Document d’architecture

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

Pour le site e-commerce de Knesh, nous recommandons une architecture en couches reposant sur un modèle client serveur moderne, articulée autour d’un front-end Angular, d’une API REST Java / Spring Boot et d’une base de données SQL. Cette structure, désormais un standard éprouvé dans l’e-commerce, sépare clairement l’interface, la logique métier et les données. Elle répond directement aux exigences exprimées : disponibilité élevée, rapidité d’exécution, réduction des coûts et évolutivité.

Disponibilité et pérennité

Le modèle client serveur isole l’application Angular (client) de l’API Spring Boot (serveur), encapsulant matériel, logiciels et responsabilités. Chaque partie peut ainsi évoluer ou être mise à jour indépendamment sans interrompre l’autre, un atout essentiel pour un site nécessitant une disponibilité continue. La structure en couches du back-end (contrôleurs, services, accès aux données) ajoute une isolation supplémentaire : une modification dans une couche n’impacte pas les autres, renforçant la stabilité générale. Angular et Spring Boot sont des technologies matures, largement adoptées et maintenues à long terme, ce qui garantit une évolution prévisible et une forte résistance à l’obsolescence.

Rapidité et expérience utilisateur

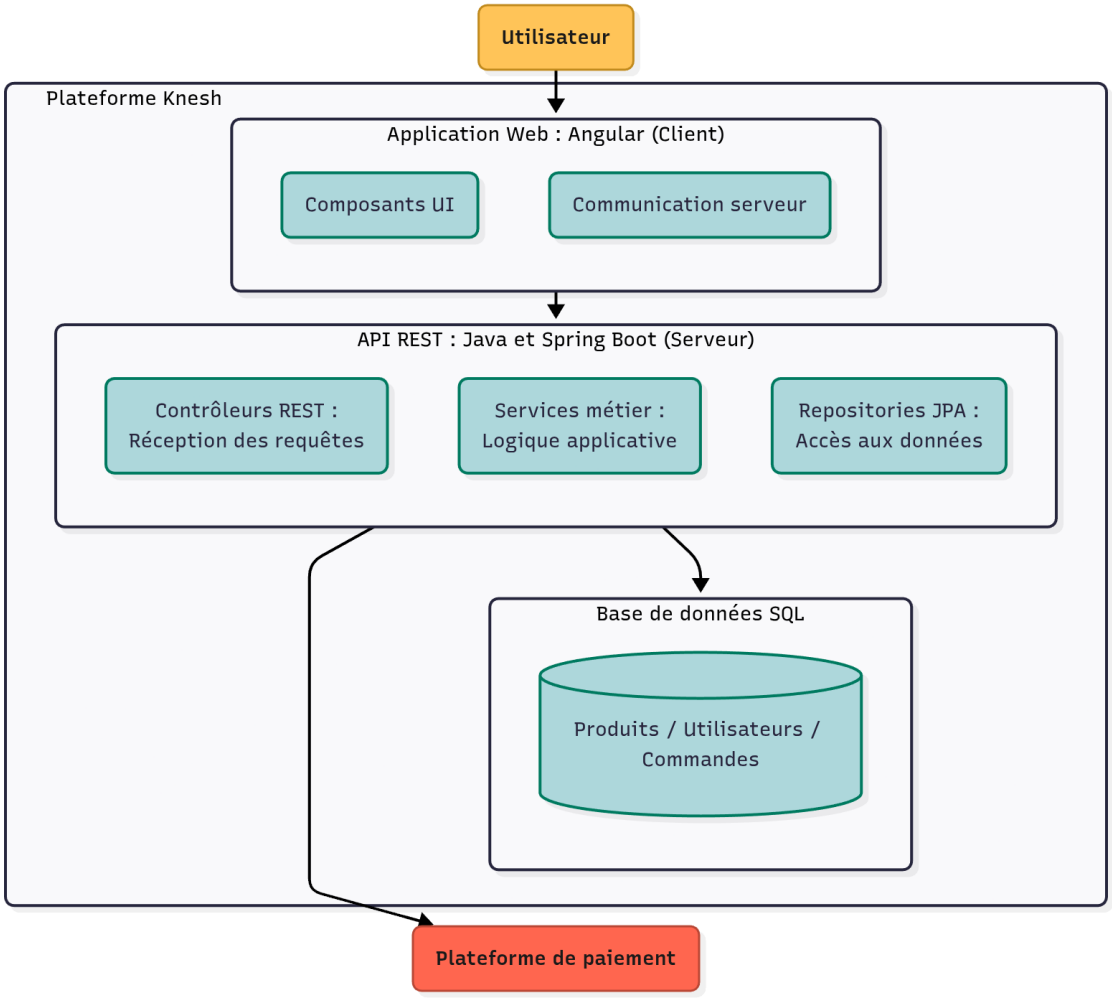
Angular, exécuté dans le navigateur, offre une interface fluide et réactive avec des transitions rapides et une navigation quasi instantanée, sur mobile comme sur desktop. Côté serveur, Spring Boot structure les traitements métier et assure une exécution efficace des opérations clés (catalogue produits, création de commandes, authentification). La séparation nette entre interface, logique métier et données optimise les performances tout au long du parcours utilisateur.

