Dossier d'architecture

Diffusion : OpenClassrooms – Joeffrey Hernandez (mentor) – M. X (Evaluateur de soutenance)

Version	***Date***	***Auteur***	***Motif***
1	12/02/2025	Christophe Pierrès	Création du document
2			

Contenu ***Page***

1. Introduction 3

2. Choix et justification de l'architecture technique 4

3. Choix et justification des librairies..... 7

3.1. Librairies de test 7

3.1.1. Font-end 7

3.1.2. Back-end 8

3.2. Librairie de composants visuels 8

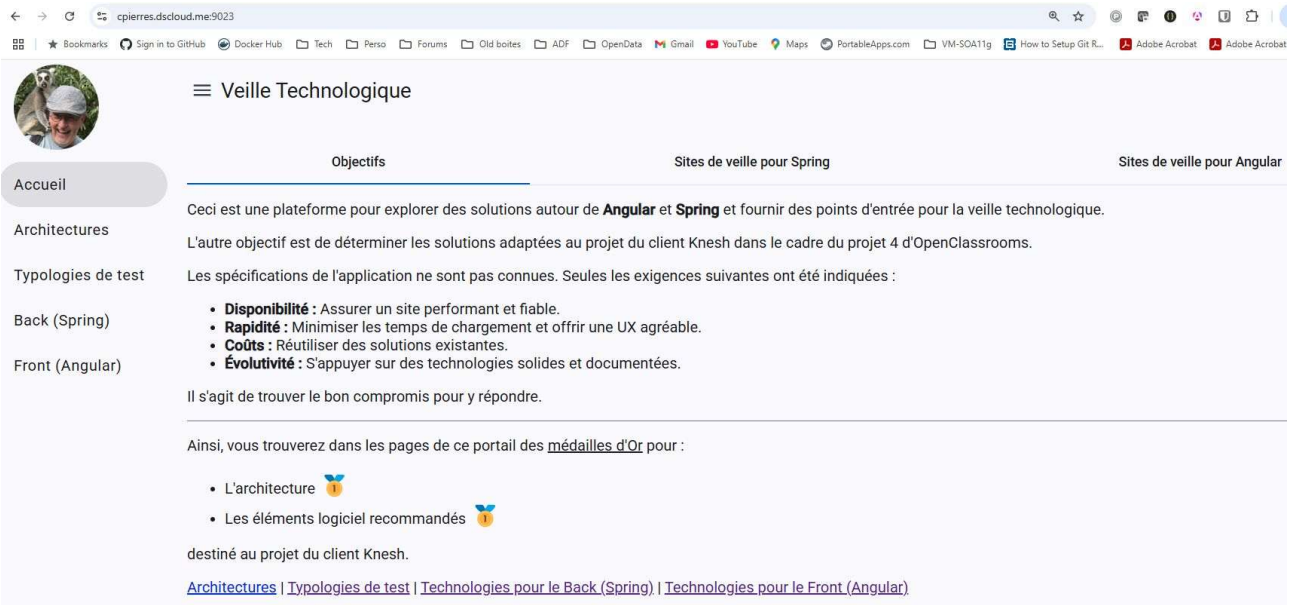
4. Choix et justification des paradigmes de programmation .. 10

1. Introduction

Ce dossier d'architecture répond aux exigences techniques du client Knesh.

Il s'appuie sur notre site de veille technologique : https://cpierres.dscloud.me:9023/

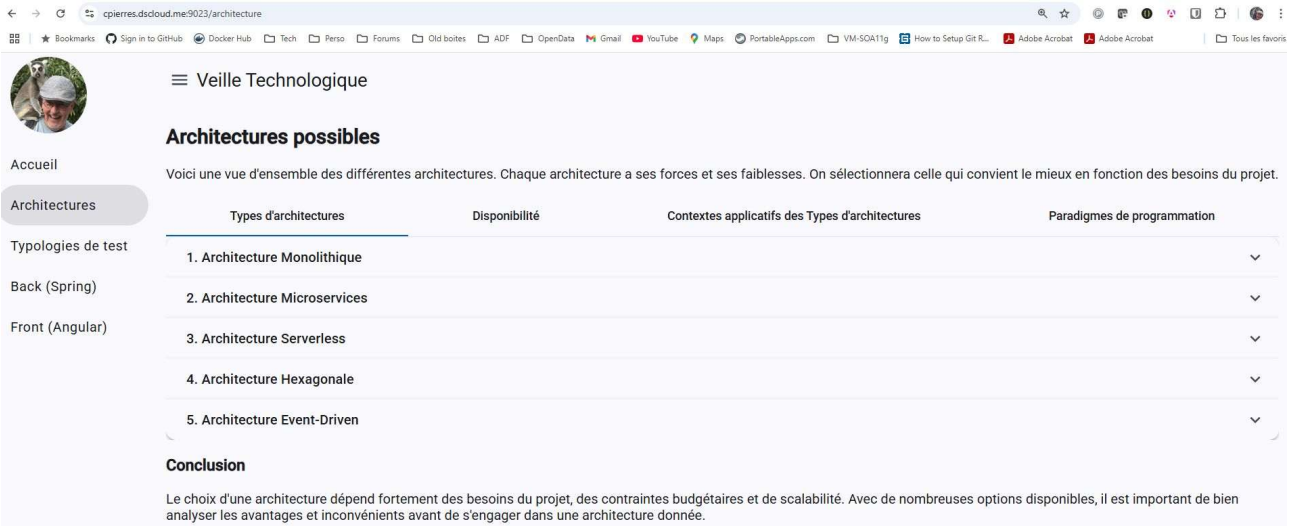
La page d'accueil rappelle les besoins de Knesh:



