NUMÉRO 3 : JUIN 2026

QUALITÉ & TESTS

DOSSIER : TESTS MOBILES

APPIUM, ESPRESSO, DETOX

INTERVIEW D'OLIVIER DENOO : L'ISTQB SANS DÉTOUR

LE MAG DU TESTEUR

LES TESTS MOBILES

MOT DE L'EDITRICE

Tester une application mobile, ce n'est pas seulement exécuter des scénarios ou choisir un framework d'automatisation. C'est aussi composer avec la fragmentation des appareils, la diversité des systèmes, les contraintes matérielles, les interruptions, les conditions réseau… et surtout avec l'usage réel des utilisateurs. Le mobile nous oblige à sortir d'une vision trop théorique du test pour revenir à ce que vit réellement l'utilisateur sur son appareil, dans son contexte, avec ses contraintes.

Dans ce numéro, j'ai voulu proposer ce nouveau contenu : construire un laboratoire de test mobile, comprendre la fragmentation, cibler les bons devices, tirer parti des fermes d'appareils, adapter sa stratégie par rapport au web, et relier le référentiel ISTQB® à la réalité du terrain.

Mon objectif n'est pas de fournir une recette unique, mais des repères utiles pour tester avec plus de justesse, de recul et de pertinence. Et d'ailleurs, ce n'est toujours que mon point de vue.

Je souhaite aussi adresser un merci particulier à mon réseau, qui a participé à sa manière à la construction de ce numéro. Le sommaire a été imaginé avec votre aide, à travers vos idées, vos retours et vos suggestions. Cette contribution collective donne encore plus de sens à ce magazine, qui a vocation à faire circuler les pratiques, les expériences et les réflexions autour de la qualité logicielle.

J'espère que ce numéro vous apportera des idées, des pistes concrètes et l'envie de continuer à faire progresser nos pratiques de test mobile.

Un immense merci à Caroline Nazin pour sa relecture et ses retours d'experte et Romuald Lenormand pour ses conseils sur la structure du mag (changements à venir pour le prochain numéro !)

Fanny Velsin

SOMMAIRE

6 Définir sa stratégie de test mobile

7 Identifier les plateformes et devices à cibler

12 Adapter sa stratégie mobile vs web

16 Référentiel ISTQB® appliqué au mobile

38 Interview d'Olivier Denoo : L'ISTQB sans détour

41 Panorama du test mobile

42 Se construire un labo de test mobile

45 Comprendre la fragmentation mobile : android, iOS, tailles d'écran, OS, ...

49 Solutions de fermes d'appareils (Sauce Labs, BrowserStack, Kobiton)

62 Outils et frameworks d'automatisation

63 Maestro : écrire des scénarios lisibles et légers

64 Appium, Espresso, XCUITest, Detox

73 Eggplant

75 Accessibilité et qualité d'usage

76 Importance du tab order, Lecture d'écran : TalkBack, VoiceOver, ...

81 Bonnes pratiques

82 Structurer les scénarios pour plus de clarté

84 Gérer les données et états d'application

85 Inclure les tests d'accessibilité dans le cycle

86 Tendances et perspectives

87 Multiplateforme

88 IA et test mobile

89 Le futur du métier de testeur mobile

Informations éditoriales

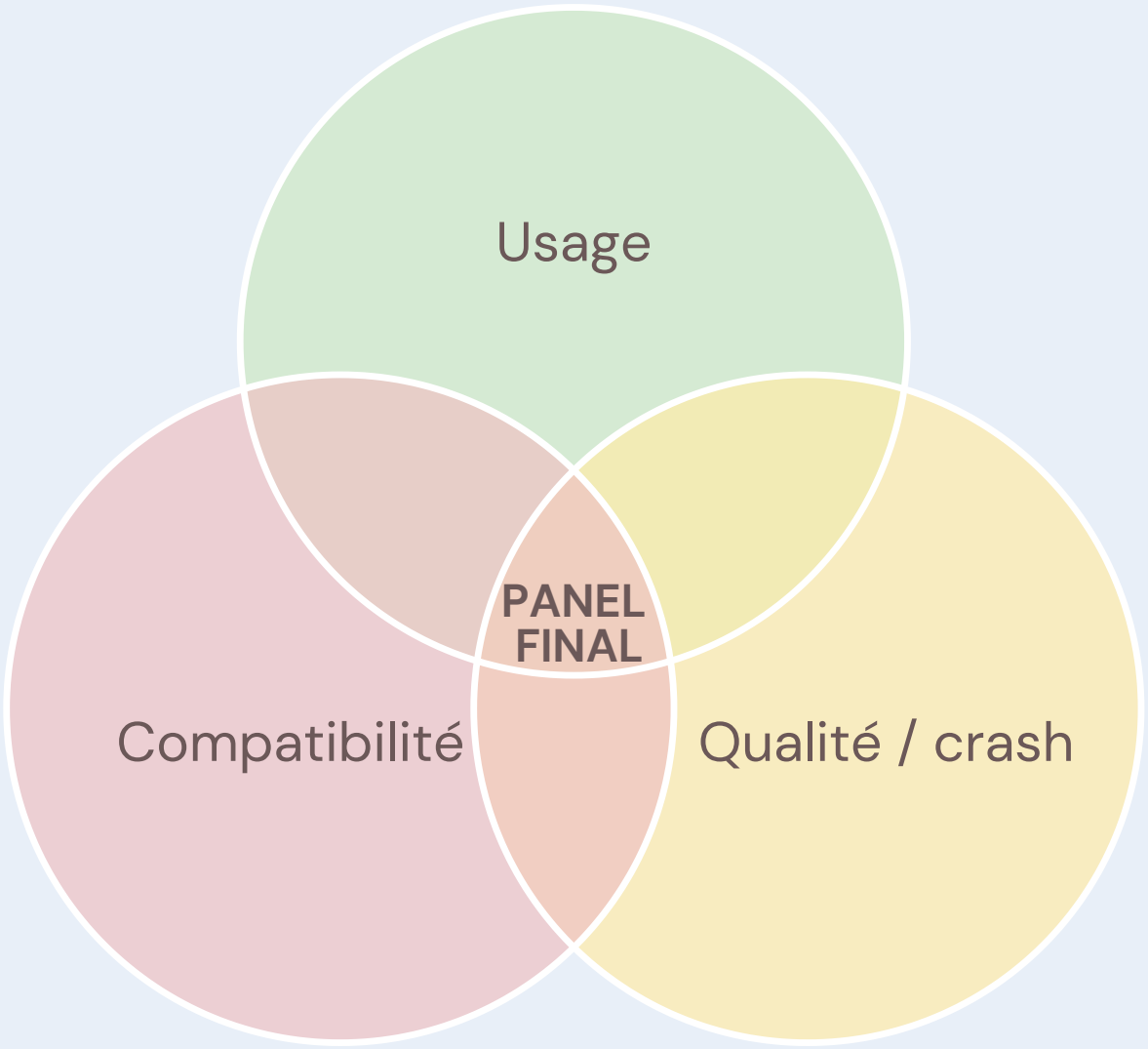
Le mag du testeur est un magazine numérique indépendant consacré à la qualité logicielle, aux pratiques de test et aux retours d'expérience du terrain.

Directrice de publication et rédactrice : Fanny Velsin
Site : lemagdutesteur.fr
Contact : feedback@lemagdutesteur.fr
Conception du site : Juliet HOUDART - HOULACOM
ISSN : demande en cours auprès de la BnF
Diffusion : magazine numérique gratuit

DÉFINIR SA STRATÉGIE DE TEST MOBILE

IDENTIFIER LES PLATEFORMES ET DEVICES A CIBLER

Ne testez pas tous les téléphones. Choisissez un panel qui maximise la couverture utilisateur + risque avec un effort réaliste. La méthode la plus robuste consiste à croiser 3 sources : usage réel, qualité/crash, capacité marché/compatibilité.



L’usage réel

Si votre application est déjà en ligne, vous pouvez collecter les données sur plusieurs jours. Partez sur une plage d’au moins 30 jours.

Vous allez collecter :

- l’OS : iOS ou Android
- les version d’OS : par exemple iOS 17 ou Android 15
- le type de device : tablette ou téléphone
- les principaux modèles avec les tailles d’écrans, si si ça a de l’importance !

Comment retrouver ça?

Pour iOS, vous pouvez retrouver les informations sur l’App Store Connect. App Analytics permet d’utiliser des dimensions et filtres (ex. device type, territory, et des dimensions comme OS version).
Source:
https://developer.apple.com/help/app-store-connect/view-app-analytics/view-app-metrics

SIGNATURE DU MAG

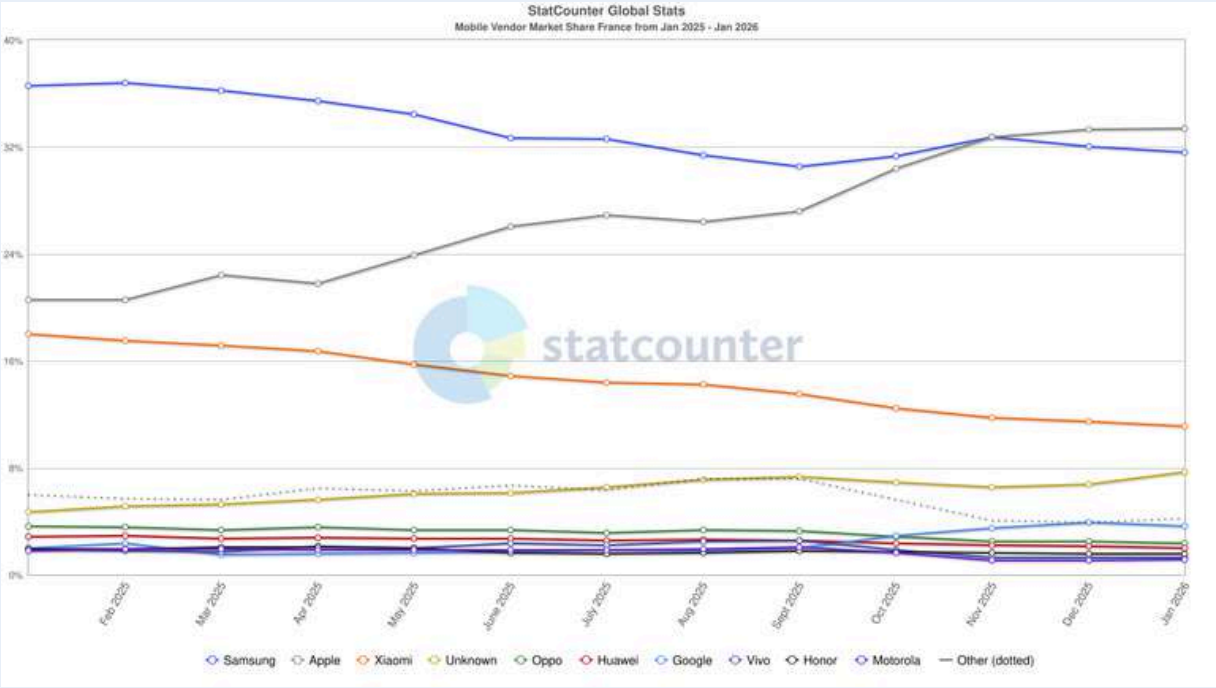
Décision QA

Prenez les devices/versions qui couvrent 80-90% de vos sessions. Vous pouvez réserver le reste pour des tests exploratoires.

Pour android, Google play console dans le device catalog vous permet de voir les appareils disponibles ou compatibles et de comparer installs, revenus par device.
Source :
https://play.google.com/console/about/devicecatalog

Si votre application n’est pas en ligne, vous pouvez vous aider de site comme : https://gs.statcounter.com/vendor-market-share/mobile qui vous donne les statistiques des mobiles par pays par exemple ou sur toute la planète. Vous pouvez également filtrer sur une période.

Si nous prenons la période des 12 derniers mois pour la France, cela donne :



AVIS D'EXPERTE

Caroline Nazin

🪲 </Quality Engineer Web> & 👩‍💻 </Cheffe de projet fonctionnel> · Luxe & E-commerce international · AI-Powered QE · Automation

Le test mobile hors Android/ iOS classiques

Précédemment, **t**u t'appuies sur StatCounter France, Google Play Console et App Store Connect pour choisir les devices à tester. C'est logique mais c'est très occidental.

Chez nous on couvre plus de vingt stores internationaux, et je me rends compte que sur certains marchés le vrai point d'entrée ecommerce n'est même pas l'app native.

En Chine c'est souvent un MiniProgramme WeChat ou Alipay. Rednote fait de la découverte produit, Douyin fait du paiement in-stream, Wosai passe par un QR code scanné par l'app bancaire. Pareil en Corée avec KakaoTalk, NaverPay et KakaoPay préinstallés, au Japon avec LINE et Amazon Pay qui dominent. Cette réalité concerne pas mal de projets internationaux.

Il n'empêche que en effet nous utilisons également les données GA4 et Geotracking pour mieux savoir quels device sont utilisés réellement.

La qualité

Le but est d'inclure des devices ou des versions qui sont vos "peines" même si ce ne sont pas ces devices qui sont le plus utilisés.

Il faut donc collecter :

- les top modèles/OS qui concentrent vos crashes / ANR / incidents
- les parcours associés aux crashs (par exemple, le login, le paiement, ...)

Pour cela, vous pouvez par exemple utiliser : https://firebase.google.com/products/crashlytics

ou
https://sentry.io/solutions/mobile-developers/

SIGNATURE DU MAG

Décision QA

Ne ciblez pas seulement les appareils les plus populaires.

Ciblez aussi ceux qui :

- concentrent les incidents,
- exposent les parcours critiques,
- représentent vos plus gros risques qualité.

Le bon réflexe peut être de tester un device peu utilisé s'il concentre une forte part des crashs ou touche un parcours critique.

Application figée : Android et iOS

ANR pour Application Not Responding. C'est un événement Android qui se produit quand une appli ne répond plus pendant un certain temps (l'UI est bloquée), et Android peut afficher une boîte de dialogue "L'application ne répond pas".

Sur iPhone, on parle plutôt de :

- hang : une période pendant laquelle l'app devient non réactive / figée. Apple définit un hang comme une période d'absence de réponse perceptible lors du traitement d'une interaction utilisateur.
- watchdog termination : si l'app bloque trop longtemps le main thread ou met trop de temps à se lancer/répondre, iOS peut la tuer automatiquement via le watchdog.

Pour les mordus, ce petit jeu de mots en hexadécimal : "ate bad food". Apple l'associe aux watchdog terminations dans sa documentation développeur.

0X8BADF00D ("ATE BAD FOOD")

La diversité et la compatibilité

Le but est de couvrir la diversité Android + vérifier que votre application est installable/compatible sur le terrain. Il faut couvrir les “angles morts”

Pour android, vous pouvez vous baser sur des distributions fiables. Le “tableau de bord de la distribution” fournit des distributions par version Android, RAM, SoC, ABI, métriques d’écran, etc., avec historiques et exports CSV.

C’est très utile pour choisir des devices “représentatifs” :

- entrée de gamme (RAM faible)
- milieu/haut de gamme
- petit écran / grand écran
- ABI / SoC différents (si votre application est sensible perf/graphique)

Compatibilité & exclusions (Google Play)

- La doc explique comment contrôler quels devices ont accès à l’app (features requises, filtrage…) et comment se préparer à atteindre la bonne audience.
- Le support Play recommande de revoir régulièrement les listes “supported/excluded devices” dans le Device catalog.

ABI, ça veut dire quoi ?
ABI = Application Binary Interface
Côté Android, Google explique que chaque combinaison CPU + jeu d’instructions possède sa propre ABI. L’ABI définit notamment comment le code machine de l’app interagit avec le système à l’exécution.

Exemples d’ABI Android supportées par le Native Development Kit (NDK) :

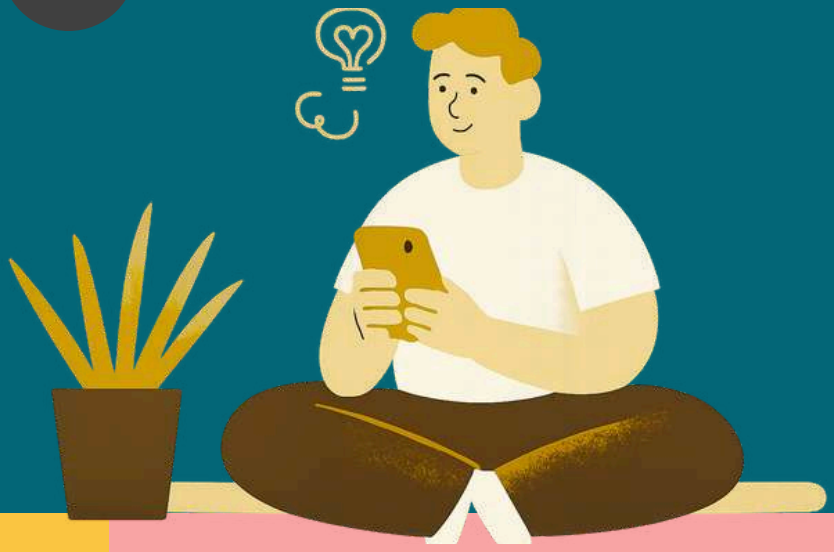
- armeabi-v7a
- arm64-v8a
- x86
- x86_64

SoC, ça veut dire quoi ?
SoC = System on Chip
Deux téléphones Android avec la même version d’OS peuvent se comporter différemment si leur SoC est différent.

Pourquoi c’est utile en tests ?
Parce que le SoC peut influencer :

- les performances
- la chauffe
- l’autonomie
- certains comportements graphiques / vidéo / IA / photo
- parfois des bugs très device-specific

WEB VS MOBILE



01. COUVERTURE

On cible surtout navigateurs, versions et responsive

02. CONTEXTE

Environnement plus prévisible, mais variable selon le navigateur, la résolution, la session et les extensions.

03. PERFORMANCE

on surveille souvent le temps de réponse et le rendu

04. PRIORISATION

Parcours critiques, navigateurs les plus utilisés, pages à fort impact

01. COUVERTURE

On cible les combinaisons OS, device et contexte d'usage

02. CONTEXTE

l'utilisateur reçoit des notifications, perd le réseau, change d'orientation, refuse une permission, passe sur une autre application puis revient plus tard.

03. PERFORMANCE

il faut aussi surveiller la fluidité, la chauffe, la batterie, la mémoire et la stabilité dans le temps.

04. PRIORISATION

Comme il est impossible de tout tester, la stratégie mobile doit être encore plus guidée par le risque, l'usage réel et les retours terrain.

AVIS D'EXPERTE

Caroline Nazin

🪲 </Quality Engineer Web> & 👩‍💻 </Cheffe de projet fonctionnel> · Luxe & E-commerce international · AI-Powered QE · Automation

Cross-device journey :

Chez nous sur de l'horlogerie luxe, personne ne valide une montre à 5000 euros sur son téléphone dans le métro. L'utilisateur découvre sur mobile, met en favoris, et finit sur desktop le soir.

Du coup les vrais tests qu'on doit faire c'est la persistance du panier entre devices, la wishlist synchronisée, le SSO app vers web, les deep links conditionnels selon que l'app est installée ou pas.

J'ai eu un bug cette année de panier vidé au passage mobile vers desktop à cause d'un cookie mal scopé, détectable uniquement en testant la bascule explicitement.

Il y a une fragmentation des devices mais aussi une fragmentation du parcours à travers les devices.

Idem pour le côté culturel, en chine ils sont full mobile mais avec une pratique de navigation et habitudes bien différente de la nôtre donc cela peut justifier un travail de pairing avec des collègues locaux pour observer la manière de naviguer ect.

> “L'utilisateur découvre sur mobile, met en favoris, et finit sur desktop le soir.”

Cartographie mobile-only / desktop-only / cross-device

AVIS D'EXPERTE

Caroline Nazin

Un point méthodologique qui me tient à coeur : avant d'adapter des tests existants, cartographier en amont les comportements propres à chaque contexte. Cette approche partagée avec produit et dev clarifie les priorités et élimine la duplication inutile d'efforts.

📱 MOBILE-ONLY

- Apple Pay/ Google Pay
- Twint (bascule vers app bancaire)
- Wosai, Alipay, WeChat Pay
- Scan de carte par caméra

🖥️ DESKTOP-ONLY

- Impression PDF de devis
- Comparateurs en grille large
- Configurateurs complexes

🔄 CROSS-DEVICE

- Parcours commencés sur un canal et terminés sur un autre
- Panier, wishlist, favoris
- SSO app / web
- Deep links et reprise de session

Pourquoi cartographier en amont ?

