République Algérienne Démocratique Et Populaire

Ministère De L'Enseignement Supérieur Et De La Recherche Scientifique

Université Saad Dahlab DE BLIDA 1

Mémoire Présenté En Vue De L'obtention Du Diplôme De

***Master* En Informatique**

Spécialité : Ingénierie de Logiciel(IL)

Thème :

Mise en place d'une Plateforme de test fonctionnel pour les applications d'aide à la décision : Cas d'un Tableau de Bord

Présenter par :

ANDOH MICHAEL

MOHAMED MOHAMED YAHYA

Membre du jury :

M. HACENE DERRAR Encadreur

Mme. CHIKHI IMANE Président

Mme. BOUMAHDI FATIMA Examinatrice

Année universitaire
2018/2019

Nous tenons tout d'abord à remercier le Bon Dieu Tout Puissant de nous avoir aidés à réaliser ce modeste travail.

Nous le dédions également à toutes nos familles respectives : parents et amis…

Aussi, nous adressons un grand merci à l'endroit de notre aimable encadreur Mr.Darrar pour avoir accepté de nous encadrer, pour l'aide qu'il nous a apporté pour la confiance qu'il nous a accordé et pour les conseils et les informations qu'il nous a prodigué. Il n'a ménagé aucun effort afin que nous puissions juguler tous les écueils qui s'étaient dressés devant nous dans l'aboutissement de notre projet. Qu'il trouve ici l'expression de notre meilleur sentiment de respect et de reconnaissance.

Nous tenons également à, et c'est le lieu idoine, remercier l'ensemble du corps professoral du département d'informatique

De Blida 1 spécialité : Ingénierie de logiciel.

Un grand merci également à tous nos amis (algériens et étrangers respectifs) de la promotion 2018/2019.

Enfin, nos plus chaleureux remerciements pour toute personne ayant participée de près ou de loin à la réalisation de ce Modeste travail.

ADON MICHEAL&MOHAMED MOHAMED YAHYA.

Table des matières

Liste des Tableaux

Liste des Figures

Résumé :

Dans nos jours l'informatique est devenu le moyen le plus important utilisé dans chaque domaine sophistiqué, le développement des applications et logiciels sont en constante évolution. Les entreprises pensent alors à gagner en temps et en performances afin d'automatiser la procédure de tests logiciels, notre travail a pour but de concevoir une application de test logiciel cette application est basé sur les tests fonctionnels, et de gérer la gestion automatique de cette application en utilisant un outil d'automatisation de test fonctionnel(Sélénium).

الملخص:

في علوم الكمبيوتر اليوم أصبحت أهم الوسائل المستخدمة في كل مجال متطور، وتطوير التطبيقات والبرامج تتغير باستمرار. تفكر الشركات بعد ذلك في اكتساب الوقت والأداء لتمت عملية اختبار البرمجيات، يهدف عملنا إلى تصميم تطبيق اختبار البرنامج الذي يستند إلى اختبار وظيفي، وإدارة الإدارة التلقائية لهذا تطبيق باستخدام أداة التشغيل الآلي للاختبار الوظيفي.(Selenium)

Abstract:

In today's computer science has become the most important means used in every sophisticated field, the development of applications and software are constantly changing. Companies then think of gaining time and performance to automate the software testing process, our work aims to design a software test application this application based on functional testing, and manage the automatic management of this application using a functional test automation tool (Selenium).

Introduction Générale

1. Contexte :

Notre époque d'aujourd'hui se caractérise par une utilisation indispensable des nouvelles technologies dans tous les secteurs ou corps de métier. Dans une économie de plus en plus globalisée, sophistiquée et potentiellement concurrentielle, les entreprises qu'elles soient multinationales ou des PME (petite et moyenne entreprises) se livrent une concurrence farouche pour la conquête de leur part de marché. Ce qui nécessite la mise en place d'outils de gestion et d'aide à la décision efficaces afin d'avoir un système managérial efficient.

Les outils d'analyses des données et de reporting sont devenus de plus en plus demandés par les entreprises pour l'extraction de connaissance en vue de l'aide à la décision. Le reporting est probablement l'application la plus utilisée de l'informatique décisionnelle cela permet aux gestionnaires de :

- Sélectionner des données relatives à telle période, telle production, tel secteur de clientèle ;
- Réaliser des statistiques à travers divers calculs (totaux, moyennes, écarts, comparatifs d'une période à autre) ;
- Représenter le résultat d'une manière synthétique ou détaillée, le plus souvent graphique selon leurs besoins ou les attentes des dirigeants de l'entreprise.

Au regard de ce qui est susmentionné, il va sans dire qu'aucune entreprise digne de ce nom ne saurait donc se départir de l'utilisation de l'informatique et des nouvelles technologies dans leur stratégie de management. Ce recours massif à l'utilisation de ces outils d'analyse et d'aide à la décision nécessite pour qu'ils soient fiables de les tester et de s'assurer que les résultats obtenus ne sont pas erronées. L'erreur dans ce contexte peut engendrer des conséquences graves pour l'entreprise. C'est dans ce domaines précis que s'inscrit notre projet de fin d'études qui consiste à concevoir une plateforme de teste automatiques des

Applications d'aide à la décision est plus précisément celles se basant sur les tableaux de bord.

2. Problématique :

L'analyse prescriptive, descriptive ou prédictive nécessitent d'abord un volume de donnée important et préalablement nettoyé et traité afin de rechercher respectivement des alternatives de choix à présenter aux décideurs, ou trouver de structure ou des modèles qui lient les données pour comprendre leurs interactions par des applications des méthodes issues du domaine des statistiques ou enfin l'application de méthode de data ming pour extraire des connaissances afin de prédire des évènements futurs.

Aussi, il est impératif que l'utilisateur sache comment mieux exploiter toutes les méthodes d'extraction de l'information, car s'il introduit une erreur dans sa requête cela va forcément impacter le résultat dans l'affichage.

Notre travail consiste à développer un outil d'automatisation des tests fonctionnels appliquée sur les tableaux de bord d'aide à la décision. L'objectif recherché c'est d'abord de s'assurer que les tableaux de bords générés à partir des données conçues pour l'aide de décision ne renferment des erreurs en termes de besoins fonctionnels et répondent aux spécifications pour lesquelles elles ont été développées

3. Objectifs : les objectifs visés par ce projet sont les suivants :

-Automatiser les tests effectués.

-Améliorer la performance des tests.

- S'assurer de la pertinence des résultats de tests effectués.

- Diminuer le temps d'exécution des tests.

4. L’organisation du mémoire

Dans le but de faciliter la lecture de ce mémoire, nous l’avons structuré en quatre (4) chapitres :

Chapitre -I- Etat de l’Art de test logiciel : Dans cette partie, nous avons explicité le test tout en montrant ses différentes typologies. Puisque les tests sont multiples, nous n’avons pris que quelques types de tests : test fonctionnel, test de régression, smoke test…

Chapitre -II- Automatisation de Test fonctionnel : ce chapitre comporte les outils utilisés dans l’automatisation des tests fonctionnels. Nous avons ensuite comparés les méthodes de test : manuel et automatique tout en mentionnant leurs avantages et inconvénients via un tableau comparatif.

Chapitre -III- La Conception de notre système : Dans ce chapitre il est essentiellement question de modélisation de notre outil.

Chapitre – IV- Implémentation et Test : Ici, nous avons implémenté les outils de notre système et avons testés les résultats obtenus.

Conclusion Générale : Nous avons terminé notre travail par une conclusion générale dans laquelle nous mentionnons le but et la particularité de notre travail et les objectifs atteints.

Chapitre I :

Etat de l'Art de Test Logiciel

Introduction : Dans ce chapitre nous présentons d'une manière générale les tests logiciels C'est quoi les tests, leurs différents types et leurs méthodes existant pour qu'on puisse effectuer ces tests, en plus nous dévons aussi introduire les notions d'un logiciel comment est-il définit leurs qualités et ça cycle de vie.

1. Définition d'un logiciel

Un ensemble plus ou moins complexe d'instructions, écrites dans un langage de programmation (qui peut être un langage compilé ou un langage interprété) ayant vocation à être exécutées par un processeur, c'est-à-dire par un ordinateur, un terminal mobile (téléphone, tablette, etc.) ou par des dispositifs plus rudimentaires ou plus évolués (lecteur de carte à puce, robot, objet connecté tel qu'enceinte intelligente, etc.). Il existe de nombreuses typologies de logiciels, selon qu'on se place d'un point de vue technique, juridique, fonctionnel, etc. :

Logiciel serveur ou logiciel client,

Micro-logiciel (firmware),

Logiciel spécifique (développé par ou pour une entreprise utilisatrice donnée) ou progiciel (logiciel standard destiné à répondre aux besoins communs de nombreux utilisateurs),

Logiciel de gestion (logiciel de comptabilité, de paie, logiciel de gestion commerciale ou CRP, logiciel ERP ou PGI, SCM, WMS...),

Application mobile,

Logiciel propriétaire ou logiciel libre (dont le code source est publié et/ou librement modifiable), etc. [2]

2. Cycle de vie d’un logiciel

Le « cycle de vie d'un logiciel » : désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre.

L'origine de ce découpage provient du constat que les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation. Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés. [6]

Il existe deux différents types de catégorie de cycle de vie d’un logiciel une en « V » et l’autre en «Cascade ».

2.1. Modèle en cascade : Le modèle de cycle de vie en cascade a été mis au point dès 1966, puis formalisé aux alentours de 1970. Il définit des phases séquentielles à l'issue de chacune desquelles des documents sont produits pour en vérifier la conformité avant de passer à la suivante : comme montre la figure ci-dessous.

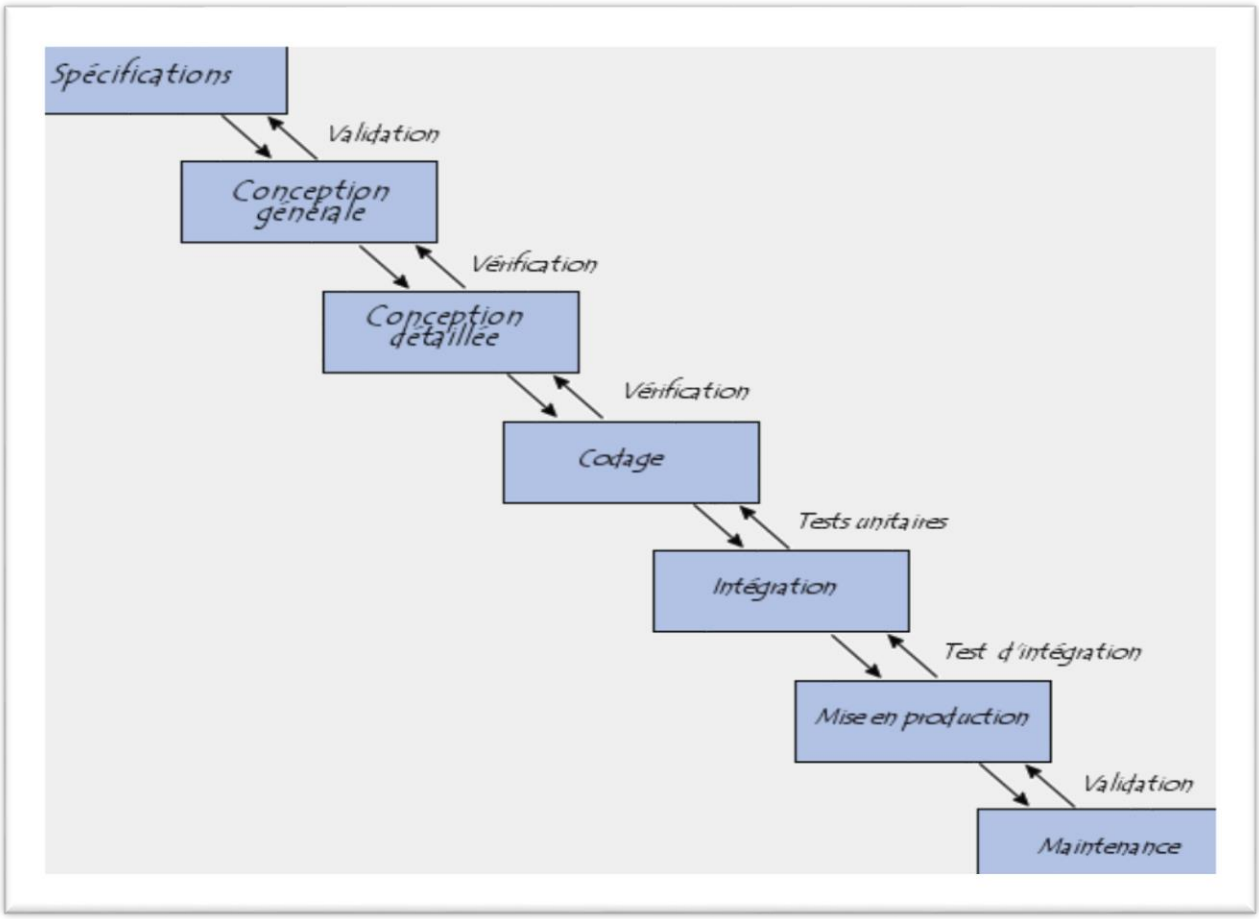


Figure I. 1 : Modèle en Cascade [22].

2.2. Modèle en V : Le modèle de cycle de vie en V part du principe que les procédures de vérification de la conformité du logiciel aux spécifications doivent être élaborées dès les phases de conception.

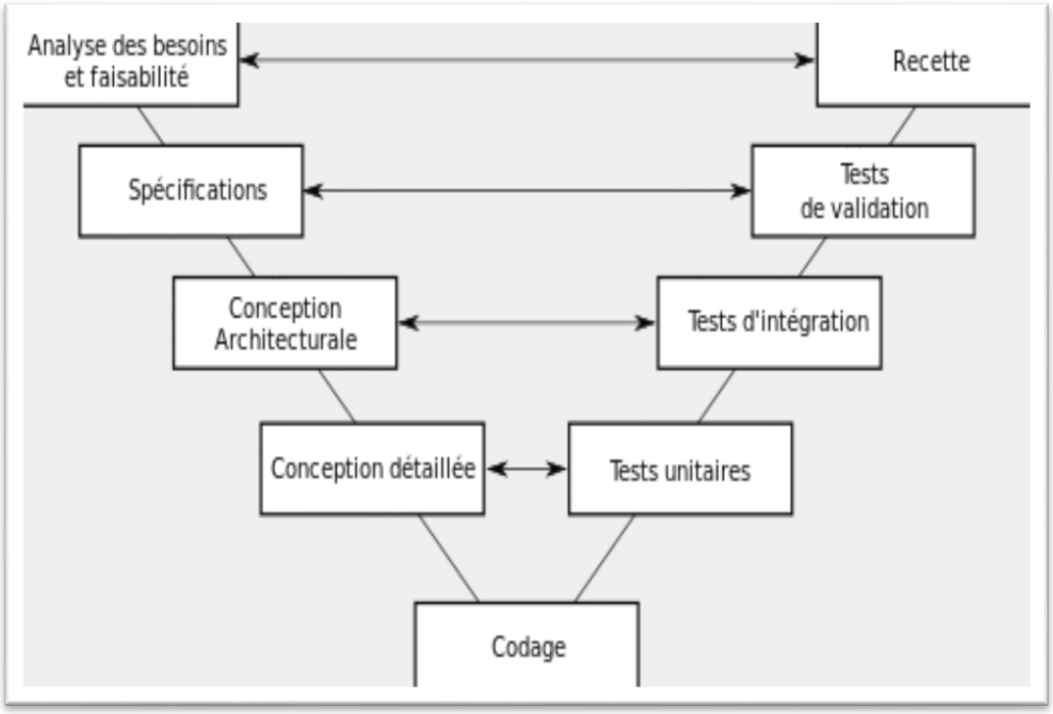


Figure I.2 : Modèle en V [23].

