# Recherche sur les framesworks utilisable sur Angular pour la création et l’execution de test unitaire :

## Liste de frameworks utilisés sur Angular :

Il y a plusieurs frameworks de test unitaire pour Angular, voici quelques exemples courants:

* **Jasmine :** un framework de test unitaire populaire pour JavaScript en général, qui est également compatible avec Angular. Il offre une syntaxe simple et une intégration facile avec les outils de construction tels que Karma.
* **Karma :** un outil de lancement de navigateur pour les tests unitaires JavaScript. Il peut être utilisé avec Jasmine ou d'autres frameworks de test.
* **Protractor :** un framework de test automatisé pour les applications Angular qui utilise WebDriverJS pour simuler des interactions utilisateur dans un navigateur. Il est conçu pour tester des applications Angular spécifiques.
* **Jest :** un framework de test de JavaScript utilisé pour les applications React, mais peut aussi être utilisé pour les tests unitaires sur Angular. Il a une intégration native avec les outils de construction de TypeScript et les dépendances de l'application sont automatiquement mocks.
* **Mocha :** un framework de test JavaScript général qui peut être utilisé avec Angular en combinaison avec d'autres outils tels que Chai ou Sinon.js pour les assertions et les spies.
* **ng-test-runner :** un outil de test unitaire pour Angular qui utilise Mocha et se concentre sur la simplicité d'utilisation et la rapidité de configuration.
* **TestBed :** un outil de test unitaire fourni par Angular lui-même, qui permet de tester les composants et les services de l'application de manière isolée.
* **Cypress :** Un framework de test automatisé pour les applications web, il permet de créer des tests d'interface utilisateur en utilisant une syntaxe simple, il est facile à installer et à utiliser. Il peut être utilisé pour tester des applications AngularJS et Angular.

Il existe encore d'autres frameworks de test unitaire disponibles pour Angular, mais ces options sont également populaires et bien documentées. Il est important de choisir celui qui convient le mieux à vos besoins de test spécifiques.

Source : Information tiré de Chat GPT

## Avantage et inconvénients des frameworks les plus utilisés sur Angular :

Jasmine :

**Avantages :**

* Simple à utiliser et à comprendre pour les développeurs qui n'ont pas d'expérience en matière de test unitaire.
* Intégration facile avec Karma pour lancer les tests dans des navigateurs différents.

**Inconvénients :**

* Peut ne pas être adapté aux applications Angular plus complexes nécessitant des tests d'intégration et de performance.

Karma :

**Avantages :**

* Permet de lancer des tests unitaires sur différents navigateurs et plateformes.
* Facile à configurer et à utiliser.

**Inconvénients :**

* Peut nécessiter des plugins supplémentaires pour intégrer des frameworks de test tels que Jasmine.

Jest :

**Avantages :**

* Intégration native avec les outils de construction de TypeScript et les dépendances de l'application sont automatiquement mocks
* Il a une forte communauté et de bonnes performances.

**Inconvénients :**

* Il est conçu pour les applications React, il peut être plus difficile à utiliser pour les applications Angular spécifiques.

Protractor :

**Avantages :**

* Conçu pour tester les applications Angular spécifiques en simulant des interactions utilisateur.
* Intégration native avec WebDriverJS pour les tests d'end-to-end.

**Inconvénients :**

* Peut-être plus difficile à configurer et à utiliser pour les développeurs qui n'ont pas d'expérience avec WebDriverJS.

Mocha :

**Avantages :**

* Un framework de test JavaScript général qui peut être utilisé avec Angular en combinaison avec d'autres outils tels que Chai ou Sinon.js pour les assertions et les spies.
* Il est facile à utiliser et à configurer.

**Inconvénients :**

* Il n'est pas spécifique à Angular, il peut manquer certaines fonctionnalités pour les applications Angular spécifiques.

ng-test-runner :

**Avantages :**

* Un outil de test unitaire pour Angular qui utilise Mocha et se concentre sur la simplicité d'utilisation et la rapidité de configuration.
* Il est facile à utiliser et à configurer.