L'idée reçue dans les équipes est d'adapter les tests existants au mobile après coup. L'expérience terrain montre l'inverse : cartographier avant les comportements par contexte d'usage génère des gains considérables. Certaines fonctionnalités n'existent tout simplement que sur un seul canal — les tester sur l'autre est du gaspillage pur. Cette cartographie partagée avec les équipes produit et développement transforme le dialogue. Elle permet de prioriser les efforts de test avec précision, d'allouer les ressources sur les bons contextes, et d'arrêter de dupliquer bêtement des scénarios qui ne correspondent pas à la réalité d'usage.

Ce que ça change concrètement

- **Suites dédiées full mobile distinctes des suites desktop**
- **Réduction de la redondance entre équipes**
- **Priorités QA alignées avec produit et dev**
- **Couverture plus précise des comportements réels**

AVIS D'EXPERTE

Caroline Nazin

In-app browsers

Pour les parcours e-commerce, il faut aussi penser aux navigateurs embarqués dans les applications sociales.

Quand un utilisateur clique sur un lien depuis Instagram, Facebook, TikTok ou LinkedIn, le lien ne s'ouvre pas toujours dans Safari ou Chrome. Il peut s'ouvrir dans une WebView embarquée, avec ses propres restrictions et ses propres comportements.

Cela peut avoir un impact sur :

- les cookies tiers
- l'autofill
- la biométrie
- les paiements
- les redirections
- les parcours 3D Secure
- la persistance de session
- l'ouverture de deep links
- le retour vers l'application source

Un flow de paiement qui fonctionne parfaitement dans Chrome ou Safari peut échouer dans le navigateur intégré d'une application sociale. Pour un e-commerce qui investit dans des campagnes Meta, TikTok ou LinkedIn, ce n'est pas un détail technique : c'est un risque direct sur la conversion.

Un parcours mobile ne commence pas toujours dans l'application ou dans le navigateur par défaut. Il peut commencer dans Instagram, Facebook, TikTok ou LinkedIn. Tester le mobile, c'est aussi tester le contexte réel d'entrée dans le parcours.

RÉFÉRENTIEL ISTQB APPLIQUÉ AU MOBILE

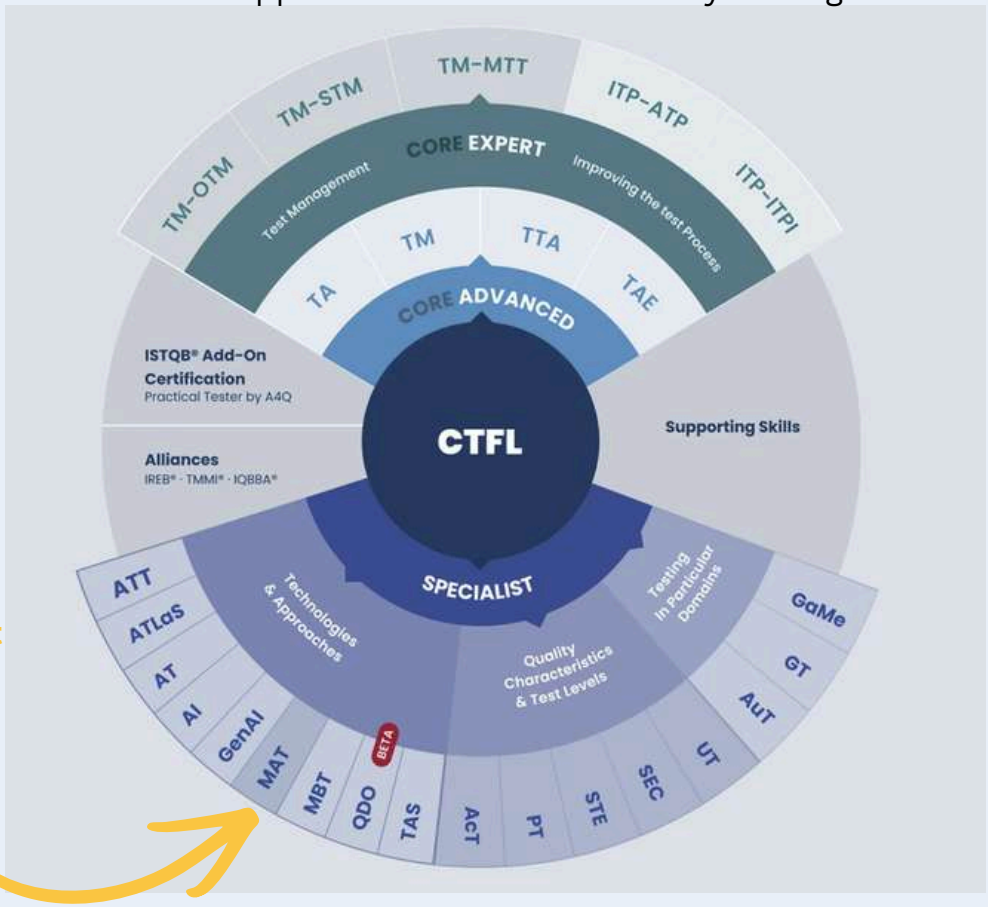
REFERENTIEL ISTQB APPLIQUE AU MOBILE

Tester les applications mobiles avec l'ISTQB® : du syllabus officiel à la réalité terrain

Les applications mobiles sont devenues des produits du quotidien. Banque, santé, e-commerce, transport, administration : elles concentrent aujourd'hui des usages sensibles, fréquents et peu tolérants à l'erreur. Pourtant, le test mobile est encore souvent abordé comme une simple déclinaison du test web, "mais sur téléphone".

Pour dépasser cette vision réductrice, il est pertinent de s'appuyer sur le cadre proposé par l'ISTQB®, et plus précisément sur son syllabus Test d'application mobile. L'objectif n'est pas d'appliquer une recette toute faite, mais de structurer une réflexion qualité adaptée à un contexte instable, fragmenté et fortement dépendant de l'usage réel. Il est disponible sur le site du CFTL dont le GASQ est leur partenaire pour l'organisation des examens en France : https://cftl.fr/tests-logiciels/documents-associes-certifications/

Le schéma ci-dessous présente l'ensemble des certifications et syllabus proposés par l'ISTQB®, et permet de situer le Test d'application mobile dans l'écosystème global du test logiciel.



ISTQB® : Test d'application mobile

SOURCE : HTTPS://ISTQB®.ORG/WHAT-WE-DO/

Le syllabus ISTQB® Test d'application mobile : ce que c'est vraiment

Lorsqu'on évoque le syllabus ISTQB® Test d'application mobile, beaucoup de testeurs imaginent un document technique décrivant des outils ou des pratiques spécifiques au mobile. En réalité, ce syllabus n'a pas vocation à expliquer comment tester une application mobile pas à pas. Il vise avant tout à structurer une réflexion qualité adaptée à un environnement très particulier, marqué par la diversité des devices, la dépendance au système d'exploitation et l'usage en conditions réelles.

Publié par l'ISTQB®, ce syllabus sert de base à la certification Certified Tester - Test d'application mobile (CT-MAT). Il s'inscrit dans le parcours officiel des certifications ISTQB® et constitue une spécialisation, accessible uniquement aux testeurs déjà certifiés Niveau Fondation (CTFL).

Ce prérequis n'est pas anodin : il rappelle que le test mobile ne se pense pas en rupture avec les fondamentaux du test logiciel, mais comme une extension de ceux-ci à un contexte plus contraint et plus instable. Le syllabus ne revient donc pas en détail sur ce que couvre le Foundation Level. Il part du principe que ces notions sont acquises, et se concentre sur ce que le mobile change réellement dans la manière de tester.

Le mobile : un contexte qui transforme la pratique du test

Tester une application mobile, ce n'est pas seulement tester une interface plus petite.

C'est tester un logiciel :

- fortement dépendant du matériel (capteurs, batterie, mémoire),
- étroitement lié au système d'exploitation,
- soumis à des interruptions fréquentes,
- utilisé dans des situations variées et souvent imprévisibles.

Le syllabus distingue les différents types d'applications mobiles (native, web, hybride) et rappelle que chacune implique des risques et des stratégies de test différents. Il insiste également sur le rôle des permissions, du multitâche, des notifications et des changements de contexte dans le comportement de l'application.

SIGNATURE DU MAG

À travers ces éléments, le message est clair : le mobile rend visibles des risques qui existent déjà ailleurs, mais de manière plus brutale et plus immédiate.

AVIS D'EXPERTE

Caroline Nazin

🪲 </Quality Engineer Web> & 👩‍💻 </Cheffe de projet fonctionnel> · Luxe & E-commerce international · AI-Powered QE · Automation

Les applications hybrides méritent une attention particulière.

Dans certaines applications hybrides ou cross-platform, y compris des applications construites avec React Native, Cordova, Flutter ou d'autres technologies, une partie du parcours peut être affichée dans une WebView plutôt que dans un écran totalement natif. Cela peut concerner une page de paiement, une page d'aide, une fiche produit, une page de connexion ou un module tiers.

Le piège, c'est de considérer cette WebView comme "du web classique dans une app". En réalité, elle constitue un environnement d'exécution à part.

Les comportements peuvent varier sur :

- **les permissions caméra**
- **le stockage local**
- **les cookies**
- **les notifications**
- **les redirections**
- **l'ouverture de liens externes**
- **l'authentification**
- **les paiements**
- **les performances**
- **la gestion du retour arrière**

Une WebView n'est donc ni complètement native, ni complètement web. Elle doit être identifiée dans la stratégie de test comme un contexte spécifique.

Une approche qualité guidée par le risque et l'usage

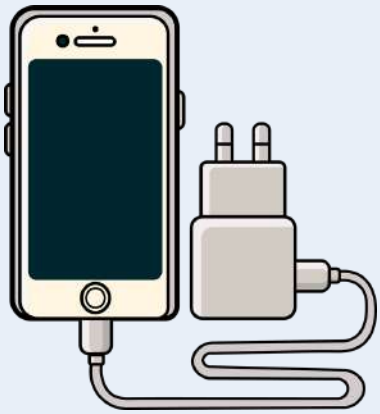
> C'est aussi le premier syllabus ISTQB à avoir introduit la notion d'objectifs pratiques (basés sur la pratique et des exercices, même si ceux-ci ne sont formellement pas évalués en raison du mode d'évaluation basé sur des QCMs). D'autres (et notamment la nouvelle certification GenAI) suivront cette approche qui se justifie pleinement et répond à certaines critiques jugeant le corpus ISTQB comme trop théorique.
>
> - Olivier Denoo

Le syllabus ISTQB® Test d'application mobile met fortement en avant l'approche basée sur les risques. Ce choix est directement lié aux contraintes du mobile : la diversité des devices et des environnements rend toute tentative d'exhaustivité irréaliste.

Le testeur est donc invité à :

- comprendre les usages principaux de l'application,
- identifier les profils utilisateurs,
- évaluer les impacts potentiels d'un défaut,
- prioriser les scénarios de test en conséquence.

Cette approche permet de concentrer l'effort de test là où il apporte le plus de valeur. Sur mobile, un défaut visible ou gênant peut entraîner une désinstallation immédiate de l'application. Le test devient alors un levier de réduction du risque perçu par l'utilisateur final, et non une simple activité de conformité.



Les tests non fonctionnels au cœur du test mobile

Le syllabus accorde une place importante aux tests non fonctionnels, et ce point prend tout son sens dans le contexte mobile. Les caractéristiques de qualité bien connues : utilisabilité, performance, compatibilité, sécurité, fiabilité,... deviennent immédiatement concrètes.

Une application mobile peut être fonctionnellement correcte tout en étant rejetée par ses utilisateurs parce qu'elle :

- consomme trop de batterie,
- fait chauffer le téléphone,
- se comporte différemment selon le device,
- ne reprend pas correctement après une interruption,
- ou expose maladroitement des données personnelles.

SIGNATURE DU MAG

Le syllabus rappelle ainsi que la qualité logicielle ne se limite pas au respect des exigences fonctionnelles. Sur mobile, la qualité perçue joue un rôle déterminant dans le succès ou l'échec d'une application.

Automatisation, tests manuels et observation humaine

Concernant l'automatisation, le syllabus adopte une position mesurée. Il reconnaît son intérêt pour la non-régression et les parcours stables, mais souligne aussi ses limites dans un environnement mobile fortement dépendant du contexte.

Certains comportements : gestes tactiles complexes, interactions avec les capteurs, interruptions imprévues,... restent difficiles à automatiser de manière fiable. Le syllabus valorise donc les tests exploratoires et l'observation humaine, qui permettent de détecter des défauts souvent invisibles dans des scénarios scriptés.

Le testeur mobile est invité à se comporter comme un utilisateur réel, à manipuler l'application, à provoquer des situations inattendues et à analyser la robustesse du système face à ces contraintes.

Ce que le syllabus apporte... et ce qu'il n'impose pas

Le syllabus ISTQB® Test d'application mobile fournit un cadre structurant, un langage commun et une grille de lecture cohérente pour aborder le test mobile. Il permet de mieux argumenter les choix de test, de dialoguer avec les équipes produit et de positionner le test mobile comme une expertise à part entière.

En revanche, il ne prétend pas :

- imposer une stratégie unique,
- remplacer l'expérience terrain,
- couvrir tous les cas possibles.

C'est un cadre, pas une méthode clé en main. Et c'est précisément ce qui le rend utile dans un domaine aussi évolutif que le mobile.

Qualité logicielle : relire les caractéristiques ISTQB® à travers le prisme mobile

Dans le référentiel ISTQB®, les tests fonctionnels visent à vérifier que le système fait ce qui est attendu. Sur mobile, cette vérification prend une dimension supplémentaire : une fonctionnalité n'existe jamais indépendamment de son contexte d'exécution.

Une même fonctionnalité peut :

- fonctionner sur un device et pas sur un autre,
- se comporter différemment selon la version de l'OS,
- devenir inutilisable si une permission est refusée,
- échouer lorsque l'application passe en arrière-plan.

Tester la fonctionnalité mobile, ce n'est donc pas seulement valider un scénario nominal.

C'est vérifier que la fonctionnalité résiste aux conditions réelles d'utilisation. Le testeur doit se poser des questions simples mais déterminantes : que se passe-t-il si l'utilisateur refuse l'accès à la localisation ? Si l'application est interrompue ? Si le réseau "disparaît" au mauvais moment ?

Le mobile rappelle ainsi que la fonctionnalité est toujours conditionnelle.

Tester mobile, ce n’est pas tester pareil

Fiabilité

Survivre aux interruptions système, à la mise en arrière-plan et à la reprise d’état

Sécurité

Permissions, données locales, biométrie, niveaux d’autorisation

Compatibilité

OS, versions, constructeurs, tailles d’écran : gérer la fragmentation mobile

Utilisabilité

Gestes, ergonomie tactile, usage en mobilité : une qualité immédiatement sanctionnée

Accessibilité

Fonctionnalités d’accessibilité natives des OS, non négociables

Localisation

Géolocalisation, contexte physique, usage nomade

AVIS D'EXPERTE

Caroline Nazin

Tester en responsive desktop à la souris ne teste pas le mobile !

Je vois des équipes qui croient faire du test mobile alors qu'elles redimensionnent juste leur Chrome. La thumbzone qui rend un bouton en haut à droite inatteignable sur iPhone 15 Pro Max, le swipe système qui entre en conflit avec un carousel, l'appui long qui déclenche un menu contextuel, la précision tactile qui révèle que des cibles trop petites passent à la souris mais bloquent au doigt, tout ça n'existe pas sur desktop.

L’expérience utilisateur : une qualité immédiatement sanctionnée

L’ISTQB® considère l’UX (expérience utilisateur) comme une caractéristique de qualité à part entière. Sur mobile, elle devient un facteur décisif.

L’utilisateur mobile :

- interagit avec ses doigts,
- utilise l’application dans des contextes variés,
- dispose de peu de temps et de patience.

Un bouton trop petit, un geste mal interprété ou un parcours confus, un texte peu lisible notamment dans des conditions d’éclairages défavorables ou sur des écrans petits ou peu performants, ne sont pas perçus comme de simples défauts ergonomiques. Ils sont vécus comme des points bloquants ou des points d’irritation sévères conduisant dans les cas les plus graves à l’abandon ou à la désinstallation de l’application. Sur mobile, l’utilisabilité n’est pas un bonus, c’est une condition d’acceptation.

Le testeur mobile doit donc adopter une posture différente : se mettre dans la peau de l’utilisateur, manipuler l’application comme elle sera réellement utilisée, et observer les frictions. Ces frictions sont souvent invisibles dans des tests purement scriptés, mais évidentes lors de tests exploratoires.

Performance : une notion élargie par le mobile

Dans de nombreux projets, la performance est encore réduite à une question de temps de réponse. Le mobile oblige à élargir cette vision.

Une application mobile performante est une application qui :

- démarre rapidement,
- reste fluide dans le temps,
- n’épuise pas la batterie,
- n’accapare pas excessivement la mémoire,
- ne provoque pas de chauffe anormale du device.

Ces critères ont un impact direct sur l’expérience utilisateur. Une application qui “répond vite” mais qui consomme trop de ressources sera désinstallée sans hésitation.

Le test mobile remet ainsi en question certaines métriques classiques et pousse à considérer la performance comme une expérience globale, pas comme une simple mesure technique.

AVIS D'EXPERTE

Caroline Nazin

🪲 </Quality Engineer Web> & 👩‍💻 </Cheffe de projet fonctionnel> · Luxe & E-commerce international · AI-Powered QE · Automation

Le clavier virtuel est souvent oublié dans les stratégies de test mobile. Pourtant, c'est un vrai "bug magnet".

Le clavier n'est pas un simple détail d'interface : c'est une surface système, pas un élément de l'application. Il se superpose à l'UI avec sa propre logique, son propre comportement et ses propres contraintes.

Quelques cas très concrets à tester :

- un champ code postal qui force un clavier alphabétique au lieu d'un clavier numérique
- un bouton "Valider" qui devient invisible dès que le clavier s'ouvre, parce qu'il est positionné dans la partie masquée du viewport
- une validation temps réel qui se déclenche trop tôt pendant la saisie
- les claviers CJK, où l'étape de composition pinyin (saisie phonétique en lettres latines avant sélection du caractère chinois final) peut casser la validation des champs
- les champs email, téléphone, code postal, montant, code de sécurité ou OTP qui n'ouvrent pas le bon type de clavier
- le comportement du formulaire quand le clavier s'ouvre, se ferme, ou quand l'utilisateur passe d'un champ à l'autre.

Un mauvais inputmode, un champ mal configuré ou un bouton masqué peuvent ajouter de la friction immédiatement visible dans un parcours critique. Sur mobile, la saisie n'est jamais neutre : elle fait partie intégrante de l'expérience utilisateur.

Compatibilité : une qualité challengée

La compatibilité est un sujet incontournable du test mobile, en particulier dans des écosystèmes fragmentés.

Compatibilité signifie ici :

- diversité des tailles et résolutions d'écran,
- différences de comportement entre versions d'OS,
- spécificités liées aux constructeurs,
- variations matérielles importantes.

Face à cette diversité, le testeur mobile ne peut pas tout tester. Le syllabus ISTQB® Test d'application mobile encourage donc une approche raisonnée, basée sur :

- les profils utilisateurs,
- les statistiques d'usage,
- les risques métier.

La compatibilité devient un exercice de priorisation et de stratégie, pas une quête impossible de couverture totale.

Sécurité : visible, concrète, et souvent locale

La sécurité fait partie intégrante de la qualité logicielle selon l'ISTQB®. Sur mobile, elle prend une forme très concrète, souvent perceptible par l'utilisateur lui-même.

Le testeur mobile est confronté à des questions telles que :

- où sont stockées les données ?
- quelles permissions sont demandées, et pourquoi ?
- que se passe-t-il en cas de perte ou de vol du device ?
- comment sont gérées l'authentification et la biométrie ?

Il ne s'agit pas de mener des audits de sécurité avancés, mais d'identifier des risques visibles, compréhensibles et potentiellement bloquants pour l'utilisateur. Le mobile rapproche ainsi la sécurité de l'usage, et rappelle que la confiance fait partie intégrante de la qualité.

AVIS D'EXPERTE

Caroline Nazin

🪲 </Quality Engineer Web> & 👩‍💻 </Cheffe de projet fonctionnel> · Luxe & E-commerce international · AI-Powered QE · Automation

Face ID, Touch ID et l'empreinte Android sont souvent associés à une expérience fluide. Pourtant, en test mobile, il ne faut pas seulement vérifier que la biométrie fonctionne quand tout va bien.

