NEXT.JS

most import features to know

by Alessio Burbulia
& Francesco Labate

2+ consulting

Che cos’è

NEXT.JS

Perche scegliere

NEXT.JS

Come migliora le performance

NEXT.JS

NEXT.JS

Peculiarita’

Better styling

NEXT.JS

Peculiarita’

Client component

Server component

NEXT.JS

Peculiarita’

‘use server’

‘use client’

‘use cache’

NEXT.JS

Peculiarita’

server action

route API

Page routing

NEXT.JS

Peculiarita’

Static Page

Dynamic page

NEXT.JS

Peculiarita’

SEO

sitemap

manifest

robots

Autenticazione: middleware e NextAuth

Il **Middleware** in Next.js è una funzionalità che ci permette di eseguire logica personalizzata prima che una richiesta raggiunga una rotta di destinazione.

Next Auth è una libreria open source che rappresenta un ottima soluzione per gestire le autenticazioni con Next.js

Grazie a tutti per l’attenzione

NEXT.JS

most import features to know

by Alessio Burbulia
& Francesco Labate

2+ consulting

Next JS v15 features

non tutte ma un bel po’

1. Auto gestione delle **route**
2. Moduli e **CSS &** google font fortemente implementato
3. Gestione **private route**
4. **use client** & **use server**
5. Generazione componenti **HTML statici** e **dinamici**
6. **Gestione CACHE**
7. Gestione **API facilitata**
8. Gestione **AUTH**
9. Gestione della **SEO** & **performance**
10. Gestione **BACKEND**

0

Before starting

0

Configurazione consigliata

Assicurati di aver installato nodejs e npm e poi lancia il comando:

- **npx create-next-app@latest**

Dopo aver dato un nome al tuo progetto ti verrà chiesto quali impostazioni vuoi che vengano utilizzate all'interno della tua applicazione:

```
PS C:\Users\Stage\Desktop> npx create-next-app@latest name-for-your-project
PS C:\Users\Stage\Desktop> npx create-next-app@latest name-for-your-project
Need to install the following packages:
create-next-app@15.1.7
Ok to proceed? (y) y

√ Would you like to use TypeScript? ... No / Yes
√ Would you like to use ESLint? ... No / Yes
√ Would you like to use Tailwind CSS? ... No / Yes
√ Would you like your code inside a `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to use Turbopack for `next dev`? ... No / Yes
√ Would you like to customize the import alias (`@/*` by default)? ... No / Yes
Creating a new Next.js app in C:\Users\Stage\Desktop\name-for-your-project.
```

0

Configurazione consigliata

Turbopack

Builder ottimizzato per JS e TS, velocizza alcuni processi fino a 20 volte piu veloce rispetto a non averlo.

FONTE

Alias import `@/`

le rotte sono dinamicamente gestite anche con la `@` che fara' partire dalla route il path. Perche' dovresti cambiarlo? Not make sense

