Document d’architecture

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

Considérant la nature du projet qui est le développement d’un site de e-commerce d’un magasin de stature internationale, l’architecture de l’application sera la suivante :

L’application sera découpée en trois couches :
- une couche de présentation
- une couche métier
- une couche de données

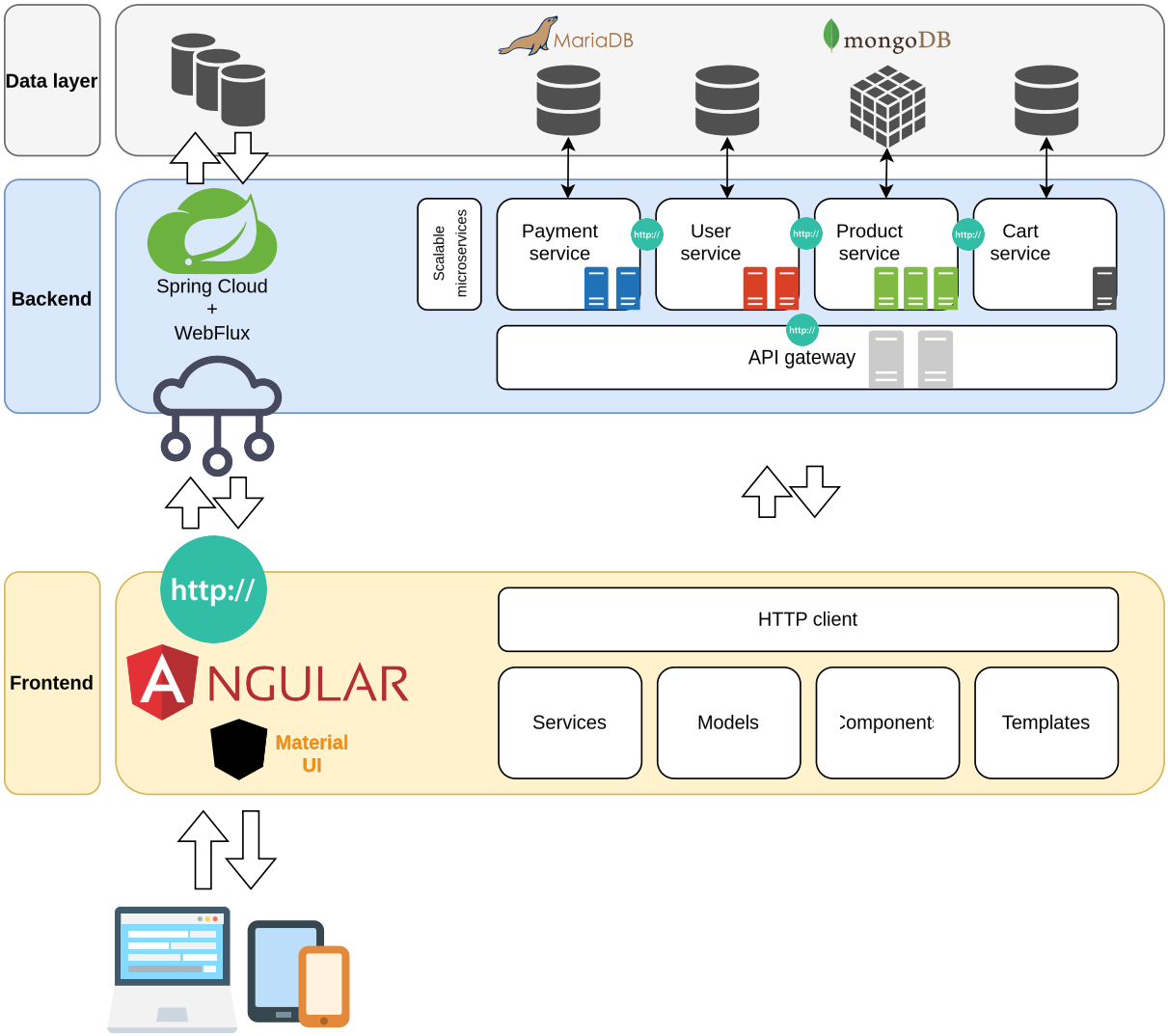
Ce choix permettra de découpler les différents aspects de l’application et de permettre dans une certaine mesure des développements parallèles, de la maintenance par composant individuel et de la mise à l’échelle sélective, en impactant le moins possible le reste de l’application.

