Document d’architecture

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

L’architecture sera séparé en deux applications, une application back-end (le serveur) et une application front-end (le client).

Le back-end sera constitués de différentes couches :
- Des composants d’accès aux données, en relation avec la base de donnée.
- Des services, qui appliqueront la logique metier sur les différents objets avec lesquels on travaillera
- Des controllers, qui exposeront la donnée au front-end a travers des endpoint.

Dans le front-end, les services feront les appels pour récuperer la donnée et la transmetteront aux composant à travers des observables.

Avoir une architecture avec le back-end et le front-end séparé permet de développer indépendamment les deux et d’être cross-plateform.

La communcation entre le client et le serveur se fera au travers d’appels HTTP. Le back-end exposera une api REST, proposant des données structurées sur des endpoint définis que le front-end viendra requeter.

Choix et justification des librairies

Librairies de test

Frontend

Pour tester le front-end, la librairie de test unitaires **Jest** sera utilisée.

Jest propose une syntaxe simple et concise pour écrire des tests, ce qui facilite leur compréhension et leur maintenance. Cela permet d'écrire des tests clairs et concis sans avoir à se soucier de la complexité syntaxique, ce qui améliore la lisibilité du code de test et accélère le processus de développement.

En tant que l'une des librairies de test les plus populaires pour les applications front-end JavaScript, Jest bénéficie d'une vaste communauté donnant accès à une multitude ressources pour résoudre les problèmes rencontrés lors du développement des tests.

Jest s'intègre facilement avec d'autres outils populaires de développement comme Babel, Webpack et ESLint. Cette compatibilité native simplifie l'intégration des tests unitaires dans le workflow existant, ce qui permet de bénéficier des avantages de Jest sans avoir à apporter des modifications majeures à notre infrastructure de développement.

Pour completer la strategie de test dans la partie front-end, **Cypress** sera utilisé. Cypress permet de simuler des interactions réelles de l'utilisateur avec l'application. Cela permet de tester l'ensemble du parcours utilisateur en validant la bonne intégration du client avec le serveur.

Cypress propose une interface graphique interactive facilitant l'identification et la résolution des problèmes.
Également, et tout comme Jest, il pourra s'intégrer dans des pipeline de développement et d'intégration continue afin de valider la qualité des livrables à chaque lot.

Backend

Pour tester le back-end, **JUnit** est retenue comme librairie de test unitaires. C’est un standard en termes de tests dans l’écosystème Java et ce framework est très largement utilisé.
JUnit peut ếtre étendu avec des framework de mocking comme Mockito, permettant de mocker des composant et services pour isoler la logique que l’on souhaite tester.

JUnit s'intègre facilement dans les outils de build et d'intégration continue tels que Maven et Jenkins. Cela permet d'automatiser l'exécution des tests et d'obtenir des rapports sur les résultats, ce qui facilite la validation des déploiements au fil de l’eau

En choisissant Junit pour tester le backend, nous optons pour une solution fiable et très documentée par sa large communauté.

Cela en fait un choix idéal pour valider le comportement du code back-end.

Librairie de composants visuels

Différents frameworks de composants visuel on été évalués.

Angular Material
Ce framework implémentant le material design de Google est très largement répandu et utilisé. Une grande partie des applications que nous utilisons quotidiennement présentent des composants de librairies implémentant ce design system. Cela permettera d'avoir très rapidement des composants de grande qualité, facile à rendre responsive et à travailler, réduisant donc fortement les coûts de production.

NG-Bootstrap
Cette librairie est construite au dessus de bootstrap CSS, elle permet d'utiliser des composant bootstrap comme des directives angular. Elle permet de créer rapidement des application web et mobile responsive, avec une syntaxe bien connue des développeurs.

PRIME NG

Prime NG fournit des composants qui s'intègrent dans les applications angular en utilisant ses directives et ses hooks. Tout comme Angular Material elle permet donc d'avoir des composants bien pensés et facile à rendre responsive.

PrimeNG propose d'excellentes fonctionnalités d'internationalisation, en définissant des fichiers contenant des valeurs par message et par langue, on proposera des éléments dans une langue définie par la variable locale,

Prime NG est donc retenue comme librairie de composants visuels puisque le site de e-commerce que nous développons est pour Knesh, un célèbre magasin international

Choix et justification des paradigmes de programmation

Front-end

Angular requête et expose la donnée en utilisant RX.js, une librairie orientée événements.

Les services proposent des observables auxquels les composant vont s'abonner pour recevoir la donnée sous forme de flux d'évenements.

On utilisera des méthodes qui pourront itérer sur ce flux d'évenement comme sur une collection.

Le choix de paradigme de programmation est donc la programmation réactive.

Back-end

Pour le back-end, on utilisera des classes pour définir les entités que l'on va manipuler, mais également pour définir les repository, les services, les controllers.

Au lancement de l'application, les classes contenant la logique et définissant la configuration seront instanciées par une chaîne de dépendances.

Lors d'appels sur l'application (création d'un utilisateur par exemple) un objet correspondant à la classe définissant la structure de l'entité sera instancié pour accueillir les propriétés effectives de l'objet, c'est cet objet qui sera manipulé lors de traitement puis sauvegardé en base au travers des méthodes d'un ORM par exemple.

Le choix de paradigme de programmation pour les back-end est donc la programmation orientée objet.