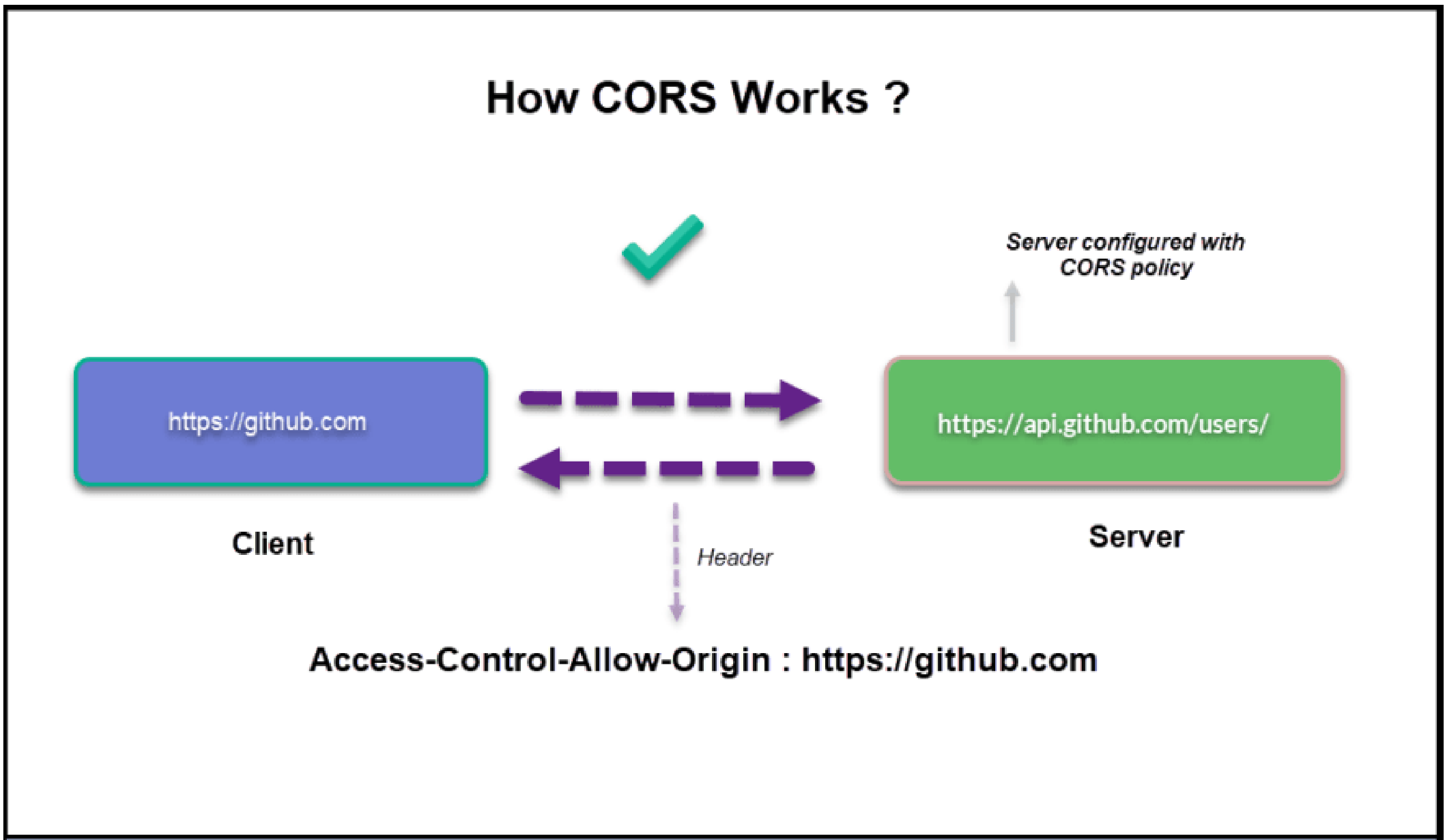
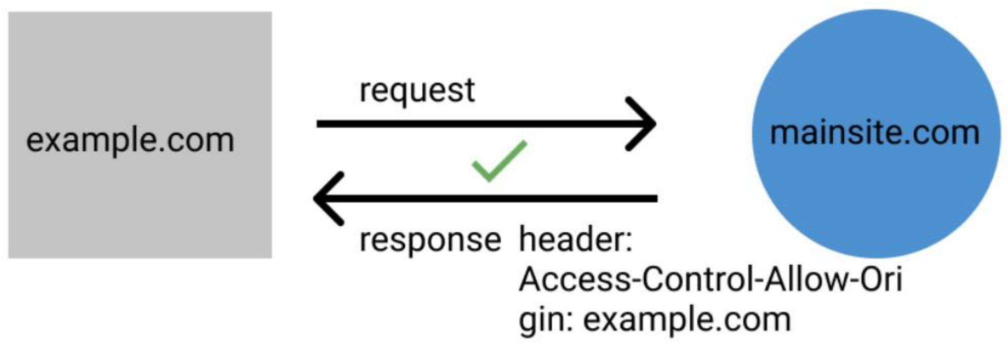
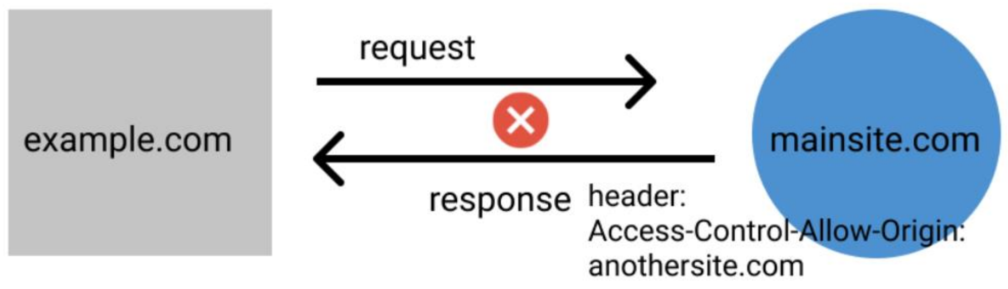
Cors is a node js package that adds a layer of security to your api/server and limit who can access it or limit who can access a certain route.


