Fiche résumé : Testez une application full-stack avec Vitest et Playwright 1.0

1. Découvrir les niveaux de test et le rôle de Vitest
2. Installer et configurer Vitest dans un projet Next.js
3. Écrire des tests unitaires avec la méthode AAA (Préparer, Agir, Vérifier)
4. Tester les routes API Next.js avec des tests d'intégration
5. Valider l'interface avec RTL puis les parcours E2E avec Playwright

Code 💻

TypeScript

```ts
describe("addition", () => {
  it("should add two numbers", () => {
    expect(1 + 1).toBe(2);
  });
});
```

```ts
const result = await hashPassword(password);
expect(result).not.toBe(password);
```

```ts
beforeEach(() => {
  resetStore();
});
```

```ts
const onSubmit = vi.fn();
render(<TaskForm onSubmit={onSubmit} />);

await user.type(
  screen.getByRole("textbox", { name: /titre/i }),
  "Réviser le chapitre",
);
await user.click(screen.getByRole("button", { name: /ajouter la tâche/i }));

expect(onSubmit).toHaveBeenCalledWith({ title: "Réviser le chapitre" });
```

Bonnes pratiques 👍

- ✅ Vérifier l'environnement avec un test simple avant les tests métier
- ✅ Structurer les tests selon le modèle Arrange Act Assert (Préparer, Agir, Vérifier)
- ✅ Donner des noms explicites aux scénarios de test
- ✅ Réinitialiser les données avec `beforeEach` et `resetStore`
- ✅ Utiliser `userEvent` pour simuler les actions utilisateur
- ✅ Privilégier `getByRole` et `getByLabelText` pour les requêtes RTL
- ✅ Employer des locators sémantiques dans Playwright

Erreurs classiques 👎

- ❌ Comparer un mot de passe haché au mot de passe original
- ❌ Modifier le code sans analyser le message d'erreur
- ❌ Oublier de réinitialiser le store entre les tests
- ❌ Utiliser des noms de test vagues comme test 1
- ❌ Déclencher manuellement des événements au lieu de `userEvent`
- ❌ Privilégier `data-testid` sans nécessité
- ❌ Utiliser des sélecteurs CSS fragiles dans Playwright

Définitions 🔍

Test unitaire
test vérifiant une fonction isolée du reste de l'application.

Test d'intégration back-end
test validant le comportement complet d'une route API.

Test de composant
test vérifiant l'affichage et les interactions d'un composant React.

Test end-to-end (E2E)
test reproduisant un parcours utilisateur complet.