La couche métier sera elle-même divisée en plusieurs microservices qui pourront être mises à l’échelle individuellement et selon les besoins avec des technologies comme Docker et Kubernetes. Ces services seront accessibles via un portail qui exposera une api publique, en arrière plan les services peuvent communiquer entre eux pour agréger les données.

La couche de données comportera les bases de données relatives à chaque service, le type de base de données utilisé variera en fonction de la nature et le volume des données traitées. Ainsi pour une base de données de produits qui comporte potentiellement de très larges volumes et sera soumise à un nombre de requêtes important, nous utiliserons une base de type NoSQL comme MongoDB qui permet l’utilisation de structure dynamiques, offre d’excellentes performances et facilite la mise à l’échelle.

La couche de présentation comportera l’interface utilisateur, typiquement l’interface de e-commerce consultable avec un navigateur web, qui permettra à l’utilisateur de consulter les produits proposés par l’enseigne, de constituer un panier d’achat, d’effectuer des commandes ainsi que toutes les opérations usuelles pour un site de e-commerce.

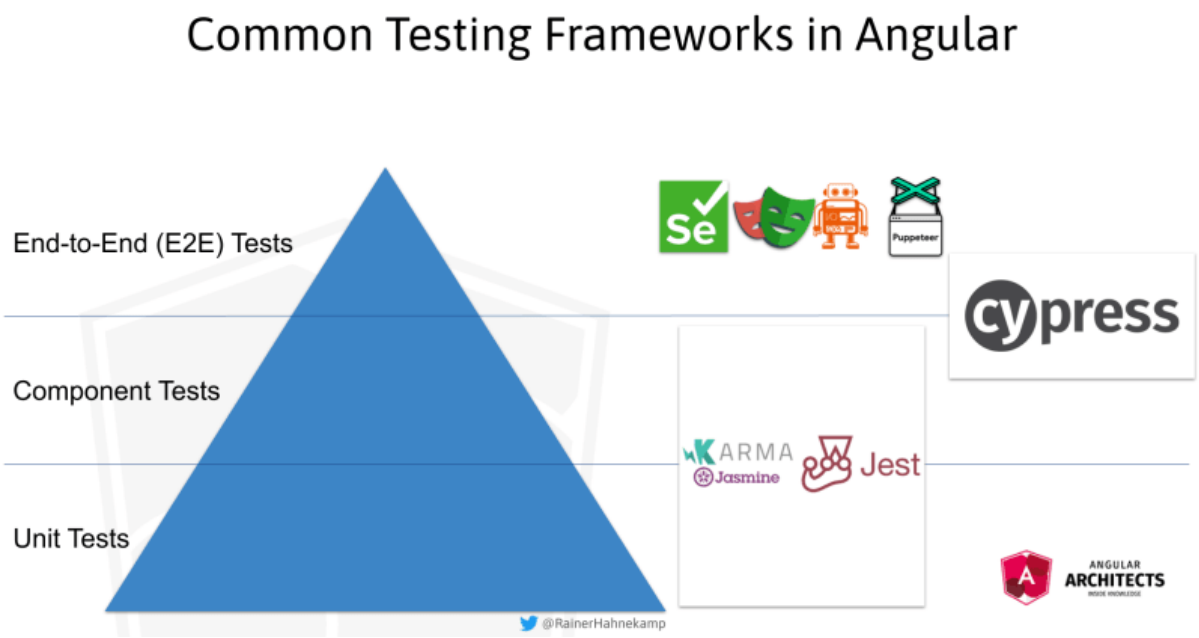
Architecture applicative



2. Choix et justification des librairies

2.1. Librairies de test

Une stratégie de test complète et exhaustive devrait comprendre comme illustrés ci-dessous :
- des tests unitaires nombreux
- des tests de composants, moins nombreux que les tests unitaires
- des tests de bout en bout / UI, moins nombreux car plus lents mais critiques pour valider les scénarios utilisateur complets.