Les cas d'échec sont tout aussi importants :

- Face ID ne reconnaît plus l'utilisateur
- l'utilisateur porte un masque, des lunettes ou se trouve dans de mauvaises conditions de lumière
- l'empreinte n'est plus reconnue
- la biométrie est désactivée sur le device
- un changement de configuration invalide l'authentification
- le wallet ou le moyen de paiement associé n'est plus valide
- le device impose un retour au PIN, au mot de passe ou à un autre facteur d'authentification

La biométrie intervient souvent sur des moments critiques : connexion, validation d'action sensible, paiement Apple Pay ou Google Pay. Un bug à ce moment-là peut faire perdre une vente ou bloquer un utilisateur dans un parcours sensible.

Le fallback doit donc être testé avec sérieux : retour au PIN, mot de passe, réauthentification, annulation propre, message compréhensible et reprise du parcours sans perte de données.

Fiabilité : survivre aux interruptions

Le mobile est un environnement **instable** par nature. Appels entrants, notifications, changement de réseau, mise en veille : les interruptions font partie de l'usage normal.

Une application fiable est une application capable de :

- reprendre son état correctement,
- ne pas perdre de données,
- ne pas se bloquer après une interruption.

Le test mobile met souvent en évidence des défauts de fiabilité que d'autres contextes masquent. Tester ces situations, volontairement, fait pleinement partie de la mission du testeur mobile.

Les approches de test recommandées par l'ISTQB®, adaptées au mobile

Le test mobile impose une réalité que beaucoup de projets découvrent tardivement : il est impossible de tout tester. Multiplicité des devices, diversité des systèmes d'exploitation, usages variés, conditions réseau instables… vouloir appliquer une stratégie exhaustive conduit rapidement à l'épuisement des équipes, sans pour autant garantir une meilleure qualité.

C'est pour cette raison que le syllabus Test d'application mobile de l'ISTQB® met en avant des approches de test complémentaires, particulièrement adaptées au contexte mobile : le test basé sur les risques, le test exploratoire et une automatisation ciblée. L'enjeu n'est pas de choisir une seule approche, mais de les combiner intelligemment.

Tester par les risques : une nécessité sur mobile

L'approche basée sur les risques est un pilier du référentiel ISTQB®. Sur mobile, elle devient presque une condition de survie.

Tester par les risques consiste à se poser des questions simples mais structurantes :

Quels usages sont les plus fréquents ?

Quels parcours génèrent le plus de valeur métier ?

Quels défauts auraient l'impact le plus fort pour l'utilisateur ?

Dans quelles situations l'application est-elle la plus fragile ?

Cette approche permet de concentrer les efforts de test sur les zones sensibles : authentification, paiement, données personnelles, parcours critiques. Elle évite de disperser l'énergie sur des scénarios peu utilisés, tout en renforçant la pertinence du test.

Sur mobile, le risque n'est pas seulement technique. Il est aussi perceptif. Un défaut mineur sur le plan fonctionnel peut devenir rédhibitoire s'il gêne l'utilisateur dans un contexte réel.

Le test exploratoire : observer l'usage réel

Le syllabus ISTQB® reconnaît explicitement la valeur du test exploratoire, et le mobile en est un terrain privilégié. Là où les tests scriptés vérifient des comportements attendus, le test exploratoire cherche à observer ce qui se passe réellement lorsque l'application est utilisée sans scénario prédéfini.

Le testeur mobile explore :

les gestes

pincer, appui long, glisser, balayer

les transitions

navigation, retours, états intermédiaires

les interruptions

appels, notifs, changement d'app

les imprévus

actions "hors scénario", erreurs, latences

AVIS D'EXPERTE

Caroline Nazin

Sur mobile, tester les interruptions ne consiste pas seulement à cocher une catégorie : il faut descendre au niveau des scénarios opérationnels, ceux qui reproduisent les situations réelles vécues par l'utilisateur.

L'appel entrant pendant la saisie du CVV, la notification push qui masque le bouton valider, l'alerte batterie faible à 10% qui change le focus, la bascule vers WhatsApp pour chercher un code reçu puis le retour dans ton app, le retour d'un appel FaceTime ou d'une visio Teams. Chacun de ces scénarios génère ses propres bugs.

SIGNATURE DU MAG

N'hésitez pas à changer l'orientation d'écran, à passer l'application en arrière-plan, à couper le réseau, de refuser une permission, à revenir plus tard. Ces manipulations, proches de l'usage réel, révèlent souvent des défauts invisibles dans des cas de test formalisés.

Le test exploratoire permet aussi de détecter des problèmes d'utilisabilité, de cohérence ou de fluidité, difficiles à exprimer dans des critères strictement fonctionnels. Sur mobile, cette capacité d'observation est un atout majeur pour améliorer la qualité perçue.

AVIS D'EXPERTE

Caroline Nazin

🪲 </Quality Engineer Web> & 👩‍💻 </Cheffe de projet fonctionnel> · Luxe & E-commerce international · AI-Powered QE · Automation

Sur mobile, une session utilisateur est rarement longue, linéaire et parfaitement maîtrisée.

L'utilisateur ouvre l'application quelques secondes, répond à un message, perd le réseau, verrouille son téléphone, revient plus tard, clique sur un lien depuis un email marketing ou reprend un panier abandonné plusieurs heures après.

Ce cycle de vie fragmenté change beaucoup de choses en test.

Quelques scénarios à prévoir :

- **reprise d'état après qu'iOS a tué l'application en arrière-plan sous pression mémoire**
- **expiration de session côté serveur quand l'utilisateur revient plusieurs heures plus tard**
- **panier abandonné puis repris après une longue interruption**
- **deep link qui réveille l'application sur une fiche produit précise depuis un email ou une notification**
- **token d'authentification qui expire pendant une saisie de formulaire**
- **retour dans l'application après verrouillage du téléphone**
- **reprise d'un paiement ou d'un parcours critique après interruption.**

Ces cas sont très mobiles dans leur nature. Ils ne se voient pas toujours dans un test fonctionnel classique, exécuté d'un seul bloc, dans un environnement stable.

Automatisation : un levier, pas une fin en soi

L'automatisation des tests est souvent présentée comme un objectif à atteindre. Le syllabus Test d'application mobile adopte une position plus nuancée, particulièrement adaptée au mobile.
Automatiser sur mobile est pertinent lorsque :

- les parcours sont stables,
- les comportements sont reproductibles,
- la valeur ajoutée est claire (non-régression, gain de temps).

En revanche, certaines dimensions du mobile restent difficiles à automatiser de manière fiable :

- gestes complexes,
- interactions avec les capteurs,
- comportements dépendants du contexte réel,
- animations et transitions visuelles.

Le risque, sur mobile, est de confondre couverture automatisée et qualité réelle. Une suite de tests automatisés peut être verte tout en laissant passer des problèmes majeurs d'expérience utilisateur.

Le syllabus invite donc à considérer l'automatisation comme un outil au service de la stratégie de test, et non comme une mesure de maturité en soi. Elle complète le test manuel et exploratoire, sans les remplacer.

Trouver l'équilibre entre les approches

L'une des forces du référentiel ISTQB® appliqué au mobile est de ne pas opposer les approches entre elles. Le test mobile efficace repose sur un équilibre :

- **le test basé sur les risques pour décider où investir l'effort,**
- **le test exploratoire pour comprendre l'usage réel,**
- **l'automatisation pour sécuriser les parcours critiques dans le temps.**

Cet équilibre évolue selon le contexte du projet, la maturité de l'équipe et les contraintes techniques. Le syllabus ne fournit pas de formule universelle, mais un cadre de réflexion permettant d'adapter la stratégie.
Sur mobile, tester n'est pas seulement vérifier que l'application fonctionne. C'est anticiper les situations réelles, accepter l'imprévu et construire une stratégie capable de s'adapter à un environnement changeant.

Le testeur mobile : un rôle d'anticipation

À travers ces approches, le testeur mobile occupe une position particulière. Il ne se contente pas d'exécuter des cas de test. Il anticipe les usages, identifie les fragilités et alerte sur des risques parfois invisibles pour les autres rôles du projet.

Le syllabus ISTQB® Test d'application mobile valorise cette posture analytique. Il rappelle que le test mobile est une discipline à part entière, qui exige à la fois rigueur méthodologique et compréhension fine de l'expérience utilisateur.

C'est cette combinaison qui permet de construire des applications mobiles robustes, utilisables et acceptées par leurs utilisateurs.

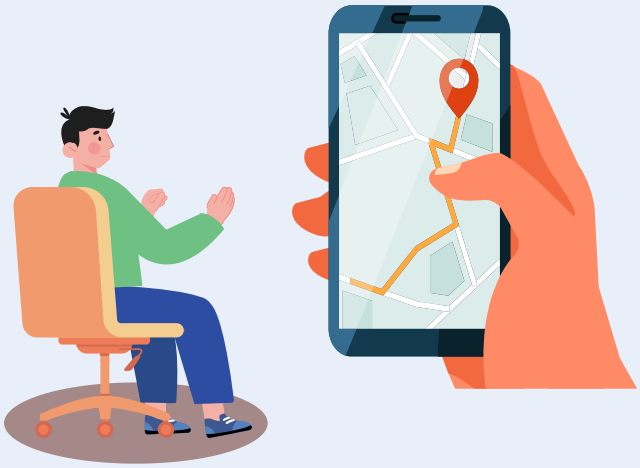
L'usage réel ne se lit pas dans un syllabus

Le syllabus parle d'utilisateurs, de contextes et de risques. Mais il ne peut pas décrire la réalité fine de l'usage.

Sur mobile, l'application est utilisée :

- dans les transports (merci les tunnels et la perte du réseau !)
- en marchant,
- avec une connexion instable,
- parfois d'une seule main ou avec un gant,
- souvent dans un état d'attention partielle.

Ces situations influencent directement la perception de la qualité. Une application parfaitement conforme aux exigences peut devenir pénible, voire inutilisable, dans un contexte réel.



SIGNATURE DU MAG

Le syllabus Test d'application mobile de l'ISTQB® fournit un cadre solide pour penser le test mobile. Il structure les concepts, met en lumière les risques et repositionne le test mobile comme une spécialisation à part entière.

Mais comme tout référentiel, il ne peut pas tout couvrir. Et surtout, il ne prétend pas remplacer l'expérience du terrain.

Le test mobile confronte très vite le testeur à des réalités qui dépassent le cadre théorique. Non pas parce que le syllabus est incomplet, mais parce que le mobile est un environnement vivant, mouvant, et fortement influencé par les usages réels

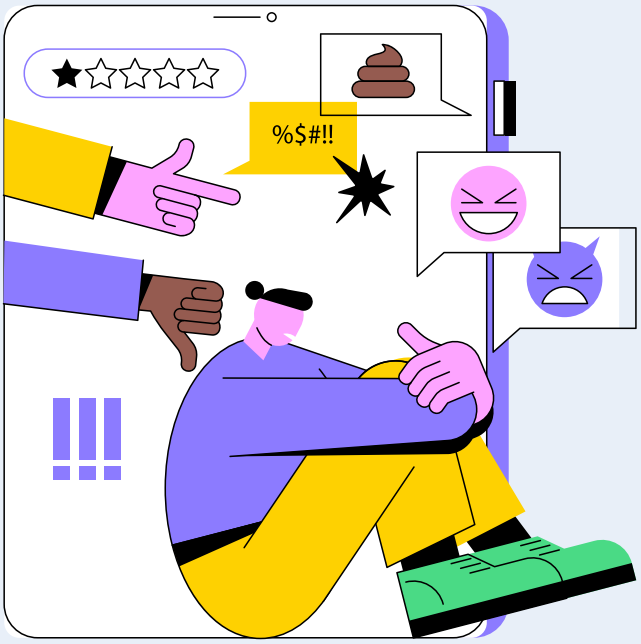
Les retours utilisateurs : un signal qualité incontournable

Le syllabus ne détaille pas l'exploitation des retours utilisateurs. Pourtant, sur mobile, ils sont omniprésents.

Avis sur les stores, commentaires, notes, signalements de crash : ces données constituent une source d'information précieuse sur la qualité perçue. Elles révèlent souvent :

- des problèmes d'utilisabilité,
- des incompatibilités spécifiques à certains devices,
- des défauts de performance ou de stabilité.

Le testeur mobile gagne à intégrer ces retours dans sa réflexion. Ils ne remplacent pas les tests, mais ils permettent de détecter des angles morts et de prioriser les corrections. Le terrain parle, parfois brutalement, et il serait dommage de l'ignorer.



La donnée terrain : crash reports et analytics

Autre aspect peu détaillé dans le syllabus : l'exploitation des données issues de l'application en production.

Crash reports, taux d'erreur, parcours abandonnés, temps de chargement réel… Ces indicateurs fournissent une vision concrète de ce que vit l'utilisateur. Ils permettent de comprendre :

- où l'application échoue réellement,
- dans quelles conditions,
- et pour quels profils d'utilisateurs.

Sur mobile, la frontière entre test et observation post-déploiement est plus floue que sur d'autres environnements. Le testeur ne disparaît pas une fois l'application publiée.

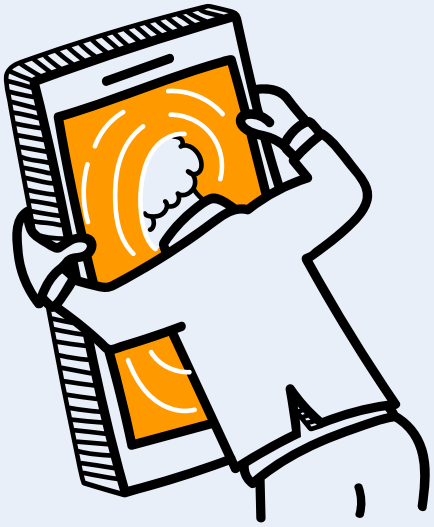
Il continue à analyser, à interpréter et à alimenter l'amélioration continue.

Les contraintes organisationnelles et matérielles

Le syllabus ne peut pas anticiper les contraintes propres à chaque organisation.

Or, sur mobile, ces contraintes pèsent lourdement sur la stratégie de test : accès limité aux devices physiques, dépendance à des plateformes externes, délais de mise à disposition des builds, restrictions liées aux stores : autant de réalités qui façonnent le quotidien du testeur mobile.

Le terrain impose souvent des compromis. Le rôle du testeur est alors d'adapter le cadre théorique à ces contraintes, sans perdre de vue les objectifs de qualité. C'est là que l'expérience et le discernement prennent le relais du référentiel.



Le test mobile est une activité profondément collaborative. Développeurs, designers, product owners, équipes support : chacun détient une partie de la compréhension du produit et de ses usages. Sur mobile, le dialogue avec l'UX, le design et le support utilisateur est particulièrement précieux.

Les problèmes de qualité mobile ne sont pas toujours purement techniques. Ils sont souvent liés à des choix de conception, à des compromis ergonomiques ou à des hypothèses d'usage erronées. Le testeur mobile joue alors un rôle de médiateur, en traduisant les signaux du terrain en actions concrètes.

SIGNATURE DU MAG

Ne vous enfermez pas dans une méthode figée. Utilisez les repères, un langage commun mais imposez-vous l'humilité, l'observation et un ajustement permanent.

C'est dans l'articulation entre ces deux dimensions : cadre théorique et réalité vécue, que se construit une pratique du test mobile réellement utile

Certification ISTQB® Mobile : à quoi ça sert vraiment ?

Passer une certification représente un investissement, en temps comme en argent.

Ce qu'une certification ISTQB® peut apporter

- de structurer sa compréhension du test logiciel et du test mobile
- d'acquérir un vocabulaire commun, utile pour échanger avec d'autres testeurs et avec les équipes projet
- de découvrir une vision globale de la qualité, au-delà de l'exécution de cas de test
- de légitimer une montée en compétence, notamment dans un contexte de reconversion ou de spécialisation

Ce que la certification ne garantit pas

- ne remplace pas l'expérience terrain
- ne garantit ni un poste, ni une augmentation
- ne dispense pas d'apprendre sur des projets réels
- ne fait pas de quelqu'un un bon testeur mobile sans pratique

Sur mobile en particulier, la capacité à observer l'usage réel, à explorer une application et à comprendre les contraintes du terrain reste déterminante.

Pour la certification ISTQB® Test d'application mobile (CT-MAT), les coûts observés sont généralement les suivants :

En candidat libre : environ 265 € pour l'examen seul (tarif en exemple à Paris)

---

adapté à des testeurs ayant déjà de l'expérience et souhaitant formaliser leurs connaissances ou valider une spécialisation.

Via une formation : environ 1 500 € HT pour une formation de deux jours incluant le passage de la certification

---

adapté à des profils juniors ou en reconversion, qui ont besoin d'un cadre pédagogique, d'échanges et d'exemples pour s'approprier les concepts, mais pas que, beaucoup d'entreprises passent par cette filière grâce au budget OPCO pour mettre ses équipes à niveau ou au CPF

Pour se former à l'ISTQB® Niveau fondation, vous avez également la possibilité de le passer suite à une formation ou également en candidat libre. Si vous le passez en candidat libre, je vous conseille de lire le livre de Jean-François

Jean-François Frési est un professionnel du test logiciel qui partage régulièrement du contenu sur LinkedIn, a créé un podcast, une newsletter et a publié un guide de préparation à la certification ISTQB® Foundation.

Son livre, Se préparer à la certification ISTQB® Fondation v4 avec 400 questions pour réussir, est une ressource d'entraînement intensive centrée sur 400 questions alignées sur le syllabus Foundation v4, avec des explications détaillées pour chaque réponse, et il est disponible en version électronique.

50%

PROMO CODE

ISTQB50

Jusqu'au 15 juin

L'ISTQB SANS DETOUR

Souvent résumé à une certification, l'ISTQB est pourtant au cœur de questions bien plus larges : langage commun, professionnalisation du test, formation des juniors et évolution du métier. Olivier Denoo nous livre ici sa lecture du sujet.

Entretien avec Olivier Denoo, vice-président de l'ISTQB, président du CFTL et vice-président de ps_testware SAS

Selon vous, quel est le rôle de l'ISTQB® aujourd'hui dans l'évolution du métier de testeur ?

Le rôle principal de l'ISTQB® est d'assurer une standardisation des meilleures pratiques et des termes employés au niveau mondial, ce qui suppose quelquefois des compromis tant les cultures et les approches peuvent être diverses et variées selon l'endroit où l'on se trouve.

Grâce à cette présence mondiale, les testeurs ne sont plus seuls, isolés dans leurs pratiques. Ils font partie d'une communauté plus large qui sera encore amenée à se renforcer dans un futur proche.

Avec le temps, le corpus de connaissance a dû se diversifier, tant sur les aspects techniques que sur les métiers ou les domaines spécifiques couverts. La mise en lumière de ces métiers du test, en tant que tels, a permis leur reconnaissance et leur acceptation au sens large au sein des métiers des TIC.

Enfin, les certifications valident des acquis normalisés, reconnus mondialement, ce qui facilite grandement les processus RH centrés sur les compétences des testeurs logiciels.

Qu'est-ce que l'ISTQB® apporte de vraiment utile aux professionnels du test sur le terrain ?

La mise en place d'un vocabulaire contrôlé, et ce dans de nombreuses langues, permet de briser les tours de Babel et de réduire les approximations (un bug, un défaut, un incident, une défaillance…).

La recommandation des meilleures pratiques permet aux jeunes (et moins jeunes) testeurs d'aborder les problématiques directement par le bon bout.

Qu'ils doivent analyser et améliorer un existant ou mettre en place de nouveaux processus, le corpus de connaissances mis en place, depuis plus de 20 ans, par l'ISTQB® leur apporte des solutions et des approches concrètes.

Quels sont les malentendus ou usages réducteurs que vous observez le plus souvent autour de l'ISTQB® ?

Le premier reproche adressé à l'ISTQB® porte sur ses examens, jugés trop théoriques, négligeant la pratique et l'expérience et qui ne suffiraient donc pas à évaluer ou à « faire » de bons testeurs.

Si cette critique fait sens, le malentendu vient surtout du fait que ce n'est pas la vocation de l'ISTQB®, malgré l'introduction, de plus en plus généralisée, d'objectifs pratiques (HO) et d'outils d'entraînement (Practitional Tester…).

Il faut plutôt voir nos examens comme une sorte de permis de conduire ou comme un diplôme. Un premier pas (souvent) nécessaire vers la professionnalisation, mais pas suffisant.

Il va de soi que la pratique, l'expérience et la connaissance du contexte et du métier sont indispensables à tout testeur qui se respecte.

Si l'ISTQB® en est consciente et le reconnaît sans peine, ce mauvais procès et le mythe du super-testeur-que-l'on-fabriquerait-en-3 jours-chrono tendent à persister.

Certaines critiques portent aussi sur les contenus, quelquefois jugés trop légers ou trop consensuels.

