Plan de test

Projet : Création d'un POC

Sommaire

Sommaire **2**

Processus du plan de test **3**

Cadre du plan de test **3**

Hypothèses d'interface sur les apis **5**

Service Patient 5

Service Hôpital 5

Service Docteur 6

Service Rendez-vous 6

Service Urgence 6

Tests unitaires **7**

Plan de test **8**

Tests de performance / charge **9**

Exemple de KPI (Key Performance Indicator) 10

Test de résilience **11**

Déterminer le contexte 11

Composants critiques 11

Résilience architecturale 12

Mise en place de tests de résilience 12

Test de sécurité **12**

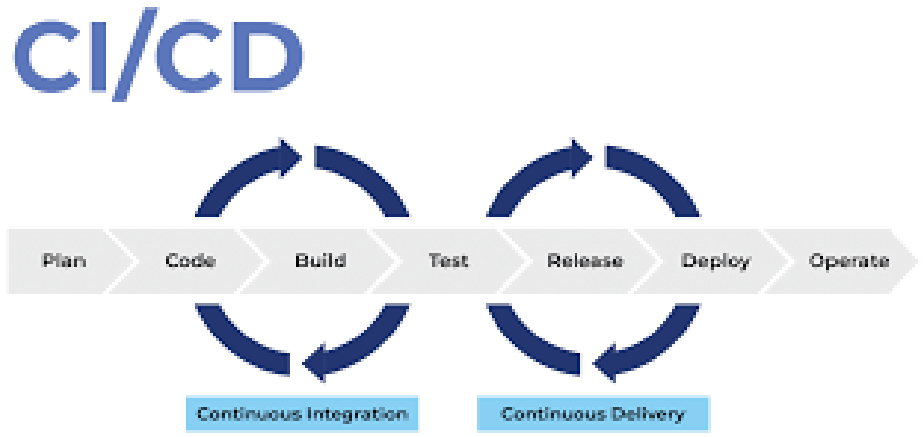
Test de contrat **13**

Processus du plan de test

Les processus à mettre en place pour le plan de test sont les suivants :

- Mise en place d'un environnement de test sur lesquels les tests seront déroulés par nos testeurs et nos utilisateurs de tests;
- Déroulement de tous les tests à la fin de chaque sprint et avant chaque mise en production;
- Plus globalement, chaque livraison de code doit être éprouvée par la validation du plan de test;
- Début des tests dès le début du développement de l'architecture jusqu'à la fin du projet;

Bien évidemment, si les résultats des tests ne sont pas conformes à ce à quoi nous nous attendions, la mise en production sera retardée ou le contenu du sprint retravaillé. Le but est de ne livrer que fonctionnalités parfaitement fonctionnelles et éprouvées par le plan de test.



Processus de livraison de code automatique

Nous utiliserons également les différentes applications de récolte de données pour évaluer le taux d'erreur, le niveau des erreurs et la quantité en fonction des scénarios. C'est un critère à prendre en compte lors de la validation du plan de test.

Cadre du plan de test

Services existants	Description du service	Nécessité d'ajout de tests	Implémenté dans le POC

Service Utilisateur	Service qui gère les utilisateurs, leurs identifiants, leurs rôles et leurs informations personnelles.	Oui	Non
Service Patient	Service qui gère les patients, leurs données médicales, leurs historiques d'interventions et de rendez-vous.	Oui	Oui
Service Hôpital	Service qui gère les hôpitaux, les lits et le personnel disponibles.	Oui	Oui
Service Docteur	Service qui gère les docteurs enregistrés dans le système, leur emploi du temps et leurs rendez-vous.	Oui	Oui
Service Rendez-vous	Service qui gère la prise de rendez-vous d'un patient dans un hôpital.	Oui	Oui
Service Urgence	Service qui gère les urgences et la redirection des patients vers l'hôpital disponible le plus proche.	Oui	Oui

Type de test	Description	Récupération des données de test
Test unitaire	Test des composants applicatifs de manière individuelle. Ces tests valident le bon fonctionnement ainsi que la non régression des composants de l'application.	Utilisation de l'application SonarQube, outil de CI/CD
Test API	Test des différents paramètres et différentes actions possibles pour chaque API	Utilisation de l'application JMeter
Test intégration	Tests des composants lorsqu'ils sont utilisés ensembles	Utilisation de l'application ELK
Test système	Évaluation complète et intégrale du système et de sa souplesse	Récupération manuelle des données de test
Test sécurité	Test de la sécurité de l'architecture et des composants	Utilisation des évaluations O'Wasp et d'un audit de sécurité
Test charge	Test de la montée en charge	JMeter et ELK
Contract Test	Test indépendant des composants en mode 'boite noire', c'est à dire sans s'intéresser à leur fonctionnement propre	Utilisation de l'application SonarQube, ELK