Source : https://dev.to/rainerhahnekamp

2.1.1. Front-end

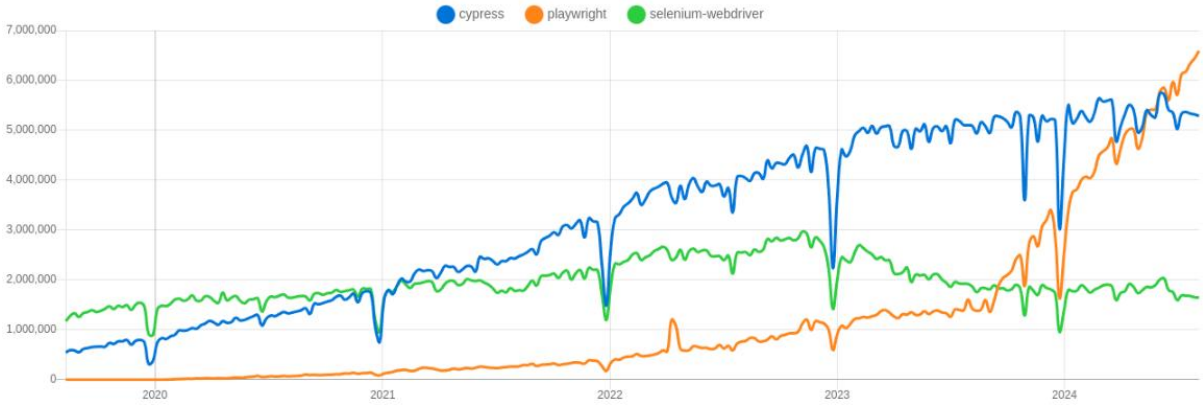
Nous éliminons d'emblée Protractor car c'est un projet en fin de vie depuis décembre 2022.

La vision technique nous propose deux types de bibliothèques de tests.

D'un côté et nous avons des bibliothèques de type de "bout en bout" / "end-to-end" (E2E), qui permet de tester les applications dans un navigateur réel en simulant les interactions utilisateurs afin de valider un fonctionnement complet.
De l'autre nous avons des bibliothèques axé BDD ("test de comportement" / "behavior driven development"), qui peuvent aussi faire du test unitaire.
Nous ferons donc usage des outils nécessaires pour couvrir une stratégie de tests exhaustive en choisissant un framework E2E et un framework de tests unitaires permettant aussi de tester les composants.

E2E :
Puisque Protractor n'est plus d'actualité, nous opposerons Cypress, Playwright et Selenium dans cette catégorie.

Tendance de 3 frameworks de test E2E



Source : https://npmtrends.com/cypress-vs-playwright-vs-selenium-webdriver

Playwright est une option puissante et moderne qui offre une API riche en fonctionnalités, un excellent support multi-navigateurs et des fonctionnalités avancées comme la gestion des contextes multiples et les tests sur mobile. Il est particulièrement bien adapté aux projets nécessitant des tests rapides, fiables, et multi-navigateurs.

Cypress est un outil adapté aux applications modernes mais il peut pêcher question rapidité ; il nécessite des plugins pour accomplir certaines tâches ou bien une license payante.
Selenium est plus complexe à configurer, manque de fonctionnalités comme les rapports détaillés, nécessite également des plugins et peut également s'avérer plus lent à l'exécution.

Selenium fut historiquement l'outil de choix pour les test E2E mais il a accumulé du retard en terme de fonctionnalités et en facilité d'usage.

