Test d'applications mobiles: Android & iOS - Un guide complet

Sommaire

Introduction

Le marché des applications mobiles est en plein essor, avec des millions d'applications disponibles sur les deux plateformes dominantes : Android et iOS. Pour se démarquer, il est crucial de proposer des applications performantes, fiables et agréables à utiliser. Ce cours vous guide à travers les différents aspects des tests d'applications mobiles, des fondamentaux des tests manuels à l'automatisation avancée.

Partie 1: Comprendre Android et iOS

Avant de plonger dans le monde des tests, il est essentiel de comprendre les spécificités des plateformes Android et iOS.

Android

- Système d'exploitation open source développé par Google.
- Offre une grande liberté de personnalisation et un vaste choix d'appareils.
- Fragmentation importante: De nombreuses versions d'Android coexistent sur le marché, ce qui complexifie les tests de compatibilité [Schéma de fragmentation d'Android].
- Outils de développement principaux: Android Studio, Java, Kotlin.

iOS

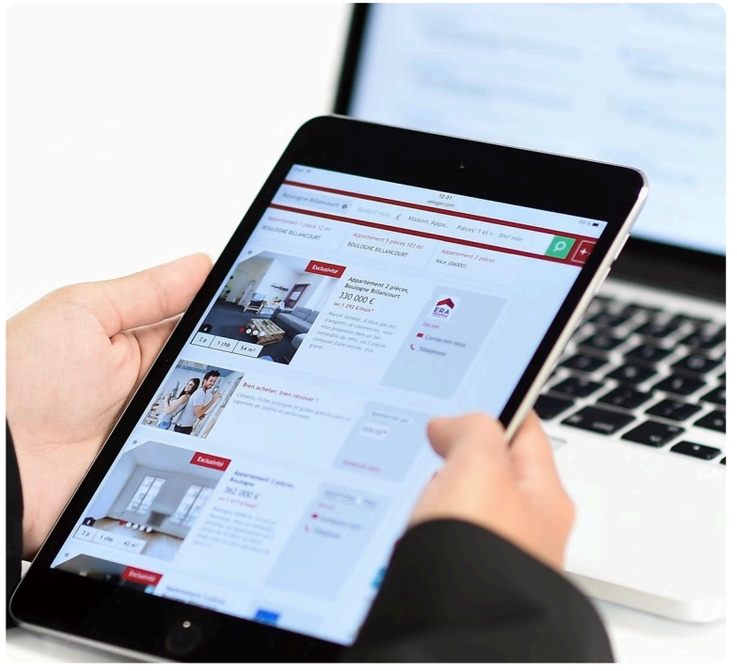
- Système d'exploitation fermé développé par Apple.
- Contrôle strict sur l'écosystème et les applications disponibles sur l'App Store.
- Meilleure homogénéité des appareils et des versions de système d'exploitation, ce qui facilite les tests de compatibilité [Schéma de l'écosystème iOS].
- Outils de développement principaux: Xcode, Swift, Objective-C.

Partie 2: Tests manuels et fonctionnels

Les tests manuels sont la première ligne de défense pour garantir la qualité de votre application.