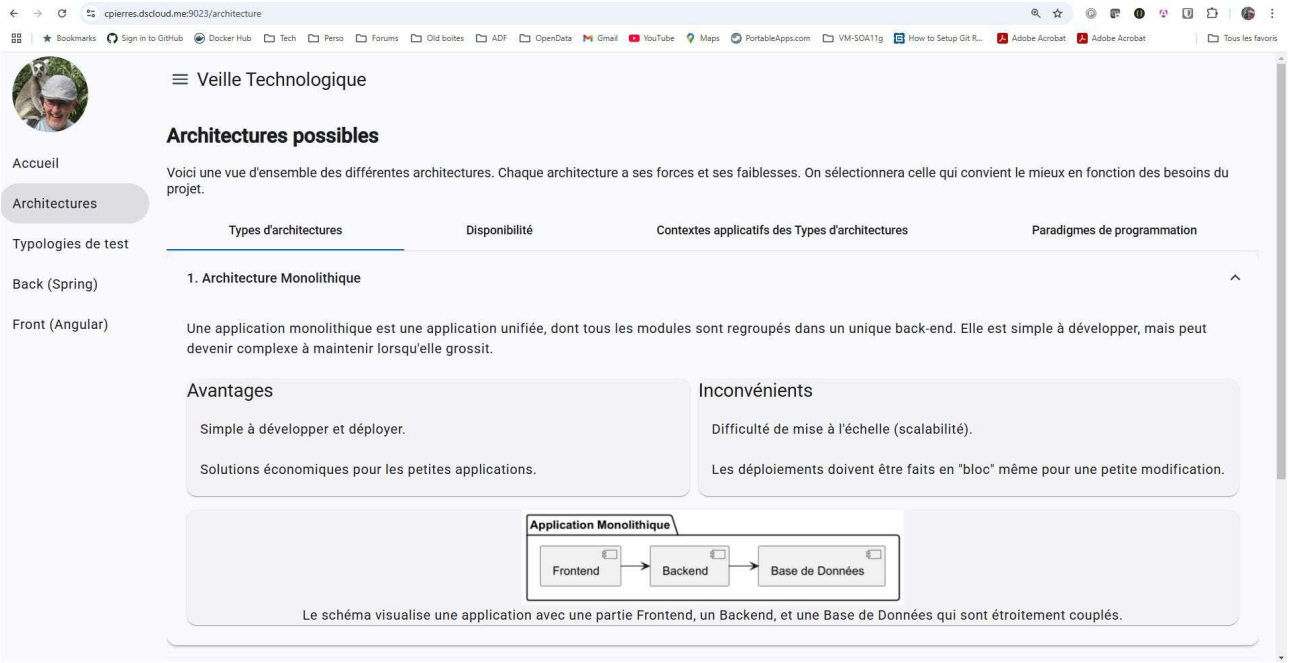
Nous proposons des choix mais notre site pourrait vous permettre d'affiner vos besoins si nécessaire.

2. Choix et justification de l'architecture technique

L'écran suivant décrit les types d'architecture : https://cpierres.dscloud.me:9023/architecture



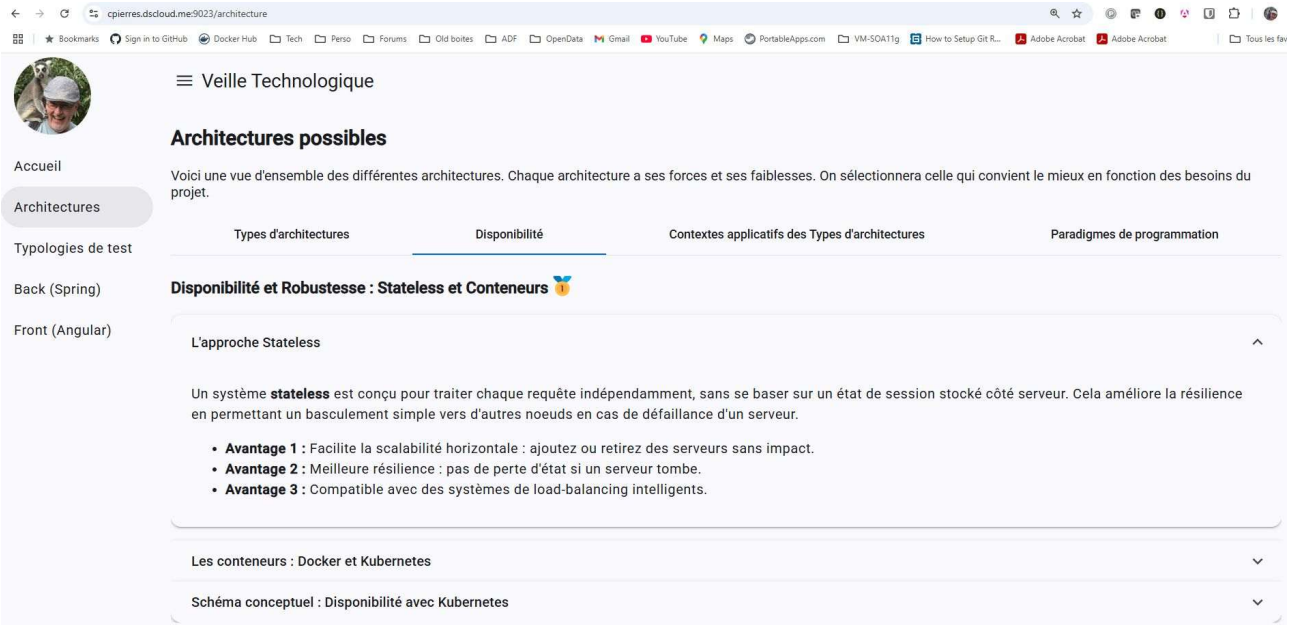
Cliquez sur chaque titre permet d'afficher une description avec les avantage et inconvénients :



Nous vous conseillons de lire les différents types d'architecture afin de vérifier l'adéquation à vos besoins. D'après les exigences exprimées, la notion de **coût** est importante et donc de **développement rapide**. **L'architecture monolithique** est **simple à mettre en œuvre et éprouvée**.

La **disponibilité** est un critère important. Le développement en architecture REST (Representational State Transfer) a l'énorme avantage d'être stateless, ce qui permet d'assurer la robustesse et la disponibilité.