**Inconvénients :**

* Il est un outil de test unitaire pour Angular qui utilise Mocha, il peut manquer certaines fonctionnalités pour les applications Angular spécifiques.

TestBed :

**Avantages :**

* Un outil de test unitaire fourni par Angular lui-même, qui permet de tester les composants et les services de l'application de manière isolée.
* Il a une forte intégration avec Angular et permet de tester les composants de l'application de manière isolée.

**Inconvénients :**

* Peut-être plus difficile à utiliser pour les développeurs qui n'ont pas d'expérience avec les outils de test Angular.

Cypress :

**Avantages :**

* Il permet de créer des tests d'interface utilisateur en utilisant une syntaxe simple, il est facile à installer et à utiliser.
* Il a une forte communauté et de bonnes performances.

**Inconvénients :**

* Il est principalement conçu pour les applications web, il peut manquer certaines fonctionnalités pour les applications Angular spécifiques.

Il est important de noter que chaque framework a ses propres avantages et inconvénients et que le choix dépendra des besoins spécifiques de votre application et de votre équipe de développement. Il est également important de considérer la facilité d'utilisation, la communauté de développeurs et la compatibilité avec d'autres outils et technologies lors de la sélection d'un framework de test unitaire pour Angular.

## Quel framework est le meilleur pour réaliser les tests unitaires d’un ERP ?

Il n'y a pas de réponse unique à cette question car cela dépend des besoins spécifiques de votre ERP et de votre équipe de développement. Cependant, voici quelques considérations pour aider à faire un choix :

* Jasmine ou Jest : Ces frameworks sont populaires pour les tests unitaires JavaScript en général et peuvent être utilisés pour tester des fonctionnalités spécifiques de votre ERP. Ils offrent une syntaxe simple et une intégration facile avec d'autres outils de construction.
* TestBed : Ce framework est fourni par Angular lui-même et permet de tester les composants et les services de l'application de manière isolée. Il est particulièrement adapté pour les tests unitaires des composants Angular de l'ERP.
* Cypress : Il permet de créer des tests d'interface utilisateur en utilisant une syntaxe simple, il est facile à installer et à utiliser. Il est particulièrement adapté pour les tests d'interface utilisateur et les tests d'intégration.
* Protractor : Il est conçu pour tester les applications Angular spécifiques en simulant des interactions utilisateur. Il peut être utilisé pour les tests d'intégration et de performance de votre ERP.

# Glossaire :

**Test unitaire :**

Le test unitaire est un type de test de logiciel dans lequel des unités ou des composants individuels sont testés. L’objectif principal des tests unitaires est de valider chaque unité du code logiciel et de déterminer si elle fonctionne conformément aux attentes.

Objectifs :

Tester les unités de code de manière isolée.

Vérifier si le code est correct.

Tester chaque fonction ainsi que la procédure.

Détecter les bogues plus tôt et les corriger dans le cycle de développement tout en réduisant les coûts.

Aider les développeurs à comprendre le code de base et leur permet d’apporter rapidement des modifications.

Aider également à la réutilisation du code.

Le but ultime des tests unitaires est de décomposer chaque partie des codes sources en unités et de voir si chaque partie fonctionne correctement. Cela signifie que si un ensemble d’entrées n’est pas fonctionnel, la sortie de retour est attendue.

Source : <https://mobiskill.fr/blog/conseils-emploi-tech/tests-unitaires-vs-tests-fonctionnels-quelles-differences/>

**Test Fonctionnel :**

Les tests fonctionnels sont définis comme une méthode permettant de tester la fonctionnalité d’une application logicielle. Le plus souvent, les tests fonctionnels sont utilisés pour vérifier des scénarios ou des modèles d’utilisation de bout en bout. Les tests fonctionnels peuvent aller du simple chargement de page et des scénarios de « happy path » jusqu’aux critères d’acceptation complets. Des documents d’exigences détaillés (qui peuvent inclure des cas d’utilisation, des user cases ou des scénarios) sont souvent nécessaires pour comprendre pleinement ce qui doit être testé. L’exécution des tests fonctionnels peut prendre plus de temps que celle des tests unitaires et peut impliquer l’utilisation de dépendances externes.