Types de tests essentiels

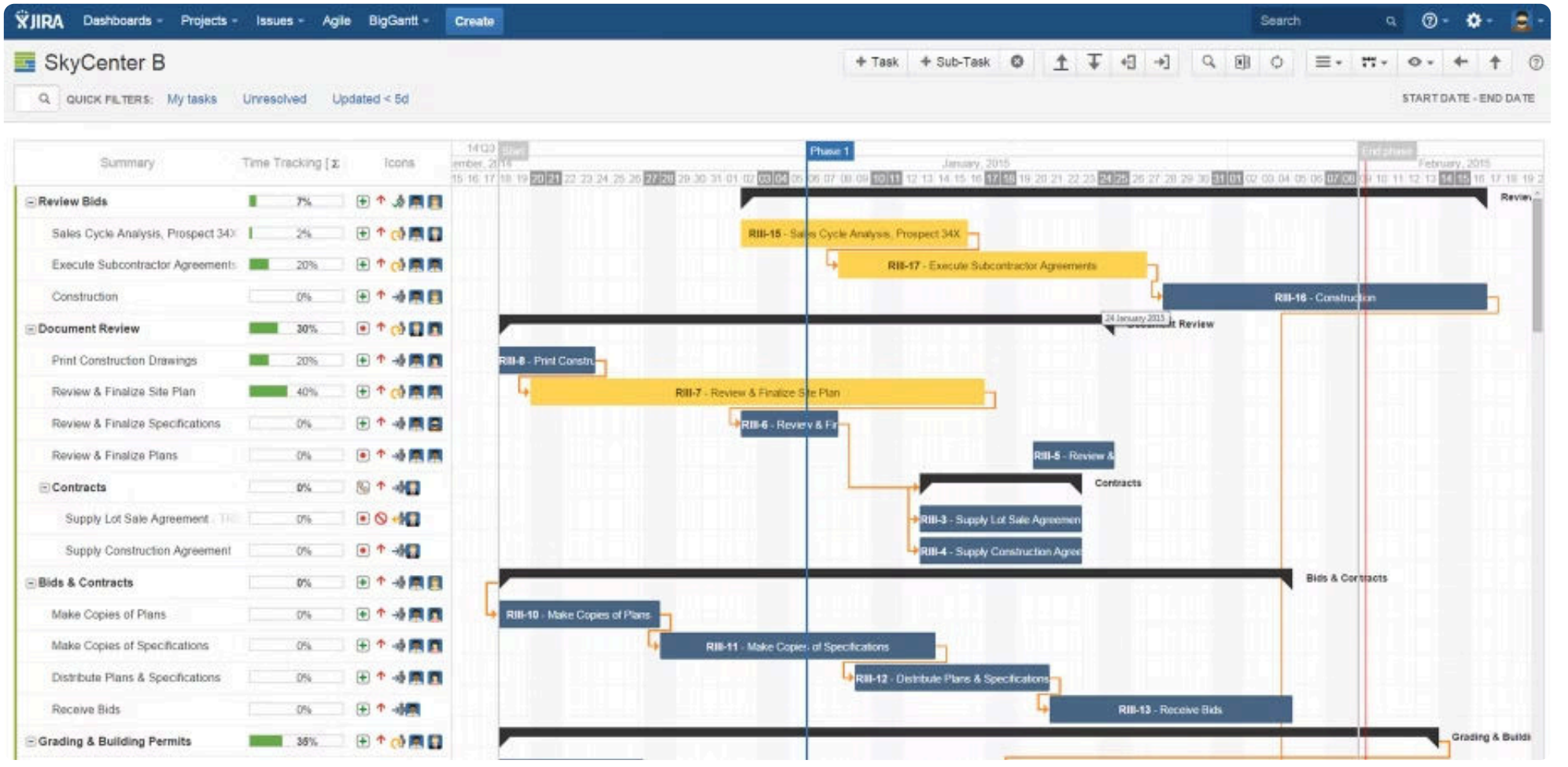
- Fonctionnels: Vérifient si chaque fonctionnalité fonctionne comme prévu (connexion, navigation, achats, etc.).



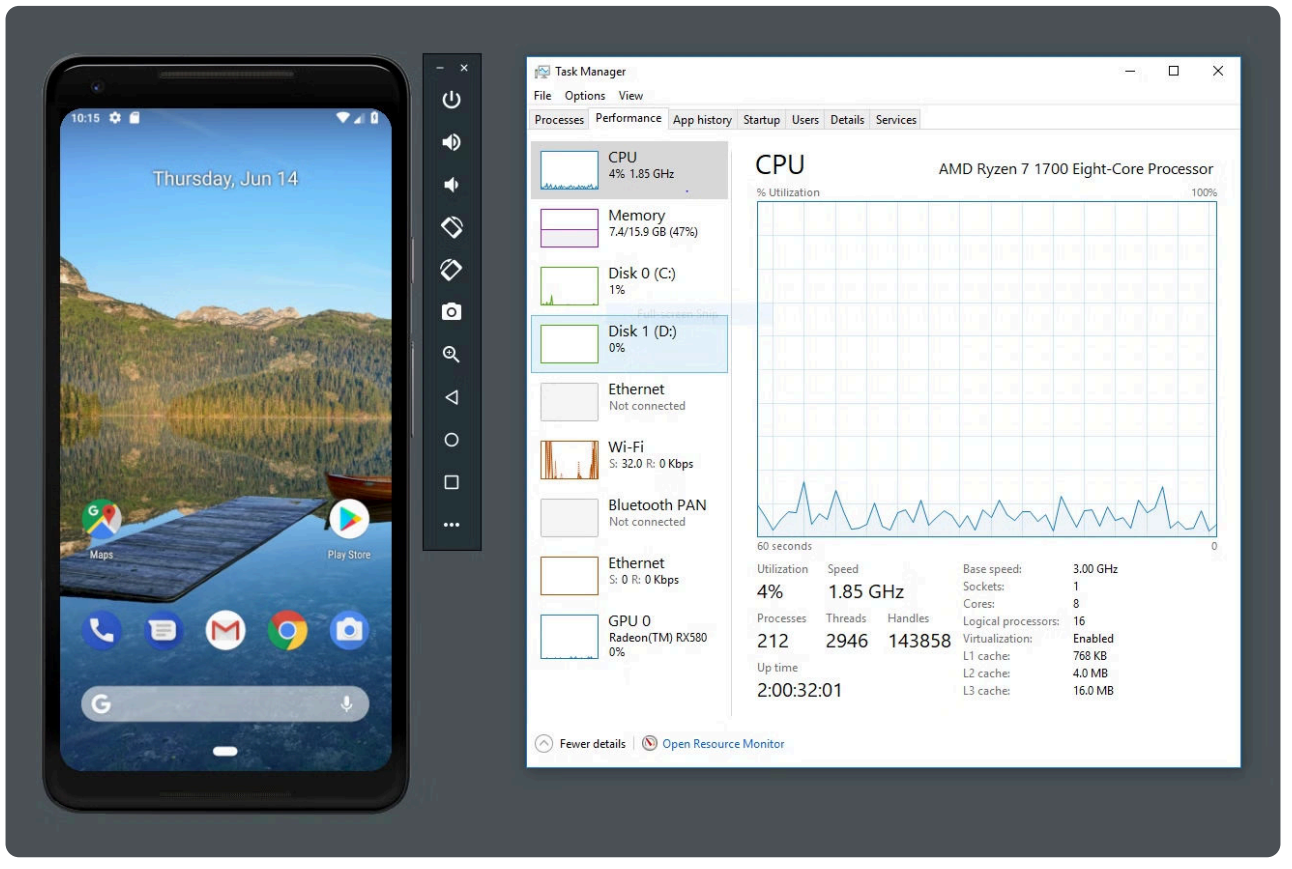
- Interface utilisateur (UI) et ergonomie: S'assurent que l'interface est intuitive, facile à utiliser et respecte les guidelines de chaque plateforme.
- Compatibilité: Vérifient le bon fonctionnement sur différentes tailles d'écran, résolutions et versions de système d'exploitation.
- Performance: Analysent le temps de chargement, la fluidité des animations, la consommation de la batterie et la gestion de la mémoire.
- Interruption: Vérifient la réaction de l'application aux interruptions comme les appels, les SMS, les notifications, etc.
- Installation et mises à jour: S'assurent que l'installation, la désinstallation et les mises à jour se déroulent correctement.

