DOCUMENTATION BY AI

UADIA Framework

Documenter une application comme une trajectoire causale.

Méthode réutilisable pour reconstruire fonctionnalités, signaux, états, routes, workers, autorité, commit, hydratation et parcours UX depuis les sources réelles.

Principe Ne jamais documenter ce que le système semble faire. Reconstruire ce qu'il fait réellement.

Règle Toute flèche doit correspondre à une preuve : appel, événement, message, dépendance, observer ou transition d'état.

Livrable central Atlas causal + diagramme HTML/CSS interactif + registres de jonctions, risques et sous-systèmes.

Mode d’emploi

Ce document peut être fourni avec un fichier unique, un monolithe HTML/JS/PHP, une application multi-fichiers ou une architecture distribuée. L’analyse doit être menée depuis les sources accessibles, par passes successives si nécessaire, tout en maintenant un registre global unique.

Partie	Objet	Couverture
I	Mission et modèle causal	Sections 0-2
II	Inventaires et registres	Sections 3-9
III	Chargement, navigation et concurrence	Sections 10-19
IV	Synergies, risques et performance	Sections 20-26
V	Livrables, diagrammes et vérité documentaire	Sections 27-34
VI	Commande opérationnelle	Section 35

0. Mission

- Analyser l'application comme un système réel et produire une documentation exhaustive, traçable et diagrammable.
- Reconstruire : fonctionnalités, parcours utilisateurs, voies de signalisation, états, propriétaires, fonctions obligatoires, bus, workers, appels serveur, concurrence, commit, générations/epochs, hydratations, fallbacks, convergences, synergies, redondances, voies mortes et coûts continus.
- Toute affirmation doit être dérivée du code observé. Les commentaires et noms ne sont que des indices.

1. Principe d’analyse

- Analyser selon : USER / ENVIRONMENT -> INTENT -> STATE -> SIGNAL -> ROUTING -> SCHEDULING -> ACQUISITION -> VALIDATION -> COMMIT -> HYDRATION -> CAPABILITY -> USER-visible EFFECT.
- Pour chaque mécanisme : déclencheur, états lus/écrits, signaux reçus/émis, frontière asynchrone, risque de dépassement, arbitre de validité et effet observable final.

2. Séparer les catégories

- Détection : click, scroll, wheel, touch, resize, IntersectionObserver, MutationObserver, hashchange, pageshow.
- Intention : selectThread, setScope, selectCity, openPost, changeContext. État : variables et stores persistants ou temporaires.
- Signalisation : CustomEvent, callbacks, emitters. Ordonnancement : scheduler, microtask, rAF, timers, DAG.
- Routage : endpoint/context resolver. Transport : fetch, worker postMessage, WebSocket, WebRTC, HTTP.
- Autorité : generation, epoch, sequence, request id, AbortController, transaction id. Commit : modification effective. Hydratation : comportement après matérialisation.

3. Inventaire structurel initial

- Compter fichiers, lignes, scripts, styles, fonctions, classes, globals, listeners, CustomEvent, fetch, workers, observers, timers, storages, formulaires, endpoints, routes et modèles mentionnés.

- Cartographier les hotspots par zone/lignes, volume de fonctions, événements, dépendances et responsabilité probable.

4. Inventaire des fonctionnalités

- Pour chaque fonctionnalité : ID, nom, domaine, origine utilisateur, fonction d'entrée, état principal, signaux, réseau, conteneur/ressource, commit, hydratation, fallback, sortie utilisateur, scripts et lignes.

5. Origines causales

- Classer les origines utilisateur, navigateur, observers, réseau et métier. Attribuer O1, O2, O3... et identifier quelles origines convergent vers le même aval.

6. Registre des voies de signalisation

- Pour chaque signal : type, émetteurs, consommateurs, payload, moment causal, mode d'exécution et criticité.
- Classer point-to-point, fan-out, broadcast, commit signal, capability signal, diagnostic et orphaned. Identifier amplification et redondance.

7. Registre des jonctions

- Une jonction est un passage de responsabilité. Pour chaque J#: amont, aval, mécanisme, données, sync/async, autorité, effet final et lignes.
- Exemples : UI->fonction, fonction->bus, scheduler->loader, main->worker, worker->commit, commit->event, event->hydrator.

8. Reconstruction de la spine principale

- Identifier le chemin minimal nécessaire au résultat principal : USER -> intent -> state -> view -> load -> router -> request -> authority -> network/worker -> validation -> commit -> post-commit -> capability.
- Distinguer cette spine de tous les mécanismes transversaux.