Le cycle en V est un cycle composé de 3 grandes phases contenant 8 étapes de conception d'un produit :

- **La phase de conception**
  - Analyse des besoins et faisabilité : c'est-à-dire l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes.
  - Spécifications
  - Conception Architecturale : Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel.
  - Conception détaillée : consistant à définir précisément chaque sous-ensemble du logiciel.
- **La phase de réalisation**
  - Codage : (Implémentation ou programmation), soit la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.

- Tests unitaires : permettant de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.

❖ **La phase de validation**

- Tests d'intégration : dont l'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de tests d'intégration consignés dans un document.
- Test de validation
- Recette : c'est-à-dire la vérification de la conformité du logiciel aux spécifications initiales.

❖ **La phase de conception** : Le client et le prestataire définissent un cahier des charges détaillant toutes les fonctionnalités recherchées dans le produit final avant la conception du produit final. Le cahier des charges est un document indispensable dans un projet car il lie le client et le prestataire sous un contrat.

Le prestataire détermine les spécifications techniques de la solution dans la technologie choisie, il effectue une estimation de délai pour chaque fonctionnalité à développer.

La phase de réalisation : L'équipe du projet se lance dans le développement du produit sur la base des spécifications techniques. Cette phase du projet consiste à concevoir les différentes fonctionnalités du produit final dans les délais attendus. A chaque fois qu'un composant logiciel est développé, il est testé afin de déterminer s'il fonctionne correctement.

La phase de validation : Les composants logiciels sont intégrés dans la solution finale pour vérifier que l'intégration ne provoque pas d'anomalies. Le produit est ensuite testé au regard des spécifications fonctionnelles.

Une dernière validation du produit est effectuée avant la mise en production. Suite à la mise en production, le client vérifie, dans le cadre de la recette, que le produit correspond bien à ses attentes.

3. Définition d'un test logiciel

Le test est une discipline vaste qui permet de s'assurer de la qualité des logiciels. Cette qualité comprend différentes dimensions que l'on peut regrouper dans les trois catégories suivantes : les fonctionnalités, l'ingénierie et l'adaptabilité : [6]

- ✓ **Fonctionnalités** :
  - Exactitude
  - Fiabilité
  - Utilisabilité
  - Intégrité
- ✓ **Ingénierie** :
  - Efficacité
  - Testabilité
  - Documentation
  - Structure
- ✓ **Adaptabilité** :
  - Flexibilité
  - Réutilisabilité
  - Maintenabilité
  - Montée en charge

4. Les différents types de test logiciel

- **Test fonctionnel** : est un type de test logiciel par lequel le système est testé par rapport aux exigences / spécifications fonctionnelles.

Les fonctions (ou fonctionnalités) sont testées en leur fournissant une entrée et en examinant la sortie. Les tests fonctionnels permettent de s'assurer que l'application répond bien aux exigences. Ce type de test ne concerne pas la façon dont le traitement se déroule, mais plutôt les résultats du traitement. Il simule l'utilisation réelle du système mais ne fait aucune hypothèse de structure système.

Lors des tests fonctionnels, la technique Black Box Testing est utilisée dans laquelle la logique interne du système testé n'est pas connue du testeur.

Les tests fonctionnels sont normalement effectués aux niveaux de test du système et de test d'acceptation.

En règle générale, les tests fonctionnels comportent les étapes suivantes :

- Identifiez les fonctions que le logiciel est censé exécuter.
- Créez des données d'entrée basées sur les spécifications de la fonction.
- Déterminez la sortie en fonction des spécifications de la fonction.
- Exécutez le scénario de test.
- Comparez les résultats réels et attendus.

Les tests fonctionnels sont plus efficaces lorsque les conditions de test sont créées directement à partir des besoins de l'utilisateur / de l'entreprise. Lorsque les conditions de test sont créées à partir de la documentation système (spécifications système / documents de conception), les défauts de cette documentation ne sont pas détectés lors des tests, ce qui peut être la cause de la colère des utilisateurs finaux lors de l'utilisation finale du logiciel. [3]

- **Tests de bout en bout** : Reproduisent le comportement d'un utilisateur avec le logiciel dans un environnement applicatif complet. Ils vérifient que les différents flux d'utilisateurs fonctionnent comme prévu et peuvent être aussi simples que le chargement d'une page Web ou la connexion. Des scénarios beaucoup plus complexes peuvent aussi vérifier les notifications

par email, les paiements en ligne, etc. Les tests de bout en bout sont très utiles, mais ils sont coûteux à réaliser et peuvent être difficiles à gérer lorsqu'ils sont automatisés. Il est recommandé d'avoir quelques tests clés de bout en bout et de s'appuyer davantage sur des tests de niveau inférieur (tests unitaires et d'intégration) pour être en mesure d'identifier rapidement les changements de dernière minute. [1]

- **Smoke tests** : Les smokes tests sont des tests simples qui vérifient le fonctionnement de base de l'application. Ils sont conçus pour être rapides à exécuter, et leur but est de vous donner l'assurance que les caractéristiques principales de votre système fonctionnent comme prévu. [1]
- **Test de régression** : est un type de test de logiciel visant à garantir que les modifications (améliorations ou corrections d'erreur) apportées au logiciel ne l'ont pas affecté de manière préjudiciable. [3]

La probabilité qu'un changement de code impacte des fonctionnalités qui ne sont pas directement associées au code est toujours présente et il est essentiel que des tests de régression soient effectués pour s'assurer que la correction d'une chose n'a pas cassé autre chose.

Pendant les tests de régression, les nouveaux scénarios de test ne sont pas créés, mais les scénarios de test précédemment créés sont ré-exécutés.

Régression [nom] signifie littéralement l'acte de revenir à un lieu ou à un état antérieur ; retour ou retour.

- ✓ **Niveaux**

Les tests de régression peuvent être effectués à tout niveau de test (unité, intégration, système ou acceptation), mais ils sont principalement pertinents lors des tests du système.

- ✓ **Ampleur**

Dans un cas idéal, un test de régression complet est souhaitable, mais il existe souvent des contraintes de temps / ressources. Dans ce cas, il est essentiel de procéder à une analyse d'impact des modifications pour identifier les zones du logiciel les plus susceptibles d'être affectées par ces modifications et celles des utilisateurs en cas de dysfonctionnement, ainsi que les tests ciblés sur ces zones. .

En raison de l'ampleur et de l'importance des tests de régression, de plus en plus d'entreprises et de projets adoptent des outils d'automatisation des tests de régression.

- ❖ **Test d'utilisation** : est un type de test effectué du point de vue de l'utilisateur final pour déterminer si le système est facilement utilisable.

✓ **Définition 01 : ISTQB**

Tests d'utilisabilité : tests permettant de déterminer dans quelle mesure le produit logiciel est compris, facile à apprendre, facile à utiliser et attrayant pour les utilisateurs dans des conditions spécifiées.

✓ **CSTE CBOK Définition 02 :**

- ❖ **Test d'utilisabilité** : Cet événement a pour objectif de passer en revue l'interface utilisateur de l'application et les autres facteurs humains de l'application avec les personnes qui l'utiliseront. Cela permet de garantir que la conception (mise en page, séquence, etc.) permet aux fonctions commerciales d'être exécutées aussi facilement et intuitivement que possible.
- ❖ **Le test non-fonctionnel** : Les tests non-fonctionnels vérifient des propriétés qui ne sont pas directement liées à une utilisation du code. Il s'agit de vérifier des caractéristiques telles que : Test de performance/ Test de sécurité/ Test de fumée (Smoke) / Test de stress
- ❖ **Test de sécurité** : est un type de test logiciel visant à découvrir les vulnérabilités du système et à déterminer que ses données et ses ressources sont protégées contre les intrusions éventuelles. [3]

Il prend en compte les domaines suivants :

- ✓ **Sécurité réseau** : Cela implique de rechercher des vulnérabilités dans l'infrastructure réseau (ressources et stratégies).
- ✓ **Sécurité du logiciel système** : il s'agit d'évaluer les faiblesses des différents logiciels (système d'exploitation, système de base de données et autres logiciels) dont dépend l'application.
- ✓ **Sécurité des applications côté client** : il s'agit de s'assurer que le client (navigateur ou autre outil de ce type) ne peut pas être manipulé.
- ✓ **Sécurité des applications côté serveur** : Cela implique de s'assurer que le code du serveur et ses technologies sont suffisamment robustes pour empêcher toute intrusion.

- **Test de stress** : Le test de stress est une forme de test de performance qui mesure la vitesse de traitement du programme lorsque la charge du système augmente. La charge du système signifie généralement le nombre d'utilisateurs de programmes concurrents (clients). Il peut également se référer au volume de données à traiter, au nombre d'enregistrements dans une base de données, à la fréquence des messages entrants ou à des facteurs similaires [7].
- **Test basé sur le code** : Le test basé sur le code traite directement avec le code source, il doit vérifier si le code du programme génère des erreurs d'exécution. Les tests basés sur le code sont également utilisé pour évaluer l'efficacité des algorithmes [7].

❖ **Le test structurel** : Le test structurel est une approche dans laquelle les tests sont dérivés de la connaissance de la structure du logiciel ou de son implémentation interne (code source).

Chaque type admet une ou plusieurs méthodes pour tester et évaluer la qualité et le rendement du logiciel afin de bien satisfaire les besoins des clients.

5. Les niveaux de test

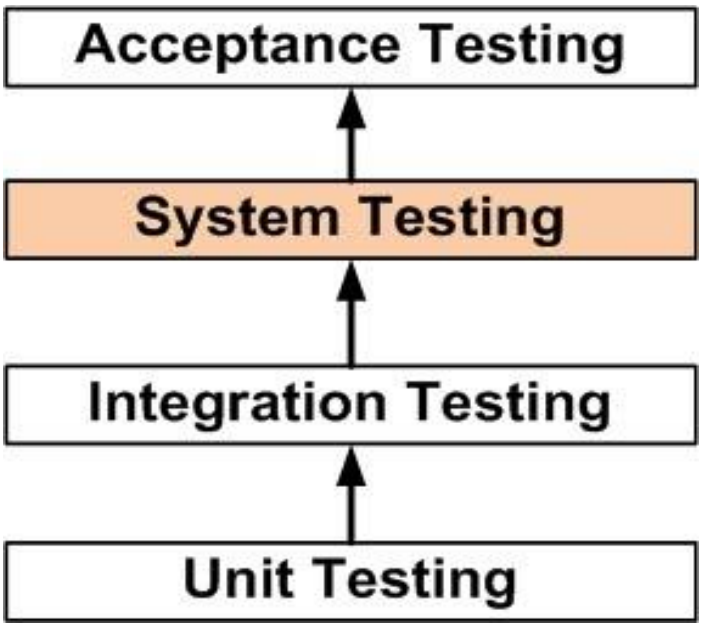


Figure I.3 : enchainement de niveaux de test [24].

❖ **Test d'acceptation** : Les tests d'acceptation sont des tests formels exécutés pour vérifier si un système répond à ses exigences métier. Ils nécessitent que l'application soit entièrement opérationnelle et se concentrent sur la simulation du comportement des utilisateurs. Ils peuvent aussi aller plus

loin et mesurer la performance du système et rejeter les changements si certains objectifs ne sont pas atteints.

- **Test d'intégration** : Les tests d'intégration vérifient que les différents modules ou services utilisés par votre application fonctionnent bien ensemble. Par exemple, ils peuvent tester l'interaction avec la base de données ou s'assurer que les micro-services fonctionnent ensemble comme prévu. Ces types de tests sont plus coûteux à exécuter, car ils nécessitent que plusieurs parties de l'application soient fonctionnelles.
- **Test unitaire** : Les tests unitaires sont de très bas niveau, près de la source de votre application. Ils consistent à tester les méthodes et fonctions individuelles des classes, des composants ou des modules utilisés par votre logiciel. Les tests unitaires sont en général assez bon marché à automatiser et peuvent être exécutés très rapidement par un serveur d'intégration continue.
- **Test système** : est un niveau de test logiciel où un logiciel complet et intégré est testé. Le but de cet essai est d'évaluer la conformité du système aux exigences spécifiées.

Définition par ISTQB :

- **Test du système** : processus consistant à tester un système intégré pour vérifier qu'il répond aux exigences spécifiées.

6. Les caractéristiques du test logiciel

- **Test de robustesse** : est une méthodologie d'assurance qualité centrée sur la robustesse d'un logiciel. Les tests de robustesse ont également été utilisés pour décrire le processus de vérification de la robustesse (c'est-à-dire de la correction) des tests élémentaires dans un processus de test.

ANSI et IEEE ont défini la robustesse comme le degré auquel un système ou un composant peut fonctionner correctement en présence d'entrées non valides ou de conditions environnementales stressantes.

- **Test de performance** : est un type de test logiciel destiné à déterminer le comportement d'un système en termes de réactivité et de stabilité sous une certaine charge.

Il existe essentiellement quatre types de tests de performance :

- ✓ **Les types**
  - ➢ **Le test de charge** : est un type de test de performance réalisé pour évaluer le comportement d'un système face à une charge de travail croissante.
  - ➢ **Le test de résistance** : est un type de test de performance réalisé pour évaluer le comportement d'un système à la limite de sa charge de travail ou au-delà.
  - ➢ **Les tests d'endurance** : sont un type de tests de performance conduits pour évaluer le comportement d'un système lorsqu'une charge de travail importante est donnée en permanence.
  - ➢ **Spike Testing** : est un type de test de performance réalisé pour évaluer le comportement d'un système lorsque la charge augmente brusquement et substantiellement.
- ❖ **Test de conformité** : également appelé test de conformité, test de réglementation, test de normalisation, est un type de test permettant de déterminer la conformité d'un système aux normes internes ou externes. [3]
- ✓ Les normes internes pourraient être des normes établies par l'entreprise elle-même. Par exemple, une société de développement d'applications Web peut définir la norme selon laquelle toutes les pages Web doivent être réactives.
- ✓ Les normes externes peuvent être des normes établies à l'extérieur de l'entreprise. Par exemple, la loi sur la transférabilité et la responsabilité en matière d'assurance maladie (HIPAA) a établi des réglementations pour le secteur de la santé.

Les tests de conformité pourraient également être effectués par une organisation externe. Cela aboutit normalement à une sorte de certification de conformité.

La méthode et le type d'essais à effectuer lors des essais de conformité dépendent de la réglementation / norme spécifique évaluée.

La profondeur des tests de conformité peut aller d'un audit de haut niveau sur une base d'échantillonnage à un examen détaillé de chaque norme spécifiée.

7. Les méthodes de test logiciel

7.1. Test de la boite noire (black box testing) : également connu sous le nom de test comportemental, est une méthode de test logiciel dans laquelle la structure / conception / implémentation interne de l'élément

testé n'est pas connue du testeur. Ces tests peuvent être fonctionnels ou non fonctionnels, bien que généralement fonctionnels.

