# **Lab 11**

## HTTP Basic Authentication

Middleware can help you with HTTP Basic Authentication if you want to use that in your application. Despite being frequently maligned as outdated and unsafe, Basic Authentication, when combined with SSL and HTTPS, can be a reasonable and sufficiently robust security solution for your applications.

## How to use

The module will export a function, that you can call with an options object to get the middleware:

Graphical user interface, text, application

Description automatically generated

## Setup

npm install express

npm install express-basic-auth

Example

app.js

|  |
| --- |
| const app = require('express')();  const basicAuth = require('express-basic-auth');  app.use(basicAuth({  users: { 'admin': 'abc123' } //Basic YWRtaW46YWJjMTIz  }));  app.get('/', function(req, res){  res.send('secret message that only auth\'d users can see\n');  });  app.listen(8080); |

Turn on the server:

node app.js

Send request in Postman

Graphical user interface, text, application

Description automatically generated

The middleware will check incoming requests for a basic auth (Authorization) header, parse it and check if the credentials are legit. If there are any credentials, an auth property will be added to the request, containing an object with user and password properties, filled with the credentials, no matter if they are legit or not.

**If a request is found to not be authorized**, it will respond with HTTP 401 and a configurable body (default empty).

## To Encode

This is at frontend, we don’t do too much. We may run the following command in Browser DevTool to figure out the encoded code.

window.btoa(unescape(encodeURIComponent("admin:abc123")))

Ref: <https://developer.mozilla.org/en-US/docs/Glossary/Base64>

Exercise #1

Make it work with a different credential as aladdin:opensesame

app.js

|  |
| --- |
| const app = require('express')();  const basicAuth = require('express-basic-auth');  app.use(basicAuth({      users: { 'admin': 'NodeJS\_isCool' } //Basic YWRtaW46Tm9kZUpTX2lzQ29vbA==  }));  app.get('/', function(req, res){      res.send('secret message that only auth\'d users can see\n');  });  app.listen(8080); |

Screenshot of your Postman showing request header settings

|  |
| --- |
|  |

Screenshot of your Postman with response

|  |
| --- |
|  |

### Static Users

If you simply want to check basic auth against one or multiple static credentials, you can pass those credentials in the users option:

Graphical user interface

Description automatically generated with low confidence

The middleware will check incoming requests to have a basic auth header matching one of the three passed credentials.

### Custom authorization

Alternatively, you can pass your own authorizer function, to check the credentials however you want. It will be called with a username and password and is expected to return true or false to indicate that the credentials were approved or not.

When using your own authorizer, make sure **not to use standard string comparison (== / ===)** when comparing user input with secret credentials, as that would make you vulnerable against [**timing attacks**](https://en.wikipedia.org/wiki/Timing_attack). Use the provided safeCompare function instead - always provide the user input as its first argument. Also make sure to use bitwise logic operators (| and &) instead of the standard ones (|| and &&) for the same reason, as the standard ones use shortcuts.

app.use(basicAuth( { authorizer: myAuthorizer } ))

function myAuthorizer(username, password) {

const userMatches = basicAuth.safeCompare(username, 'customuser');

const passwordMatches = basicAuth.safeCompare(password, 'custompassword');

return (userMatches & passwordMatches);

}

This will authorize all requests with the credentials 'customuser:custompassword'. In an actual application you would likely look up some data instead ;-) You can do whatever you want in custom authorizers, just return true or false in the end and stay aware of timing attacks.