Outils pour les tests manuels

- Plateformes de gestion de projet: Jira, Trello, Asana (organisation et suivi des bugs) .



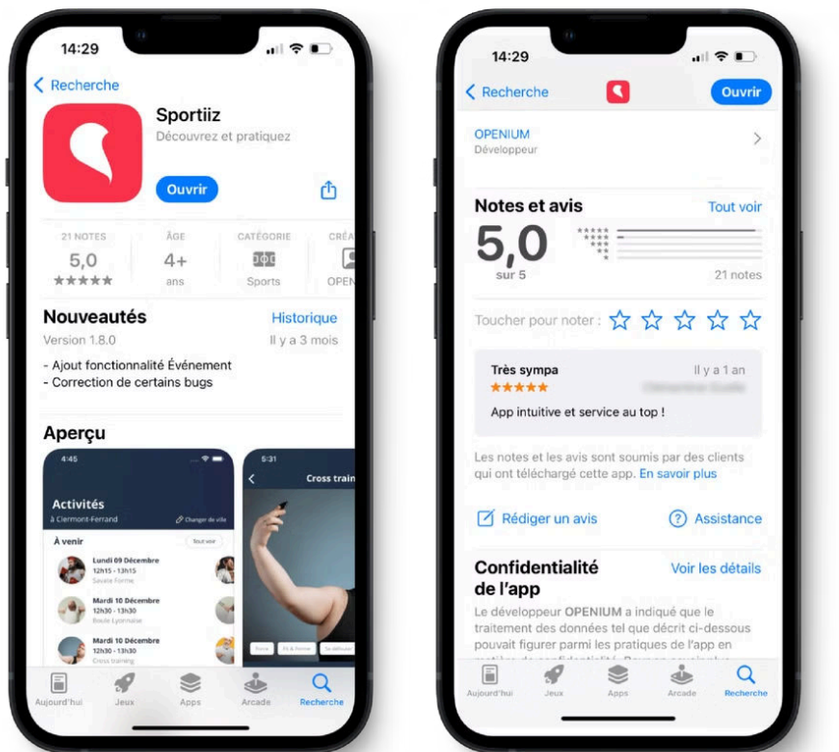
- Outils de capture d'écran et d'enregistrement vidéo: Pour documenter les erreurs et les problèmes visuels.
- Simulateurs et émulateurs: Android Studio Emulator, Xcode Simulator (tests rapides sur différentes configurations virtuelles).



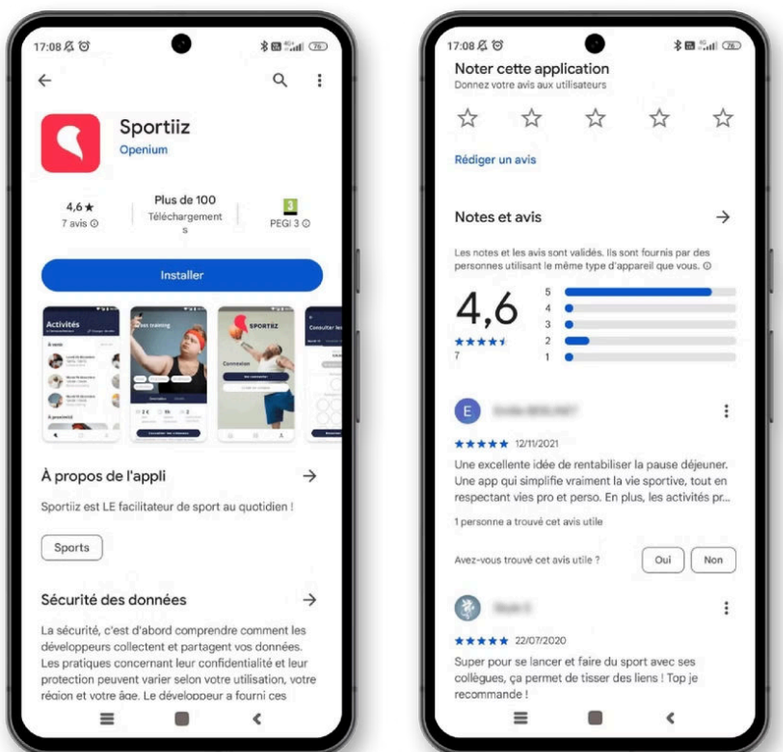
- Appareils réels: Indispensables pour des tests réalistes sur différents modèles de téléphones et de tablettes.

