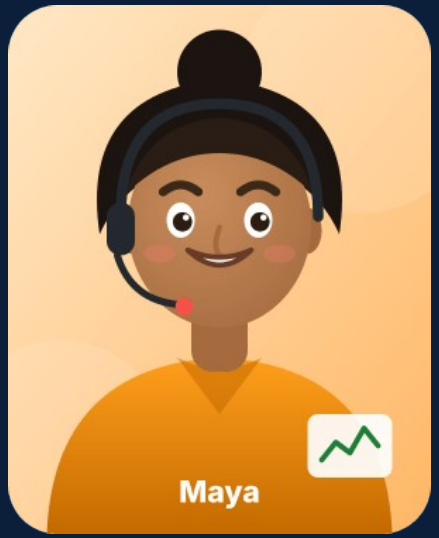
HEF-110 · MISSION 06 / 10

Tests E2E



Un test de bout en bout se comporte comme une vraie personne devant l'application. Puissant, lent, fragile : il faut en faire peu, mais les faire bien.

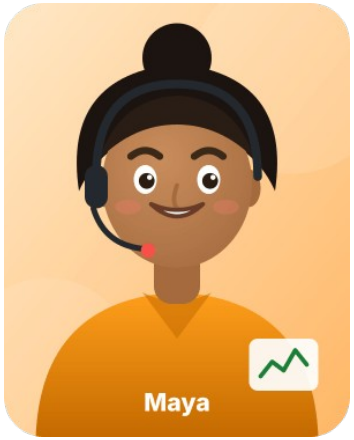


Badge : Utilisateur automatique

100 min · intermédiaire

L'histoire

Toutes les couches de HashBox passent leurs tests. Pourtant, lundi, personne ne peut s'inscrire à un cours : le bouton « S'inscrire » envoie le formulaire vers une mauvaise adresse après une refonte des pages. Aucun test unitaire ni d'API n'avait « cliqué » sur ce bouton.



> *« Il manque un test qui fasse exactement ce que fait Léo, dit Maya : se connecter, voir les cours, s'inscrire, vérifier sa liste. Un seul parcours, du début à la fin. C'est le rôle des tests E2E. »*

Tu apprends à automatiser un parcours utilisateur complet, à le rendre stable (attentes explicites, sélecteurs fiables, page objects) et à choisir les rares parcours qui méritent ce coût.

Ta mission

À la fin, je serai capable de…

1. expliquer le rôle et le coût des tests E2E dans la pyramide
2. automatiser un parcours utilisateur (connexion, action, vérification) sans navigateur externe
3. remplacer les `sleep` par des attentes explicites
4. structurer les tests avec le modèle *page object*
5. choisir quels parcours tester en E2E et lesquels laisser aux niveaux inférieurs

Défi d'échauffement

Défi d'échauffement

Tu ouvres un restaurant. Avant l'ouverture, tu fais jouer le parcours d'un client du début à la fin : entrer, commander, être servi, payer. Quelles défaillances ce « test à blanc » trouve-t-il que la vérification séparée de chaque poste (cuisine, caisse, salle) n'aurait pas vues ? Pourquoi ne peut-on pas le faire jouer 500 fois par jour ?

Le concept

Ce qu'est un test E2E

Un test **de bout en bout** (*end-to-end*) exerce l'application **complète** — interface, serveur, base — comme un utilisateur réel. Il vérifie que **le parcours fonctionne**, pas chaque détail.

Avantage	Inconvénient
détecte les ruptures entre toutes les couches	lent (secondes à minutes)
teste ce que voit réellement l'utilisateur	fragile (interface qui change, timing)
confiance forte sur les parcours critiques	diagnostic large : « quelque part, ça casse »

D'où la règle de la pyramide : **peu** de tests E2E, sur les **parcours critiques** (connexion, achat, inscription, paiement), pas sur chaque règle de gestion.

