Aurelia Http

Http Clients

- Aurelia comes with 2 http client libraries:
    - aurelia-http-client - A basic HttpClient based on XMLHttpRequest. It supports all HTTP verbs, JSONP and request cancellation.
    - aurelia-fetch-client - A more forward-looking HttpClient based on the Fetch specification. It supports all HTTP verbs and integrates with Service Workers, including Request/Response caching.

Fetch API

see all contributors

IN THIS ARTICLE +

This is an experimental technology
Because this technology's specification has not stabilized, check the compatibility table for usage in various browsers. Also note that the syntax and behavior of an experimental technology is subject to change in future versions of browsers as the specification changes.

The Fetch API provides an interface for fetching resources (including across the network). It will seem familiar to anyone who has used `XMLHttpRequest`, but the new API provides a more powerful and flexible feature set.

Concepts and usage

Fetch provides a generic definition of `Request` and `Response` objects (and other things involved with network requests). This will allow them to be used wherever they are needed in the future, whether it's for service workers, Cache API and other similar things that handle or modify requests and responses, or any kind of use case that might require you to generate your own responses programmatically.

It also provides a definition for related concepts such as CORS and the HTTP origin header semantics, supplanting their separate definitions elsewhere.

For making a request and fetching a resource, use the `GlobalFetch.fetch` method. It is implemented in multiple interfaces, specifically `Window` and `WorkerGlobalScope`. This makes it available in pretty much any context you might want to fetch resources in.

The `fetch()` method takes one mandatory argument, the path to the resource you want to fetch. It returns a promise that resolves to the

aurelia-http-client

- Provides a comfortable interface to the browser's XMLHttpRequest object.
- Not included in the modules that Aurelia's bootstrapper installs, since it's completely optional and many apps may choose to use a different strategy for data retrieval.
- Must install it first…

Installing aurelia-http-client

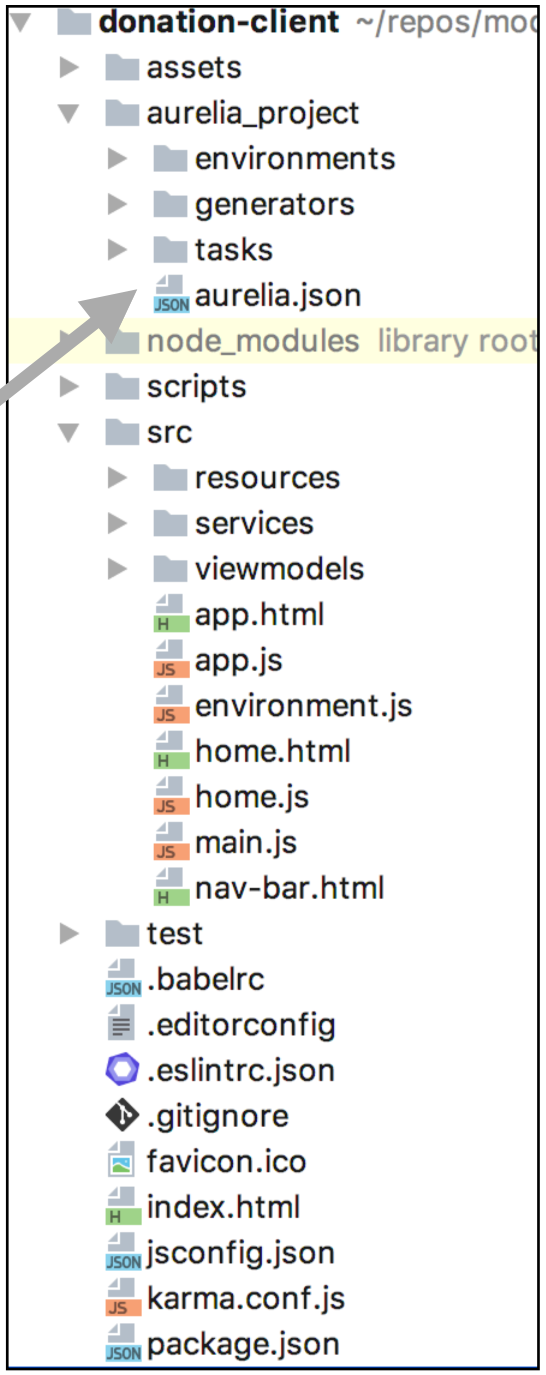
- Step 1: Install the component via npm

```
npm install aurelia-http-client -save
```