Comparatif de 3 frameworks de tests E2E

Caractéristique	Cypress	Playwright	Selenium
Année de sortie	2015	2020	2004
Créateur	Cypress.io	Microsoft	Jason Huggins (ThoughtWorks)
Langages supportés	JavaScript, TypeScript	JavaScript, TypeScript, Python, C#, Java	Java, Python, C#, Ruby, JavaScript, PHP
Support multi-navigateurs	Chrome, Edge, Firefox	Chromium (Chrome, Edge), Firefox, WebKit (Safari)	Chrome, Firefox, Safari, Edge, Internet Explorer
Exécution en parallèle	Oui, via le Dashboard Cypress (payant)	Oui, support natif	Oui, via Selenium Grid
Tests de mobile	Limité, nécessite des plugins externes	Oui, support intégré pour les appareils mobiles	Limité, nécessite des outils externes comme Appium
API de contrôle du navigateur	API facile à utiliser, mais limitée à certains navigateurs	API moderne, riche, contrôle avancé du navigateur	API WebDriver, puissante mais plus complexe
Installation et configuration	Simple, configuration minimale	Simple, configuration minimale	Plus complexe, nécessite souvent plus de configuration
Richesse de l'API	Riche mais axée sur les tests UI	Très riche, inclut des fonctionnalités avancées (multi-tabs, mocks réseau, etc.)	Très riche, mais nécessite souvent des configurations supplémentaires
Captures d'écran/vidéos	Oui, intégré	Oui, intégré	Oui, mais nécessite souvent des

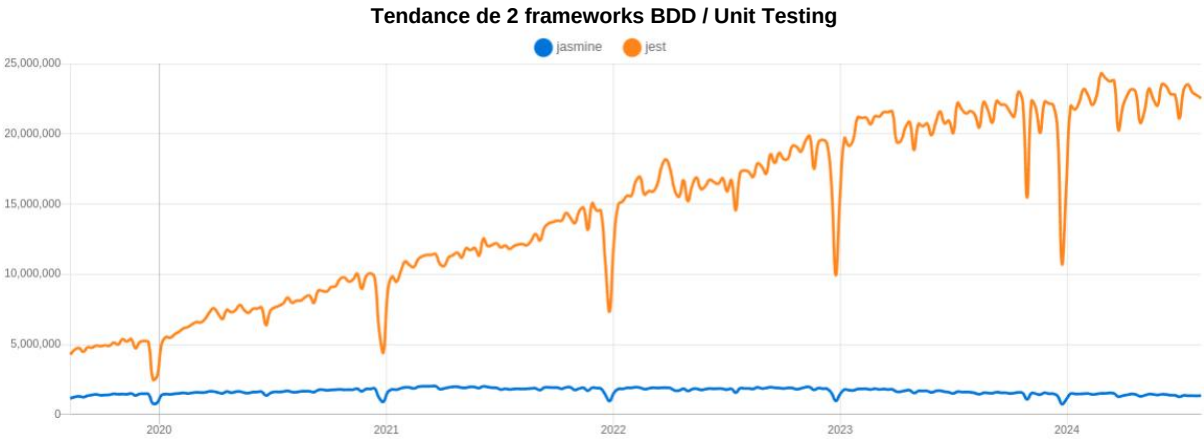
			configurations externes
Attente automatique des éléments	Oui, Cypress attend automatiquement que les éléments soient prêts	Oui, Playwright attend automatiquement que les éléments soient prêts	Non, nécessite des Explicit Waits ou Implicit Waits
Tests de snapshot	Non intégré (nécessite des bibliothèques tierces)	Oui, intégré	Non, nativement supporté uniquement avec des bibliothèques tierces
Rapidité d'exécution	Rapide, mais peut être limité par le support des navigateurs	Très rapide, optimisé pour les tests parallèles	Peut être plus lent, surtout avec des suites de tests volumineuses
Support communautaire	Croissant, documentation officielle solide	Croissant, soutenu par Microsoft, documentation claire	Très large, vaste écosystème et documentation abondante
Cas d'utilisation privilégiés	Tests de bout en bout pour des applications web modernes, surtout pour Chrome/Firefox	Tests de bout en bout pour des applications modernes avec besoins avancés, support multi-navigateurs	Tests sur une large gamme de navigateurs et d'environnements, y compris des environnements anciens ou exotiques
Intégration CI/CD	Oui, bien supportée avec des plugins pour de nombreux systèmes CI	Oui, facile à intégrer avec les systèmes CI modernes	Oui, largement utilisé dans des environnements CI/CD

Verdict : Nous choisissons Playwright pour sa versatilité, sa simplicité de prise en main et ses bonnes performances.