Le concept

Ce qu'on automatise

- **Navigateur piloté** : Playwright, Selenium, Cypress. Ils démarrent un vrai navigateur, cliquent, remplissent, lisent.
- **Client HTTP qui simule le navigateur** : plus léger, suffisant pour des pages simples (c'est ce qu'on fait ici avec la bibliothèque standard).

```
# Exemple Playwright (à lancer sur ta machine après installation)
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    nav = p.chromium.launch()
    page = nav.new_page()
    page.goto("http://localhost:8000/login")
    page.get_by_label("Nom").fill("ada")
    page.get_by_role("button", name="Entrer").click()
```

Le concept

Ce qu'on automatise (suite)

```
    page.get_by_test_id("cours-HEF-010").get_by_role("button", name="S'inscrire").
click()
    assert "HEF-010" in page.locator("#mes-cours").inner_text()
    nav.close()
```

La stabilité : la grande bataille

Cause d'instabilité	Remède
`sleep(2)` « au cas où »	**attente explicite** d'une condition, avec délai maximal
sélecteurs CSS dépendant de la mise en page (`div > div:nth-child(3)`)	sélecteurs **stables** : rôles, libellés, attributs `data-testid`
données partagées entre tests	chaque test crée ses données
dépendance à un service externe	doubles en E2E, ou environnement dédié
ordre des tests	parcours indépendants

Le concept

Attente explicite

```
attendre(condition, délai_max) :
    tant que (temps < délai_max) :
        si condition() est vraie : retourner
        pause courte
    lever TimeoutError
```

Un `sleep(5)` fait **toujours** perdre 5 s et échoue quand la machine est lente. Une attente explicite rend la main dès que c'est prêt.

Le modèle *page object*

On regroupe ce qu'on sait d'une **page** (adresses, champs, actions) dans une classe. Les tests racontent le parcours avec des verbes métier ; si l'interface change, **un seul endroit** à corriger.

Le concept

Le modèle *page object* (suite)

```
page_login.se_connecter("ada")
page_cours.s_inscrire("HEF-010")
assert "HEF-010" in page_compte.mes_cours()
```

Smoke tests

Un **test de fumée** (*smoke test*) est un E2E minimal exécuté après chaque déploiement : « l'application démarre, la page d'accueil répond, on peut se connecter ». S'il échoue, on annule le déploiement.

Quand diagnostiquer un échec E2E

Garde des **preuves** : capture d'écran, journal réseau, HTML de la page au moment de l'échec, trace. Sans preuves, un test E2E qui échoue est une énigme.

Le concept

Quand diagnostiquer un échec E2E (suite)

🧠 ***Astuce*** *: pour chaque comportement, demande-toi « quel est le test* ***le plus bas*** *de la pyramide qui l'attraperait ? ». N'élève le test que si les couches doivent vraiment travailler ensemble.*

Exemple pas à pas

Problème : tester le parcours *connexion → inscription → vérification* d'une petite application web avec un « navigateur » minimal (cookies, redirections, formulaires), des *page objects*, et une attente explicite.

```
import html.parser, http.cookiejar, io, secrets, threading, time, unittest, urllib
   .parse, urllib.request
from http.server import BaseHTTPRequestHandler, HTTPServer

# ================ L'application (côté serveur) ================
COURS = {"HEF-010": "Algorithmes", "HEF-020": "Structures de données"}
SESSIONS, INSCRIPTIONS = {}, {}
PRET_A = {"t": 0.0}

class App(BaseHTTPRequestHandler):
    def _page(self, corps, statut=200, entetes=None):
        donnees = f"<html><body>{corps}</body></html>".encode()
        self.send_response(statut)
        self.send_header("Content-Type", "text/html; charset=utf-8")
```

Exemple pas à pas (suite)

```
        self.send_header("Content-Length", str(len(donnees)))
        for k, v in (entetes or {}).items():
            self.send_header(k, v)
        self.end_headers()
        self.wfile.write(donnees)

    def _rediriger(self, vers, entetes=None):
        self._page("", 303, dict({"Location": vers}, **(entetes or {})))

    def _membre(self):
        for morceau in self.headers.get("Cookie", "").split(";"):
            nom, _, valeur = morceau.strip().partition("=")
            if nom == "session":
                return SESSIONS.get(valeur)

    def do_GET(self):
        if self.path == "/login":
            return self._page('<form method="post" action="/login"><label>Nom <inp
ut name="nom"></label><button>Entrer</button></form>')
```

