Xavier Penet Gillig

- Demande : mini rapport dans RAPPORT.md détaillant
    - points de blocage, solutions, points difficiles
    - utilisation IA générative, limites, problèmes, apprentissages

RAPPORT.md

Dossier – nom : atelier

1. La mise en place de React Router

Au début, j'ai un peu galéré avec le système de routes.
Je comprenais la logique générale, mais j'étais perdu entre :

- `<Route index …>`
- les chemins imbriqués dans `<Layout>`
- comment `<Outlet />` affichait réellement les pages

J'ai dû relire plusieurs fois la doc et tester différents placements des routes avant que tout fonctionne correctement. Maintenant le fonctionnement me paraît plus clair.

2. Le typage TypeScript dans les états

J'ai eu un blocage sur le typage de mes tâches.
J'avais l'impression de bien faire, mais TS me lançait des erreurs quand je faisais `useState<Tache[]>(...)`.
J'ai compris que mes types devaient être cohérents partout et que la moindre faute de signature entraînait des incohérences lors du JSON.parse.
C'est un truc que j'ai dû ajuster plusieurs fois.

3. Le localStorage

J'ai eu du mal à comprendre à quel moment charger les données et à quel moment les sauvegarder.
J'ai testé plusieurs `useEffect` avant de comprendre que la meilleure façon était de :

- charger les tâches **directement dans le useState via une fonction anonyme**

- puis sauvegarder les tâches dans un `useEffect` dépendant de `taches`

Avant ça, j'avais des comportements bizarres : des tâches dupliquées, des reset au mauvais moment… Ça m'a un peu frustré, mais au final j'ai compris comment organiser les effets correctement.

4. Remplir correctement le formulaire contrôlé

Le formulaire ne mettait pas à jour la valeur au début.
J'avais oublié le classique `value={texte}` + `onChange`.
C'est simple, mais c'est le genre de détail qui t'arrête net tant que tu ne le vois pas.

Points difficiles

- Comprendre suffisamment bien le workflow React pour organiser les composants et la logique.
- Écrire du code propre **et** typé, sans tomber dans des erreurs TS incompréhensibles.
- Trouver où mettre quelle logique : dans le composant ? dans un useEffect ? dans le handler ?
- Réussir à faire fonctionner les routes comme je voulais, avec un Layout commun.

Je dirais que le plus difficile a été d'équilibrer **simplicité** et **bonne structure**, surtout avec TypeScript.

Utilisation de l'IA générative (ChatGPT)

Comment je m'en suis servi

J'ai utilisé l'IA comme un assistant ponctuel pour :

- comprendre certaines erreurs TypeScript
- vérifier mes syntaxes de `useEffect`
- trouver la bonne structure pour le gestionnaire de tâches
- me débloquer quand j'étais coincé sur le routing

- revoir comment fonctionne `map`, `filter` et les patterns de mise à jour d'un state React

Ses limites

- L'IA donne parfois des morceaux de code qui ne collent pas à ma structure de projet.
- Parfois elle mélange les versions de React Router (v5/v6), ce qui m'a induit en erreur.
- Certaines réponses partent trop loin ou proposent des solutions plus complexes que nécessaire.
- Elle ne voit pas mes fichiers en entier (sauf quand je les fournis), donc elle peut faire des suppositions fausses.

Problèmes rencontrés

- Certains conseils d'IA ont créé de nouvelles erreurs TS.
- Parfois, les solutions semblaient correctes mais ne marchaient pas dans mon contexte actuel.
- L'IA n'explique pas toujours *pourquoi* ça marche ou pas, il faut creuser soi-même.

Ce que j'ai appris

Ce projet m'a réellement permis de :

- mieux comprendre React Router et l'organisation des routes
- manipuler des états typés en TypeScript
- comprendre la persistance locale avec `localStorage`
- éviter les pièges des `useEffect`
- structurer une petite application React de façon propre
- utiliser l'IA intelligemment, comme un outil de support et non comme une solution magique

RAPPORT.md – Projet Next.js -

Dossier – nom : api-next

Points de blocage que j’ai rencontrés

1. Comprendre la structure du dossier `/app`

C’est sûrement le truc qui m’a donné le plus de mal au début.
J’avais encore les réflexes du premier projet (React + Router), donc pour moi, une page = une route déclarée quelque part dans un `<Route>`.

