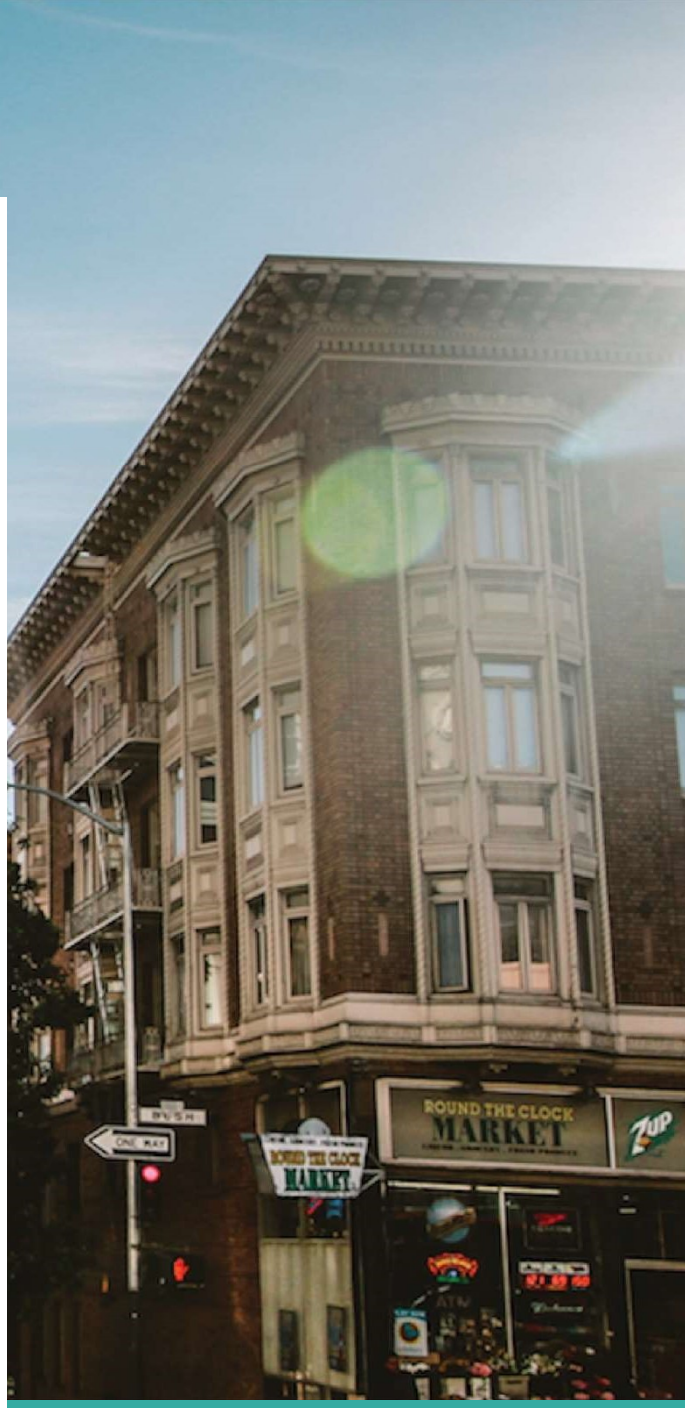
Document de reporting 2024



6 JUIN

Open Classrooms
Créé par : Ouelaa Icham

Logo
Nom

Document de reporting

NORMES ET RGPD

Dans mon projet, je veille à respecter les normes, notamment le RGPD, ainsi que les principes de sécurité de plusieurs manières :

- Je sécurise ma base de données : En m'assurant que ma base de données est sécurisée, je garantis la confidentialité et l'intégrité des données stockées. Cela signifie que seules les personnes autorisées ont accès aux données et que des mesures de sécurité robustes sont mises en place pour prévenir les accès non autorisés.

- J'assure un accès sécurisé à ma base de données : La mise en place d'un système d'authentification sécurisé, tel qu'une page de connexion, garantit que seules les personnes autorisées peuvent accéder à ma base de données. Cela réduit le risque d'accès non autorisé et assure que seules les personnes ayant les droits appropriés peuvent consulter ou modifier les données.

- Je respecte le RGPD : En sécurisant ma base de données et en contrôlant l'accès aux données sensibles, je respecte les principes fondamentaux du RGPD, tels que la protection des données personnelles, la limitation de leur traitement aux fins pour lesquelles elles ont été collectées, et la garantie de leur confidentialité et de leur sécurité.

