



Document d'architecture

1. Choix et justification de l'architecture de l'application

Pour l'architecture du projet, j'ai opté pour une architecture en couches, elle offre de nombreux avantages:

- **Modularité:** *Chaque couche est conçue pour être autonome, ce qui permet de développer, tester et maintenir chaque couche de manière isolée. Cela facilite la gestion du code et optimise le temps de développement.*
- **Séparation des tâches:** *Chaque couche est dédiée à une tâche spécifique, par exemple dans cette application d'e-commerce:*
 - **Interface utilisateur:** *Cette couche va être visible dans le navigateur de l'utilisateur et avec lequel il va pouvoir interagir.*
 - **Backend:** *Cette couche va être responsable de gérer les requêtes de l'interface utilisateur, traiter les opérations métier et renvoyer les réponses appropriées.*
 - **Service métier:** *Cette couche va être responsable de la logique métier, encapsulant les règles métier et traitant les demandes des contrôleurs.*
 - **API REST:** *Cette couche va être responsable des contrôleurs, cela va permettre à l'interface utilisateur de communiquer avec le backend, de traiter les requêtes HTTP entrantes et invoquer les services métier appropriés.*

- **L'accès aux données:** Cette couche va gérer l'accès aux données persistantes dans l'application, en interagissant avec la base de données et en fournissant des opérations CRUD pour les entités.

Cette séparation permet une meilleure organisation du code, le rendant plus lisible et simplifiant sa maintenance.

- **Evolutivité:** L'architecture en couches permet d'ajouter, de modifier ou de supprimer des fonctionnalités plus facilement, car les changements peuvent être limités à une seule couche sans affecter les autres couches.
- **Sécurité:** Chaque couche peut bénéficier de contrôles d'accès et de mesures de sécurité, ce qui renforce la sécurité de l'application contre de potentielles menaces.

2. Choix et justification des librairies

2.1. Librairies de test

2.1.1. Front-end

Pour le Frontend, j'ai opté pour les librairies open source Jest et Cypress. Jest est reconnue et largement utilisée, elle compte + de 12M d'utilisateurs sur GitHub et + de 43K étoiles. Elle présente de nombreux avantages:

- **Performances:** Jest optimise l'exécution des tests en réorganisant les suites selon leur durée et peut lancer plusieurs suites à la fois, ce qui permet de gagner du temps lors de tests répétitifs ou volumineux.
- **Facilité d'apprentissage:** La syntaxe de Jest est simple/claire ce qui en fait une librairie facile à prendre en main et à maintenir même pour les débutants.
- **Couverture de test:** Jest offre de nombreuses fonctionnalités pour couvrir le code (avec notamment `-coverage`), elle propose également la simulation d'objets (avec les mock).
- **Documentation:** Le site officiel propose une doc

complète et régulièrement mise à jour (env. à chaque release). De nombreuses ressources sont disponibles en ligne, telles que + de 22K questions sur Stackoverflow, une communauté Discord active et de nombreux tutoriels sur YouTube.

J'ai également choisi Cypress car, bien que Jest soit efficace pour tester les fonctionnalités du code, elle ne permet pas de tester les interactions en temps réel depuis le navigateur. L'ajout de Cypress permet donc de tester le code dans son ensemble. Bien que Cypress soit moins populaire que Jest avec seulement 1.3M d'utilisateurs sur GitHub, il a pourtant une meilleure note avec + de 45K étoiles. Cypress présente ces avantages:

- **Fiabilité:** Cypress exécute ses tests directement dans le navigateur en temps réel, ce qui permet de simuler le comportement des utilisateurs de tester de manière plus fidèle les interactions utilisateurs.
- **Facilité d'apprentissage:** Grâce à son interface intuitive, Cypress simplifie le processus de débogage, permettant de visualiser chaque étape.
- **Documentation:** Le site officiel de Cypress propose une doc approfondie, comprenant une FAQ détaillée, des exemples d'applications et une chaîne YouTube officielle. La documentation est mise à jour très régulièrement, et de nombreuses ressources sont disponibles en ligne, telles que + de 9K question sur Stackoverflow et une communauté Discord active.

2.1.2. Back-end

Pour le Backend, j'ai opté pour la librairie open source JUnit 5. Avec + de 400K utilisateurs sur GitHub et + de 6K étoiles, JUnit 5 est une référence dans le domaine du test unitaire en Java. Elle présente de nombreux avantages:

- **Rapidité d'exécution:** JUnit 5 s'efforce à améliorer les temps d'exécution des tests, ce qui permet de réduire le temps pour exécuter les suites de tests.