Bonnes pratiques

- Créer des plans de test clairs et concis.
- Définir des cas de test spécifiques et mesurables.
- Documenter les bugs de manière précise et reproductible.
- Impliquer les utilisateurs finaux dans les tests pour obtenir un retour d'expérience précieux.



APP STORE (iOS)



PLAY STORE (ANDROID)

Plan de Test pour une Application Lourde Android (basé sur le cours)

Nom de l'application: [Nom de l'application]

Version: [Numéro de version]

Date: [Date du plan de test]

Numéro du document: [Assigner un numéro unique pour le suivi]

Historique des modifications:

Date	Auteur	Modification
[Date]	[Nom]	[Description de la modification]

1. Introduction

Ce document décrit le plan de test pour l'application [Nom de l'application], une application lourde Android destinée à [bref descriptif de l'application et de son objectif]. Le plan de test suit les recommandations du cours et vise à garantir un produit final fonctionnel, performant et conforme aux attentes.

2. Éléments à tester et objectifs

Fonctionnalité:

- **Tests de composants**: Valider le bon fonctionnement de chaque fonctionnalité de manière isolée (ex: connexion utilisateur, affichage du fil d'actualité, publication de contenu, système de recherche, etc.).
- **Tests d'intégration**: Vérifier l'interaction harmonieuse des différents composants de l'application (ex: publier du contenu et le voir apparaître dans le fil d'actualité, effectuer une recherche et afficher les résultats pertinents, etc.).
- **Tests système**: S'assurer du bon fonctionnement de l'application dans son ensemble, en simulant des scénarios d'utilisation réels et en vérifiant la cohérence des données entre les différentes fonctionnalités.
- **Tests d'acceptation**: Confirmer que l'application répond aux besoins et aux attentes des utilisateurs finaux, en se basant sur des critères prédéfinis et en recueillant les retours des testeurs.

Performance:

- **Temps de chargement**: Mesurer le temps de chargement de l'application au démarrage et lors de l'accès aux différentes fonctionnalités.
- **Fluidité**: Évaluer la fluidité des animations, des transitions et du défilement dans l'application.
- **Consommation de ressources**: Surveiller l'utilisation de la RAM, du processeur et de la batterie lors de l'utilisation de l'application.

Compatibilité:

- **Versions d'Android**: S'assurer que l'application fonctionne correctement sur un large éventail de versions d'Android (à partir d'Android [version minimale] jusqu'à la dernière version stable).
- **Tailles d'écran**: Vérifier l'adaptabilité de l'interface utilisateur aux différentes tailles d'écran et résolutions.
- **Connectivité**: Tester l'application avec différents types de connexions internet (Wi-Fi, 4G, 3G) et en mode hors ligne si applicable.

Sécurité:

- **Authentification**: Vérifier la sécurité du système d'authentification et la protection des données sensibles.
- **Stockage des données**: S'assurer que les données utilisateur sont stockées de manière sécurisée sur l'appareil et lors de la transmission.

Ergonomie et interface utilisateur:

- **Facilité d'utilisation**: Évaluer l'intuitivité de l'interface utilisateur et la facilité de navigation pour les utilisateurs.
- **Esthétique**: Vérifier l'aspect visuel de l'application, la cohérence graphique et l'attractivité de l'interface.

Éléments non testés (avec justification):

[Lister les fonctionnalités ou aspects non testés et expliquer pourquoi]

3. Stratégie de test

Méthodes de test:

- **Tests exploratoires**: Laisser les testeurs explorer l'application librement pour découvrir des bugs et des problèmes d'utilisabilité.
- **Tests basés sur des scénarios**: Définir des scénarios de test précis pour simuler des cas d'utilisation réels et vérifier le bon fonctionnement de l'application.

Méthodes de gestion de projet:

- **Méthode Scrum**: Organiser des sprints de test d'une durée définie, avec des objectifs clairs et un suivi régulier de l'avancement.
- **Cycle en V**: Appliquer une approche séquentielle pour la planification et l'exécution des tests, en validant chaque phase avant de passer à la suivante.

