NextJS

https://nextjs.org/

> **INFO**
>
> Il est necessaire de bien comprendre les concepts de base de React pour pouvoir utiliser correctement NextJS.
>
> Consulter d'abord le cours sur React

NextJS est un framework frontend pour React. Il permet de créer des applications web performantes avec rendu coté serveur, optimisées pour les utilisateurs et le SEO.

Il est aujourd'hui largement répandu et les plus grandes entreprises utilisent NextJS pour leurs sites web (stripe, OpenAI, Nike, etc).

Créé par la même équipe que Vercel, il y est très bien integré et il est très facile d'y déployer des applications.

Fonctionnalités

- Routeur integré, basé sur la structure des dossiers
- Differentes méthodes de rendu (SSR, SSG, ISR, RSC, CSR, ...)
- API serverless
- Optimisation des resources
- Optimisation du SEO

Evolution de NextJS

NextJS a beaucoup évolué depuis sa création, notamment avec la migration depuis le "page router" vers le "app router", où la façon de s'en servir a

beaucoup changé. Il est important de bien comprendre la difference entre ces méthodes et d'exploiter l'"app router" de la meilleure façon possible.

> Attention quand on lit la documentation NextJS à la version sur laquelle on travaille.

Structure du projet

Un projet NextJS est composé dans une structure bien définie qui determine le comportement de l'application.

https://nextjs.org/docs/app/getting-started/project-structure

Dossiers

- app: dossier contenant les pages et les composants (app router)
- ~~pages: dossier contenant les pages et les composants (page router)~~
- public: fichiers statiques

Tous les dossiers définis dans le dossier `app` sont considérés comme des routes. L'url `monsite.com/a/b/c` sera rendue par le composant `/app/a/b/c/page.tsx`

Les routes dynamiques sont définies par des dossiers avec des crochets tel que `[id]`. Par exemple, `monsite.com/user/2134` sera rendue par le composant `/app/user/[id]/page.tsx`

- https://nextjs.org/docs/app/getting-started/layouts-and-pages
- https://nextjs.org/docs/app/building-your-application/routing/loading-ui-and-streaming

Route files (.tsx)

Certains noms de fichiers provoquent des comportements speciaux.

- layout.tsx: composant englobant la page

- page.tsx: page de la route définie par l'arborescende de fichiers
- route.ts: route handler (API)
- loading.tsx: composant de chargement (rendu serveur)
- not-found.tsx: composant de page non trouvée (dynamiques)
- error.tsx: composant d'erreur
- sitemap.ts: fichier de sitemap
- opengraph-image.tsx: fichier de génération d'image OpenGraph
- favicon.ico
- ...

Sauf pour `page` et `route`, ils ont une consequence sur toutes leur sous-routes. (`layout` , `error` , `not-found` , ...)

Variables d'environnement

https://nextjs.org/docs/app/guides/environment-variables

Les données sensibles ne doivent pas être stockées dans le code source de l'application mais dans des variables d'environnement.

Elles peuvent être définies dans le fichier `.env` , qui ne doit pas être versionné. Lors du déploiement, les variables d'environnement sont modifiables sur le dashboard de configuration de la plateforme.

Seules les variables prefixées par `NEXT_PUBLIC_` sont accessibles côté client.

```
DATABASE_PASSWORD=abcd // Uniquement accessible côté serveur
NEXT_PUBLIC_API_URL=https://api.example.com // Accessible côté clien
```

Hydration

Lorsque NextJS est utilisé en mode server side rendering, toute la page sera generée côté serveur, pas seulement les Server Components (RSC).

Cela signifie que les composants avec la directive `use client` seront d'abord generés côté serveur, puis hydratés côté client.

L'hydration est le processus par lequel le client va prendre le contenu generé côté serveur, le transformer en composants React et les reconnecter au DOM.

Il est très important de faire attention au rendu initial d'un composant, et de faire en sorte qu'il soit exactement le même des deux côtés, sinon une erreur d'hydration apparaitra et l'application ne fonctionnera pas correctement.

```tsx
'use client'

function MyComponent() {
    // 🧯 La valeur initiale sera differente côté serveur et côté cl
    const [value, setValue] = useState(Math.random());
    return <div>{value}</div>
}
```

Rendering

NextJS propose plusieurs méthodes de rendu, qui déterminent le moment où les données sont récupérés. Ces méthodes peuvent être choisies independemment au niveau des composants.