Sources :

https://www.lambdatest.com/blog/cypress-vs-playwright/	Ioan Solderea : Ambassadeur pour Cypress
https://www.checklyhq.com/blog/cypress-vs-selenium-vs-playwright-vs-puppeteer-speed-comparison/	Giovanni Rago : Head of Customer Solutions chez Checkly (solution de monitoring & test de bout en bout d'application web et API)
https://medium.com/@mohsenny/deciding-between-cypress-and-playwright-a-comprehensive-guide-4d883d1be147	Mohsen Nasiri : rédacteur IT (orienté testing / QA) sur medium.com
https://blog.appsignal.com/2024/05/22/cypress-vs-playwright-for-node-a-head-to-head-comparison.html	Antonello Zanini : ingénieur logiciel auteur de nombreux articles (700+)

Tests unitaires & composants :



Source : https://npmtrends.com/jasmine-vs-jest

Jasmine est un framework orienté BDD (Behavior-Driven Development). Développé par Pivotal Labs et sorti en 2010, il bénéficie par son ancienneté d'une large base d'utilisateurs et d'une adoption étendue. Les ressources documentaires sur Jasmine sont nombreuse. Le framework bénéficie d'une excellente intégration avec Angular. Par défaut l'outil Angular CLI pré-configure l'environnement pour utiliser Jasmine + Karma (un test runner) à la création d'un nouveau projet et génère des fichiers de tests à la création de nouveaux composants.

Jest est développé et maintenu par Facebook / Meta ; sorti en source publique en 2014. Il est rapidement devenu populaire et a dépassé Jasmine en nombre de téléchargements fin 2017, cela grâce à son association avec React et l'écosystème de Facebook. La communauté est très active et la documentation abondante. Jest nécessite une configuration minimale car il n'est pas intégré comme Jasmine au CLI d'Angular, mais il se suffit à lui-même pour l'exécution des tests.

Tests sans navigateur Rapidité d'exécution	Jest utilise jsdom pour avoir accès à un dom simulé (ainsi que d'autres API du navigateur) ce qui apporte un gain de performance et permet de se passer d'un véritable navigateur Cela est souhaitable dans certains environnements et facilite l'usage avec les outils d'intégration/développement continue. Pour les tests nécessitant des fonctions spécifique & avancées, nous utilisons une solution E2E comme Playwright.
Exécution "intelligente"	À la demande : lors d'une session de test, ne réexécute que les tests impactés par les changements apportés dans le code.
Puissants outils intégrés	Mocking automatique Rapports de couverture Snapshot testing : capturer l'état d'un composant et le comparer à des exécutions futures
Exécution parallèle	Non fournie par Jasmine

Verdict :

Nous retiendront Jest pour sa rapidité, sa puissante API ainsi que autonomie pour exécuter les tests. De plus, même si Angular n'a pas officiellement annoncé une migration vers Jest comme framework

de test par défaut, il y a une tendance croissante au sein de la communauté vers l'adoption de Jest en raison de ses nombreux avantages en terme de performance, de configuration et de fonctionnalités modernes. Nous utiliserons Jest principalement pour faire du test unitaire, mais aussi pour faire des tests de comportement si besoin afin de faire le lien avec les tests E2E.