Définition par ISTQB

- ✓ **Test de la boîte noire** : test, fonctionnel ou non, sans référence à la structure interne du composant ou du système.
- ✓ **Technique de conception des tests en boîte noire** : Procédure permettant de dériver et / ou de sélectionner des scénarios de test sur la base d'une analyse de la spécification, fonctionnelle ou non fonctionnelle, d'un composant ou d'un système sans référence à sa structure interne. Elle est utilisée pour les niveaux de test suivant : test d'intégration/test système/test d'acceptation.

7.2. Test de la boite blanche : (également connu sous le nom de test de boîte transparente, test de boîte ouverte, test de boîte de verre, test de boîte transparente, test basé sur un code ou test structurel) est une méthode de test logiciel dans laquelle la structure / conception / mise en œuvre interne de l'élément testé est connu du testeur. Le testeur choisit les entrées pour parcourir le code et détermine les sorties appropriées. Le savoir-faire en matière de programmation et la connaissance de la mise en œuvre sont essentiels. Les tests de boîte blanche testent au-delà de l'interface utilisateur et dans les moindres détails d'un système.

Définition par ISTQB :

- ✓ **Test en boîte blanche** : test basé sur une analyse de la structure interne du composant ou du système.
- ✓ **Technique de conception de test en boîte blanche** : Procédure permettant de dériver et / ou de sélectionner des cas de test sur la base d'une analyse de la structure interne d'un composant ou d'un système.

La méthode de test de la boîte blanche est applicable aux niveaux de test logiciel suivants :

- o **Test unitaire** : Pour tester des chemins dans une unité.
- o **Test d'intégration** : pour tester les chemins entre les unités.
- o **Test du système** : pour tester les chemins entre les sous-systèmes.

7.3. Test de la boîte grise (grey box testing) : est une méthode de test logiciel qui combine les méthodes Black Box Testing et White Box Testing. Dans le test de la boîte noire, la structure interne de l'élément testé est inconnue du testeur et dans la méthode de test de la boîte blanche, la structure interne est connue. Dans Gray Box Testing, la

structure interne est partiellement connue. Cela implique d'avoir accès à des structures de données internes et à des algorithmes afin de concevoir les scénarios de test, mais d'effectuer des tests au niveau de l'utilisateur ou au niveau de la boîte noire. Elle est utilisée pour le test d'intégration.

7.4. Test agile : est une approche itérative et collaborative, capable de prendre en compte les besoins initiaux du client et ceux liés aux évolutions. [4]

Principes de base

La méthode Agile se base sur un cycle de développement qui porte le client au centre. Le client est impliqué dans la réalisation du début à la fin du projet. Grâce à la méthode agile le demandeur obtient une meilleure visibilité de la gestion des travaux qu'avec une méthode classique.

L'implication du client dans le processus permet à l'équipe d'obtenir un feedback régulier afin d'appliquer directement les changements nécessaires.

Cette méthode vise à accélérer le développement d'un logiciel. De plus, elle assure la réalisation d'un logiciel fonctionnel tout au long de la durée de sa création.

Le principe de base consiste à proposer une version minimale du logiciel puis à intégrer des fonctionnalités supplémentaires à cette base, par processus itératif. Le processus itératif regroupe une séquence d'instructions à répéter autant de fois que possible, selon le besoin. En ce qui concerne la réalisation d'un logiciel, les instructions à répéter sont les suivantes :

- Les tests unitaires à la fin de chaque itération
- Le développement de l'application web
- L'intégration
- La relecture et amélioration des codes

La méthode agile nommée Manifeste Agile repose sur quatre grands principes :

- ✓ Collaboration : Communication et cohésion d'équipe passent avant les outils et les processus.

- ✓ Equipe : Le privilège de la relation équipe/client est mis en avant plutôt que la négociation contractuelle.
- ✓ Application : Préférer une application bien construite à une documentation détaillée.
- ✓ Acceptation : Le choix de l'acceptation du changement et de la flexibilité au détriment d'un plan rigide.

8. **Le test ad hoc** : également connu sous le nom de test aléatoire ou test de singe, est une méthode de test logiciel sans planification ni documentation. Les tests sont effectués de manière informelle et aléatoire, sans procédure formelle ni résultats escomptés.

Le testeur improvise les étapes et les exécute de manière arbitraire (comme un singe qui tape en dansant). Bien que les défauts trouvés en utilisant cette méthode soient plus difficiles à reproduire (en l'absence de tests élémentaires écrits), on trouve parfois des défauts très intéressants qui n'auraient jamais été détectés si des tests élémentaires écrits existaient et étaient strictement suivis. Cette méthode est normalement utilisée lors des tests d'acceptation.

Le succès des tests ad hoc dépend de la créativité et de la ténacité du testeur (et bien sûr de la chance). [3]

9. Processus de test selon l'ISTQB :

Le processus de test se décompose en différentes activités

- Planification et contrôle
- Analyse et conception
- Implémentation et exécution
- Evaluation des critères de sortie et reporting
- Activités de clôture

Certaines de ces activités de test peuvent être menées en parallèle (2 et 3), d'autres en séquence (4 et 5)

9.1. Planification et contrôle

La planification consiste à identifier toutes les activités et ressources nécessaires pour mener à bien l'objectif de test décrit dans la stratégie de test

L’approche de test basé sur les risques va permettre de mettre en phase, via le planning, les activités de tests afin de réduire les risques produits identifiés. De même cette approche va permettre de prioriser les activités.

Le contrôle du planning est une activité permanente qui consiste à comparer les progrès réalisés avec l’avancement prévu et de réviser la planification des activités si besoin.

Les métriques liées à cette activité peuvent inclure

- Couverture des risques par les tests
- Découverte d’anomalies
- Temps passé / planifié à développer le test-ware et exécuter les cas de test

9.2. Analyse et conception des tests

Cette activité consiste à identifier les conditions de test, créer les cas de tests en regard des conditions.

Durant cette activité de conception et la suivante (implémentation et exécution), la priorisation des critères identifiés durant la phase d’analyse de risque et de planification des tests doit être appliquée. [8]

9.3. Implémentation et exécution

Implémentation des tests consiste à :

- Organiser les cas de tests en procédures, en scénarios de test
- Finaliser les données de test et les environnements
- Formaliser un planning d’exécution des cas de test
- Vérifier les critères d’entrée

Cette implémentation doit prendre en compte les objectifs définis dans la stratégie de test, notamment en termes de priorité de test vis à vis par exemple de la criticité des exigences.

Concernant les données de test, c’est durant cette activité que les données peuvent être chargées en base de données, voire crées via des scripts si besoin.

9.4. Exécution des tests

Cette activité démarre lorsque l'objet du test est livré et que les critères d'entrée sont satisfaits.

Les tests manuels doivent être exécutés selon les procédures de test avec un degré de liberté pour permettre la couverture de scénarios intéressants - chaque anomalie devra alors être suffisamment décrite pour pouvoir être reproduite.

Les tests automatisés doivent être exécuté comme prévus.

Les métriques de cette activité peuvent être :

- Pourcentage d'environnement de tests configurés
- Pourcentage de données de tests chargés en base
- Pourcentage de conditions de test et de cas de tests exécutés
- Pourcentage de cas de test automatisés.

9.5. Evaluation des critères de sortie

Du point de vue de la procédure de tests, le suivi d'avancement des tests implique la collecte d'informations. Les métriques vont prendre en compte les critères de sorties, tels que validés durant la planification.

Cela peut être :

- Nombre de cas de test planifiés, exécutés, dont en anomalie et passés.
- Nombre d'anomalies réparties par gravité, et priorité pour celles qui sont corrigée et en cours.
- Nombre d'évolution atteints, pris en compte et testés.
- Pour le reporting au niveau des tests, l'IEEE 829 préconise un Rapport résumé, constitué de :
- Identifiant du rapport
- Résumé
- Ecarts
- Evaluation complète
- Résumé des résultats
- Résumé des activités
- Approbations
- Evaluation

- Le reporting des tests peut être produit à la fin des différents niveaux de test (tests unitaires, tests d'intégration, tests système)

9.6. Activités de clôture des tests

Ces activités peuvent être regroupées en 4 ensembles.

- S'assurer que chaque activité de test est menée à terme, c'est à dire que les tests planifiés ont été exécutés, ou ajournés ; les anomalies ont été corrigées, recyclées, remises à plus tard ou déclarées contournables.
- Donner des taches aux bons profils en attribuant des anomalies typées (régression, performance, fonctionnelle) à ceux qui ont la compétence requise.
- Participer à des retours d'expérience avec rédaction de documentation sur les leçons apprises à la fois sur le projet de test et sur le cycle de vie du logiciel.
- Archiver les résultats et les documents
- Les métriques de suivi des activités de clôture des tests peuvent inclure
- Pourcentage de cas de test exécutés
- Pourcentage de cas de test identifiés comme réutilisable dans le répertoire des cas de tests
- Pourcentage de cas de test identifié comme test de non régression.
- Ratio de cas de test automatisés, manuels.

Conclusion : Dans ce chapitre nous mettons l'accent sur les notions de base, nous avons citez la définition d'un logiciel en informatique et les différents cycle de vie d'un logiciel plus les tests d'un logiciel et les méthodes existant pour réaliser ce test logiciel, Dans ce travail nous devons focaliser sur le test fonctionnel dans le processus de prise de décision.

La chapitre qui suit traitera d'une maniéré détaille. L'Automatisation des tests logiciels (test fonctionnel).

Chapitre II : Automatisation d’un test logiciel

Introduction :

Le test logiciel, comme il a été précisé dans le chapitre précèdent, permettant de vérifier et de s'assurer de la fiabilité du logiciel d'un point de vue développement ou fonctionnel. Ce test peut s'effectue selon deux approches : manuel ou automatisé

1. **Test manuel** : Le test manuel est un test du logiciel où les tests sont exécutés manuellement par un analyste de l'assurance qualité. Il est effectué pour découvrir des bugs dans les logiciels en développement.
   En test manuel, le testeur vérifie toutes les fonctionnalités essentielles de l'application ou du logiciel donné. Dans ce processus, les testeurs de logiciels exécutent les scénarios de test et génèrent les rapports de test sans l'aide d'aucun outil de test de logiciel d'automatisation.
   C'est une méthode classique de tous les types de tests et permet de détecter les erreurs dans les systèmes logiciels. Il est généralement effectué par un testeur expérimenté pour mener à bien le processus de test du logiciel. [10]

1.1. Avantages des tests manuels :

- Obtenez un retour visuel rapide et précis
- C'est moins coûteux, car vous n'avez pas besoin de dépenser votre budget pour les outils et processus d'automatisation.
- Le jugement humain et l'intuition profitent toujours à l'élément manuel
- Lors du test d'un petit changement, un test d'automatisation nécessiterait un codage qui pourrait prendre beaucoup de temps. Bien que vous puissiez tester manuellement à la volée.

1.2. Inconvénients du test manuel :

- Méthode de test moins fiable car réalisée par l'homme. Par conséquent, il est toujours sujet aux erreurs et aux erreurs.
- Le processus de test manuel ne peut pas être enregistré, il est donc impossible de réutiliser le test manuel.
- Dans cette méthode de test, certaines tâches sont difficiles à effectuer manuellement, ce qui peut nécessiter un temps supplémentaire de la phase de test du logiciel.

2. **Test automatisé :** Dans Test automatisé de logiciels, les testeurs écrivent des scripts de code / de test pour automatiser l'exécution du test. Les testeurs utilisent des outils d'automatisation appropriés pour développer les scripts

de test et valider le logiciel. L'objectif est d'achever l'exécution du test en moins de temps.

Les tests automatisés reposent entièrement sur le test pré-scripté qui s'exécute automatiquement pour comparer les résultats réels aux résultats attendus. Cela aide le testeur à déterminer si une application fonctionne comme prévu.

Les tests automatisés vous permettent d'exécuter des tests répétitifs et des tests de régression sans l'intervention d'un testeur manuel. Même si tous les processus sont effectués automatiquement, l'automatisation nécessite un certain effort manuel pour créer des scripts de test initiaux.

2.1. **Avantage des tests automatisés :**

- Les tests automatisés vous aident à trouver plus de bogues par rapport à un testeur humain
- Comme la majeure partie du processus de test est automatisée, vous pouvez avoir un processus rapide et efficace
- Le processus d'automatisation peut être enregistré. Cela vous permet de réutiliser et d'exécuter le même type d'opérations de test.
- Les tests automatisés sont effectués à l'aide d'outils logiciels, de sorte que le test manuel fonctionne sans fatigue ni fatigue contrairement aux humains.
- Il peut facilement augmenter la productivité car il fournit des résultats de test rapides et précis
- Les tests automatisés prennent en charge diverses applications
- La couverture de test peut être augmentée grâce à l'outil de test d'automatisation, n'oubliez jamais de vérifier même la plus petite unité

2.2. **Inconvénients des tests automatisés :**

- Sans élément humain, il est difficile d'obtenir un aperçu des aspects visuels de votre interface utilisateur tels que les couleurs, la police de caractères, la taille, le contraste ou la taille des boutons.
- Les outils permettant d'exécuter des tests d'automatisation peuvent être coûteux, ce qui peut augmenter le coût du projet de test.
- L'outil de test d'automatisation n'est pas encore une preuve complète. Chaque outil d'automatisation a ses limites, ce qui réduit la portée de l'automatisation.
- Le débogage du script de test est un autre problème majeur du test automatisé. La maintenance des tests est coûteuse.

3. Etude comparatif entre test manuel et automatique :

Paramètre	Test d'automatisation	Test manuel
Définition	Les tests d'automatisation utilisent des outils d'automatisation pour exécuter des cas de test.	Dans les tests manuels, les scénarios de test sont exécutés par un testeur humain et un logiciel.
Temps de traitement	Les tests automatisés sont nettement plus rapides qu'une approche manuelle.	Les tests manuels prennent beaucoup de temps et prennent beaucoup de ressources humaines.
Essais exploratoires	L'automatisation ne permet pas les tests aléatoires	Les tests exploratoires sont possibles dans les tests manuels
Investissement initial	L'investissement initial dans les tests automatisés est plus élevé. Bien que le retour sur investissement soit meilleur à long terme.	L'investissement initial dans les tests manuels est comparativement plus faible. Le retour sur investissement est inférieur aux tests d'automatisation à long terme.
Fiabilité	Le test automatisé est une méthode fiable, car il est exécuté par des outils et des scripts. Il n'y a pas de test de fatigue.	Les tests manuels ne sont pas aussi précis en raison de la possibilité d'erreurs humaines.
Observation humaine	Les tests automatisés n'impliquent aucune considération humaine. Cela ne garantit donc jamais une convivialité et une expérience client positives.	La méthode de test manuel permet l'observation humaine, ce qui peut être utile pour offrir un système convivial.
Exécution parallèle	Ce test peut être exécuté sur différentes plates-formes d'exploitation en parallèle et réduit le temps d'exécution du test.	Les tests manuels peuvent être exécutés en parallèle, mais il faudrait augmenter vos ressources humaines, ce qui est coûteux
Tests par lots	Vous pouvez grouper plusieurs scripts de test	Les tests manuels ne peuvent pas être groupés.

	pour une exécution nocturne.	
Connaissances en programmation	La connaissance de la programmation est indispensable dans les tests d'automatisation.	Pas besoin de programmation en test manuel.
Installer	Le test d'automatisation nécessite une configuration d'exécution moins complexe.	Les besoins de tests manuels ont une configuration d'exécution plus simple des tests
Approche idéale	Les tests d'automatisation sont utiles lors de l'exécution fréquente du même ensemble de cas de test	Le test manuel s'avère utile lorsque le scénario de test n'a besoin d'être exécuté qu'une ou deux fois.
Conception de test	Les tests unitaires automatisés appliquent / pilotent la conception de développement pilotée par les tests.	Les tests unitaires manuels n'entraînent pas la conception dans le processus de codage

Tableau II.1 : Tableau comparatif de test manuel/automatique.

