TP1 - Mise en place de l'environnement de test TypeScript

Objectifs

- Installer **Node.js** et le gestionnaire de paquets **npm**
- Creer un projet TypeScript from scratch
- Installer et configurer **Vitest**, le framework de test
- Ecrire et lancer votre **premier test unitaire**
- Comprendre la structure d'un projet de test TypeScript

Etape 1 - Installer Node.js

Node.js est le moteur d'execution JavaScript/TypeScript cote serveur. Il inclut `npm`, le gestionnaire de paquets.

Installation

Rendez-vous sur https://nodejs.org et telechargez la version **LTS** (Long Term Support).

Ou, si vous utilisez un gestionnaire de versions (recommande) :

```
# Sur macOS / Linux avec nvm
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh |
nvm install --lts
nvm use --lts
```

Verification

```
node --version   # Doit afficher v20.x.x ou superieur
```

```
npm --version    # Doit afficher 10.x.x ou superieur
```

Si ces commandes affichent un numero de version, Node.js est correctement installe.

Etape 2 - Creer le projet

Creez un nouveau dossier et initialisez un projet npm :

```
mkdir mon-premier-projet-test
cd mon-premier-projet-test
npm init -y
```

La commande `npm init -y` cree un fichier `package.json` avec des valeurs par defaut. Ce fichier liste les dependances et les scripts du projet.

Verification

```
ls          # Doit afficher : package.json
cat package.json  # Affiche le contenu du fichier
```

Etape 3 - Installer TypeScript

TypeScript est un sur-ensemble de JavaScript qui ajoute le typage statique. Il se transpile en JavaScript avant l'execution.

```
npm install --save-dev typescript @types/node
```

L'option `--save-dev` (ou `-D`) indique que c'est une dependance de developpement, pas une dependance de production.

Creez le fichier de configuration TypeScript `tsconfig.json` :

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ESNext",
    "moduleResolution": "Bundler",
    "strict": true,
    "outDir": "dist",
    "rootDir": "src"
  },
  "include": ["src/**/*", "tests/**/*"]
}
```

Verification

```
npx tsc --version   # Doit afficher Version 5.x.x
```

Etape 4 - Installer Vitest

Vitest est le framework de test. Il est rapide, supporte TypeScript nativement, et propose une API proche de Jest.

```
npm install --save-dev vitest
```

Ajoutez les scripts de test dans `package.json` :

```
{
  "scripts": {
    "test": "vitest",
    "test:run": "vitest run"
  }
}
```

- `npm test` (ou `vitest`) : lance les tests en mode **watch** — re-execute les tests a chaque modification de fichier
- `npm run test:run` (ou `vitest run`) : execute les tests **une seule fois** et quitte

Verification

```
npm run test:run
```

Vitest doit demarrer et afficher :

```
No test files found, exiting with code 1
```

C'est normal — vous n'avez pas encore de tests. Vitest est bien installe.

Etape 5 - Creer la structure du projet

Creez les dossiers `src/` et `tests/` :

```
mkdir src tests
```

Structure cible :

```
mon-premier-projet-test/
├── src/
│   └── Calculator.ts       ← votre code de production
├── tests/
│   └── Calculator.test.ts  ← vos tests
├── package.json
└── tsconfig.json
```

Etape 6 - Ecrire le premier code

Creez le fichier `src/Calculator.ts` :

```
export class Calculator {
    add(a: number, b: number): number {
        return a + b;
    }

    subtract(a: number, b: number): number {
        return a - b;
    }
}
```

Etape 7 - Ecrire le premier test

Creez le fichier `tests/Calculator.test.ts` :

```
import { describe, it, expect } from 'vitest';
import { Calculator } from '../src/Calculator';

describe('Calculator', () => {
    it('add - 1 + 2 retourne 3', () => {
        // Arrange
        const calculator = new Calculator();

        // Act
        const result = calculator.add(1, 2);

        // Assert
        expect(result).toBe(3);
    });

    it('subtract - 5 - 3 retourne 2', () => {
        const calculator = new Calculator();

        const result = calculator.subtract(5, 3);

        expect(result).toBe(2);
```

```
    });
});
```

Comprendre la structure

Element	Role
`describe('Calculator', ...)`	Regroupe les tests d'une meme classe
`it('add - 1 + 2 retourne 3', ...)`	Declare un test avec un nom descriptif
`// Arrange`	Preparation des donnees et objets
`// Act`	Execution du code a tester
`// Assert`	Verification du resultat avec `expect`
`expect(result).toBe(3)`	Assertion : verifie que `result` vaut `3`

Etape 8 - Lancer les tests

```
npm run test:run
```

Vous devriez voir :

```
✓ tests/Calculator.test.ts (2)
  ✓ Calculator > add - 1 + 2 retourne 3
  ✓ Calculator > subtract - 5 - 3 retourne 2

Test Files  1 passed (1)
Tests       2 passed (2)
```

Felicitations ! Vos deux premiers tests passent au vert.

Mode watch

Lancez maintenant :

```
npm test
```

Vitest reste en attente. Modifiez `src/Calculator.ts` et changez `return a + b` en `return a - b`. Sauvegardez. Vitest re-execute les tests automatiquement et affiche un echec.

Corrigez le code. Les tests repassent au vert automatiquement.

Etape 9 - Faire echouer un test intentionnellement

Pour comprendre les messages d'erreur, modifiez un test pour qu'il echoue :

```
it('add - 1 + 2 retourne 3', () => {
    const calculator = new Calculator();
    const result = calculator.add(1, 2);
    expect(result).toBe(99); // Intentionnellement faux
});
```

Lancez `npm run test:run`. Lisez le message d'erreur :

```
AssertionError: expected 3 to be 99
  - Expected: 99
  + Received: 3
```

Vitest vous montre **la valeur attendue** et **la valeur recue**. Revertez la modification.

Recapitulatif — Ce que vous avez installe

Outil	Role	Commande d'installation

Node.js	Moteur d'execution JavaScript/TypeScript	Depuis nodejs.org ou nvm
npm	Gestionnaire de paquets	Inclus avec Node.js
TypeScript	Typage statique pour JavaScript	`npm install -D typescript`
@types/node	Types TypeScript pour Node.js	`npm install -D @types/node`
Vitest	Framework de test	`npm install -D vitest`

Recapitulatif — Ce que vous avez appris

Concept	Description
`package.json`	Fichier de configuration du projet : dependances, scripts
`tsconfig.json`	Configuration du compilateur TypeScript
`npm install -D`	Installe une dependance de developpement
`npm test`	Lance Vitest en mode watch
`npm run test:run`	Lance Vitest une seule fois
`describe(...)`	Regroupe des tests logiquement
`it(...)`	Declare un test
`expect(...).toBe(...)`	Assertion d'egalite stricte
Pattern AAA	Arrange / Act / Assert — structure d'un test

Pour aller plus loin

Pour les prochains TPs, vous utiliserez ces commandes frequemment :

```
npm install -D <package>   # Installer une nouvelle dependance de test
npm run test:run           # Verifier que tous les tests passent
npm test                   # Mode watch pendant le developpement
npx tsc --noEmit           # Verifier les types sans compiler
```