Exercise #2

Work out an implementation of a new myAuthorizer with checking a list of credentials

Assume we have:

const credentials = [ {'admin': 'pwd'}, {'root': 'p@ssword'},{'sam': 'abc'}, {'bob': 'xyz'}];

|  |
| --- |
| const app = require('express')();  const basicAuth = require('express-basic-auth');  const credentials = [ {'admin': 'pwd'}, {'root': 'p@ssword'},{'sam': 'abc'}, {'bob': 'xyz'}];  app.use(basicAuth( { authorizer: myAuthorizer } ))  function myAuthorizer(username, password) {      for (i of credentials){          for (let key in i) {              const userMatches = basicAuth.safeCompare(username, key); //this is the key in the dict              const passwordMatches = basicAuth.safeCompare(password, i[key]); // this is the value of the dict              if (userMatches & passwordMatches){ //check if condition and if true return and if not just continue looping                  return (userMatches & passwordMatches);              }          }      }  }    app.get('/', function(req, res){      res.send('secret message that only auth\'d users can see\n');  });  app.listen(8080); |

### Custom Async Authorization

Note that the authorizer function above is expected to be synchronous. This is the default behavior, you can pass authorizeAsync: true in the options object to indicate that your authorizer is asynchronous. In this case it will be passed a callback as the third parameter, which is expected to be called by standard node convention with an error and a boolean to indicate if the credentials have been approved or not. Let's look at the same authorizer again, but this time asynchronous:

Graphical user interface, text, application, email

Description automatically generated

### Unauthorized Response Body

Per default, the response body for unauthorized responses will be empty. It can be configured using the unauthorizedResponse option. You can either pass a static response or a function that gets passed the express request object and is expected to return the response body. If the response body is a string, it will be used as-is, otherwise it will be sent as JSON:

Graphical user interface

Description automatically generated with medium confidence

### TypeScript usage

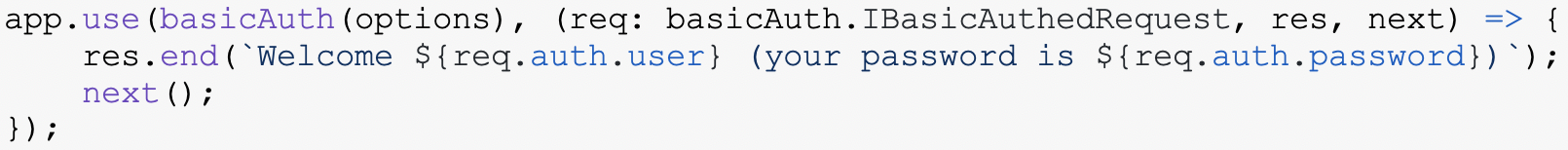
A declaration file is bundled with the library. You don't have to install a @types/ package.



**Using req.auth**

express-basic-auth sets req.auth to an object containing the authorized credentials like { user: 'admin', password: 'supersecret' }.

In order to use that req.auth property in TypeScript without an unknown property error, use covariance to downcast the request type:



Exercise #3

Based on the given index.html from Socket.io, to work out a frontend page and backend (app.js) to allow user put in their username and password to login.

Hint: you may hardcode credentials or load them from a json.

Index.html

|  |
| --- |
|  |

app.js

|  |
| --- |
|  |

credentials.json (optional)

|  |
| --- |
|  |

Exercise #4 (challenge)

Based on your Exercise #3, add on a signup page. (input validations are optional)

Hint: If you hardcoded credentials in Ex3, you may revise it to be a list, and add the new credential.

index.html (for signup)

|  |
| --- |
|  |

login.html (for login)

|  |
| --- |
|  |

app.js

|  |
| --- |
|  |

credentials.json (optional)

|  |
| --- |
|  |

## Deployment

Recall what we have in the lab of Socket.io, if you didn’t get Heroku part done.

Try the following steps to revise your package.json

1) To step down the node version to 14, which is supported by Heroku.

2) To add a "start" key into scripts section, with the node command and the server file name, to provide an explicit starting point.

3) To mirror the DevDependencies, with the "Dependencies" key and values, and placed them before DevDependencies. Or re-run command:

npm install express socket.io

After verifying everything works as expected at your local, try given steps to upload your project to Heroku.

Exercise #5

Make a screenshot of your browser running your Heroku app, and then paste it below.

|  |
| --- |
|  |