Source : <https://mobiskill.fr/blog/conseils-emploi-tech/tests-unitaires-vs-tests-fonctionnels-quelles-differences/>

Les tests fonctionnels sont généralement écrits par des analystes métier, des QA et des testeurs pour vérifier que l’application logicielle fonctionne correctement du point de vue de l’utilisation.

**Mocks :**

En programmation orientée objet, les mocks (simulacres ou mock object) sont des objets simulés qui reproduisent le comportement d'objets réels de manière contrôlée. Un programmeur crée un mock dans le but de tester le comportement d'autres objets, réels, mais liés à un objet inaccessible ou non implémenté. Ce dernier est alors remplacé par un mock.

Source Wikipédia : [Mock (programmation orientée objet) — Wikipédia (wikipedia.org)](https://fr.wikipedia.org/wiki/Mock_(programmation_orient%C3%A9e_objet))

System under test ou SUT

L’objet ou l’ensemble d’objets que l’on souhaite tester.

Un « mock » ou « imitateur » est un objet utilisé par le SUT et se comportant d’une manière prédéfinie de manière déclarative par le testeur. Il ne peut jamais s’agir d’un objet réel du métier. Le scénario de son imitation est configuré, réalisé, vérifié et remis à zéro pour chaque méthode de test. Toute discordance entre le scénario prévu et l’imitation exécutée donne lieu à une erreur.

Une forme spécifique de doublure.

Source : <https://blog.zenika.com/2013/08/28/mocks-versus-stubs/>

**Stubs :**

Un module logiciel simulant à minima la présence d'un autre, soit parce que cet autre composant est indisponible, soit parce qu'il n'est pas encore connu. Ce remplacement sert principalement lors de phase de test ou de portage ;

Source : <https://fr.wikipedia.org/wiki/Stub>

Un « stub » ou « bouchon » est un objet utilisé par le SUT et se comportant d’une manière prédéfinie par le testeur. Il peut dans certains cas s’agir d’un objet réel du métier, souvent configuré par le setup du test, ou bien d’un objet de remplacement et dont l’implémentation de test est à écrire pour le test.

Une forme spécifique de doublure

Source : <https://blog.zenika.com/2013/08/28/mocks-versus-stubs/>

**Fixture :** Une fixture est un morceau de code qui permet de fixer un environnement logiciel pour exécuter des tests logiciels. Cet environnement constant est toujours le même à chaque exécution des tests. Il permet de répéter les tests indéfiniment et d'avoir toujours les mêmes résultats.

Source : [Test fixture — Wikipédia (wikipedia.org)](https://fr.wikipedia.org/wiki/Test_fixture#:~:text=Une%20fixture%20est%20un%20morceau%20de%20code%20qui,tests%20ind%C3%A9finiment%20et%20d%27avoir%20toujours%20les%20m%C3%AAmes%20r%C3%A9sultats.)

Les fixtures ou données de tests permettent au développeur d’initialiser les données de la base de données dans un état connu. Par exemple, pour l’entité User, il serait possible de charger plusieurs utilisateurs avec différents rôles et caractéristiques. On pourrait créer un administrateur, un utilisateur standard et un utilisateur avec un accès désactivé. Par la suite, il serait possible d’écrire des tests fonctionnels afin de vérifier que notre page d’authentification (login) fonctionne bien avec tous ces types d’utilisateurs.

Dans bien des cas, il faudra recharger les fixtures entre chaque test. De cette manière, les effets de bord seront évités. Un état stable et connu sera initialisé entre chaque test.

