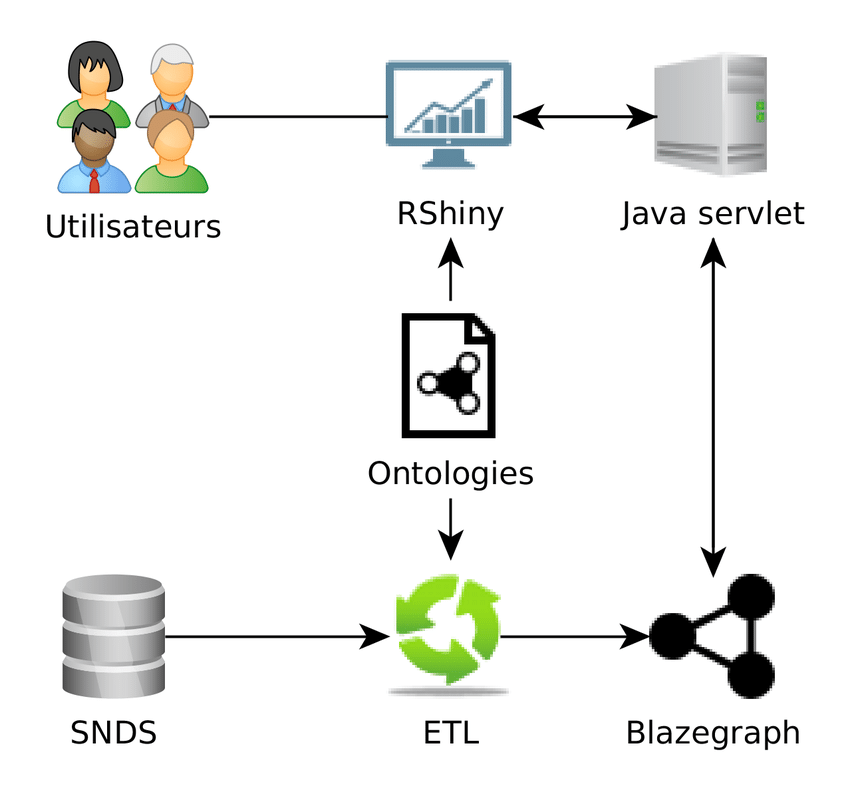
**PROJET 5 : TEST AVEC JEST**

**THEM :** **Mettre en place un environnement de test avec jest sur un projet.**

1. INTRODUCTION
2. INTERRET DES TEST ET LES FRAMEWORK QUI EXISTE POUR TESTER
3. PRESENTATION DE JEST
4. COMMENT SE SERVIR DE JEST POUR TESTER DU JS QUI A ETE ECRIT DANS UN PROJET
5. CONCLUSION
6. **INTRODUCTION**

Durent notre parcours de développeur, nous serons amenés à produire du code pour satisfaire les besoins des utilisateurs. De ce fait, pour garantir la qualité du produit offert à ceux-ci, notre code doit être soumis à des test.

Lors du développement de logiciels, il est essentiel de s'assurer de la qualité du code  et du bon fonctionnement des fonctionnalités . Pour ce faire, la mise en place d'un environnement de test solide est cruciale. Jest est un framework de test populaire pour JavaScript, réputé pour sa simplicité et sa puissance. Il offre des fonctionnalités avancées telles que le mocking, les assertions, les tests unitaires et les tests d'intégration.

Dans cet article, nous allons présenter l’intérêt des tests et les framework qui existent pour tester, présenter jest et comment s’en servir pour tester du javascript qui a été écrit dans un projet. En utilisant Jest, nous serons en mesure de valider le comportement attendu de notre code et de détecter rapidement les éventuelles erreurs ou régressions.

Que vous soyez un développeur débutant ou expérimenté, ce guide vous accompagnera dans la mise en place d'un environnement de test robuste avec Jest, vous permettant ainsi d'améliorer la fiabilité de votre code et de gagner en confiance lors de vos développements. Prêt à plonger dans le monde des tests automatisés avec Jest ?