Une fois encore, les attentes ne sont pas en phase avec la mission de l'ISTQB® ou avec ses objectifs. S'il est indéniable que l'on puisse « faire mieux » localement, notre vision nous impose de nécessaires compromis et ajustements. Par exemple l'Agilité est beaucoup moins présente sur les marchés asiatiques en raison de différences culturelles au sein des organisations, il nous faut donc composer avec des visions parfois très divergentes, afin d'assurer cette reconnaissance mondiale qui est notre force.

Nos certifications sont, et doivent rester, reconnues partout dans le monde de sorte qu'un testeur certifié à Paris pourra travailler à Bangkok avec la même facilité et avec la même reconnaissance de ses acquis.

A ces critiques, je réponds souvent qu'1.1 Mio de « followers » certifiés et convaincus ne peuvent se tromper.

Beaucoup de juniors rencontrent des difficultés pour accéder à un premier poste dans le test, parce qu'on leur demande déjà de l'expérience. Quel conseil leur donneriez-vous pour entrer dans le métier ?

Le phénomène n'est, hélas, pas limité au test logiciel. Notre course effrénée à la rentabilité à court terme nous pousse parfois à scier la branche sur laquelle nous sommes assis. Négliger ou ne pas intégrer les jeunes constitue, selon moi, une erreur fatale que nous paierons un jour ou l'autre.

Cela dit, puisque tout se mesure à l'aune de la valeur immédiate, les jeunes testeurs ont aussi des atouts, de nombreuses choses à apporter. Ils disposent d'un esprit naturellement « frais » et curieux.

“Négliger ou ne pas intégrer les jeunes constitue, selon moi, une erreur fatale que nous paierons un jour ou l'autre.”

Je leur conseillerai donc de « sortir de la boîte et des carcans», de « démonter » les systèmes et les pratiques pour mieux les comprendre et les repenser, d'exercer leur sens critique et d'adopter une posture d'utilisateur (Ux) ; car si le marché demande toujours plus de technique et d'automatisation, il n'en reste pas moins que nos systèmes sont faits pour être utilisés par des humains.

Il leur faudra donc oser et jongler avec des compétences multiples.

Enfin, au risque de céder aux tendances du moment, je leur recommanderais d'apprivoiser l'IA, de rester en veille permanente, d'apprendre à « prompter » efficacement ; car qu'il y ait une bulle IA ou non, la pratique s'inscrira durablement dans nos métiers de demain.

C'est peut-être leur atout le plus évident, en tant que « digital natives ».

Qu'aimeriez-vous voir évoluer en priorité dans la manière dont on forme et accompagne les testeurs aujourd'hui ?

J'aimerais déjà qu'on les forme. Aujourd'hui, les formations au test logiciels sont encore trop peu présentes et trop peu professionnelles dans nombres de cursus.

Faire rentrer le test (sans entrer dans le débat entre métier ou tâche) dans les universités est le challenge du groupe de travail « Academia » de l'ISTQB®. Au travers de programmes de reconnaissance et de soutien, l'ISTQB® tente, modestement de prêcher « la bonne parole » de la qualité logicielle – souvent le parent pauvre face à la technique ou au développement – et de faciliter son accès au sein de l'enseignement.

J'aimerais aussi que l'on enseigne plus les aspects « métier » et Ux au sein des cursus universitaires et des hautes écoles. Nos logiciels sont conçus pour faciliter la vie des humains, pas pour explorer les limites de la technique et des algorithmes, il ne faut jamais l'oublier.

> **Au sens du marché, le testeur idéal aujourd'hui est hybride. Il maîtrise la technique, l'automatisation et l'utilisation raisonnée de l'IA dans un contexte métier. Et si ce n'était pas encore assez, ce mouton à 5 pattes dispose d'un solide esprit critique, d'un sens inné du détail et de l'observation. Il est enfin un bon négociateur et un bon communicant…tout un programme !**

DOSSIER

PANORAMA DU TEST MOBILE

SE CONSTRUIRE UN LABORATOIRE DE TEST MOBILE

2302
MOBILES VENDUS PAR MINUTE *

Chaque minute, des milliers de nouveaux appareils arrivent sur le marché, chacun avec ses spécificités matérielles, logicielles et d'usage.
Face à cette diversité, vouloir tout tester est une illusion.
Le véritable enjeu du test mobile n'est pas la couverture totale, mais la capacité à faire les bons choix.

- Chiffre de 2022, source : https://www.agirpourlenvironnement.org/publications/mobiles-vendus-dans-le-monde/

Les grands types de tests mobiles

Avant de se lancer dans la mise en place d'un labo, il faut comprendre la typologie des tests à couvrir :

- Tests fonctionnels : vérifier que les fonctionnalités clés se comportent comme attendu.
- Tests UI/UX : évaluer la cohérence de l'interface sur divers écrans et la fluidité des parcours utilisateurs.
- Tests de performance : mesurer les temps de chargement, la consommation de batterie ou de mémoire.
- Tests de compatibilité : s'assurer que l'application tourne correctement sur un ensemble de combinaisons OS/appareil.
- Tests de sécurité et de conformité : vérifier le chiffrement, la gestion des permissions, et la protection des données.
- Tests réseau : simuler différents niveaux de connectivité, modes avion, pertes de signal, etc.

Les approches possibles : simulateurs, émulateurs et devices réels

Les simulateurs et émulateurs
Ce sont des outils logiciels qui reproduisent le comportement d'un appareil mobile.

- Les simulateurs (principalement pour iOS) **imitent** le système d'exploitation sans reproduire le matériel.

- Les émulateurs (courants pour Android) **simulent** à la fois le matériel et le logiciel.

Avantages : rapides à mettre en place, parfaits pour les tests de validation rapide ou d'intégration continue.

Limites : ils ne reflètent pas toujours la réalité (performances, capteurs, latence réseau, etc.). Exemples : Android Studio Emulator, Xcode Simulator, Genymotion.

Les appareils physiques
Les tests sur devices réels restent la référence pour mesurer le comportement de l'application.

Avantages : Ils permettent de détecter des anomalies invisibles en simulation :
- surchauffe, drain de batterie, lenteurs, incompatibilités matérielles, etc.

Limites : Mais leur gestion logistique (achat, maintenance, mise à jour) peut vite devenir un fardeau pour une équipe QA.

Construire son laboratoire de test mobile

Créer un labo de test mobile n'est pas qu'une question de matériel : c'est une stratégie d'équilibre entre coûts, couverture, et maintenance.

Définir les critères de sélection
Avant tout, il faut définir :
- Les OS cibles (Android 12, iOS 17, etc.)

- Les devices prioritaires selon les statistiques d'usage de vos utilisateurs (Google Analytics, Firebase, App Store Console, etc.)

- Le niveau de test attendu : automatisé, manuel, exploratoire ?

Opter pour une combinaison hybride
L'idéal est souvent un mix :
- Quelques devices physiques représentatifs (1 ou 2 iPhones récents, 2 à 3 Android de gammes différentes).

- Un pool d'émulateurs/simulateurs pour l'automatisation en CI/CD.

- Un accès à une ferme de devices à distance pour élargir la couverture.

SIGNATURE DU MAG

La vérité du terrain

Tester sur un émulateur peut valider un parcours, mais ne détectera ni la surchauffe, ni le drain batterie, ni les bugs liés aux optimisations constructeur.

Si un bug apparaît "chez les utilisateurs mais jamais en test", le device réel est souvent la réponse.

AVIS D'EXPERTE

Caroline Nazin

🪲 </Quality Engineer Web> & 👩‍💻 </Cheffe de projet fonctionnel> · Luxe & E-commerce international · AI-Powered QE · Automation

Tester sur device réel ne veut pas dire tester en boîte noire.

Un téléphone branché en USB peut donner accès à beaucoup plus d'informations qu'on ne l'imagine. Safari Web Inspector sur iPhone ou Chrome DevTools via chrome://inspect sur Android permettent d'observer le DOM, le réseau, la console et certains comportements JavaScript, exactement comme sur desktop, mais avec le rendu réel du device.

Sur Android, ADB et logcat complètent le kit. Ils permettent de récupérer des logs système, d'observer certains comportements de l'application, de comprendre une erreur silencieuse ou de mieux qualifier un bug avant de le transmettre à l'équipe de développement.

C'est un réflexe très utile sur un bug mobile :

- **brancher le device**
- **ouvrir l'inspecteur**
- **reproduire le problème**
- **regarder si l'erreur vient d'un timeout réseau, d'une erreur JavaScript silencieuse, d'un cookie bloqué, d'un problème de session ou d'un comportement spécifique au navigateur mobile**

Beaucoup de testeurs fonctionnels ignorent ces outils parce qu'ils pensent qu'ils sont réservés aux développeurs. Pourtant, ils permettent de mieux documenter un bug, de gagner du temps dans l'investigation et d'éviter les tickets trop vagues du type "ça ne marche pas sur mobile".

COMPRENDRE LA FRAGMENTATION MOBILE

Le défi du mobile : entre fragmentation et exigence utilisateur

Cette fragmentation se manifeste à plusieurs niveaux :

- Multiplicité des systèmes d'exploitation (Le nombre de plateformes différentes sur lesquelles votre application doit fonctionner : Android, iOS, parfois HarmonyOS). Chaque système a ses propres API, comportements et outils de test.

- Variété des versions d'OS en circulation. Par exemple, Android 12, 13 et 14 coexistent sur le marché, tout comme iOS 16 et 17. Chacune peut introduire des changements subtils (permissions, performances, compatibilité).

- Diversité des tailles d'écran, des résolutions, et même des capacités matérielles (RAM, CPU, GPU, capteurs, batteries...).

À cela s'ajoutent des contextes d'usage bien plus imprévisibles : perte de réseau, bascule Wi-Fi/4G, notifications intrusives, permissions non accordées, etc.

Comprendre cette fragmentation, c'est comprendre pourquoi un test mobile ne peut jamais se limiter à "un Android et un iPhone".

Android : la liberté... et le chaos

L'univers Android est ouvert, accessible à des centaines de constructeurs (Samsung, Xiaomi, Oppo, OnePlus, Google...).

Mais cette liberté a un prix : une dispersion extrême.

Chaque constructeur personnalise Android avec sa propre interface, ses optimisations énergétiques, et parfois ses propres applications système.

Résultat : deux téléphones sous Android 14 peuvent se comporter différemment, que ce soit sur la gestion de la batterie, les notifications ou les performances graphiques.

Les conséquences pour les tests sont que :

- Les tests doivent inclure plusieurs modèles de constructeurs.

- Les tests UI doivent être surveillés car certaines surcouches modifient la présentation ou le comportement des composants.

- Les tests de compatibilité (matérielle et logicielle) deviennent indispensables.

iOS : homogène, mais pas sans piège

Apple maîtrise tout : matériel, logiciel, distribution.

Cette cohérence réduit la fragmentation… mais ne la fait pas disparaître.

Les testeurs doivent composer avec :

- Des appareils aux tailles d'écran différentes (du SE au Pro Max).

- Des versions d'iOS qui coexistent parfois pendant plusieurs années.

- Des changements de comportement à chaque mise à jour (permissions, WebView, politique de confidentialité…).

Même si la couverture semble plus simple, un bug sur un iPhone 13 peut ne pas exister sur un iPhone SE, simplement à cause d'une différence de densité d'écran ou de ratio.

Les conséquences pour les tests :

- Ne pas se limiter à "le dernier iPhone".

- Tester les principales tailles d'écran (petit, moyen, grand format).

- Vérifier le comportement des mises à jour iOS sur des versions précédentes.

Les versions d'OS : un patchwork mouvant

Chaque année, de nouvelles versions sortent, mais les utilisateurs ne les adoptent pas tous au même rythme.

Sur Android, la fragmentation des versions est particulièrement marquée :

- Certains téléphones restent bloqués sur Android 11 ou 12.

- Les mises à jour peuvent dépendre du constructeur ou de l'opérateur.

Côté iOS, l'adoption est plus rapide, mais certains utilisateurs restent sur des versions plus anciennes pour éviter des ralentissements ou par contrainte de stockage.

Les conséquence pour les tests :

- Il faut définir une stratégie de couverture par version d'OS, basée sur les statistiques réelles des utilisateurs (Google Play Console, Firebase, App Store Connect).

- Les tests automatisés doivent pouvoir s'exécuter sur plusieurs environnements en parallèle.

SIGNATURE DU MAG

Bug réel

Une application validée sur Android 14 ne recevait aucune notification sur certains Xiaomi à cause de l'optimisation de la batterie trop agressive du constructeur.

Visible uniquement sur device réel, invisible sur émulateur.

Tailles et résolutions d'écran : un impact sur l'expérience utilisateur

Les tailles d'écran vont du petit smartphone de 5 pouces à la tablette 12,9 pouces.

Ce facteur influe directement sur :

- La disposition des éléments (layout responsive).
- Le comportement des gestes (scroll, swipe, tap).
- La lisibilité et l'ergonomie globale.

Un testeur mobile doit toujours vérifier le rendu visuel et fonctionnel sur plusieurs résolutions.

Les simulateurs iOS ou Android permettent de visualiser les extrêmes, mais rien ne remplace un test sur device réel pour observer le comportement tactile.

Les conséquence pour les tests sont qu'il faut:

- Utiliser des outils de tests visuels (Applitools, Percy) pour détecter les écarts d'affichage.
- Définir une grille de résolutions cibles en fonction du public visé (marché, appareil le plus utilisé).

Les surcouches et environnements constructeurs

C'est la face cachée d'Android : chaque constructeur ajoute sa propre "personnalité".

Samsung One UI, Xiaomi MIUI, Oppo ColorOS, Google Pixel UI… autant d'environnements qui modifient la manière dont certaines APIs, permissions ou optimisations s'exécutent.

Par exemple :

- La gestion du multitâche diffère d'un constructeur à l'autre.
- Les optimisations de batterie peuvent interrompre des processus en arrière-plan.
- Certaines animations ou transitions d'écran changent le timing de l'UI.

Les conséquence pour les tests :

- Inclure au moins deux ou trois grandes marques dans la campagne de tests.
- Surveiller les scénarios d'arrière-plan (notifications, tâches planifiées, synchronisations).

Fragmentation et automatisation : un duo sous tension

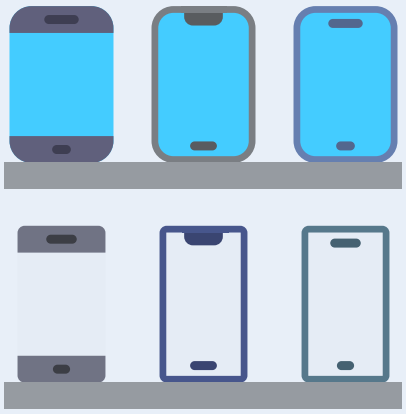
L'automatisation mobile (avec Appium, Detox, Espresso, XCUITest…) doit composer avec cette diversité.

Chaque environnement impose ses propres particularités :

- Les sélecteurs peuvent varier entre Android et iOS.
- Les identifiants d'éléments changent selon les builds ou les frameworks utilisés.
- Certains composants réagissent différemment à la simulation d'un tap ou d'un swipe.

SOLUTION DE FERMES D'APPAREILS MOBILE

SOLUTIONS DE FERMES D'APPAREILS



Qu'est-ce qu'une ferme d'appareils ?

Une ferme d'appareils est une plateforme cloud permettant de tester une application mobile sur des smartphones et tablettes réels, accessibles à distance.

Contrairement aux émulateurs, les tests sont exécutés sur de vrais devices, avec leurs contraintes matérielles et système.

Pourquoi les utiliser ?

Les fermes permettent de :

- élargir la couverture (OS, devices, versions)
- exécuter des tests automatisés en CI/CD
- éviter la gestion d'un parc physique coûteux
- reproduire plus facilement des bugs spécifiques

Elles complètent les devices physiques.

Les principales solutions du marché

- **Sauce Labs** : plateforme complète orientée automatisation et intégration CI/CD, adaptée aux environnements avancés.
- **BrowserStack** : solution simple à prendre en main, idéale pour des tests rapides et une large couverture de devices.
- **Kobiton** : positionnée sur les tests sur devices réels avec des fonctionnalités avancées de debug et d'analyse.

SIGNATURE DU MAG

Décision QA

Faut-il acheter 10 téléphones ou passer par une ferme cloud ?

Le mieux est l'hybride :

- devices réels pour les scénarios critiques
- ferme cloud pour la couverture et la CI
- coût et maintenance maîtrisés

SAUCE LABS

QUELLE PLACE DANS UNE STRATEGIE DE TEST MOBILE MODERNE ?

La brique mobile : la ferme d'appareils Sauce Labs

Côté mobile, Sauce Labs met à disposition des smartphones et tablettes réels Android et iOS accessibles à distance via une interface web ou une CI utilisables pour des tests manuels exploratoires ou pour des tests automatisés

L'outil s'intègre avec les principaux frameworks du marché :

- Appium
- Espresso
- XCUITest
- Detox

Les tests peuvent être lancés depuis un pipeline CI/CD, avec récupération des logs, captures d'écran, vidéos d'exécution et rapports associés.

Nous avions déjà parlé de Sauce labs dans le numéro 2 du magazine. Vous pouvez retrouver l'article sur le site : https://www.lemagdutesteur.fr/sauce-labs/ et télécharger le mag sur : Le mag du testeur : Numéro 2

Cas d'usage

Sauce Labs est particulièrement adapté dans les situations suivantes :

- Validation de compatibilité sur un large panel de devices Android et iOS
- Campagnes de non-régression mobile automatisées avant une release
- Tests parallélisés pour réduire le temps d'exécution en CI/CD
- Vérification rapide sur des appareils peu disponibles en interne
- Support aux équipes distribuées travaillant à distance

Dans ces contextes, la ferme mobile permet de gagner en flexibilité et en réactivité, tout en limitant la charge logistique côté QA.

Sauce Labs dans une stratégie de test mobile globale

Comparé à une approche uniquement basée sur des émulateurs, il apporte un niveau de réalisme bien supérieur.

Points forts

- Plateforme mature et largement adoptée
- Large couverture de devices et de versions d'OS
- Intégration fluide avec les pipelines CI/CD
- Support des principaux frameworks d'automatisation
- Centralisation des artefacts de tests (logs, vidéos, rapports)

Points de vigilance

- Coût pouvant augmenter rapidement si les campagnes ne sont pas maîtrisées
- Dépendance à la plateforme et à la connectivité réseau
- Certaines limitations sur les scénarios nécessitant un accès physique prolongé au device
- Courbe de prise en main pour les équipes peu habituées aux tests cloud

SIGNATURE DU MAG

La vérité du terrain - Rappel

Une ferme d'appareils ne règle pas tous les problèmes de test mobile.

Elle élargit la couverture, mais ne reproduit pas toujours fidèlement les conditions réelles d'usage : qualité réseau locale, interactions physiques prolongées, comportements spécifiques à certains capteurs.

Utilisée seule, elle peut donner une fausse impression de maîtrise.

Tarifs indicatifs observés en mai 2026, susceptibles d'évoluer. :

Essai gratuit — 28 jours d'essai

39$ /mois — Tests manuels multiplateformes et d'applications mobiles sur des milliers de navigateurs de bureau, de véritables appareils mobiles, d'émulateurs et de simulateurs

149$ /mois — Tests automatisés et manuels multiplateformes pour applications mobiles et navigateurs sur des milliers de combinaisons navigateur/système d'exploitation de bureau, émulateurs et simulateurs mobiles.

199$ /mois — Tests automatisés et manuels de compatibilité multiplateforme et d'applications mobiles sur des milliers d'appareils mobiles réels

Le tarif affiché est par parallèle de test. Ce détail est important car dans une stratégie CI/CD à grande échelle, le coût réel dépendra du nombre de sessions simultanées que vous voulez exécuter.

Ce n'est pas un "prix fixe" pour toute l'équipe, mais plutôt un coût unitaire qu'on étend selon les besoins.

