> Menu

App Router > ... > Components > <Form>

<Form>

This API is currently in the canary channel and not yet available in a stable version.

The `<Form>` component extends the HTML `<form>` element to provide **prefetching** of loading UI, **client-side navigation** on submission, and **progressive enhancement**.

It's useful for forms that update URL search params as it reduces the boilerplate code needed to achieve the above.

Basic usage:

/app/page.tsx TypeScript

```tsx
import Form from 'next/form'

export default function Page() {
  return (
    <Form action="/search">
      {/* On submission, the input value will be appended to
          the URL, e.g. /search?query=abc */}
      <input name="query" />
      <button type="submit">Submit</button>
    </Form>
  )
}
```

Reference

The behavior of the `<Form>` component depends on whether the `action` prop is passed a `string` or `function`.

- When `action` is a **string**, the `<Form>` behaves like a native HTML form that uses a **`GET`** method. The form data is encoded into the URL as search params, and when the form is submitted, it navigates to the specified URL. In addition, Next.js:
  - Prefetches the path when the form becomes visible, this preloads shared UI (e.g. `layout.js` and `loading.js`), resulting in faster navigation.
  - Performs a client-side navigation instead of a full page reload when the form is submitted. This retains shared UI and client-side state.
- When `action` is a **function** (Server Action), `<Form>` behaves like a React form ↗, executing the action when the form is submitted.

`action` (string) Props

When `action` is a string, the `<Form>` component supports the following props:

Prop	Example	Type	Required
`action`	`action="/search"`	`string` (URL or relative path)	Yes
`replace`	`replace={false}`	`boolean`	-
`scroll`	`scroll={true}`	`boolean`	-

- **`action`**: The URL or path to navigate to when the form is submitted.
  - An empty string `""` will navigate to the same route with updated search params.
- **`scroll`**: Controls the scroll behavior during navigation. Defaults to `true`, this means it will scroll to the top of the new route, and maintain the scroll position for backwards and forwards navigation.

- `replace`: Replaces the current history state instead of pushing a new one to the browser's history ↗ stack. Default is `false`.

`action` (function) Props

When `action` is a function, the `<Form>` component supports the following prop:

Prop	Example	Type	Required
`action`	`action={myAction}`	`function` (Server Action)	Yes

- `action`: The Server Action to be called when the form is submitted. See the React docs ↗ for more.

> **Good to know**: When `action` is a function, the `replace` and `scroll` props are ignored.

Caveats

- `onSubmit`: Can be used to handle form submission logic. However, calling `event.preventDefault()` will override `<Form>` behavior such as navigating to the specified URL.
- `formAction`: Can be used in a `<button>` or `<input type="submit">` fields to override the `action` prop. Next.js will perform a client-side navigation, however, this approach doesn't support prefetching.
  - When using `basePath`, you must also include it in the `formAction` path. e.g. `formAction="/base-path/search"`.
- `method` ↗, `encType` ↗, `target` ↗: Are not supported as they override `<Form>` behavior.
  - Similarly, `formMethod`, `formEncType`, and `formTarget` can be used to override the `method`, `encType`, and `target` props respectively, and using them will fallback to native browser behavior.
  - If you need to use these props, use the HTML `<form>` element instead.

- `key`: Passing a `key` prop to a string `action` is not supported. If you'd like to trigger a re-render or perform a mutation, consider using a function `action` instead.
- `<input type="file">`: Using this input type when the `action` is a string will match browser behavior by submitting the filename instead of the file object.

Examples

Search form that leads to a search result page

You can create a search form that navigates to a search results page by passing the path as an `action`:

/app/page.tsx TypeScript

```
import Form from 'next/form'

export default function Page() {
  return (
    <Form action="/search">
      <input name="query" />
      <button type="submit">Submit</button>
    </Form>
  )
}
```

When the user updates the query input field and submits the form, the form data will be encoded into the URL as search params, e.g. `/search?query=abc`.

> **Good to know**: If you pass an empty string `""` to `action`, the form will navigate to the same route with updated search params.

On the results page, you can access the query using the `searchParams` `page.js` prop and use it to fetch data from an external source.

/app/search/page.tsx TypeScript

```
import { getSearchResults } from '@/lib/search'

export default async function SearchPage({
  searchParams: { [key: string]: string | string[] | undefined }
}) {
  const results = await getSearchResults(searchParams.query)

  return <div>...</div>
}
```

When the `<Form>` becomes visible in the user's viewport, shared UI (such as `layout.js` and `loading.js`) on the `/search` page will be prefetched. On submission, the form will immediately navigate to the new route and show loading UI while the results are being fetched. You can design the fallback UI using `loading.js`:

/app/search/loading.tsx TypeScript

```
export default function Loading() {
  return <div>Loading...</div>
}
```

To cover cases when shared UI hasn't yet loaded, you can show instant feedback to the user using `useFormStatus`↗.

First, create a component that displays a loading state when the form is pending:

/app/ui/search-button.tsx TypeScript

```tsx
'use client'
import { useFormStatus } from 'react-dom'

export default function SearchButton() {
  const status = useFormStatus()
  return (
    <button type="submit">{status.pending ? 'Searching...' : 'Search'}</button>
  )
}
```

Then, update the search form page to use the `SearchButton` component:

/app/page.tsx TypeScript

```tsx
import Form from 'next/form'
import { SearchButton } from '@/ui/search-button'

export default function Page() {
  return (
    <Form action="/search">
      <input name="query" />
      <SearchButton />
    </Form>
  )
}
```

Mutations with Server Actions

You can perform mutations by passing a function to the `action` prop.

/app/posts/create/page.tsx TypeScript

```tsx
import Form from 'next/form'
import { createPost } from '@/posts/actions'

export default function Page() {
  return (
    <Form action={createPost}>
      <input name="title" />
      {/* ... */}
      <button type="submit">Create Post</button>
    </Form>
  )
}
```

After a mutation, it's common to redirect to the new resource. You can use the `redirect` function from `next/navigation` to navigate to the new post page.

> **Good to know**: Since the "destination" of the form submission is not known until the action is executed, `<Form>` cannot automatically prefetch shared UI.

/app/posts/actions.ts TypeScript

```ts
'use server'
import { redirect } from 'next/navigation'

export async function createPost(formData: FormData) {
  // Create a new post
  // ...

  // Redirect to the new post
  redirect(`/posts/${data.id}`)
}
```

Then, in the new page, you can fetch data using the `params` prop:

/app/posts/[id]/page.tsx TypeScript

```tsx
import { getPost } from '@/posts/data'

export default async function PostPage({ params }: { params: { id: string } }) {
  const data = await getPost(params.id)

  return (
    <div>
      <h1>{data.title}</h1>
      {/* ... */}
    </div>
  )
}
```

See the Server Actions docs for more examples.

Previous
Font

Next
<Image>

Was this helpful?

Vercel

Resources
Docs
Learn
Showcase
Blog
Analytics
Next.js Conf
Previews

More
Next.js Commerce
Contact Sales
GitHub
Releases
Telemetry
Governance

About Vercel
Next.js + Vercel
Open Source Software
GitHub
X

Legal
Privacy Policy

Subscribe to our newsletter

Stay updated on new releases and features, guides, and case studies.

you@domain.com Subscribe