RouterResponseController

August 18, 2026

RouterResponseController

Kind: Class

Source: `packages/core/router/router-response-controller.ts`

Part of: Core

`RouterResponseController` centralizes how route handler results are converted into HTTP responses. It manages status codes, headers, redirects, rendered views, server-sent events, and final response application so router behavior remains consistent across handlers.

Methods

Method	Signature	Returns
`apply`	`apply(result: TInput, response: TResponse, httpStatusCode: number)`	`void`
`redirect`	`redirect(resultOrDeferred: TInput, response: TResponse, redirectResponse: RedirectResponse)`	`void`
`render`	`render(resultOrDeferred: TInput, response: TResponse, template: string)`	`void`
`transformToResult`	`transformToResult(resultOrDeferred: any)`	`void`
`getStatusByMethod`	`getStatusByMethod(requestMethod: RequestMethod)`	`number`
`setHeaders`	`setHeaders(response: TResponse, headers: CustomHeader[])`	`void`
`setStatus`	`setStatus(response: TResponse, statusCode: number)`	`void`

Method	Signature	Returns
`sse`	`sse(result: TInput	Promise, response: TResponse, request: TRequest, options: { additionalHeaders?: AdditionalHeaders; statusCode?: number; })`

Where it refuses work

- `RouterResponseController` stops the work with `ReferenceError` when `!isObservable(value)` — “You must return an Observable stream to use Server-Sent Events (SSE).”.
- `RouterResponseController` stops the work with an early return when `settled`, in 4 places.
- `RouterResponseController` stops the work with an early return when `isObservable(resultOrDeferred)`.
- `RouterResponseController` stops the work with an early return when `response.writableEnded`.
- `RouterResponseController` stops the work with an early return when `settled || closeRequested`.
- `RouterResponseController` stops the work with an early return when `isObject(message)`.

Diagram

```
graph LR
  Handler[Route Handler] --> Controller[RouterResponseController]
  Controller --> Status[setStatus / getStatusByMethod]
  Controller --> Headers[setHeaders]
  Controller --> Result[transformToResult]
  Result --> Apply[apply]
  Controller --> Redirect[redirect]
  Controller --> Render[render]
  Controller --> SSE[sse]
  Apply --> Response[HTTP Response]
  Redirect --> Response
  Render --> Response
  SSE --> Response
```

Usage

```
import { RouterResponseController } from '@your-package/core/router';

// The router typically creates and provides the controller to handler code.
function createUser(
  responseController: RouterResponseController,
  user: { id: string; email: string },
) {
  responseController.setStatus(201);
  responseController.setHeaders({
    'content-type': 'application/json',
    'x-resource-created': 'user',
  });

  return responseController.apply({
    data: user,
  });
}

// Other response patterns:
// responseController.redirect('/sign-in');
// responseController.render('profile', { user });
// responseController.sse(eventStream);
```

AI Coding Instructions

- Use `setStatus()` and `setHeaders()` before calling `apply()` so response metadata is finalized with the result.
- Prefer `redirect()`, `render()`, and `sse()` for their dedicated response types instead of manually constructing equivalent raw HTTP responses.
- Let `getStatusByMethod()` provide method-aware defaults when no explicit status code is required.
- Do not write directly to the underlying HTTP response after `apply()`, `redirect()`, `render()`, or `sse()` has finalized it.
- Keep route handlers focused on producing data; rely on `transformToResult()` and `apply()` to normalize handler output into router-compatible responses.

How it works

`RouterResponseController` is a router-layer class that applies a route handler's result to an HTTP response through an injected `HttpServer` adapter. Its constructor stores that

adapter, and `RouterExecutionContext` creates one instance from its application adapter.

`router-response-controller.ts:28-31` `router-execution-context.ts:62-78`