité dans les applications nécessitant une communication en temps réel. Grâce à son architecture légère et rapide, il est devenu un choix populaire pour les startups comme pour les grandes entreprises.

## Comment installer Node.js ?

Il existe plusieurs méthodes pour installer Node.js en fonction de votre système d'exploitation, de vos besoins (environnement de production ou de développement) et de votre préférence pour la gestion des versions. Voici les méthodes d'installation les plus courantes :

1. Installer Node.js avec le programme d'installation officiel

Allez sur le site officiel de Node.js nodejs.org et téléchargez l'installateur pour votre système d'exploitation (Windows, macOS ou Linux). Un fois sur le site officiel, il faudra cliquer sur le bouton "Télécharger Node.js (LTS)", ce qui nous permettra d'obtenir la version v22.11.0 étant la plus adaptée aujourd'hui car plus stable.

Une fois le téléchargement terminé, exécutez le programme d’installation et suivez les instructions à l’écran.

Cette méthode est simple et rapide, mais elle ne permet pas de gérer plusieurs versions de Node.js facilement.

2. Utiliser un gestionnaire de versions de Node (nvm)

nvm (Node Version Manager) est un gestionnaire de versions Node.js qui permet d’installer et de basculer entre différentes versions de Node.js sur le même système.

Pour l'installer, téléchargez nvm depuis le dépôt officiel sur GitHub et suivez les instructions.

Une fois nvm installé, vous pouvez utiliser des commandes comme nvm install <version> pour installer une version spécifique de Node.js, et nvm use <version> pour basculer entre les versions.

Cette méthode est idéale pour les développeurs qui ont besoin de tester des applications sur différentes versions de Node.js.

. Installer avec Docker

Si vous travaillez avec des conteneurs, vous pouvez utiliser Docker pour créer un environnement Node.js sans l'installer directement sur votre système.

Créez un fichier Dockerfile avec l’image Node.js, par exemple :

dockerfile

FROM node:18

Cette méthode est utile pour isoler les environnements et garantir que votre application tourne de manière cohérente sur différents systèmes.

Ensuite pour vérifier que vous avez éffectivement installé Node.js, dans le cas d'un OS Windows :

- nous ouvrirons l'invite de commande encore plus communément appelé cmd pour taper ensuite les commandes :

- node -v qui nous affichera la version de Node.js présente dans notre système.

- npm -v qui nous affichera la version de Node Package Manager présente dans notre système.

II / Visual Studio Code

Visual Studio et Visual Studio Code sont deux outils de développement créés par Microsoft, mais ils ont des objectifs et des fonctionnalités différentes.

Visual Studio Code, souvent abrégé en VS Code, est un éditeur de code source léger et flexible, conçu pour être rapide et personnalisable. Contrairement à Visual Studio, VS Code est plutôt un éditeur de texte avancé qu'un IDE complet.

Fonctionnalités et utilisations principales de Visual Studio Code :

- Édition de code source : Il prend en charge une large gamme de langages de programmation, comme JavaScript, Python, TypeScript, Go, et bien d’autres, avec la possibilité d’ajouter des extensions pour d’autres langages.

- Personnalisation avec les extensions : VS Code a un écosystème d’extensions très riche, permettant aux développeurs d’ajouter des fonctionnalités supplémentaires, comme le débogage, le contrôle de version, le formatage de code, et même l’intégration avec des services cloud.

- Développement web et JavaScript : Il est très populaire pour le développement web grâce à des fonctionnalités intégrées comme l’auto-complétion, le débogage JavaScript, et l’intégration avec des outils de développement frontend.

- Interface de ligne de commande intégrée : Il offre un terminal intégré qui permet aux développeurs de lancer des commandes directement depuis l'éditeur, facilitant l'exécution de scripts, de commandes Git, ou encore de commandes de gestion de packages comme npm.

- Débogage : Il propose un environnement de débogage basique, idéal pour les projets JavaScript, Python, et Node.js, entre autres, avec des points d’arrêt et l’inspection des variables.

- Contrôle de version : Il intègre des outils de contrôle de version comme Git, ce qui permet aux développeurs de gérer leur code source directement dans l’éditeur.

Pourquoi utiliser Visual Studio Code ?

VS Code est largement utilisé pour son léger, sa vitesse et son extensibilité. Il est idéal pour :

- Les développeurs front-end et back-end.

- Les projets légers ou les prototypes.

-Ceux qui veulent un éditeur personnalisable avec des outils de développement modernes.

- Ceux qui recherchent un éditeur multiplateforme, car VS Code est disponible sur Windows, macOS et Linux.

En résumé, Visual Studio est un IDE complet pour des projets complexes, tandis que Visual Studio Code est un éditeur léger et modulaire, particulièrement adapté aux développeurs de web et de projets nécessitant de la flexibilité.

# Comment installer Visual Studio Code ?

Visual Studio code est en principe très facile d'acquisition et d'accès et nous avons la possibilité de l'obtenir en suivant les étapes qui suivent ce magnifique paragraphe introductif qui sera brutalement interrompu par :

**1. Accédez au site officiel**

* Rendez-vous sur le site officiel de Visual Studio Code en utilisant ce lien : [Visual Studio Code](https://code.visualstudio.com/).

**2. Sélectionnez votre système d'exploitation**

* Sur la page d'accueil, cliquez sur le bouton correspondant à votre système d'exploitation :
  + **Windows**
  + **macOS**
  + **Linux**

**3. Téléchargez l'installateur**

* Une fois que vous avez sélectionné votre système d'exploitation, le téléchargement du fichier d'installation commencera automatiquement.
* Si ce n’est pas le cas, cliquez sur le lien fourni pour démarrer le téléchargement.