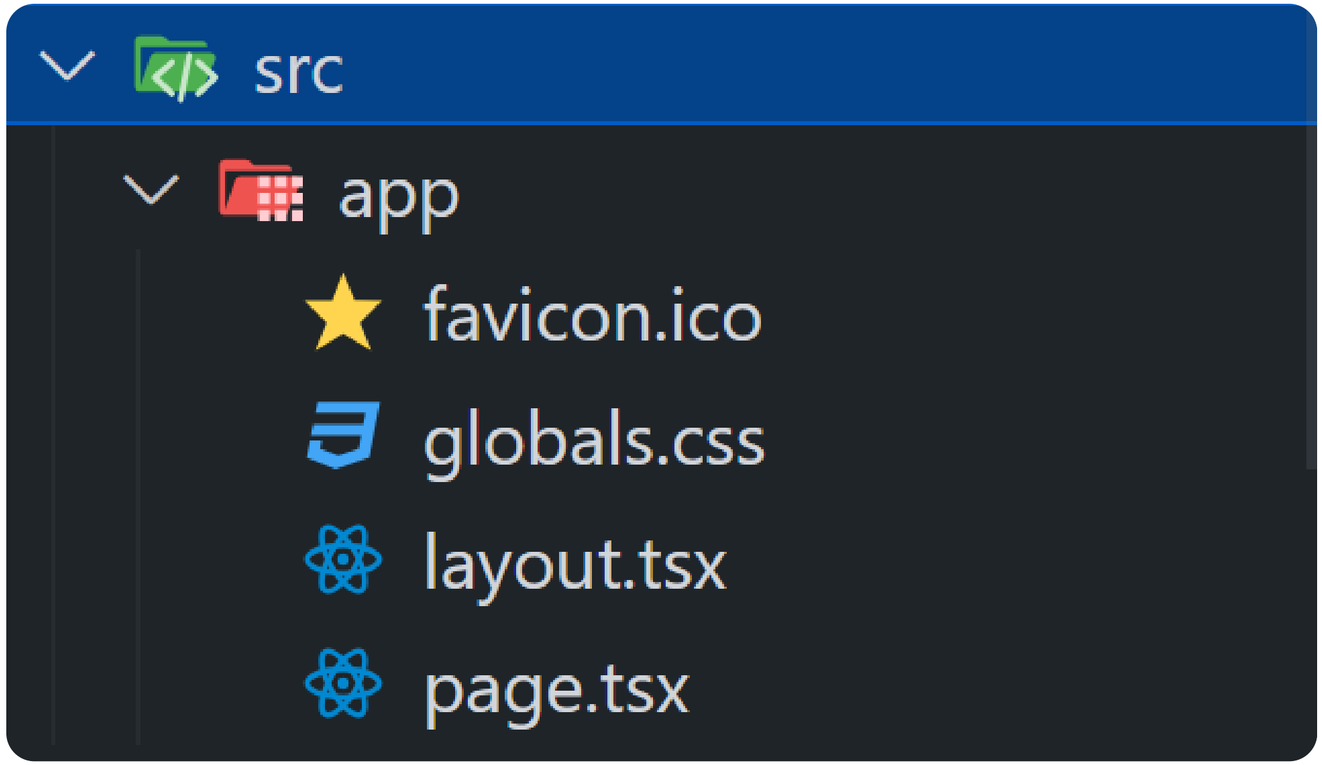
---

SRC

comodo per rendere la applicazione piu' ordinata

0

Ambiente di lavoro

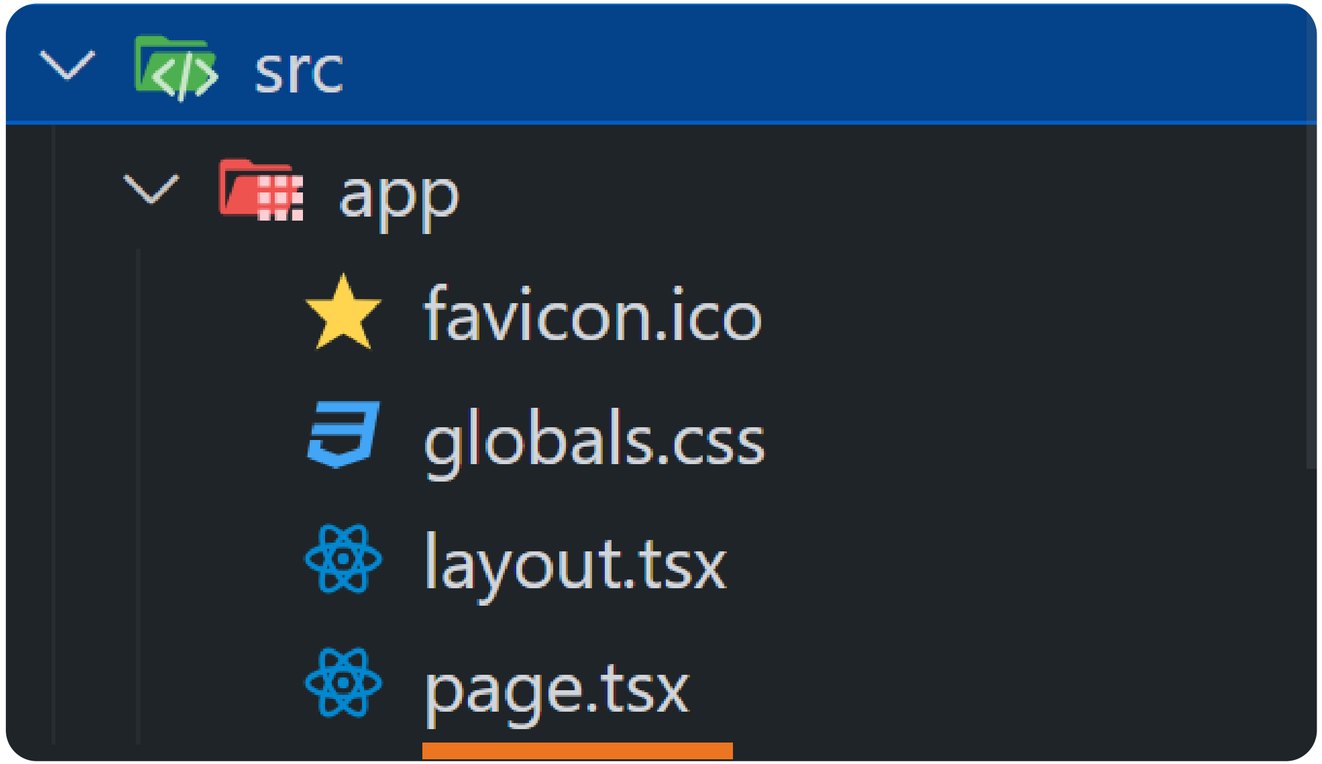


I componenti, le azioni, le chiamate, ecc.. pe ordine: [SRC],

le pagine le inseriamo in: [SRC/APP],

1

Dynamic Route



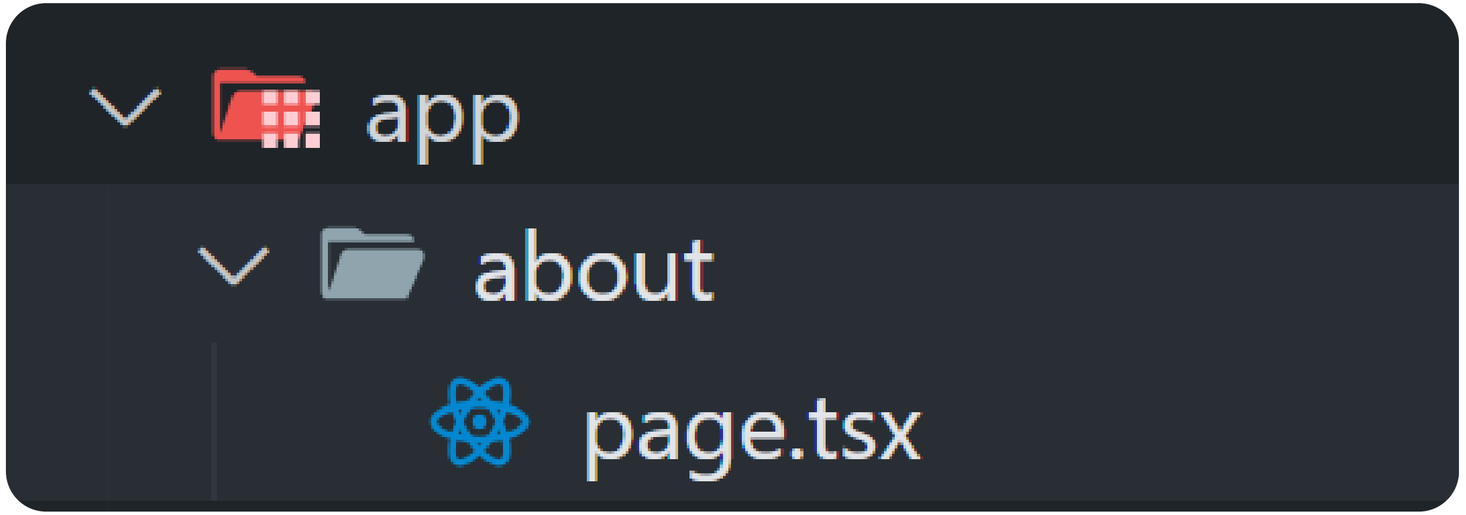
page.tsx
rappresenta la nostra home page [/]

Un' altro modo per gestire la route di home page e' la creazione di una cartella nominata
(route) tra un po' capiremo perche' puo' avere un senso.