- Protection de la vie privée : En mettant en place des mesures de sécurité telles que le cryptage des données sensibles et la limitation de l'accès aux informations personnelles, je garantis la protection de la vie privée des utilisateurs de mon projet. Cela contribue à établir un climat de confiance entre l'utilisateur et le service proposé.

- Transparence : Je m'efforce de maintenir un haut niveau de transparence concernant la collecte, l'utilisation et la gestion des données au sein de mon projet. Cela inclut la fourniture d'informations claires sur les politiques de confidentialité et les pratiques de traitement des données, permettant ainsi aux utilisateurs de prendre des décisions éclairées quant à la manière dont leurs données sont utilisées.

- Minimisation des données : Je limite la collecte et le stockage des données personnelles au strict nécessaire pour atteindre les objectifs de mon projet. En adoptant une approche de minimisation des données, je réduis le risque de collecte excessive ou non pertinente de données, tout en respectant le principe de proportionnalité.

- Intégrité des données : Je mets en place des mécanismes de sécurité et de contrôle de qualité pour assurer l'intégrité des données stockées dans ma base de données. Cela garantit que les informations

manipulées par mon projet sont exactes, complètes et fiables, ce qui renforce la confiance des utilisateurs dans le système.

TECHNOLOGIES BACK-END ET FRONT-END

Technologie back end

J'ai choisi d'utiliser Java avec le framework Spring pour le développement back-end de mon projet pour plusieurs raisons. Tout d'abord, Java est un langage de programmation mature et largement utilisé dans l'industrie, ce qui garantit une grande fiabilité et une vaste communauté de support. En utilisant Java, je peux bénéficier de sa robustesse et de sa performance pour gérer efficacement les charges de travail importantes.

De plus, Spring est un framework de développement d'applications Java qui offre une architecture modulaire et extensible. Il simplifie le développement en fournissant des fonctionnalités telles que l'injection de dépendances, la gestion des transactions et la sécurité, ce qui accélère le processus de développement et améliore la maintenabilité du code.

En choisissant Spring, je peux également profiter de ses nombreux modules et extensions, tels que Spring Boot, qui simplifie la configuration et le déploiement des applications, et Spring Security, qui fournit des fonctionnalités robustes pour la sécurisation des applications web.

En résumé, l'utilisation de Java avec Spring pour le développement back-end de mon projet me permet de bénéficier d'une combinaison de puissance, de fiabilité et de productivité, ce qui me permet de créer des applications web robustes, évolutives et sécurisées.

Technologie front end :

J'ai choisi d'utiliser HTML, CSS et Angular pour le développement front-end de mon projet pour plusieurs raisons. Tout d'abord, HTML est le langage de balisage standard pour la création de la structure de base d'une page web, offrant une syntaxe simple et claire pour organiser le contenu de manière logique.

Ensuite, CSS est essentiel pour styliser et mettre en forme le contenu HTML, permettant de créer des mises en page esthétiques et réactives. Avec ses fonctionnalités avancées telles que les sélecteurs, les cascades et les grilles, CSS offre une flexibilité et une puissance pour concevoir des interfaces utilisateur attrayantes et conviviales.

Enfin, j'ai opté pour Angular, un framework JavaScript moderne et robuste, pour ajouter des fonctionnalités interactives et dynamiques à mon projet. Angular facilite la création d'applications web à page unique (SPA) grâce à sa structure de composants, son système de liaison de données bidirectionnelle, et ses nombreuses

fonctionnalités prêtes à l'emploi telles que le routage, la validation de formulaires, et la gestion de l'état de l'application.

En combinant HTML, CSS et Angular, j'ai pu créer une expérience utilisateur riche et fluide, offrant à mes utilisateurs une interface intuitive et réactive, tout en facilitant le développement et la maintenance de mon application grâce à la modularité et à la structure bien définie de ces technologies.

RESULTATS ET DES ENSEIGNEMENTS DE LA POC

Bien sûr, voici quelques exemples de résultats du bon fonctionnement d'un projet POC et les enseignements tirés du processus de développement :

1. Validation de la faisabilité technique : Le POC démontre que la technologie envisagée peut effectivement résoudre le problème identifié. Par exemple, un POC pour une application de reconnaissance faciale montre que le modèle de machine learning sélectionné peut identifier les visages avec précision dans des conditions variées.