Source : <https://atomrace.com/creer-des-fixtures-pour-faciliter-les-tests/>

**Effet de bords :** Effet de bord (informatique) - Définition

En informatique, une fonction est dite à effet de bord si elle modifie un état autre que sa valeur de retour. Par exemple, une fonction peut modifier une variable statique ou globale, modifier un ou plusieurs de ses arguments, écrire des données vers un écran ou un fichier ou lire des données provenant d'autres fonctions à effet de bord. Les effets de bord rendent souvent le comportement des programmes plus difficiles à comprendre.

Effet de bord est un barbarisme né d'une traduction fautive de l'anglais side-effect. (Le mot side signifiant " côté " et non pas " bord ", une traduction littérale et correcte aurait donné " effet latéral " ; cependant, en français, on utilise plus fréquemment l'expression synonyme " effet secondaire ".)

La [programmation impérative](https://www.techno-science.net/definition/5384.html) emploie des effets de bord dans le fonctionnement de ses programmes. La [programmation fonctionnelle](https://www.techno-science.net/glossaire-definition/Programmation-fonctionnelle.html) cherche à minimiser les effets de bord et les isole souvent dans des structures prévues notamment pour cela : les monades.

Source :

<https://www.techno-science.net/definition/11312.html>

Glossaire Angular :

<https://runebook.dev/fr/docs/angular/guide/glossary>

Voir ça pour rapport de stage :

<https://guide-angular.wishtack.io/angular/routing/lazy-loading>

<https://guide-angular.wishtack.io/angular/http/utilisation-de-httpclient>

<https://guide-angular.wishtack.io/angular/dependency-injection/tree-shakable-services>

<https://guide-angular.wishtack.io/angular/pipes>

<https://guide-angular.wishtack.io/angular/change-detection/immutabilite>

J’ai dû installer karma-coverage-istanbul-reporter dans tilia avec cette commande :

npm install karma-coverage-istanbul-reporter --save-dev

Critères :

Fréquence de mise à jour

Communauté

Est-ce que y’a de la doc ?

Compréhensible ?

Type License ?

Dépendance ? Si oui pq ?

Est-ce que, on peut faire tous les types de test qu’on veut ?

L’un ou les deux ?

Performance ? Et pourquoi ?

Vérifier si ça fonctionne avec les composants stand alone

Resolver, pipes

Mise à jour des dépendances

Couverture de code

Ajout de fonctionnalités

Tuto jasmine, karma et test bed un peu:

<https://codecraft.tv/courses/angular/unit-testing/jasmine-and-karma/>

<https://blog.soat.fr/2018/02/tests-unitaires-avec-angular-partie-1/>

<https://blog.soat.fr/2018/02/tests-unitaires-avec-angular-partie-2/>

<https://angular.io/guide/testing>

<https://www.ux-republic.com/fiche-recapitulative-des-tests-unitaires-en-angular/>

<https://scriptverse.academy/tutorials/jasmine-spyon.html>

<https://blog.logrocket.com/angular-unit-testing-tutorial/>

<https://cdiese.fr/angular-tests/>

<https://www.testim.io/blog/karma-js-testing-what-why-and-how-to-get-going-right-now/>

<https://learn.openwaterfoundation.org/owf-learn-angular/tour-of-heroes-testing/>

<https://github.com/OpenWaterFoundation/owf-learn-angular-test/tree/main/tour-of-heroes-cypress-e2e-testing/tour-of-heroes>

<https://github.com/brandiqa/testing-tour-of-heroes>

<https://lukelindner.medium.com/unit-testing-tour-of-heroes-hero-service-dc0f22ebf40d>

<https://github.com/dwyl/learn-istanbul>

Voir : <https://github.com/ngxs/store/find/master>

Tips sur le testing :

Jasmine is a behavior-driven development framework for testing JavaScript code that plays very well with Karma. Similar to Karma, it’s also the recommended testing framework within the Angular documentation as it’s setup for you with the Angular CLI. Jasmine is also dependency free and doesn’t require a DOM.