1

Dynamic Route

[page.tsx]



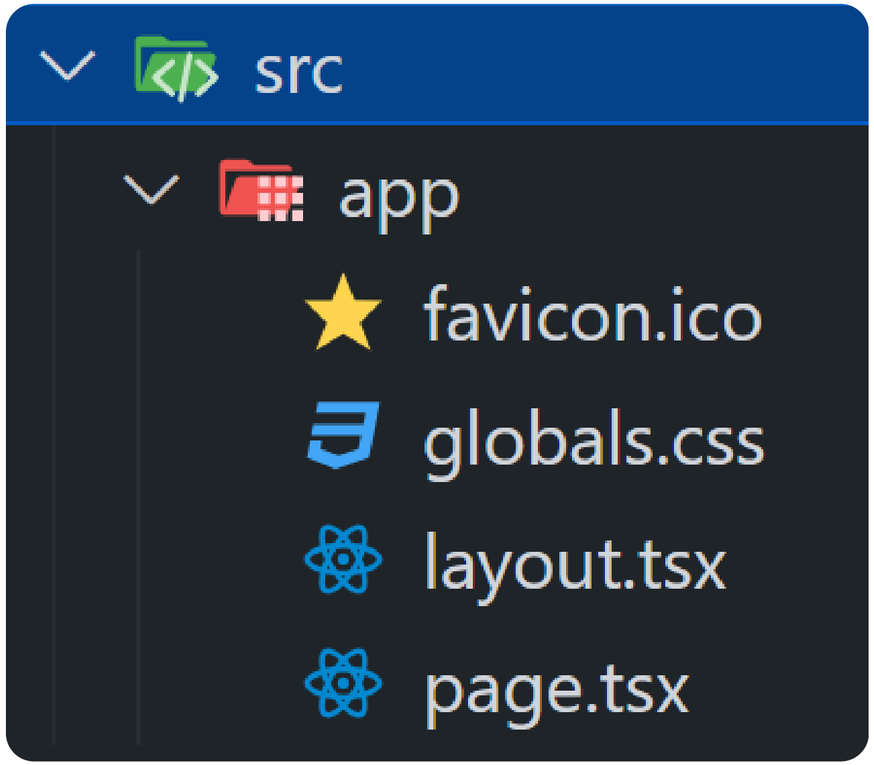
Qualsiasi file chiamato **page.tsx** rappresenta una pagina.

Se voglio creare una nuova rotta statica come [**About**] bisogna creare una cartella di nome about e creare un file page.tsx.

1

Dynamic Route

[layout.tsx]



I file di nome **layout.tsx** rappresentano i layout della pagina, in base alla cartella in cui si trova, se si trova nella route, allora il layout sara' in tutte le pagine, ecco perche' puo ever sesno creare una cartella (route), per separare i layout.

Al suo interno possiamo inserire componenti personalizzati come nell'esempio.

1

Dynamic Route

```
1   export default function RootLayout({
2     children,
3   }: {
4     children: React.ReactNode;
5   }) {
6     return (
7       <html lang="en">
8         <body>{children}</body>
9       </html>
10    );
11  }
12
```

1

Dynamic Route

E’ possibile personalizzare ulteriormente l’esperieza utente utilizzando dei file dai nomi speciali.
Si tratta di componenti (che in base al vostro progetto avranno l’estensione .js o .tsx) che in base al loro nome e alla loro posizione nel codice, nextjs sa già come renderizzare.
Oltre a page e layout troviamo anche :

- **not-found**
- **loading**
- **error**

1

Dynamic Route

NOT-FOUND.tsx

```
export default function Custom404() {
  return (
    <div>
      <h1>Oops! Pagina non trovata.</h1>
      <p>La pagina che stai cercando non è stata trovata.</p>
    </div>
  );
}
```

1

Dynamic Route

LOADING.tsx

```
export default function Loading() {
  return (
    <div>
      <h2>Caricamento dei dati in corso...</h2>
    </div>
  );
}
```

1

Dynamic Route

ERROR.tsx

Il file error.tsx viene usato per gestire gli errori in modo personalizzato. in questo modo è possibile visualizzare messaggi più chiari o gestire errori di rete, richieste API, o errori lato client fornendo all'utente una UI migliore e migliorando di conseguenza la UX.

```
"use client";

import { useEffect } from "react";

export default function Error({ error, reset }) {
  useEffect(() => {
    console.error(error);
  }, [error]);

  return (
    <div>
      <h2>Qualcosa è andato storto..</h2>
      <button
        onClick={
          () => reset()
        }
      >
        Riprova
      </button>
    </div>
  );
}
```

1 File dai nomi speciali:

Nome File	Descrizione:
page.tsx/ page.js	È il file principale per definire una pagina in Next.js
layout.tsx/layout.js	È il file che viene utilizzato per definire una struttura comune a più pagine
error.tsx/error.js	È il file che viene utilizzato per creare messaggi di errore personalizzati nel caso in cui ci siano problemi nel rendering di una pagina.
not-found.tsx/ not-found.js	È il file che viene utilizzato per creare messaggi personalizzati nel caso in cui una pagina non venga trovata (errore 404)
loading.tsx/ loading.js	È il file che viene utilizzato per creare un effetto di caricamento personalizzato.

1

Dynamic Route

[(fonder with ( ))

```
(aboutpages);
```

le cartelle con le parentesi non intaccano l’url, utili per la organizzazione dei file

1

Dynamic Route

[fonder with [ ]]

```
1  [name]
2  [id]
```

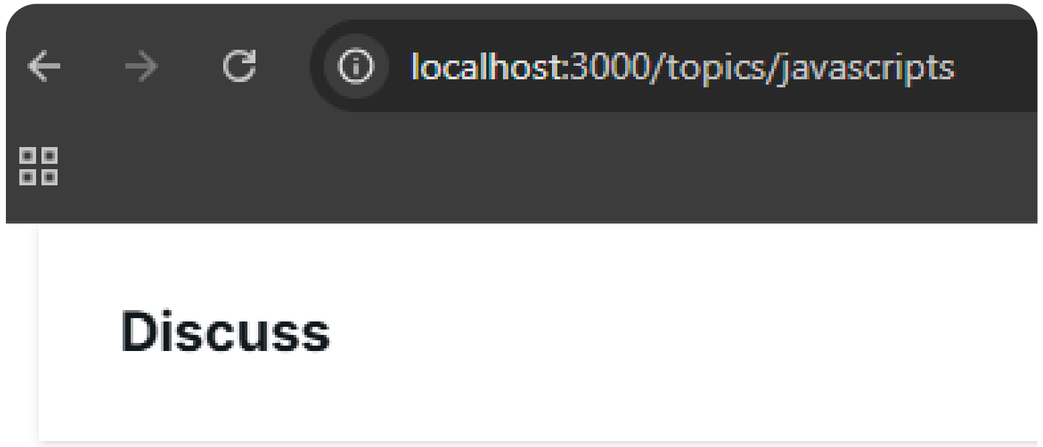
Le cartelle con [ ], avranno la rotta dinamica in base a quello che gli passeremo, nei parametri, come nell’esempio.