9. Voies parallèles

- Documenter séparément analytics, ads, navigation prediction, media planning, diagnostics, layout correction, scroll memory, recommendations, prefetch et telemetry.
- Ne jamais les dessiner comme obligatoires si le parcours principal fonctionne sans eux.

10. Parcours de chargement initial

- PAGE REQUEST -> HTML parsing -> runtime initialization -> route detection -> session/auth detection -> boot branch -> initial UI -> initial acquisition -> first valid commit -> hydration -> navigation-ready.
- Couvrir anonymous/authenticated, public route, deep-link, auth return, mobile/desktop, cached state, targeted post, default route, error et offline si présents.

11. Parcours de navigation

- Construire un flow pour chaque navigation : thread switch, default feed, filtre, search, scope, city, deep-link, post focus, lazy load, pagination, vertical/horizontal navigation, history/hash, création/édition/suppression, session et breakpoint.
- Format : ORIGIN -> INTENT -> UI IMMEDIATE -> SIGNAL -> LOAD -> AUTHORITY -> COMMIT -> HYDRATION -> FINAL STATE.

12. Ligne rapide / ligne lente

- Ligne rapide : même call stack ou frame, feedback visible, aucun réseau. Ligne sub-frame : microtask/rAF. Ligne timer : debounce/backstop. Ligne lente : worker/fetch/server/media.
- Identifier le premier feedback visible de chaque parcours.

13. Modèle d’autorité et concurrence

- Rechercher generation, epoch, sequence, request id, AbortController, locks, busy/loading flags, timestamps, signatures.
- Pour chacun : portée, création, mutation, comparaison, invalidation, consommateurs, effet d'un stale result. Répondre : qui peut écrire, quand, sous quelle condition, qui peut annuler ce droit ?

14. Propriété des états

- Pour chaque état : propriétaire, lecteurs, écrivains, durée de vie, portée, valeur initiale, transitions et signal associé. Les propriétaires multiples sont un risque à signaler.

15. DOM ownership

- Pour chaque conteneur : qui author le HTML, qui l'insère, remplace, append, modifie après commit, installe les handlers et observe les mutations.
- Distinguer AUTHOR, MATERIALIZER, COMMIT AUTHORITY, HYDRATOR, DECORATOR, OBSERVER.

16. Workers

- Pour chaque Worker : création, message entrant, payload, travail, message sortant, consommateur, impact utilisateur, fallback et statut nécessaire/auxiliaire/orphelin.
- Un Worker sans consommateur réel de sortie est OUTPUT ORPHANED.

17. Bus d’événements

- Cartographier les hubs : nombre d'émetteurs, listeners, domaines, scheduling secondaire, scans DOM et timers.
- Nommer CAUSAL AMPLIFICATION lorsqu'une interaction provoque un large éventail de réactions indirectes.

18. Observers

- Inventorier MutationObserver, IntersectionObserver, ResizeObserver : root, scope, options, callback, fréquence, cleanup, coalescing, idempotence.
- Identifier les observers globaux remplaçables par un bus partagé.

19. Timers

- Inventorier délais et raison : debounce, fallback, backstop, compensation d'ordre ou dette historique.
- Comparer les timers 'attendre le DOM' aux vrais événements de commit.

20. Fonctionnalités spécialisées

- Chaque domaine spécialisé doit définir un contrat au-dessus du contrat de base sans contaminer le core.

- Exemple : base request->generation->commit ; media/music intent->revoke->request->commit->visual-ready->capability.

21. Synergies

- Une synergie est stable lorsque les responsabilités sont distinctes, l'interface explicite, l'état non dupliqué et l'ordre déterministe.

22. Synergies instables

- Repérer dépendances à plusieurs flags/clocks, timers, mutation implicite, plusieurs owners, ordre de scripts, global mutable, absence de consumer, fallback non protégé.
- Classer : stable, fragile but guarded, dormant, orphaned, redundant, contradictory, unsafe fallback.

23. Fonctions obligatoires

- CORE : retirer casse le parcours. COHERENCE : le parcours continue mais peut devenir stale/incorrect. CAPABILITY : nécessaire à un domaine spécialisé. DECORATION : présentation, diagnostics, optimisation facultative.

24. Fonctions mortes ou inertes

- Chercher fonctions jamais appelées, events jamais consommés, workers sans consumer, branches impossibles, boutons masqués mais requis, fallbacks inaccessibles, code shadowed, scheduler sans tâches.

25. Contradictions

- Comparer fonction activée vs CSS/state impossible ; event émis vs aucun listener ; bouton requis vs display:none ; scheduler low vs aucune tâche low ; invalidation prévue vs aucun appel invalidate.

