Choix technologiques et justification:

Front :

Du côté front-end, nous avons opté pour du React en raison de sa proximité logique avec l'utilisation de javascript, React peut s'apprendre rapidement s'il on possède les fondamentaux en js,css et html. La programmation par composants est également intéressante car ceux-ci sont interchangeables et combinables ce qui limite la duplication de code et facilite le développement en équipe. React est également plus souple que les autres frameworks js car il s'agit d'une librairie. De plus, nous évitons les problèmes de rétrocompatibilité lorsqu'une nouvelle version est déployée contrairement à Vue.js ou Angular.js.

Back :

En développement back-end nous avons choisi d'utiliser node js, le javascript étant un langage pratique pour concevoir l'intégralité d'une application à la fois en arrière-plan et graphiquement. Cette souplesse d'utilisation permet de corriger d'éventuels problèmes techniques rapidement. Node-js est utilisé par une très grande communauté de développeurs ce qui facilite la résolution de problèmes et permet la programmation asynchrone facilement qui est très orientée développement web.

Afin d'accélérer le développement, nous avons choisis le framework express.js. C'est un framework léger, rapide d'utilisation et robuste. Les documentations sont nombreuses et le framework est simple à prendre en main. Son approche minimaliste et son évolutivité en font un excellent choix dans le contexte d'un développement d'un site d'e-commerce standard.

Déploiement :

Concernant le déploiement, nous répartirons les différents services qui constituent l'application en conteneurs docker. La technologie docker est devenue un incontournable dans les architectures microservices même s'il existe des alternatives (Podman ou techniques de virtualisation comme virtualbox pour contenir des applications sur des machines virtuelles). L'avantage face aux machines virtuelles est que nous n'utilisons qu'un seul système d'exploitation (celui du système hôte) ce qui est beaucoup moins contraignant en termes de ressources et d'entretien. Docker nous permet d'avoir des conteneurs linux avec ses principales fonctionnalités que nous connaissons bien dans l'équipe. Nous pouvons gérer proprement nos instances et conserver une étanchéité entre les services pour qu'ils s'exécutent de manière indépendante les uns des autres. Le système de gestion des images est également très souple notamment concernant la gestion des dépendances et l'automatisation du déploiement des applications au sein d'un environnement de micro services.

Nous utiliserons Kubernetes ou Nomad pour orchestrer le déploiement des clusters de conteneurs docker. Il s'agit d'un outil très pratique pour une architecture micro service car il permet (à l'aide de son interface graphique notamment) d'éliminer de nombreux processus manuels associés au déploiement,d'améliorer la scalabilité des applications conteneurisées et de gérer leur intégrité au fil du temps (mises à jours des librairies et dépendances etc.), un autre avantage est qu'il nous permet de définir notre infrastructure via des règles (au format hcl ou yaml) pour automatiser le déploiement. Kubernetes et Nomad proposent également un outil d'équilibrage de charge sur les "pods" (tri des conteneurs) pour s'adapter au trafic réseau entrant et éviter ainsi une surcharge réseau sur un conteneur en particulier (problème potentiellement critique en situation réelle dans un environnement de production). L'orchestrateur permet également de répliquer selon les besoins en termes de trafic réseau des instances pour éviter tout crash de microservice ce qui rend l'application plus résiliente.

Database :

Un e-shop ne requiert pas beaucoup de mises à jour de produits ou de changements d'un point de vue scalabilité, et nous souhaitons conserver facilement la trace de chaque transaction effectuée par les utilisateurs pour des raisons juridiques et de comptabilité évidentes. Nous utiliserons une base SQL pour la consistance et la structure des données dans ce type de base. Nous avons choisis d'utiliser Postgres (notre architecture microservice nous octroi toutefois la liberté de modifier ce choix selon les services) car celle-ci est particulièrement robuste sur des applications qui ont beaucoup d'utilisateurs en simultané, performante sur les sous-requêtes, propose des outils d'indexation pour améliorer les performances de lecture de la base, de scalabilité (synchronisation et réplication des bases, outils de clusterControl), des outils de gestion des transactions (ensemble de requêtes) pour augmenter la résilience de l'application ainsi que des outils de restauration et de sauvegarde des bases.

Gestion de logs:

Pour centraliser les logs nous prévoyons d'utiliser logstash pour collecter, analyser et stocker les logs de chaque microservice.