Exemple d'un test manuel

Pour la présentation du mag, j'ai crée un compte trial gratuit de 28 jours. Il me donne accès à quelques mobiles d'exemples. J'ai également utilisé leurs applications de démonstration :

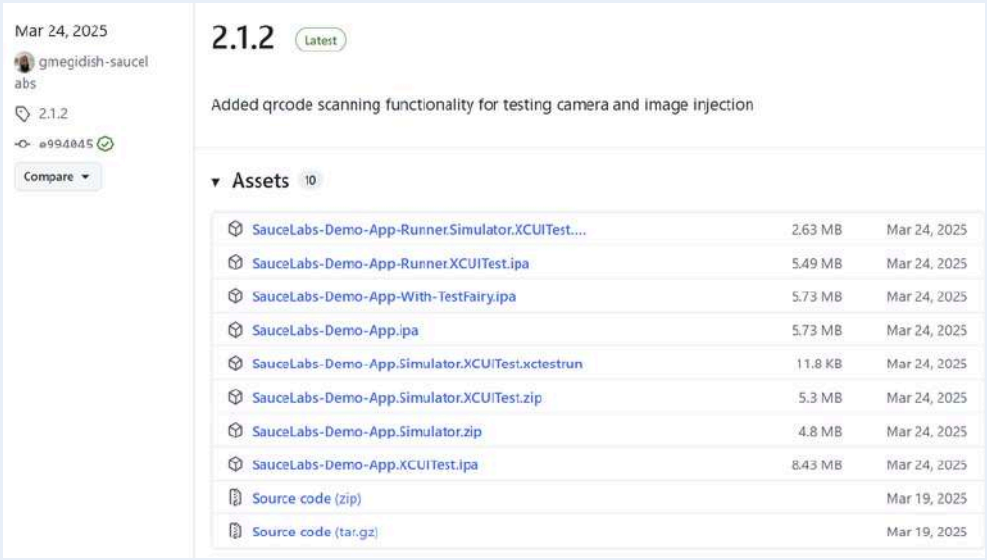
L'objectif ici n'est pas de recommander Sauce Labs plutôt qu'un autre outil, mais d'illustrer concrètement ce qu'une ferme d'appareils permet de faire. Le même raisonnement doit être appliqué à BrowserStack, Kobiton ou toute autre solution selon votre contexte.

Pour iOS :

J'ai utilisé leur repo :
https://github.com/Sauce Labs/my-demo-app-ios

A droite, il y a un encart Releases, c'est ici que vous allez retrouver les versions .ipa dont vous allez avoir besoin pour tester sauce labs.

A l'heure de l'écriture du mag, il y a plusieurs assets dans la version 2.1.2 :



Le rôle de chaque asset est :
Sauce Labs-Demo-App.ipa : c'est l'app iOS principale.

Sauce Labs-Demo-App-With-TestFairy.ipa : c'est une variante de la même app avec l'intégration TestFairy / Mobile App Distribution.

Sauce Labs-Demo-App-Runner.XCUITest.ipa : là, ce n'est plus un test manuel. C'est l'asset de test runner / testApp pour de l'automatisation XCUITest sur vrais devices.

Sauce Labs-Demo-App.XCUITest.ipa : celui-ci est aussi orienté automatisation XCUITest.

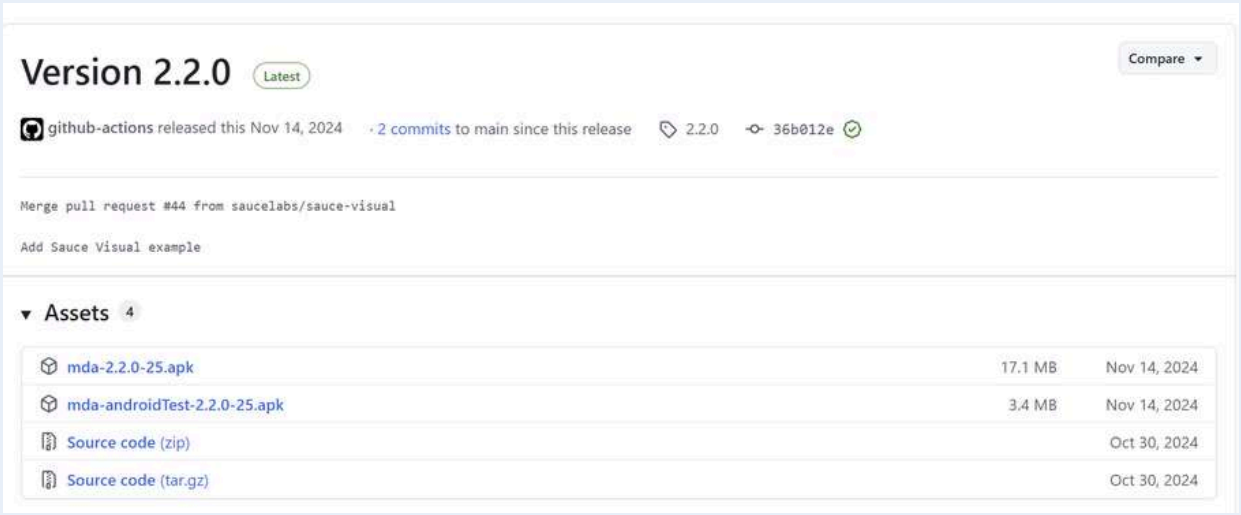
Sauce Labs-Demo-App.Simulator.XCUITest.xctestrun : c'est le fichier de configuration de test pour XCTest/XCUITest. Sauce Labs explique que le fichier .xctestrun contient la config compilée du plan de test et qu'il est requis pour exécuter un XCTest plan.

Pour android :

J'ai utilisé leur repo : https://github.com/Sauce Labs/my-demo-app-android

A droite, il y a un encart Releases, c'est ici que vous allez retrouver les versions .apk dont vous allez avoir besoin pour tester sauce labs.

A l'heure de l'écriture du mag, il y a plusieurs assets dans la version 2.2.0 :



Le rôle de chaque asset est :

- mda-2.2.0-25.apk

C'est l'application Android elle-même

- mda-androidTest-2.2.0-25.apk

C'est l'APK de tests instrumentés Android

Qu'est ce qu'un test instrumentés ?

Ce sont les tests exécutés sur appareil réel ou émulateur, packagés dans un APK séparé (androidTest), utiles notamment pour les tests UI et les scénarios dépendants du framework Android.

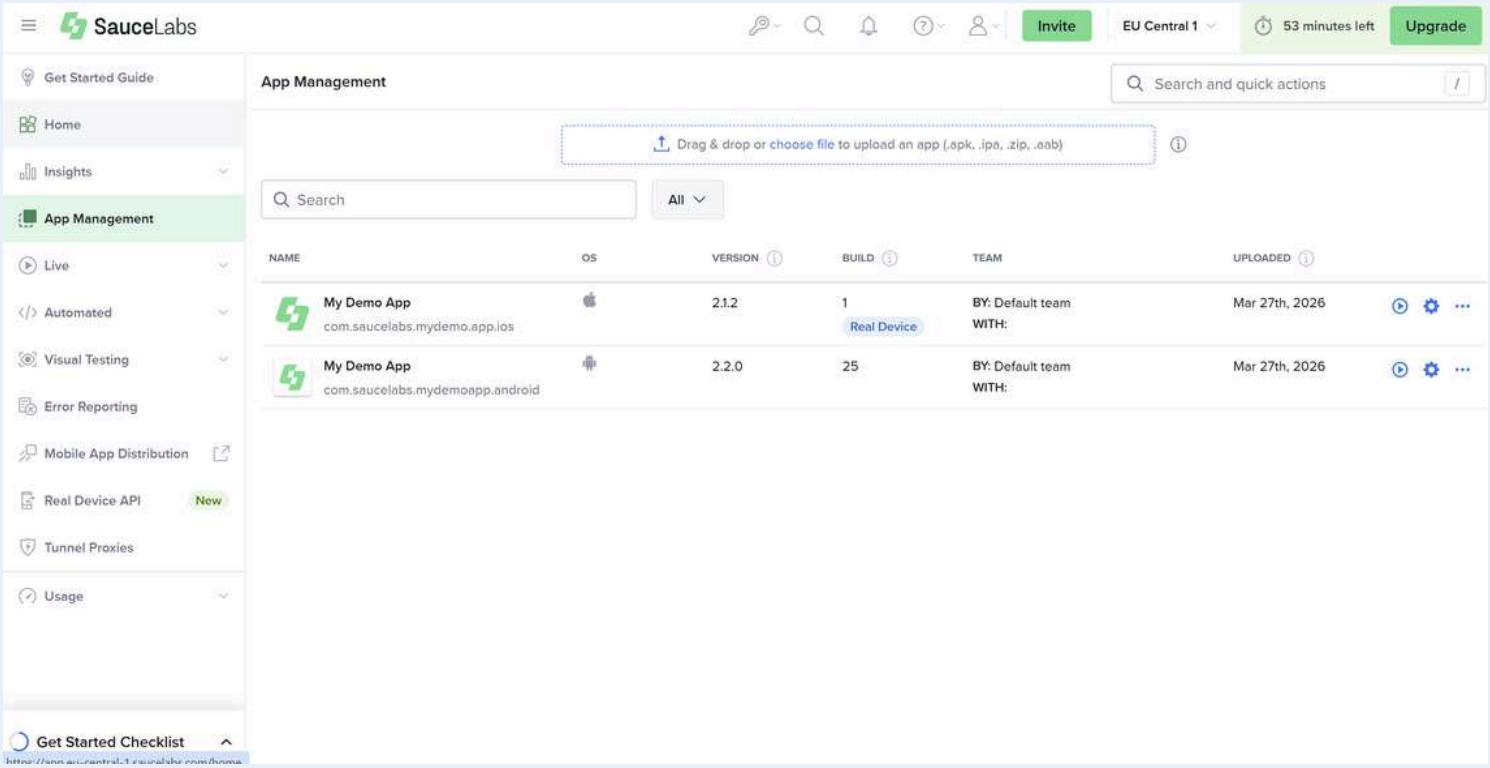
Sources:
https://developer.android.com/training/testing/instrumented-tests?hl=fr

https://developer.android.com/studio/test/test-in-android-studio?hl=fr

Importer l'application dans Sauce Labs

Avant de lancer un test manuel ou automatisé, il faut d'abord déposer l'application dans App Management. Cette étape permet de rendre l'APK ou l'IPA disponible dans la plateforme pour les futures sessions de test :

- uploader l'APK / IPA,
- vérifier que l'app apparaît bien dans la liste

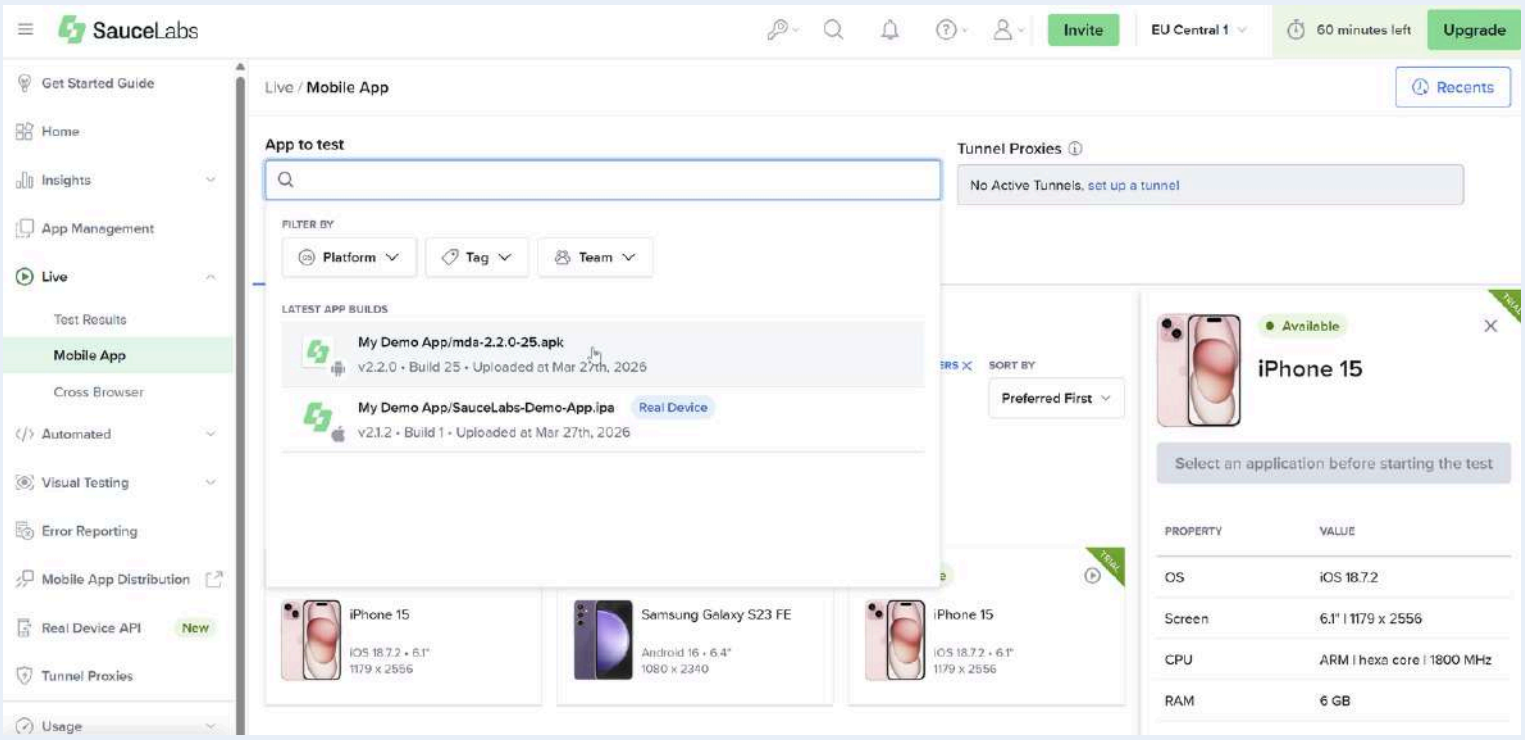


Exemple d'un test manuel

Dans Live puis dans Mobile App ou en cliquant sur le bouton play sur la ligne de votre asset, la vue de Sauce Labs ci dessous vous montre la sélection des appareils pour les tests mobiles manuels. La plateforme permet de choisir entre terminaux réels et virtuels, puis de filtrer les équipements selon la plateforme, la version de l'OS ou le constructeur. L'objectif est de lancer rapidement une session de test sur un device compatible avec l'application à valider.

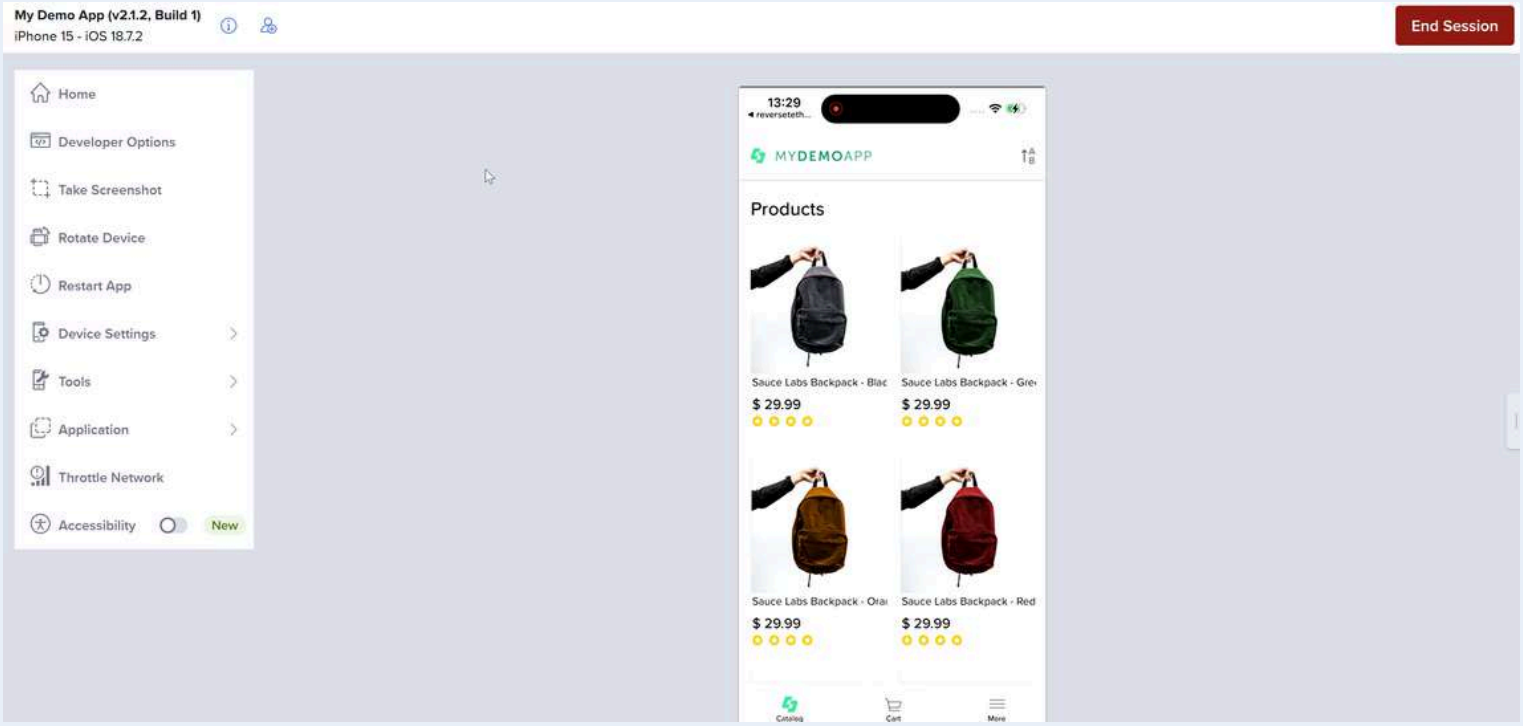
SIGNATURE DU MAG

L'intérêt d'un ferme de mobile n'est pas seulement technique.
Il est aussi organisationnel : elle réduit le temps perdu à préparer les appareils, à gérer leur disponibilité et à maintenir un laboratoire physique trop limité.

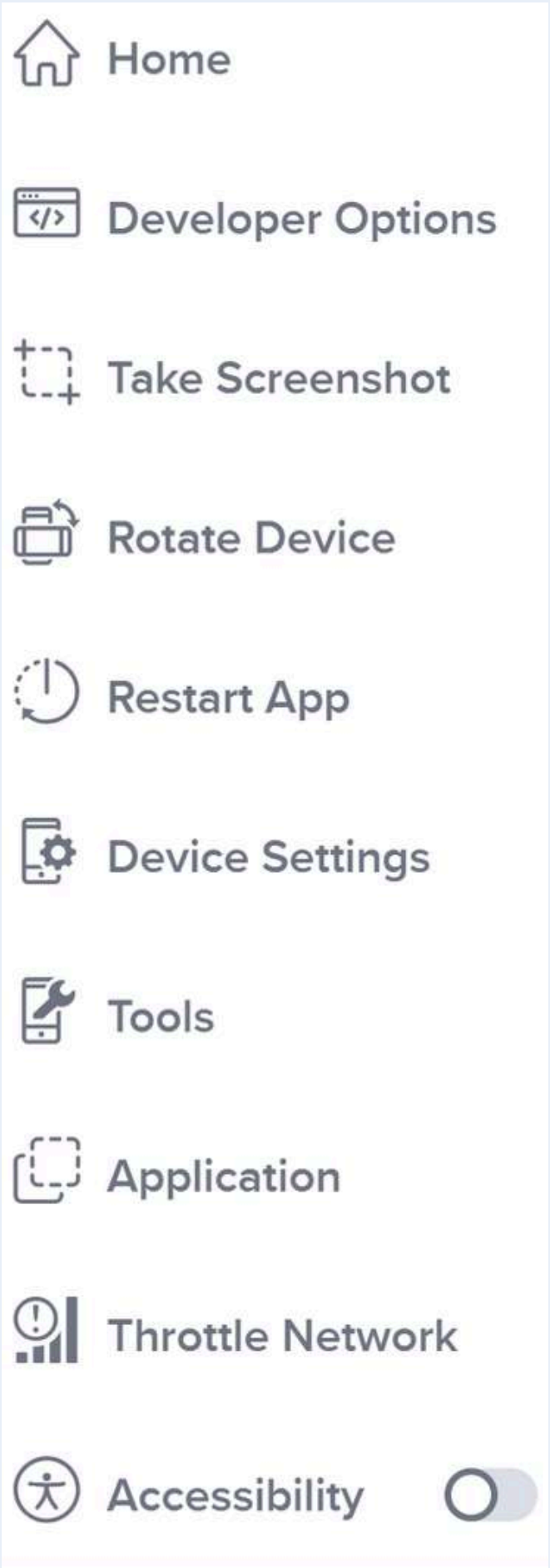


Vous devez selectionner l'application à tester puis un mobile. Tant qu'aucune application n'est sélectionnée, Sauce Labs empêche le démarrage de la session sur le device choisi. Cette étape évite de lancer un téléphone "dans le vide" et structure le parcours de test manuel. Pour tester sur des environnements privés non exposés sur Internet, comme une préproduction interne ou des API protégées, vous pouvez utilise le tunnel proxy.

Avec l'exemple, il n'y a pas besoin de proxy. Pour les tests, j'ai pris l'iphone 15. En cliquant sur le bouton play, l'application s'installe. Cela peut prendre quelques secondes.



