adonisjs / **http-server** Public

Code Issues Pull requests Actions Projects Security and quality 1

Commit `2008fb6`

thetutlage committed last week · ✕ 1 / 4 · Verified

```
feat: add isValidRedirectUrl helper, ctx on Redirect, and helper tests

Extract redirect URL validation into a reusable isValidRedirectUrl
helper that handles relative paths, protocol-relative rejection, and
optional host validation for absolute URLs. Refactor getPreviousUrl to
delegate to it. Add ctx property to Redirect for use by session-based
macros. Add comprehensive tests for both helpers.
```

8.x · v8.2.0

1 parent 929ea1a commit 2008fb6

5 files changed +212 -16 lines changed

Top

Filter files…

- src
  - helpers.ts
  - redirect.ts
  - response.ts
- tests/helpers
  - get_previous_url.spec.ts
  - is_valid_redirect_url.spec.ts

5 files changed +212 -16 lines changed

Search within code

src/helpers.ts

```
@@ -31,14 +31,71 @@ import {
31  31
32  32  export { createURL }
```

```
33  33
        34  + /**
        35  +  * Validates that a URL is safe to use as a redirect destination.
        36  +  *
        37  +  * - Relative URLs must start with `/` and not be protocol-relative (`//`)
        38  +  * - Absolute URLs must parse successfully and their host must match
        39  +  *   `currentHost` or be listed in `allowedHosts`
        40  +  *
        41  +  * When `currentHost` and `allowedHosts` are omitted, absolute URLs
        42  +  * are accepted as long as they parse successfully.
        43  +  *
        44  +  * @param url - The URL to validate
        45  +  * @param currentHost - The current request's Host header value
        46  +  * @param allowedHosts - Array of additionally allowed hosts
        47  +  */
        48  + export function isValidRedirectUrl(
        49  +   url: string,
        50  +   currentHost?: string,
        51  +   allowedHosts?: string[]
        52  + ): boolean {
        53  +   if (typeof url !== 'string' || url.trim() === '') {
        54  +     return false
        55  +   }
        56  +
        57  +   if (url.startsWith('//')) {
        58  +     return false
        59  +   }
        60  +
        61  +   if (url.startsWith('/')) {
        62  +     try {
        63  +       const parsed = new URL(url, 'http://localhost')
        64  +       return parsed.host === 'localhost'
        65  +     } catch {
        66  +       return false
        67  +     }
        68  +   }
        69  +
        70  +   try {
        71  +     const parsed = new URL(url)
        72  +
```

```
+     /**
+      * When no host constraints are provided, accept any
+      * parseable absolute URL
+      */
+     if (!currentHost && (!allowedHosts || allowedHosts.length === 0)) {
+       return true
+     }
+
+     if (currentHost && parsed.host === currentHost) {
+       return true
+     }
+
+     if (allowedHosts && allowedHosts.length > 0 &&
  allowedHosts.includes(parsed.host)) {
+       return true
+     }
+
+     return false
+   } catch {
+     return false
+   }
+ }
+
  /**
   * Returns the previous URL from the request's `Referer` header,
   * validated against the request's `Host` header and an optional
-  * list of allowed hosts.
-  *
-  * The referrer is accepted when its host matches the request's
-  * `Host` header or is listed in `allowedHosts`. Otherwise the
-  * `fallback` value is returned.
+  * list of allowed hosts using `isValidRedirectUrl`.
   *
   * @param headers - The incoming request headers
   * @param allowedHosts - Array of allowed referrer hosts
@@ -58,17 +115,8 @@ export function getPreviousUrl(
       referrer = referrer[0]
     }

-   try {
```

```
62        -     const parsed = new URL(referrer)
63        -     const host = headers['host']
64        -     if (host && parsed.host === host) {
65        -       return referrer
66        -     }
67        -     if (allowedHosts.length > 0 && allowedHosts.includes(parsed.host)) {
68        -       return referrer
69        -     }
70        -   } catch {
71        -     // malformed URL
      118 +   if (isValidRedirectUrl(referrer, headers['host'], allowedHosts)) {
      119 +     return referrer
72    120       }
73    121
74    122       return fallback
```

