Start and Route

Asst.Prof.Drusawin Vongpramate
Department of Information Technology
Faculty of Science, BRU

Hint

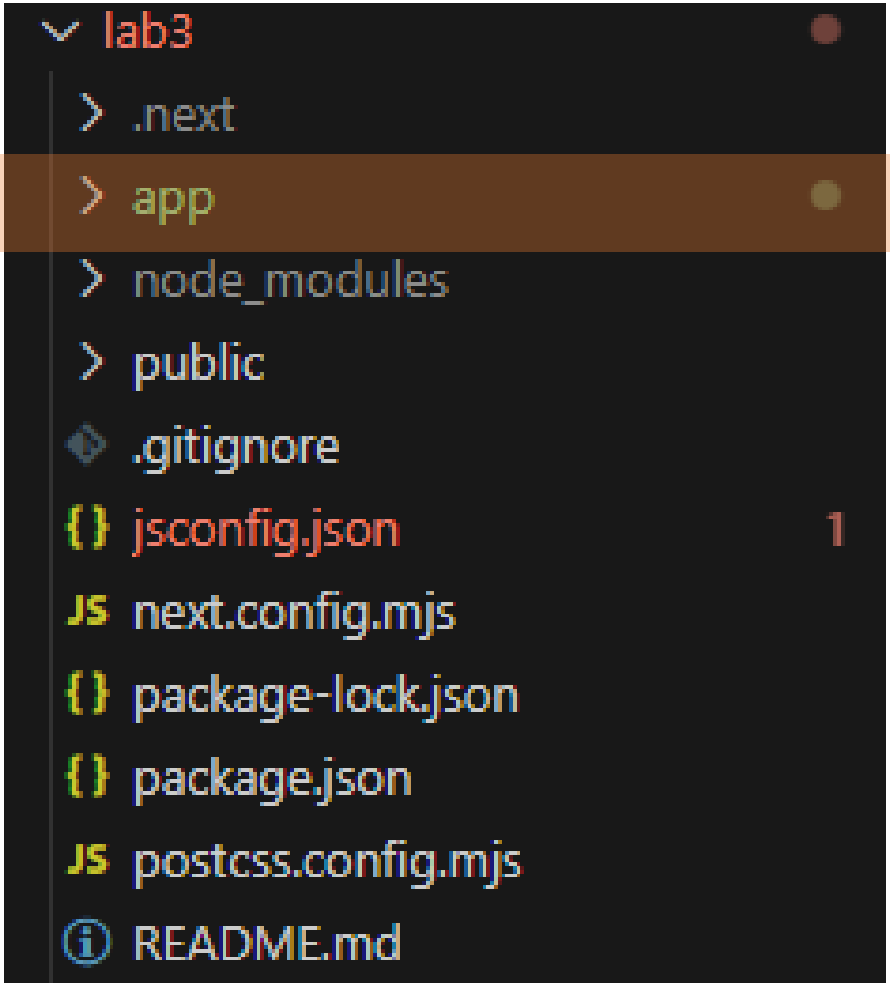
Main Topic Sub Topic

Installation

- **System Requirements:**
- Node.js 18.17 or later.
- macOS, Windows (including WSL), and Linux are supported.