Le menu :



Ramène à l'écran d'accueil du téléphone distant. C'est utile pour repartir d'un état neutre ou sortir rapidement de l'application.

Donne accès à des options techniques liées à la session. Selon la configuration, cela peut servir à afficher des informations de debug ou à manipuler certains réglages utiles pendant l'analyse.

Permet de capturer un écran pendant la session. Pratique pour documenter un bug, illustrer une anomalie visuelle ou garder une preuve d'un comportement observé.

Fait basculer l'appareil en portrait ou paysage. C'est utile pour vérifier le comportement responsive, l'affichage et la stabilité de l'interface selon l'orientation.

Relance l'application sans redémarrer toute la session. Très pratique pour rejouer un scénario depuis le début ou vérifier si un problème persiste après redémarrage.

Donne accès à certains réglages du terminal. Cela permet en général de préparer le contexte de test côté appareil plutôt que côté application.

Regroupe des outils complémentaires de manipulation ou d'analyse pendant la session. C'est souvent là qu'on retrouve les aides avancées proposées par la plateforme.

Concerne les actions sur l'application testée elle-même. Cela peut servir à consulter ou modifier certains paramètres liés à l'app dans la session.

Permet de dégrader ou limiter artificiellement le réseau. C'est très intéressant pour simuler des conditions réelles moins favorables : réseau lent, instable, ou contraint.

Active des options liées à l'accessibilité. C'est pertinent pour observer certains comportements de l'application avec des aides d'accessibilité ou dans une logique de vérification inclusive.

Les outils de simulation réseau permettent de reproduire différents contextes d'usage.

Pas de connexion (No Network / 100% packet loss) : Aucune donnée ne circule.
Vous pouvez alors tester, par exemple :

- affichage des messages hors ligne
- gestion des erreurs API
- fallback UI (écrans vides, placeholders)
- mécanismes de cache ou mode offline

Réseau très dégradé (2G / Edge) : débit très faible, latence élevée.
Vous pouvez alors tester, par exemple :

- temps de chargement des écrans
- comportement des loaders (spinners, skeletons)
- timeouts côté front
- expérience utilisateur (frustration, abandon)

Réseau lent mais stable (3G slow) : connexion fonctionnelle mais lente.
Vous pouvez alors tester, par exemple :

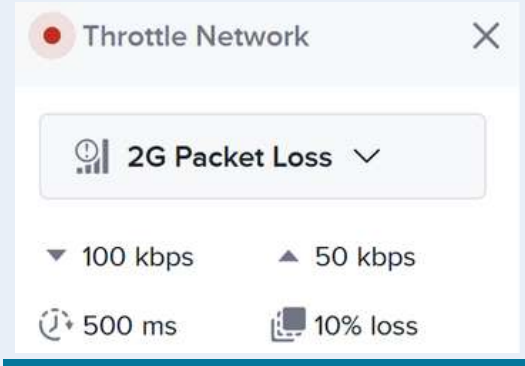
- performance perçue
- priorisation du contenu (lazy loading, progressive loading)
- affichage progressif des données

Réseau instable (packet loss) : perte partielle des paquets, comportement imprévisible.
Vous pouvez alors tester, par exemple :

- gestion des retries
- robustesse des appels API
- cohérence des données affichées
- erreurs intermittentes

Réseau nominal (4G / WiFi) : conditions idéales.
Vous pouvez alors tester, par exemple :

- parcours utilisateur standard
- validation fonctionnelle classique
- automatisation des scénarios critiques



SIGNATURE DU MAG

Simulation réseau vs réalité

Les outils de throttling ne reproduisent pas parfaitement :

- **les pertes de réseau réelles**
- **les variations locales**
- **les comportements opérateurs**

AVIS D'EXPERTE

Caroline Nazin

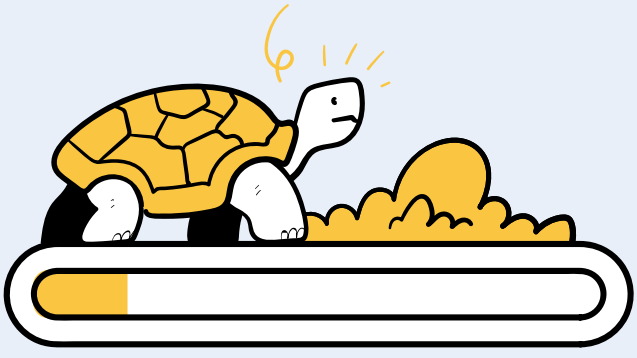
🪲 </Quality Engineer Web> & 👩‍💻 </Cheffe de projet fonctionnel> · Luxe & E-commerce international · AI-Powered QE · Automation

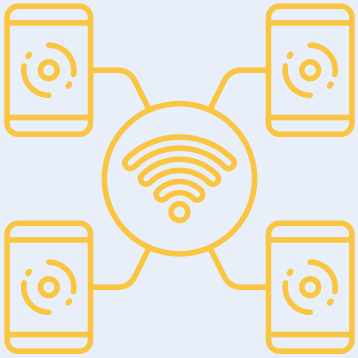
Modes dégradés réseau

Dégrader le réseau ce n'est pas juste simuler une 3G lente. C'est aussi tester le passage Wi-Fi vers 4G qui change l'IP et peut invalider la session, l'entrée dans un tunnel avec timeouts partiels, les connexions captives des hôtels qui cassent le certificate pinning, le roaming qui ajoute de la latence sans baisser le débit, et le Great Firewall chinois qui filtre Google Fonts et reCAPTCHA.

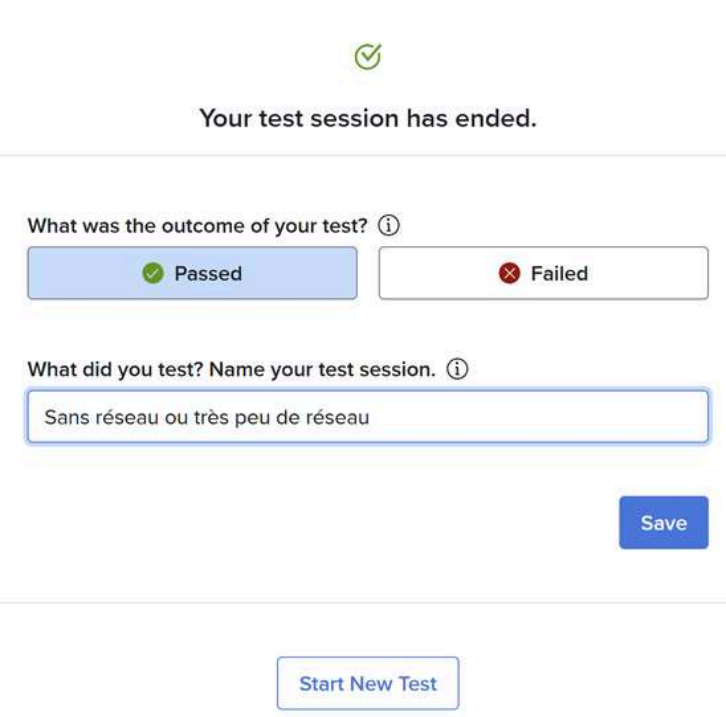
J'ai eu le cas d'un composant de paiement qui chargeait une ressource Google Fonts en Chine et qui bloquait tout le checkout, totalement invisible depuis nos environnements européens, on l'a découvert par un signalement utilisateur.

Une checklist des scénarios réseau avec leurs bugs typiques serait très utile.





FIN DU TEST



À la fin d'une session manuelle, Sauce Labs demande au testeur de qualifier le résultat :

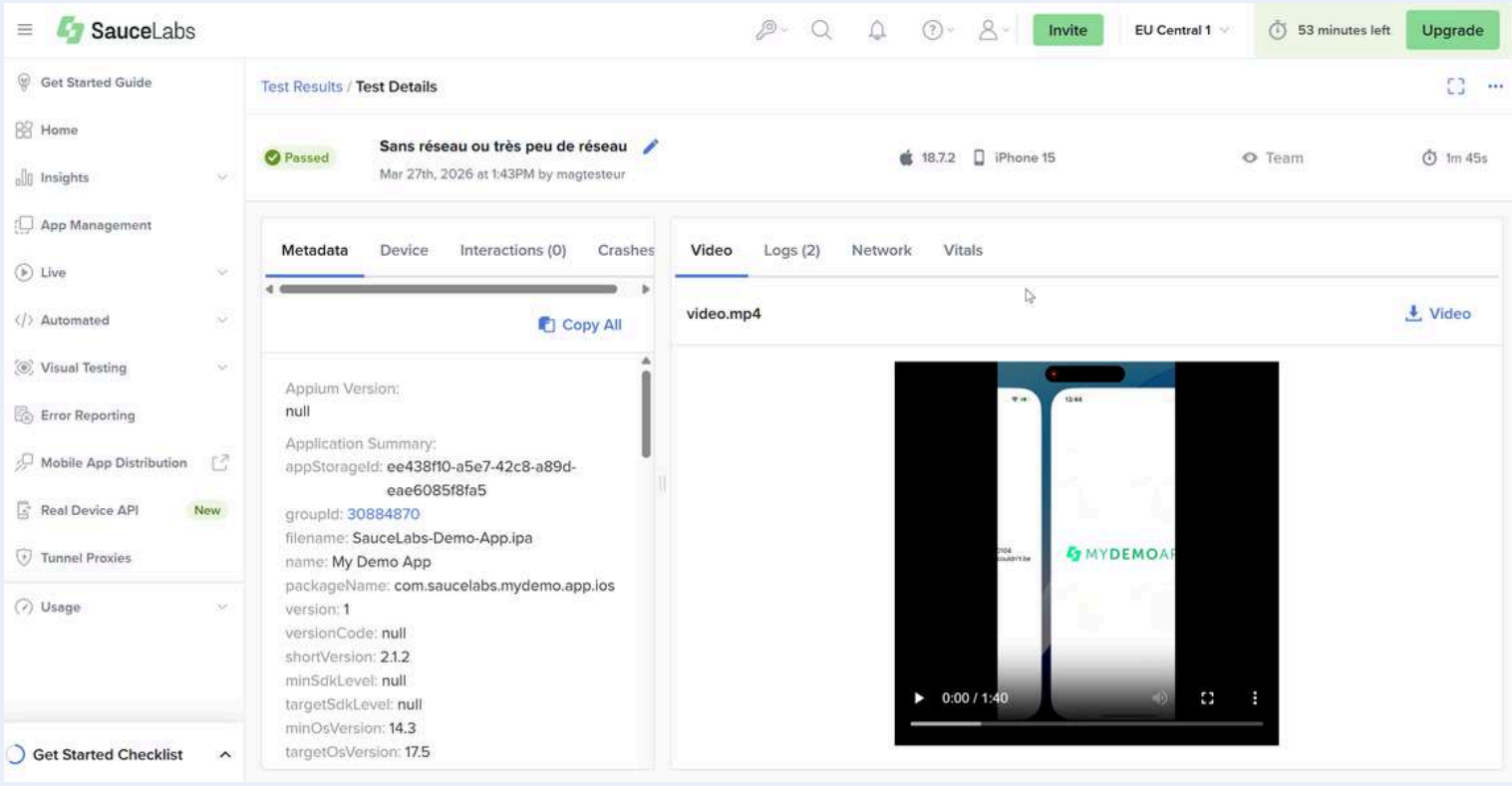
- Passed : le test est considéré comme valide
- Failed : une anomalie a été observée

Vous pouvez nommer votre test. Nous avons par exemple tester le réseau 2G Packet Loss. Vous pouvez également spécifier le build de votre application. Cela simplifie la recherche mais cette information peut être retrouvée par la suite.

Après avoir cliqué sur Sauvegarder, une page s'ouvre. Plusieurs informations s'y trouvent :

- tout ce qui est métadata avec le nom de votre build
- tout ce qui est information sur votre device : OS, nom du téléphone, ...
- la vidéo que vous pouvez visionner et/ou télécharger grâce au bouton vidéo
- les logs pour voir s'il y a eu des erreurs et les transmettre à l'équipe de développement

Toutes ces informations vous permettent d'assurer la traçabilité, notamment en cas d'investigation ou de reproduction d'un bug.



REX : LES PIEGES

L'utilisation de fermes de devices cloud apporte une grande flexibilité, mais introduit aussi des biais qu'il est important de connaître.

Un affichage parfois trompeur

Sur iOS, l'affichage via une ferme cloud peut parfois paraître légèrement flou ou "compressé". Ce phénomène est lié au streaming vidéo du device distant et non à l'application elle-même.

Il est donc recommandé de ne pas baser des validations visuelles fines uniquement sur ce type d'environnement.

Par exemple, lors de mes missions, si l'affichage en direct me paraît flou, je vérifie également la vidéo de session. Celle-ci est généralement de meilleure qualité et permet de confirmer si le problème vient du rendu réel ou simplement du streaming. Vous pouvez voir plus haut comment visualiser ou télécharger les vidéos de vos l'éxécutions de vos tests

Latence et interactions

Les interactions peuvent parfois sembler moins fluides :

- délai lors des actions
- gestes tactiles moins précis
- animations légèrement dégradées

Ces effets sont liés à la latence réseau et ne reflètent pas toujours le comportement réel sur device.

OUTILS ET FRAMEWORKS D'AUTOMATISATION

MAESTRO

Maestro est un framework de tests UI mobile et web qui met en avant des scénarios écrits sous forme de flows YAML.

Ce que Maestro change

- les scénarios peuvent être lus plus facilement par des profils QA, PO ou produit
- l'écriture est plus déclarative, moins centrée sur le code
- les flows peuvent servir de support de discussion autour des parcours critiques
- la barrière d'entrée peut être plus basse pour commencer à automatiser
- la maintenance reste nécessaire, mais le format rend souvent les intentions plus visibles.

SIGNATURE DU MAG

Maestro peut être utilisé en complément d'Appium ou Detox. Par exemple, une équipe React Native peut conserver Detox pour les tests E2E proches du code, tout en utilisant Maestro pour des smoke tests lisibles, rapides à maintenir et compréhensibles par des profils QA, produit ou métier.

La comparaison avec Gherkin est tentante, car les scénarios Maestro sont lisibles. Mais la différence est importante : Gherkin décrit un comportement, tandis qu'un flow Maestro est directement exécutable.

```
appId: com.example.app
---
- launchApp
- tapOn: "Se connecter"
- tapOn: "Email"
- inputText: "feedback@lemagdutesteur.fr"
- tapOn: "Mot de passe"
- inputText: "motdepasse123"
- tapOn: "Connexion"
- assertVisible: "Bienvenue"
```

CHOISIR UN FRAMEWORK MOBILE

Un outil de test mobile ne se choisit pas parce qu'il est populaire, mais parce qu'il correspond au type d'application, à l'équipe, au niveau d'accès au code, au besoin de couverture et aux contraintes de maintenance.

Le choix du framework influence la manière d'écrire les scénarios, la stabilité des tests, le niveau de collaboration avec les développeurs, la capacité à exécuter en CI/CD et la facilité à diagnostiquer les échecs.

Le piège classique consiste à chercher l'outil universel : celui qui couvrirait Android, iOS, webview, natif, hybride, cloud, local, tests fonctionnels, accessibilité et non-régression. En pratique, chaque outil impose un compromis.

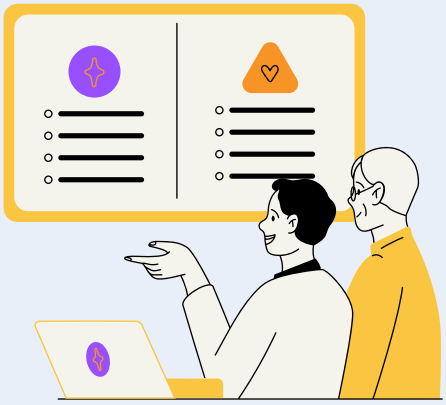
SIGNATURE DU MAG

Le mauvais choix d'outil crée de la dette de test. Le bon choix accepte ses limites dès le départ.

Les questions à vous poser avant de choisir :

- L'application est-elle native, hybride, React Native, Flutter, ou principalement web embarqué ?
- L'équipe QA a-t-elle accès au code, aux builds, aux identifiants techniques et aux environnements ?
- Le besoin principal est-il cross-platform, stabilité native, lisibilité métier ou vitesse de feedback ?
- Les tests doivent-ils tourner sur simulateur, émulateur, devices réels, ferme cloud ou pipeline CI/CD ?
- Quels parcours justifient réellement une automatisation durable ?

QUEL OUTIL POUR QUEL CONTEXTE ?



Comparer les outils n'a de sens que si l'on compare aussi les contextes : application native ou hybride, équipe QA ou dev, besoin Android/iOS, accès au code et niveau de maintenance accepté

Outil	Cible principale	Forces	Limites / vigilance
Appium	Android, iOS, web mobile, apps natives/hybrides	Approche cross-platform, écosystème WebDriver, plusieurs langages, adapté aux équipes QA outillées	Configuration plus lourde, dépendance aux drivers/capabilities, maintenance des sélecteurs
Espresso	Applications Android natives	Tests rapides et fiables côté Android, synchronisation avec l'UI, très adapté aux équipes proches du code	Moins naturel pour du pur black-box, nécessite une proximité avec l'app Android
XCUITest	Applications iOS natives	Intégré à Xcode/XCTest, adapté aux tests UI iOS, bon alignement avec l'écosystème Apple	Spécifique iOS, nécessite souvent des compétences Swift/Xcode
Detox	Applications React Native	Pensé pour React Native, scénarios E2E en JavaScript, logique cross-platform iOS/Android	Pertinent surtout si l'application est React Native et si l'équipe accepte son modèle gray-box

APPIUM

Le choix cross-platform, puissant mais exigeant

Appium est un projet open source conçu pour automatiser des interfaces sur plusieurs plateformes, notamment mobile. Son intérêt principal, dans une stratégie de test mobile, est de proposer une approche transverse : un même écosystème peut piloter Android, iOS, des applications natives, hybrides ou web mobile selon les drivers utilisés.

Quand Appium est pertinent

- votre équipe QA veut maintenir une logique commune entre Android et iOS
- vos scénarios de non-régression doivent couvrir les parcours métier critiques
- vos tests doivent pouvoir tourner sur devices réels ou fermes cloud
- votre équipe souhaite utiliser des langages déjà connus comme Java, JavaScript, Python ou C#
- votre produit comporte des parcours similaires entre plateformes.

Points de vigilance

- la configuration peut être plus lourde qu'avec un outil natif
- les capabilities et les drivers doivent être bien maîtrisés
- les sélecteurs doivent être pensés avec les développeurs (c'est valable pour beaucoup d'outils!)
- les gestes mobiles, WebViews, permissions et contextes hybrides doivent être explicitement testés
- un test vert ne garantit pas une bonne expérience réelle (encore une fois, c'est valable pour beaucoup d'outils !)

```
describe('Connexion mobile', () => {
  it('connecte un utilisateur valide', async () => {
    await $('~email').setValue('feedback@lemagdutesteur.fr');
    await $('~password').setValue('motdepasse123');
    await $('~login-button').click();

    await expect($('~home-title')).toHaveText('Bienvenue');
  });
});
```

ESPRESSO ET XCUITEST

Contrairement à Appium, qui vise une automatisation multiplateforme, Espresso et XCUITest restent liés à leur plateforme respective. Ils sont souvent plus proches du code applicatif et plus adaptés aux équipes qui veulent tester finement des comportements natifs.

Espresso côté Android

Espresso est le framework Android destiné à écrire des tests d'interface concis et fiables. Sa force vient notamment de sa capacité à se synchroniser automatiquement avec l'interface, ce qui réduit certains problèmes classiques de timing dans les tests UI.

Il est particulièrement pertinent quand l'équipe Android est impliquée dans l'automatisation, que les tests vivent près du code, et que l'objectif est de sécuriser des composants ou parcours Android de manière stable.

XCUITest côté iOS

Côté Apple, les tests d'interface s'appuient sur XCTest et XCUIAutomation. L'intérêt est de rester dans l'environnement Xcode, avec une intégration naturelle aux projets iOS et à l'écosystème Apple.

XCUITest est donc intéressant lorsque l'équipe iOS veut automatiser des parcours proches de la réalité utilisateur, avec des tests alignés sur les contraintes propres à l'app et à la plateforme.

Ce que cela change pour la QA

- La maintenance dépend davantage de la collaboration QA/dev
- Les tests peuvent être plus stables, mais moins mutualisés entre Android et iOS
- La stratégie doit accepter deux socles techniques différents
- Ces outils sont très forts pour du natif, mais moins adaptés si l'équipe cherche une seule suite lisible par tous

SIGNATURE DU MAG

Espresso et XCUITest sécurisent le natif.
Appium et Maestro sécurisent le parcours.
Les premiers sont proches du code.
Les seconds sont proches de l'utilisateur.
Une bonne stratégie mobile ne choisit pas forcément un camp : elle combine les niveaux de test selon les risques à couvrir.