Exemple pas à pas

Exemple pas à pas (suite)

```
        membre = self._membre()
        if membre is None:
            return self._rediriger("/login")
        if self.path == "/cours":
            items = "".join(
                f'<li data-testid="cours-{code}">{titre}<form method="post" action
    ="/inscrire">'
                f'<input type="hidden" name="cours" value="{code}"><button>S\'insc
    rire</button></form></li>' for code, titre in COURS.items())
            return self._page(f"<h1>Bonjour {membre}</h1><ul>{items}</ul>")
        if self.path == "/mon-compte":
            mes = "".join(f"<li>{c}</li>" for c in sorted(INSCRIPTIONS.get(membre,
     ())))
            return self._page(f'<ul id="mes-cours">{mes}</ul>')
        if self.path == "/rapport":
            return self._page("<p id='etat'>" + ("prêt" if time.monotonic() >= PRE
    T_A["t"] else "en cours") + "</p>")
        self._page("introuvable", 404)
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    def do_POST(self):
        corps = self.rfile.read(int(self.headers.get("Content-Length", 0))).decode
    ()
        champs = {k: v[0] for k, v in urllib.parse.parse_qs(corps).items()}
        if self.path == "/login":
            jeton = secrets.token_hex(8)
            SESSIONS[jeton] = champs.get("nom", "anonyme")
            return self._rediriger("/cours", {"Set-Cookie": f"session={jeton}; Htt
    pOnly"})
        membre = self._membre()
        if membre is None:
            return self._rediriger("/login")
        if self.path == "/inscrire":
            INSCRIPTIONS.setdefault(membre, set()).add(champs["cours"])
            return self._rediriger("/mon-compte")
        if self.path == "/rapport/lancer":
            PRET_A["t"] = time.monotonic() + 0.3
            return self._rediriger("/rapport")
        self._page("introuvable", 404)
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    def log_message(self, *a):
        pass

# ================ Le « navigateur » ================
class Analyseur(html.parser.HTMLParser):
    def __init__(self):
        super().__init__()
        self.formulaires, self.ids, self.texte, self._testid, self._forme, self._i
   d = [], {}, [], None, None, None
    def handle_starttag(self, balise, attrs):
        a = dict(attrs)
        if "data-testid" in a:
            self._testid = a["data-testid"]
        if "id" in a:
            self._id = a["id"]; self.ids[self._id] = ""
        if balise == "form":
            self._forme = {"action": a["action"], "champs": {}, "testid": self._te
   stid}
```

Exemple pas à pas

Exemple pas à pas (suite)

```
            self.formulaires.append(self._forme)
        if balise == "input" and self._forme is not None and "name" in a:
            self._forme["champs"][a["name"]] = a.get("value", "")
    def handle_endtag(self, balise):
        if balise == "form":
            self._forme = None
        if balise == "li":
            self._testid = None
        if balise in ("ul", "p"):
            self._id = None
    def handle_data(self, donnees):
        self.texte.append(donnees)
        if self._id:
            self.ids[self._id] += donnees

class Navigateur:
    def __init__(self, base):
        self.base = base
        self.opener = urllib.request.build_opener(urllib.request.HTTPCookieProcess
```

Exemple pas à pas (suite)

```
   or(http.cookiejar.CookieJar()))
        self.page = None
    def _charger(self, reponse):
        self.url = reponse.geturl()
        self.page = Analyseur()
        self.page.feed(reponse.read().decode())
    def aller(self, chemin):
        self._charger(self.opener.open(self.base + chemin, timeout=5))
    def soumettre(self, action, champs):
        self._charger(self.opener.open(self.base + action, urllib.parse.urlencode(
   champs).encode(), timeout=5))
    def texte(self):
        return " ".join(self.page.texte)

def attendre(condition, delai=3.0, pas=0.02):
    fin = time.monotonic() + delai
    while time.monotonic() < fin:
        if condition():
            return True
```

Exemple pas à pas

Exemple pas à pas (suite)

```
        time.sleep(pas)
    raise TimeoutError("condition non remplie à temps")

# ================= Page objects =================
class PageLogin:
    def __init__(self, nav): self.nav = nav
    def se_connecter(self, nom):
        self.nav.aller("/login")
        self.nav.soumettre("/login", {"nom": nom})
        return PageCours(self.nav)

class PageCours:
    def __init__(self, nav): self.nav = nav
    def s_inscrire(self, code):
        forme = next(f for f in self.nav.page.formulaires if f["testid"] == f"cour
   s-{code}")      # sélecteur stable
        self.nav.soumettre(forme["action"], forme["champs"])
        return PageCompte(self.nav)
```