Maîtrise des coûts

L'architecture tire pleinement parti de composants, patterns et outils existants : composants UI standardisés (Angular Material), pipeline HTTP intégré, écosystème Spring couvrant la sécurité, la gestion des erreurs, les contrôleurs REST et l'accès aux données, ainsi qu'une abondance de bibliothèques open-source. En capitalisant sur ces solutions éprouvées, le projet bénéficie d'une forte réutilisabilité, réduit les risques d'erreur et accélère les délais de livraison, ce qui contribue directement à la maîtrise des coûts.

Évolutivité et facilité de maintenance

L'organisation modulaire en couches permet d'ajouter de nouvelles fonctionnalités (moyens de paiement supplémentaires, programmes de fidélité, automatisation marketing, espace client enrichi) sans refactorings importants. Le modèle client serveur offre également une bonne compatibilité entre plateformes et environnements, facilitant les évolutions futures. Enfin, Angular et Spring Boot disposent d'une documentation abondante et de communautés très actives, ce qui simplifie l'intégration de nouveaux développeurs et pérennise la solution sur le long terme.



2. Choix et justification des librairies

2.1. Librairies de test

2.1.1. Front-end

Pour les tests unitaires et d'intégration du front-end, nous retenons Vitest, devenu le framework officiel d'Angular à partir de la version 21. Son intégration native avec Vite offre une exécution très rapide, une configuration minimale et une compatibilité durable avec l'écosystème Angular. Vitest remplace efficacement Jasmine et Karma, désormais en fin de cycle, grâce à son support TypeScript moderne et à une stabilité renforcée. Ce choix garantit une solution pérenne et alignée sur les standards recommandés, tout en accélérant les cycles de développement.

Pour les tests end-to-end, nous recommandons Playwright, particulièrement adapté aux parcours critiques d'un site e-commerce. Sa prise en charge de plusieurs navigateurs (Chrome, Firefox, WebKit) assure une couverture représentative des usages réels, et son système d'attente automatique renforce la fiabilité des tests sur des interfaces dynamiques comme celles d'Angular. Les outils intégrés de diagnostic (traces, vidéos, codegen) facilitent la résolution rapide des anomalies et réduisent les coûts de maintenance. Playwright constitue ainsi une solution robuste, performante et bien adaptée aux exigences de qualité du projet.

2.1.2. Back-end

Pour les tests unitaires et d'intégration du back-end, nous retenons JUnit 6, aujourd'hui la référence moderne de l'écosystème Java. Sa structure modulaire, son modèle d'extensions et son intégration fluide avec Spring Boot, Maven et Gradle en font une solution fiable et durable. JUnit 6 permet d'écrire des tests clairs, rapides et faciles à maintenir, tout en assurant une excellente compatibilité avec les outils actuels d'intégration continue.