> npx create-next-app@latest

Structure



Application

- **Frontend**
- **Backend**

package.json

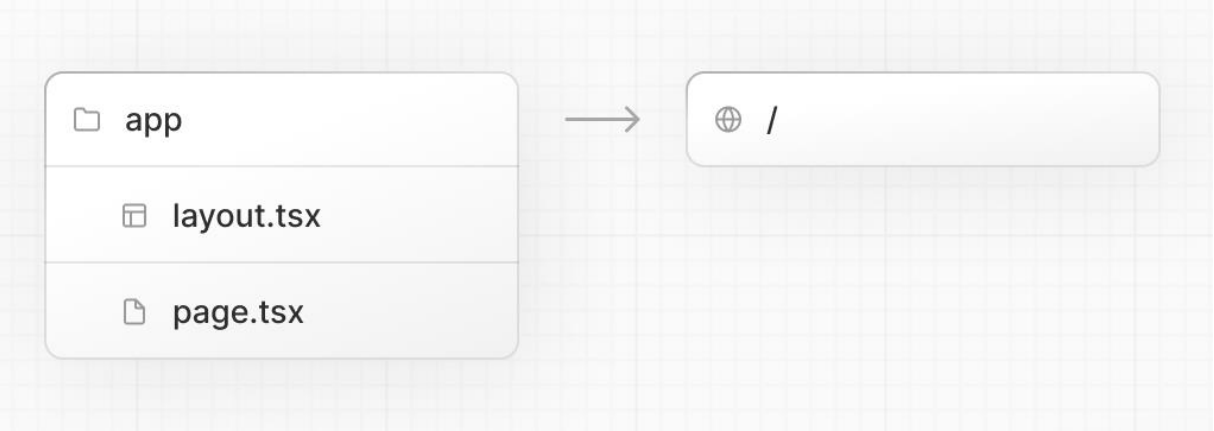
- **dev**: runs next dev to start Next.js in development mode.
- **build**: runs next build to build the application for production usage.
- **start**: runs next start to start a Next.js production server.
- **lint**: runs next lint to set up Next.js' built-in ESLint configuration.

Layout

- Create a root layout inside app/layout.js

```
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
        <div>Header</div>
        {children}
        <div>Footer</div>
      </body>
    </html>
  );
}
```