Exemple pas à pas (suite)

```
class PageCompte:
    def __init__(self, nav): self.nav = nav
    def mes_cours(self):
        return self.nav.page.ids.get("mes-cours", "")

# ================= Les tests E2E =================
class TestParcours(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.serveur = HTTPServer(("127.0.0.1", 0), App)
        cls.base = f"http://127.0.0.1:{cls.serveur.server_address[1]}"
        threading.Thread(target=cls.serveur.serve_forever, daemon=True).start()
    @classmethod
    def tearDownClass(cls):
        cls.serveur.shutdown(); cls.serveur.server_close()
    def setUp(self):
        self.nav = Navigateur(self.base)                          # navigateur NEU
   F = cookies neufs
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    def test_parcours_connexion_inscription(self):
        cours = PageLogin(self.nav).se_connecter("ada")
        self.assertIn("Bonjour ada", self.nav.texte())
        compte = cours.s_inscrire("HEF-010")
        self.assertEqual(compte.mes_cours(), "HEF-010")

    def test_deux_inscriptions_apparaissent_dans_le_compte(self):
        cours = PageLogin(self.nav).se_connecter("bob")
        cours.s_inscrire("HEF-010")
        self.nav.aller("/cours")
        compte = PageCours(self.nav).s_inscrire("HEF-020")
        self.assertEqual(compte.mes_cours(), "HEF-010HEF-020")

    def test_page_protegee_sans_connexion_renvoie_au_login(self):
        self.nav.aller("/cours")
        self.assertTrue(self.nav.url.endswith("/login"))

    def test_attente_explicite_plutot_que_sleep(self):
        PageLogin(self.nav).se_connecter("cleo")
```

Exemple pas à pas (suite)

```
        self.nav.soumettre("/rapport/lancer", {})
        self.assertIn("en cours", self.nav.texte())
        debut = time.monotonic()
        def pret():
            self.nav.aller("/rapport")
            return "prêt" in self.nav.texte()
        attendre(pret)
        duree = time.monotonic() - debut
        self.assertLess(duree, 1.0)                              # rend la main d
   ès que c'est prêt, sans attendre un sleep(5)

resultat = unittest.TextTestRunner(stream=io.StringIO()).run(unittest.defaultTestL
   oader.loadTestsFromTestCase(TestParcours))
print("parcours E2E :", resultat.testsRun, "| réussite :", resultat.wasSuccessful(
   ))
assert resultat.wasSuccessful(), resultat.failures + resultat.errors
```

Lecture :

Exemple pas à pas (suite)

- Le test E2E parcourt **toutes les couches** : serveur HTTP, session par cookie, redirections, formulaires, pages.
- Le *page object* `PageCours.s_inscrire("HEF-010")` utilise un sélecteur stable (`data-testid`), pas la position dans la page : si le design change, le test survit.
- `attendre()` interroge la condition toutes les 20 ms : le test rend la main dès que le rapport est prêt (~0,3 s), alors qu'un `sleep(5)` aurait gaspillé 5 s à chaque exécution et échoué sur une machine lente.
- Chaque test commence avec un **navigateur neuf** (cookies vides) : indépendance.

À toi, avec aide

Ajoute au scénario un test E2E **négatif** : une personne non connectée essaie d'envoyer directement le formulaire d'inscription (`POST /inscrire` avec `cours=HEF-010`) sans passer par la page de connexion. Vérifie qu'elle est redirigée vers `/login` et qu'**aucune inscription** n'a été créée.
(Utilise l'application, le navigateur et les classes de l'exemple.)

À toi, avec aide · réponse

Indices progressifs

Indice 1

Crée un `Navigateur` neuf (sans cookie) et appelle `soumettre('/inscrire', {'cours': 'HEF-010'})`.

Indice 2

`self.nav.url` finit par `/login` après la redirection.

Indice 3

Vérifie `INSCRIPTIONS` : aucune clé pour un membre anonyme.

À toi, avec aide · réponse

Réponse

```
# (à placer dans la classe TestParcours de l'exemple)
def test_inscription_directe_sans_connexion_refusee(self):
    avant = {m: set(c) for m, c in INSCRIPTIONS.items()}
    self.nav.soumettre("/inscrire", {"cours": "HEF-010"})        # navigateur neuf
    : aucun cookie de session
    self.assertTrue(self.nav.url.endswith("/login"))
    self.assertEqual(INSCRIPTIONS, avant)                         # rien n'a été c
   réé
```

Ce test vérifie un **comportement de sécurité** (contrôle d'accès côté serveur) qu'aucune interface n'exerce « naturellement » : l'interface n'offre jamais ce chemin, mais un attaquant le peut (cf. HEF-100).