4. Critères de succès et d'échec

Succès:

[Définir les critères de succès pour chaque type de test, par exemple : pourcentage de réussite des tests fonctionnels, temps de chargement maximum acceptable, etc.].

Échec:

[Définir les critères d'échec pour chaque type de test, par exemple: présence de bugs critiques, plantage de l'application, non-respect des exigences de performance, etc.].

5. Ressources

Humaines:

[Lister les membres de l'équipe de test et leurs rôles (ex: Test Manager, Testeur fonctionnel, Testeur de performance, etc.)]

Matérielles:

- Ordinateurs avec accès à un environnement de développement Android.
- Appareils Android réels pour les tests (différents modèles, versions d'Android, tailles d'écran).
- Outils de capture d'écran et d'enregistrement vidéo.

Logicielles:

- Outils de suivi des bugs (ex: Jira, Trello, Github).
- Plateformes de test cloud (ex: Firebase Test Lab, BrowserStack, AWS Device Farm).
- Outils d'automatisation des tests (ex: Selenium, Ranorex, Cucumber, TestComplete, Eggplant).
- Outils de gestion de projet et de collaboration (ex: monday.com, Excel).

6. Planning

[Définir un planning détaillé pour chaque phase de test, en précisant les dates de début et de fin, les jalons importants et les livrables attendus.]

7. Gestion des risques

[Identifier les risques potentiels liés au processus de test (ex: retards de livraison, manque de ressources, problèmes techniques) et proposer des solutions pour les atténuer.]

8. Livrables

- **Rapport de test détaillé incluant**:
  - Description des tests effectués.
  - Résultats des tests (succès/échecs).
  - Description détaillée des bugs rencontrés, avec captures d'écran et étapes pour les reproduire.
  - Recommandations pour améliorer l'application.
- **Matrice de couverture des tests**:
  - Document qui montre quels aspects de l'application ont été testés et à quel niveau de détail.

9. Glossaire

[Définir les termes techniques, abréviations et acronymes utilisés dans le plan de test.]

10. Approbation

[Liste des personnes qui doivent approuver le plan de test avant sa mise en œuvre.]

Résumé

- **Plan de Test** : Document spécifique et détaillé pour un projet particulier, décrivant comment les tests seront effectués.
- **Stratégie de Test** : Document de haut niveau définissant l'approche globale et les principes directeurs des tests au sein d'une organisation ou d'un projet.

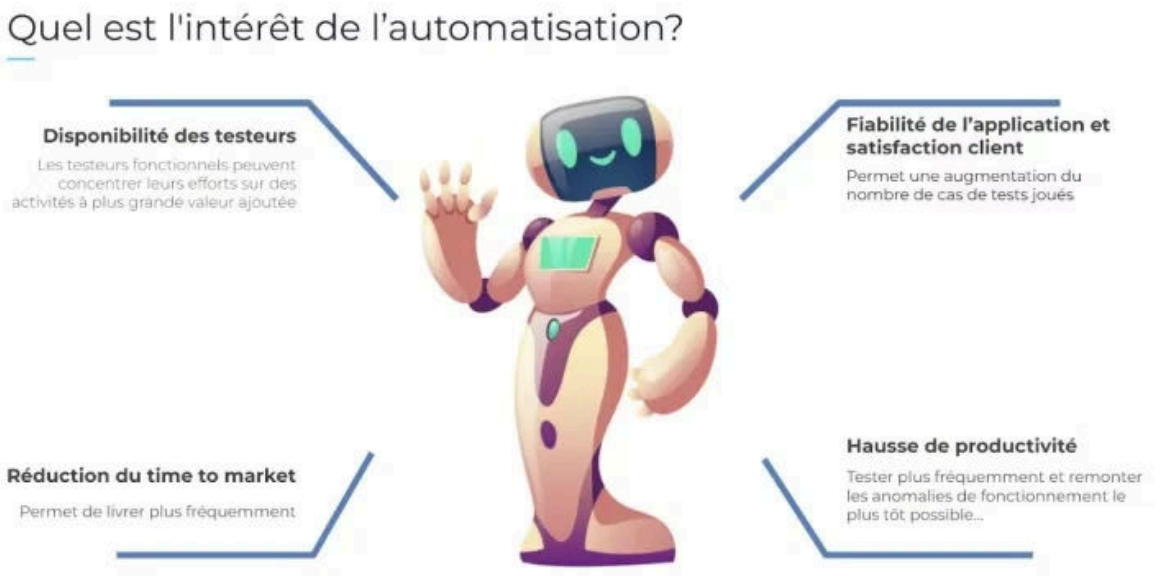
En résumé du Résumé lol :

- **Plan de Test** : Niveau projet, spécifique et détaillé.
- **Stratégie de Test** : Niveau entreprise, général et directif.

Cela permet à l'organisation d'assurer la cohérence et la qualité des tests à travers différents projets tout en s'adaptant aux besoins spécifiques de chaque projet.

Partie 3: Automatisation des tests fonctionnels

L'automatisation des tests est essentielle pour accélérer le cycle de développement et améliorer la qualité globale.