1

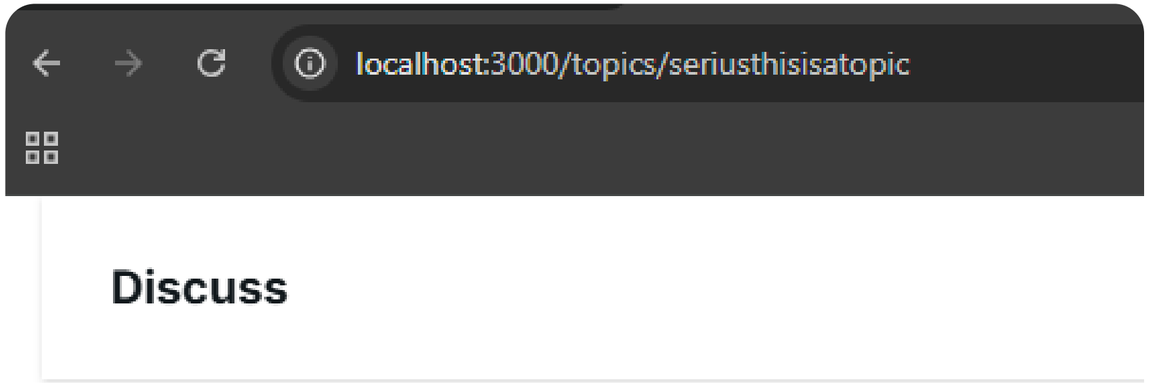
```typescript
interface PostShowPageProps {
  params: Promise<{
    slug: string;
    postId: string;
  }>;
}

export default async function PostShowPage({ params }: PostShowPageProps) {
  const { slug, postId } = await params;
```

Il valore e’ determinato dall’URL che inseriamo , se quello cambia allora, cambia anche il parametro, le rotte possono essere protette con **notFound() nel caso in cui non esistano, io qua non gestisco gli errori**

1



javascripts



seriusthisisatopic

1

utile per

Dare un **impostazione ordinata** e non a scelta del developer per la gestione delle cartelle e dei file.

generare **componenti dinamici** che avranno attributi come valori al suo interno o componenti al quale passarli.

2

Better Stylishing

2

Font Implementation

aggiungi esempio

Puoi implementare dei font con l’uso della libreria di next/font/google

2

Font Implementation

una funzione restituisce il font selezionato con la possibilita' di scegliere sottofont, o lo stile di quest'ultimo

```
export declare function Inter<T extends CssVariable | undefined = undefined>(options?: {
    weight?: '100' | '200' | '300' | '400' | '500' | '600' | '700' | '800' | '900' | 'variable' | Array<'100' | '200' | '300' | '400' | '500' | '600' | '700' | '800' | '900'>;
    style?: 'normal' | 'italic' | Array<'normal' | 'italic'>;
    display?: Display;
    variable?: T;
    preload?: boolean;
    fallback?: string[];
    adjustFontFallback?: boolean;
    subsets?: Array<'cyrillic' | 'cyrillic-ext' | 'greek' | 'greek-ext' | 'latin' | 'latin-ext' | 'vietnamese'>;
    axes?: 'opsz'[];
```

queste sono ad esempio le proprieta' accettate per il font inter

3

gestione Privare Route

3

gestione Privare Route

```
import { NextRequest, NextResponse } from "next/server";

export const MiddleWare = (request: NextRequest) => {
    const isAuth = true; //questa e' la variabile di autenticazione

    if(!isAuth) {
        return NextResponse.next();
    }

    return NextResponse.redirect(new URL('/login?error=Area Privata, effettuare il login', request.url));

}

export default MiddleWare;

export const config = {
    matcher: '/dashboard/:path*',
}
```

per la gestione delle rotte private dovrai creare un file di nome middleware.ts [meglio dentro src] dove specificare come vuoi privatizzare le rotte,

3

gestione Privare Route

Esistono altri modi per gestire le rotte private come farlo direttamente nel component.

Non e’ sempre una best practice, ma ricordati che la gestione della rotta cambia in base al tipo di componente e sopratutto su una caratteristica che vedremo successivamente.

3

gestione Privare Route

Esistono altri modi per gestire le rotte private come farlo direttamente nel component.

Non e’ sempre una best practice, ma ricordati che la gestione della rotta cambia in base al tipo di componente e sopratutto su una caratteristica che vedremo successivamente.

4

Server component

Client Component

4

Server component

viene **renderizzato sul server** anziché nel browser dell'utente.

Efficienza nel Fetching dei Dati: Spostando il fetching dei dati sul server, i Server Components possono accedere direttamente alle risorse senza dover effettuare chiamate API dal client, riducendo così la latenza e migliorando le prestazioni complessive.

Sicurezza Migliorata: Poiché il codice viene eseguito sul server, è possibile mantenere al sicuro informazioni sensibili come chiavi API o token, evitando che vengano esposte nel client.

Riduzione del Carico sul Client: Renderizzando parte dell'interfaccia utente sul server, si diminuisce la quantità di JavaScript che deve essere scaricata ed eseguita nel browser

4

Server component

Limitazioni nell'Interattività: I Server Components non possono gestire l'uso di hook tipo useState o useEffect. Per queste funzionalità, è necessario utilizzare i Client Components eventHandler [onClick - OnPress] .

Possibili Problemi di Performance sul Server: possono aumentare il carico sul server, specialmente in applicazioni con elevato traffico.

Limitazioni nell'Uso di Librerie Client-Side: Alcune librerie o funzionalità che dipendono dall'ambiente del browser potrebbero non funzionare correttamente nei Server Components, limitando le scelte degli sviluppatori.

4

Server component

```
"use server";
export async function createComment(
  { postId, parentId }: { postId: string; parentId?: string },
  formState: CreateCommentFormState,
  formData: FormData
): Promise<CreateCommentFormState> {

    await db.comment.create({
      data: {
        content: result.data.content,
        postId: postId,
        parentId: parentId,
        userId: 'cm6187jc7000gnfzu7p8lwb8u',
      },
    });
```

4

Client component

Interattività: Consentono di gestire stati locali e rispondere agli eventi dell'utente in tempo reale, utilizzando hook come useState o useEffect. nextjs.org

Aggiornamenti in Tempo Reale: Permettono di aggiornare l'interfaccia utente senza dover ricaricare l'intera pagina, offrendo un'esperienza utente più fluida.

4

Client component

Carico Maggiore sul Client: Richiedono che il browser scarichi ed esegua più JavaScript, il che può influire negativamente sulle prestazioni, soprattutto su dispositivi meno potenti o con connessioni lente.

SEO Limitata: Poiché il contenuto viene generato nel browser, i motori di ricerca potrebbero avere difficoltà a indicizzare correttamente le informazioni, influenzando negativamente la visibilità del sito.

4

```
"use client";
export default function CommentCreateForm({
  postId,
  parentId,
  startOpen,
}: CommentCreateFormProps) {
  const [open, setOpen] = useState(startOpen);
  const ref = useRef<HTMLFormElement | null>(null);
  const [formState, action, isPending] = useActionState(
    actions.createComment.bind(null, { postId, parentId }),
    { errors: {} }
  );

  useEffect(() => {
    if (formState.success) {
      ref.current?.reset();

      if (!startOpen) {
        setOpen(false);
      }
    }
  }, [formState, startOpen]);

```

Client component

Chi scegliere?