Un rendu peut etre fait:

- Une fois au moment du build (static)
- A chaque requête, côté serveur (server side rendering)
- A chaque requête, côté client (client side rendering, static)
- A intervalle ou à la demande (incremental static regeneration)

A lire:

- https://nextjs.org/docs/app/getting-started/partial-prerendering

- https://nextjs.org/docs/app/building-your-application/data-fetching/incremental-static-regeneration
- https://nextjs.org/docs/app/guides/static-exports

A lire

Articles importants à lire sur la documentation officielle:

- Data fetching
- Caching
- Server actions

En complément

- MDX

Projet d'exemple

Une application NextJS simple pour illustrer les concepts vus dans le cours.

Demo NextJS (Code source)

Fonctionnalités mises en avant

- Passage de variables entre composants avec des props
- Etat dynamique avec useState
- Effets de bord avec useEffect
- Memoisation de donnée avec useMemo
- Mémoisation de fonctions avec useCallback
- References vers les elements du DOM
- React Server Components
- Client components
- Data fetching côté serveur
- Data fetching côté client
- Data fetching côté serveur avec streaming sur des client components
- Utilisation des variables d'environnement
- Routing dynamique
- Server actions
- Route handlers

L'authentification avec NextJS

Il est recommandé de consulter le guide officiel pour l'authentification.

Cookies

Les cookies sont un espace de stockage local au navigateur qui permet d'identifier l'utilisateur auprès du serveur. C'est le serveur qui décide des cookies à créer chez le client, le navigateur les conserve et les envoiera automatiquement à chaque requête à ce serveur.

Les cookies sont stockés par origine, c'est à dire que chaque site web a ses propres cookies.

Options des cookies:

- `SameSite` : définit les origines qui peuvent accéder au cookie
- `HttpOnly` : définit si le cookie est accessible via JavaScript
- `Secure` : définit si le cookie est envoyé uniquement sur HTTPS
- `Max-Age` : définit la durée de vie du cookie

D'autres solutions existent pour l'authentification, comme passer des tokens d'authentification dans les headers de la requête (Bearer token), qui doivent alors être stockés dans le local storage du navigateur, et qui doit être envoyé volontairement par le client. Cela implique que le code javascript de l'application soit déjà lancée pour acceder à cette donnée et envoyer les requetes avec les headers.

Dans le contexte d'une application NextJS avec rendu côté serveur, lorsque le serveur reçoit une requête, pour une page, l'application n'a pas encore démarré côté client et il est donc impossible d'envoyer des token récuperés dans le local storage. Etant donné que le navigateur envoie automatiquement les cookies, ils sont donc une meilleure solution pour l'authentification.

```ts
// /app/api/login/route.ts
'use server'
import { cookies } from 'next/headers'

export async function login(request: Request) {
  const expiresAt = new Date(Date.now() + 7 * 24 * 60 * 60 * 1000)
  const token = await createJWT({ userId, expiresAt })
  const cookieStore = await cookies()

  cookieStore.set('access_token', token, {
    httpOnly: true,
    secure: true,
    expires: expiresAt,
    sameSite: 'lax',
    path: '/',
  })
}
```

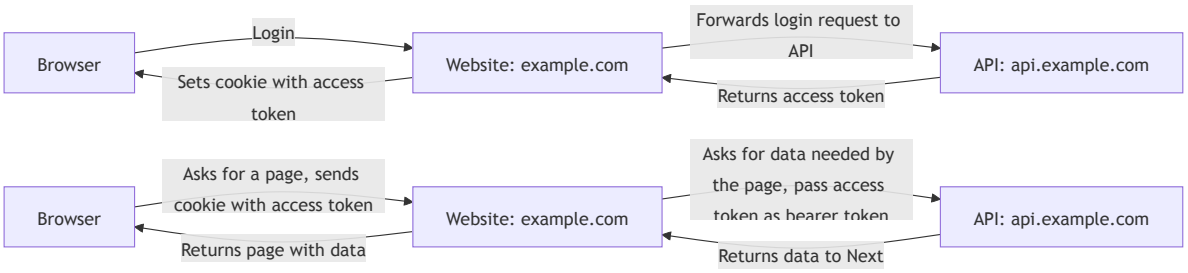
```tsx
// /app/page.tsx
import { cookies } from 'next/headers'

export default async function Profile() {
    const cookieStore = await cookies()
    const token = cookieStore.get('access_token')
    if (!token) {
        redirect('/login')
    }
    const user = await getUser(token)
    return <div>{user.name}</div>
}
```

Authentification avec une API externe

Si notre application utilise une API construite en dehors de NextJS et que l'on souhaite faire du rendu côté serveur, ce sera à NextJS d'envoyer les token

d'authentification à cette API.