Sources :

https://angularexperts.io/blog/total-guide-to-jest-esm-and-angular	Rainer Hahnekamp : Architecte Angular – Développeur chez Google
https://dev.to/this-is-angular/migrate-from-jasmine-to-jest-and-testing-in-angular-286i	Dany Paredes : Développeur médaillé "GDE" (Google Developer Expert) sur Angular en 2022, 2023, 2024.

2.1.2. Back-end

La vision technique nous propose pour le back-end Selenium, Cucumber et JUnit5.
Ce sont 3 framework qui ont un but différent et ne s'opposent pas :

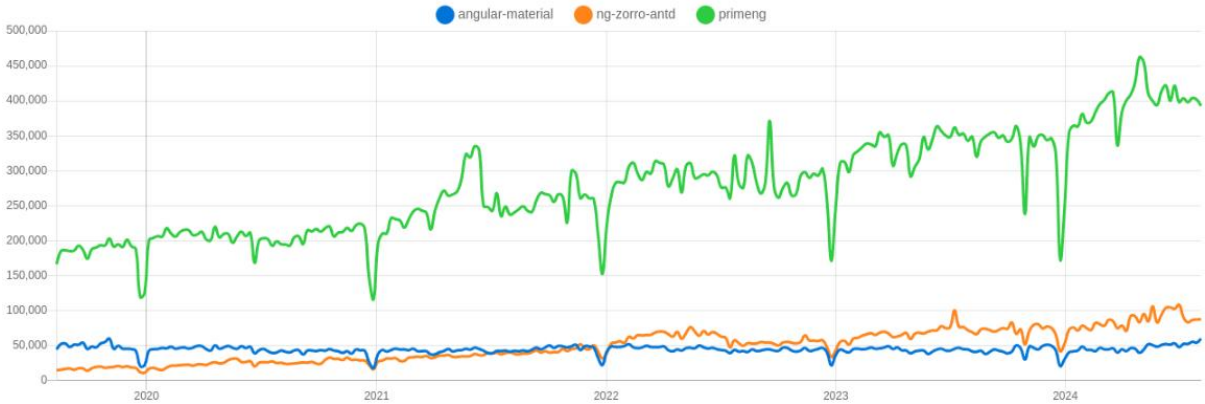
- JUnit5 il est utilisé pour faire des tests unitaires
- Cucumber pour les tests orientés BDD
- Selenium pour les tests E2E

Verdict :
Nous utiliseront Cucumber conjointement avec JUnit5 pour tester les microservices et l'API publique. Des spécifications et scénarios pourront être décrits en langage "naturel" (Gherkin) par les collaborateurs et intégrés avec Cucumber.

2.2. Librairie de composants visuels

3 frameworks sont proposés pour les composants visuels : Material, PrimeNG, NG Zorro.

Tendance de 3 bibliothèques de composants visuels



Source : https://npmtrends.com/angular-material-vs-ng-zorro-antd-vs-primeng

Comparatif de 3 bibliothèques de composants graphiques

Critère	PrimeNG	Angular Material	NG-Zorro
Origine	Développé par PrimeTek	Développé par l'équipe Angular de Google	Basé sur Ant Design, développé par Alibaba
Nombre de	Plus de 80 composants	Moins, mais bien	Plus de 60 composants

composants		optimisés	
Design	Thèmes variés, personnalisables	Matérial Design strict	Ant Design
Thématisation	Très flexible avec de nombreux thèmes disponibles	Limitée aux thèmes Material Design	Personnalisable, mais moins flexible que PrimeNG
Accessibilité	Bon, mais certains composants nécessitent des ajustements	Excellente, fort focus sur l'accessibilité	Correcte, mais dépend de la communauté
Documentation	Très complète, grande communauté	Documentation officielle exhaustive	Documentation correcte, surtout en chinois
Performances	Très bonnes performances	Excellentes performances	Bonnes performances
Courbe d'apprentissage	Modérément complexe en raison de la diversité des composants	Relativement simple grâce à une bonne documentation	Moyenne, dépend de la connaissance d'Ant Design
Communauté & Support	Large communauté, support premium disponible	Large communauté, support officiel	Fort en Asie, croissant ailleurs
Cas d'utilisation idéal	Applications complexes avec besoin de nombreuses fonctionnalités	Applications respectant le Material Design	Applications d'entreprise utilisant Ant Design