Pour les tests end-to-end côté serveur, nous privilégions Selenium 4, technologie largement adoptée et parfaitement intégrée aux environnements Java. Bien que Playwright soit retenu pour les tests end-to-end du front-end, son écosystème Java demeure moins mature, notamment en matière d'outillage, d'intégration native et de génération de rapports. Selenium reste ainsi le choix le plus robuste pour l'équipe back-end grâce à son support étendu des navigateurs, son outillage éprouvé, sa grande fiabilité et la richesse de sa documentation. Cette solution assure une automatisation stable des parcours complets, en particulier

pour valider les interactions front–back et les fonctionnalités critiques d'un site e-commerce, tout en répondant aux exigences de pérennité exprimées.

Enfin, lorsque des scénarios lisibles par des équipes non techniques sont nécessaires, Cucumber peut être associé à JUnit et Selenium. Sa syntaxe Gherkin permet de formaliser clairement les comportements métier tout en conservant une automatisation fiable. Ce choix, bien que facultatif, s'intègre naturellement dans l'architecture retenue et facilite la collaboration entre les équipes techniques et fonctionnelles.

2.2. Librairie de composants visuels

Pour les composants visuels du front-end, nous recommandons Angular Material, la bibliothèque officielle de l'équipe Angular. Elle fournit un ensemble cohérent de composants modernes et accessibles, maintenus durablement et alignés sur les standards Material Design. Son intégration native avec le CDK assure une compatibilité continue avec les futures versions du framework, garantissant une interface stable dans le temps. Angular Material offre également une base solide pour accélérer le développement grâce à des composants prêts à l'emploi et à une documentation abondante, ce qui limite les coûts et facilite l'arrivée de nouveaux développeurs. Bien que d'autres bibliothèques offrent davantage de flexibilité stylistique, Angular Material constitue le choix le plus fiable et pérenne pour assurer la cohérence visuelle et l'évolutivité de l'application.

3. Choix et justification des paradigmes de programmation

3.1. Front-end

Pour la partie front-end, l'application adoptera une approche principalement déclarative et réactive, deux paradigmes parfaitement adaptés aux besoins d'un site e-commerce moderne. L'approche déclarative permet d'exprimer l'état attendu de l'interface plutôt que la suite d'opérations nécessaires, ce qui favorise une interface stable, compréhensible et facile à faire évoluer. La réactivité garantit que l'affichage s'adapte automatiquement aux changements de données, assurant une expérience fluide lors de la consultation du catalogue, de la gestion du panier ou du passage de commande. Des principes issus de la programmation fonctionnelle complètent cette base, notamment la préférence pour des transformations prévisibles et un état maîtrisé. L'ensemble simplifie les tests, réduit les risques d'erreurs et facilite la maintenance.

3.2. Back-end

Pour la partie back-end, l'application adoptera principalement la programmation orientée objet, le paradigme naturel de Java et de Spring Boot. Cette approche structure la logique métier autour d’éléments clairement définis (contrôleurs, services, entités), ce qui facilite la compréhension du système et son évolution à long terme. Des principes issus de la programmation fonctionnelle complètent cette base lorsqu’ils renforcent la fiabilité des traitements : certaines règles métier (calcul de prix, validation du panier, application de réductions) peuvent être exprimées sous forme de transformations pures, plus prévisibles et plus simples à tester, tandis que l’usage ponctuel de données immuables limite les erreurs liées aux modifications involontaires.

Enfin, le back-end s’appuiera sur des mécanismes réactifs intégrés à Spring pour gérer efficacement les communications externes, comme les appels à la plateforme de paiement ou l’envoi de notifications. Ces traitements asynchrones évitent de bloquer le reste du système tout en conservant une architecture simple et maintenable, assurant ainsi disponibilité et réactivité pour les utilisateurs.