Directory



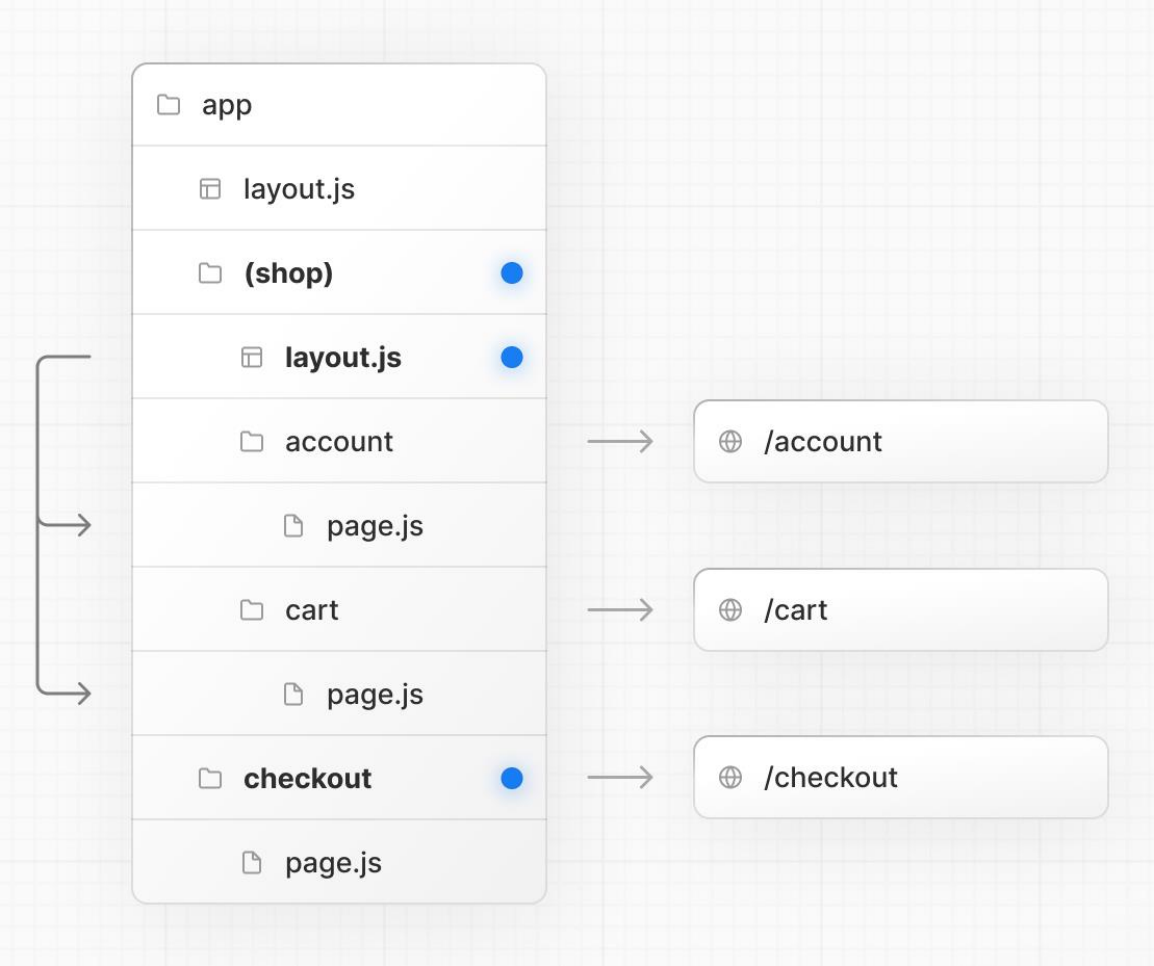
Nested Routes

`folder`	Route segment
`folder/folder`	Nested route segment

Dynamic Routes

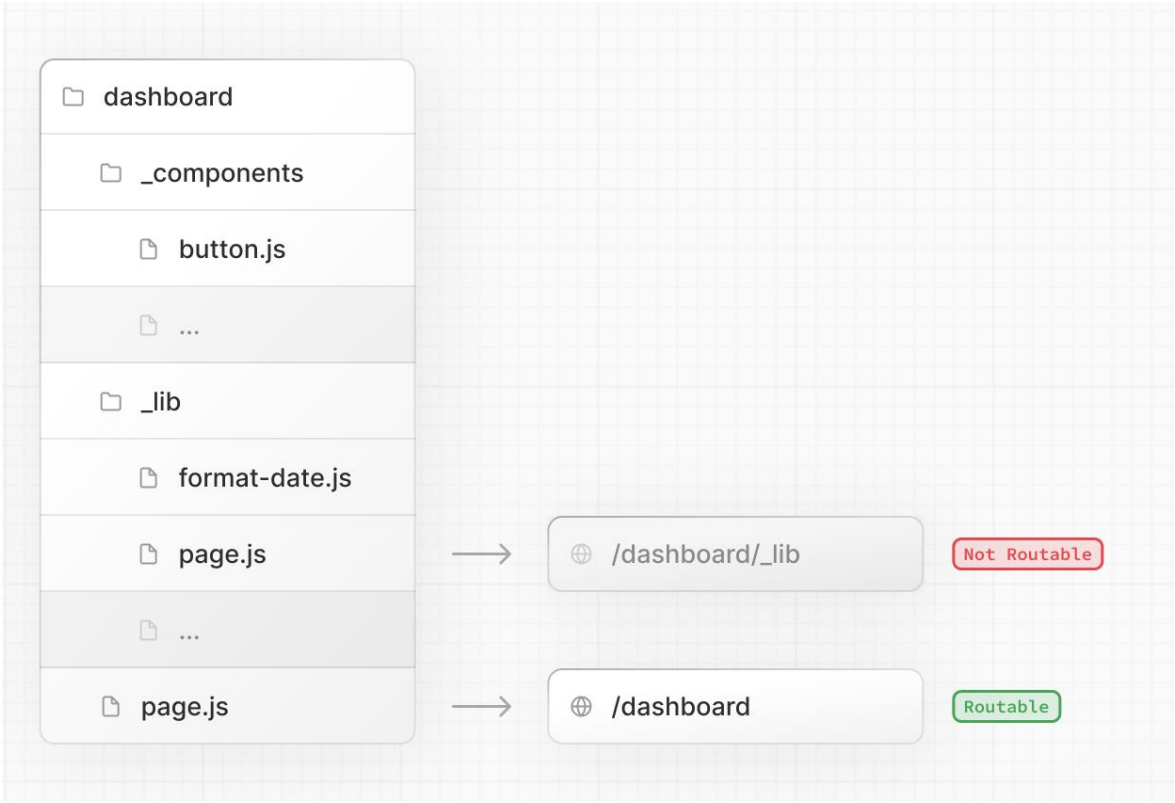
Folder convention				
`[folder]/index`	`.js`	`.jsx`	`.tsx`	Dynamic route segment
`[...folder]/index`	`.js`	`.jsx`	`.tsx`	Catch-all route segment

Route Groups



To opt specific routes into a layout, create a new route group (e.g. (shop)) and move the routes that **share** the same layout into the group (e.g. account and cart). The routes outside of the group will **not share** the layout (e.g. checkout).