As far as features go, I love that Jasmine has almost everything I need for testing built into it. The most notable example would be spies. A spy allows us to “spy” on a function and track attributes about it such as whether or not it was called, how many times it was called, and with which arguments it was called. With a framework like Mocha, spies are not built-in and would require pairing it with a separate library like Sinon.js.

The good news is that the switching costs between testing frameworks is relatively low with differences in syntax as small as Jasmine’s toEqual() and Mocha’s to.equal().

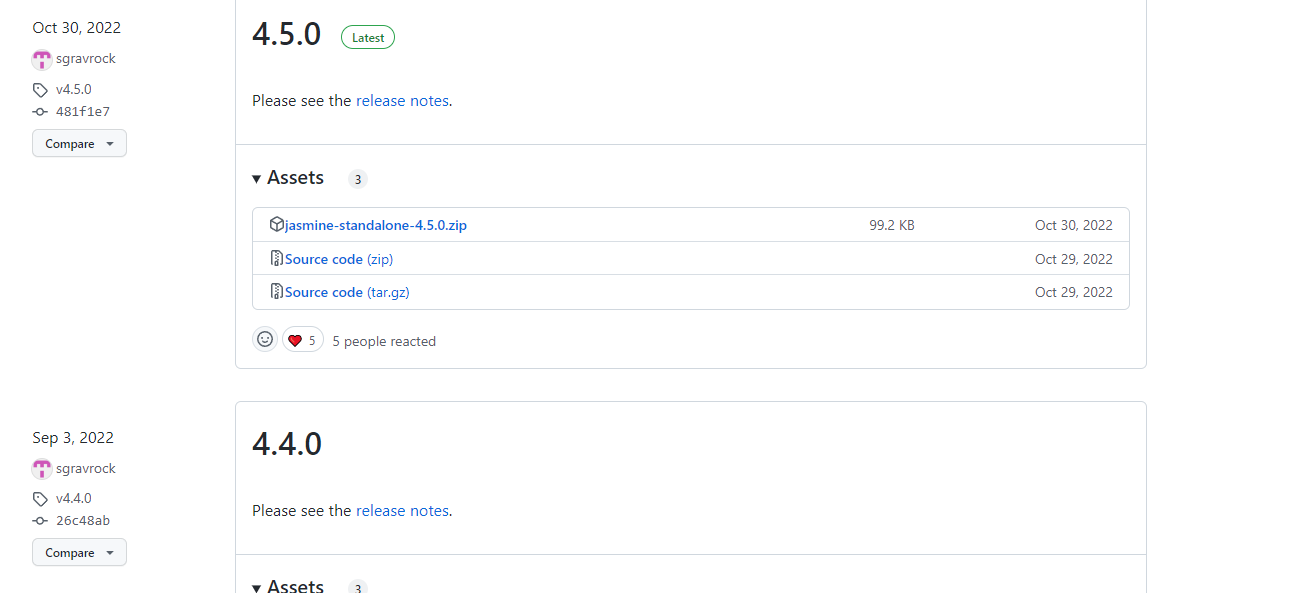
La bonne nouvelle est que les coûts de commutation entre les frameworks de test sont relativement faibles avec des différences de syntaxe aussi petites que celles de Jasmine toEqual()et Mocha to.equal().

Source : <https://www.digitalocean.com/community/tutorials/testing-angular-with-jasmine-and-karma-part-1>

Pour avoir un bon exemple de test unitaire .

<https://www.digitalocean.com/community/tutorials/testing-angular-with-jasmine-and-karma-part-1>

# Jasmine :

Fréquence de mise à jour : 

Communauté

Est-ce que y’a de la doc ? : <https://jasmine.github.io/>

Compréhensible ?

Type License ?

Dépendance ? Si oui pq ?

Est-ce que, on peut faire tous les types de test qu’on veut ?

L’un ou les deux ?

Performance ? Et pourquoi ?

Vérifier si ça fonctionne avec les composants stand alone

Resolver, pipes

Mettre sous forme de différents modules le tuto Tour-Of-Heroes :

Dashboard

Heroes

Vilains