some

August 18, 2026

some

Kind: Function

Source: `src/middleware/combine/index.ts`

Part of: Middleware

Create a composed middleware that runs the first middleware that returns true.

`some` creates a middleware composition that evaluates child middleware in order. It stops when a middleware returns `true`, allowing matching middleware to handle the current context without continuing to later handlers.

Signature

```
function some(middleware: (MiddlewareHandler | Condition)[]): MiddlewareHandler
```

Parameters

Name	Type
`middleware`	`(MiddlewareHandler

Returns: `MiddlewareHandler`

Diagram

```
graph LR
  Context[Context] --> First[First middleware]
  First -->|false| Next[Next middleware]
  First -->|true| Handled[Stop composition]
  Next -->|false| Remaining[Remaining middleware]
  Next -->|true| Handled
```

```
  Remaining -->|true| Handled
  Remaining -->|false| Unhandled[No middleware handled the context]
```

Usage

```ts
import { some } from "./middleware/combine";

type RequestContext = {
  path: string;
  response?: string;
};

const handleHealthCheck = async (ctx: RequestContext) => {
  if (ctx.path !== "/health") return false;

  ctx.response = "ok";
  return true;
};

const handleNotFound = async (ctx: RequestContext) => {
  ctx.response = "not found";
  return true;
};

const handleRequest = some(handleHealthCheck, handleNotFound);

const ctx: RequestContext = { path: "/health" };

await handleRequest(ctx);

console.log(ctx.response); // "ok"
```

AI Coding Instructions

- Keep middleware ordered from the most specific match to the fallback handler.
- Return `true` only when the middleware has handled the context and later middleware should not run.
- Return `false` when the middleware does not match so `some` can continue evaluating handlers.
- Add fallback middleware last when unmatched contexts need a default response.

Relationships

- IMPORTS → `compose`

- IMPORTS → `METHOD_NAME_ALL`