Clarté technique

PILIER #2

5 étapes pour évaluer et renforcer la solidité de ta base technique
Architecture, code, outils : pose les fondations qui ne lâchent jamais

Un guide Arkonium – Clarté, Structuration, Scalabilité

POURQUOI LA CLARTÉ TECHNIQUE CHANGE TOUT

La technique n'est pas un détail, c'est la colonne vertébrale de ton projet.
Une base maîtrisée évite l'effondrement, même quand tout accélère.

3 bénéfices d'une technique maîtrisée

Tu avances vite, sans casser : tu peux livrer, itérer, corriger sans tout remettre en cause.

Tu dors tranquille : moins de bugs cachés, moins de stress en prod, moins de risques imprévus.

Tu gagnes en crédibilité : les partenaires, clients, et devs voient que ton projet est construit pour durer.

On ne maîtrise pas un projet en empilant des features,
mais en bâtissant sur des fondations saines.

"La technique, c'est ce qui te sauve quand tout vacille.
Prends-en soin avant d'en avoir besoin."

DOCUMENTER CLAIREMENT : ÉVITER LES BLOCAGES

Une documentation accessible, c'est la boussole de ton équipe.
Elle évite les blocages et accélère l'intégration de nouveaux membres.

Prends 5 minutes pour partager ta documentation à un membre de l'équipe (ou un ami développeur).
Demande-lui de trouver une info clé sans ton aide.
Si c'est rapide et sans blocage, ta doc fait le job. Sinon, liste ce qui coince.

- [ ] Elle est accessible à tous
- [ ] Elle est à jour
- [ ] Elle est lue/utilisée

Qu'est-ce qui a bloqué lors de ton test ?

_____

_____

_____

ISOLER SES ENVIRONNEMENTS : DÉPLOYER SANS RISQUE

Des environnements bien séparés (dev, test, prod), c'est la garantie d'éviter les catastrophes.
Tu dois pouvoir tester sans risque, et déployer sans stress.

J'ai vu des équipes perdre des jours parce qu'une base de production a été écrasée par erreur en dev.
En séparant bien chaque environnement, ce genre de cauchemar ne t'arrivera pas.

Demande à un collègue ou à toi-même de lancer le projet sur un nouvel environnement (ou machine).
Chronomètre combien de temps il faut pour que tout fonctionne.

- [ ] Les environnements sont séparés (au moins dev/prod)
- [ ] Le setup est documenté et reproductible
- [ ] Les données sensibles ou de prod ne circulent jamais sur dev/test

Qu'as-tu identifié comme friction ou risque lors de ce test ?

_____

_____

_____

RENDRE LE CODE LISIBLE : FACILITER LA REPRISE ET L'ÉVOLUTION

Un code propre, c'est un code qu'on comprend et qu'on fait évoluer sans peur.
Tu dois pouvoir relire ou transmettre ton projet sans prise de tête.

Prends une partie clé de ton code (ou d'un collègue) et relis-la comme si tu découvrais le projet.
Es-tu capable d'expliquer ce que ça fait, sans avoir besoin de demander ?
Note les zones floues ou à clarifier.

- ☐ Les fonctions et variables ont des noms explicites
- ☐ Les règles de style (linter, formatter) sont appliquées
- ☐ Il existe des commentaires/clés de lecture pour le métier ou les zones complexes

Quels passages t'ont semblé les moins lisibles ?

_____

_____

_____

AUTOMATISER LES TESTS : PRÉVENIR LES BUGS CRITIQUES

Les tests automatisés protègent ton projet des bugs qui reviennent toujours au pire moment.
Même quelques tests bien placés valent mieux que rien.

“Un code non testé, c’est comme un parachute non vérifié : tu pries pour que ça tienne.”

Identifie une fonctionnalité clé du projet et écris (ou fais écrire) un test automatique dessus.
Lance-le avant chaque mise à jour.
Demande-toi ce qui t’aurait échappé sans ce test.

- [ ] Un minimum de tests automatiques existe (même sur une seule feature critique)
- [ ] Les tests sont lancés à chaque changement majeur
- [ ] Les bugs corrigés donnent lieu à un nouveau test pour éviter les régressions

Quelle fonctionnalité aurait mérité d’être testée en premier ?

_____

_____

_____

GÉRER SES DÉPENDANCES : RESTER MAÎTRE DE TA STACK

Beaucoup de projets cassent à cause d'une mise à jour imprévue ou d'une dépendance mal comprise.
Les dépendances techniques doivent être visibles, maîtrisées et auditées.