Usare sempre **server component tranne** dove viene richiesta della interazione con l'utente, come quando:

1. vogliamo gestire un **form**;
2. gestione di **notifiche** in tempo reale;
3. tutto quello che richieda una **interazione** con l'utente in **tempo reale**.

Ricorda

I componenti client non possono compiere azioni sul server come richieste tramite uso di query

5

STATIC AND DYNAMIC pages

Static Page

le pagine statiche sono dei contenuti html, generati dal server, una pagina per essere statica ha bisogno di avere al suo interno esclusivamente contenuti statici .

Per essere statica

non deve essere asincrona, e non deve accettare parametri dinamici, ad esempio un file come [id]/page.tsx non potra’ essere statico ma sara’ per forza dinamico, stessa cosa vale per il rendering dei figli che deve essere di tipo static.

5

Static Page

```
1  export default function Home() {
2    return (
3      <div className="grid grid-cols-4 gap-4 p-4">
4        <div className="col-span-3">
5          <h1 className="text-xl m-2">Top Posts</h1>
6          <PostList fetchData={fetchTopPosts} />
7        </div>
8        <div className="border shadow py-3 px-2">
9          <TopicCreateForm /> {/* client component non asincrono*/}
10         <Divider className="my-2" />
11         <h3 className="text-lg">Topics</h3>
12         <TopicList />
13       </div>
14     </div>
15   );
16 }
17
```

5

Static Page

Prestazioni elevate:
Le pagine statiche vengono servite come file HTML puri. I tempi di risposta rapidissimi, specialmente se ospitate su un CDN, non è necessaria l'elaborazione server per ogni richiesta.

Scalabilità: Possono essere facilmente distribuiti su una rete di distribuzione (CDN) gestendo grandi volumi di traffico senza dover scalare risorse di calcolo lato server.

Sicurezza: Non ci sono esecuzioni di codice che possano essere sfruttate, rendendo l'applicazione meno vulnerabile a certe tipologie di attacchi.

Costo ridotto: L'hosting di pagine statiche generalmente comporta costi inferiori, in quanto richiede meno risorse server e una minore manutenzione.

Affidabilità: Il rischio di errori dovuti a interruzioni dei servizi backend è notevolmente ridotto.

5

Static Page

Aggiornamento dei Dati: Una volta generata, la pagina rimane invariata fino a una nuova build o all'utilizzo di meccanismi è necessario rigenerare la pagina, il che può introdurre latenza o complessità aggiuntive.

Gestione della Complessità per Dati Dinamici:
Quando si ha a che fare con dati che variano molto, mantenere un approccio statico può portare a dover implementare soluzioni ibride o strategie di rigenerazione parziale, aumentando la complessità dell'architettura.

5

Dynamic Page

è una pagina il cui contenuto viene **generato in tempo reale**, spesso in risposta alle interazioni dell'utente o a dati provenienti da un database.

I contenuti nella pagina **necessitano** di un **aggiornamento in diretta** o dell’attesa del caricamento sul browser per funzionare correttamente, come l’attesa dell’URL, quindi molte le **pagine asincrone** saranno anche dinamiche

5

Dynamic Page

Scalabilità: Permettono di gestire un numero elevato di pagine senza doverle creare manualmente.

Aggiornamento dei Contenuti: Consentono di visualizzare contenuti aggiornati in tempo reale, recuperando dati da API o database al momento della richiesta.

5

Dynamic Page

Prestazioni: La generazione dinamica delle pagine può comportare tempi di caricamento leggermente superiori rispetto alle pagine statiche, soprattutto se i dati richiedono tempo per essere recuperati.

SEO: Le pagine generate dinamicamente potrebbero non essere indicizzate immediatamente dai motori di ricerca, a meno che non si utilizzino tecniche come la pre-renderizzazione o l'Incremental Static Regeneration (ISR).

5

Dynamic Page

```
1  export default async function PostShowPage({ params }: PostShowPageProps) {
2    const { slug, postId } = await params;
3    return (
4      <div className="space-y-3">
5        <Link className="underline decoration-solid" href={paths.topicShow(slug)}>
6          {"< "}Back to {slug}
7        </Link>
8        <Suspense fallback={<PostShowLoading />}>
9          <PostShow postId={postId} />
10       </Suspense>
11       <CommentCreateForm postId={postId} startOpen />
12       <CommentList postId={postId} />
13     </div>
14   );
15 }
```

5

Dynamic Page
Incremental Static Regeneration

Rende una pagina potenzialmente dinamica, statica.

Da usare con cautela e con criterio e la Incremental Static Regeneration che non e’ altro che una fetch della pagina dinamica, per ottenere una sua versione statica.

E’ un po’ una forzatura, ma puo’ aiutare in alcuni contesti

5

```
export async function getStaticPaths() {
  const posts = await fetch('https://api.example.com/posts').then(res => res.json());

  const paths = posts.map((post) => ({
    params: { id: post.id.toString() },
  }));

  return {
    paths,
    fallback: 'blocking',
  };
}

export default function Post({ post }) {
  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.content}</p>
    </div>
  );
}
```

pagina statica

5

Come capire se un componente è dinamico o statico

basta runnare un build con il
comando apposito per la creazione
di build
‘**npm run build**’

5

```
Route (app)                                Size     First Load JS
┌ ○ /                                      1.13 kB         414 kB
├ ○ /_not-found                            986 B           107 kB
├ ƒ /api/auth/[...nextauth]                138 B           106 kB
├ ƒ /search                                173 B           110 kB
├ ƒ /topics/[slug]                         850 B           173 kB
├ ƒ /topics/[slug]/posts/[postId]          6.4 kB          419 kB
└ ƒ /topics/[slug]/posts/new               138 B           106 kB
+ First Load JS shared by all              106 kB
  ├ chunks/4bd1b696-582eadacd4e07676.js    53 kB
  ├ chunks/517-e0df4b36c6a4a0df.js         50.7 kB
  └ other shared chunks (total)            2.02 kB
```

Noterai che tutte le pagine con URL dinamico sono anche dinamiche, generalmente è così.

```
○  (Static)   prerendered as static content
ƒ  (Dynamic)  server-rendered on demand
```

6

gestione della cache

6

gestione della cache

una cosa del quale potresti non accorgerti nella fase di developing e l'uso della cache da parte di Next JS, una cosa che invece è importantissima per la nostra applicazione.

Le pagine, una volta renderizzate, si salveranno nella cache per ottenere delle prestazioni migliorate, questo implica però una maggiore complessità nel codice.

6

gestione della cache

Se noi generiamo dei componenti server side e li modifichiamo, ti rederai conto che refreshando la pagina quest’ultima modifica non è visibile.