1. **INTERRET DES TEST ET LES FRAMEWORK QUI EXISTE POUR TESTER**
2. **INTERRET DES TEST**

Les tests offrent de nombreux avantages et sont essentiels dans le développement de logiciels. Voici quelques-uns des principaux intérêts des tests :

1. **Détection précoce des erreurs**

Les tests permettent de détecter les erreurs et les bugs potentiels dès le début du processus de développement. Cela permet de corriger les problèmes plus rapidement et de réduire les coûts associés aux corrections tardives.

1. **Garantie de qualité**

Les tests garantissent la qualité du code en vérifiant que les fonctionnalités se comportent comme prévu. Ils permettent de valider les résultats attendus et de s'assurer que le code répond aux spécifications.

1. **Facilitation des changements et des évolutions**

Les tests facilitent les modifications et les évolutions du code en offrant une sécurité. Lorsque vous apportez des modifications, vous pouvez exécuter les tests pour vous assurer que les fonctionnalités existantes n'ont pas été affectées.

1. **Documentation vivante**

Les tests servent de documentation vivante pour le code. Ils décrivent comment les différentes parties du code doivent fonctionner et peuvent être utilisés comme référence pour comprendre l'intention et le comportement du code.

1. **Confiance accrue dans le code**

Les tests automatisés fournissent une confiance dans le fonctionnement du code. Ils permettent aux développeurs de faire des changements sans craindre de casser des fonctionnalités existantes, car les tests servent de filet de sécurité.

1. **Meilleure collaboration en équipe**

Les tests facilitent la collaboration en équipe en offrant une base commune pour évaluer la qualité du code. Les tests définissent les attentes et les critères d'acceptation, ce qui permet aux membres de l'équipe de travailler plus efficacement ensemble.

1. **Réduction des erreurs humaines**

Les tests automatisés réduisent les erreurs humaines en exécutant systématiquement les vérifications et les assertions. Cela évite les oublis et les erreurs humaines qui pourraient survenir lors de tests manuels.

1. **Maintenance plus facile**

Les tests rendent la maintenance du code plus facile en identifiant rapidement les régressions ou les problèmes potentiels. Ils permettent de diagnostiquer les erreurs plus rapidement et d'effectuer des corrections plus précises.

En résumé, les tests améliorent la qualité du code, permettent une maintenance plus facile, offrent une confiance dans le fonctionnement du logiciel et facilitent la collaboration en équipe. Ils sont un investissement précieux pour garantir la robustesse et la fiabilité des applications logicielles.

1. **LES FRAMEWORK QUI EXISTE POUR TESTER**

Il existe de nombreux frameworks de test disponible pour tester des applications dans différents langages de programmation. Voici une liste de quelques-uns des frameworks de test populaire pour différents langages

1. **JUnit**

Un framework de test pour Java, utilisé principalement pour les tests unitaires.

1. **TestNG**

Un autre framework de test pour Java qui prend en charge les tests unitaires, les tests d'intégration et les tests de bout en bout.

1. **Pytest**

Un framework de test pour Python, connu pour sa simplicité et sa puissance. Il prend en charge les tests unitaires, les tests d'intégration et les tests fonctionnels.

1. **RSpec**

Un framework de test pour Ruby, principalement utilisé pour les tests comportementaux (Behavior-Driven Development - BDD).

1. **Mocha**

Un framework de test pour JavaScript, utilisé pour les tests côté client et côté serveur. Il prend en charge les tests asynchrones et offre une grande flexibilité.

1. **Jest**

Un framework de test pour JavaScript, spécifiquement conçu pour les applications basées sur React. Il offre une syntaxe simple, des fonctionnalités avancées telles que le mocking et une intégration transparente avec les outils populaires.