Liste toutes les dépendances critiques (librairies, SDK, services externes). Identifie celles à surveiller de près, et établis un process de mise à jour sécurisé.

- [ ] J'ai dressé une liste claire des dépendances majeures du projet
- [ ] Un suivi des mises à jour est en place (notes de version, changelog, alertes)
- [ ] Je sais lesquelles peuvent tout casser si elles évoluent

Quelle dépendance dois-je absolument surveiller en priorité ?

_____

_____

_____

AUTO-ÉVALUATION : OÙ EN ES-TU SUR LA CLARTÉ TECHNIQUE ?

Réponds honnêtement à ces 5 questions.
Coche une case par question :

La documentation est accessible et comprise par tous les membres de l'équipe.

☐ Oui ☐ Partiellement ☐ Non

Chaque environnement (dev, test, prod) est séparé, isolé et simple à mettre en place.

☐ Oui ☐ Partiellement ☐ Non

Le code est lisible, structuré, et respecte des conventions claires.

☐ Oui ☐ Partiellement ☐ Non

Des tests automatisés existent et sont lancés à chaque évolution importante.

☐ Oui ☐ Partiellement ☐ Non

Un nouveau développeur peut installer le projet et contribuer en moins d'une journée.

☐ Oui ☐ Partiellement ☐ Non

4-5 "Oui" : continue, tu as de la clarté produit !
<4 "Oui" : reprends une étape, ou demande de l'aide si besoin.

Chaque "Non" est une opportunité d'amélioration :
cible tes zones d'ombre, repasse sur les étapes concernées,
ou demande un diagnostic externe.

CHECKLIST : LES ÉTAPES CLÉS À VALIDER

Coche chaque étape lorsque tu l'as validée.
Reviens sur cette liste à chaque nouvelle version, ou dès que tu ressens un doute sur la clarté technique de ton projet.

- [ ] Documentation accessible, à jour et comprise
- [ ] Environnements de travail séparés, reproductibles et sécurisés
- [ ] Code lisible, structuré, respectant les conventions définies
- [ ] Premiers tests automatisés en place et lancés à chaque évolution
- [ ] Guide d'installation ou onboarding opérationnel pour un nouveau dev
- [ ] Liste des dépendances et configurations partagée
- [ ] Script(s) d'installation et/ou de déploiement testés
- [ ] Points de friction ou dettes techniques identifiés et suivis
- [ ] Process d'amélioration continue mis en place (1h/mois pour réviser la technique)

Seule l'action fait progresser.
Un petit pas chaque semaine = un projet bien plus solide dans 6 mois.

À ÉVITER ABSOLUMENT !

Même les meilleurs projets se plantent sur ces points.
Anticipe, et tu éviteras beaucoup de temps (et d'argent) perdus !

Repousser la documentation ou l'automatisation "à plus tard"

Résultat : Tu oublies, et tu te retrouves bloqué au pire moment, sans repère ni filet de sécurité.

Penser "ça marche sur ma machine, donc c'est bon".

Résultat : Le jour où il faut déployer ou transmettre, tout casse et la confiance s'effondre.

Être le seul à comprendre l'architecture ou le code.

Résultat : Personne ne peut reprendre, corriger, ni faire évoluer le projet sans toi. Tu deviens le goulot d'étranglement.

Si tu évites ces pièges, tu fais déjà partie des 20 % qui construisent sur du solide.

Relis ce livret avant chaque refonte technique, ou quand tu sens que la dette s'accumule.

ENVIE D'ALLER PLUS LOIN ?

Besoin d'un regard neuf sur ton architecture,
d'un coup de pouce pour organiser tes environnements ou automatiser tes tests ?
Profite d'un échange gratuit pour faire le point et repartir avec des actions concrètes.

Réserve ton créneau ici : **30 minutes de call gratuit**

Ou contacte-moi par email : arkonium.contact@gmail.com

Je suis Arnaud, fondateur d'Arkonium.

J'accompagne les porteurs de projets et équipes tech à structurer, sécuriser et faire grandir leurs idées sans bullshit.

Passionné par la clarté, la structuration et la transmission.

Ma mission : t'aider à transformer le flou en projet solide et pérenne

Merci d'avoir lu ce mini-guide
et bravo pour ton engagement à bâtir sur de bonnes bases !

Structurer. Sécuriser. Scaler.