How CORS Works ?
Server configured with CORS policy
https://github.com
https://api.github.com/users/
Client
Server
Header
Access-Control-Allow-Origin : https://github.com

diipakkr dipakkr.co diipakkr



Good: Origin is in response header



Error: Origin not in response header

Installation

This is a Node.js module available through the npm registry. Installation is done using the `npm install` command:

```
$ npm install cors
```

Usage

Simple Usage (Enable *All* CORS Requests)

```
var express = require('express')
var cors = require('cors')
var app = express()

app.use(cors())

app.get('/products/:id', function (req, res, next) {
  res.json({msg: 'This is CORS-enabled for all origins!'})
})

app.listen(80, function () {
  console.log('CORS-enabled web server listening on port 80')
})
```

Enable CORS for a Single Route

```
var express = require('express')
var cors = require('cors')
var app = express()

app.get('/products/:id', cors(), function (req, res, next) {
  res.json({msg: 'This is CORS-enabled for a Single Route'})
})

app.listen(80, function () {
  console.log('CORS-enabled web server listening on port 80')
})
```

Configuring CORS

```
var express = require('express')
var cors = require('cors')
var app = express()

var corsOptions = {
  origin: 'http://example.com',
  optionsSuccessStatus: 200 // some legacy browsers (IE11, various SmartTVs) choke
on 204
}

app.get('/products/:id', cors(corsOptions), function (req, res, next) {
  res.json({msg: 'This is CORS-enabled for only example.com.'})
})

app.listen(80, function () {
  console.log('CORS-enabled web server listening on port 80')
})
```

Configuring CORS w/ Dynamic Origin

This module supports validating the origin dynamically using a function provided to the `origin` option. This function will be passed a string that is the origin (or `undefined` if the request has no origin), and a `callback` with the signature `callback(error, origin)`.

The `origin` argument to the callback can be any value allowed for the `origin` option of the middleware, except a function. See the configuration options section for more information on all the possible value types.

