Exploration des méthodologies de test d'applications

Note de veille technologique

Cyprien PIERRE

2025-03-20

Résumé

Rédiger l'abstract

Mots clés : Visualisation de données, Graphique, Méthode

Table des matières

Introduction

Vue d'ensemble

Contexte : Architecture microservice, Vue.js, PostgreSQL Type d'application : Aggrégation et analyse de données

Stratégies

Approches théorisées : Test Driven Design, Behavior Driven Design
Quoi, Qui, Où, Quand, Comment, Pourquoi ?
Objectif : the shortest feedback loop possible
Continuous integrations rules

1. Always run commit tests locally before commiting
2. After commiting, always wait for the results of the testing pipeline
3. Fix or reverse failures within 10 minutes (start a clock, pomodoro)
4. Revert changes done by a teamate if it breaks the rules
5. Monitore your change throught the pipeline
6. Treat a broken build as a build sin (gamifying things)
7. Commit small but often
8. Maintain a fast commit feedback loop
9. Automate everything
10. Take ownership of failures

Test Driven Development => Design

- Path :
  - Express intent in the form of code
  - Test the test by running it and seeing it FAIL
  - Create the minimum code to meet the needs of the tests
  - Run it and see it PASS
  - In a stable and passing state, REFACTOR test & code for quality & generality

Tests fonctionnels

Test unitaire Valide le fonctionnement d'une fonction unique. Est très performant. Est le niveau le plus précis des types de tests. Est exécuté dans un pipeline d'intégration continue.

Test de composant
Test d'intégration
Test de gestion des erreurs

Non régression

Test de contrat

Contract Driven Testing => Specmatics & YAML

- Contract compatibility testing
- Backward compatibility
- Shared contracts specifications
- Zero code contract testing

Test d'acceptation Est rédigé avant de coder une fonctionnalité. Spécifie uniquement ce que le système est censé faire. Ce type de test est le premier pillier du TDD. Son exécution est consommatrice de ressource, ces tests se concentrent sur les comportements généraux et sont à voir comme une confirmation complémentaire aux tests unitaires. Ils sont utilisés pour initialisés et valider le développement d'une feature, pour vérifier la stabilité du système au fil des évolutions et permet de remplacer les tests manuels de regression.

Test d'approbation Vérifie qu'un code produit le même résultat entre deux exécutions. Est intéressant pour s'assurer de la stabilité d'un service et de la répétabilité de composition d'une interface.

Expérience utilisateur

Test de comportement Niveau le plus macroscopique des types de test, il vise à définir des spécifications (fonctionnalités) et des scénarions (user story) dans un langage proche du langage naturel (eg. Gherkins). Ces descriptions sont converties en code et éxecutées par des outils comme Cucumber.

Test d'interface Permet de tester les schémas de navigations dans l'application et de vérifier le comportement des éléments d'interface. Des technologies comme Cypress prennent en charge ces types de tests.

Test utilisateur Exploration manuelle de l'application client par un utilisateur "beta testeur".

Ces modes de tests et les technologies citées (Cypress et Cucumber) peuvent être utilisées conjointement pour permettre à la fois l'automatisation des tests, la collaboration des équipes diverses et l'expérimentation manuelle.

Performance

Définir des seuils maximums de temps de réponse Pour assurer une bonne expérience utilisateur Pour limiter la consommation des ressources et les frais de computing en serverless Pour limiter les goulots d'étranglements

Test de résistance Pour identifier la charge maximale admissible par le système. Monter progressivement la charge jusqu'à faire tomber les services.

Test d'endurance Maintiens de la charge maximum sur de longue période (de plusieurs heures à plusieurs jours) pour s'assurer de la stabilité des services.

Test de pointes Vérification de la capaciter à traiter des pics soudains de charge. A tester à 100%, 125%, 150% et 200% de capacité. Est associé à une analyse des probabilités d'occurences pour le calcul du taux de disponnibilité.

Test des temps de réponses

Barely equal to a benchmark

Sécurité

Test d'autenticité Test des mécanismes d'autentifications de l'API

Test d'autorisation chaque utilisateur ne doit accéder qu'à ses propres données (sécurité granulaire au niveau des lignes).

Test de pénétration "Simule une cyberattaque pour identifier les vulnérabilités de l'API qu'un attaquant potentiel pourrait exploiter. Ces tests sont effectués à l'aide d'algorithmes spécialisés qui recherchent les injections de code et arrêtent ces requêtes avant qu'elles ne puissent causer des dommages au serveur."

Dépendancy injection

[…]
Prérequis

<=====DATA=====>

Domain Driven Development => Code is the simulation of the problem we try to solve
idea : org-mode documents that describe behaviors & business logic ?

- End-to-end testing
- Behavior Driven Design => Cucumber, SpecFlow & Gherkins Language
  - Specifications + Senarios
  => Functionnal testing
- Integration testing => Not really neaded if acceptance test is well defined
  - Here to fail faster

Others

- Approval test
  - Verify the repetability of the code through the results
  - Valuable for stability
  - Great for UI stability
  - Verify that nothing has changed
- Manual testing
  - Use humans for exploratory and usability testing (human in the loop)
  - Exploratory test
- Soak test
- Sanity test
- Smoke test

Financial costs : Resource and network consumption Energy efficiency, carbon footprint

---

DORA metrics => Research papers + the "Accelerate" book (sociologic)

- Stability `is a measure of the quality of the code that we produce` https://www.youtube.com/watch?v=hbeyCECbLhk&list=PLwLLcwQlnXBwvH8Iqs9zqkbSWdvWoyX4v
  - Change failure rate => Bug count
  - Mean time to recovery => Bug count
- Throughput `is a measure of the efficiency of your approach` => Work on smaller changes
  - Lead time for changes => Features
  - Deployment frequency [x]

Database testing :

- Schema testting : test de la logique structurelle de la base (procedures, functions, tables columns…).
- Functionnal testing : test de l'intégrité des données (clefs uniques, suppression en cascade …)
- Non functionnal testing : tests de performances et de sécurité
- Unit testing : test des fonctionnalités individuelles.
- Routing testing : test des différentes routes et de leurs réponses.
- Client testing
- Component testing : test du fonctionnement d'un composant.
- E2E : test de l'intégration des composants entre eux.
- Other services testing :

Faire attention à ne tester que la logique implémentée en interne. On ne teste pas que pg sait bien faire un select ou une librairie externe.
Ce que l'on veut tester dans Rail_on at the moment :
Database : Que le schéma d'une base migré soit bien celui qui est typé dans typeORM. Que les cascades et les clefs uniques et les relations sont ce qui est attendu API : On veut tester les routes et ce qu'elles retournent On veut tester les fonctions communes utilisées dans les middlewares Client : TO BE DEFINED