End-to-End	Test de workflow complets	Utilisation de l'application SonarQube ELK

Hypothèses d'interface sur les apis

Service Patient

Description	Method	Entry	HTTP status
Ajout	POST	in : patientId, name, position	200 : OK 403 : Unauthorized
Suppression	DELETE	in : patientId	200 : OK 403 : Unauthorized 404 : Not found
Récupération des infos	GET	in : patientId	200 : OK 403 : Unauthorized 404 : Not found
Modifications	PUT/PATCH	in : patientId, name, position..	200 : OK 403 : Unauthorized 404 : Not found

Service Hôpital

Description	Method	Entry	Expected HTTP status
Ajout d'un hôpital	POST	in : hospitalId, position, beds, bedsAvailable	200 : OK 403 : Unauthorized 404 : Not found
Suppression d'un hôpital	DELETE	in : hospitalId	200 : OK 403 : Unauthorized 404 : Not found
Modification d'un hôpital	PUT/PATCH	in : hospitalId, coordinates, beds, doctors, specializations	200 : OK 403 : Unauthorized 404 : Not found
Récupération informations hopital	GET	in : hospitalId	200 : OK 403 : Unauthorized 404 : Not found

Service Docteur

Description	Method	Entry	Expected HTTP status
Ajout d'un docteur	POST	in : doctorId, hospital	200 : OK 403 : Unauthorized 404 : Not found
Suppression d'un docteur	DELETE	in : hospitalId	200 : OK 403 : Unauthorized 404 : Not found
Modification d'un docteur	PUT/PATCH	in : doctorId, hospital	200 : OK 403 : Unauthorized 404 : Not found
Récupération informations docteur	GET	in : doctorId	200 : OK 403 : Unauthorized 404 : Not found

Service Rendez-vous

Description	Method	Entry	Expected HTTP status
Création d'un rendez-vous	POST	in : rdvId, patientId, doctorId, date	200 : OK 403 : Unauthorized 404 : Not found
Modification d'un rendez-vous	PUT/PATCH	in : rdvId, hospitalId, doctorId, patientId, date	200 : OK 403 : Unauthorized 404 : Not found
Récupération informations rendez-vous	GET	in : rdvId	200 : OK 403 : Unauthorized 404 : Not found

Service Urgence

Description	Method	Entry	Expected HTTP status
Création d'une urgence	POST	in : emergencyId, patientId	200 : OK 404 : Not found

Récupération informations urgence	GET	in : emergencyId	200 : OK 403 : Unauthorized 404 : Not found

Tests unitaires

Service	Tests unitaires implémentés
Service Patient	Test que le service est bien créé; Test de la méthode findById : - quand le patient retourné est correct; - quand le patient retourné est incorrect; Test de la méthode findAll : - quand ils y a plusieurs patients retournés; - quand il n'y a pas de patient; Test de la méthode add;
Service Hôpital	Test que le service est bien créé; Test de la méthode findById : - quand le patient retourné est correct; - quand le patient retourné est incorrect; Test de la méthode findAll : - quand ils y a plusieurs patients retournés; - quand il n'y a pas de patient; Test de la méthode add;
Service Docteur	Test que le service est bien créé; Test de la méthode findById : - quand le docteur retourné est correct; - quand le docteur retourné est incorrect; Test de la méthode findAll : - quand ils y a plusieurs docteurs retournés; - quand il n'y a pas de docteur ; Test de la méthode add;
Service Rendez-vous	Test que le service est bien créé; Test de la méthode findById : - quand le rendez-vous retourné est correct; - quand le rendez-vous retourné est incorrect; Test de la méthode findAll : - quand ils y a plusieurs rendez-vous retournés; - quand il n'y a pas de rendez-vous; Test de la méthode add;
Service Urgence	Test que le service est bien créé; Test de la méthode findById : - quand l'urgence retournée est correcte; - quand l'urgence retournée est incorrecte; Test de la méthode findAll :