Nous proposons d'adopter la mise en œuvre de Docker et Kubernetes, dont l'architecture est décrite ci-après :



Nous proposons la mise en œuvre de conteneurs et de Kubernetes :

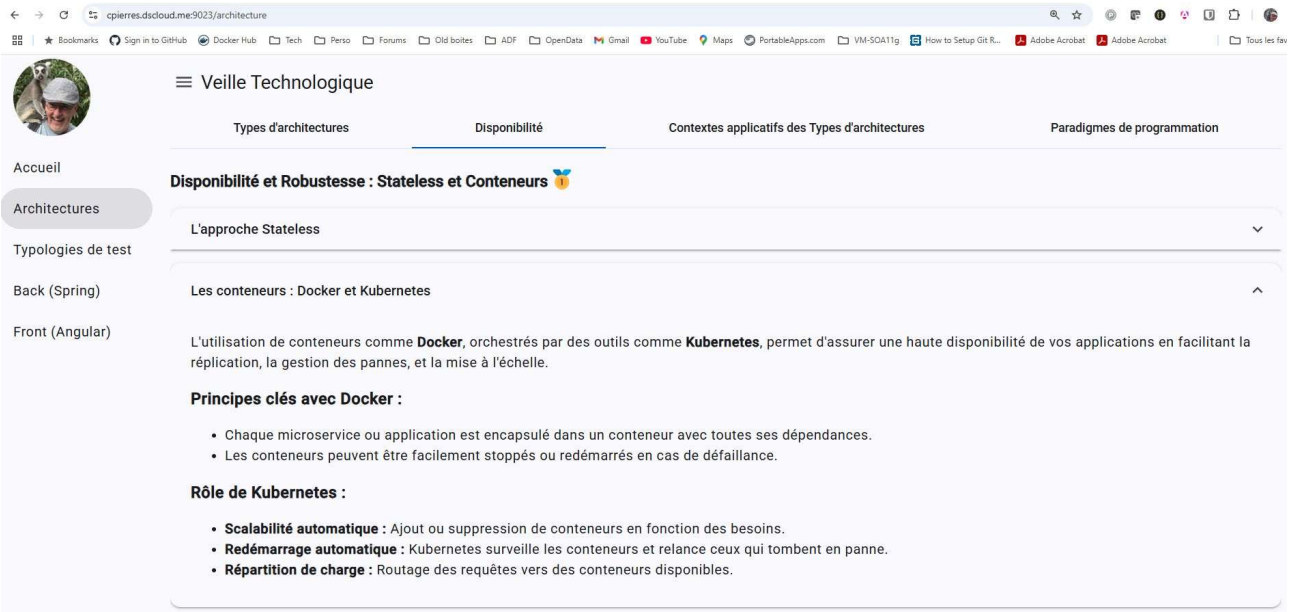
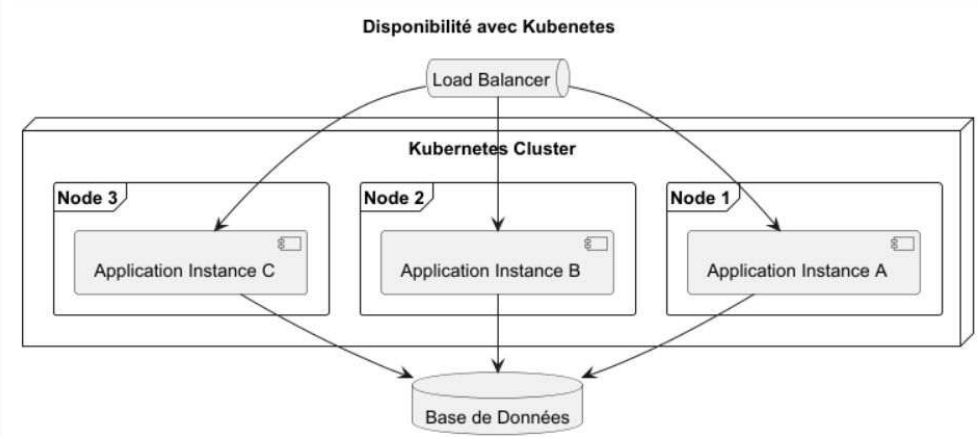
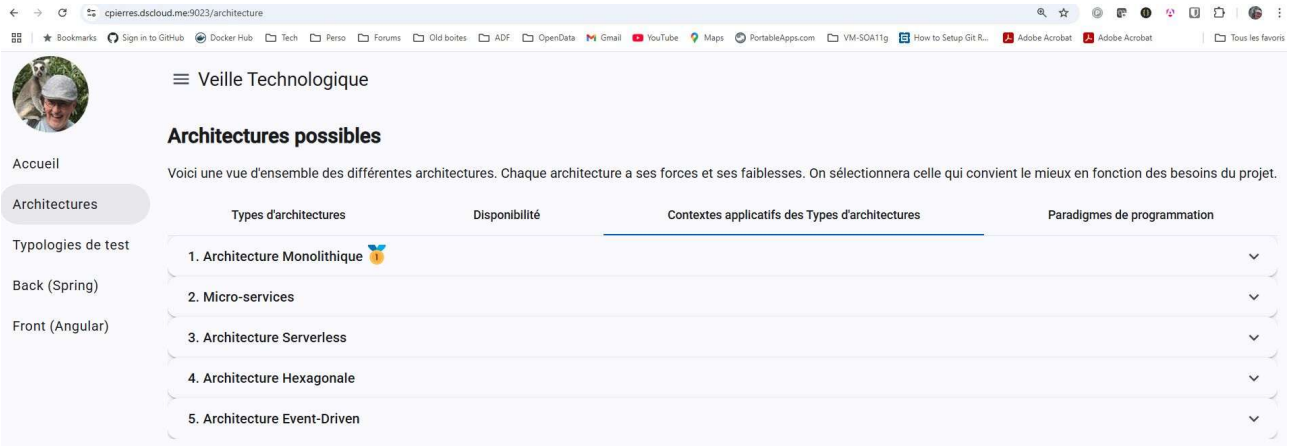