1. **PHPUnit**

Un framework de test pour PHP, utilisé pour les tests unitaires et les tests d'intégration.

1. **Cucumber**

Un framework de test basé sur le langage Gherkin, utilisé pour l'automatisation des tests fonctionnels et la collaboration entre les équipes de développement et de test.

1. **Cypress**

Un framework de test pour JavaScript, utilisé pour les tests d'interface utilisateur. Il permet de simuler et de tester les interactions utilisateur de manière automatisée.

1. **Selenium**

Un framework de test pour les tests d'interface utilisateur. Il permet d'automatiser les interactions avec les navigateurs web.

Ces frameworks de test sont quelques exemples parmi de nombreux autres disponibles dans la communauté des développeurs. Chaque framework a ses propres caractéristiques, syntaxes et approches, il est donc important de choisir celui qui convient le mieux à votre langage de programmation, à votre environnement de développement et à vos besoins spécifiques en matière de tests. Dans cet article, notre choix est fait sur jest. La suite vous donnera un parfait éclaircissement sur ce choix.

1. **PRESENTATION DE JEST**

Jest est un framework de test JavaScript largement utilisé et populaire développé principalement pour tester les applications React, mais il peut également être utilisé pour tester d'autres types d'applications JavaScript, y compris les applications Node.js. Il offre un ensemble de fonctionnalités et d'outils pour faciliter l'écriture, l'exécution et l'analyse des tests JavaScript. Jest est un Framework de test JavaScript de qualité qui met l'accent sur la simplicité. Il est un **Framework** parce qu’il est à la fois un **test Runner (exécuteur de test)** et une **assertion librairie (c.-à-d. il nous dit en sorti ce qu’il attendait et ce qu’il a reçu).** Il fonctionne avec les projets utilisant:  [Babel](https://babeljs.io/),  [TypeScript](https://www.typescriptlang.org/" \t "_blank),  [Node](https://nodejs.org/),  [React](https://reactjs.org/" \t "_blank), [Angular](https://angular.io/" \t "_blank), [Vue](https://vuejs.org/) et plus encore. Voici quelques-unes des principales utilisations et fonctionnalités de Jest :

1. **Tests unitaires :**

Jest permet d'écrire et d'exécuter des tests unitaires pour les différentes parties de votre code JavaScript, qu'il s'agisse de fonctions, de classes, de modules ou de composants. Il fournit une syntaxe simple et expressive pour écrire des assertions et des expectations afin de vérifier le comportement attendu de votre code.

1. **Mocks et stubs :**

Jest offre des fonctionnalités intégrées pour la création de mocks, de stubs et de spires. Vous pouvez simuler des dépendances, espionner les appels de fonction, et définir des comportements personnalisés pour des objets ou des fonctions, ce qui facilite l'isolation des tests et la simulation de différents scénarios.

1. **Tests asynchrones :**

Jest prend en charge les tests asynchrones de manière native. Vous pouvez utiliser des promesses, des callbacks ou des fonctions asynchrones pour effectuer des assertions sur des opérations asynchrones, et Jest gère automatiquement la gestion du temps et la synchronisation des tests.

1. **Snapshot testing :**

Jest propose une fonctionnalité appelée "Snapshot testing" qui permet de capturer des snapshots (captures d'écran) des sorties de votre code lors des tests, puis de les comparer aux snapshots précédents pour détecter les changements. Cela facilite la détection des régressions et la validation des changements attendus dans votre code.

1. **Couverture de code :**

Jest fournit des rapports de couverture de code intégrés qui vous permettent de savoir quelle partie de votre code est couverte par vos tests. Vous pouvez obtenir des métriques détaillées sur la couverture de lignes, de branches et de fonctions, ce qui facilite l'identification des zones de code non testées.