- Step 2: Specifically include in Aurelia Build Script

aurelia_project/aurelia.json

```
    "dependencies": [
        ...
        "aurelia-history-browser",
        "aurelia-http-client",
        "aurelia-loader",
        ...
```



```
▼ donation-client ~/repos/mod
  ▶ assets
  ▼ aurelia_project
    ▶ environments
    ▶ generators
    ▶ tasks
      aurelia.json
  ▶ node_modules  library root
  ▶ scripts
  ▼ src
    ▶ resources
    ▶ services
    ▶ viewmodels
      app.html
      app.js
      environment.js
      home.html
      home.js
      main.js
      nav-bar.html
  ▶ test
    .babelrc
    .editorconfig
    .eslintrc.json
    .gitignore
    favicon.ico
    index.html
    jsconfig.json
    karma.conf.js
    package.json
```

Using aurelia-http-client

- Import the client
- Create an instance (or inject it)
- Promises returned from:
  - get
  - put
  - post
  - delete
  - etc…

```
import {HttpClient} from 'aurelia-http-client';

let client = new HttpClient();

client.get('http://localhost:4000/api/candidates').then(data => {
  console.log(data.content);
});
```

DonationService

- Retrieve candidates & users from api server

```
@inject(Fixtures, EventAggregator, HttpClient)
export default class DonationService {

  donations = [];
  methods = [];
  candidates = [];
  users = [];
  total = 0;

  constructor(data, ea, hc) {
    this.methods = data.methods;
    this.ea = ea;
    this.hc = hc;
    this.getCandidates();
    this.getUsers();
  }

  getCandidates() {
    this.hc.get('http://localhost:4000/api/candidates').then(res => {
      this.candidates = res.content;
    });
  }

  getUsers() {
    this.hc.get('http://localhost:4000/api/users').then(res => {
      this.users = res.content;
    });
  }
...
```

DonationService

```
...

donate(amount, method, candidate) {
  const donation = {
    amount: amount,
    method: method
  };
  this.hc.post('http://localhost:4000/api/candidates/' + candidate._id + '/donations', donation)
     .then(res => {
       const returnedDonation = res.content;
       this.donations.push(returnedDonation);
       console.log(amount + ' donated to ' + candidate.firstName + ' ' +
                                           candidate.lastName + ': ' + method);
       this.total = this.total + parseInt(amount, 10);
       console.log('Total so far ' + this.total);
       this.ea.publish(new TotalUpdate(this.total));
     });
}

...
```

- Create a donation
- post to API Server

→ Response callback →

- save in donations array
- log it
- update total
- dispatch event to interested parties