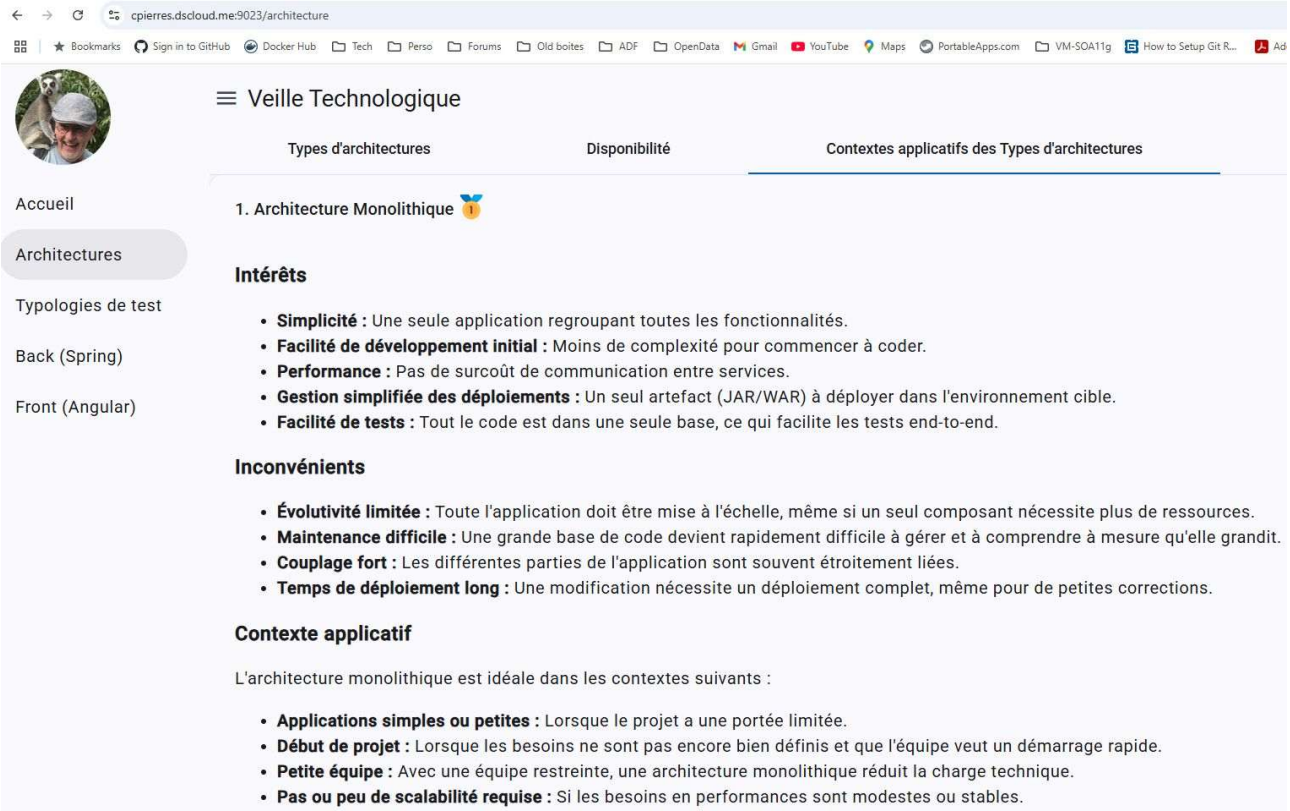
Schéma conceptuel : Disponibilité avec Kubernetes



Pour confirmer que notre proposition est conforme à vos attentes, veuillez consulter les différentes sections de l'onglet **Contextes applicatifs** :



Voici la justification plus détaillée :

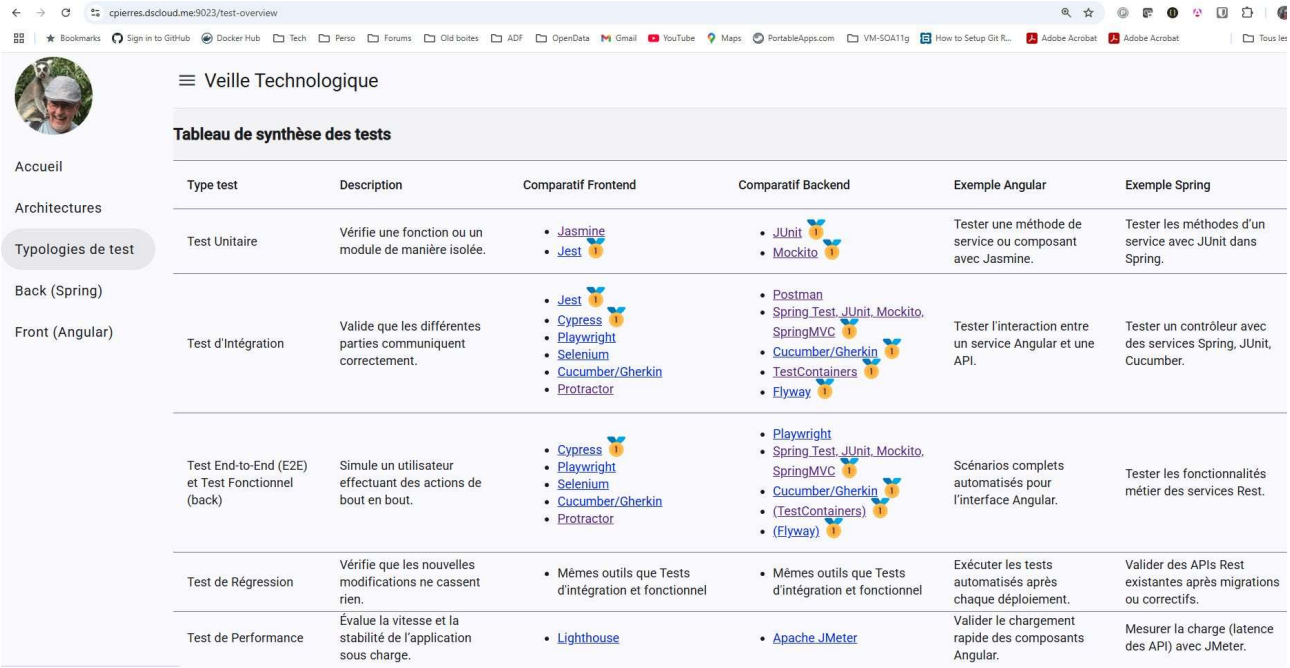


Grâce à l'approche Kubernetes, la disponibilité et la scalabilité peuvent être assurées. Veuillez consulter néanmoins les autres sections afin de comprendre les impacts budgétaires des autres possibillités.