src/redirect.ts

```
          @@ -24,6 +24,7 @@ import type {
24    24    } from './types/url_builder.ts'
25    25    import { safeDecodeURI } from './utils.ts'
26    26    import Macroable from '@poppinss/macroable'
      27  + import type { HttpContext } from './http_context/main.ts'
27    28
28    29    /**
29    30     * Provides a fluent API for constructing HTTP redirect responses.
          @@ -48,6 +49,12 @@ import Macroable from '@poppinss/macroable'
48    49     * ```
49    50     */
50    51    export class Redirect extends Macroable {
      52  +   /**
      53  +    * HTTP context reference, set by the response when creating
      54  +    * the redirect instance during request handling.
      55  +    */
      56  +   ctx?: HttpContext
      57  +
51    58      /**
52    59       * Array of allowed hosts for referrer-based redirects.
53    60       * When empty, only the request's own host is allowed.
```

src/response.ts

```
@@ -1039,6 +1039,7 @@ export class HttpResponse extends Macroable {
1039 1039        statusCode: number = ResponseStatus.Found
1040 1040      ): Redirect | void {
1041 1041        const handler = new Redirect(this.request, this, this.#router, this.#qs,
     this.#config.redirect)
     1042 +      handler.ctx = this.ctx
1042 1043
1043 1044        if (forwardQueryString) {
1044 1045          handler.withQs()
```

tests/helpers/get_previous_url.spec.ts

```
@@ -0,0 +1,62 @@
1  + /*
2  +  * @adonisjs/http-server
3  +  *
4  +  * (c) AdonisJS
5  +  *
6  +  * For the full copyright and license information, please view the LICENSE
7  +  * file that was distributed with this source code.
8  +  */
9  +
10 + import { test } from '@japa/runner'
11 + import { getPreviousUrl } from '../../src/helpers.ts'
12 +
13 + test.group('getPreviousUrl', () => {
14 +   test('return fallback when no referer header is set', ({ assert }) => {
15 +     assert.equal(getPreviousUrl({}, [], '/fallback'), '/fallback')
16 +   })
17 +
18 +   test('return fallback when referer header is empty', ({ assert }) => {
19 +     assert.equal(getPreviousUrl({ referer: '' }, [], '/'), '/')
20 +   })
21 +
22 +   test('accept referer matching the host header', ({ assert }) => {
23 +     const headers = { referer: 'https://example.com/foo', host: 'example.com' }
24 +     assert.equal(getPreviousUrl(headers, [], '/'), 'https://example.com/foo')
```

```
25 +   })
26 +
27 +   test('accept referer matching an allowed host', ({ assert }) => {
28 +     const headers = { referer: 'https://admin.example.com/bar', host:
     'example.com' }
29 +     assert.equal(
30 +       getPreviousUrl(headers, ['admin.example.com'], '/'),
31 +       'https://admin.example.com/bar'
32 +     )
33 +   })
34 +
35 +   test('return fallback when referer host does not match', ({ assert }) => {
36 +     const headers = { referer: 'https://evil.com/phish', host: 'example.com' }
37 +     assert.equal(getPreviousUrl(headers, [], '/'), '/')
38 +   })
39 +
40 +   test('return fallback for malformed referer', ({ assert }) => {
41 +     const headers = { referer: '//evil.com', host: 'example.com' }
42 +     assert.equal(getPreviousUrl(headers, [], '/'), '/')
43 +   })
44 +
45 +   test('accept relative referer URLs', ({ assert }) => {
46 +     const headers = { referer: '/foo/bar', host: 'example.com' }
47 +     assert.equal(getPreviousUrl(headers, [], '/'), '/foo/bar')
48 +   })
49 +
50 +   test('use the first value when referer is an array', ({ assert }) => {
51 +     const headers = {
52 +       referer: ['https://example.com/a', 'https://example.com/b'] as any,
53 +       host: 'example.com',
54 +     }
55 +     assert.equal(getPreviousUrl(headers, [], '/'), 'https://example.com/a')
56 +   })
57 +
58 +   test('use referrer header when referer is not set', ({ assert }) => {
59 +     const headers = { referrer: 'https://example.com/foo', host: 'example.com' }
     as any
60 +     assert.equal(getPreviousUrl(headers, [], '/'), 'https://example.com/foo')
61 +   })
62 + })
```