Enseignement: Il confirme la viabilité technique de l'approche choisie et permet d'identifier les éventuelles limites ou défis techniques à surmonter.

2. Retour rapide sur investissement : Le POC est livré dans des délais courts avec un budget limité, fournissant des résultats tangibles. Par exemple, le POC pour l'application medhead génère un prototype fonctionnel en quelques semaines avec une petite équipe de développement.

Enseignement: Il démontre l'efficacité de l'équipe et la pertinence des méthodes de développement utilisées, tout en mettant en évidence les processus qui pourraient être améliorés pour accélérer encore les itérations futures.

3. Identification des besoins utilisateur : Le POC inclut des tests utilisateur qui révèlent des informations précieuses sur les besoins et les préférences des utilisateurs finaux. Par exemple, un POC pour une application de gestion de projet révèle que les utilisateurs apprécient particulièrement une fonctionnalité de suivi en temps réel.

Enseignement: Il met en lumière l'importance de l'écoute des utilisateurs dès les premières étapes du développement et souligne l'importance de l'itération basée sur les commentaires des utilisateurs.

4. Réduction des risques : Le POC permet de résoudre des problèmes potentiels ou d'identifier des risques plus tôt dans le processus de développement. Par exemple, un POC pour un système de gestion de bases de données distribuées révèle des goulets d'étranglement de performances à grande échelle.

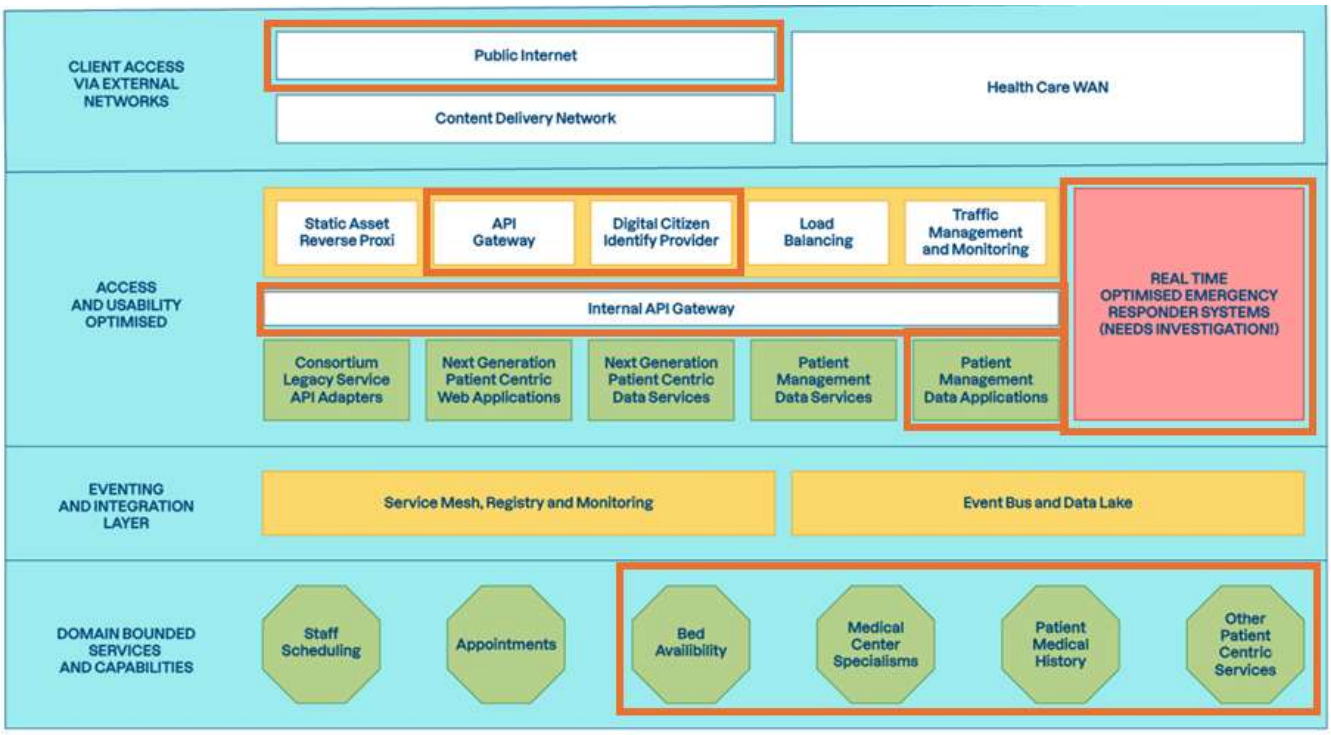
Enseignement: Il souligne l'importance de l'analyse précoce des risques et de l'exploration de solutions alternatives pour atténuer ces risques.