3. *Choix et justification des librairies*

3.1. *Librairies de test*

https://cpierres.dscloud.me:9023/test-overview



Veille Technologique

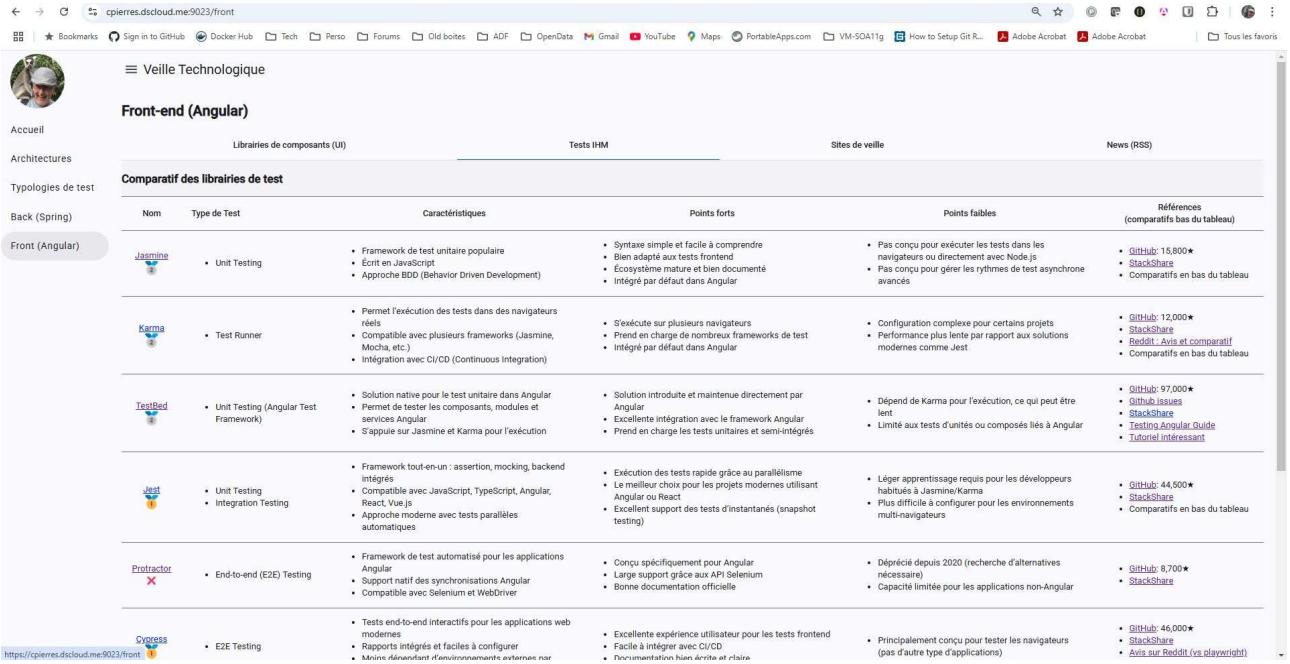
Accueil
Architectures
Typologies de test
Back (Spring)
Front (Angular)

Tableau de synthèse des tests

Type test	Description	Comparatif Frontend	Comparatif Backend	Exemple Angular	Exemple Spring
Test Unitaire	Vérifie une fonction ou un module de manière isolée.	• Jasmine • Jest	• JUnit • Mockito	Tester une méthode de service ou composant avec Jasmine.	Tester les méthodes d'un service avec JUnit dans Spring.
Test d'Intégration	Valide que les différentes parties communiquent correctement.	• Jest • Cypress • Playwright • Selenium • Cucumber/Gherkin • Protractor	• Postman • Spring Test, JUnit, Mockito, SpringMVC • Cucumber/Gherkin • TestContainers • Flyway	Tester l'interaction entre un service Angular et une API.	Tester un contrôleur avec des services Spring, JUnit, Cucumber.
Test End-to-End (E2E) et Test Fonctionnel (back)	Simule un utilisateur effectuant des actions de bout en bout.	• Cypress • Playwright • Selenium • Cucumber/Gherkin • Protractor	• Playwright • Spring Test, JUnit, Mockito, SpringMVC • Cucumber/Gherkin • (TestContainers) • (Flyway)	Scénarios complets automatisés pour l'interface Angular.	Tester les fonctionnalités métier des services Rest.
Test de Régression	Vérifie que les nouvelles modifications ne cassent rien.	• Mêmes outils que Tests d'intégration et fonctionnel	• Mêmes outils que Tests d'intégration et fonctionnel	Exécuter les tests automatisés après chaque déploiement.	Valider des APIs Rest existantes après migrations ou correctifs.
Test de Performance	Évalue la vitesse et la stabilité de l'application sous charge.	• Lighthouse	• Apache JMeter	Valider le chargement rapide des composants Angular.	Mesurer la charge (latence des API) avec JMeter.

3.1.1. *Font-end*

Pour une analyse détaillée des choix de librairies, consultez le lien : https://cpierres.dscloud.me:9023/front sur l'onglet **Tests IHM** :



Veille Technologique

Accueil
Architectures
Typologies de test
Back (Spring)
Front (Angular)

Front-end (Angular)

Librairies de composants (UI) | Tests IHM | Sites de veille | News (RSS)

Comparatif des librairies de test

Nom	Type de Test	Caractéristiques	Points forts	Points faibles	Références (comparatifs bas du tableau)
Jasmine	• Unit Testing	• Framework de test unitaire populaire • Écrit en JavaScript • Approche BDD (Behavior Driven Development)	• Syntaxe simple et facile à comprendre • Bien adapté aux tests frontend • Écosystème mature et bien documenté • Intégré par défaut dans Angular	• Pas conçu pour exécuter les tests dans les navigateurs ou directement avec Node.js • Pas conçu pour gérer les rythmes de test asynchrone avancés	• GitHub: 15,800★ • StackShare • Comparatifs en bas du tableau
Karma	• Test Runner	• Permet l'exécution des tests dans des navigateurs réels • Compatible avec plusieurs frameworks (Jasmine, Mocha, etc.) • Intégration avec CI/CD (Continuous Integration)	• S'exécute sur plusieurs navigateurs • Prend en charge de nombreux frameworks de test • Intégré par défaut dans Angular	• Configuration complexe pour certains projets • Performance plus lente par rapport aux solutions modernes comme Jest	• GitHub: 12,000★ • StackShare • Reddit : Avis et comparatif • Comparatifs en bas du tableau
TestBed	• Unit Testing (Angular Test Framework)	• Solution native pour le test unitaire dans Angular • Permet de tester les composants, modules et services Angular • S'appuie sur Jasmine et Karma pour l'exécution	• Solution introduite et maintenue directement par Angular • Excellente intégration avec le framework Angular • Prend en charge les tests unitaires et semi-intégrés	• Dépend de Karma pour l'exécution, ce qui peut être lent • Limité aux tests d'unités ou composés liés à Angular	• GitHub: 97,000★ • Github issues • StackShare • Testing Angular Guide • Tutoriel intéressant
Jest	• Unit Testing • Integration Testing	• Framework tout-en-un : assertion, mocking, backend intégrés • Compatible avec JavaScript, TypeScript, Angular, React, Vue.js • Approche moderne avec tests parallèles automatiques	• Exécution des tests rapide grâce au parallélisme • Le meilleur choix pour les projets modernes utilisant Angular ou React • Excellent support des tests d'instantanés (snapshot testing)	• Léger apprentissage requis pour les développeurs habitués à Jasmine/Karma • Plus difficile à configurer pour les environnements multi-navigateurs	• GitHub: 44,500★ • StackShare • Comparatifs en bas du tableau
Protractor ✕	• End-to-end (E2E) Testing	• Framework de test automatisé pour les applications Angular • Support natif des synchronisations Angular • Compatible avec Selenium et WebDriver	• Conçu spécifiquement pour Angular • Large support grâce aux API Selenium • Bonne documentation officielle	• Déprécié depuis 2020 (recherche d'alternatives nécessaire) • Capacité limitée pour les applications non-Angular	• GitHub: 8,700★ • StackShare
Cypress	• E2E Testing	• Tests end-to-end interactifs pour les applications web modernes • Rapports intégrés et faciles à configurer • Moins dépendant d'environnements externes par	• Excellente expérience utilisateur pour les tests frontend • Facile à intégrer avec CI/CD • Documentation bien écrite et claire	• Principalement conçu pour tester les navigateurs (pas d'autre type d'applications)	• GitHub: 46,000★ • StackShare • Avis sur Reddit (vs playwright)