4. Outils d'automatisation de test fonctionnel :

4.1. **Sélénium :** Sélénium est un outil open source utilisé pour automatiser les tests effectués sur les navigateurs Web (les applications Web sont testées à l'aide de n'importe quel navigateur Web). Étant donné que Sélénium est une source ouverte, aucun coût de licence n'est impliqué, ce qui représente un avantage majeur par rapport aux autres outils de test. [15]

4.2. **Watir :** est un outil de test d'automatisation permettant de tester des applications Web. C'est une bibliothèque open source de Ruby et tout le monde peut l'obtenir sur Git Hub. C'est très utile pour automatiser les navigateurs Web et vous permet de créer des tests simples et faciles à gérer. Contrairement aux outils de test basés sur le protocole HTTP qui simule les requêtes du navigateur, Watir simule l'interaction de l'utilisateur avec le navigateur à l'aide du protocole OLE, basé sur l'architecture COM. Ruby prend en charge OLE, ce qui permet l'automatisation de Microsoft Internet Explorer. [12]

Il est caractérisé par :

- Teste toute application Web basée sur la langue.
- Test inter-navigateur.
- Compatible avec les outils de développement orientés métier tels que RSpec, Cucumber et Test / Unit.
- Teste les boutons, formulaires, liens et réponses de la page Web.

4.3. **Protractor :** est un Framework d'automatisation fonctionnelle open source (également connu sous le nom de Framework de test End to End) spécialement conçu pour vérifier la santé des applications web AngularJS. C'est un programme Node.js qui supporte les frameworks de test Jasmine, Mocha et Cucumber. Il utilise Sélénium Web Driver pour piloter les navigateurs et simuler l'interaction de l'utilisateur avec une application AngularJS exécutée dans un navigateur. L'attente automatique de Protractor peut automatiquement exécuter l'étape suivante de votre test au moment où la page Web termine les tâches en attente. Protractor est une infrastructure de test de bout en bout pour les applications AngularJS et fonctionne comme un intégrateur de solutions combinant des outils et des technologies puissants tels que NodeJS, Sélénium

Web Driver, Jasmine, Cucumber et Mocha. Il a été initialement développé par Google Developers pour prendre en charge les applications angulaires et plus tard, il est publié en tant que Framework open source. Le rapporteur prend désormais en charge les applications angulaires et non angulaires. Le rapporteur est un wrapper écrit sur Webdriver.js, toutes les fonctionnalités prises en charge par Sélénium Web driver sont prises en charge, en plus des fonctionnalités spécifiques à l'angle.

WebDriverJs est l'implémentation JavaScript officielle du sélénium. Il utilise le protocole Sélénium JSON-Wire-Protocol pour interagir avec le navigateur comme le fait le sélénium java. Le rapporteur dépend de WebdriverJs pour interagir avec le navigateur. [13]

Il est caractérisé par :

Prend en charge les localisateurs spécifiques : l'application angulaire est fournie avec des localisateurs spécifiques tels que ng-model, ng-bind, ng-

repeat, etc., de sorte que le rapporteur a étendu la prise en charge de ces localisateurs. Vous n'avez pas besoin de créer un XPath complexe pour les localisateurs angulaires, dans Protractor, ces localisateurs sont prêts pour vous, vous pouvez donc simplement utiliser by.model, by.repeater, etc.

Prise en charge des applications non angulaires et angulaires : Protractor offre une prise en charge étendue des applications angulaires, mais également des applications non angulaires. Cela signifie que si votre partie de l'application est non angulaire et angulaire, vous pouvez toujours choisir la structure de rapporteur et effectuer l'automatisation complète des tests d'automatisation.

Prend en charge les tests entre navigateurs : nous pouvons exécuter nos scripts dans plusieurs navigateurs tels que Chrome, Firefox, Safari, IE11, Edge. La configuration pour les tests inter-navigateurs est facile et ne prend pas beaucoup de temps. Nous aborderons ce sujet en détail dans nos prochains tutoriels.

Prise en charge de l'exécution parallèle : considérez que nous avons un grand nombre de scénarios de test. Si tous les scénarios de test doivent être exécutés de manière séquentielle, c'est-à-dire, l'un après l'autre, cela prend beaucoup de temps. L'exécution en parallèle aide dans ce cas. L'exécution en parallèle exécute des cas de test dans plusieurs instances du navigateur, par exemple si vous avez 4 cas de test. Dans une exécution sur deux instances du navigateur, chacun exécute deux ou deux cas de test à un moment donné. Donc, en moins de temps, plus de tests les cas seront exécutés.

Prise en charge du navigateur sans tête : Un navigateur sans tête est un sans interface utilisateur. Protractor prend également en charge le navigateur sans tête.

Prend en charge l’extensibilité : puisque rapporteur est une application

node.js, il peut utiliser la grande variété de packages disponibles dans le nœud. On peut donc étendre son Framework ou ajouter des fonctionnalités supplémentaires en installant des packages de nœuds. Par exemple, si vous avez besoin d'un rapport HTML, vous pouvez simplement utiliser Jasmine HTML Reporter. Pour le code épuré, vous pouvez installer eslint ou tslint. De même, vous pouvez installer les packages de nœuds de votre choix.

Prend en charge les plug-ins de création de rapports : comparé au cadre d'automatisation open source non basé sur node.js, il est facile de configurer le rapport dans le rapporteur. Jasmine HTML Reporter est l'un des frameworks de génération de rapports. Qui produit des rapports HTML pour vos spécifications. Ci-dessous l'exemple de rapport

Prise en charge des plates -formes de test sur le cloud : les plates -formes de test basées sur le cloud nous permettent d'exécuter nos spécifications sur plusieurs navigateurs (Chrome, Firefox, Safari, etc.) et sur plusieurs plates-formes (Windows, Mac, Linux, Mobile, etc.). Protractor est pris en charge par les principales plates-formes de tests sur le cloud telles que SauceLabs et CrossBrowserTesting.com.

Prise en charge de plusieurs bibliothèques d'assertion : Protractor prend en charge plusieurs bibliothèques d'assertion telles que Jasmine, Mocha ou vous pouvez même utiliser vos bibliothèques personnalisées. Par défaut, le rapporteur utilise la structure Jasmine pour les assertions.

Prise en charge du flux de contrôle : l'API est basée sur des promesses gérées par un flux de contrôle et adaptées à Jasmine. Les API de rapporteur sont entièrement asynchrones. Toutes les fonctions renvoient des promesses. Il maintient une file d'attente de promesses en attente, appelée flux de contrôle, pour que l'exécution soit organisée. (Nous en parlerons davantage dans Promises dans les prochaines sections du didacticiel).

Prise en charge de divers IDE et rédacteurs : vous pouvez choisir parmi une multitude d'EDI sur le marché. Les éditeurs les plus populaires sont Visual Studio Code, Web Storm, Visual Studio Professional, Atom et sublime. (Nous couvrirons en détail nos prochains tutoriels).

Prise en charge de CI / CD : une fois les scripts d'automatisation en place, il n'est pas nécessaire de les exécuter manuellement et de les surveiller. Chaque fois qu'un nouveau code est envoyé dans le référentiel, les tests du rapporteur doivent être exécutés automatiquement et vous fournir le rapport. Cela signifie que nous devons automatiser le processus. Nous pouvons y parvenir en intégrant le test du rapporteur afin de créer des outils tels que Jenkins, TFS ou Azure DevOps. Protractor prend en charge l'intégration de l'outil de génération. (Nous traiterons étape par étape des détails de configuration dans nos prochains tutoriels).

4.4. Robot Framework : (Licence : Open-source) : est un Framework d'automatisation open-source qui implémente l'approche par mot clé pour les tests d'acceptation et le développement piloté par les tests d'acceptation (ATDD). Robot Framework est indépendant du système d'exploitation et des applications. Le Framework de base est implémenté en utilisant Python et fonctionne également sur Jython (JVM) et IronPython (.NET). Ses capacités de test peuvent être étendues par des bibliothèques de tests implémentées avec Python ou Java, et les utilisateurs peuvent créer de nouveaux mots-clés de niveau supérieur à partir de ceux existants en utilisant la même syntaxe que celle utilisée pour créer des cas de test. [14]

4.5. Katalon Studio (Licence : Gratuit) : est une solution de test d'automatisation gratuite développée par Katalon LLC. Le logiciel est construit sur les infrastructures d'automatisation open source Sélénium et Appium, avec une interface IDE spécialisée pour les tests d'API, Web et Mobile. Sa première publication publique a eu lieu en septembre 2016.

Katalon Studio fournit une double interface interchangeable pour les scripts :

Un éditeur de tableaux d'enregistrement pour les utilisateurs moins techniques

Un IDE de script destiné aux testeurs expérimentés qui peuvent créer des tests d'automatisation avec mise en évidence de la syntaxe et complétement intelligent du code.

Les éléments de l'interface graphique et les méthodes de l'API peuvent être capturés à l'aide de l'utilitaire d'enregistrement et stockés dans le référentiel d'objets, qui est accessible et réutilisable dans différents scénarios de test.

La planification des tests peut être structurée à l'aide de suites de tests avec des variables d'environnement. L'exécution du test peut être paramétrée et mise en parallèle à l'aide de profils.

L'exécution à distance peut être déclenchée par les systèmes de CI via le conteneur Docker ou l'interface de ligne de commande (CLI).

Les informations de débogage et les rapports de test peuvent être visualisés avec Katalon Studio, exportés au format JUnit ou analysés par le service de compte rendu de test Katalon Analytics - Katalon.

Katalon Recorder est un complément du navigateur permettant d'enregistrer les actions de l'utilisateur dans les applications Web et de générer des scripts de test. Katalon Recorder prend en charge Chrome et Firefox. Il fonctionne de la même manière que l'utilitaire d'enregistrement de Katalon Studio, mais il peut également exécuter des étapes de test et exporter des scripts de test dans de nombreux langages tels que C#, Java et Python. [19]

4.6. **Unified Functional Testing** (UFT) : (Licence : Commercial) UFT / QTP est un outil de test fonctionnel automatisé de Micro Focus qui utilise des tests automatisés pour identifier les bogues dans une application testée.

UFT signifie U nified F unctional T Esting. Il a été précédemment connu sous le nom QTP (Q uick T is P rofessional). En fait, les vétérans de cet outil continuent de le désigner par QTP.

QTP a été conçu à l'origine par la société Mercury Interactive, acquise par Hewlett Packard (HP) en 2006. En 2011, avec l'introduction de la version 11.5, QTP a été renommé UFT.

En septembre 2017, HPE spin a fusionné dans Micro Focus. Depuis lors, UFT est conçu, pris en charge et géré par Micro Focus.

UFT est principalement utilisé pour les tests de fonctionnement, de régression et de service. À l'aide de UFT, vous pouvez automatiser les actions des utilisateurs sur une application Web ou basée sur un client, tester et identifier des bogues des mêmes actions pour différents utilisateurs, différents ensembles de données, différents systèmes d'exploitation Windows et / ou différents navigateurs. L'automatisation utilisant UFT, si elle est planifiée et exécutée de manière appropriée, permet de gagner beaucoup de temps et d'argent par rapport aux tests manuels.

UFT est l'un des outils de test d'automatisation commerciaux les plus largement utilisés sur le marché aujourd'hui. Il est connu pour sa facilité d'utilisation et son support par le fournisseur et la grande communauté de testeurs en automatisation. Pour cette raison, les professionnels UFT qualifiés sont toujours en demande. [20]

4.7. TestComplete : est un environnement de test automatisé pour un large éventail de types d'applications et de technologies, notamment Windows, .NET, WPF, Visual C ++, Visual Basic, Delphi, C ++ Builder, Java et les applications et services Web.

TestComplete est orienté également vers les tests fonctionnels et unitaires. Il fournit un support supérieur pour les tests de régression quotidiens et

prend en charge de nombreux autres types de tests : tests basés sur les données, tests distribués, etc.

Vous créez des tests en les enregistrant ou en modifiant des commandes de test dans des panneaux et des éditeurs TestComplete. Les tests peuvent être exécutés à partir de TestComplete ou ils peuvent être exportés vers une application externe et exécutés là-bas.

TestComplete reconnaît les objets et les contrôles dans les applications testées et propose des commandes spéciales pour simuler les actions de l'utilisateur avec eux. Il offre également des points de contrôle spécifiques qui vous permettent de vérifier facilement l'état de l'application pendant l'exécution du test.

Si les moyens intégrés ne suffisent pas pour simuler les actions de l'utilisateur sur l'application testée ou pour vérifier l'état de l'application, vous pouvez tirer parti de l'accès aux objets internes, méthodes et propriétés de l'application pour effectuer les tâches nécessaires. [15]

4.8. Tricentis Tosca (Licence : Commercial) : est un outil d'automatisation de test basé sur un modèle qui fournit un ensemble assez large de fonctionnalités pour les tests en continu, y compris les tableaux de bord, les analyses et les intégrations pour prendre en charge les méthodologies agiles et DevOps. Comme beaucoup d'autres logiciels de test d'automatisation, il prend en charge un large éventail de technologies et d'applications telles que le Web, les mobiles et les API. Tricentis Tosca offre également une intégration prête à l'emploi avec des outils populaires tels que Jenkins, Jira, GitHub, Docker, Puppet, Visual Studio, etc. Les autres fonctionnalités comprennent la gestion des risques et la gestion de l'intégration.

4.9. Ranorex : est un puissant outil d'automatisation de tests. Il s'agit d'une infrastructure d'automatisation de test d'interface graphique utilisée pour tester des applications Web, de bureau et mobiles. Ranorex. N'a pas son propre langage de script pour automatiser l'application. Il utilise des langages de programmation standard tels que VB.NET Shop et C #. [16]

4.10. Telerik TestStudio (Licence : Commercial) : est une solution de tests fonctionnels automatisés pour les tests de régression, de fonctionnement, de charge et d'API automatisés.

Contrairement à plusieurs autres solutions de test qui imposent leur propre langage propriétaire, Telerik Test Studio utilise le langage C # ou VB.Net. L'utilisation d'un vrai langage de codage signifie qu'il est plus facile de trouver de l'aide en ligne. Les scripts de test peuvent également être partagés via le contrôle de source avec d'autres membres de l'équipe. [17]

Telerik est également un outil basé sur des icônes qui automatise les tests fonctionnels ou de régression de toute application Web ou de bureau. Même un testeur peu ou pas connaissant le codage peut également utiliser Telerik. Il peut être utilisé à la fois pour l'enregistrement et la lecture, ainsi que pour les scripts descriptifs. Les testeurs peuvent écrire des scripts ou utiliser simplement la fonctionnalité étendue d'enregistrement et de lecture proposée par Telerik.

