Response Helpers

The Server Response object, (often abbreviated as `res`) includes a set of Express.js-like helper methods to improve the developer experience and increase the speed of creating new API endpoints.

The included helpers are:

- `res.status(code)` - A function to set the status code. `code` must be a valid HTTP status code
- `res.json(body)` - Sends a JSON response. `body` must be a serializable object
- `res.send(body)` - Sends the HTTP response. `body` can be a `string`, an `object` or a `Buffer`
- `res.redirect([status,] path)` - Redirects to a specified path or URL. `status` must be a valid HTTP status code. If not specified, `status` defaults to "307" "Temporary redirect".
- `res.unstable_revalidate(urlPath)` - Revalidate a page on demand using `getStaticProps`. `urlPath` must be a `string`.

Setting the status code of a response

When sending a response back to the client, you can set the status code of the response.

The following example sets the status code of the response to `200` (`OK`) and returns a `message` property with the value of `Hello from Next.js!` as a JSON response:

```
export default function handler(req, res) {
  res.status(200).json({ message: 'Hello from Next.js!' })
}
```

Sending a JSON response

When sending a response back to the client you can send a JSON response, this must be a serializable object. In a real world application you might want to let the client know the status of the request depending on the result of the requested endpoint.

The following example sends a JSON response with the status code `200` (`OK`) and the result of the async operation. It's contained in a try catch block to handle any errors that may occur, with the appropriate status code and error message caught and sent back to the client:

```
export default async function handler(req, res) {
  try {
    const result = await someAsyncOperation()
    res.status(200).json({ result })
```

```
  } catch (err) {
    res.status(500).json({ error: 'failed to load data' })
  }
}
```

Sending a HTTP response

Sending an HTTP response works the same way as when sending a JSON response. The only difference is that the response body can be a `string`, an `object` or a `Buffer`.

The following example sends a HTTP response with the status code `200` (`OK`) and the result of the async operation.

```
export default async function handler(req, res) {
  try {
    const result = await someAsyncOperation()
    res.status(200).send({ result })
  } catch (err) {
    res.status(500).send({ error: 'failed to fetch data' })
  }
}
```

Redirects to a specified path or URL

Taking a form as an example, you may want to redirect your client to a specified path or URL once they have submitted the form.

The following example redirects the client to the `/` path if the form is successfully submitted:

```
export default async function handler(req, res) {
  const { name, message } = req.body
  try {
    await handleFormInputAsync({ name, message })
    res.redirect(307, '/')
  } catch (err) {
    res.status(500).send({ error: 'failed to fetch data' })
  }
}
```

Adding TypeScript types

You can make your response handlers more type-safe by importing the `NextApiRequest` and `NextApiResponse` types from `next`, in addition to those, you can also type your response data:

```
import type { NextApiRequest, NextApiResponse } from 'next'
```

```ts
type ResponseData = {
  message: string
}

export default function handler(
  req: NextApiRequest,
  res: NextApiResponse<ResponseData>
) {
  res.status(200).json({ message: 'Hello from Next.js!' })
}
```

To view more examples using types, check out the TypeScript documentation.

If you prefer to view your examples within a real projects structure you can checkout our examples repository:

- Basic API Routes
- API Routes with REST