https://cpierres.dscloud.me:9023/front

		• Moins dépendant d'environnements externes par rapport à d'autres outils	• Documentation bien écrite et claire	(pas d'autre type d'applications)	• Avis sur Reddit (vs playwright) • Autre avis sur Reddit
Playwright	• E2E Testing	• Framework moderne pour le test E2E • Compatible avec les navigateurs modernes (Chromium, Firefox, WebKit) • Supporte les tests via plusieurs langages (JavaScript, TypeScript, Python, Java, C#)	• Performances rapides grâce à des sessions de navigateur headless • Fonctionnalités avancées comme les tests parallèles, les fixtures et les traces interactives • Documentation riche et exemples faciles à suivre • Automatisation multiplateforme (Windows, macOS, Linux)	• Pas conçu pour les tests unitaires • Configuration initiale plus complexe pour les débutants • Maîtrise requise des promesses pour gérer les tests asynchrones	• GitHub: 69,200★ • StackShare • Reddit : avis Playwright (vs cypress et selenium)

Autres comparatifs

- Reddit : Discussion intéressante
- reddit: jasmine vs jest
- reddit: avis jasmine, karma vs jest
- Reddit : Comparatif JEST / Jasmine
- Reddit : Autre comparatif JEST / Jasmine
- lemagit : comparatif Cypress / Playwright

Angular intègre par défaut Jasmine, Karma et TestBed. Cependant, notre veille technologique (consultez les liens) montre que beaucoup se sont tournés vers Jest et Cypress. Dans un post récent sur reddit, on apprend que l'équipe d'Angular elle-même envisage de changer leurs propositions par défaut (éventuellement vers Jest), ce qui confirme notre proposition.

3.1.2. Back-end

Pour Spring, l'intégration des capacités de test est particulièrement bien soignée et les propositions par défaut sont excellentes. Spring permet d'intégrer avec facilité des briques optionnelles que nous avons retenues (TestContainers). Le tableau vu précédemment au début de cette section confirme les choix suivants :

Junit, Mockito, SpringTest, SpringMVC, Cucumber/Gherkin (expression des tests en langage naturel permettant aux fonctionnels de mieux comprendre le résultat des tests d'intégration ou fonctionnels), TestContainers et flyway pour la mise en œuvre de containers réalistes si besoin pour des jeux de tests plus poussés.

3.2. Librairie de composants visuels

Une analyse des librairies de composants est disponible sur : https://cpierres.dscloud.me:9023/front.

La solution par défaut incluse dans Angular est Angular Materials. Cette solution est robuste et intéressante. Néanmoins, le nombre de composants est faible. Pour une meilleure évolutivité sur les besoins, la solution de compromis conseillée est **PrimeNG** 🥇. Cette solution est la plus populaire d'après l'historique des stars. Néanmoins, ce tableau pourrait vous permettre d'affiner ce choix car si vous avez des demandes spécifiques (grosse volumétrie sur les tableaux ou composants spécifiques par exemple), d'autres solutions peuvent être préférables.

cpierres.dscloud.me:9023/front

Bookmarks | Sign in to GitHub | Docker Hub | Tech | Perso | Forums | Old boites | ADF | OpenData | Gmail | YouTube | Maps | PortableApps.com | VM-SOA11g | How to Setup Git R... | Adobe Acrobat | Adobe Acrobat | Tous les favoris

Veille Technologique

Front-end (Angular)

Accueil
Architectures
Typologies de test
Back (Spring)
Front (Angular)

Librairies de composants (UI) | Tests IHM | Sites de veille | News (RSS)

Comparatif des librairies

Nom	Avantages	Inconvénients	Caractéristiques principales	Cas d'utilisation	Références (Autres comparatifs en bas du tableau)
Angular Material 🥇	• Intégration native avec Angular • Fiable • Communauté active	• Manque parfois de personnalisation comparée à d'autres librairies.	• Prise en charge des bonnes pratiques Material Design • Mise à jour régulière.	• Applications Angular nécessitant une interface utilisateur conforme aux normes de Material Design.	• GitHub: 24,500★ • Angular Blog • Composants Material
PrimeNG 🥇	• Large éventail de composants riches en fonctionnalités, composants premium.	• Plus complexe à personnaliser (sauf depuis v18) • Qualité esthétique dépend des thèmes • Semble assez buggé et parfois mal optimisée (reddit - amélioration depuis v18 ?)	• Plus de 80 composants gratuits • Thèmes prêts à l'emploi, support d'accessibilité. • PrimeTek propose version payante avec support.	• Lorsque vous nécessitez un large choix de fonctionnalités et des thèmes prédéfinis.	• GitHub: 11,000★ • Site officiel • Article Medium
NG Zorro 🥈	• Basé sur Ant Design, dispose de composants élégants et modernes.	• Moins connu que d'autres librairies comme Angular Material • Moindre communauté francophone.	• Prise en charge d'Ant Design • Système de composants riche pour des interfaces complexes.	• Applications d'administration, dashboards, ou projets nécessitant des interfaces élégantes.	• GitHub: 8,900★ • Documentation • Avis reddit • Comparatif reddit
Taiga UI 🥈	• 130 composants. • Moderne, hautement personnalisable • Support complet pour Angular • Documentation détaillée. • En forte progression.	• Moins de support communautaire comparé à des librairies plus populaires comme Angular Material.	• Large gamme de composants flexibles avec des animations subtiles • Support thématique • Légèreté.	• Applications modernes nécessitant un design minimaliste et flexible.	• GitHub: 3,400★ • Documentation • Avis reddit • Avis reddit
AG-Grid 🥇	• Performances élevées pour le rendu de données volumineuses • Fonctionnalités avancées pour les tableaux	• Composants communautaires peu nombreux. • Licence payante nécessaire pour les fonctionnalités avancées	• Composants de tableaux dynamiques et hautement personnalisables • Idéal pour des applications data-centric.	• Applications nécessitant des tableaux complexes et interactifs manipulant de larges volumes de données.	• GitHub: 13,400★ • Documentation officielle • Tarifs