✓ **Avantages du Telerik Test Studio :**

- Il utilise l'écran actif pour l'enregistrement, ce qui signifie que le testeur peut se référer à l'écran pendant l'enregistrement des scripts.
- Telerik utilise la structure DOM de la page Web pour identifier les propriétés de l'objet.
- Telerik Test Studio est également capable d'interagir avec tout système de contrôle de source basé sur des fichiers et s'intègre directement à Team Foundation Server (TFS) et à Git.
- La maintenance des scripts de test est assez facile et toute modification dans les scripts peut être facilement intégrée
- Rapports détaillés
- Exécution à distance et planification des tests

✓ **Inconvénients de Telerik Test Studio :**

- Cela vient avec un coût de licence
- Le support en ligne est disponible avec le renouvellement de la licence

- L'utilisation de propriétés d'objet relatives peut être difficile
- Nécessité d'installer un complément pour le contrôle de source

4.11. **Appium :** Appium est un outil d'automatisation de test pour les applications mobiles. Il permet de tester tous les trois types d'applications mobiles : native, hybride et web. Il permet également d'exécuter les tests automatisés sur des périphériques, des émulateurs et des simulateurs réels. [21]

5. Etude comparatif entre quelque outil d'automatisation de test :

	Selenium	TestComplete	Ranorex	Appium
Plateforme de test	Multiplateforme	Windows	Multiplateforme	iOS, Android, Windows
Type d'applications	Web	Web, Desktop, Mobile	Web, Desktop, Mobile	Desktop, Mobile
Langage de programmation supporté	Java, C#, Perl, Python, JavaScript, Ruby, PHP, R	JavaScript, Python, VBScript, JScript, Delphi, C++, C#	JavaScript, Python, PHP, Perl, Ruby, C++, C#, Java	Ruby, Python, Java, JavaScript, PHP, C#
Compétences en programmation	Nécessite des compétences avancées pour intégrer divers outils	Non requis	Non requis	Moyen

Courbe d'apprentissage	Difficile	Moyen	Moyen	Facile
Facilité d'utilisation	Difficulté dans l'installation et l'intégration des divers outils	Facile à installer et à utiliser	Facile à installer et à utiliser	Facile à installer et à utiliser
Type de test supporté	Test fonctionnel, test de régression	Test unitaire, test fonctionnel, test de régression, test de monté en charge, test de couverture	Test fonctionnel, test de régression, test de navigateur	Test fonctionnel
Simulateur / Emulateur	Oui	Oui	Oui	Oui
Documentation / Support de communauté	Beaucoup de documentation et une communauté active	Documentation moyenne et une communauté active	Documentation moyenne et une communauté active	Immense documentation et une large communauté active
Enregistrement / Lecture des scénarios	Oui	Oui	Oui	Non, nécessite un outil

				suppléme ntaire
Type de licence	Open-Source	Propriétaire	Propriétaire	Open-Source

Tableau II.2 : Tableau comparatif d'outil d'automatisation de test.

De nos jours, beaucoup, may-être la Plupart des applications logicielles sont écrites sous la forme d'applications Web à exécuter dans un navigateur Internet. L'efficacité des tests de ces applications variées d'une entreprise à l'autre. Une the software of high software and interactive gile, l'automatisation des tests devient souvent une exigence pour les projets logiciels. L'automatisation des tests est souvent la solution. Automatisation des tests signifie utiliser un outil logiciel pour exécuter des tests répétés sur l'application à testeur. Pour les tests de régression, cela fournit cette réactivité. Testeur l'automatisation présente de nombreux avantages. Donc la pluparts liés à la répétabilité des tests et à la vitesse à laquelle ils peuvent être exécutés. Un certain nombre d'outils commerciaux et open source sont disponibles pour vous dans le développement de l'automatisation des tests. Le sélénium est peut-être la solution ouverte source la plus largement utilisée.

Ce guide de l'utilisateur les utilisateurs débutants et les utilisateurs expérimentés de Sélénium à apprendre des techniques efficaces d'automatisation des tests pour les applications Web.

Enfin nous choisissons l'outil Sélénium pour la gestion automatique de notre application.

Conclusion : cette partie elle a été consacrée à la présentation des approches des tests, le test manuel et le test automatique en présentant les avantages et inconvénients de l'une de ces types des tests ainsi que certains outils utilisés. Dans ce chapitre nous mettons l'accent seulement sur l'outil d'automatisation de test fonctionnel qui fera l'objet de l'outil que nous allons développer pour le test des tableaux de bord. Aussi, à l'issue de l'étude comparative présentée dans le chapitre précèdent nous avons choisi l'outil sélénium pour l'automatisation de test fonctionnel.

Chapitre III:

La Conception De Notre Système

Introduction

Nous allons présenter dans ce chapitre la conception de notre application en modélisant le système via les diagrammes d'UML tel que cas d'utilisation, classe et séquence.

1. Présentation du langage de modélisation UML

Le Langage de Modélisation Unifié, de l'anglais Unified Modeling Language (UML), est un langage de modélisation conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est provenu de l'intégration de langages précédents de modélisation objet tel qu'OMT (Objet Modeling Technique) et OOSE (Object Oriented Software Engineering). Il offre un standard de modélisation pour représenter l'architecture logicielle.

1.1. Diagramme de cas d'utilisation Globale de notre système

Il décrit les principales fonctionnalités de notre application en spécifiant les actions que l'utilisateur peut effectuer sur elle. Ci-dessous représente le diagramme :

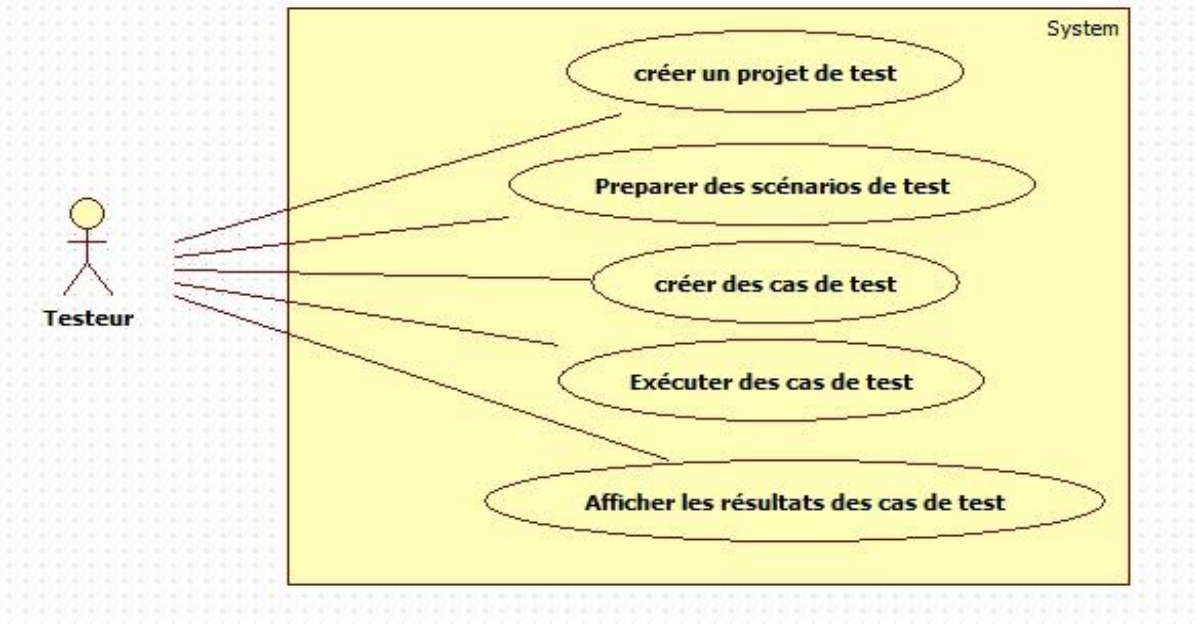


Figure III.4 : Diagramme de cas d'utilisation globale du système

1.2. Diagramme de cas d'utilisation des scénarios de test

Le diagramme de scénarios de test désigne une fonctionnalité de base de notre système qui permet au testeur d'ajouter ou modifier ou supprimer et ensuite enregistrer pour les exécuter autant de fois qu'il veut. Ci-dessous représente le diagramme :

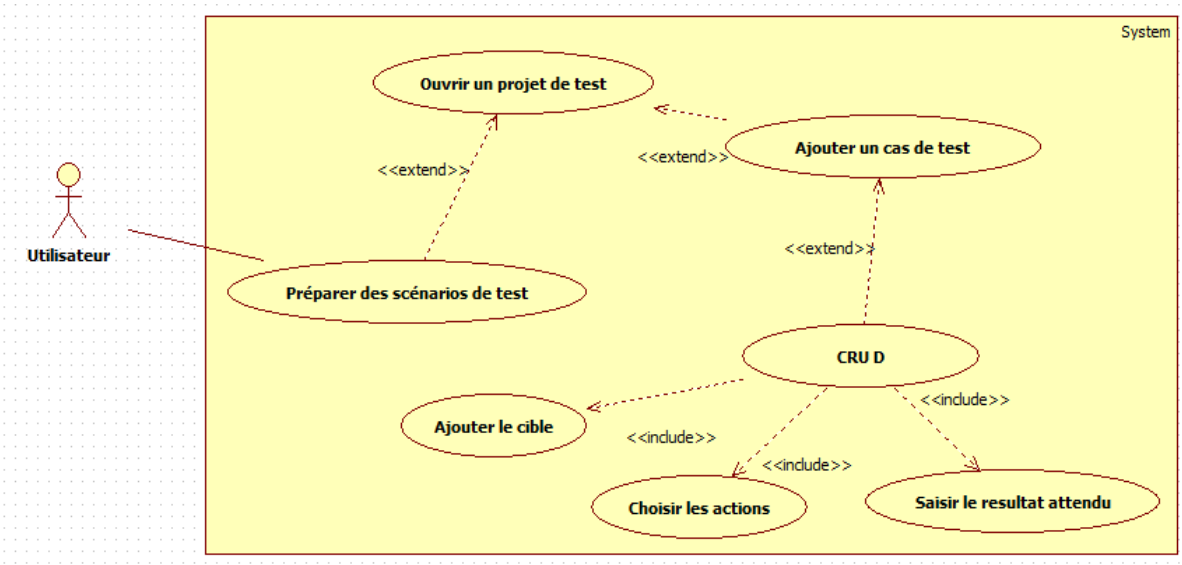


Figure III.5 : Diagramme de cas d'utilisation de scénario de test

1.3. Diagramme de cas d'utilisation d'exécution des cas de test

Ce diagramme représente la fonctionnalité d'exécution des tests. Il désigne les actions que le testeur pourrait faire pour exécuter les cas de test. Le testeur peut manœuvrer l'ordre de l'exécution, ajouter ou supprimer des cas de tests ainsi qu'il peut choisir l'application qui sera testé. La figure suivante montre ce diagramme :

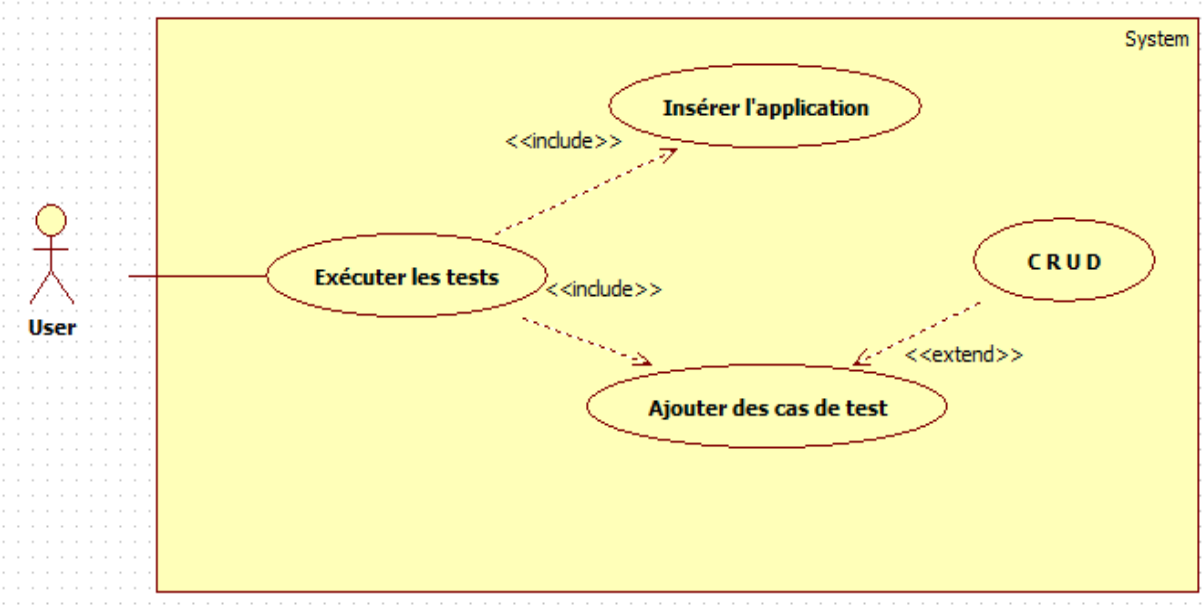


Figure III. 6 : Diagramme cas d'utilisation d'exécution des cas de test

1.4. Diagramme de séquence de la création de projet de test

Il décrit la création des projets de test en spécifiant l'échange des messages entre l'utilisateur et le système lui-même. Le testeur peut créer un ou plusieurs projets de tests autant qu'il veut. La figure ci-dessous illustre ce diagramme :

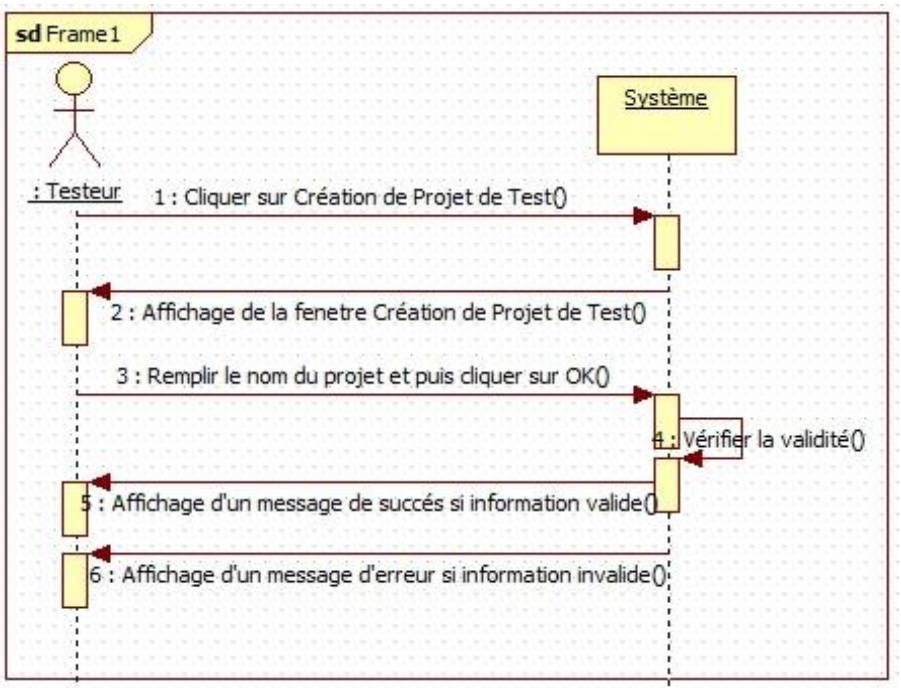


Figure III.7 : Diagramme de séquence de création de projet de test

1.5. Diagramme de séquence de la création des cas de test

Ce diagramme représente le scénario de création des cas de test qui représentent les différentes conditions qu'on peut appliquer sur chaque fonctionnalité. Le testeur peut créer un ou plusieurs cas de test. Ci-dessous illustre ce diagramme :