DETOX

Quand React Native influence la stratégie de test

Detox est un framework open source de test end-to-end pour les applications React Native. Sa promesse est d'exécuter l'application sur simulateur, émulateur ou device et d'interagir avec elle comme un utilisateur, tout en s'inscrivant dans l'écosystème JavaScript/React Native.

Quand Detox est pertinent

- l'application est majoritairement React Native
- l'équipe maîtrise déjà JavaScript et l'écosystème React Native
- les développeurs participent à l'écriture ou à la maintenance des tests
- les parcours critiques sont suffisamment stables pour être automatisés
- la CI/CD doit exécuter des scénarios E2E régulièrement.

Points de vigilance

- Detox n'est pas le choix naturel pour toutes les applications mobiles
- son intérêt dépend fortement de l'architecture React Native
- les tests peuvent rester sensibles aux états, aux données et aux timings
- les parcours WebView, natifs spécifiques ou dépendants de services externes doivent être cadrés

```
describe('Login', () => {
  beforeAll(async () => {
    await device.launchApp();
  });

  it('connecte un utilisateur valide', async () => {
    await element(by.id('email')).typeText('feedback@lemagdutesteur.fr');
    await element(by.id('password')).typeText('motdepasse123');
    await element(by.id('loginButton')).tap();

    await expect(element(by.id('homeScreen'))).toBeVisible();
  });
});
```

EXEMPLE

Un scénario fonctionnel peut être automatisé de plusieurs façons selon l'objectif de test, le niveau de fiabilité attendu et les compétences de l'équipe.

Prenons un cas simple :

Scénario : achat rapide sur mobile

1. **Ouvrir l'application**
2. **Se connecter**
3. **Rechercher un produit**
4. **L'ajouter au panier**
5. **Vérifier que le panier contient bien l'article**

Ce scénario peut être automatisé avec une approche orientée E2E multiplateforme, une approche lisible et légère, ou une approche native proche du code.

Approche 1 : Appium

```
const { remote } = require('webdriverio');

(async () => {
  const driver = await remote({
    capabilities: {
      platformName: 'Android',
      'appium:automationName': 'UiAutomator2',
      'appium:deviceName': 'Pixel_7',
      'appium:app': '/path/to/app.apk'
    }
  });

  await driver.$('~login-button').click();
  await driver.$('~email-input').setValue('feedback@lemagdutesteur.fr');
  await driver.$('~password-input').setValue('Password123');
  await driver.$('~submit-login').click();

  await driver.$('~search-input').setValue('montre');
  await driver.$('~product-card-0').click();
  await driver.$('~add-to-cart').click();

  const cartBadge = await driver.$('~cart-badge').getText();

  if (cartBadge !== '1') {
    throw new Error(`Panier attendu à 1, obtenu : ${cartBadge}`);
  }

  await driver.deleteSession();
})();
```

Approche 2 : Maestro

Maestro peut être utilisé en complément, pour des flows très lisibles, rapides à écrire, faciles à relire par QA / PO / métier, ou pour faire de la smoke test mobile.

```
appId: com.example.shop
---
- launchApp
- tapOn: "Se connecter"
- tapOn: "Email"
- inputText: "feedback@lemagdutesteur.fr"
- tapOn: "Mot de passe"
- inputText: "Password123"
- hideKeyboard
- tapOn: "Connexion"

- tapOn: "Rechercher"
- inputText: "montre"
- tapOn: "Montre Classic"
- tapOn: "Ajouter au panier"

- assertVisible: "1 article"
```

Approche 3 : Espresso / XCUITest

Espresso côté Android et XCUITest côté iOS sont pensés pour tester au plus près de l'application native. Ils sont souvent plus stables et rapides, mais demandent une proximité plus forte avec le code et les équipes de développement.

```
@Test
fun addProductToCart() {
    onView(withId(R.id.loginButton)).perform(click())

    onView(withId(R.id.emailInput))
        .perform(typeText("feedback@lemagdutesteur.fr"), closeSoftKeyboard())

    onView(withId(R.id.passwordInput))
        .perform(typeText("Password123"), closeSoftKeyboard())

    onView(withId(R.id.submitLogin)).perform(click())

    onView(withId(R.id.searchInput))
        .perform(typeText("montre"), closeSoftKeyboard())

    onView(withText("Montre Classic")).perform(click())
    onView(withId(R.id.addToCartButton)).perform(click())

    onView(withId(R.id.cartBadge))
        .check(matches(withText("1")))
}
```

GARTNER

Le Magic Quadrant (ou cadran magique) est une méthode d'analyse publiée par Gartner, un cabinet mondialement connu pour ses études technologiques.

Il sert à représenter sur un graphique la position des principaux fournisseurs d'un marché donné : IA, cybersécurité, outils de test, cloud, etc.

L'objectif est d'aider les entreprises à comprendre qui sont les acteurs fiables, qui innovent, qui exécutent réellement, et qui sont encore en progression.

Le graphique : deux axes, quatre zones

Le cadran se compose de deux axes :

- Axe horizontal : la vision. Il évalue la stratégie long terme du fournisseur : innovation, compréhension des besoins du marché, évolution du produit, pertinence des orientations techniques.

- Axe vertical : la capacité à réaliser. Il analyse tout ce qui concerne l'exécution : qualité du produit, stabilité, satisfaction client, support, solidité financière, mise en œuvre, adoption réelle.

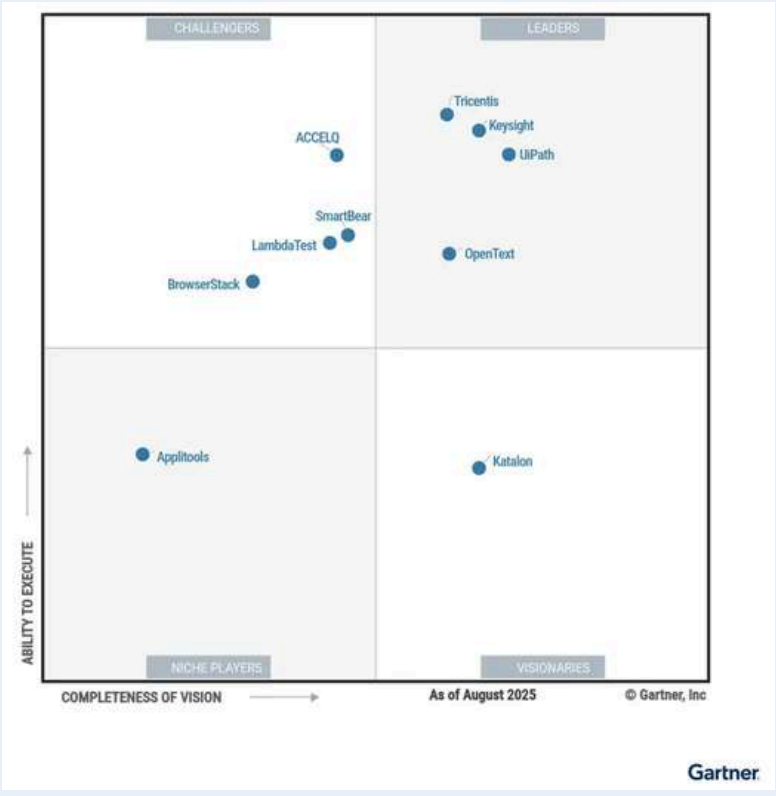
Ces deux axes créent quatre zones :

1) Leaders
Ce sont les acteurs qui combinent vision forte et exécution solide.
Ils influencent le marché et sont souvent choisis par les grandes entreprises.

2) Visionnaires
Ils ont beaucoup d'idées et une direction très moderne, mais leur mise en œuvre est encore perfectible.
Ils innovent rapidement.

3) Challengers
Les solutions fonctionnent très bien au quotidien: produit stable, clients satisfaits, adopté à grande échelle.
Mais leur stratégie manque parfois d'ambition ou de différenciation.

4) Niche Players
Ils adressent souvent un besoin très précis, ou une zone géographique réduite.
Ils sont fiables dans leur domaine, mais pas forcément adaptés à tous les usages.



SIGNATURE DU MAG

un Magic Quadrant n'est pas une preuve d'adéquation à votre contexte

Le 11 novembre 2025, Keysight a annoncé avoir été classée « Leader » dans le rapport 2025 Gartner Magic Quadrant for AI-Augmented Software Testing Tools.

Le rapport évalue les éditeurs en fonction de leur capacité à exécuter et de la cohérence de leur vision dans le domaine en pleine évolution des outils de test pilotés par l'IA.

Keysight a mis en avant son portefeuille de solutions, notamment via la plateforme Eggplant, incluant des fonctionnalités variées :

- génération automatique de cas de test (« Autonomous Test Design ») dans des environnements sécurisés, ce qui permet d'augmenter la couverture tout en préservant la confidentialité des données
- validation visuelle en temps réel, pour détecter les régressions d'interface, les anomalies d'affichage ou des erreurs sur des systèmes embarqués, des aspects souvent difficiles à couvrir avec les tests purement "classiques".
- compatibilité avec des environnements variés (cloud, on-premises ou même "air-gapped"), ce qui la rend pertinente pour des secteurs fortement réglementés comme la santé, la défense, l'aéronautique ou la finance

Ce que cela signifie pour la QA & l'industrie

- **Le rapport indique que l'usage des outils de test augmentés par l'IA devrait s'accélérer : d'environ 20 % des entreprises en début 2025, à 70 % d'ici 2028.**
- **Pour les équipes QA et les entreprises soucieuses de la qualité logicielle, c'est un signal fort : l'automatisation va probablement dépasser les tests unitaires ou fonctionnels classiques, en intégrant davantage d'intelligence dans la conception, l'exécution et la maintenance des tests.**
- **Plutôt que de remplacer les testeurs, ces outils visent à libérer les équipes des tâches répétitives, afin qu'elles se concentrent sur l'analyse, la stratégie de test, la qualité utilisateur, la conformité, etc.**

QU'EST CE QUE LE COMPUTER VISION ?

La computer vision (ou vision par ordinateur) est un ensemble de techniques qui permettent à une machine de :

- analyser une image,
- reconnaître des objets,
- comprendre une scène,
- prendre une décision en fonction de ce qu'elle "voit".

C'est l'équivalent pour une machine de notre perception visuelle.

Pourquoi c'est utile dans le test logiciel ?

La computer vision permet à l'outil de repérer visuellement un bouton, un formulaire, une icône, un texte, une couleur, un état d'écran, sans dépendre du code.

Concrètement :

- l'outil regarde l'écran comme un humain,
- il identifie un élément visuel,
- il clique dessus,
- il vérifie que la bonne action s'est produite.

Vous n'avez pas besoin du DOM, d'un sélecteur CSS, d'un ID, d'un XPath.

C'est indépendant du code et de la technologie.

Par exemple :

Vous voulez cliquer sur un bouton "Envoyer".

Avec un outil classique : vous cherchez un sélecteur CSS ou XPath (souvent fragile).

Avec la computer vision : l'outil cherche l'apparence du bouton "Envoyer" (couleur, forme, texte).

Même si la technologie change, même si l'élément n'a pas d'ID, même si l'UI est dans un environnement virtualisé, l'outil le voit.

QU'EST CE QUE L'OCR?

L'OCR signifie Optical Character Recognition, ou reconnaissance optique de caractères.

C'est une technologie qui permet à un ordinateur de lire du texte présent dans une image, comme si c'était du texte normal.

Quand un texte n'est pas du vrai texte (exemple : une capture d'écran, une photo, un PDF scanné…), un ordinateur ne peut normalement pas le comprendre.

L'OCR analyse :

- les formes des lettres,
- la disposition,
- l'orientation,
- les variations de polices,
- la texture visuelle,
- les symboles complexes,

… puis reconstruit le texte sous une forme exploitable.

Par exemple :

- Lire du texte dans une capture d'écran

Un montant bancaire, un identifiant, un message d'erreur…

- Lire une facture en PDF scannée

L'OCR "transforme" les pixels en mots utilisables.

- Lire du texte sur une interface d'application non accessible

Très utile pour les logiciels anciens, les écrans virtualisés ou les interfaces d'ERP industriels.

- Lire du texte en plusieurs langues

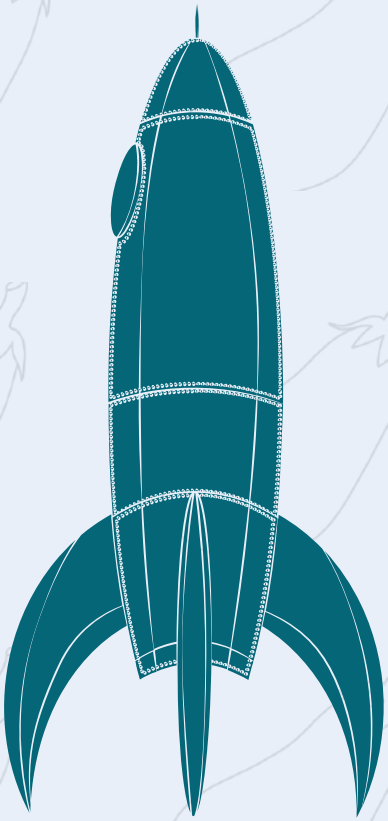
Même dans des alphabets non latins (chinois, japonais…), lorsque l'OCR est suffisamment performant.

EGGPLANT

L’univers du test logiciel évolue sans cesse, mais certains outils bousculent véritablement les habitudes.

Eggplant fait partie de ceux-là.
Sa particularité ? Il repose sur une combinaison puissante : vision par ordinateur + langage naturel, ce qui lui permet d’aborder les tests d’une manière totalement différente des frameworks traditionnels axés sur le DOM, le code ou les API.

Dans cet article, on plonge au cœur de ce qui fait la force d’Eggplant, de ses usages à ses avantages sur le terrain, notamment dans les environnements industriels, les applications complexes ou les systèmes embarqués.



2002

Eggplant est à l’origine une entreprise britannique, spécialisée dans l’automatisation de tests logiciels.

2020

l’éditeur a été acquis par Keysight Technologies.

Lorsque l'on parle d'automatisation de tests, on pense souvent à des outils dépendants du code, du DOM ou d'API internes.

Eggplant casse totalement ce cadre en s'appuyant sur la **computer vision**, **l'OCR** et le langage naturel pour offrir une nouvelle manière de tester : plus souple, plus universelle, et adaptée aux écosystèmes complexes.

Ce mode de fonctionnement le rend particulièrement adapté aux systèmes où les sélecteurs sont inexistants ou instables. Il est donc possible de tester :

- des clients lourds (ex : ServiceNow client lourd, Solife pour l'assurance vie)
- des écrans mainframe (AS400), des outils métier vieillissants mais toujours critiques
- des applications industrielles (CAD, PLM comme 3DX, MES, ERP)
- des applications mobiles Android / iOS
- des interfaces embarquées
- des parcours multi-briques impliquant plusieurs logiciels

Vrai testing mobile : interactions réelles

Eggplant ne simule pas seulement un mobile : il interagit avec un vrai appareil, appareil photo inclus.

Par exemple :

- scanner un QR code avec la caméra réelle du téléphone
- lancer un parcours contenant des étapes physiques
- réaliser un test de paiement nécessitant une action sur un terminal ou une app dédiée.

Cas client :

BANKSERVAFRICA

Pour le système de traitement des prélèvements bancaires "DebiCheck", la création manuelle des fichiers de test (XML, transactions multiples) était difficile, lente et sujette aux erreurs.

Avec Eggplant, l'automatisation a permis de générer des fichiers complexes en quelques minutes, là où c'était quasi impossible manuellement pour de gros volumes.

Cela a permis d'augmenter la couverture des tests, d'améliorer la consistance, et de réduire les erreurs de fichiers (taux d'erreur d'environ 20% ramené à zéro). Par conséquent, cela a également permis d'accroître l'efficacité, car l'analyse de chaque erreur prenait jusqu'à 30 minutes

ACCESSIBILITÉ ET QUALITÉ D'USAGE : TESTER CE QUE L'UTILISATEUR VIT VRAIMENT

C’est tout l’enjeu de l’accessibilité et de la qualité d’usage : ne pas s’arrêter à la simple réussite d’un scénario, mais vérifier si le parcours reste compréhensible, cohérent et réellement exploitable dans des conditions d’usage variées.

Sur ce sujet, le test mobile oblige à regarder au-delà de l’écran visible. Ce que voit l’équipe projet n’est pas toujours ce que perçoit l’utilisateur. Un enchaînement d’actions qui semble fluide à l’œil peut devenir laborieux avec un lecteur d’écran, incohérent au clavier, ou pénible lorsque l’ordre de navigation ne suit pas la logique du parcours.

L’importance du tab order

Lorsqu’un utilisateur navigue sans s’appuyer directement sur l’écran tactile, il a besoin d’un parcours logique : atteindre les éléments dans un ordre cohérent, comprendre immédiatement où il se trouve, et accéder à l’action principale sans effort inutile.

Un mauvais tab order ne rend pas forcément l’application “cassée”, mais il la rend vite fatigante.

Un bouton peut être techniquement accessible tout en arrivant trop tard dans la navigation.

Un champ important peut être annoncé après des éléments secondaires.

Un écran peut sembler propre visuellement, tout en devenant confus dès que l’on suit le focus au lieu de le regarder.

Tester l’ordre de navigation, c’est donc vérifier une logique d’usage.
L’utilisateur doit pouvoir avancer sans deviner, sans revenir en arrière inutilement, et sans se demander si l’interface a été pensée pour lui.

Sur mobile, ce point prend une importance particulière dès que l’on travaille avec des aides techniques, des claviers externes, des parcours formulaires ou des interfaces très chargées.

Lecture d’écran : TalkBack, VoiceOver

Les lecteurs d’écran comme TalkBack sur Android ou VoiceOver sur iOS vérifient si l’écran est compréhensible, correctement annoncé, bien positionné dans l’ordre de lecture et réellement exploitable.

C’est souvent à ce moment-là que certains défauts deviennent visibles :

- un bouton annoncé de façon trop vague
- un champ sans libellé explicite
- une icône décorative lue comme un élément important
- un ordre de lecture qui ne correspond pas à la logique visuelle de l’écran
- un message d’erreur affiché, mais jamais réellement annoncé à l’utilisateur

Un écran peut donc être visuellement réussi et pourtant très mal “raconté” par la technologie d’assistance.

Ce type de test change aussi le regard du testeur. Il ne s’agit plus seulement de constater qu’un élément est là, mais de se demander :

- l’utilisateur comprend-il ce qu’il peut faire ?
- Sait-il où il se trouve ?
- Peut-il corriger une erreur sans se perdre ?
- Peut-il terminer son parcours dans de bonnes conditions ?

Activer un lecteur d’écran sur mobile

Une fois activé, le comportement du téléphone change : un simple toucher sélectionne un élément, et un double toucher permet de l’activer. Le balayage horizontal permet de passer d’un élément à l’autre dans l’ordre de lecture.

Pour l’activer sur iPhone (VoiceOver)

- ouvrir Réglages
- aller dans Accessibilité
- sélectionner VoiceOver
- activer l’option

Pour l’activer sur android (talkBack), ça peut varier selon votre modèle :

- ouvrir Paramètres
- aller dans Accessibilité
- sélectionner TalkBack
- activer l’option

RGAA

Les tests avec TalkBack ou VoiceOver permettent d'observer ce que l'utilisateur perçoit réellement. Pour structurer vos vérifications, vous pouvez aussi vous appuyer sur le RGAA, le Référentiel général d'amélioration de l'accessibilité.

En France, il constitue un cadre de référence pour l'accessibilité numérique. La dernière version est le RGAA 4.1.2. Dans la méthodologie d'évaluation, les écrans des applications mobiles sont pris en compte comme des pages à auditer.

Le RGAA ne remplace pas les essais terrain, mais il aide à objectiver ce que l'on observe : ordre de lecture, cohérence de navigation, restitution correcte des composants, compatibilité avec les technologies d'assistance. Il donne donc une grille de lecture utile pour transformer un ressenti de test en constat plus formalisé.

1. 106 critères : images, cadres, couleurs, multimédia, ...
2. obligatoire pour les services de communication au public en ligne, voir le site officiel
3. Déclaration d'accessibilité : résultat d'une évaluation

AVIS D'EXPERTE

Caroline Nazin

🪲 </Quality Engineer Web> & 👩‍💻 </Cheffe de projet fonctionnel> · Luxe & E-commerce international · AI-Powered QE · Automation

Tester avec VoiceOver ou TalkBack est indispensable, mais il est aussi utile de relier les constats terrain à des critères formalisables.
Les WCAG 2.1 et 2.2 contiennent plusieurs critères particulièrement importants sur mobile. Ils permettent de passer d'un ressenti utilisateur à une base plus structurée pour documenter les problèmes.