Non per questo non è avvenuta, semplicemente tu vedrai una “vecchia versione” salvata nella cache.

Esistono diversi modi per rivalidare la path e richiedere la risorsa aggiornata della tua pagina.

6

```
  // let topic: Topic;
  try {
    // topic = await db.topic.create({
    await db.topic.create({
      data: {
        slug: result.data.name,
        description: result.data.description,
      },
    });
  } catch (err: unknown) {
    if (err instanceof Error) {
      return {
        errors: {
          _form: [err.message],
        },
      };
    } else {
      return {
        errors: {
          _form: ["Something went wrong"],
        },
      };
    }
  }

  revalidatePath("/");
  // redirect(paths.topicShow(topic.slug));
  redirect(`/`);
}
```

Revalidate path

funzione che adrà a fare una nuova richiesta di pagina spiecifica per quella path

Si può definire **best practice** e il modo migliore per gestire la rivalidazione del della path

6

time revalidation

di deafault la rivalidazione è settata ogni 365 giorni, ma è modificabile, infatti la cache può essere rivalidata a in tempo a nostra scelta come nell’esempio di codice

```
export async function getStaticProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.id}`);
  const post = await res.json();

  return {
    props: {
      post,
    },
    revalidate: 60, // Rigenera la pagina ogni 60 secondi
  };
}
```

Sconsigliata in generale
non è una buona UX, ma può tornare utile in alcuni contesti

6

gestione della cache

I componenti client possono essere salvati nella cache con una nuova implementazione, ancora non opportunamente testata, ma usufruibile in Next v15.

‘use cache’

Si possono memorizzare nella cache degli elementi , con la ‘use cache’,

Va aggiunta alla configurazione di Next JS nel ifle “next.config.ts”

L’aggiornamento della cache puo’ effettuarsi si in base a un tempo che dando un nome alla chaceTag e

Non e’ possibile verificare le funzioni di cache se non in fase di build infatti e’ ritenuto sperimentale

6

‘use cache’

```
// Utilizzo della direttiva a livello di file
'use cache';

export default function CachedComponent() {
  // Componente cacheabile
}

// Utilizzo della direttiva a livello di funzione
export default function Component() {
  'use cache';
  // Funzione o componente cacheabile
}
```

utililizzabile in maniera ibrida nelle nostre pagine e component,

può essere rivalidata.

6

‘use cache’

```
'use cache';
import { unstable_cacheTag as cacheTag } from 'next/cache';

export default async function getProduct(productId) {
  cacheTag('dettagli-prodotto');
  const res = await fetch(`https://api.example.com/products/${productId}`);
  const product = await res.json();
  return product;
}
```

cacheTag e’ la istanza
della cache.

Ci possono essere piu’ istanze e possoono essere rivalidate con....

6

‘use cache’

```
// app/actions.ts
import { revalidateTag } from 'next/cache';

export async function updateData(newData) {
  // Logica per aggiornare i dati
  await revalidateTag('data-tag');
}
```

revalidateTag();

serve per rivalidare una istanza di una cache.

7

gestione SEO

NextJS e’ ottimizzato per essere facilmente e genericamente letto dai crawler dei motori di ricerca.

Le pagine vengono renderizzate server side e servite successivamente ai browser, quasta operazione aiuta notevolmente la SEO.

7

gestione SEO

La chace assume un ruolo fondamental eper la SEO, le prestazioni, che sono estremamente impattanti per la SEO possono essere notevolmente migliorate con una buona gestione della cache.

7

gestione SEO

MetaTAG

```
// either Static metadata
export const metadata: Metadata = {
  title: '...',
}

// or Dynamic metadata
export async function generateMetadata({ params }) {
  return {
    title: '...',
  }
}
```

I metatag possono essere anche un oggetto esportato statico oppure una funzione per la generazione dinamica e' possibile creare ogni sinogolo metatag possibile, sta poi al developer una buona construzione di quest i ultimi.

gestione SEO

MetaTAG

un esempio puo’ essere passare la query string nelle pagine che andiamo a cercare nella searchBar e perche no? Cacharle in moda che se viene fatta la stessa ricerca, viene renderizzata in diretta.

7

gestione SEO

sitemap.ts

robots.ts

manifest.ts

Si occupano della generazione di file
indispensabili per la SEO,
(Manifest solo nel caso delle PWA)

7

gestione SEO

sitemap.ts

un file che si occupa della generazione
di un file XML che sara’ la nnostra site map.

7

```
// app/sitemap.ts
import { MetadataRoute } from 'next';

export default async function sitemap(): Promise<MetadataRoute.Sitemap> {
  // Recupera i dati dei tuoi articoli o pagine dinamiche
  const articoli = await getArticoli(); // Sostituisci con la tua funzione di recupero dati

  // Crea la lista delle URL
  const urls = [
    {
      url: 'https://www.tuosito.com',
      lastModified: new Date(),
    },
    ...articoli.map((articolo) => ({
      url: `https://www.tuosito.com/articoli/${articolo.slug}`,
      lastModified: articolo.dataModifica,
    })),
  ];

  return urls;
}
```

7

```xml
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>https://www.tuosito.com</loc>
    <lastmod>2025-02-17T11:35:13.000Z</lastmod>
    <changefreq>daily</changefreq>
    <priority>1.0</priority>
  </url>
  <url>
    <loc>https://www.tuosito.com/articoli/articolo-1</loc>
    <lastmod>2025-02-16T10:00:00.000Z</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.8</priority>
  </url>
  <url>
    <loc>https://www.tuosito.com/articoli/articolo-2</loc>
    <lastmod>2025-02-15T09:30:00.000Z</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.8</priority>
  </url>
  <!-- Altre URL generate dinamicamente -->
</urlset>
```

gestione SEO

manifest.ts

Nel caso delle PWA e’ consigliata l’uso delle PWA che descrive meglio alcune caratteristiche della nostra web app.