- **Facilité d'apprentissage:** JUnit 5 est conçu pour simplifier le processus d'écriture des tests, avec une syntaxe claire/simple et des fonctionnalités intuitives. Les tests de JUnit 5 sont également faciles à maintenir.
- **Documentation:** Le site officiel de JUnit 5 propose une doc complète et régulièrement mise à jour (env. chaque release). D'autres ressources sont également disponibles en ligne, telles que des tutoriels sur YouTube, il y a également + de 3K questions sur Stackoverflow.

Pour accélérer le processus de développement des tests et les rendre plus compréhensible, je préconise d'adopter des scénarios BDD (Behavior Driven Development) pour chaque test. Cette approche permettra de réduire les erreurs lors de l'écriture des tests. Les scénarios BDD se composent généralement de 3 parties: Given (Étant donné), When (Lorsque), Then (Alors). En suivant cette structure, les tests deviennent plus faciles à comprendre et à maintenir.

2.2. Librairie de composants visuels

Pour les composants, j'ai choisi la librairie open source Material Angular. Avec + de 809K d'utilisateurs sur GitHub et + de 24.1K d'étoiles de notations. C'est une librairie largement utilisée dans le développement d'interfaces Angular. Voici ces nombreux avantages:

- **Large gamme de composants:** Angular Material propose une large gamme de composants prêt à l'emploi.
- **UX:** Elle assure une expérience utilisateur familière et intuitive pour les utilisateurs.
- **UI:** Les composants ont une apparence professionnelle et moderne.
- **Normes et accessibilité:** Material Angular se conforme aux normes WCAG et RGAA, en mettant l'accent sur l'accessibilité. Ses composants conçus pour être utilisables par tous (notamment les personnes limitées physiquement ou cognitivement) grâce à des fonctionnalités telles que le

support de lecteurs d'écran, gestion de la navigation au clavier, contraste des couleurs approprié, etc...

- **Compatibilité navigateurs:** Material Angular est testé pour assurer une compatibilité avec les principaux navigateurs (Chrome, Firefox, Safari et Edge).
- **Performances:** La bibliothèque est conçue pour être légère et réactive, tout en minimisant les ralentissements et les temps de chargement de la page pour offrir une utilisation fluide et agréable même sur des appareils moins performants.

3. Choix et justification des paradigmes de programmation

3.1. Front-end

Pour le frontend, j'ai choisi la programmation réactive. Ce paradigme est axé sur la gestion des flux de données asynchrones et des événements changeant dans l'application. Elle offre de nombreux avantages:

- **Performances:** La programmation réactive est conçue pour optimiser les performances. Elle réduit les appels réseaux inutiles en réagissant uniquement aux changements de données pertinents, ce qui minimise les mises à jour de l'interface et améliore la réactivité de l'application.
- **Fiabilité:** La programmation réactive est conçue pour rester disponible même en cas de panne.
- **Facilité d'apprentissage:** La programmation réactive facilite la composition des fonctionnalités complexes. Avec des concepts tels que les observables et les opérateurs, le code devient plus facile à comprendre et à prédire. Ce qui facilite la compréhension et la maintenance du code.

Avec la programmation réactive, je préconise d'utiliser la librairie open source RxJS. Elle simplifiera le développement du code, notamment grâce à l'utilisation des observables. Cela permettra d'avoir de meilleures performances, une meilleure modularité et une réduction des bugs. Cependant, il sera obligatoire d'inclure une copie complète de la licence Apache-2.0 dans un fichier LICENSE ou README du projet. Cela garantira

que le projet respecte les termes de la licence et que vous puissiez l'utiliser en toute conformité.

3.2. Back-end

Pour le backend, j'ai opté pour la programmation orientée objet, étant donnée que c'est le paradigme principal en Java. La POO offre une structure de code bien organisée, notamment avec l'utilisation de DTO, d'entités, de modèles, de répertoires, de services, etc. En plus de cette organisation, la POO présente plusieurs avantages:

- **Facilité d'apprentissage:** La POO favorise la modélisation des objets du monde réel, rendant le code plus facile à comprendre, à développer et à maintenir.
- **Facilité de code:** Elle permet de réutiliser le code grâce à l'héritage et le polymorphisme, la POO permet également de diviser le code avec des classes, des objets, etc. Ce qui réduit les risques d'erreurs et facilite la maintenabilité.
- **Performances:** En utilisant une structure de données et un des algorithmes appropriés, la POO peut permettre des performances élevées.