tests/helpers/is_valid_redirect_url.spec.ts

```
@@ -0,0 +1,78 @@
+ /*
+  * @adonisjs/http-server
+  *
+  * (c) AdonisJS
+  *
+  * For the full copyright and license information, please view the LICENSE
+  * file that was distributed with this source code.
+  */
+
+ import { test } from '@japa/runner'
+ import { isValidRedirectUrl } from '../../src/helpers.ts'
+
+ test.group('isValidRedirectUrl', () => {
+   test('accept relative URLs starting with /', ({ assert }) => {
+     assert.isTrue(isValidRedirectUrl('/'))
+     assert.isTrue(isValidRedirectUrl('/foo'))
+     assert.isTrue(isValidRedirectUrl('/foo/bar'))
+     assert.isTrue(isValidRedirectUrl('/foo?bar=baz'))
+     assert.isTrue(isValidRedirectUrl('/foo#section'))
+   })
+
+   test('reject empty or non-string values', ({ assert }) => {
+     assert.isFalse(isValidRedirectUrl(''))
+     assert.isFalse(isValidRedirectUrl('   '))
+     assert.isFalse(isValidRedirectUrl(null as any))
+     assert.isFalse(isValidRedirectUrl(undefined as any))
+     assert.isFalse(isValidRedirectUrl(123 as any))
+   })
+
+   test('reject protocol-relative URLs', ({ assert }) => {
+     assert.isFalse(isValidRedirectUrl('//evil.com'))
+     assert.isFalse(isValidRedirectUrl('//evil.com/path'))
+   })
+
+   test('reject relative URLs that trick the URL parser', ({ assert }) => {
+     assert.isFalse(isValidRedirectUrl('/\\evil.com'))
+   })
```

```
38 +
39 +   test('accept absolute URLs when no host constraints are provided', ({ assert }) => {
40 +     assert.isTrue(isValidRedirectUrl('https://example.com'))
41 +     assert.isTrue(isValidRedirectUrl('https://example.com/path'))
42 +     assert.isTrue(isValidRedirectUrl('http://localhost:3000/foo'))
43 +   })
44 +
45 +   test('reject malformed absolute URLs', ({ assert }) => {
46 +     assert.isFalse(isValidRedirectUrl('not-a-url'))
47 +     assert.isFalse(isValidRedirectUrl('ftp://'))
48 +   })
49 +
50 +   test('accept absolute URLs matching currentHost', ({ assert }) => {
51 +     assert.isTrue(isValidRedirectUrl('https://example.com/foo', 'example.com'))
52 +     assert.isTrue(isValidRedirectUrl('http://localhost:3000/foo', 'localhost:3000'))
53 +   })
54 +
55 +   test('reject absolute URLs not matching currentHost', ({ assert }) => {
56 +     assert.isFalse(isValidRedirectUrl('https://evil.com/foo', 'example.com'))
57 +     assert.isFalse(isValidRedirectUrl('https://example.com:8080/foo', 'example.com'))
58 +   })
59 +
60 +   test('accept absolute URLs in allowedHosts', ({ assert }) => {
61 +     assert.isTrue(isValidRedirectUrl('https://app.example.com/foo', undefined, ['app.example.com']))
62 +     assert.isTrue(
63 +       isValidRedirectUrl('https://admin.example.com/foo', 'app.example.com', [
64 +         'admin.example.com',
65 +       ])
66 +     )
67 +   })
68 +
69 +   test('reject absolute URLs not in allowedHosts or currentHost', ({ assert }) => {
70 +     assert.isFalse(
71 +       isValidRedirectUrl('https://evil.com/foo', 'example.com', ['app.example.com'])
```

```
72 +     )
73 +   })
74 +
75 +   test('relative URLs ignore host constraints', ({ assert }) => {
76 +     assert.isTrue(isValidRedirectUrl('/foo', 'example.com',
   ['app.example.com']))
77 +   })
78 + })
```

Comments 0

Please sign in to comment.