Verra’ generato un file JSON

7

```typescript
// app/manifest.ts
import { MetadataRoute } from 'next';

export default function manifest(): MetadataRoute.Manifest {
  return {
    short_name: 'MDN',
    name: 'MDN Web Docs',
    icons: [
      {
        src: '/favicon-192x192.png',
        sizes: '192x192',
        type: 'image/png',
      },
      {
        src: '/favicon-512x512.png',
        sizes: '512x512',
        type: 'image/png',
      },
    ],
    start_url: '.',
    display: 'standalone',
    theme_color: '#000000',
    background_color: '#ffffff',
  };
}
```

7

```json
{
  "short_name": "MDN",
  "name": "MDN Web Docs",
  "icons": [
    {
      "src": "/favicon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/favicon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

JSON/LD

per la generazione dei JSON ld non esistono apposite librerie, o oggetti, ma possiamo crearli importando una delle librerie per la costruzione di head, di Next.

7

```
// pages/articolo.js
import { NextSeo } from 'next-seo';

const Articolo = ({ articolo }) => (
  <>
    <NextSeo
      title={articolo.titolo}
      description={articolo.descrizione}
      openGraph={{
        type: 'article',
        article: {
          publishedTime: articolo.dataPubblicazione,
          authors: [articolo.autore],
```

```
      jsonLd={{
        '@context': 'https://schema.org',
        '@type': 'Article',
        headline: articolo.titolo,
        description: articolo.descrizione,
        datePublished: articolo.dataPubblicazione,
        author: {
          '@type': 'Person',
          name: articolo.autore,
        },
        publisher: {
          '@type': 'Organization',
          name: 'Nome del tuo sito',
          logo: {
            '@type': 'ImageObject',
            url: 'https://www.tuosito.com/logo.png',
          },
        },
      }}
    />
```

7

```
// pages/articolo.js
import Head from 'next/head';

const Articolo = ({ articolo }) => (
  <>
    <Head>
```

```
      <script
        type="application/ld+json"
        dangerouslySetInnerHTML={{
          __html: JSON.stringify({
            "@context": "https://schema.org",
            "@type": "Article",
            headline: articolo.titolo,
            description: articolo.descrizione,
            datePublished: articolo.dataPubblicazione,
            author: {
              "@type": "Person",
              name: articolo.autore,
            },
            publisher: {
              "@type": "Organization",
              name: "Nome del tuo sito",
              logo: {
                "@type": "ImageObject",
                url: "https://www.tuosito.com/logo.png",
              },
            },
          }),
        }}
      />
```

7

Performance

per le performance esitono dei componenti ottimizzati, per NextJS che semplificano la vita di un developer

<Image />

```
<Image
src={profilePic}
alt="Foto del profilo"
width={500}
height={500}
priority
style={{ borderRadius: '50%' }}
/>
```

<Image />

Crea un immagine che viene servita al client per essere delle dimensioni massime necessarie, migliorando notevolmente le performance.

Nel caso di compoatibilita’ con il browser (99.99%) produce una immagine di tipo Webp, estensione nata per una migliore compatibilita’ con il mondo web.

<Link />

```
<p>Visita la nostra
  <Link href="/about">
    pagina Chi Siamo
  </Link>
  per saperne di più.</p>
```

<Link />

Da non confendere con il componente Link di REACT, si differisce leggermente in sintassi ma il funzionamento e’ lo stesso, anche esso ottimizzato per SPA e per le applicazioni NEXT.

7

<Suspense />

```
1  import React, { Suspense } from 'react';
2
3  // Un componente che impiega del tempo per caricarsi
4  const MyComponent = React.lazy(() => import('./MyComponent'));
5
6  export default function Page() {
7    return (
8      <Suspense fallback={<div>Loading...</div>}>
9        <MyComponent />
10     </Suspense>
11   );
12 }
13
```

<Suspense />

Componente utile per la gestione di rendering di altri componenti figli, componente fondamentale per la gestione delle performance di una pagina web.

8

Gestione Api facilitata

Gestione Api

Come abbiamo già detto, nextjs è un framework fullstack. Ci permette quindi di lavorare sia con il frontend, sia con il backend. Per quanto riguarda la gestione delle API, nextjs può contare su un suo comodo sistema integrato.

Questa gestione avviene tramite la creazione di uno o più file all’interno del nostro progetto che, una volta creati, svolgeranno diverranno dei veri e propri endpoint implementati nella nostra applicazione.

Gestione Api

Pages router:

Nel pages router, dobbiamo creare un file all'interno della cartella "api", e possiamo chiamarlo come vogliamo esempio **"data.js"**

App router:

All'interno dell'app router invece, possiamo inserire le nostre Api dove vogliamo e non per forza nella cartella Api.
Tuttavia il nome del file dovrà essere "**route.js**"

Gestione Api

Esempio App Router

```
1  export function GET(req) {
2    console.log("Chiamata API");
3    return Response.json({
4      message: "Hello world",
5    });
6  }
```

Gestione Api

Esempio pages router

```
1  import type { NextApiRequest, NextApiResponse } from 'next'
2
3  type ResponseData = {
4    message: string
5  }
6
7  export default function handler(
8    req: NextApiRequest,
9    res: NextApiResponse<ResponseData>
10 ) {
11   res.status(200).json({ message: 'Hello from Next.js!' })
12 }
```

Gestione Api

All’interno di questo nuovo file possiamo creare le nostre funzioni (GET, POST etc.) e potremo quindi usarle nel nostro progetto.

Un processo simile viene fatto con l’uso delle server action, ma ci sono due macro-differenze:
le rotte API sono **pubbliche**, e **serverless**

9

Gestione AUTH

Middleware

Il middleware in Next.js è una funzione che viene eseguita prima che una richiesta raggiunga la logica principale di un'applicazione, agendo come “filtro” aiutando a gestire al meglio le logiche di autorizzazioni e autenticazione. Il middleware può essere usato su più rotte senza dover ripetere il codice in ogni file specifico.

Di solito, il file Middleware.ts o js si trova nella root principale del progetto o all'interno di specifiche cartelle per applicare logiche solo su determinate sezioni dell'app.

9

Middleware

```
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'

// This function can be marked `async` if using `await` inside
export function middleware(request: NextRequest) {
  return NextResponse.redirect(new URL('/home', request.url))
}

// See "Matching Paths" below to learn more
export const config = {
  matcher: '/about/:path*',
}
```

Middleware

WARNING:

Il Middleware è un file globale, e pertanto verrà invocato in OGNI rotta del progetto. E’ possibile tuttavia, configurare il Middleware affinchè agisca solo su una o più pagine (ma non tutte) sfruttando il ***matcher***:

9

Middleware

Matcher:

Rotta singola:

```
1  export const config = {
2    matcher: '/about/:path*',
3  }
```

Rotta multipla:

```
1  export const config = {
2    matcher: ['/about/:path*', '/dashboard/:path*'],
3
4
5  }
```

Middleware

Matcher con le regex:

```
1  export const config = {
2    matcher: [
3      /*
4       * Match all request paths except for the ones starting with:
5       * - api (API routes)
6       * - _next/static (static files)
7       * - _next/image (image optimization files)
8       * - favicon.ico, sitemap.xml, robots.txt (metadata files)
9       */
10     '/((?!api|_next/static|_next/image|favicon.ico|sitemap.xml|robots.txt).*)',
11   ],
12 }
```

9

Middleware

Bypassare il middleware, in base al tipo di richiesta