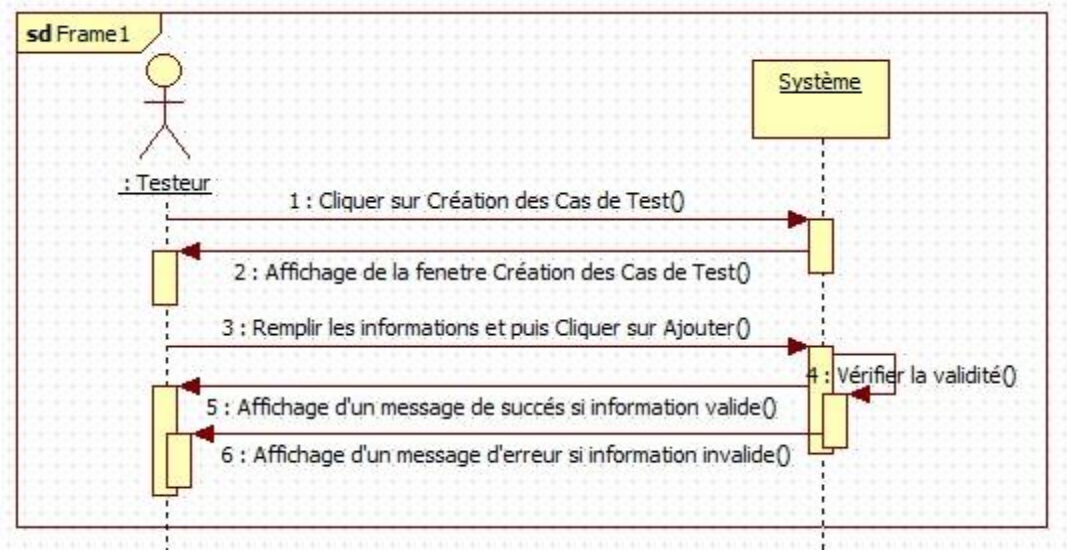


Figure III.8 : Diagramme de séquence de la création des cas de test

1.6. Diagramme de séquence de l'exécution des cas de test

Ce diagramme représente l'exécution des cas de tests où le testeur peut choisir d'exécuter un ou tous les cas de test qui existent. La figure suivante illustre ce diagramme :

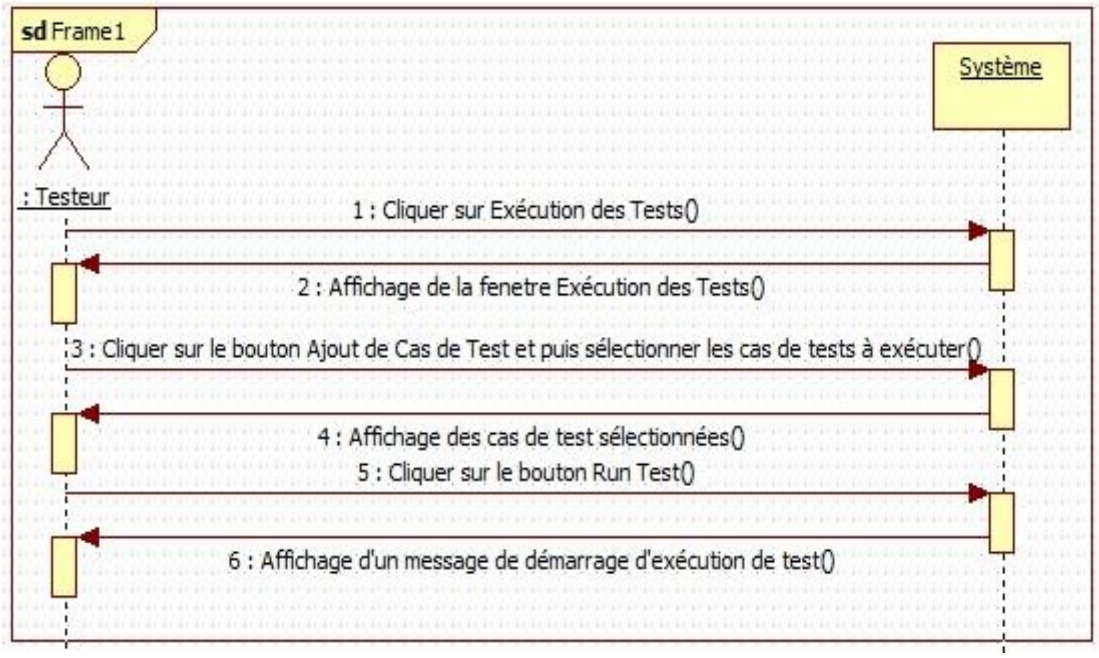


Figure III.9 : Diagramme de séquence de l'exécution des cas de test

2. Règles de gestion

Le diagramme de classe de notre système est basé sur les règles de gestion suivante :

- ❖ Un projet est composé d'un ou plusieurs référentiels.
- ❖ Un référentiel est associé à un seul projet.
- ❖ Un référentiel est composé d'un ou plusieurs scénarios.
- ❖ Un scénario appartient à un seul référentiel.
- ❖ Un scénario est composé d'un ou plusieurs cas de test.
- ❖ Un cas de test appartient à un ou plusieurs scénarios.

3. Diagramme de classe du système

Ce diagramme permet de fournir une représentation abstraite des objets du système qui interagissent. Ci-dessous nous présentons le diagramme de classe de notre système.

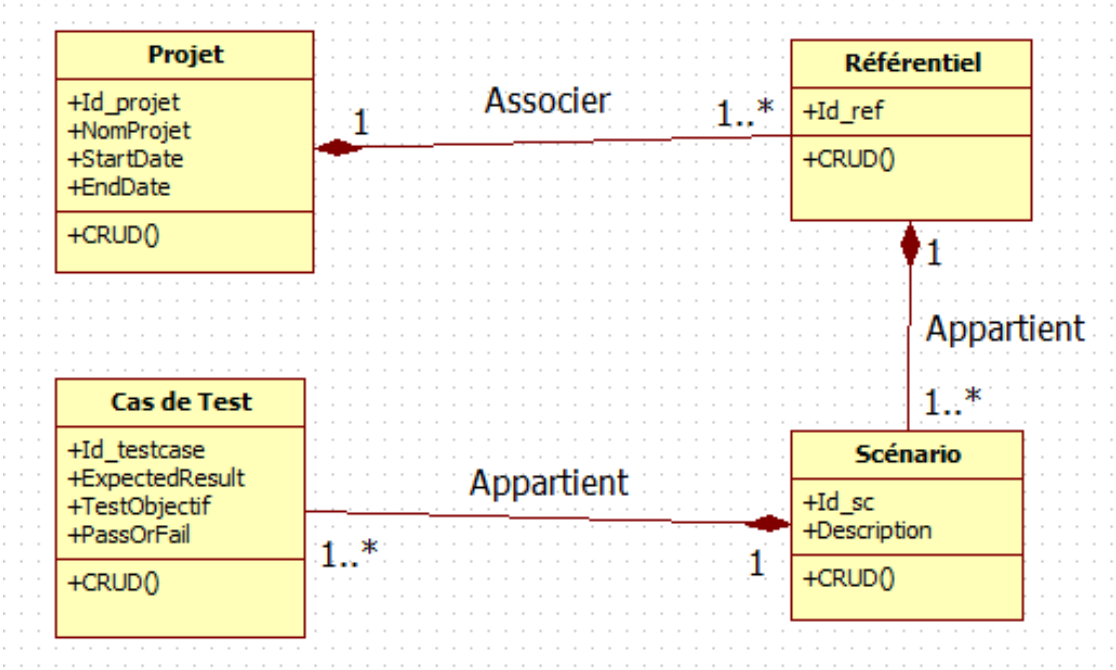


Figure III.10 : Diagramme de classe du système

Conclusion : Pour conclure, nous avons décrit notre système via une conception en le modélisant suivant le langage UML tel que le diagramme de classe, cas d'utilisation et le diagramme de séquence, ce qui nous fournit une base solide pour passer à l'implémentation.

Chapitre IV: Implémentation Et Test

Introduction

Dans ce chapitre, nous allons présenter notre implémentation qui est basée sur la conception de notre système.
D'abord, nous commencerons par présenter les outils utilisés (langage de programmation java, sublime text 3 …) et puis notre outil de test via captures d'écrans des interfaces. Deuxièmement, nous allons effectuer des différents cas de tests sur notre application (tableau de bord) et enfin nous montrons les résultats obtenus via captures d'écrans.

1. Présentation de la plateforme de test Sélénium

Sélénium est un outil d'automatisation de test open source destiné à tester les applications Web. Il fonctionne dans plusieurs navigateurs et plusieurs systèmes d'exploitation. Il se compose de deux parties :

- Sélénium IDE : c'est une extension de Firefox, qui permet d'enregistrer une suite d'actions, qu'il sera possible de rejouer à volonté ;
- Sélénium Web Driver : il s'agit cette fois d'une API, disponible pour plusieurs langages, permettant de programmer des actions sur l'interface, et à vérifier les réponses. Les actions à réaliser peuvent être exportées depuis Sélénium IDE.

  Dans le cadre du développement d'une application, quelle qu'elle soit, les tests sont indispensables, et prennent une part non négligeable du développement. Il en existe plusieurs types : unitaires, intégration, fonctionnels, qualification, etc. Aujourd'hui, la plupart sont automatisés, ce qui permet un gain de temps substantiel, ainsi qu'une plus grande fiabilité.

2. Architecture de Sélénium

- Une fois que vous avez cliqué sur exécuter, la bibliothèque client sélénium communiquera avec l'API sélénium.
- L'API Sélénium envoie la commande issue de la liaison de niveau linguistique au pilote de navigateur à l'aide du protocole filaire JSON.
- L'API Sélénium envoie la demande au pilote du navigateur, qu'il s'agisse du pilote Firefox, du pilote IE, du pilote Chrome.

- ❖ Le pilote de navigateur utilise le serveur HTTP pour obtenir la demande HTTP et le serveur HTTP filtre toutes les commandes à exécuter.
- ❖ Ensuite, les commandes de votre script sélénium seront exécutées sur le navigateur.
- ❖ Enfin, le serveur HTTP renvoie la réponse au script de test d'automatisation.

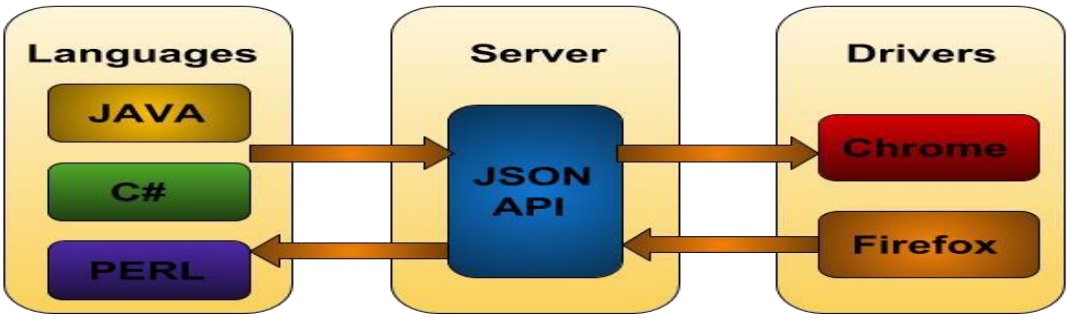


Figure IV.11 : Architecture de sélénium [21].

3. Présentation de l'outil de test

Dans cette partie, nous présentons les interfaces de notre outil de test via des captures d'écrans. C'est-à-dire les interfaces à partir de la création de projet de test jusqu'à l'exécution de tests et les résultats.

Interface de la création de projet de test

Cette interface fournit à l'utilisateur de créer un nouveau projet de test ou bien ouvrir un projet existant, où l'utilisateur peut effectuer de test.

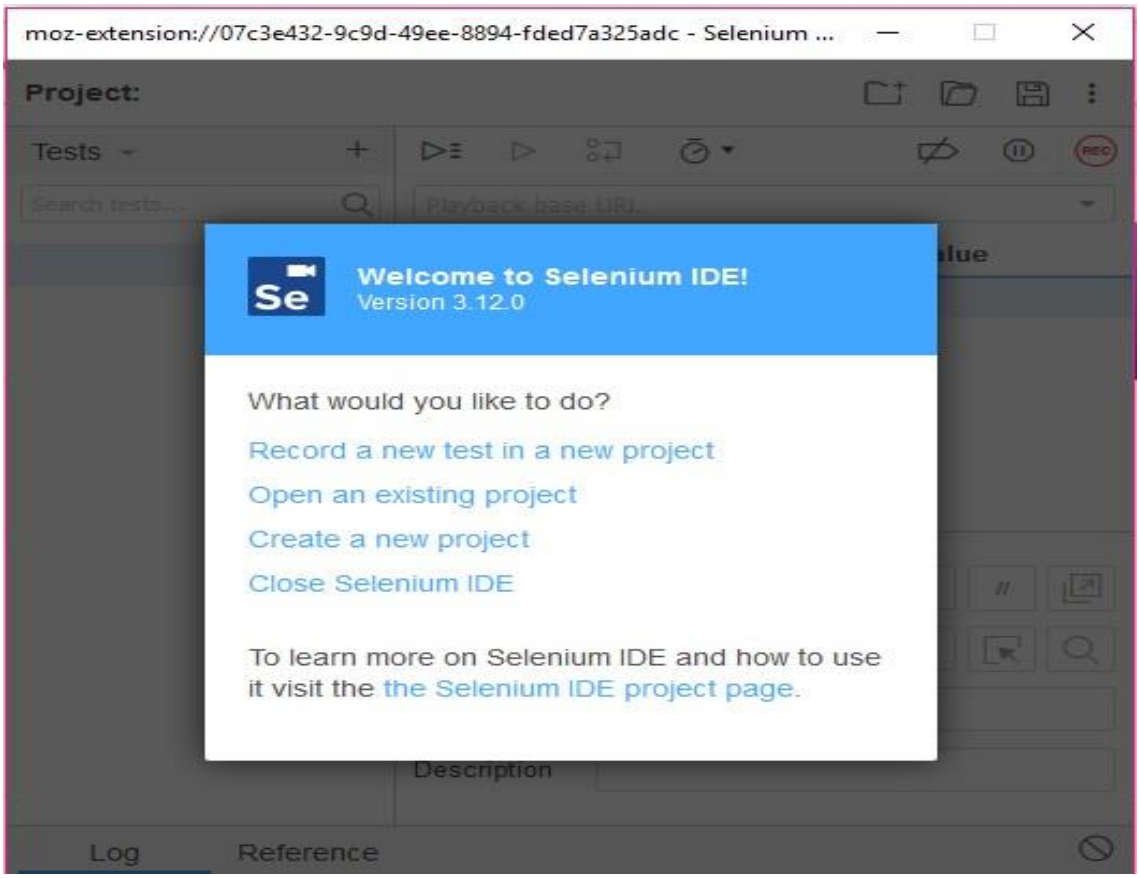


Figure IV.12 : Interface de la création de projet de test (a)