This function is designed to allow the dynamic loading of allowed origin(s) from a backing datasource, like a database.

```
var express = require('express')
var cors = require('cors')
var app = express()

var corsOptions = {
  origin: function (origin, callback) {
    // db.loadOrigins is an example call to load
    // a list of origins from a backing database
    db.loadOrigins(function (error, origins) {
      callback(error, origins)
    })
  }
}

app.get('/products/:id', cors(corsOptions), function (req, res, next) {
  res.json({msg: 'This is CORS-enabled for an allowed domain.'})
})

app.listen(80, function () {
  console.log('CORS-enabled web server listening on port 80')
})
```

Enabling CORS Pre-Flight

Certain CORS requests are considered ‘complex’ and require an initial `OPTIONS` request (called the “pre-flight request”). An example of a ‘complex’ CORS request is one that uses an HTTP verb other than GET/HEAD/POST (such as DELETE) or that uses custom headers. To enable pre-flighting, you must add a new OPTIONS handler for the route you want to support:

```
var express = require('express')
var cors = require('cors')
var app = express()

app.options('/products/:id', cors()) // enable pre-flight request for DELETE
request
app.del('/products/:id', cors(), function (req, res, next) {
  res.json({msg: 'This is CORS-enabled for all origins!'})
})

app.listen(80, function () {
  console.log('CORS-enabled web server listening on port 80')
})
```

You can also enable pre-flight across-the-board like so:

```
app.options('*', cors()) // include before other routes
```

NOTE: When using this middleware as an application level middleware (for example, `app.use(cors())`), pre-flight requests are already handled for all routes.

Configuring CORS Asynchronously

```
var express = require('express')
var cors = require('cors')
var app = express()

var allowlist = ['http://example1.com', 'http://example2.com']
var corsOptionsDelegate = function (req, callback) {
  var corsOptions;
  if (allowlist.indexOf(req.header('Origin')) !== -1) {
    corsOptions = { origin: true } // reflect (enable) the requested origin in the
CORS response
  } else {
    corsOptions = { origin: false } // disable CORS for this request
  }
  callback(null, corsOptions) // callback expects two parameters: error and
options
}

app.get('/products/:id', cors(corsOptionsDelegate), function (req, res, next) {
  res.json({msg: 'This is CORS-enabled for an allowed domain.'})
})
```

```
app.listen(80, function () {
  console.log('CORS-enabled web server listening on port 80')
})
```

Configuration Options

- `origin`: Configures the **Access-Control-Allow-Origin** CORS header. Possible values:
  - `Boolean` - set `origin` to `true` to reflect the request origin, as defined by `req.header('Origin')`, or set it to `false` to disable CORS.
  - `String` - set `origin` to a specific origin. For example if you set it to `"http://example.com"` only requests from "http://example.com" will be allowed.
  - `RegExp` - set `origin` to a regular expression pattern which will be used to test the request origin. If it's a match, the request origin will be reflected. For example the pattern `/example\.com$/` will reflect any request that is coming from an origin ending with "example.com".
  - `Array` - set `origin` to an array of valid origins. Each origin can be a `String` or a `RegExp`. For example `["http://example1.com", /\.example2\.com$/]` will accept any request from "http://example1.com" or from a subdomain of "example2.com".
  - `Function` - set `origin` to a function implementing some custom logic. The function takes the request origin as the first parameter and a callback (called as `callback(err, origin)`, where `origin` is a non-function value of the `origin` option) as the second.
- `methods`: Configures the **Access-Control-Allow-Methods** CORS header. Expects a comma-delimited string (ex: 'GET,PUT,POST') or an array (ex: `['GET', 'PUT', 'POST']`).
- `allowedHeaders`: Configures the **Access-Control-Allow-Headers** CORS header. Expects a comma-delimited string (ex: 'Content-Type,Authorization') or an array (ex: `['Content-Type', 'Authorization']`). If not specified, defaults to reflecting the headers specified in the request's **Access-Control-Request-Headers** header.
- `exposedHeaders`: Configures the **Access-Control-Expose-Headers** CORS header. Expects a comma-delimited string (ex: 'Content-Range,X-Content-Range') or an array (ex: `['Content-Range', 'X-Content-Range']`). If not specified, no custom headers are exposed.
- `credentials`: Configures the **Access-Control-Allow-Credentials** CORS header. Set to `true` to pass the header, otherwise it is omitted.
- `maxAge`: Configures the **Access-Control-Max-Age** CORS header. Set to an integer to pass the header, otherwise it is omitted.
- `preflightContinue`: Pass the CORS preflight response to the next handler.
- `optionsSuccessStatus`: Provides a status code to use for successful `OPTIONS` requests, since some legacy browsers (IE11, various SmartTVs) choke on `204`.

The default configuration is the equivalent of:

```
{
  "origin": "*",
  "methods": "GET,HEAD,PUT,PATCH,POST,DELETE",
  "preflightContinue": false,
  "optionsSuccessStatus": 204
}
```