



Next.js

# REACT AVANZATO

# Introduzione


- Cos'è

- Framework **React** per lo sviluppo di applicazioni
  - **Server-Side Rendering**
  - **Static Site Generation**

```
// Installazione
npx create-next-app@latest my-app --typescript
// Esempio di una pagina base
const Home = () => {
  return <h1>Benvenuto in Next.js!</h1>;
};
export default Home;
```



# Rendering

- **SSR** – Server Side Rendering
    - Pagina generata server-side ad ogni richiesta
  - **SSG** – Server Site Generation
    - Pagina generata staticamente in fase di building
  - **ISR** – Incremental Static Regeneration
    - Aggiorna le pagine statiche senza rebuild
- 

# Rendering

- SSR

- Pagina generata completamente sul server ad ogni richiesta
- Il server invia al browser un HTML completamente renderizzato e pronto per essere visualizzato
- Migliora l'accessibilità e il SEO
  - Perché i motori di ricerca e i client possono interpretare direttamente l'HTML renderizzato

# Rendering


- **Next.js** permette di implementare **SSR** utilizzando la funzione **getServerSideProps()**
  - Questa funzione viene eseguita sul server ad ogni richiesta, recupera i dati necessari e li passa al componente React come **"props"**
    1. Richiesta di pagina SSR
    2. Elaborazione della richiesta con esecuzione di **getServerSideProps()**
    3. Generazione della pagina HTML e invio al client
- Questo processo si traduce in un'ottima esperienza per l'utente, poiché ottiene contenuti "pronti per l'uso" fin dal primo caricamento

# Rendering

- Vantaggi di SSR
  - SEO Ottimale
    - Il contenuto è immediatamente disponibile per i motori di ricerca
  - Esperienza utente migliorata
    - L'utente vede una pagina completamente renderizzata molto più velocemente rispetto ai metodi di rendering lato client
  - Personalizzazione dinamica
    - Ogni utente può ottenere contenuti specifici per la propria richiesta



# Rendering

- Svantaggi di SSR
    - Maggiore carico del server
      - Ad ogni richiesta il server riesegue il rendering completo della pagina
    - Prestazioni ridotte
      - Per richieste ad alto traffico ci potrebbero essere dei rallentamenti
- 

# Esempio di SSR

```
// Pagina SSR in Next.js
import { GetServerSideProps } from 'next';

interface PageProps {
  data: string;
}

const SSRPage = ({ data }: PageProps) => {
  return (
    <div>
      <h1>Server-Side Rendering</h1>
      <p>{data}</p>
    </div>
  );
};
```

```
export const getServerSideProps: GetServerSideProps = async (context) => {
  // Recupero dati dinamici (ad esempio da un'API esterna)
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();

  return {
    props: {
      data: data.message, // Passaggio dei dati al componente
    },
  };
};

export default SSRPage;
```





# Perché SSR?

- Contenuti dinamici
  - Pagine che richiedono aggiornamenti frequenti in base alle richieste dell'utente
    - Es.: Dashboard personalizzate
- SEO Prioritario
  - Siti che necessitano di contenuti indicizzabili subito
    - Es.: E-Commerce con schede di prodotto dinamiche
- Sicurezza
  - Necessità di nascondere dati sensibili non presentabili client-side

# Rendering

- Le tecnologie sono utilizzabili in modalità ibrida

```
// Server-Side Rendering
export const getServerSideProps = async () => {
  const data = await fetchData();
  return { props: { data } };
};

// Static Site Generation
export const getStaticProps = async () => {
  const data = await fetchData();
  return { props: { data } };
};
```



ISR

```
export const getStaticProps = async () => {
  const data = await fetchData();
  return {
    props: { data },
    revalidate: 60, // Ricarica ogni 60 secondi
  };
};
```

# Ottimizzazione Immagini

- Implementa caratteristiche quali
  - Ridimensionamento automatico
  - Formato Webp


```
import Image from 'next/image';

const MyImage = () => (
  <Image
    src="/example.jpg"
    alt="Esempio"
    width={500}
    height={300}
    priority
  />
);

export default MyImage;
```



# Formato Webp

- Formato sviluppato da Google
  - Vantaggi
    - Compressione efficiente
      - Lossy e Lossless
      - Dimensioni ridotte
    - Qualità migliore
    - Alpha channel
    - Caricamento più veloce
    - Supporto per le animazioni
    - Risparmio di banda
- 

# Internazionalizzazione

- Configurabile nel file `next-config.js`

```
module.exports = {  
  i18n: {  
    locales: ['en', 'it', 'fr'],  
    defaultLocale: 'it',  
  },  
};
```

# Configurazioni Personalizzate

- Presenza di middleware per reindirizzamenti dinamici

```
import { NextResponse } from 'next/server';

export function middleware(req) {
  if (req.nextUrl.pathname === '/old-path') {
    return NextResponse.redirect('/new-path');
  }
}
```




# Performance e Ottimizzazione

- Adozione di meccanismi di
  - Prefetching
  - Lazy loading



# Routing

- Modello di routing file-based
    - La struttura delle cartelle definisce le rotte
  - Due modalità principali
    - **pages** directory
      - Il modello tradizionale
    - **app** directory
      - Il nuovo sistema con supporto per il routing gerarchico e funzioni avanzate
- 



# Directory pages

- Ogni file in **pages/** diventa una rotta (statica o dinamica)


```
// File: pages/about.tsx
const About = () => <h1>About Page</h1>;
export default About;
```

```
// File: pages/user/[id].tsx
import { useRouter } from 'next/router';

const User = () => {
  const router = useRouter();
  const { id } = router.query;
  return <h1>Profilo utente: {id}</h1>;
};
export default User;
```



# Directory app

- Introdotto in Next.js 13+ per migliorare la modularità
  - Basato su routing gerarchico
  - Concetti chiave
    - Supporto per routing annidati
    - Server components per il rendering
- 

# Directory app

- Pagina accessibile con rotta **/profile**

```
// File: app/profile/page.tsx
const ProfilePage = () => <h1>Pagina Profilo</h1>;
export default ProfilePage;
```

- Rotta dinamica **/user/123**

```
// File: app/user/[id]/page.tsx
type UserProps = {
  params: { id: string };
};

const UserPage = ({ params }: UserProps) => {
  return <h1>Profilo utente: {params.id}</h1>;
};
export default UserPage;
```

# API Routes

- Creazione di API backend direttamente in Node.js
  - Basata su directory `pages/api` o `app/api`

```
// File: pages/api/hello.ts
import { NextApiRequest, NextApiResponse } from 'next';

export default function handler(req: NextApiRequest, res: NextApiResponse) {
  res.status(200).json({ message: 'Ciao da Next.js!' });
}
```



# Vantaggi del Routing Next.js

- Facilità d'uso
  - Zero configuration
- Flessibilità con supporto dinamico e statico
- Modelli modulari e moderni con la directory **app**



# Domande e Discussione



Grazie per l'attenzione!