Test logiciel

None

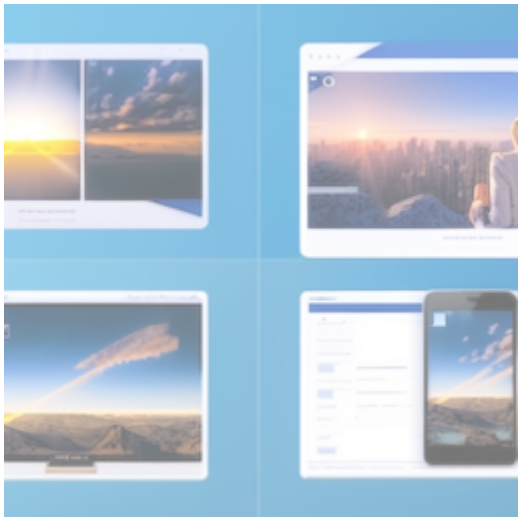
None

None

Table of contents

1. Bienvenue au cours: Test logiciel

made with mkdocs material



1.1 Ressources et liens utiles

- Software testing par IBM
- Test et Validation du Logiciel (LaBRI)

1.2 License

- Ce support de cours est disponible sous licence Creative Commons Attribution 4.0 International (CC BY 4.0):
  License CC BY 4.0
- Logo généré par https://hotpot.ai/

2. Généralités

2.1 Définition

IBM

Le test logiciel est le processus qui consiste à évaluer et à vérifier qu'un produit ou une application logicielle fait ce qu'il ou elle est censé(e) faire.

Les avantages des tests, toujours selon IBM

Les avantages du test comprennent la prévention des bogues, la réduction des coûts de développement et l'amélioration des performances.

2.2 Classification des tests

Il est compliqué de trouver une classification universelle des tests logicielles. Dans la suite, je compile quelques classifications proposées par différentes sources.

2.2.1 Différents types de tests

Source atlassian

Voici quelques types de tests sans classification particulière:

- Tests unitaires
- Tests d'intégration
- Tests fonctionnels
- Tests de bout en bout
- Tests d'acceptation
- Tests de performance
- Smoke tests

2.2.2 Pyramide des tests par niveau

Source all4test

- Les tests unitaires
- Les tests d’intégration
- Les tests systèmes
- Les tests d’acceptation

2.2.3 Classification selon la nature

Source all4test

- Tests fonctionnels
- Tests non fonctionnels, quelques exemples:
- Les tests de robustesse
- Les tests de performance
- Les tests de montée en charge
- Les tests de compatibilité de plateforme
- Les tests d’ergonomie
- Les tests d’interface graphique
- Les tests de sécurité

2.2.4 Tests manuels ou automatisés ?

Source atlassian

Les tests manuels sont effectués par des humains tandis que les tests automatisées sont effectuées par un logiciel.

3. Quelques librairies de tests

SQLite est réputé pour avoir une proportion importante de tests.

3.1 JUnit

Framework de test Java permet de tester son code via des assertions. Explorons le projet de démarrage fourni par le guide officiel

3.1.1 Exercices

1. Ecrire un test unitaire `StringTest` qui test ces méthodes de la classe `String` : `toUpperCase()`, `toLowerCase()` et `charAt()`.
2. Exo 2 de cette série
   - Ignorer les CE invalides b5 et b6
3. Exercice 2 de cette série qui nécessite d'abord de faire d'abord ces exercices: partie1, partie2
4. Cet exo de librecours.net. Le code est à traduire du JS vers du Java.
5. Cet exercice de librecours.net. Le code est à traduire du JS vers du Java.
6. Ce TD qui provient de labri. Le `makefile` est l'équivalent de `maven` pour nous. Faire la question 3 avec cette commande à la place de l'outil proposé `mvn surefire-report:report`.

> **elques**
>
> 1. RàS
> 2. Fonctions corriger / modifier: `checkDay`, `checkYear`, `checkDayLimits`, renommer ou enlever `maxDayOfMonth(month)`, `testFebruaryValidLimitDates`, `testValidLimitDates`

3.2 Tests dans un projet Spring MVC

3.2.1 API REST sans base de données

- Générer un projet Spring avec initializr, en choisissant les dépendances suivantes: Spring Web et Spring Boot DevTools.
- Ouvrir le projet sur VSCode
- Créer un `@RestController` avec deux routes en @GET et en @POST, un modèle et un service qui gère une liste statique en mémoire.
- Je vous propose d'utiliser comme modèle une classe `Manga` avec trois champs `ISBN`, `name` et `nbPages`
- Lancer votre serveur et vérifier qu'il fonctionne avec la bonne commande (avec gradle `./gradlew bootRun`, avec maven `./mvnw springboot:run`) ou depuis votre IDE
- Ecrire des tests unitaires pour le service
- Spring propose deux façons de tester le contrôleur (en d'autres termes l'API REST).
- En lançant un serveur web (avec la stack HTTP complète) via la classe `TestRestTemplate`
- En lançant un serveur bouchonné (on n'a pas la stack HTTP complète) via la classe `MockMVC`
- Ecrire des tests pour le contrôleur

3.2.2 API REST avec une base de données

Nous allons utiliser la BDD H2 pour sa simplicité car c'est une BDD relationnelle (SQL) qui ne nécessite pas de serveur et réside en mémoire (RAM) par défaut.

- Générer un projet Spring avec le créateur en ligne en choisissant les dépendances suivantes: Spring Data JPA, H2 Database, Spring Boot DevTool et Spring Web.
- Ouvrir le projet sur VSCode
- Créer une classe "Model" avec l'annotation `@Entity` et les annotations `@Id` et `@GeneratedValue` sur sa clé primaire.
- Créer une interface `xxxxRepository: JpaRepository<Product, Long>` où xxx est le nom de votre modèle
- Créer un contrôleur avec les routes en @GET, @POST, PUT et DELETE, un modèle et un service qui gère communique avec votre repository

3.2.3 Liens et références

- http://deptinfo.cnam.fr/~graffion/UES/GLG101/tps/java/index.html
- https://gayerie.dev/docs/testing/test/junit.html
- https://github.com/mjeanroy/exercices-java/blob/master/exercices-junit.txt

4. Projet

Choisir un sujet et réaliser une présentation abordant les points suivants:

- définition
- Solutions alternatives ou similaires
- avantages / inconvénients
- Démo en direct ou enregistrée sinon (la démo en direct sera préférée)

Sujets:

- TDD
- Les 3 mieux notés Actions GitHub d'intégration avec Discord et leur utilité (ou non) dans le cadre des tests
- Outil testquality et intégration avec GitHub
- appium
- JIRA vs YouTrack
- SonarQube et son utilité pour les tests
- Tests de performance
- TestContainers