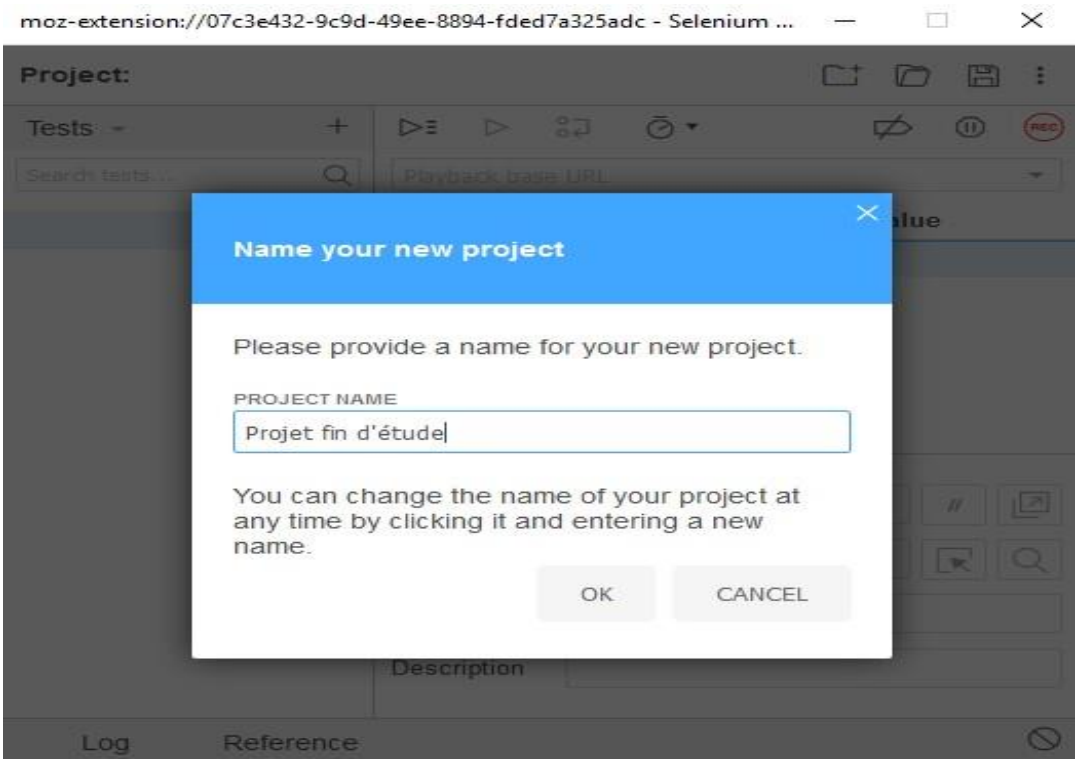


Figure IV.13 : Interface de la création de projet de test (b)

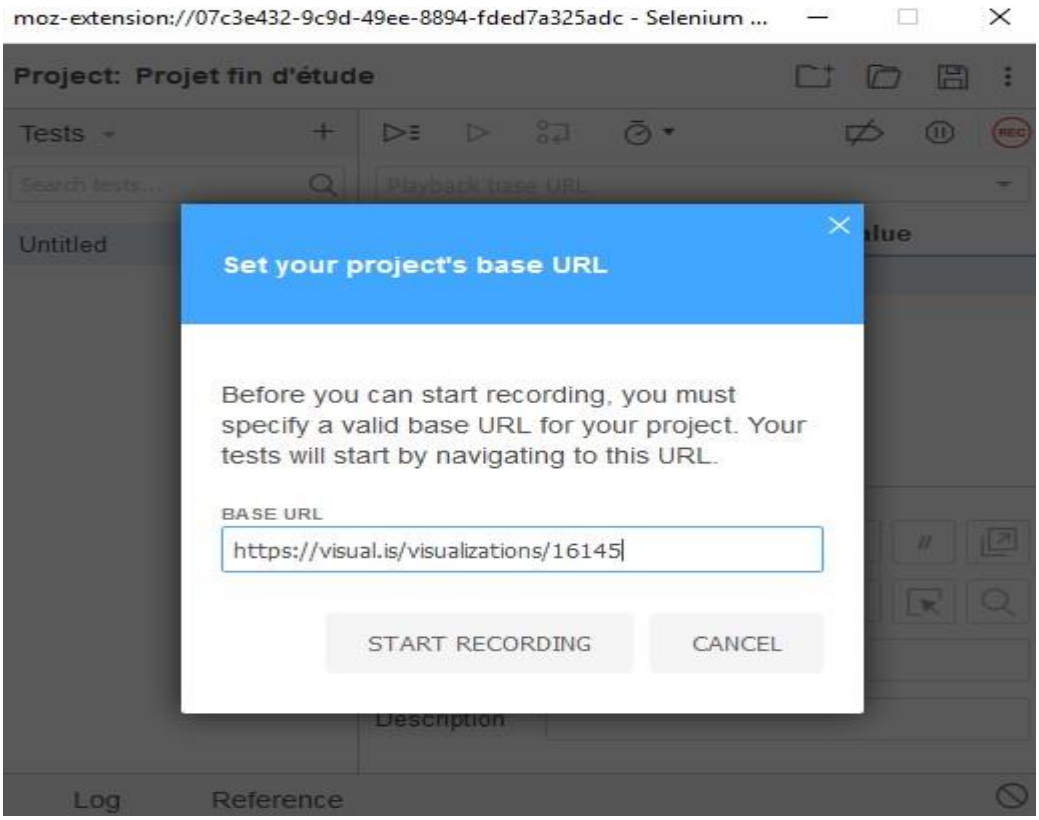


Figure IV.14 : Interface de la création de projet de test (c**)**

4. Interface d'accueil

Comme indiqué ci-dessous, chaque commande comporte quatre champs

- Command - la commande actuelle à exécuter sur la page
- Target - élément sur la page
- Value - valeur à utiliser en cas de commandes comme typeText
- Description - fournir des informations supplémentaires concernant la commande utilisée

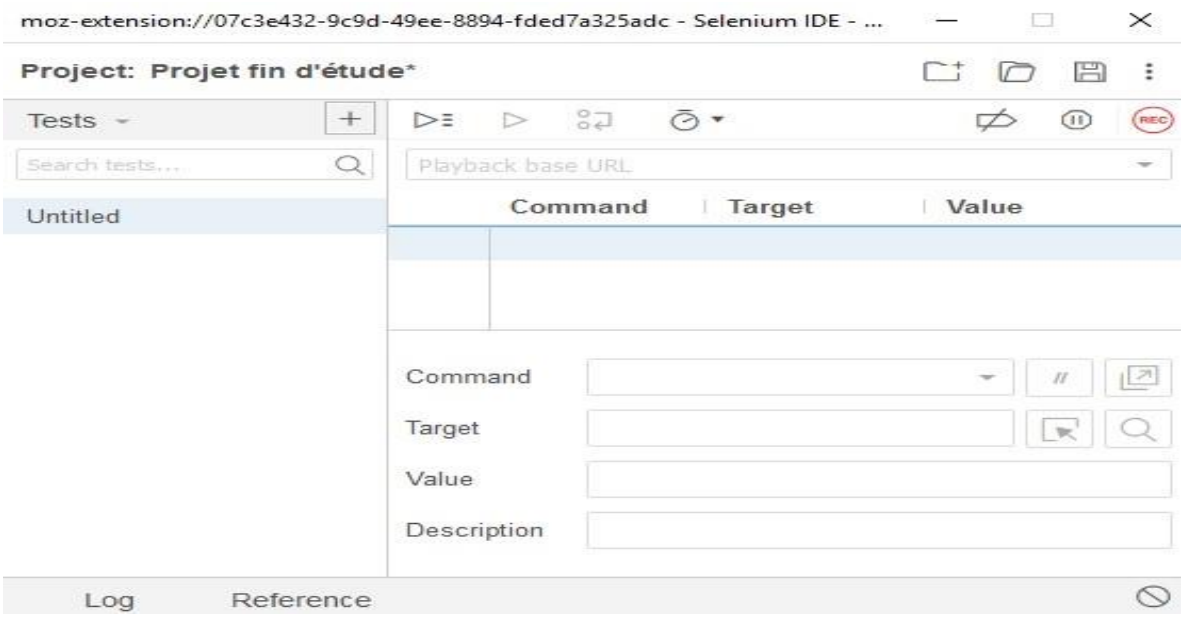


Figure IV.15 : Interface d'accueil

4.1. Interface de la création des cas de test

Cette interface permet de créer, modifier ou supprimer des cas de test :

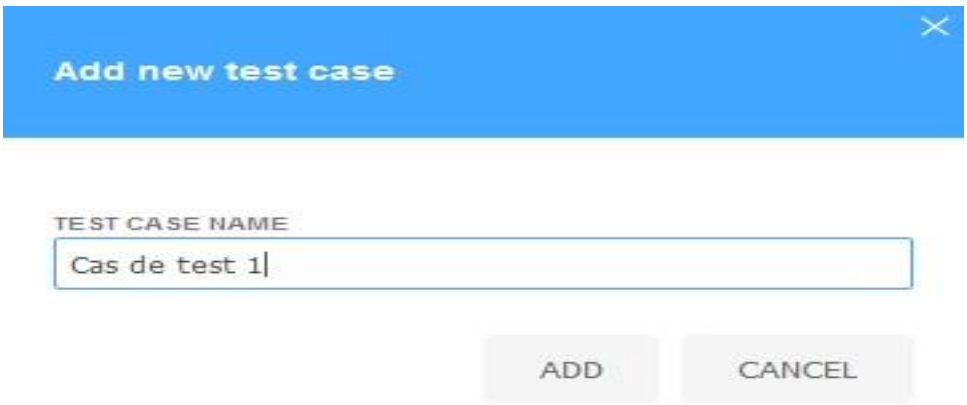


Figure IV.16 : Interface de la création d'un cas de test

4.2. Interface de l'exécution des tests

Pour pouvoir exécuter des tests, le testeur doit d'abord ajouter les cas de tests qu'il veut exécuter, sinon il peut aussi supprimer les cas de tests qu'il ne veut pas. Puis, il doit fournir et insérer la base URL de l'application (tableau de bord dans notre cas) qu'il va tester dans le champ Playback base URL de Sélénium IDE. Après ça, il clique sur le bouton Enregistrer pour pouvoir enregistrer toutes les activités qu'il veut tester sur l'application et ensuite il arrête l'enregistrement. Enfin, il peut démarrer

le test en cliquant sur le bouton Run Test. Le Sélénium IDE va exécuter le test en lançant l'application cible, renvoie le résultat de l'exécution sur la console comme illustré ci-dessous :

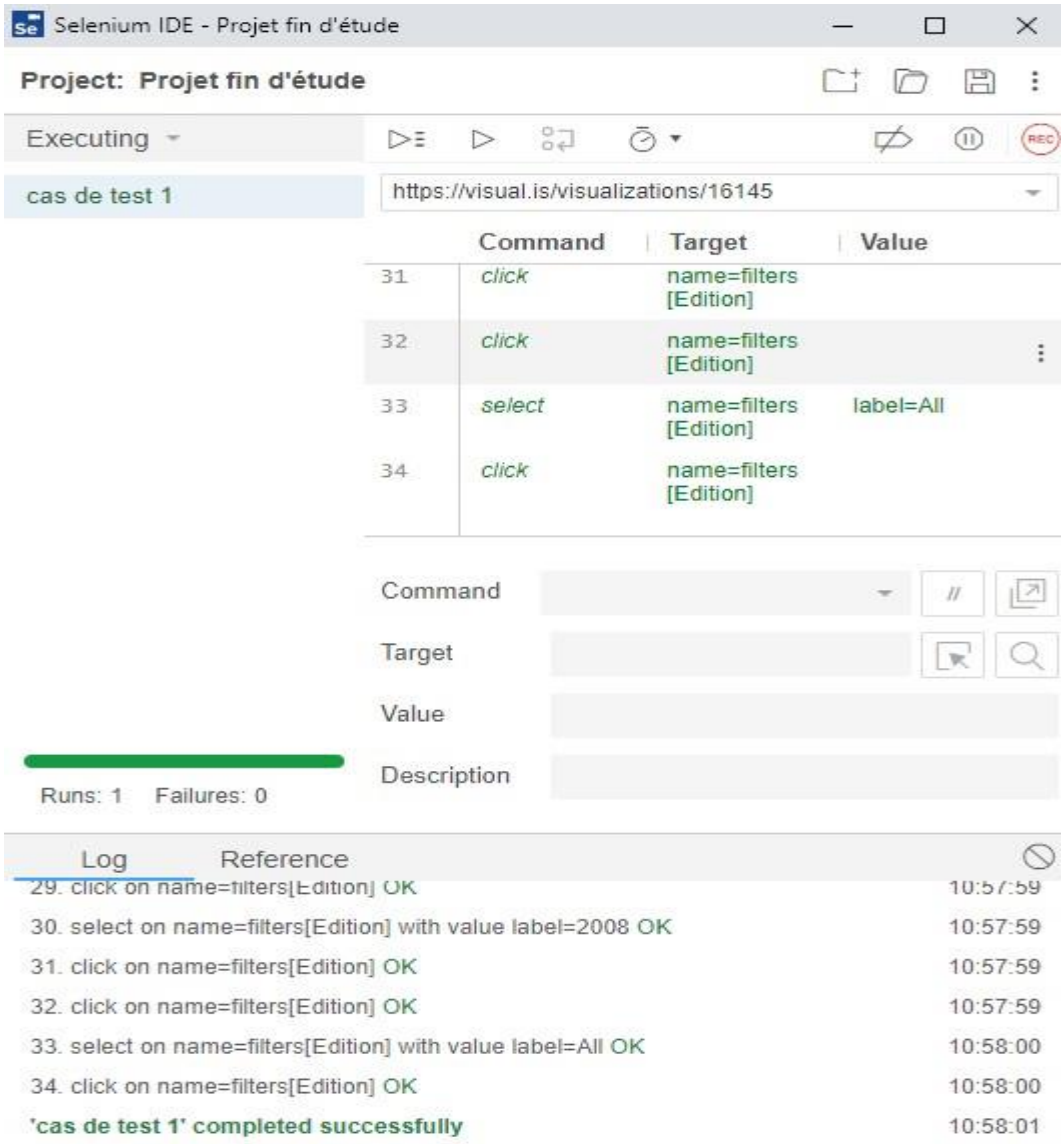


Figure IV.17 : Interface de l'exécution des tests

5. Test et résultats

Nous allons présenter tableau de bord, son plan de test et aussi indiquer les résultats des cas de test exécutés suivant les fonctionnalités de l'application.

5.1. Plan de test de l'application

Le plan de test définit ce que l'on va tester, comment on va le tester mais aussi ce qui ne va pas être testé. Il regroupe les éléments à tester, les fonctionnalités à tester, les types de tests à réaliser ainsi que les risques

associés au plan. Parmi les types de plan de tests, nous avons choisi le plan de test fonctionnel de l'application.

5.2. Stratégie de test de l'application

La stratégie de tests présente l'approche recommandée d'évaluation des cibles tests. Il consiste d'abord à identifier les techniques de tests et à identifier les critères de complétion de tests. Les tests doivent être exécutés dans des environnements sécurisés avec des données connues et contrôlées.

6. Fonctionnalités à tester

Nous allons présenter les fonctionnalités à tester sur le tableau de bord Summer Olympic Games :

- ***Choix d'édition-*** un utilisateur peut sélectionner ou changer l'édition de jeux olympiques d'été en cliquant sur le bouton Edition.
- ***Choix de pays-*** un utilisateur peut choisir le pays participant dans les jeux olympiques d'été en cliquant sur le bouton Country.
- ***Choix de médailles-*** un utilisateur peut sélectionner le type de médaille
  (L'or ou l'argent ou le bronze) remportée par chaque pays participant dans les jeux olympiques d'été.
- ***Les indicateurs de performance-*** un utilisateur peut pointer et vérifier les indicateurs de performance

7. Risques associés au plan de tests

- Si le client n'est pas habitué avec l'outil de test, donc il suffit de définir un guide d'utilisation dans l'outil de test.
- Si l'utilisateur n'est pas compétent pour les tests, donc il suffit de proposer de formation pour lui.