Là, non :

- tu mets un dossier `contact/` → automatiquement ça devient `/contact`
- un `page.tsx` dedans → c’est la page
- un `layout.tsx` au-dessus → ça s’applique à toutes les routes

J’ai perdu un peu de temps à comprendre pourquoi rien n’apparaissait ou pourquoi ma page ne se chargeait pas : j’étais persuadé que je devais tout déclarer quelque part.
Quand j’ai enfin compris la logique “filesystem routing”, ça m’a paru simple, mais avant ça, c’était vraiment confusant.

2. Le `layout.tsx` asynchrone (et le fetch)

Dans mon layout, j’ai mis :

```
const response = await fetch("http://localhost:3000/api/users");
const data = await response.json();
```

Au début, j’ai trouvé ça bizarre de pouvoir faire un `await` dans un composant React.
J’avais vraiment l’impression de “casser” le modèle React normal.
J’ai découvert que **les composants du dossier /app sont server components par défaut**.
Ça veut dire qu’ils s’exécutent côté serveur, donc qu’on peut faire des fetchs, sans useEffect, sans state, sans rien.

J’ai dû fouiller sur Internet et dans la doc pour être sûr que c’était normal.
Maintenant je comprends mieux ce que Next.js apporte comme surcouche à React.

3. L'API interne (`route.ts`)

Créer un fichier `route.ts` et voir qu'il devenait automatiquement une API était nouveau pour moi.

J'ai eu plusieurs problèmes :

- je ne savais pas dans quel dossier exact mettre `route.ts`
- j'ai dû deviner que l'endpoint serait `/api/users`
- j'ai mis dans mon layout `data.message` alors que ma route renvoyait `{ users: [...] }`, ce qui faisait un undefined

Ce n'était pas un gros bug, mais c'était suffisamment pénible pour me faire perdre du temps.

J'ai compris que Next.js utilise un système très minimaliste pour les routes API, donc il faut être attentif à ce qu'on renvoie et comment on le consomme.

4. Le template Next.js/Tailwind trop rempli

Quand tu lances `create-next-app`, il te génère une page d'accueil super complète, avec plein de div, de classes Tailwind et d'éléments de design.

C'était un peu déroutant.
J'ai dû lire, supprimer, réorganiser et comprendre ce qui était utile pour moi.
Ce n'était pas compliqué techniquement, mais ça a rendu le début plus long que prévu.

Points difficiles pour moi

- **Me défaire de mes habitudes React classiques**
  Next.js ne fonctionne pas du tout comme un client React classique.
- **Le concept de Server Component**
  J'ai eu du mal à comprendre ce qui s'exécute côté serveur et ce qui s'exécute côté client.
- **La multiplicité des fichiers**
  Entre `layout.tsx`, `page.tsx`, `globals.css`, `route.ts`, `metadata`, les fonts, etc., on peut vite s'y perdre.

- **La structure imposée par Next**
  On n’est pas libre de tout faire comme on veut, il faut respecter les conventions.

Utilisation de l'IA générative

Comment je l’ai utilisée

J’ai surtout utilisé l’IA pour :

- comprendre pourquoi mon Layout pouvait être asynchrone
- clarifier la différence entre Server/Client Components
- vérifier comment créer une API Route (GET) dans `route.ts`
- trouver où placer mes fichiers
- vérifier mes erreurs quand j’obtenais un `undefined` dans mon fetch

L’IA m’a aidé à débloquer des concepts rapidement quand la doc officielle était trop dense.

Ses limites

- l’IA mélange parfois l'ancien système Next.js (`pages/`) et le nouveau (`app/`)
- elle me donnait parfois des exemples qui ne correspondent pas à ma version
- certaines réponses étaient trop générales ou trop longues
- elle ne voit pas automatiquement ma structure réelle (sauf quand je lui ai donné)
- parfois elle me donnait une solution plus compliquée que nécessaire

Finalement l’IA m’a aidé, mais j’ai dû beaucoup vérifier moi-même, si on comprend pas un minimum le code bah l’ia peut me coder des choses qui n’ont aucun rapport si je n’utilise pas les bons prompts

Ce que j’ai appris réellement

- J’ai compris comment Next.js organise ses routes à partir des dossiers
- J’ai appris à utiliser un layout global pour ma navigation
- J’ai découvert comment créer une API interne sans Express
- Je comprends mieux le fonctionnement des Server Components