À toi, en solo

Rends un E2E fragile robuste. Voici trois mauvaises pratiques dans un script de test : un `time.sleep(2)` après chaque clic, un sélecteur par position (« le 3e bouton »), et un test qui dépend d'un compte créé par un autre test. Pour chacune :

1. écris un petit exemple minimal qui **échoue de façon intermittente** (simule la lenteur avec un délai aléatoire à graine fixe) ;
2. corrige-le avec une attente explicite, un sélecteur stable (`data-testid`), et une fabrique de compte par test ;
3. mesure le temps gagné et le taux d'échec avant/après sur 200 exécutions simulées.

À toi, en solo · réponse

Pistes de correction

```
import random, time

def simuler(attente, essais=200, graine=11):
    # la page « devient prête » après un délai aléatoire (ms simulées), parfois le
   nt
    rng = random.Random(graine)
    echecs, temps_total = 0, 0.0
    for _ in range(essais):
        pret_apres = rng.choice([0.1, 0.2, 0.3, 0.5, 2.5])           # secondes si
   mulées, parfois très lent
        reussi, ecoule = attente(pret_apres)
        echecs += (not reussi)
        temps_total += ecoule
    return echecs, temps_total

def avec_sleep(pret_apres, duree=2.0):
    return pret_apres <= duree, duree                                 # attend tou
   jours 2 s ; échoue si c'est plus lent
```

À toi, en solo · réponse

Pistes de correction (suite)

```
def avec_attente_explicite(pret_apres, delai_max=10.0):
    return pret_apres <= delai_max, pret_apres                          # s'arrête d
   ès que c'est prêt, tolère jusqu'à 10 s

e1, t1 = simuler(avec_sleep)
e2, t2 = simuler(avec_attente_explicite)
print(f"sleep(2)  : {e1} échecs sur 200, temps cumulé {t1:.0f} s")
print(f"attente   : {e2} échecs sur 200, temps cumulé {t2:.0f} s")
assert e1 > 0 and e2 == 0 and t2 < t1
```

L'attente explicite est à la fois **plus fiable** (tolère les cas lents) et **plus rapide** (ne perd pas de temps quand tout va bien). Pour les deux autres défauts : les `data-testid` survivent aux refontes de mise en page ; une fabrique `creer_compte()` par test supprime la dépendance d'ordre.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Trop de tests E2E	Suite lente, instable, coûteuse	Peu de parcours critiques ; le reste plus bas dans la pyramide
`sleep` fixes	Lents et pourtant intermittents	Attentes explicites sur une condition
Sélecteurs liés à la mise en page	Cassent à chaque refonte	Rôles, libellés, `data-testid`
Tests qui s'enchaînent	Un échec en entraîne dix	Parcours indépendants, données fabriquées

Mini-boss

Mini-boss

Le parcours critique de HashBox. Étends l'application de l'exemple avec : inscription impossible quand le cours est complet (message « complet »), déconnexion (`POST /logout` qui détruit la session), et un second utilisateur.

1. écris les *page objects* nécessaires ;
2. écris 5 parcours E2E : (a) nominal, (b) cours complet, (c) déconnexion puis accès refusé, (d) deux utilisateurs indépendants, (e) rapport asynchrone avec attente explicite ;
3. écris un **smoke test** de 3 assertions exécutable après déploiement ;
4. fais sauvegarder le HTML de la page quand un test échoue (preuve de diagnostic) ;
5. justifie dans un tableau pourquoi chacun de ces 5 comportements mérite (ou non) un E2E plutôt qu'un test plus bas.

Mini-boss · réponse

Pistes de correction

Un bon travail : classes `PageLogin`, `PageCours`, `PageCompte`, un `setUp` qui crée un `Navigateur` neuf, un `tearDown` ou un `addCleanup` qui écrit `nav.page` dans un fichier si le test a échoué (recherche de `self._outcome`), un smoke test (page de login répond, connexion possible, liste de cours affichée). Le tableau de justification attendu : nominal, déconnexion et accès refusé méritent l'E2E (ils traversent session + serveur + pages) ; la règle « cours complet » se teste plutôt au niveau service/unitaire, avec un seul E2E pour vérifier que le **message** atteint bien l'utilisateur.