**4. Installez Visual Studio Code**

**Pour Windows :**

1. Double-cliquez sur le fichier téléchargé (ex. VSCodeSetup.exe).
2. Suivez les instructions de l’assistant d’installation :
   * Acceptez les termes et conditions.
   * Sélectionnez les options comme l’ajout de VS Code au menu contextuel.
3. Cliquez sur **Installer**, puis sur **Terminer** une fois l’installation achevée.

**5. Lancez Visual Studio Code**

* Une fois installé, ouvrez l'application et explorez son interface conviviale.

CYPRESS

**Initialisation du projet**

1. **Créez un répertoire de projet** (si ce n'est pas déjà fait) :

mkdir mon-projet-cypress

cd mon-projet-cypress

1. **Initialisez un fichier package.json** dans le projet :

npm init -y

Cela génère un fichier package.json avec les configurations par défaut.

**3. Installation de Cypress**

1. Installez Cypress en tant que dépendance de développement avec npm :

npm install cypress --save-dev

1. Vérifiez que Cypress a été installé correctement en exécutant :

npx cypress verify

**4. Ouvrir Cypress**

1. Pour la première fois, ouvrez Cypress pour initialiser les fichiers de configuration :

npx cypress open

1. Une interface graphique s'ouvrira. Cypress crée automatiquement un répertoire cypress/ et un fichier de configuration cypress.config.js dans votre projet.

**6. Ajouter un script pour Cypress dans package.json**

Ajoutez une commande pour exécuter Cypress depuis le terminal :

"scripts": {

"test": "cypress run",

"cypress:open": "cypress open"

}

Cela vous permet de démarrer Cypress avec :

npm run cypress:open

**3. Intégration avec le pipeline CI/CD**

Cypress peut être intégré dans des outils de CI/CD comme Jenkins, GitHub Actions, GitLab CI, etc., pour exécuter vos tests automatiquement à chaque commit.

1. **Ajoutez Cypress aux scripts npm** : Modifiez votre package.json pour inclure les commandes suivantes :

"scripts": {

"test": "cypress run",

"cypress:open": "cypress open"

}

1. **Configurer Cypress pour CI/CD** :
   * Ajoutez un fichier de configuration spécifique à votre outil CI/CD.

name: Cypress Tests

on:

push:

branches:

- main

jobs:

cypress-run:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Setup Node.js

uses: actions/setup-node@v3

with:

node-version: 16

- name: Install dependencies

run: npm install

- name: Run Cypress tests

run: npm run test

**Résumé des bénéfices**

* **Installation simple** : Une commande suffit pour démarrer.
* **Configuration flexible** : Adaptez les paramètres à votre projet.
* **Intégration fluide avec CI/CD** : Automatisez vos tests pour garantir une livraison continue fiable.

**• Workflow de Tests : Description du processus de création, validation, et exécution des tests**

**Workflow de Tests avec Cypress**

Le workflow de tests avec Cypress est un processus structuré qui inclut la création, la validation, et l’exécution des tests.

**1.3 Rédaction des Tests**

Utilisez des commandes Cypress pour écrire des tests. Exemple d’un test de connexion :

describe('Test de connexion', () => {

it('Vérifie qu’un utilisateur peut se connecter avec des identifiants valides', () => {

cy.visit('/login'); // Accès à la page de connexion

cy.get('#email').type('utilisateur@example.com'); // Remplir le champ email

cy.get('#password').type('motdepasse123'); // Remplir le champ mot de passe

cy.get('button[type="submit"]').click(); // Soumettre le formulaire

cy.url().should('include', '/dashboard'); // Vérifier la redirection

});

});

**2. Validation des Tests**

**2.1 Exécution locale pour validation**

* Lancez les tests en local pour vérifier leur comportement.
* Commande pour exécuter les tests :

npx cypress open

* Choisissez le test dans l’interface graphique de Cypress.

**2.2 Débogage**

* Utilisez les outils intégrés de Cypress pour inspecter les tests.
  + **Console d’exécution** : Affiche les commandes exécutées et leurs résultats.
  + **Time Travel** : Permet de revoir chaque étape d’un test.
* Exemple de validation d’éléments :

cy.get('#message').should('contain.text', 'Connexion réussie');

**2.3 Refactorisation**

* Optimisez les tests pour qu’ils soient :
  + **Lisibles** : Utilisez des fonctions utilitaires.
  + **Réutilisables** : Centralisez les étapes répétées dans support/commands.js.

**3. Exécution des Tests**

**3.1 Exécution locale**

* Exécutez les tests via l’interface graphique ou en ligne de commande :

npx cypress run

**3.3 Génération de rapports**

* Utilisez des outils comme **Mochawesome** pour créer des rapports détaillés :

npm install mochawesome --save-dev

**3.4 Surveillance continue**

* Enregistrez les résultats des tests (logs, vidéos, captures d’écran) pour identifier les régressions dans les versions ultérieures.

**Importance du Workflow**

1. **Création structurée** :
   * Réduit les erreurs dans la rédaction des tests.
   * Assure une couverture complète des fonctionnalités critiques.
2. **Validation efficace** :
   * Facilite la détection et la correction des erreurs avant l’intégration.
3. **Exécution automatisée** :
   * Permet un feedback rapide sur la qualité du code.
   * Garantit une livraison continue fiable.