8. Ressources Requises

Les ressources requises sont tout type de ressources indispensables qui permettent à la réalisation du projet. Elles sont les suivants

- Ordinateur – nécessite un ordinateur exécutant Windows 7 ou une version ultérieure, RAM 4 ou plus et un CPU i3 ou plus.
- Application – besoin d'une application tableau de bord.
- Base de données – un serveur de base de données est requis.
- Outil de test – besoin d'un outil de test comme Selenium qui exécute les tests automatiquement et générer les résultats de tests.
- Connexion – une connexion stable de 3G ou 4G est requis

9. Résultats final d'exécution des tests

Nous allons entamer l'exécution des tests pour pouvoir exhiber les fonctionnalités de l'application. Ci-dessous illustre l'exécution des tests :

- **Cas de test 1 :** Nous commençons à ouvrir l'interface de tableau de bord qui est lié avec sélénium.

  A travers le sélénium, nous choisissons une édition (année=1896) et puis nous sélectionnons trois pays (Alegria, United Kingdom et United States) correspondants à cette édition.

  Ensuite nous cliquons sur le bouton Run pour exécuter ces actions qui sont enregistrées par sélénium.

  En cours d'exécution, sélénium examine chaque action enregistrée et donne le résultat correspondant à cette action comme illustré ci-dessous :

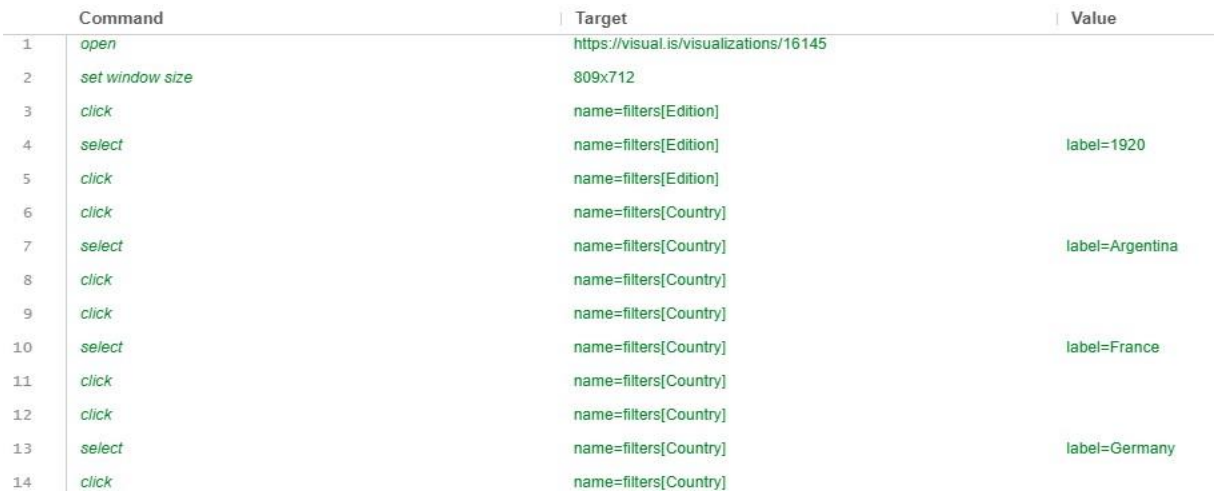
	Command	Target	Value
1	*open*	https://visual.is/visualizations/16145	
2	*set window size*	809x712	
3	*click*	name=filters[Edition]	
4	*select*	name=filters[Edition]	label=1896
5	*click*	name=filters[Edition]	
6	*click*	name=filters[Country]	
7	*select*	name=filters[Country]	label=Algeria
8	*click*	name=filters[Country]	
9	*click*	name=filters[Country]	
10	*select*	name=filters[Country]	label=United Kingdom
11	*click*	name=filters[Country]	
12	*click*	name=filters[Country]	
13	*select*	name=filters[Country]	label=United States
14	*click*	name=filters[Country]	

Figure IV.18 : Exécution de cas de test 1

Figure IV.19 : Résultat de cas de test 1 avec succès

- **Cas de test 2 :** Dans le cas de test 2, nous choisissons une autre édition (année=1920) et puis nous sélectionnons trois autres pays (Argentina, France et Germany) correspondants à cette édition.
  Ensuite nous cliquons sur le bouton Run pour exécuter ces actions qui sont enregistrées par sélénium.

  En cours d'exécution, sélénium examine chaque action enregistrée et donne le résultat correspondant à cette action comme illustré ci-dessous :



	Command	Target	Value
1	open	https://visual.is/visualizations/16145	
2	set window size	809x712	
3	click	name=filters[Edition]	
4	select	name=filters[Edition]	label=1920
5	click	name=filters[Edition]	
6	click	name=filters[Country]	
7	select	name=filters[Country]	label=Argentina
8	click	name=filters[Country]	
9	click	name=filters[Country]	
10	select	name=filters[Country]	label=France
11	click	name=filters[Country]	
12	click	name=filters[Country]	
13	select	name=filters[Country]	label=Germany
14	click	name=filters[Country]	

Figure IV.20 : Exécution de cas de test 2

Figure IV.21 : Résultat de cas de test 2 avec succès

- **Cas de test 3 :** Dans ce test, nous choisissons une édition (année=1980) et puis nous sélectionnons le pays (Madagascar) correspondants à cette édition. En cliquant sur le bouton d'exécution Run, sélénium examine que le pays sélectionné ne se trouve pas dans la base de données qui entraine une erreur dans le résultat d'exécution.

	Command	Target	Value
1	open	https://visual.is/visualizations/16145	
2	set window size	707x675	
3	click	name=filters[Edition]	
4	select	name=filters[Edition]	label=1980
5	click	name=filters[Edition]	
6	click	name=filters[Country]	
7	select	name=filters[Country]	label= Madagascar
8	click	name=filters[Country]	
9	click	name=filters[Medal]	
10	select	name=filters[Medal]	label=Silver
11	click	name=filters[Medal]	
12	select		

Figure IV.22 : Exécution de cas de test 3

Figure IV.23 : Résultat de cas de test 3 avec échec

<u>Remarque</u> : le cas de test 3 a terminé avec un échec parce que nous avons sélectionné un pays (Madagascar) qui ne se trouve pas dans la base de données (Count).

10. Diagnostique du résultat du test

L'exécution des cas de test 1 et 2 a été réussie avec des résultats qui signifient que les cas de test 1 et 2 ont aboutis, à l'exception de cas de test 3 qui a échoué, il représente la fonctionnalité de sélection de pays.

Conclusion : En terminant notre mémoire, nous avons montré l'outil d'automatisation de test et comment il fonctionne, l'application tableau de bord à travers des captures d'écrans des interfaces les plus importantes et aussi l'exécution et le résultat des tests.

En plus, la nécessité de tests fonctionnels automatisés augmente exponentiellement avec la complexité du projet. Les tests automatisés ne font pas perdre de temps, au contraire ils en font gagner, car une fois écrits, les tests peuvent être rejoués à volonté sans prendre de temps supplémentaire aux testeurs.

CONCLUSION GENERALE

De nos jours l'informatique décisionnelle dévient indispensable dans la prise de décision dans la gestion des organisations. La mise en place de fonction de contrôle devient de ce fait une obligation pour les entreprises. Car, elle permet de détecter les erreurs et les risques et les réparer à temps. Et l'outil qui est par excellence indiqué pour cette tâche est bien évidemment le Tableau de bord. Car c'est un document de référence, outil de management et d'aide à la décision et de prévision, permet par ce contenu documenté et structuré d'anticiper les obstacles (alertes, clignotantes) de conduire l'entreprise sur la bonne route avec la meilleure visibilité possible (indicateur de gestion) pour atteindre la bonne destination (respect des objectifs).

Le but de notre travail était de mettre en place une plateforme de test d'automatisation d'un tableau de bord. Ce qui nous a amené à choisir un tableau de bord de test fonctionnel. Et nous l'avons appliqué sur une base de données Summer Games Olympics.

Pour atteindre nos objectifs, nous avons utilisé une plateforme d'automatisation de test fonctionnel (sélénium). Nous avons amplement atteint les objectifs que nous avions fixés au début de notre projet. Et nous avons suivis certaines étapes nécessaires pour aboutir au résultat escompté. Voici ces étapes :

- En premier lieu, nous avons procédé certains tests fonctionnels dans le but de voir les défaillances du logiciel.
- En second lieu, nous avons consacré cette partie au choix du test manuel ou automatique. Nous avons opté pour ce dernier, car il est de loin celui qui permet de faire moins d'erreur. Ce qui n'est le cas du test manuel. Puisqu'il peut conduire à certaines dont : erreur de compétence ou erreur purement humaine induite par inadvertance.

- La troisième partie se porte sur l'outil d'automatisation. L'outil d'automatisation choisi est le Sélénium. Et nous l'avons appliqué sur un tableau de bord avec Summer Games Olympics comme exemple.
- La quatrième étape a été consacrée à la conception de notre outil de test. Et pour cela, nous avons utilisé le diagramme de classe et diagramme de cas d'utilisation pour modéliser notre outil.
- En dernière étape, nous procédé à l'implémentation de notre outil afin de le tester sur un tableau.

L'aboutissement de notre projet s'est fait non pas sans problème. Nous avons eu certains écueils parmi lesquels nous pouvons mentionnés :
Nous nous sommes butés sur un manque criant de documentation, le manque d'expérience dans l'utilisation de l'outil.

Malgré le succès rencontré dans l'exécution de notre projet, il est important de noter qu'en général le tableau de bord automatique présente certaines insuffisances parmi lesquelles, nous pouvons citer quelques-unes : le choix des indicateurs pour assurer la performance et la pertinence du tableau de bord. Nous avons aussi découvert une multiplicité de tableau bord. Cette multiplicité dénote d'un manque de tableau de bord standardisé. Demain des innovations ou amélioration peuvent et doivent être entreprises pour résoudre ce problème. Puisque le manque de tableau standard constitue un problème notamment dans la transmission de manuel standard qui ne pourra que faciliter la formation des utilisateurs.

Bibliographie

[1] Sten Pittet., (2019), Les différents types de tests dans Software Atlassian .Disponible à : https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing [En ligne le 5/Mars.2019].

[2] Serge Braudo, and Alexis Baumann. (sd). Conseiller honoraire à la Cour d'appel de Versailles, Définition de **Logiciel.Disponible à :** https://www.dictionnaire-juridique.com/definition/logiciel.php **[En ligne le 5/Fév.2019].**

[3] Software Testing Fundamentals (STF)!,.(sd),types de tests logiciels .Disponible à : http://softwaretestingfundamentals.com/ [En ligne 2/Janv.2019].

[4] Ideematic,.(2015),culture web,Méthodes Agiles, des méthodes modernes.Disponible à : https://www.ideematic.com/actualites/2015/01/methodes-agiles-definition/ [En ligne le 4/Avril.2019].

[5] Génie logiciel - Cycle de vie,(2012), encyclopédie informatique Comment Ça Marche ,cycle de vie du logiciel.Disponible à : https://web.maths.unsw.edu.au/~lafaye/CCM/genie-logiciel/cycle-de-vie.htm [En ligne le 2/Aout.2019].

[6] Daniel Cohen-Zardi., (2016),SoftFluent,Sas[FR] ,Les meilleures pratiques du test logiciel.Disponible à : https://www.softfluent.fr/blog/societe/Les-meilleures-pratiques-du-test-logiciel [En ligne le 3.Janv/2019].

[7] Rätzmann, M. and De Young, C. (2003). Software testing and internationalization. Salt Lake City :Lemoine International, Inc., pp.49-55.

[8] km. (2012), Web-ntic MOE MOA, Tests et recette, **Procéssus de test selon l'ISTQB.Disponible à :** https://phortail.org/webntic/Processus-de-test-selon-l-ISTQB.html [En ligne le 2/Aout.2019].

[9] Bastien L., (2018), Analytics, OLAP : définition d'une technologie d'analyse multidimensionnelle, Disponible à : https://www.lebigdata.fr/olap-online-analytical-processing [online le 24/Avril.2019].

[10] Guru 99,(2019), Tests d'automatisation Test manuel: quelle est la différence?, Disponible à : https://www.guru99.com/difference-automated-vs-manual-testing.html [online le 18 /Avril.2019].

[11] Vardhan.,(2019), Qu'est-ce que le sélénium? Premiers pas avec Selenium Automation Testing, Disponible à : https://www.edureka.co/blog/what-is-selenium/ [online 3/Aout.2019].

[12] YANA GUSTI.,(2019), Meilleurs outils de test d'automatisation en 2017: frameworks de test d'automatisation,Disponible à : https://geteasyqa.com/fr/blog/best-automation-testing-tools/ [online 3/Aout.2019].

[13] Ganesh Hegde.,(2019), Qu'est-ce que le rapporteur et comment aide-t-il à automatiser les applications angulaires?,Disponible à : https://www.toolsqa.com/protractor/what-is-protractor/ [online le 3/Aout.2019].
[14] ARC Optimizer.,(2019), Top 30 des outils d'automatisation de processus en 2019 - ARC Optimizer,Disponible à : https://arcoptimizer.com/top-30-des-outils-dautomatisation-de-processus-en-2019 [online le 3/Aout.2019].
[15] SMARTBEAR.,(2019),TestComplète Documentation/Informations générales : À propos de TestComplete | Documentation TestComplete,Disponible à : https://support.smartbear.com/testcomplete/docs/general-info/introducing-testcomplete.html [online 3/Aout.2019].
[16] Rahul R Pandya (fondateur de Step2QA, architecte d'automatisation des tests)., (2017), What is some information about Ranorex automation tools? – Quora,Disponible à : https://www.quora.com/What-is-some-information-about-Ranorex-automation-tools [3/Aout.209].
[17] Anshu Ranjan.,(2017), Studio de test /TelerikIntroduction au studio de test Telerik : Introduction au studio de test Telerik,Disponible à : https://www.toolsqa.com/telerik-test-studio/telerik-test-studio-introduction/ [online le 3/Aout.2019].
[18] CLUSIF (Club de la Sécurité des Systèmes d'Informations Français), « Démarche de conception d'un Tableau de Bord qualité appliqué à la sécurité », Juin 1997, P. 9
[19] Josselin A., (2016), E2E Testing : Katalon Studio,Disponible à : http://netforceteam.com/E2ETesting/index.php.[online 13/Aout.2019].
[20] Ankur Jain., (2006), Qu'est-ce que UFT (QTP)? (Test fonctionnel unifié Micro Focus),Disponible à : https://www.learnqtp.com/what-is-qtp/ [online 13/Aout.2019].
[21] Vinod Rebby, Selenium WebDriver Architecture - JournalDev https://www.journaldev.com/25698/selenium-webdriver-architecture [8/oct.2019]
[22] Introduction au Génie Logiciel, https://fr.slideshare.net/guest0032c8/introduction-au-gnie-logiciel [8/oct.2019].
[23] Mémoire Online - Conception et Developpement d'un logiciel de gestion commerciale - Mchangama Ismaila, https://www.memoireonline.com/02/09/2005/m_Conception-et-Developpement-dun-logiciel--de-gestion-commerciale15.html [8/oct.2019].
[24] Software Testing, Test du système - Principes de base du test de logiciel, http://softwaretestingfundamentals.com/system-testing/ [8/oct.2019].