5. Validation du modèle économique : Le POC montre que le modèle économique envisagé est viable et peut générer des revenus. Par exemple, un POC pour une plateforme de location de voitures démontre que le modèle de tarification par minute est attractif pour les utilisateurs.

Enseignement: Il valide l'aspect commercial du projet et fournit des données tangibles pour étayer les projections financières.

En résumé, les résultats du bon fonctionnement d'un projet POC incluent la validation de la faisabilité technique, un retour rapide sur investissement, une meilleure compréhension des besoins utilisateurs, la réduction des risques et la validation du modèle économique. Les enseignements tirés du processus de développement du projet POC fournissent des informations précieuses pour guider les itérations futures du projet et maximiser ses chances de succès.

SCHEMA D'ARCHITECTURE ET POC



Principaux éléments liés au POC :

Couche Client Access via external Networks

L'internet public représente le réseau mondial qui relie les ordinateurs et les appareils dans le monde entier. C'est le principal moyen pour les clients d'accéder à l'application via un navigateur (Chrome, Firefox, Safari ou autres).

Couche Access and usability Optimised

La passerelle API qui a été mise en place est utilisée pour gérer et sécuriser les demandes d'API, en offrant des fonctionnalités telles que l'accès à des points d'entrées liés à la base de données et la gestion et l'acheminement des demandes (appli API Hôpital).

Le composant AuthController est responsable de la validation des informations d'identification des utilisateurs, il utilise une méthode d'authentification basée sur des identifiants et des mots de passe pour gérer les connexions utilisateurs.
Il permet également de gérer des profils lors de la connexion (patients ou professionnels de santé).

La passerelle API interne est utilisée pour fournir un accès aux services internes de l'application. Par exemple, elle peut être utilisée pour donner accès au service de gestion des patients à partir de l'application web (appli API Hôpital)

L’application fournis une fonctionnalité de gestion des patients à l’aides de certains composants comme le controller PatientController et le service PatientService. Ils sont utilisés pour gérer les informations et les différents besoins des patients.

L’application permet d'optimiser en temps réel les systèmes d'intervention d'urgence en pouvant gérer de nombreuse intervention en même temps (tests de stress). L’api a également été testé pour pouvoir gérer de nombreuses requêtes en simultanées et donc améliorer l'efficacité des opérations d'intervention.

Pour optimiser le traitement en temps réel de notre projet, j'ai mis en place plusieurs stratégies.

1. Parallélisme des tâches, balance de charges et stress test :
J'ai mis en place des solutions pour gérer pour le parallélisme des tâches, permettant de traiter les événements de manière asynchrone et en parallèle. J'ai également gérer le traitement des flux de données en temps réel. Pour garantir que le système peut gérer des charges élevées, j'ai effectué des stress tests avec **JMeter**, simulant des scénarios de forte demande pour identifier et corriger les goulets d'étranglement.

2. System Load Balancing :
J'ai mis en place un équilibrage de charge (load balancer) pour distribuer les demandes entrantes de manière uniforme entre plusieurs instances de serveurs, assurant ainsi une utilisation optimale des ressources et évitant les surcharges sur un seul serveur.

3. Argumenter demande traitée en live*:
Avec ce système en place, nous sommes capables de traiter les demandes des utilisateurs en temps réel. Les requêtes sont instantanément traitées permettant une réponse immédiate et une expérience utilisateur fluide.

4. Calculer et donner le nombre de demandes simultanées possibles :
Grâce aux stress tests effectués avec JMeter, j'ai déterminé que notre infrastructure peut gérer jusqu'à 2000 demandes simultanées sans dégradation de performance. Cela a été validé par des tests rigoureux, garantissant que le système reste stable et réactif sous des charges élevées.