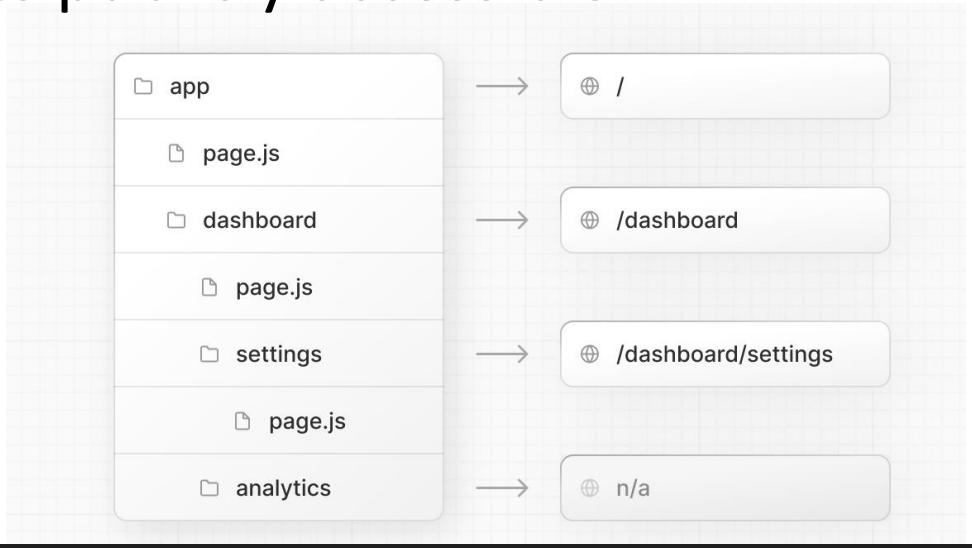
Private Folders



Private folders can be created by prefixing a folder with an underscore: **_folderName**

Page

- A special page.js file is used to make route segments publicly accessible.




```
1 export default function Home() {
2   return <div>Home</div>;
3 }
4
```

Import

testBtn.js same directory

```
1 export default  function testBtn() {
2     return(
3         <button>Test</button>
4     )
5 }
```

plus.js in **lib** directory

```
1 export default function Plus(a, b) {
2   return a + b;
3 }
4
```

Import

page.js same testBtn.js directory

```
1  import Plus from '../../lib/plus'
2  import TestBtn from './testBtn';
3
4  export default function Page() {
5
6      console.log(Plus(1,2));
7
8      return (
9        <div>
10         <TestBtn/>
11        {Plus(1,2)}
12       </div>
13     );
14   }
```

Dynamic Routes

A Dynamic Segment can be created by wrapping a folder's name in square brackets: [folderName]. For example, [id] or [slug].

Convention

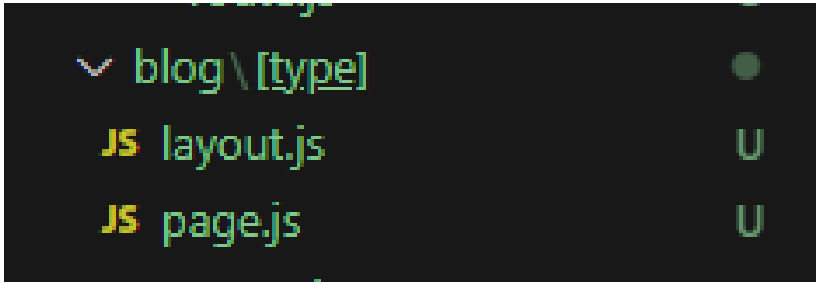
Route	Example URL	params
`app/blog/[slug]/page.js`	`/blog/a`	`{ slug: 'a' }`
`app/blog/[slug]/page.js`	`/blog/b`	`{ slug: 'b' }`