https://cpierres.dscloud.me:9023/front

Accueil

Architectures

Typologies de test

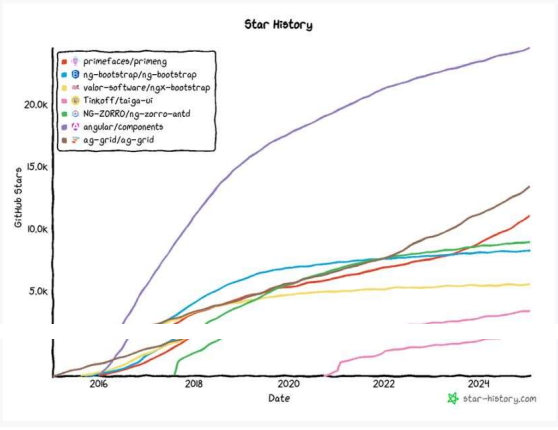
Back (Spring)

Front (Angular)

	(tri, filtrage, édition) et les graphes.	• Apprentissage assez difficile.	• Idéal pour des applications data-centric.		• Tarifs • Articles Medium
NG Bootstrap	• Utilisation native des composants Bootstrap • Fidèle à la simplicité.	• Basé uniquement sur Bootstrap, ce qui limite la personnalisation.	• Composants intuitifs basés sur Bootstrap.	• Applications qui utilisent déjà Bootstrap et nécessitent une intégration rapide.	• GitHub: 8,200★ • Comparatif
ngx-bootstrap	• Grande collection de composants Bootstrap réutilisables pour Angular.	• Moins fréquent dans les projets modernes comparé à Angular Material.	• Prise en charge des fonctionnalités de Bootstrap 3 et 4 • large éventail de composants comme les Modals, Alerts, Tabs, etc.	• Applications nécessitant une approche basée sur Bootstrap avec une intégration facile.	• GitHub: 5,500★ • Github Issues

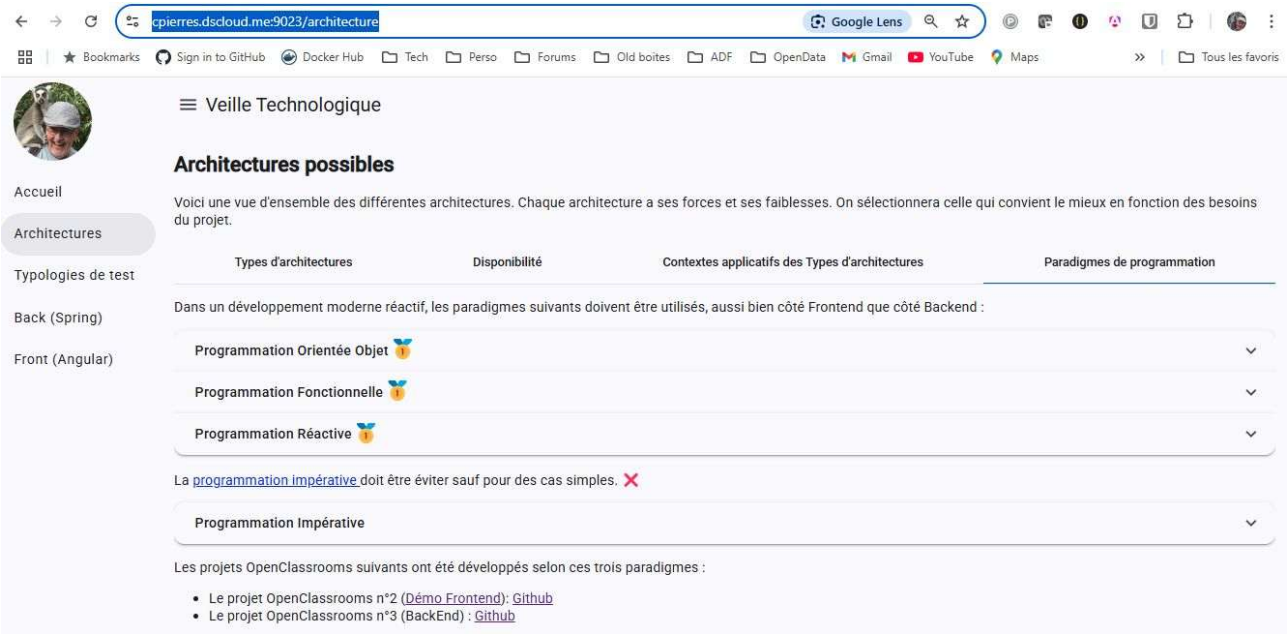
Autres comparatifs

- Comparaison star-history.com
- Comparatif reddit intéressant entre PrimeNG, Material et NG Zorro
- Reddit : Comparaison de librairies UI pour Angular
- 10 Angular UI Libraries to Create a World-Class User Experience
- 9 bibliothèques de composants Angular
- PrimeNg vs NgBootstrap vs NgxBootstrap