	- quand ils y a plusieurs urgences retournées; - quand il n'y a pas d'urgence; Test de la méthode add; Test de l'algorithme de recherche d'hôpital correspond : - lorsque l'on utilise seulement la proximité des hôpitaux avec le patient; - lorsque l'on utilise seulement la proximité des hôpitaux et que des valeurs sont nulles; - lorsque l'on utilise la proximité des hôpitaux et les lits disponibles; - lorsque l'on utilise la proximité des hôpitaux et la spécialisation requise pour l'urgence; - lorsqu'on l'on utilise la proximité des hôpitaux, les lits disponibles et la spécialisation requise pour l'urgence; - lorsqu'il n'y a aucun hôpital atteignable;

Plan de test

Test	Date	Action	Résultats attendus	Résultats réels	Réussis?
Création d'une urgence		Avec trois hôpitaux valables	Choix du plus proche		
2.		Avec trois hôpitaux, dont deux sans la bonne spécialité	Choix du troisième avec bonne spécialité		
3.		Avec 10 hôpitaux	Choix du plus proche		
Création d'un rdv		Avec un docteur et un patient et une date correcte	Création du rdv		
2		Avec un docteur, un patient et une date incorrecte	Erreur lors de la création du rdv		
3		Avec un docteur sans hôpital, un patient et une date incorrecte	Erreur lors de la création du rdv		

Tests de performance / charge

Dans le cadre de la modification de l'application, il est important de valider que la nouvelle version de l'application va répondre dans de bonnes conditions et sera capable de répondre aux sollicitations et de tenir le volume de trafic attendus.

Pour cela, il est nécessaire de définir les paramètres suivants :

- trafic journalier sur l'application
- nombre de connexion d'utilisateurs simultanés
- volume de données échangés par utilisateur
- durée moyenne d'une session
- temps moyen de visionnage d'une vidéo
- temps de réponse maximum pour une api

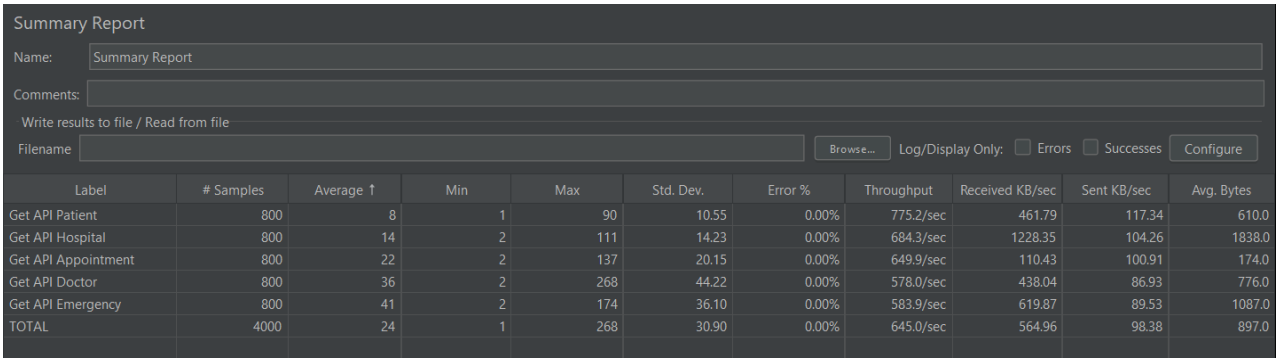
Pour effectuer les tests de charges et performance, il faut utiliser des logiciels comme JMeter, Gatling ou Funload.
On décrit par N le nombre d'utilisateurs maximum, définis par l'équipe de qualité, avec lesquels les APIs doivent continuer de fonctionner avec des performances acceptables.

Tests de charge	Résultats attendus	Résultats observés	Réussis?
1er scénario de tests Mesurer le temps d'une requête faite par un utilisateur et après on rejoue ces tests avec n utilisateurs	Multiplication du temps par n jusqu'à N		
3eme scénario de tests Simuler l'utilisation des différentes API simultanément avec n, 3n, 5n et 10n utilisateurs	Multiplication du temps par n jusqu'à N		
4ème scénario de tests Tester les limites des APIs en augmentant toutes les secondes le nombre d'utilisateurs de N jusqu'à ce que l'application ne répondent plus	L'application doit répondre jusqu'à N utilisateurs secondes		
5eme scénario de tests Tester les APIs en situation de stress en simulant l'activité maximale pendant environ n heures	Les APIs fonctionnent pendant N heures		
6eme scénario de tests Tester la robustesse du système en simulant l'activité d'un nombre important d'utilisateurs pendant une durée longue	Les APIs doivent fonctionner pendant N heures		

