Université de Bretagne Sud
École nationale supérieure d'ingénieurs de Bretagne Sud
Cycle d'ingénieur informatique et cybersécurité – Parcours Étudiant

---

TP : Architecture Decision Records (ADR) pour la Documentation des Décisions d'Architecture Logicielle

réalisé par : **OUCHTA Nazih - ZENDOUR Younes**

Année universitaire : **2025-2026**

Partie 1 : Étude théorique

1. Qu'est-ce qu'un ADR ? Quels sont ses avantages par rapport à une documentation classique ?

Un ADR (Architecture Decision Record) est un mini-document qui capture une décision architecturale importante, avec son contexte, la décision, les alternatives, et les conséquences. C'est une façon légère, rapide et durable de garder une trace des choix d'architecture.

Avantages par rapport à une documentation classique

- Focalisé sur le "pourquoi", pas uniquement sur le "quoi".
- Très court, facile à maintenir, contrairement aux gros documents rarement mis à jour.
- Versionnable dans Git → historique clair, traçabilité naturelle.
- Aide les nouveaux arrivants à comprendre rapidement les choix du projet.
- Agile-friendly : un ADR = une décision, un commit → parfaitement intégré dans le workflow dev.

2. Trois outils ou méthodes pour gérer les ADR et comparaison

Outil / Méthode	Description	Avantages	Inconvénients
adr-tools (Nat Pryce)	CLI pour créer, lister et lier les ADR.	Simple, standard, rapide.	Peu personnalisable.
MADR (Markdown Architectural Decision Records)	Template moderne et structuré en Markdown.	Clair, expressif, bien structuré.	Plus verbeux qu'un ADR minimaliste.
Gestion manuelle dans Git (Markdown + conventions)	Dossier doc/adr/ géré manuellement par l'équipe.	Très flexible, aucun outil à installer.	Risque d'incohérence si l'équipe ne suit pas le template.

3. Pourquoi documenter les alternatives et les conséquences ?

Documenter seulement la décision ne suffit pas.

Les **alternatives** montrent :

- qu'un vrai raisonnement a été mené,
- pourquoi certaines options ont été rejetées,
- évitent de refaire les mêmes analyses plus tard.

Les **conséquences** rendent explicites :

- les impacts positifs,
- les limites et risques techniques,
- les effets à long terme sur la maintenabilité et l'évolutivité.

En pratique, ça permet de comprendre *la logique derrière la décision* et d'anticiper ce qu'il faudra adapter si le système évolue.

4. Risques liés à une mauvaise gestion des décisions d'architecture

- Dette technique s'accumule silencieusement.
- Décisions incohérentes entre équipes, perte d'alignement.
- Perte de contexte : on ne sait plus pourquoi une décision a été prise.
- Refactoring coûteux car les impacts ne sont pas connus.
- Dépendance au savoir oral des développeurs → fragile, non pérenne.

Partie 4 : Analyse et discussion

1. Avantages et limites des ADR pour notre projet

Avantages	Limites
Traçabilité claire des décisions : chaque décision est isolée dans un fichier, versionnée dans Git et facile à retrouver. Permet de comprendre pourquoi un choix a été fait même plusieurs mois plus tard.	**Rigueur nécessaire** : il faut de la discipline pour rédiger un ADR à chaque décision importante. Certaines décisions sont parfois prises trop vite pour être documentées.
Documentation légère et maintenable : les ADR sont courts, ciblés et simples à mettre à jour, contrairement à un document d'architecture monolithique. Parfait pour un contexte agile ou étudiant.	**Granularité difficile à définir** : pas toujours évident de décider si une décision mérite un ADR (trop petit = surcharge ; trop gros = perte de précision).
Alignement de l'équipe : les ADR jouent le rôle de mémoire collective, garantissant que tout le monde comprend les choix architecturaux.	**Multiplication des fichiers** : sur un long projet, les ADR peuvent devenir nombreux, rendant la navigation difficile sans index ou organisation stricte.

Intégration naturelle dans Git : un ADR = un commit → historique propre, diffs lisibles, revue facile. Les hooks et tags peuvent renforcer le processus.	

2. Comment automatiser la gestion des ADR ?

Plusieurs niveaux d'automatisation sont possibles

a) Validation automatique via un hook Git (déjà implémenté)

- Le hook **pre-commit** vérifie que chaque ADR contient les sections obligatoires du template.
- C'est un **premier niveau d'assurance qualité**.

b) GitHub Actions / GitLab CI

Un pipeline CI pourrait :

- Vérifier que chaque ADR respecte une structure Markdown standard (regex, linter).
- **Bloquer une PR** si le format n'est pas respecté.
- **Générer automatiquement un sommaire** des ADR après chaque merge.

Exemple : Un job CI exécute un script Python/Bash pour scanner le dossier doc/adr/ et produire un fichier index.md.

c) Bot GitHub spécialisé (type Renovate)

Un bot pourrait :

- Détecter qu'un fichier ADR a été modifié sans mise à jour des liens vers d'autres ADR.
- Proposer automatiquement la création d'un nouvel ADR à partir d'un template.
- Ajouter ou mettre à jour des **tags Git** (ex. : v1.1-adr-update).

d) Intégration dans un site statique (MkDocs, Docusaurus)

À chaque push, le bot reconstruit une documentation web avec une navigation propre, une table des matières, un moteur de recherche, etc. Cela transforme les ADR en véritable documentation vivante.

3. Une amélioration au processus de rédaction des ADR

Ajouter une section "Risques & Atténuations"
Actuellement, les ADR incluent "Conséquences", mais cela reste général, Une section dédiée permettrait d'aller plus loin en clarifiant :

- les risques techniques (ex : couplage, dette technique),
- les risques opérationnels (ex : rotation de secrets mal gérée),
- les risques de sécurité,
- les stratégies d'atténuation envisagées.

Exemple :

```
## Risques & Atténuations
- Risque : verrouillage excessif dans PostgreSQL lors de pics d'emprunts.
  Atténuation : surveillance des métriques et optimisation des transactions si charge augmente.
```

Cela améliore la lisibilité pour les reviewers, rend les impacts plus explicites et facilite la transition si une décision doit être revue.