```
1  export const config = {
2    matcher: [
3      /*
4       * Match all request paths except for the ones starting with:
5       * - api (API routes)
6       * - _next/static (static files)
7       * - _next/image (image optimization files)
8       * - favicon.ico, sitemap.xml, robots.txt (metadata files)
9       */
10     {
11       source:
12         '/((?!api|_next/static|_next/image|favicon.ico|sitemap.xml|robots.txt).*)',
13       missing: [
14         { type: 'header', key: 'next-router-prefetch' },
15         { type: 'header', key: 'purpose', value: 'prefetch' },
16       ],
17     },
18   ],
19 }
```

Next Auth

NextAuth.js è una libreria di Next.js.
E’ estremamente utile nel caso in qui vogliamo implementare in modo semplice delle logiche di autenticazione all’interno del nostro progetto.
La libreria offre un modo facile per gestire la registrazione e l'autenticazione degli utenti, sfruttando le rotte API.
Possiamo installarla dal comando:

- ***npm install next-auth***

Fatto ciò, sarà sufficiente creare all’interno del nostro progetto un file chiamato **[...nextauth].js** in ***pages/api/auth***.

9

Next Auth

```
1  import NextAuth from "next-auth"
2  import GithubProvider from "next-auth/providers/github"
3
4  export const authOptions = {
5    // Configure one or more authentication providers
6    providers: [
7      GithubProvider({
8        clientId: process.env.GITHUB_ID,
9        clientSecret: process.env.GITHUB_SECRET,
10     }),
11     // ...add more providers here
12   ],
13 }
14
15 export default NextAuth(authOptions)
```

Next Auth

Tutte le future richieste a api/auth (signIn, callback, signOut, etc.) saranno automaticamente gestite da nextAuth.

Per poterlo utilizzare, questa nuova autenticazione, dovrai innanzitutto importare il “session provider all’interno della tua applicazione:

Warning!!
Se stai utilizzando l’app router ricorda che il session provider può essere usato SOLO nei client-component

9

Next Auth

```
1   import { SessionProvider } from "next-auth/react"
2
3   export default function App({
4     Component,
5     pageProps: { session, ...pageProps },
6   }) {
7     return (
8       <SessionProvider session={session}>
9         <Component {...pageProps} />
10      </SessionProvider>
11    )
12  }
```

Next Auth

Se vogliamo fare una verifica o un controllo nel caso in cui qualcuno abbia fatto l'accesso, lato frontend, il metodo più semplice è utilizzare l'hook **usession()**:

Di seguito, un esempio all'interno di una possibile rotta:

"**component/login-btn.jsx"**

9

Next Auth

```
1  import { useSession, signIn, signOut } from "next-auth/react"
2  export default function Component() {
3    const { data: session } = useSession()
4    if (session) {
5      return (
6        <>
7          Signed in as {session.user.email} <br />
8          <button onClick={() => signOut()}>Sign out</button>
9        </>
10     )
11   }
12   return (
13     <>
14       Not signed in <br />
15       <button onClick={() => signIn()}>Sign in</button>
16     </>
17   )
18 }
```

Next Auth

lato backend invece, possiamo utilizzare il metodo **getServerSession()** per proteggere le rotte API e impedire l'accesso a persone non autorizzate:
Di seguito, un possibile esempio di una pagina "**pages/api/restricted.js**"

9

Next Auth

```
1  import { getServerSession } from "next-auth/next"
2  import { authOptions } from "./auth/[...nextauth]"
3  export default async (req, res) => {
4    const session = await getServerSession(req, res, authOptions)
5    if (session) {
6      res.send({
7        content:
8          "This is protected content. You can access this content because you are signed in.",
9      })
10   } else {
11     res.send({
12       error: "You must be signed in to view the protected content on this page.",
13     })
14   }
15 }
```

Next Auth

per la gestione tramite credenziali e’ cosigliata la visione della docuementazione ufficiale

NextAuth

NextAuthCallback

sign in prove

Next Auth

Caratteristiche principali:

- **Sicurezza elevata:** Fornisce una gestione robusta delle sessioni, crittografia dei cookie e supporto per i moderni standard di sicurezza come l'autenticazione basata su JWT e OAuth2.
- **Scalabilità**: Può supportare sia piccole applicazioni che progetti di grandi dimensioni.
- **Integrazione semplice**: Offre supporto per l'interazione con diversi database come MongoDB, MySQL, PostgreSQL
- **Supporto**: Auth.js offre una vasta gamma di provider di autenticazione integrati, come Google, Facebook, Twitter, GitHub etc.

10

Gestione Backend

10

Gestione Backend

La gestione del backend in Next.js è molto semplice e integrata, grazie alla sua struttura ibrida che permette di gestire sia il front-end che il back-end all'interno dello stesso progetto. In Next.js, il backend viene gestito principalmente tramite le API routes, ma è possibile usare anche altre logiche come le server actions.

Gestione Backend

Server actions:

Le Server Actions sono una funzionalità introdotta per semplificare l'esecuzione di codice sul server direttamente all'interno delle pagine, senza l’ausilio delle routes API.

Ci permettono quindi di scrivere funzioni , che possiamo richiamare quando ne abbiamo bisogno (esempio il submit di un form), senza fare chiamate HTTP. Quando invocata la funzione, il codice viene eseguito sul server (lato backend), invece che sul client.

Questo significa che puoi eseguire operazioni più complesse, come l'accesso a un database senza dover configurare manualmente una route API separata.

10

Gestione Backend

```
1  "use server";
2
3  import { db } from "@/db";
4  import { redirect } from "next/navigation";
5  export async function editSnippet(id: number, code: string) {
6    await db.snippet.update({
7      where: { id },
8      data: { code },
9    });
10   redirect(`/snippets/${id}`);
11 }
12
13 export async function deleteSnippet(id: number) {
14   await db.snippet.delete({
15     where: { id },
16   });
17   redirect("/");
18 }
19
```

Gestione Backend

Server actions:

Nell’esempio abbiamo un esempio concreto di quello che sono le ***server-action:***

A livello della root principale ho creato una cartella “actions” e al suo interno un file “index.ts”.

Qui, con l’uso della direttiva “***use server***” (perchè voglio che queste operazioni siano private, e che vengano eseguite **solo** sul server). Ho impostanto il collegameno con un db e inserito la mia logica.
In questo caso, funzioni che editano e cancellano file da un db.

Gestione Backend

Interazione con il db

Per gestire il backend in nextjs, spesso è necessario interagire con un database (ad esempio, MongoDB, PostgreSQL, MySQL, ecc.).

Questa interazione è molto facilitata, grazie all'uso delle rotte Api , delle server actions, ma anche dalla compatibilità di next con i db, e librerie per la gestione di questi db.
Un esempio può essere la libreria ***PRISMA***:
La sua configurazione permette di creare facilmente un modello per il db e fare operazioni CRUD.