Catch-all

Route	Example URL	params
`app/shop/[...slug]/page.js`	`/shop/a`	`{ slug: ['a'] }`
`app/shop/[...slug]/page.js`	`/shop/a/b`	`{ slug: ['a', 'b'] }`
`app/shop/[...slug]/page.js`	`/shop/a/b/c`	`{ slug: ['a', 'b', 'c'] }`

Dynamic Routes

Create [folderName] e.g. [type]

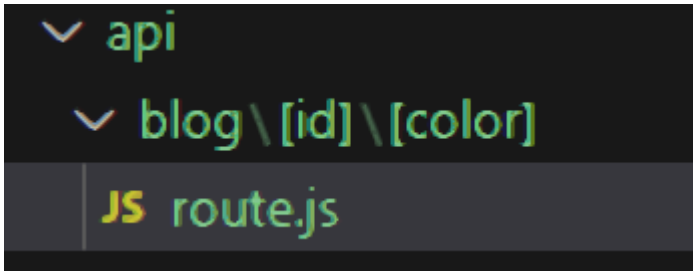


page.js

```
1 export default  function Page({ params }) {
2     return (
3       <div>
4         blog @{params.type}
5       </div>
6     );
7   }
```

Route

Create [folderName] e.g. [id][color]



route.js

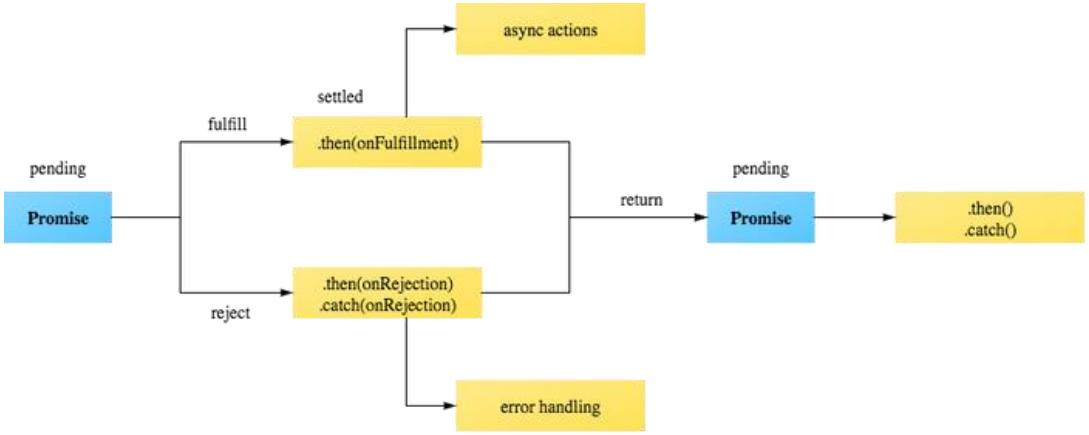
```
1 export function GET(request, { params }) {
2   console.log("blog");
3   return Response.json([{
4     name: "Drusawin",
5     id: params.id,
6     color: params.color,
7   }]);
8 }
9
```