Quelques critères à garder en tête :

- 1.3.4 Orientation : l'usage ne doit pas être bloqué par une seule orientation d'écran, sauf si cette orientation est essentielle
- 1.4.10 Reflow : le contenu doit rester utilisable sur des largeurs réduites, notamment autour de 320 px
- 2.5.1 Pointer Gestures : les gestes complexes doivent avoir une alternative simple à un seul point de contact
- 2.5.4 Motion Actuation : une action déclenchée par mouvement doit pouvoir être désactivée ou remplacée par une alternative
- 2.5.5 Target Size et 2.5.8 Target Size : les zones tactiles doivent être suffisamment grandes pour être activées sans erreur

Ces critères rejoignent directement des problèmes très mobiles : thumbzone, petits boutons, gestes tactiles complexes, orientation forcée, interface qui casse en zoom ou en police agrandie.

Il faut aussi penser aux réglages système :

- Dynamic Type / taille de police augmentée
- dark mode
- reduced motion
- contraste renforcé
- inversion de couleurs
- lecteur d'écran activé

Ces réglages peuvent casser des interfaces qui semblaient correctes en test nominal.
Avec l'entrée en application de l'European Accessibility Act en 2025 pour plusieurs produits et services numériques, l'accessibilité devient aussi un sujet juridique pour de nombreux acteurs, notamment les services e-commerce concernés.

ACCESSIBILITE EN CI/CD

Les outils automatisés peuvent détecter des problèmes courants, mais ils ne garantissent jamais qu'une application est réellement accessible.

Sur Android, il est possible d'ajouter des contrôles d'accessibilité dans des tests Espresso existants.

Google propose aussi Accessibility Scanner, qui peut signaler des problèmes comme des libellés manquants, des zones tactiles trop petites, des éléments cliquables ou des contrastes insuffisants.

Apple recommande également de tester avec les réglages d'accessibilité et les technologies d'assistance.

Ce que l'automatisation peut aider à détecter

- labels ou descriptions manquantes
- zones tactiles trop petites
- contrastes insuffisants
- éléments cliquables problématiques
- certains problèmes de structure ou d'information accessible

Ce qu'il reste à tester manuellement

- la compréhension réelle du parcours avec TalkBack ou VoiceOver
- la qualité des annonces vocales
- la cohérence du focus dans un parcours complet

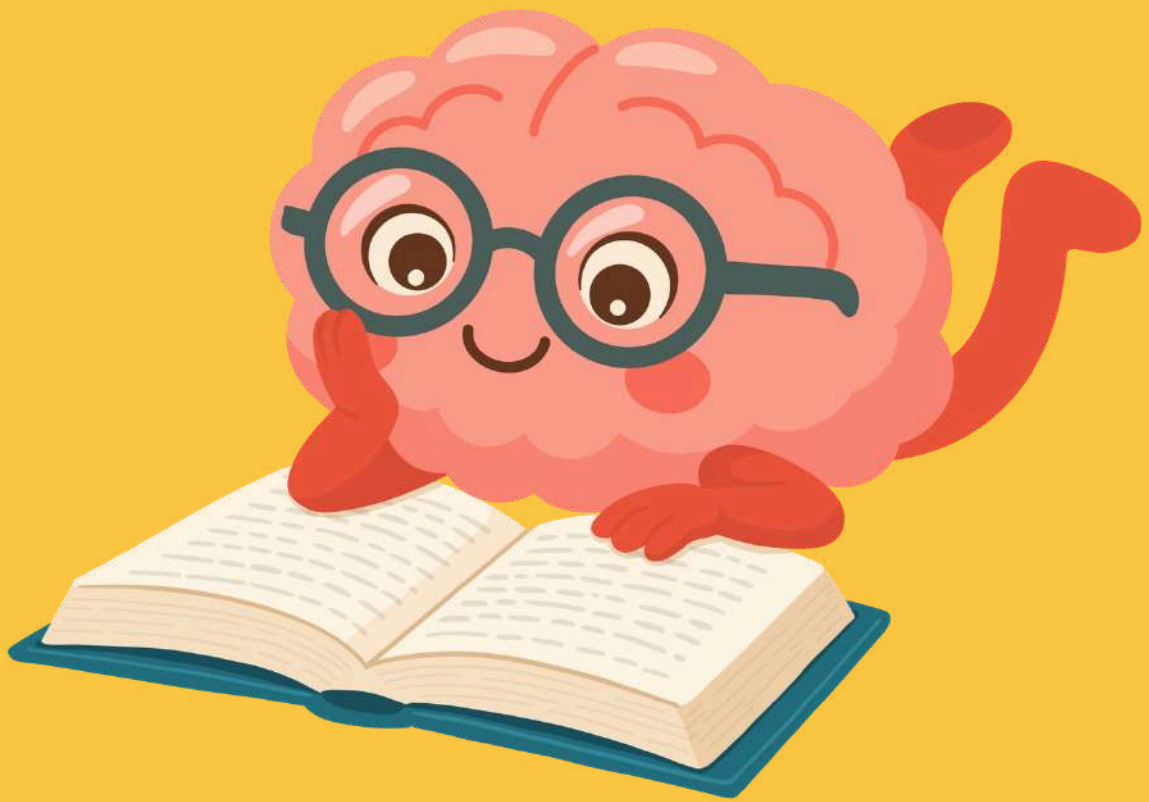
SIGNATURE DU MAG

Automatiser l'accessibilité, c'est utile. Croire que l'automatisation suffit, c'est dangereux.

- les effets de Dynamic Type, reduced motion ou dark mode
- la capacité d'un utilisateur à terminer une action sans aide

L'automatisation doit donc servir de filet de sécurité, pas de certificat de conformité. Elle permet de réduire les oublis, mais elle ne remplace pas l'observation d'un vrai parcours avec les aides techniques activées.

BONNES PRATIQUES



STRUCTURER LES SCENARIOS

Un bon scénario mobile ne décrit pas seulement une action. Il précise le contexte, l'état du device et le risque que l'on cherche à couvrir.

Sur mobile, un scénario trop générique devient vite insuffisant. Dire "l'utilisateur se connecte" ne suffit pas : il faut savoir depuis quel device, avec quelle version d'OS, dans quel état applicatif, avec quelles permissions et dans quelles conditions réseau.

La bonne pratique consiste à écrire des scénarios lisibles par toute l'équipe, mais assez précis pour être exécutables sans interprétation. Le scénario doit raconter un usage réel, pas seulement une suite de clics.

À ajouter dans vos scénarios

• le contexte d'usage : premier lancement, retour après interruption, session expirée, mode hors-ligne

• le device ou la famille de devices visée : petit écran, grand écran, Android constructeur, iPhone ancien

• les préconditions : permissions acceptées/refusées, utilisateur connecté, panier existant, données locales présentes

• le comportement attendu : résultat fonctionnel, message utilisateur, conservation des données, retour à l'état précédent

• le risque couvert : perte de donnée, abandon de parcours, bug de reprise, incohérence cross-device

Exemple trop vague	Version plus utile
Quand l'utilisateur ajoute un produit au panier, le panier est mis à jour.	**Sur Android, après une perte réseau temporaire, l'ajout au panier doit être conservé et synchronisé dès le retour de la connexion.**

Par exemple, voici une structure simple :

- Précondition : utilisateur, données, device, état de l'application
- Action principale : ce que l'utilisateur essaie réellement de faire
- Événements mobiles : clavier, permission, notification, perte réseau, rotation
- Résultat attendu : état métier et état UI
- Critère de sortie : ce qui prouve que le parcours est terminé

Evitez :

- les scénarios trop longs qui testent tout en une seule fois
- les validations implicites impossibles à comprendre
- les dépendances cachées entre tests
- les noms de tests techniques qui ne disent rien du risque couvert
- les scénarios copiés du web sans adaptation mobile

SIGNATURE DU MAG

Un bon scénario doit pouvoir être lu par une personne qui n'a pas écrit le test. S'il est compréhensible, il devient aussi plus facile à maintenir.

GERER LES DONNEES

Beaucoup de tests mobiles échouent moins à cause de l'outil que de l'état de départ : utilisateur déjà connecté, panier non vide, permission refusée, session expirée, cache incohérent.

Dans les missions d'automatisation comme dans les audits qualité, la question des données revient presque toujours. On parle souvent de frameworks, de sélecteurs, de pipelines ou de devices, mais un test automatisé ne peut être fiable que si les données qui le supportent le sont aussi.

Un scénario peut être parfaitement écrit et pourtant devenir instable si les données changent, disparaissent, sont déjà consommées, ne sont pas disponibles dans l'environnement ou ne correspondent plus au cas métier à tester.

Un test flaky n'est donc pas toujours un problème d'outil. Il peut être le symptôme d'un environnement mal maîtrisé ou d'un jeu de données trop fragile.

Les états à maîtriser

- application fraîchement installée ou déjà utilisée
- utilisateur connecté, déconnecté ou session expirée
- permissions acceptées, refusées ou demandées à nouveau
- panier vide, panier existant, wishlist synchronisée
- données locales, cache, stockage, cookies et WebView
- réseau stable, lent, coupé ou changeant

Bonnes pratiques

- préparer explicitement l'état de départ de chaque test
- éviter les tests qui dépendent de l'ordre d'exécution
- utiliser des comptes dédiés ou des jeux de données isolés
- nettoyer ou réinitialiser ce qui doit l'être entre deux tests
- documenter les états non réinitialisables
- prévoir des tests spécifiques pour la reprise d'état et l'expiration de session

SIGNATURE DU MAG

Un test fiable commence avant la première action utilisateur : il commence par un état connu, contrôlé et assumé.

INCLURE LES TESTS D'ACCESSIBILITE DANS LE CYCLE

L'accessibilité ne doit pas arriver comme une vérification finale qui découvre trop tard que l'interface doit être repensée.

Une bonne pratique consiste à intégrer l'accessibilité à chaque étape du cycle : conception, développement, test fonctionnel, automatisation, revue et suivi des anomalies. Plus elle est traitée tôt, plus elle devient une exigence de qualité normale, et non une dette à corriger en urgence.

SIGNATURE DU MAG

Inclure l'accessibilité dans le cycle, ce n'est pas ajouter une contrainte. C'est éviter que certains utilisateurs découvrent les défauts à notre place.

Dans le cycle produit :

01 Conception
vérifier contrastes, tailles de cible, ordre de lecture, alternatives aux gestes complexes

02 Développement
nommer les éléments, gérer les états accessibles, respecter les réglages système

03 Test fonctionnel
inclure TalkBack/ VoiceOver sur les parcours critiques

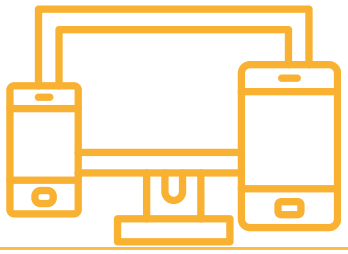
04 Automatisation
ajouter des contrôles simples quand l'outil le permet

05 Recette
documenter les critères WCAG/RGAA concernés et les limites connues

06 Production
suivre les retours utilisateurs et les blocages récurrents

TENDANCES ET PERSPECTIVES

MULTIPLATEFORME



Le mobile n'est plus un canal isolé. Il devient souvent le point d'entrée d'un parcours qui continue ailleurs : desktop, tablette, montre connectée, paiement externe, notification, deep link ou application partenaire.

Pendant longtemps, parler de test mobile revenait surtout à opposer Android et iOS.

Deux systèmes, deux stores, deux logiques de publication, deux familles de devices. Mais cette lecture devient trop étroite. Aujourd'hui, un produit numérique vit rarement sur une seule plateforme. L'utilisateur peut découvrir un service sur son téléphone, reprendre son parcours sur ordinateur, puis finaliser une action depuis une autre interface.

La vraie question devient : l'expérience reste-t-elle cohérente lorsque l'utilisateur change de contexte ? C'est là que le multiplateforme transforme la stratégie de test. Une base de code partagée, un design system commun ou un framework cross-platform ne garantissent pas une expérience identique. Un écran peut être techniquement correct partout, mais ne pas avoir la même valeur selon le support utilisé.

Tester en multiplateforme ne consiste donc pas à dupliquer les mêmes scénarios sur toutes les plateformes. C'est même souvent l'erreur classique : on prend un cas de test web, on le transpose sur mobile, puis on considère que la couverture est suffisante. En réalité, cette approche produit beaucoup de redondance et peu de valeur. Elle oublie ce qui fait la spécificité du mobile : l'usage en mobilité, les interruptions, les permissions, la saisie tactile, les paiements natifs, la caméra, la géolocalisation ou encore les contraintes réseau.

La bonne approche consiste à cartographier les usages. Quels parcours sont réellement mobile-only ? Quels parcours sont desktop-only ? Quels parcours sont cross-device ? Quelles données doivent rester synchronisées ? À quels moments la session, le panier, les favoris, les préférences ou l'authentification peuvent-ils se casser ? Cette cartographie permet d'éviter les tests inutiles et de concentrer l'effort sur les ruptures possibles dans l'expérience utilisateur.

Le multiplateforme oblige aussi le testeur à sortir d'une logique purement écran. Il doit comprendre l'architecture du produit, les points de passage entre plateformes, les règles de synchronisation, les dépendances backend, les deep links, les notifications et les usages métier. Le test devient moins une vérification d'interface qu'une vérification de continuité.

La tendance est donc stratégique. Demain, le test mobile ne sera pas isolé dans une suite dédiée. Il devra s'intégrer dans une vision produit globale, capable de suivre l'utilisateur d'un canal à l'autre. Le vrai enjeu ne sera plus seulement de prouver que chaque plateforme fonctionne séparément, mais que l'expérience reste fiable, fluide et cohérente quand elles se répondent.

IA ET TEST MOBILE

ACCELERATEUR, ASSISTANT… MAIS PAS PILOTE AUTOMATIQUE

L'IA promet de générer, analyser, prioriser et maintenir plus vite. Mais sur mobile, aller plus vite ne suffit pas : il faut encore savoir ce qui mérite vraiment d'être testé.

L'intelligence artificielle arrive dans le test mobile avec beaucoup de promesses : génération de cas de test, aide à la maintenance des scripts, analyse visuelle, résumé de logs, détection d'écarts d'interface, priorisation des anomalies ou création de données de test. Dans un contexte mobile fragmenté, coûteux à couvrir et parfois difficile à automatiser, la promesse est séduisante.

Mais il faut rester lucide : l'IA ne supprime pas la complexité du test mobile. Elle la déplace. Elle peut aider à produire plus vite, à analyser plus largement et à repérer certains signaux faibles. En revanche, elle ne décide pas à la place de l'équipe de ce qui est critique, acceptable ou réellement représentatif de l'usage terrain.

Un premier apport concerne la conception des tests. À partir d'une user story, d'un écran ou d'un parcours, un outil assisté par IA peut proposer des scénarios : cas nominaux, cas limites, erreurs possibles, permissions refusées, réseau instable, changement d'orientation, interruption par notification. C'est utile pour ouvrir la réflexion, surtout en cadrage ou en atelier produit. Mais ces propositions doivent être relues, triées et contextualisées. Produire beaucoup de cas ne veut pas dire produire les bons cas.

Un deuxième apport concerne la maintenance de l'automatisation. Sur mobile, les scripts peuvent être fragiles : changements d'interface, sélecteurs instables, animations, différences entre devices, comportements spécifiques Android ou iOS. Les approches basées sur l'analyse visuelle ou la reconnaissance d'éléments à l'écran peuvent aider, notamment lorsque l'interface évolue souvent. Mais là encore, l'outil ne remplace pas le diagnostic : un test qui échoue peut révéler un bug, une donnée incohérente, un environnement instable ou une limite de l'automatisation.

Un troisième apport concerne l'analyse des résultats. Les campagnes mobiles génèrent beaucoup d'artefacts : vidéos, captures, logs, traces réseau, crash reports, résultats de device farms. L'IA peut aider à regrouper des erreurs similaires, résumer les échecs ou orienter l'investigation. Cette aide peut faire gagner du temps, à condition de ne pas confondre synthèse automatique et compréhension réelle du problème.

Enfin, il faudra aussi tester les applications mobiles qui intègrent elles-mêmes de l'IA. Là, les enjeux changent : résultats non déterministes, dépendance aux données, biais possibles, qualité des réponses, robustesse face aux entrées inattendues. Le testeur mobile devra donc comprendre à la fois le contexte mobile et les limites des fonctionnalités intelligentes qu'il teste.

LE FUTUR DU METIER DE TESTEUR MOBILE

MOINS EXECUTANT, PLUS STRATEGE TERRAIN

Le métier ne disparaît pas : il se déplace. Le testeur mobile de demain ne cliquera pas seulement sur plus de téléphones. Il devra mieux lire les usages, les risques et les signaux du terrain. Plus les outils progressent, plus la valeur humaine se déplace vers ce qui est difficile à automatiser : observer, questionner, prioriser, arbitrer et raconter le risque.

Le métier de testeur mobile est en train de changer. Non pas parce que les tests disparaissent, mais parce que le mobile devient plus complexe, plus transverse et plus stratégique. Pendant longtemps, le testeur mobile a pu être perçu comme la personne qui installe une build, exécute des scénarios sur quelques téléphones, vérifie Android et iOS, puis remonte les anomalies.

Cette vision est déjà dépassée. Le testeur mobile doit aujourd'hui comprendre bien plus que l'écran qu'il manipule. Il doit tenir compte des usages réels, des devices, des versions d'OS, des permissions, de la performance perçue, de l'accessibilité, des environnements de test, des fermes d'appareils, des analytics, des crash reports et des retours utilisateurs.

Son rôle devient plus proche d'un analyste qualité orienté terrain. Il ne s'agit plus seulement de vérifier que l'application respecte une exigence, mais d'anticiper ce qui peut se passer dans la vraie vie : une notification qui masque un bouton, un appel entrant pendant une saisie sensible, une perte de réseau au moment du paiement, une reprise d'état mal gérée, une session rompue entre mobile et desktop, un comportement différent selon le device.

Le testeur mobile ne sera pas remplacé par une device farm. Ces plateformes sont précieuses pour élargir la couverture, exécuter en parallèle et tester sur de nombreux appareils. Mais elles ne reproduisent pas à elles seules toutes les conditions réelles : qualité réseau locale, manipulation physique prolongée, contexte culturel, habitudes d'usage, environnement lumineux, fatigue, urgence ou attention partielle.

Il ne sera pas non plus remplacé par l'automatisation. L'automatisation reste indispensable pour sécuriser les parcours critiques, accélérer les régressions et donner du feedback régulier. Mais elle ne capte pas tout : inconfort tactile, confusion visuelle, perte de confiance, mauvaise lisibilité, parcours pénible ou comportement acceptable techniquement mais irritant pour l'utilisateur.

Le futur du métier sera aussi pédagogique. Le testeur devra expliquer pourquoi un parcours fonctionnellement correct peut rester mauvais en conditions réelles. Il devra aider les équipes produit, design et développement à comprendre les risques mobiles, sans réduire la qualité à une checklist technique.

JEU CONCOURS

DEFI DU TESTEUR

Tentez de remporter un exemplaire du jeu “BUGS END” créé par les sociétés K-Lagan et Mr Suricate, spécialisées dans le test informatique.

5 dessins sont cachés dans le magazine (en plus de celle-ci), à vous de les trouver et d’indiquer les numéros des pages où ils se trouvent

Image à trouver !

Comment participer ?

Envoyez votre réponse par email avant le 1er juillet 2026 à 23h59 à :
feedback@lemagdutesteur.fr.
Un seul gagnant.

Lot à gagner :

1 jeu **professionnel** Bugs End à gagner (valeur 79€)

- Le règlement complet est consultable ici ou à la demande par email:

- Concours gratuit – Une seule participation par personne – Données supprimées à l’issue du jeu

- Les gagnants seront contactés par mail

TEASING

Partagez-nous vos retours !

Ce magazine a été conçu pour vous accompagner dans l'amélioration de vos tests. Vos impressions, idées et questions sont précieuses pour que Le Mag du Testeur continue de grandir et de s'adapter à vos besoins.

N'hésitez pas à m'écrire à **feedback@lemagdutesteur.fr** pour échanger autour de vos expériences et de vos suggestions !

Un avant-goût du prochain numéro...
IA & Test logiciel : alliée, menace ou illusion ?

Merci pour votre fidélité et à très bientôt dans les pages de Le Mag du Testeur !