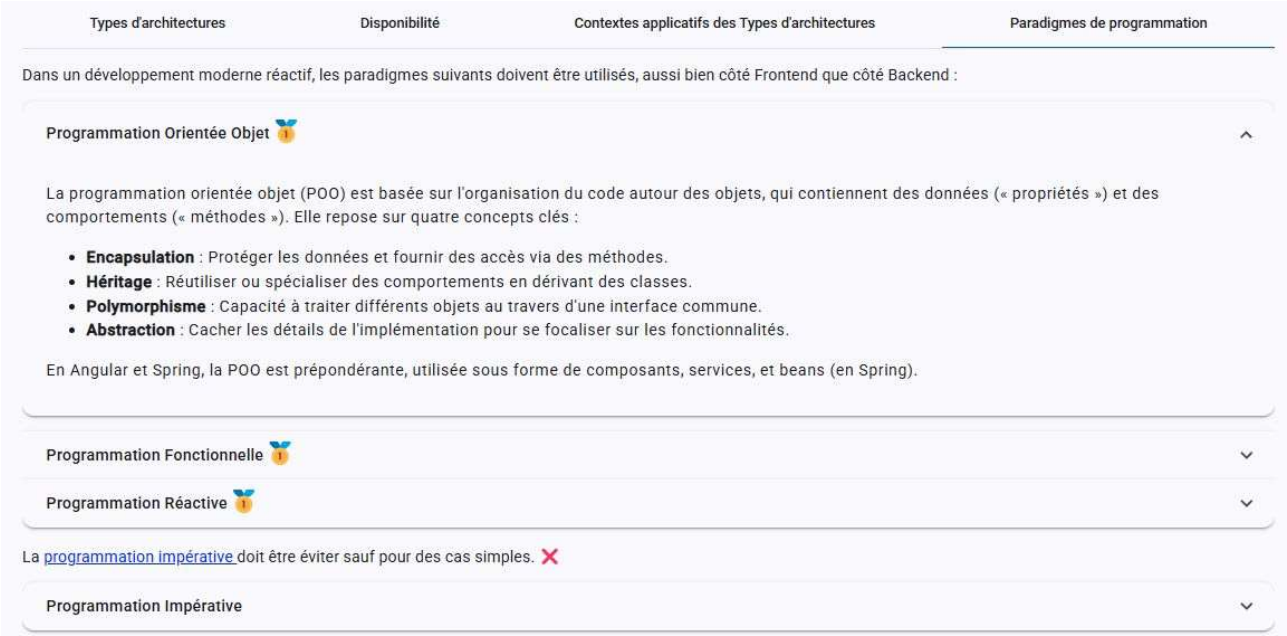


4. Choix et justification des paradigmes de programmation

Le choix des paradigmes de programmation a été placé sur le lien https://cpierres.dscloud.me:9023/architecture , onglet **Paradigmes de programmation** :



Vous pouvez cliquer sur chaque section pour la justification :



Types d'architectures | Disponibilité | Contextes applicatifs des Types d'architectures | Paradigmes de programmation

Dans un développement moderne réactif, les paradigmes suivants doivent être utilisés, aussi bien côté Frontend que côté Backend :

Programmation Orientée Objet

Programmation Fonctionnelle

La programmation fonctionnelle repose sur des fonctions pures pour manipuler des données. Elle privilégie la **déclarativité**, l'**immutabilité**, ainsi que des fonctions d'ordre supérieur.

Grâce aux **Streams** (comme en Java ou RxJS dans Angular), ce paradigme devient extrêmement concis et efficace :

- **Concision :** Les Streams réduisent considérablement le code requis par rapport à des boucles impératives.
- **Efficacité :** Approche basée sur l'exécution paresseuse (lazy evaluation), ne calculant que ce qui est nécessaire.
- **Chaînage :** Les opérations comme `map`, `filter`, `reduce` clarifient la logique.

Utilisation typique avec Angular et Spring :

- En Angular, RxJS permet un chaînage clair des actions sur des Observables, parfait pour traiter des données dynamiques.
- En Spring, depuis Java 8, les Streams permettent de filtrer et transformer les collections de manière concise et performante.

Programmation Réactive

La programmation impérative doit être éviter sauf pour des cas simples. ❌

Types d'architectures | Disponibilité | Contextes applicatifs des Types d'architectures | Paradigmes de programmation

Dans un développement moderne réactif, les paradigmes suivants doivent être utilisés, aussi bien côté Frontend que côté Backend :

Programmation Orientée Objet

Programmation Fonctionnelle

Programmation Réactive

La programmation réactive met l'accent sur la gestion des flux de données **asynchrones**. Ce paradigme est particulièrement utile pour traiter des événements, ou des flux en temps réel.

- Les Observables (côté Angular via RxJS) et les Publishers/Flux (côté Spring via Reactor) permettent de manipuler et transformer des flux dynamiques de manière similaire aux Streams, mais avec une gestion automatique de l'asynchronisme.
- **Exemple pratique Angular :** Utiliser RxJS pour filtrer, regrouper ou transformer des événements utilisateurs avant de les afficher.
- **Exemple pratique Spring :** Spring WebFlux propose une gestion réactive des requêtes et des réponses HTTP.

La combinaison des paradigmes fonctionnels et réactifs est particulièrement puissante pour optimiser les performances des applications modernes.

La programmation impérative doit être éviter sauf pour des cas simples. ❌

Programmation Impérative

La programmation impérative doit être évitée comme expliqué ci-après :

Types d'architectures | Disponibilité | Contextes applicatifs des Types d'architectures | Paradigmes de programmation

Dans un développement moderne réactif, les paradigmes suivants doivent être utilisés, aussi bien côté Frontend que côté Backend :

Programmation Orientée Objet

Programmation Fonctionnelle

Programmation Réactive

La programmation impérative doit être éviter sauf pour des cas simples. ❌

Programmation Impérative

La programmation impérative repose sur des instructions et des structures de contrôle comme les boucles, conditions, etc. Elle est souvent inefficace et difficile à maintenir dans un contexte moderne :

- **Moins concise :** Nécessite souvent plus de lignes de code en manipulant directement les structures de données.
- **Mutable :** Conduit souvent à des erreurs dues à la modification de l'état des variables.

Toutefois, elle peut être utile pour des scripts ou des tâches simples mais doit être évitée dans des applications professionnelles modernes.

Les projets OpenClassrooms suivants ont été développés selon ces trois paradigmes :

- Le projet OpenClassrooms n°2 (Démo Frontend): Github
- Le projet OpenClassrooms n°3 (BackEnd) : Github

Les projets OpenClassrooms suivants ont été développés selon ces trois paradigmes :

- projet n°2 (Démo Frontend): Github
- projet n°3 (BackEnd) : Github