1. **Intégration transparente :**

Jest s'intègre facilement avec d'autres outils et bibliothèques populaires de l'écosystème JavaScript. Il fonctionne bien avec Babel pour la prise en charge des dernières fonctionnalités de JavaScript, avec webpack pour la gestion des dépendances, et il est compatible avec de nombreux outils de CI/CD tels que Jenkins, Travis CI et CircleCI.

En résumé, Jest est utilisé pour écrire, exécuter et analyser des tests JavaScript. Il fournit une syntaxe simple, des fonctionnalités avancées de mocking et de test asynchrone, ainsi que des outils intégrés pour la couverture de code et les tests de snapshots. Jest est apprécié pour sa simplicité d'utilisation, sa rapidité d'exécution et son intégration transparente avec d'autres outils populaires, ce qui en fait un choix courant pour les tests unitaires et d'intégration dans le développement JavaScript.

1. **COMMENT SE SERVIR DE JEST POUR TESTER DU JS QUI A ETE ECRIT DANS UN PROJET**

Pour tester du code JavaScript écrit dans un projet en utilisant jest, vous pouvez suivre ces étapes :

1. **Initialisation du projet**

Tout d'abord, vous devez initialiser votre projet s’il ne l’est pas encore via la commande **npm init**, afin que l’installation des dépendances puisse etre possible

1. **Installation de Jest**

Vous devez installer Jest dans votre projet. Vous pouvez le faire en exécutant la commande suivante dans votre terminal :

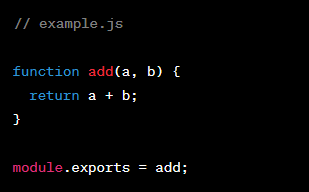


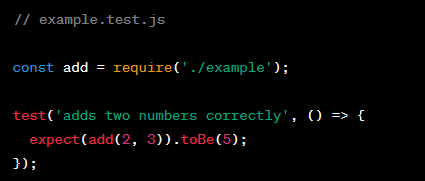
Cela installera Jest en tant que dépendance de développement dans votre projet.

**2. Écriture de tests**

Une fois Jest installé, vous pouvez commencer à écrire vos tests. Les fichiers de test Jest doivent être placés dans un répertoire nommé **`\_\_tests\_\_`** ou avoir l'extension **`.test.js`** ou **`.spec.js`**. Par exemple, si vous avez un fichier `**example.js**` que vous souhaitez tester, vous pouvez créer un fichier `**example.test.js**` pour y écrire vos tests. Dans vos fichiers de test, vous pouvez utiliser les fonctions de Jest, telles que **`test()`** ou **`it()`**, pour définir des cas de test. Vous pouvez utiliser des assertions ou des expectations pour vérifier le comportement attendu de votre code.

Voici un exemple simple de test :





**3. Exécution des tests**

Une fois vos tests écrits, vous pouvez exécuter Jest pour les exécuter. Vous pouvez le faire en exécutant la commande suivante dans votre terminal :

**npm jest**

Jest exécutera tous les fichiers de test présents dans votre projet et affichera les résultats dans votre terminal. Il indiquera les tests réussis et échoués, ainsi que les erreurs éventuelles.

**4. Options de configuration**

Jest offre de nombreuses options de configuration pour personnaliser son comportement. Par exemple, vous pouvez configurer des chemins de fichiers spécifiques, définir des options de couverture de code, spécifier des configurations de test asynchrone, et bien plus encore.

Vous pouvez créer un fichier `**jest.config.js**` à la racine de votre projet pour définir ces options.

Ces étapes vous permettent de commencer à utiliser Jest pour tester votre code JavaScript dans votre projet. Vous pouvez écrire des tests pour différentes parties de votre code, y compris des fonctions, des classes, des modules ou des composants.

1. **CONCLUSION**

Jest facilite l'écriture de tests avec sa syntaxe simple, ses fonctionnalités de mocking et sa prise en charge des tests asynchrones, tout en offrant des rapports détaillés sur la couverture de code.