NextAuth

NextAuth est une librairie avancée de gestion de l'authentification pour NextJS.

https://next-auth.js.org/

Exercices NextJS

Ces exercices sont des entrainements aux concepts de bases à maitriser avant de se lancer dans un projet plus complexe.

Pour travailler sur ces exercices, vous devrez forker le projet de démonstration et créer une nouvelle branche, puis créer une pull request sur le projet d'origine pour valider votre travail.

Instructions generales

Pour tous les exercices, il faudra faire en sorte d'écrire des composants les plus optimisés possible. Cela signifie d'utiliser au maximum les Server Component, Server Actions, le streaming, et une optimisation des cycles de vie des composants.

Pages

Memecoin list

- Ajouter une page `/memecoins`
- Dans cette page, ajouter une section qui affiche une liste de memecoins
  - Récupérer la liste de memecoins depuis l'adresse
  - `https://nuxt-demo-blush.vercel.app/api/get-memecoins`
  - Les afficher dans la page
- Découper la page en composants (MemecoinList + MemecoinItem par exemple)
- Dans cette page, ajouter un nouveau composant qui contiendra un formulaire pour créer un memecoin
  - Entrées du formulaire :

    - `name`, 4-16 caractères, obligatoire
    - `symbol`, 2-4 caractères, obligatoire, uniquement des majuscules
    - `description`, 0-1000 caractères, pas obligatoire
    - `logoUrl`, 0-200 caractères, pas obligatoire, doit être une URL valide
  - Envoyer sous forme de requête POST à cette adresse
    - `https://nuxt-demo-blush.vercel.app/api/create-memecoin`
  - Gérer la validation des entrées dans l'interface
    - Empêcher l'envoi du formulaire si les conditions ne sont pas respectées
    - Afficher des messages d'erreurs de validation
  - Afficher le résultat de l'operation
  - Rafraîchir la liste des memecoins après l'ajout
  - Rafraichir régulierement la liste automatiquement

> L'API proposée retourne volontairement des erreurs de manière aléatoire afin d'assurer une gestion d'erreurs de manière robuste dans le frontend

Memecoin details

- Ajouter une page `/memecoins/[id]` qui affichera un seul memecoin.
  - Récupérer le memecoin depuis l'adresse
  - `https://nuxt-demo-blush.vercel.app/api/get-memecoins/[id]`
- Retourner des métadonnées SEO (opengraph) pour ces pages

Authentification

Ajouter un système d'authentification:

- Creer un route handler sur `/api/login` qui:
  - Reçoit en POST un mot de passe à définir en constante
  - Crée un token JWT et le définit sur le client dans un cookie
- Ajouter une page `/login` qui permet de se connecter
- Faire évoluer l'UI en fonction de l'état connecté

- Si on est connecté :
  - Afficher un bouton pour se déconnecter dans la navbar
- Si on est déconnecté :
  - Afficher un bouton pour se connecter dans la navbar
- Creer un middleware qui redirigera l'utilisateur si il accède aux pages de login et de creation de memecoin sans etre connecté, et à la page de login si il est deja conencté

Alternatives

- Choisir un composant qui récupère des données, et le dupliquer en plusieurs composants qui chacun recupèrent les données en mode RSC, CSR, ISR, SSG et streaming.
- Creer un composant qui affiche et stocke une donnée dans le localStorage

---

Route handlers

Reproduire les routes utilisées (`https://nuxt-demo-blush.vercel.app/api/get-memecoins` etc...) dans ce projet. Commencer avec des données stockées dans des variables, puis utiliser une base de données et un ORM type prisma.

[Projet] Plateforme de memecoin

Maintenant que vous maitrisez les concepts de base, vous allez créer une plateforme de memecoins !

Description du projet

L'application est une plateforme de memecoins, où les utilisateurs peuvent créer et acheter/vendre des memecoins.