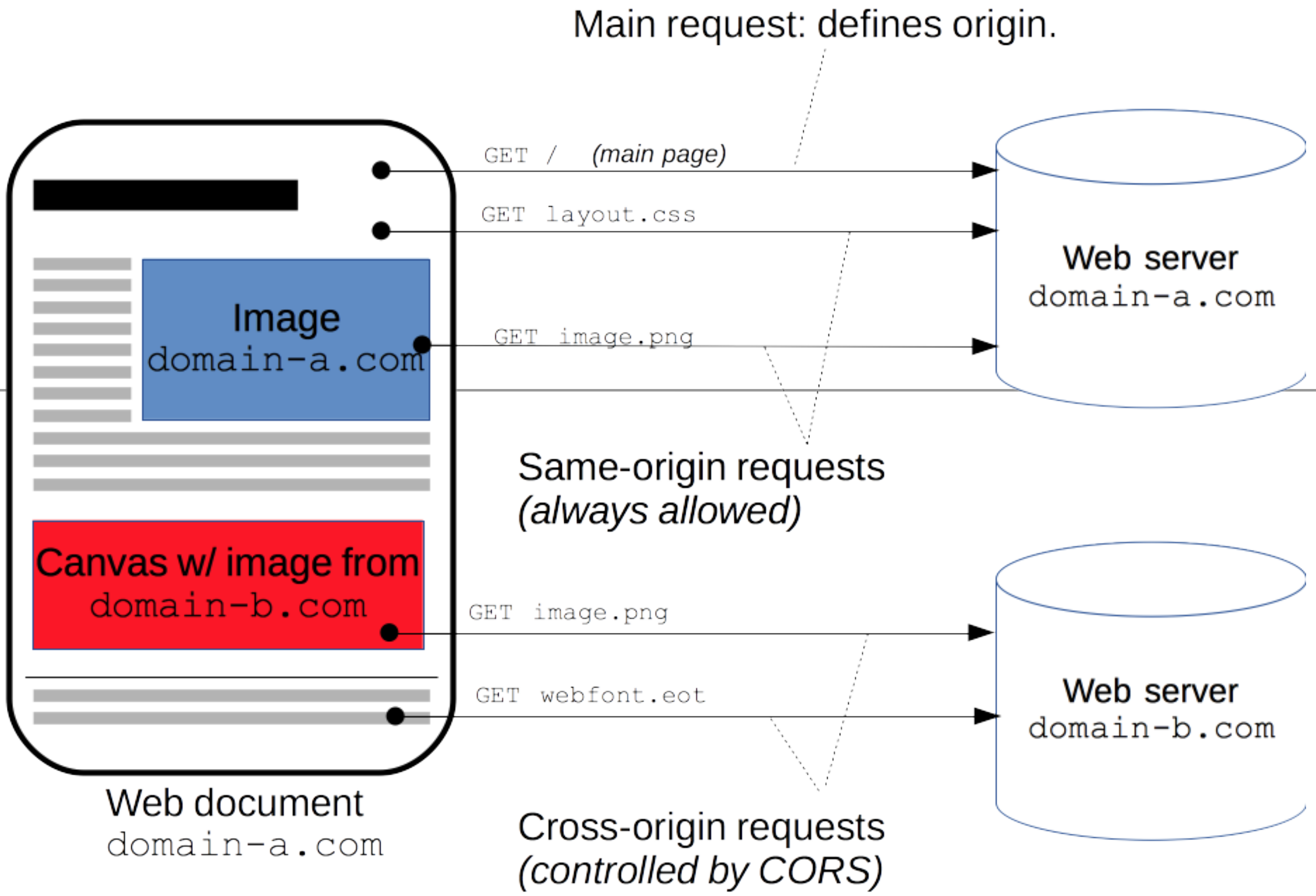
Cross Origin Requests

- A resource makes a cross-origin HTTP request when it requests a resource from a different domain than the one which the first resource itself serves.
- For example, an HTML page served from http://domain-a.com makes an <img> src request for http://domain-b.com/image.jpg.
- Many pages on the web today load resources like CSS stylesheets, images and scripts from separate domains.