Ces mesures combinées assurent une haute performance et une fiabilité du traitement en temps réel, répondant efficacement aux besoins des utilisateurs et aux exigences du projet.

Couche domain bounded services and capabilities

Les composants hôpital services et controller fournissent des fonctionnalités de planification et de disponibilité pour le personnel et les lits. Ils peuvent être utilisés pour gérer la disponibilité des lits et leur réservation (également en fonction d'une adresse pour permettre une réservation dans l'hôpital le plus proche.

Le composant Hôpital fournit des informations sur les spécialités des différents hôpitaux et permettent de gérer les réservations de lits en fonction des besoins en termes de spécialité.

Le composant Patient permet d'accéder aux antécédents médicaux d'un patient. Il peut être utilisé pour récupérer des informations sur les spécialités, les blessures et les historiques de traitements dans des hôpitaux d'un patient.
Il est également possible via ce service de gérer les autres informations du patient tels que son adresse, son numéro etc.

Role des différentes couches :

1. Client Access via external Networks :

Cette couche représente le point d'entrée du système depuis des réseaux externes, tels qu'Internet ou des réseaux de partenaires. Les interactions dans cette couche sont principalement orientées vers l'accès et l'authentification des utilisateurs externes, ainsi que la gestion des requêtes entrantes.

2. Access and Usability Optimised :

Cette couche se situe juste après la couche d'accès client et vise à optimiser l'expérience utilisateur en fournissant des fonctionnalités telles que la gestion de session, la personnalisation du contenu et l'optimisation de l'interface utilisateur. Les interactions dans cette couche impliquent souvent la récupération et la manipulation de données pour répondre aux besoins spécifiques des utilisateurs.

3. Eventing and Integration Layer :

Cette couche agit comme un hub central pour la gestion des événements et l'intégration avec d'autres systèmes. Les interactions dans cette couche incluent la réception et la diffusion d'événements, ainsi que l'intégration avec des services externes. Cette couche facilite également la communication entre les différents composants de l'application.

4. Domain Bounded Services and Capabilities:

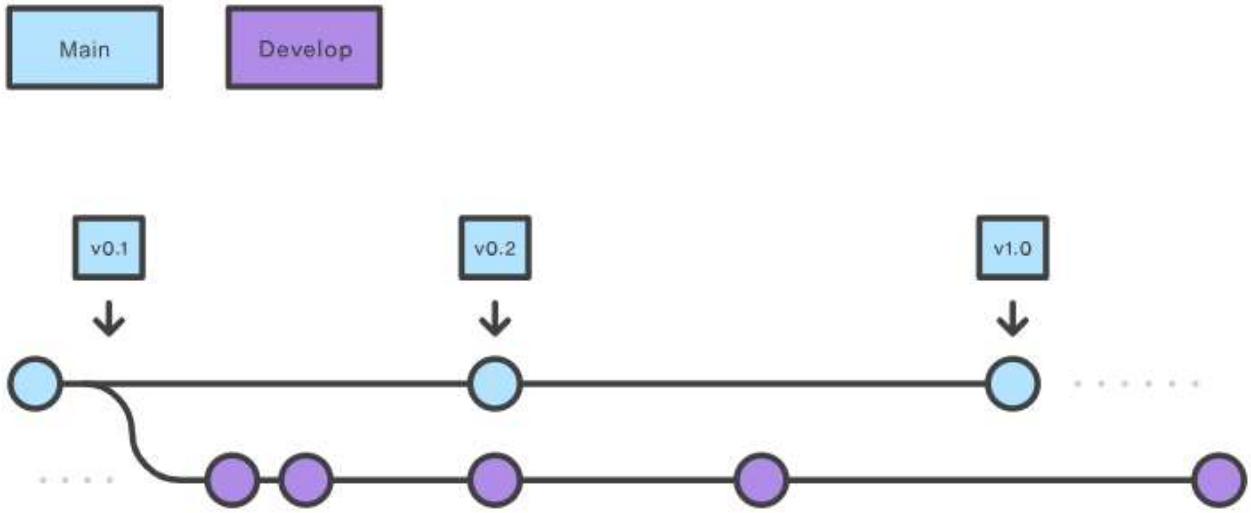
Cette couche représente le cœur fonctionnel de l'application, où se trouvent les services métier et les fonctionnalités principales. Les interactions dans cette couche sont axées sur l'exécution des fonctionnalités métier, telles que le traitement des commandes, la gestion des utilisateurs, ou d'autres opérations spécifiques liés à la gestion des patients ou des hopitaux. Cette couche peut également interagir avec la couche d'intégration pour accéder à des données externes ou déclencher des événements.