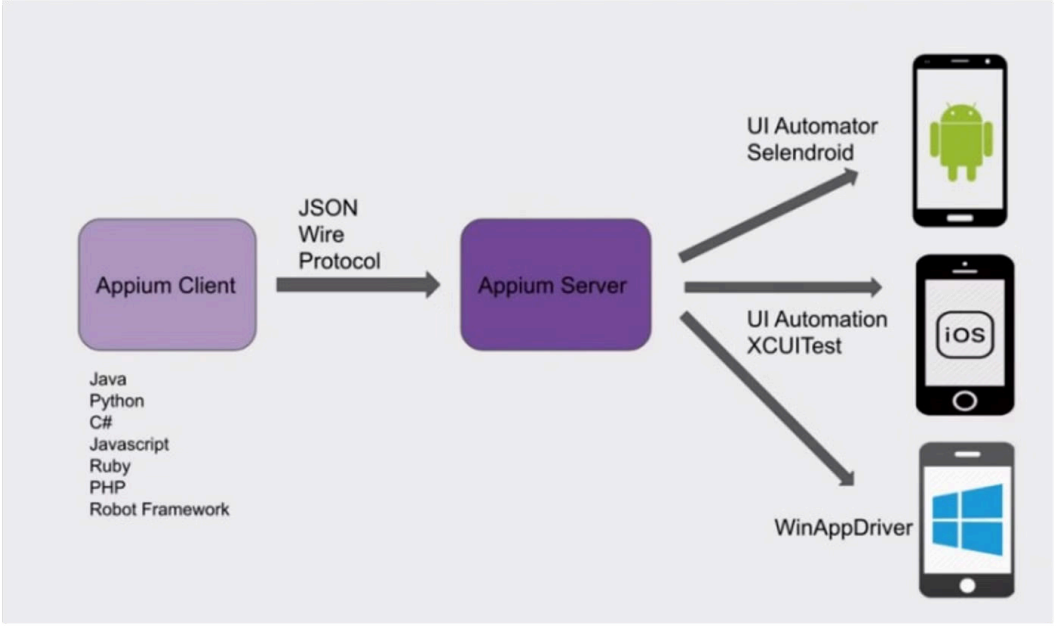


Avantages de l'automatisation

- Rapidité et répétabilité: Exécutez des centaines de tests en quelques minutes, et ce, de manière reproductible.
- Couverture étendue: Automatisez les tests de régression pour vous assurer que les nouvelles modifications n'introduisent pas de bugs dans les fonctionnalités existantes.
- Fiabilité accrue: Éliminez les erreurs humaines potentielles des tests manuels

Frameworks et outils d'automatisation

- Appium: Outil open-source et multiplateforme pour automatiser les tests d'interface utilisateur sur Android et iOS.



- Espresso (Android): Framework de test d'interface utilisateur natif à Android, idéal pour les tests au niveau de l'interface utilisateur.
- XCUITest (iOS): L'équivalent d'Espresso pour iOS, intégré à Xcode.
- Selenium: Principalement conçu pour les applications web, Selenium peut également être utilisé pour tester les parties web des applications hybrides

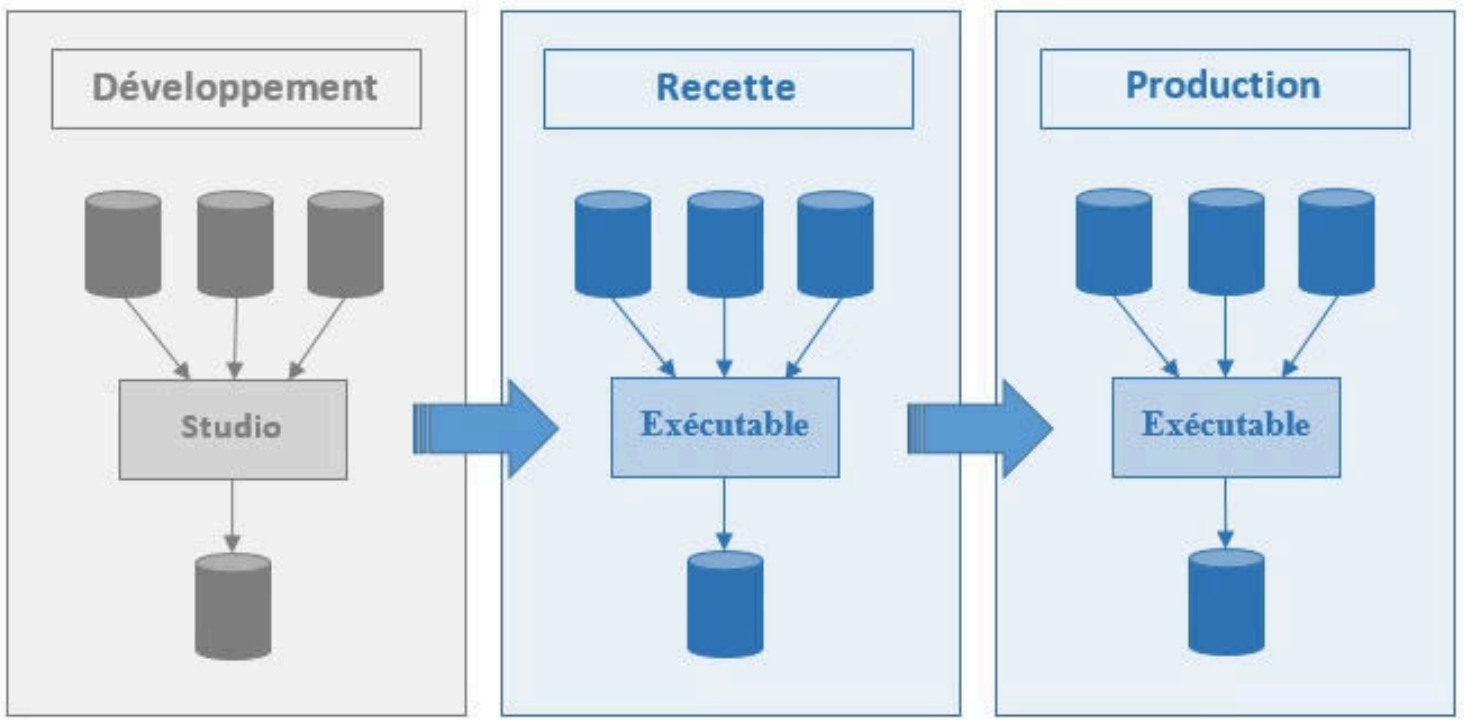
Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, World!"
        android:textSize="18sp"
        android:layout_marginBottom="16dp" />