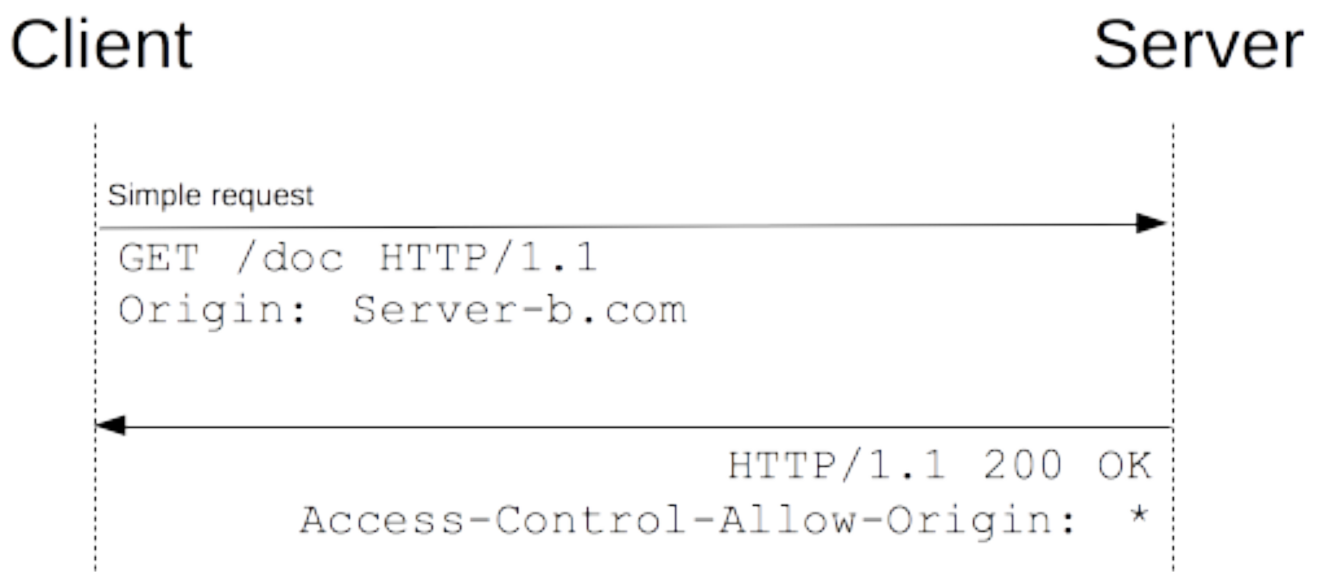
Restrictions

- For security reasons, browsers restrict cross-origin HTTP requests initiated from within scripts.
    - XMLHttpRequest follows the same-origin policy.
    - So, a web application using XMLHttpRequest could only make HTTP requests to its own domain.
- To improve web applications, developers asked browser vendors to allow cross-domain requests.

Cross Origin Resource Sharing (CORS)