fetch

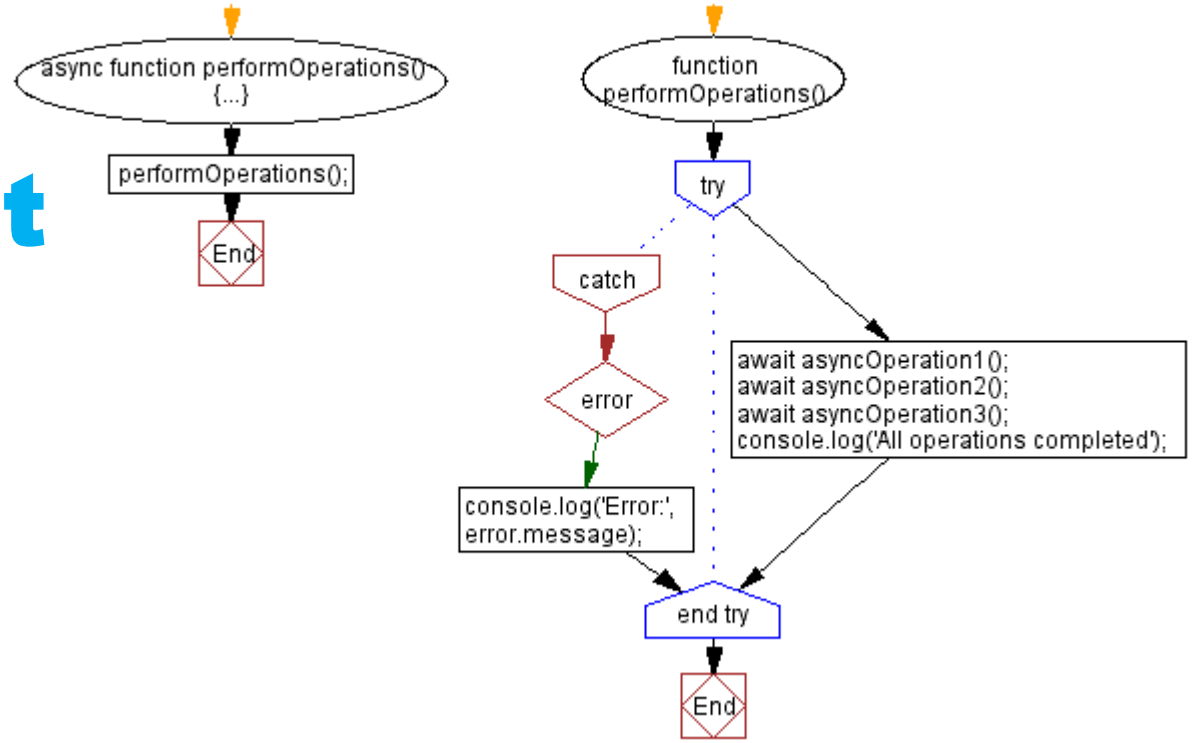
page.js

```
1  async function getBlogs() {
2    const host = "http://127.0.0.1:3000";
3    const response = await fetch(`${host}/api/blog/1/red`);
4
5    if (!response.ok) throw new Error("can not fetch blog");
6
7    return response.json();
8  }
9
10 export default async function Page() {
11   const blogs = await getBlogs();
12   console.log(blogs);
13
14   return (
15     <div>
16       blog
17       {blogs.color}
18       {blogs.id}
19       {blogs.name}
20     </div>
21   );
22 }
```

Promises



Async/Await



Middleware

Integrating Middleware into your application can lead to significant improvements in performance, security, and user experience. Some common scenarios where Middleware is particularly effective include:

- Authentication and Authorization
- Server-Side Redirects
- Path Rewriting
- Bot Detection
- Logging and Analytics
- Feature Flagging

Middleware

middleware.js in root application directory

```
1  import { NextResponse } from "next/server";
2
3  // This function can be marked `async` if using `await` inside
4  export function middleware(request) {
5    //   return NextResponse.redirect(new URL('/', request.url))
6    return NextResponse.next();
7  }
8
9  // See "Matching Paths" below to learn more
10 export const config = {
11   matcher: ["/about/:path*", "/dashboard/:path*"],
12 };
13
```

CORS

You can set CORS headers in Middleware to allow cross-origin requests, including **simple** and **preflighted** requests.

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

const allowedOrigins = ['https://test.com', 'https://bru.ac.th']

const corsOptions = {
  'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE, OPTIONS',
  'Access-Control-Allow-Headers': 'Content-Type, Authorization',
}

export function middleware(request) {
  // Check the origin from the request
  const origin = request.headers.get('origin') ?? ''
  const isAllowedOrigin = allowedOrigins.includes(origin)

  // Handle preflighted requests
  const isPreflight = request.method === 'OPTIONS'

  if (isPreflight) {
    const preflightHeaders = {
      ...(isAllowedOrigin && { 'Access-Control-Allow-Origin': origin }),
      ...corsOptions,
    }
    return NextResponse.json({}, { headers: preflightHeaders })
  }

  // Handle simple requests
  const response = NextResponse.next()

  if (isAllowedOrigin) {
    response.headers.set('Access-Control-Allow-Origin', origin)
  }

  Object.entries(corsOptions).forEach(([key, value]) => {
    response.headers.set(key, value)
  })

  return response
}

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

Q & A