26. Performance causale

- Mesurer listeners par intention, scans DOM, timers, observers persistants, requêtes, commits et hydratations.
- Classifier coûts O(boot), O(interaction), O(commit), O(session), O(continuous). Repérer Maps/sets augmentant en O(session).

27. Documents à produire

- A Atlas fonctionnel. B Atlas des voies de signalisation. C Diagramme général HTML/CSS. D Flow chargement->navigation. E Registre des jonctions. F Registre des risques. G Inventaire des sous-systèmes.

28. Règles du diagramme HTML/CSS

- Préférer HTML + CSS + SVG + JavaScript léger : responsive, filtrable, recherchable, accessible, imprimable et sans dépendance externe obligatoire.
- Une couleur représente une nature de mécanisme - UI, bus, scheduler, worker/network, authority, commit, hydration, capability, warning - jamais un domaine métier.

29. Filtres du diagramme

- Prévoir ALL, BOOT, NAVIGATION, NETWORK, WORKERS, EVENTS, AUTH, PUBLIC, MOBILE, DESKTOP, POST, SEARCH, SOCIAL, MEDIA et recherche par fonction/event/container/endpoint/script/subsystem.

30. Règles de vérité documentaire

- Toujours distinguer CONFIRMÉ PAR LE CODE, INFÉRENCE ARCHITECTURALE, COMMENTAIRE DU CODE, HYPOTHÈSE À VÉRIFIER.
- Ne jamais inventer table, endpoint, worker, fonctionnalité, listener, pipeline ou relation.

31. Gestion des incohérences

- Si la documentation précédente contredit le code, le code observé prévaut. Signaler la divergence au lieu de la corriger silencieusement.

32. Vérification finale

- Chaque nœud existe-t-il ? Chaque flèche a-t-elle une preuve ? Chaque worker a-t-il un consumer ? Chaque génération est-elle placée au bon endroit ?
- Le routeur est-il là où l'URL est construite ? Commit et hydratation sont-ils distincts ? Boot et navigation sont-ils séparés ? Les fonctions natives du navigateur ne sont-elles pas requalifiées en mécanismes applicatifs ?

33. Format de restitution final

- Ordre recommandé : résumé architectural, inventaire quantitatif, carte des domaines, spine principale, boot, navigation, bus, workers, autorité/générations, commit, hydratation, fonctionnalités, risques, recommandations.

34. Instruction opérationnelle

- Analyser intégralement avant de simplifier graphiquement. Construire l'inventaire, vérifier les causalités, identifier les jonctions, reconstruire les flows, puis seulement produire les diagrammes.
- Citer les lignes/fichiers utilisés et signaler explicitement toute zone non démontrable. Sur gros corpus, procéder par passes mais conserver un registre global unique.

35. Commande

- Analyse l'application fournie selon ce framework et produis une documentation causale, fonctionnelle et structurelle complète.
- Ne te limite pas aux fonctionnalités dominantes. Recense tous les sous-systèmes réellement présents. Reconstruis les parcours depuis le chargement initial jusqu'à toutes les situations de navigation, interaction, commit et ré-entrée.
- Ne crée aucune fonctionnalité ou relation non démontrée. À la fin, produis un diagramme général HTML/CSS interactif fidèle au système observé.

Annexe A - Fiche canonique d’une fonctionnalité

Champ	Contenu attendu
ID	Identifiant stable
Nom / Domaine	Fonction utilisateur et famille métier
Origine	Geste, navigateur, observer, réseau ou métier
Entrée	Fonction ou handler
État	Propriétaire + transitions
Signal	Émission/écoute + payload

Réseau	Endpoint / worker / protocole
Autorité	Generation / epoch / abort / autre
Commit	Point de matérialisation
Hydratation	Handlers / media / layout / social
Fallback	Mode dégradé
Effet final	Résultat observable
Preuve	Fichier + lignes

Annexe B - Schéma minimal UADIA_SYSTEM_ATLAS.json

```
{
  "origins": [],
  "states": [],
  "signals": [],
  "junctions": [],
  "workers": [],
  "routes": [],
  "flows": [],
  "subsystems": [],
  "risks": [],
  "evidence": []
}
```

Annexe C - Prompt court prêt à l’emploi

Analyse l’application fournie selon le framework Documentation By AI - UADIA. Inventorie les sources, sépare les catégories causales, reconstruis origines, signaux, états, routes, workers, générations, commit et hydratation ; établis la spine principale, les voies latérales, le flow chargement-navigation, les registres de jonctions/risques et les preuves par lignes ; ne crée aucune relation non démontrée ; produis ensuite le diagramme HTML/CSS interactif.