7eme scénario de tests Tester la résilience du système en simulant une activité légèrement supérieure à la normale couplée à la mise en panne de certains composants. Le système est censé continuer de fonctionner normalement.	Le système doit continuer de fonctionner en proposant des performances correctes	*Non testé : pas de redondance dans le cadre du POC*	

Ce tableau sera à remplir en fonction du nombre d'utilisateurs estimés et de la durée prévue d'utilisation d'un utilisateur moyen. Cependant, nous avons pris le soin d'exécuter des tests de stress dans le cadre de la preuve de concept. Voici deux exemples simples de rapport des tests de stress que nous avons effectués avec JMeter.

Test avec 800 utilisateurs par service un à un


Summary Report

Name: Summary Report

Comments:

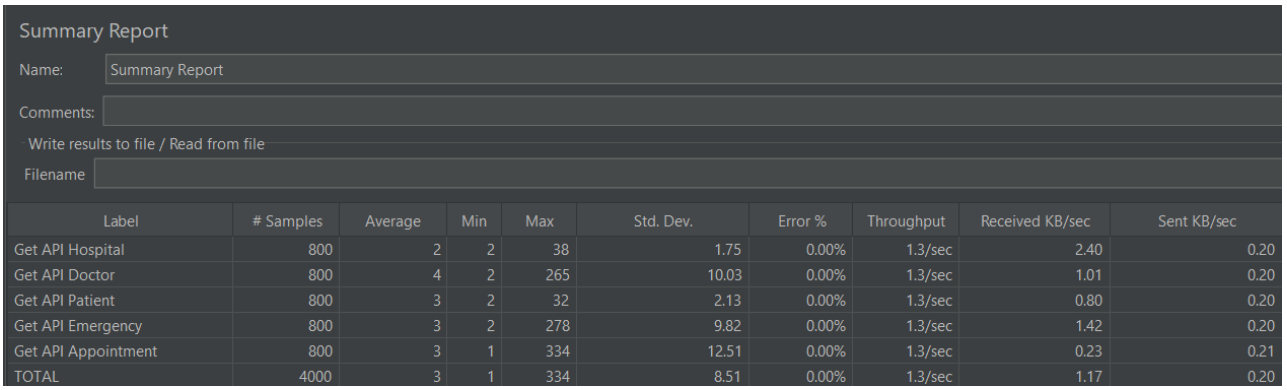
Write results to file / Read from file

Filename Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get API Patient	800	8	1	90	10.55	0.00%	775.2/sec	461.79	117.34	610.0
Get API Hospital	800	14	2	111	14.23	0.00%	684.3/sec	1228.35	104.26	1838.0
Get API Appointment	800	22	2	137	20.15	0.00%	649.9/sec	110.43	100.91	174.0
Get API Doctor	800	36	2	268	44.22	0.00%	578.0/sec	438.04	86.93	776.0
Get API Emergency	800	41	2	174	36.10	0.00%	583.9/sec	619.87	89.53	1087.0
TOTAL	4000	24	1	268	30.90	0.00%	645.0/sec	564.96	98.38	897.0

Rapport de test de stress avec 800 utilisateurs par service un à un

Test avec 800 utilisateurs par service un à un (nombre croissant sur 10 minutes)


Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec
Get API Hospital	800	2	2	38	1.75	0.00%	1.3/sec	2.40	0.20
Get API Doctor	800	4	2	265	10.03	0.00%	1.3/sec	1.01	0.20
Get API Patient	800	3	2	32	2.13	0.00%	1.3/sec	0.80	0.20
Get API Emergency	800	3	2	278	9.82	0.00%	1.3/sec	1.42	0.20
Get API Appointment	800	3	1	334	12.51	0.00%	1.3/sec	0.23	0.21
TOTAL	4000	3	1	334	8.51	0.00%	1.3/sec	1.17	0.20

Rapport de test de stress avec 800 utilisateurs par service un à un (nombre croissant sur 10 minutes)

Exemple de KPI (Key Performance Indicator)