Les interactions entre ces couches varient en fonction des besoins spécifiques de l'application, mais généralement, les données et les requêtes passent à travers les différentes couches, avec des transformations et des actions spécifiques à chaque couche. Par exemple, une requête provenant d'un client externe pourrait passer par les couches d'accès client et d'optimisation de l'accessibilité avant d'atteindre les services métier dans la couche des services métier et des capacités. De même, les événements générés par les services métier pourraient être propagés à travers la couche d'intégration pour déclencher des actions dans d'autres systèmes externes. En résumé, chaque couche joue un rôle spécifique dans le traitement des données et des événements, contribuant ainsi au fonctionnement global du système.

Dans le cadre de ce projet j'ai utilisé un workflow Git utilisant une branche `develop` avant de pousser les changements sur la branche `main`. C'est une pratique courante pour organiser et gérer le développement de logiciels de manière structurée.

WORKFLOW :

1. Schéma workflow



2. Principe du workflow Git avec branche `develop`

a) Stabilité de la branche `main` : La branche `main` (ou `master` dans certains projets) est considérée comme la branche principale où le code doit toujours être en état stable et prêt pour la production. Elle ne doit contenir que du code qui a été soigneusement testé et approuvé.

b) Développement sur `develop` : La branche `develop` est utilisée pour intégrer les fonctionnalités en cours de développement. C'est une sorte de pré-production où les développeurs fusionnent leurs branches de fonctionnalité (feature branches) après les avoir développées et testées.

3. Fonctionnement du workflow

a) Création des branches de fonctionnalité (feature branches) :
- Chaque nouvelle fonctionnalité ou amélioration commence par la création d'une nouvelle branche à partir de `develop`. Ces branches sont nommées de manière descriptive, par exemple, `feature/nouvelle-fonctionnalite`.
- Les développeurs travaillent sur ces branches indépendamment, permettant un développement parallèle sans affecter la branche principale.

b) Intégration des fonctionnalités dans `develop` :
- Une fois qu'une fonctionnalité est terminée et testée localement, la branche de fonctionnalité est fusionnée dans `develop`. Cela peut se faire via une Pull Request (PR) ou une Merge Request (MR) pour faciliter la revue de code par d'autres développeurs.
- Les tests automatisés (CI/CD) peuvent être exécutés à ce stade pour s'assurer que l'intégration de la nouvelle fonctionnalité n'introduit pas de bugs dans le code existant sur `develop`.

c) Préparation pour la mise en production :

- Une fois que plusieurs fonctionnalités ont été intégrées dans `develop` et que le code est jugé suffisamment stable, on peut préparer une nouvelle version pour la production.
- Cela implique de fusionner `develop` dans `main`. Avant cette fusion, une branche de release (`release`) peut être créée à partir de `develop` pour effectuer les derniers tests, corrections de bugs, et autres préparations nécessaires.

d) Fusion dans `main` :
- Une fois que la branche de release est prête et testée, elle est fusionnée dans `main`.
- Des étiquettes (tags) peuvent être utilisées sur `main` pour marquer les versions spécifiques (par exemple, `v1.0.0`).
- La branche `main` mise à jour peut ensuite être déployée en production.

e) Maintenance :
- Si des bugs sont découverts en production, des branches de hotfix peuvent être créées à partir de `main` pour corriger ces problèmes. Une fois corrigés, ces hotfix sont fusionnés à la fois dans `main` et `develop` pour assurer que les corrections sont propagées dans toutes les versions du code.

4. Avantages de ce workflow

- Isolation des fonctionnalités : Chaque nouvelle fonctionnalité est développée indépendamment, ce qui minimise les conflits entre les développeurs.
- Stabilité du code : La branche `main` reste toujours en état stable et prêt pour la production.
- Intégration continue : Les fonctionnalités sont intégrées progressivement dans `develop`, facilitant la détection précoce des bugs et les tests continus.
- Gestion des versions : L'utilisation de branches de release permet une préparation soignée avant de déployer une nouvelle version en production.