Verdict :
Nous choisissons Material pour sa meilleure intégration avec Angular. En effet les développements étant réalisés par l'équipe d'Angular, nous présageons une introduction de bugs moins importante et une plus grande réactivité à les résoudre.
Nous n'avons pas retenu PrimeNG malgré un nombre de composants attrayant car il semble souffrir de bugs récurrents (notamment à chaque nouvelle version) et d'une prise en main plus difficile.

Sources :

https://www.reddit.com/r/Angular2/comments/12dd82e/primeng_vs_angular_material/	Sondage et avis sur Reddit (publié en 2023)
https://www.reddit.com/r/Angular2/comments/1cvn2zp/downsides_of_primeng/	Avis sur Reddit dont le créateur de PrimeNG (publié en 2024)

3. Choix et justification des paradigmes de programmation

3.1. Front-end

Nous utiliseront la programmation réactive avec Angular, qui offre une gestion des opérations asynchrone simplifiée, facilite la gestion des flux de données et les changement d'état (propagation des modification grâce au modèle Observable/Observer), de manière à offrir une expérience utilisateur fluide et dynamique.

Le framework Angular (créé et principalement maintenu par Google) est en développement actif. Il est très bien documenté, comporte une large communauté de développeurs et d'utilisateurs, ce qui augure une certaine pérénité quand à son évolution future (correctifs, nouvelles fonctionnalités), ainsi que l'abondance d'information sur son fonctionnement.

3.2. Back-end

Programmation réactive avec Spring WebFlux (Reactor) pour traiter les flux de données de manière asynchrone et non bloquante, assurer une haute disponibilité et de bonnes performances en optimisant les ressources physiques, faciliter la mise à l'échelle et augmenter la tolérance aux pannes.

Spring est un framework également très populaire et en développement actif. Il fait parti des incontournables en Java. Il bénéficie comme Angular d'une large communauté de développeurs et d'utilisateurs, ce qui lui assure une évolutivité future et une grande disponibilité d'information.

3.3. Front-end et back-end

La "programmation réactive" est particulièrement adapté au web pour les raisons évoquées plus haut. En effet, les applications web modernes sont

L'utilisation d'un même paradigme "programmation réactive" facilite également le travail du développeur amené à développer des composants à la fois pour la couche métier et celle de présentation. Il retrouvera une "philosophie" et une manière de procéder commune.

L'augmentation des performances induit une réduction des coûts à prévoir (à performance égale pour une solution moins optimale) ainsi qu'une meilleure expérience utilisateur.

À noter : la programmation réactive fait usage de la programmation fonctionnelle par l'usage d'expression "lambda", pour définir des comportements après qu'une tâche ait été exécutée ("callback"), manipuler des flux de données par le biais d'opérateurs.

À noter également : L'utilisation massive d'annotations dans un projet Spring constitue une forme de programmation déclarative dans la mesure où elles décrivent généralement un comportement ou un concept (rôle, responsabilité) en laissant le soin au framework de gérer les détails d'implémentation et les mécaniques de fonctionnement internes.

Par exemple, l'annotation @Entity décrit le concept d'entité utilisé par l'ORM (Object Relationnal Mapping) JPA (Java Persistance API), qui sert décrire des objets qui font le lien avec une base de donnée.
De même, @Transactionnal décrit qu'une méthode doit être exécutée dans le cadre d'une transaction sans que le développeur ait à spécifier explicitement (sauf si il le souhaite) comment la transaction doit être gérée.

Citons également l'HTML pour les pages web, qui est un langage déclaratif.

Ainsi, même si nous utilisons des frameworks axés sur un paradigme dominant - dans notre cas la programmation réactive – plusieurs paradigmes sont en fait combinés et utilisés conjointement : orienté objet, impératif, fonctionnel, déclaratif, réactif.