Document d’architecture

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

Si l'on se réfère aux besoins techniques énoncés du client, le choix de l'architecture de l'application se fera sur les besoins 1 et 2, respectivement la disponibilité et la rapidité. Concernant la disponibilité, le site e-commerce se doit d'être, quoi qu'il arrive, disponible en permanence. Pour la rapidité, les clients doivent bénéficier d'un niveau de performance elevé du site. Après analyse des différentes architectures à notre connaissance, celle qui peut potentiellement répondre aux deux besoins est l'architecture orientée services. Elle délègue des responsabilités à des services tierces, notamment une pour procéder à des transactions de manière sécurisée. Le seul inconvénient est que le site principal peut être la cible d'attaques externes, la condition pour que l'architecture orientée services soit une bonne option serait de protéger le site. On ne constate pas véritablement d'inconvénients au niveau de la disponibilité et de la performance, ce sera la raison pour laquelle l'architecture orientée service sera choisie pour ce projet de site e-commerce.

2. Choix et justification des librairies

2.1. Librairies de test

2.1.1. Front-end

Jest, une librairie de tests compatibles avec plusieurs frameworks Javascript, dont Angular. Utilisé par des millions de développeurs, Jest est simple d'utilisation, sa documentation en ligne est complète et accessible à tous. Les tests unitaires sont simplifiées grâce à des fonctionnalités déjà existantes (exemple: toBe(), toBeEqual(), toHaveProperty(), ...)

Cypress, une librairie de test E2E compatible avec Angular. Comme Jest, Cypress est simple d'utilisation, sa documentation est également accessible à tous. Les tests E2E sont simple à rédiger et leurs rendus sont visibles dans un navigateur, toutes les instructions du test y sont détaillés.

2.1.2. Back-end

JUnit, la librairie de tests la plus utilisée pour Java, comme pour Jest, Junit dispose d'une documentation officielle en ligne complète. Les différentes catégories de tests sont les tests unitaires, les tests d'acceptation, les tests d'intégration, les tests de sécurité et les tests de robustesse.

2.2. Librairie de composants visuels

PrimeNG, la librairie UI de composants visuels d'Angular la plus complète, plus de 90

composants utilisables, templates utilisables. Il dispose de quelques composants qui peuvent afficher des articles e-commerce de manière esthétique et simple, nous pouvons le constater en consultant la documentation. On peut citer les composants suivants : Table, DataView, Card et bien d'autres répondant à des besoins divers. PrimeNG est utilisé par des grandes marques, voir site officiel.

Bootstrap, la librairie CSS la plus populaire, elle inclut la fonctionnalité responsive, qui nous sera d'une grande utilité pour rendre l'application accessible sur tous les formats d'écrans (mobile, tablette, ordinateur).

3. Choix et justification des paradigmes de programmation

3.1. Front-end

Programmation orientée objet, Angular fonctionne avec des composants sous forme de classes. Ces classes en implémente d'autres nécessaires au fonctionnement d'un projet Angular, on peut citer la classe OnInit qui requiert la déclaration de la méthode NgOnInit() pour opérer lors de la mise en place du composant. N'oublions pas la classe OnDestroy qui requiert la déclaration de la méthode NgOnDestroy() pour opérer lors de la destruction du composant.

Programmation reactive, Angular peut fonctionner sur le principe d'observateur. On définit des observables qui vont émettre des événements interceptés par des observateurs. La gestion de données s'opère dans le temps de manière asynchrone, d'ou l'avantage de la programmation reactive, la librairie RxJS est généralement la plus utilisée dans Angular.

Programmation fonctionnelle, un paradigme de programmation qui repose sur le principe de composition de fonction et de ses appels, utile lors de développement de méthodes dans des composants Angular.

3.2. Back-end

Programmation orientée objet, le paradigme généralement utilisé en Java. Comme pour Angular, les classes sont utilisés pour simplifier la programmation. L'avantage de la programmation orientée objet est la facilité et la simplicité à executer un projet Java, les developpeurs disposent d'une architecture claire et robuste. Le principe DRY "Don't repeat yourself" consiste à eviter la répétition du code afin de faciliter la maintenance et d'évoluer sans risquer d'eventuels confilts.

Programmation fonctionnelle, pour la même raison citée pour le Front-end, elle peut être utilisée pour développer des méthodes au sein des classes Java.