Ce workflow est souvent appelé "Git Flow" et est particulièrement adapté aux projets de grande envergure avec plusieurs développeurs travaillant en parallèle sur différentes fonctionnalités.

CI/CD

1. Introduction au CI/CD avec GitLab

GitLab CI/CD est un outil d'intégration continue et de déploiement continu (CI/CD) intégré à GitLab. Il permet d'automatiser les processus de construction, de test et de déploiement de logiciels. Une pipeline CI/CD dans GitLab est définie dans un fichier `.gitlab-ci.yml`, qui spécifie les différentes étapes (stages) et les tâches (jobs) à exécuter.

2. Pipeline GitLab CI/CD

La pipeline CI/CD que j'ai mis en place est définie dans un fichier `.gitlab-ci.yml` et se compose de plusieurs étapes qui couvrent la construction, les tests et le déploiement d'une application hospitalière. Voici un résumé de chaque étape :

a) Stages (Étapes) :
- `build_hopital_api` : Construction de l'API de l'hôpital.
- `build_hopital_webapp` : Construction de l'application web de l'hôpital.
- `test_jmeter` : Test de performance avec JMeter.
- `test_selenium_Scenario1` : Test Selenium pour le scénario 1.
- `test_selenium_Scenario2` : Test Selenium pour le scénario 2.
- `deploy` : Déploiement de l'application.

b) Jobs (Tâches) :
build_hopital_api :
- Utilise l'image Docker `maven:latest`.
- Compile l'API de l'hôpital avec Maven.

build_hopital_webapp :
- Utilise l'image Docker `maven:latest`.
- Compile l'application web de l'hôpital avec Maven.

test_jmeter :
- Utilise l'image Docker `justb4/jmeter:latest`.
- Exécute un test de performance avec JMeter.

test_seleniumScenario1 :
- Utilise l'image Docker `selenium/standalone-chrome:latest`.
- Installe Java (JRE et JDK).
- Exécute un test Selenium pour le scénario 1 des patients.

test_seleniumScenario2 :

- Utilise l'image Docker `selenium/standalone-chrome:latest`.
- Installe Java (JRE et JDK).
- Exécute un test Selenium pour le scénario 2 des professionnels de santé.

deploy :
- Simule le déploiement de l'application.
- Inclut des messages de log pour indiquer le début et la fin du déploiement.

3. Résumé de la Pipeline

Cette pipeline GitLab CI/CD automatise la construction, les tests et le déploiement d'une application hospitalière. Elle commence par la construction de l'API et de l'application web, puis exécute des tests de performance avec JMeter et des tests fonctionnels avec Selenium. Enfin, elle simule le déploiement de l'application. Ce flux de travail assure que le code est correctement construit et testé avant d'être déployé en production, garantissant ainsi une meilleure qualité et une plus grande fiabilité du logiciel.

Gouvernance

La gouvernance d'architecture est le cadre de gestion et de supervision des processus, des normes et des pratiques permettant de guider et de contrôler l'architecture d'un système ou d'une organisation. Son objectif est de garantir que les décisions architecturales alignent les projets informatiques avec les objectifs stratégiques de l'entreprise, assurent la conformité aux normes et aux réglementations, et optimisent l'efficacité et la cohérence des systèmes informatiques. En somme, elle veille à ce que l'architecture soit robuste, scalable et en phase avec les besoins opérationnels et stratégiques de l'organisation.

Architecture Micro Services :

J'ai choisi de passer d'une architecture client-serveur à une architecture microservices dans le cadre d'une preuve de concept pour plusieurs raisons clés.
Tout d'abord, une architecture microservices permet une meilleure évolutivité et une flexibilité accrue. Chaque service étant indépendant, je peux développer, déployer et mettre à jour des composants individuels sans perturber l'ensemble du système. Cela facilite également la résilience et l'isolation des pannes : une défaillance dans un microservice n'affecte pas nécessairement les autres.
De plus, cette approche permet une meilleure gestion des équipes de développement, chaque équipe pouvant se concentrer sur un microservice spécifique en utilisant la technologie la plus adaptée à ses besoins.
Enfin, l'architecture microservices est particulièrement adaptée aux méthodologies DevOps et CI/CD, ce qui favorise une livraison continue et une intégration plus rapide des nouvelles fonctionnalités.