Métrique	Valeur cible

Augmentation du temps de latence des requêtes en fonction de la charge (jusqu'à limite)	*A définir par l'équipe qualité*
Augmentation du temps de connexion en fonction de la charge (jusqu'à limite)	*A définir par l'équipe qualité*
Nombre d'erreurs dans les API en fonction de la charge (jusqu'à limite)	*A définir par l'équipe qualité*
Temps de latence entre deux tâches	A définir par l'équipe qualité

Test de résilience

Déterminer le contexte

La résilience de l'API est définie par sa résistance aux imprévus et aux arrêts de service. Afin d'être en mesure de mesurer efficacement la résilience, les paramètres suivants doivent être déterminés avec soin :

- les situations critiques pendant lesquelles les erreurs ne sont pas acceptées (par exemple, gestion d'urgence);
- la disponibilité et la charge attendue;
- le RTO et le RPO, qui interviendront notamment sur le choix de la stratégie de back-up;
- la définition, dans le contexte de notre application, d'une catastrophe (arrêt de service primordial, corruption de données utilisateur..);

Ces différents paramètres pourront justifier une attention particulière sur certains cas d'erreurs, certains composants de l'architecture ou certaines stratégies de recouvrement.

Composants critiques

Afin d'assurer la performance du système, des tests de résilience et des analyses de l'architecture doivent être effectuées. Il est important tout d'abord de repérer les composants critiques. Nous citerons notamment :

- les bases de données;
- le SSO et le LDAP;
- l'API Gateway;
- le front-end;

Dans le cadre de l'API Medhead, nous utiliserons également des données fournies par des APIs externes. Certaines de ces APIs sont également considérées comme critiques car le

bon fonctionnement de notre système dépendra de données fournies en temps réel par celles-ci.

Résilience architecturale

Une grande partie de la résilience d'un système provient de son architecture. Le tableau ci-dessous décrit différentes problématiques de résiliences qui peuvent exister, et les réponses architecturales que nous mettrons en place pour les éviter.

Problématique de résilience	Solution architecturale
Arrêt de service d'un des composants critiques	Mise en place d'une redondance des composants critiques
Excès de charge sur un service	Utilisation d'un équilibreur de charge et d'un orchestrateur
Disponibilité réduite	Mise en place de SLA, utilisation d'un orchestrateur
Corruption des données et respect du RTO/RPO	Mettre en place des sauvegardes automatiques des données Valider régulièrement les données entrantes Gérer la cohérence des données lors de fail-over Mettre en place un plan de recouvrement des données

Mise en place de tests de résilience

Pour tester la résilience du système et mettre en place des solutions pour faire face à ces problèmes, différents méthodologies de tests doivent être mises en place :

- Effectuer des tests par injection de panne et vérifier le temps de récupération;
- Mettre le système sous stress et identifier les problèmes survenant dans ces cas précis;
- Effectuer régulièrement le plan de récupération après sinistre;
- Effectuer des tests de fail-over;
- Configurer des sondes d'état, notamment pour les équilibreurs de charge et gestionnaires de trafic;
- Mettre en place des rapports fiables et clairs pour les systèmes de surveillance;
- Inclure les services tiers dans les différents tests;

Test de sécurité

Composant	Tests	Résultat attendu	Résultat

API Gateway	Envoie de requêtes HTTP frauduleuses	Interception de la requête par le composant	
1	DDoS	Implémentation des mesures anti DDoS (blocage des IP, augmentation du throttle..)	
Bases de données	Insertion de données non valables dans la base	Échec avec message d'erreur	
Accès aux services	Accès aux services en fonction du rôle	Chaque utilisateur n'a accès que aux fonctionnalités de son rôle	
Accès réseaux	Modifier les accès au réseau avec plusieurs utilisateurs	Seul les responsables réseaux sont autorisés	
Accès extérieurs	Tests des différentes failles connues via les protocoles (SMTP, SQL, RDP..)	Refus des accès	

Des tests de sécurité, et plus particulièrement de protection des données doivent être mis en place annuellement. La RGPD et plus généralement les normes de sécurité relatives à la nature sensible du domaine métier doivent être éprouvées et validées.

Des audits de sécurité doivent être planifiés et les tests de sécurité O'Wasp doivent être mis en place et validés à 100% avant la mise en production du système.

Test de contrat

Les tests de contrat consistent à tester l'application en mode "boite noire", c'est-à-dire en ignorant le fonctionnement technique des fonctionnalités. Le but est de simplement tester et vérifier le fonctionnement attendu, en ignorant complètement la manière dont les composants ont été développés.