<Button
    android:id="@+id/element_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click Me"
    android:layout_marginBottom="16dp" />

<EditText
    android:id="@+id/edit_text"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter text here"
    android:layout_marginBottom="16dp" />

<ImageView
    android:id="@+id/image_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher_foreground" />

</LinearLayout>
```

```
from appium import webdriver
from appium.options.android import UiAutomator2Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from appium.webdriver.common.appiumby import AppiumBy

# Définir les options pour Appium
options = UiAutomator2Options()
options.platform_name = 'Android'
options.device_name = 'Emulator'  # Remplacez par le nom de votre émulateur
options.app = 'C:\\Tools\\Python\\Appium\\m.apk'  # Assurez-vous que le chemin vers le fichier APK est correct
options.automation_name = 'UiAutomator2'

# URL du serveur Appium
url = 'http://127.0.0.1:4723'

# Créer une instance du driver Appium
driver = webdriver.Remote(command_executor=url, options=options)

# Attendre que l'élément soit visible et interagir avec lui
wait = WebDriverWait(driver, 20)
element = wait.until(EC.visibility_of_element_located((AppiumBy.ID, 'com.example:id/element_id')))
element.click()

# Effectuer d'autres actions
# ...

# Fermer le driver
driver.quit()
```

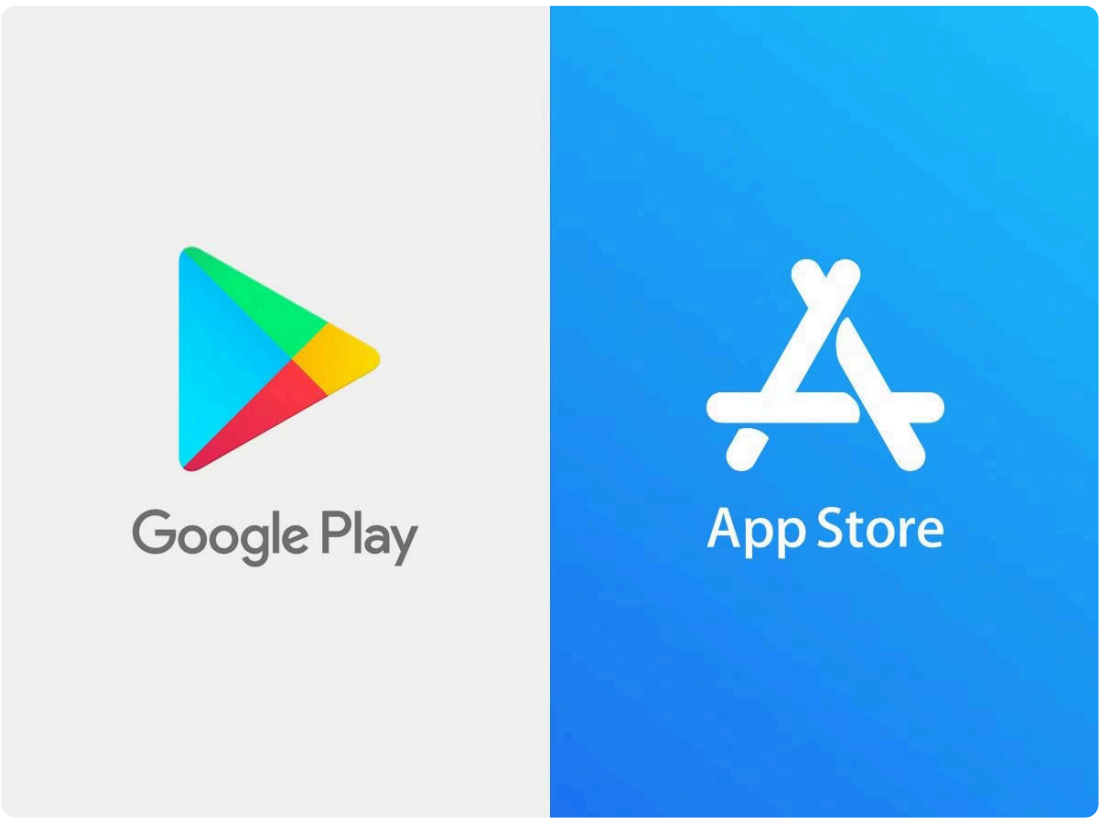
Défis et solutions

- Maintenance des tests: Les applications mobiles évoluent constamment. Concevez des tests robustes et faciles à maintenir.
- Gestion des environnements de test: Assurez-vous que vos tests fonctionnent correctement sur différentes versions de système d'exploitation et sur différents appareils



Partie 4: Déploiement et tests en conditions réelles

Le processus de déploiement et les tests en conditions réelles sont essentiels pour garantir une expérience utilisateur optimale.

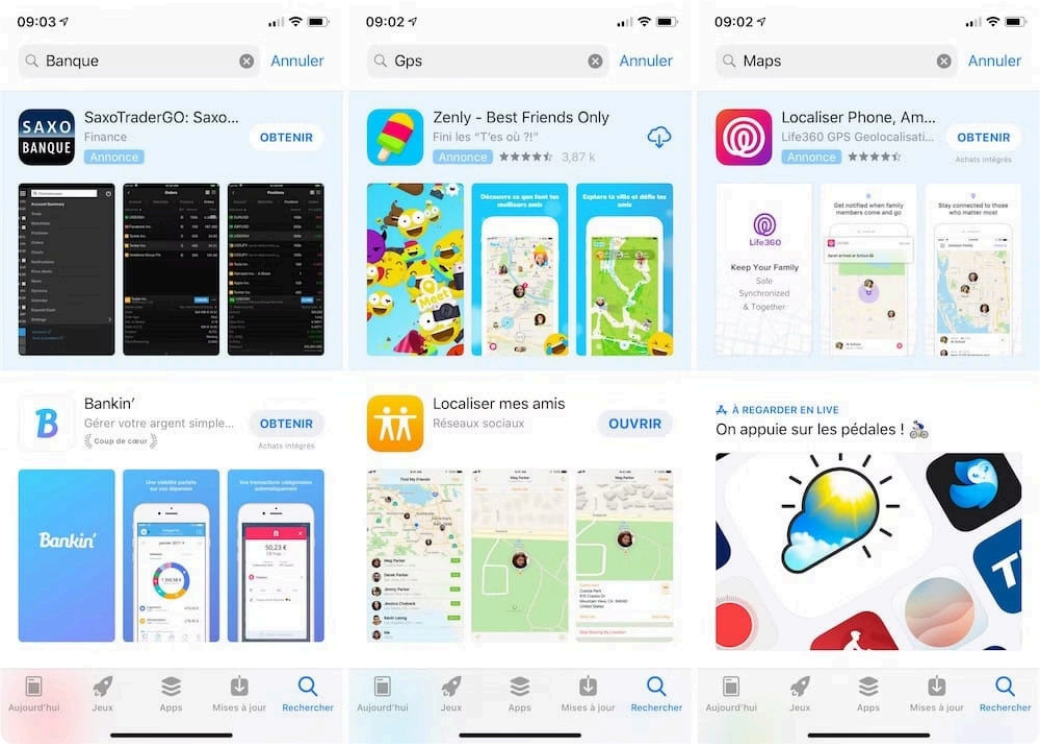


Déploiement sur Google Play Store (Android)

- Créer un compte développeur Google Play (**25 USD**).
- Préparer les éléments de l'application (APK, captures d'écran, description, etc.).
- Tests bêta: Si vous venez de créer un compte de développeur personnel, **vous devez exécuter un test fermé pour votre application auprès d'au moins 20 testeurs inscrits depuis au moins 14 jours sans interruption**.
- Soumettre l'application à Google Play pour revue.

Déploiement sur l'App Store (iOS)

- S'inscrire au programme Apple Developer (**299$/an**).
- Utiliser Xcode pour configurer les informations de l'application et générer un profil de provisioning.
- Soumettre l'application à l'App Store pour revue.
- TestFlight: Service d'Apple permettant de distribuer des versions bêta à des testeurs internes ou externes .



Outils d'analyse et de suivi

- Firebase (Google): Suite complète d'outils pour le développement, le suivi des performances, l'analyse des données et la monétisation des applications.
- App Annie: Offre des données sur le marché des applications mobiles et des analyses de la concurrence