Quiz éclair

Quiz éclair

1. Un test E2E est approprié pour :

A. chaque règle de calcul isolée
B. un parcours utilisateur critique complet
C. la syntaxe d'une fonction

2. Pour attendre qu'une page soit prête, on préfère :

A. un `sleep` fixe de 10 secondes
B. relancer jusqu'à ce que ça passe
C. une attente explicite d'une condition avec délai maximal

3. Le modèle *page object* sert à :

A. centraliser ce qu'on sait d'une page pour que les tests restent lisibles et robustes

Quiz éclair

Quiz éclair (suite)

B. accélérer le serveur
C. chiffrer les cookies

4. 🔁 *Révision mission 05* : Un changement compatible d'une API est :

A. le renommage d'un champ de réponse
B. l'ajout d'un champ optionnel dans la réponse
C. le changement du type d'un champ

5. 🔁 *Révision mission 05* : Un test de contrat côté consommateur :

A. remplace tous les tests
B. teste la vitesse du réseau
C. déclare ce dont le client dépend et le fait vérifier par le fournisseur

Quiz éclair · réponse

Réponses

1. **B**.
2. **C**.
3. **A**.
4. **B**.
5. **C**.

Je réfléchis

1. Quel est le parcours dont la panne coûterait le plus cher à ton activité ?
2. Quel test lent de ton projet pourrait être descendu d'un niveau dans la pyramide ?

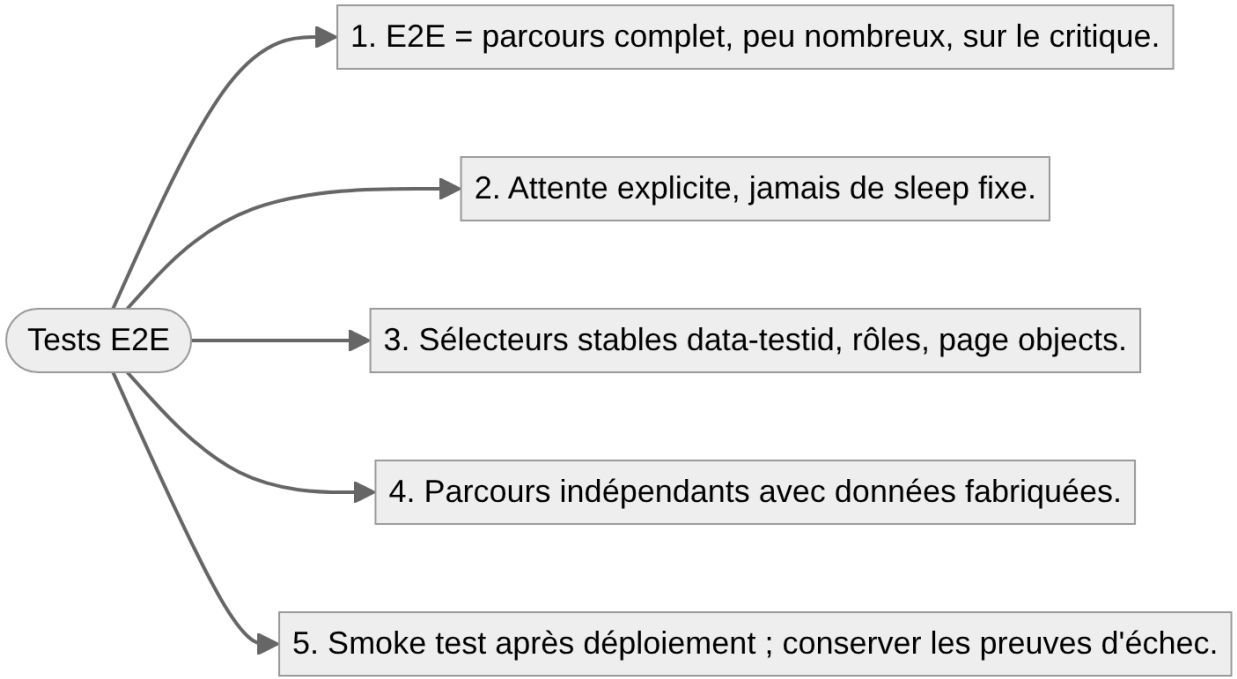
À retenir

À retenir

1. E2E = parcours complet, peu nombreux, sur le critique.
2. Attente explicite, jamais de `sleep` fixe.
3. Sélecteurs stables (`data-testid`, rôles), page objects.
4. Parcours indépendants avec données fabriquées.
5. Smoke test après déploiement ; conserver les preuves d'échec.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Utilisateur automatique

Prochaine mission : Doubles de test