- The Cross-Origin Resource Sharing (CORS) mechanism gives web servers cross-domain access controls, which enable secure cross-domain data transfers.



```
GET /doc HTTP/1.1
Origin: Server-b.com
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

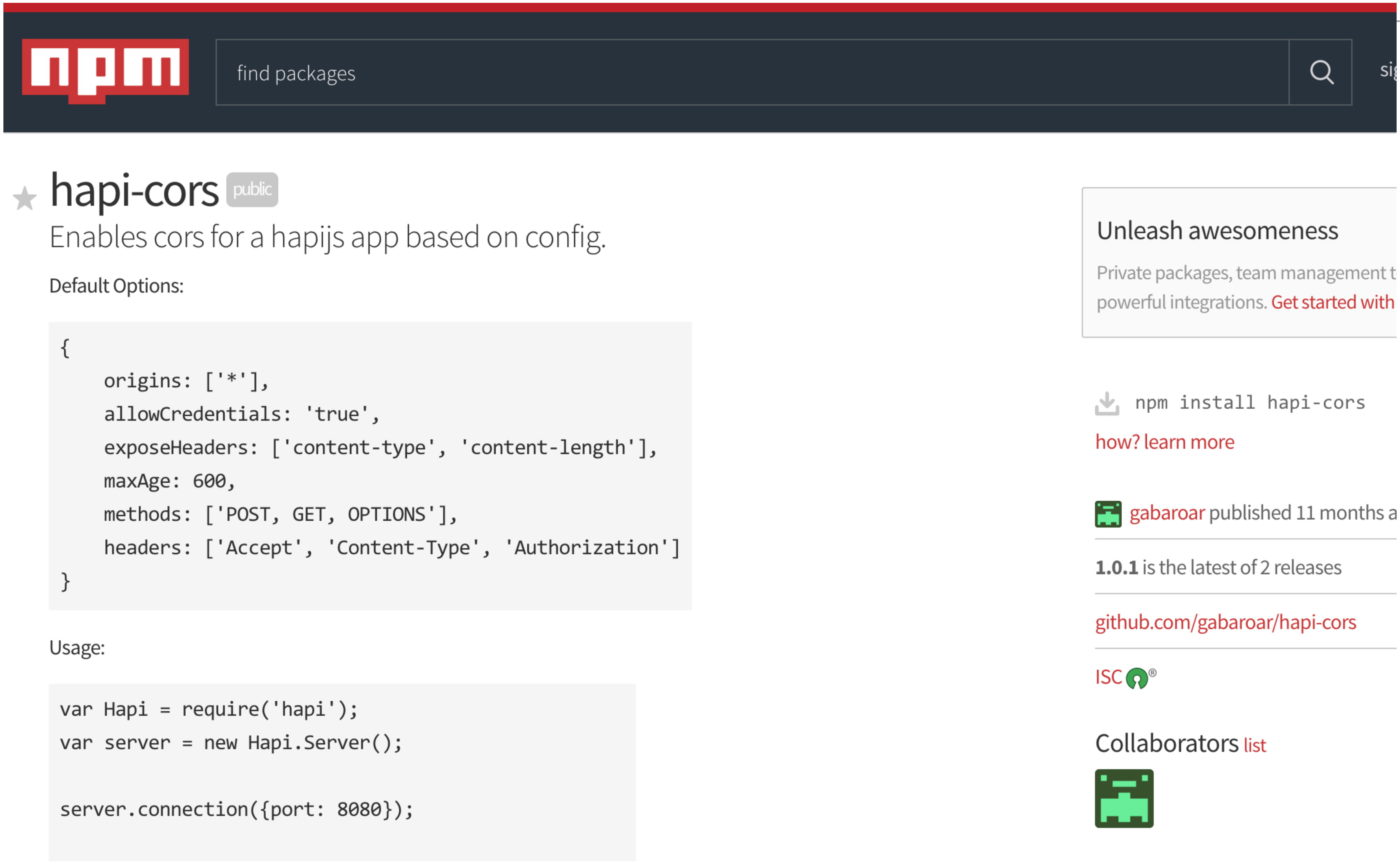
Cross Origin Request (COR)

- These requests to donation-web will fail due to COR restrictions
- The server will need some small modifications to permit this

```
getCandidates() {
  this.hc.get('http://localhost:4000/api/candidates').then(res => {
    this.candidates = res.content;
  });
}

getUsers() {
  this.hc.get('http://localhost:4000/api/users').then(res => {
    this.users = res.content;
  });
}
```

Hapi Cors Module



find packages

si

hapi-cors public

Enables cors for a hapijs app based on config.

Default Options:

```
{
    origins: ['*'],
    allowCredentials: 'true',
    exposeHeaders: ['content-type', 'content-length'],
    maxAge: 600,
    methods: ['POST, GET, OPTIONS'],
    headers: ['Accept', 'Content-Type', 'Authorization']
}
```

Usage:

```
var Hapi = require('hapi');
var server = new Hapi.Server();

server.connection({port: 8080});
```

Unleash awesomeness

Private packages, team management t
powerful integrations. Get started with

```
npm install hapi-cors
```

how? learn more

gabaroar published 11 months a

1.0.1 is the latest of 2 releases

github.com/gabaroar/hapi-cors

ISC

Collaborators list

Update to donation-web

- Install cors module

```
npm install hapi-cors-headers
```

- Index.js modifications:

```
const corsHeaders = require('hapi-cors-headers');

...

server.ext(‘onPreResponse’, corsHeaders);
server.route(require('./routes'));
server.route(require(‘./routesapi'));
...
```