Les utilisateurs peuvent gagner du ZTH (la monnaie native de la plateforme) en créant des memecoins, en achetant stratégiquement des coins tendance, et en les vendant au moment optimal. Le solde ZTH d'un utilisateur sert de score sur le classement de la plateforme.

> Bien que le site web et ses fonctionnalités ressembleront à un projet web3, tout se déroulera off-chain et n'utilisera aucune blockchain.

Plateforme d'exemple

Une application de démonstration qui fournit les fonctionnalités demandées est disponible à l'adresse suivante: https://zero-to-hundred-frontend.onrender.com

L'objectif est de reproduire une application similaire.

Travail demandé

L'interface utilsateur devra être soignée, réactive et avec un design élégant.

Vous pouvez vous inspirer de sites tels que pump.fun memecoin.org deployyyyer

Contraintes techniques

- Application developpée avec NextJS v15 avec App Router
- Optimisation de l'application, utilisation au maximum des server components et server actions
- Optimisation des composants (~ utilisation de useMemo et useCallback)
- Authentification sécurisée
- Bonne gestion du SEO, sitemap et metadata
- Utilisation stricte de Typescript
- Utiliser à differents endroits au moins une fois chacune des methodes de data fetching suivantes:
  - RSC pur
  - CSR pur
  - RSC > CSR sans streaming
  - RSC > CSR streaming

Recommandations

- Base de données: PostgreSQL avec Prisma/Drizzle
- Style: TailwindCSS avec shadcn/ui ou équivalent

Bonus

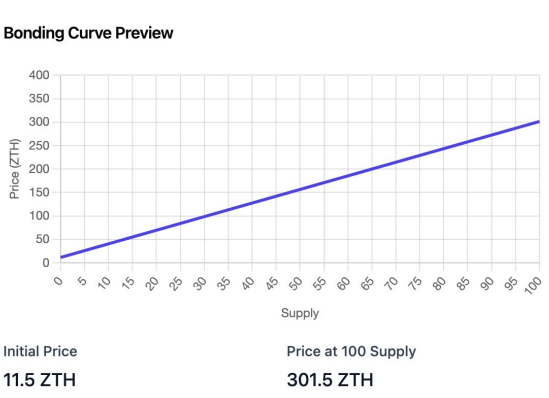
- Tests unitaires
- Tests E2E
- Mise à jour de l'interface en temps réel (prix, transactions, ...)

Tokenomics

- Chaque utilisateur reçoit 100 ZTH à l’inscription.
- Créer un memecoin coûte 1 ZTH

- Le trading des memecoin se base sur un mécanisme de *bonding curve*
  - L'échange ne se fait pas entre utilisateurs mais via une reserve de liquidité en ZTH
  - Le prix est directement lié à la quantité de token existant
  - Lorsqu'un utilisateur achète (mint) un token (avec du ZTH), le prix de celui-ci augmente et les ZTH dépensés sont placés dans la reserve
  - Lorsqu'un utilisateur vend (burn) un token, son prix diminue et il reçoit des ZTH venant de la reserve
  - Le montant de ZTH dans la reserve est toujours égale au prix de vente de la totalité des tokens existant
- Toute cette logique est déjà gérée par le backend fourni

> Dans un premier temps il sera accepté de fournir une logique de trading très basique afin de se concentrer sur l'interface en elle même.



Bonding curve preview

Formule de prix

Pour une bonding curve linéaire, le prix P d'un token est directement proportionnel à la quantité de tokens en circulation S (supply):

`P = a * S + b`

Avec a (slope) et b (starting price) des constantes.

Pour calculer le prix d'achat ou de vente, il faut donc calculer l'intégrale de la fonction de prix entre la quantité de tokens en circulation actuelle et la quantité de tokens en circulation après l'achat ou la vente.

Pour acheter X tokens, le cout C revient à :

```
C = a * ((X+S)^2 - S^2) / 2 + X * b
```

Projets alternatifs

Le projet proposé est une idée d'exemple, mais vous pourrez partir sur un autre sujet de votre choix si vous le souhaitez, tant que les contraintes techniques sont respectées.

Voici d'autres suggestions de projets:

- Projet perso de votre choix
- Application de gestion de playlist des plateforme de streaming (spotify, deezer, ...) et synchronisation entre les plateformes (exemple)