Sessions in Hapi

Agenda

- Simple precursor to Sessions
- Sessions via Cookies in Hapi

Sharing Information across an App

- Before server launches, ‘bind’ an array of donations to the server object.

- Most commonly used to share database connection information

index.js

```
...
server.bind({
  donations: [],
});
...
```

Sharing Information across an App

- This ‘donations’ array can subsequently be accessed in all handlers.
- Each handler can read/write to this shared data structure
- All users donations held in effectively a global array in memory

index.js

```
...
server.bind({
  donations: [],
});
...
```

```
exports.donate = {

  handler: function (request, reply) {
    const data = request.payload;
    this.donations.push(data);
    reply.redirect('/report');
  },

};
```

donations.js

Separating out User Donations

- Try to keep track of
  - all users,
  - the current user
  - all donations.

index.js

```
...
server.bind({
  currentUser: {},
  users: {},
  donations: [],
});
...
```

Registering & Authenticating Users

```
exports.register = {

  handler: function (request, reply) {
    const user = request.payload;
    this.users[user.email] = user;
    reply.redirect('/login');
  },

};

exports.authenticate = {

  handler: function (request, reply) {
    const user = request.payload;
    if ((user.email in this.users) && (user.password === this.users[user.email].password)) {
      this.currentUser = this.users[user.email];
      reply.redirect('/home');
    } else {
      reply.redirect('/signup');
    }
  },

};
```

- Record user object at registration
- Record current user at login

accounts.js

Creating & Listing Donations

```
exports.donate = {

  handler: function (request, reply) {
    let data = request.payload;
    data.donor = this.currentUser;
    this.donations.push(data);
    reply.redirect('/report');
  },

};
```

donations.js

- Record donation + donor when creating donation

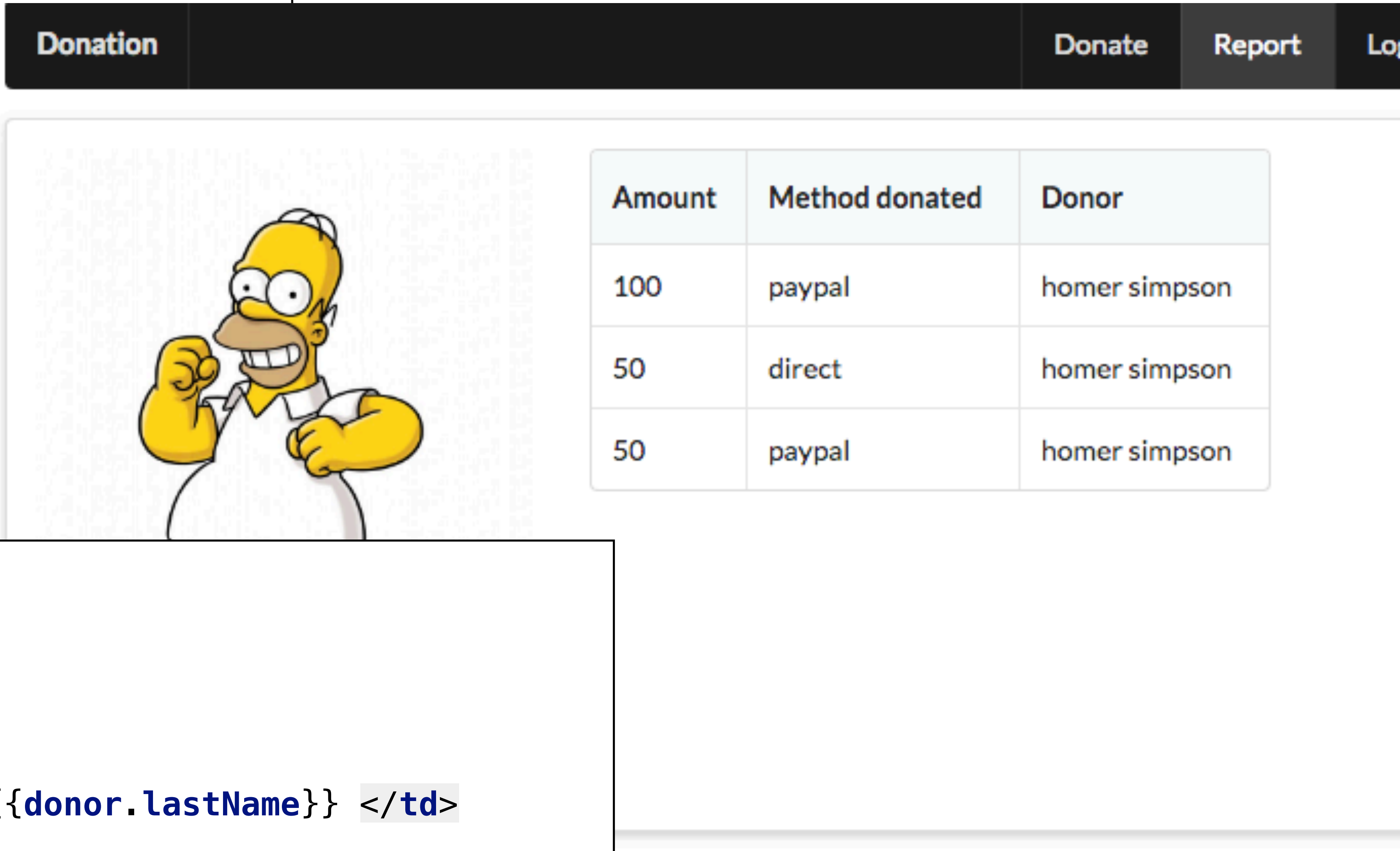
- Send all donations to the view

```
exports.report = {

  handler: function (request, reply) {
    reply.view('report', {
      title: 'Donations to Date',
      donations: this.donations,
    });
  },

};
```

donations.js



Amount	Method donated	Donor
100	paypal	homer simpson
50	direct	homer simpson
50	paypal	homer simpson

```
...
<tbody>
  {{#each donations}}
    <tr>
      <td> {{amount}} </td>
      <td> {{method}} </td>
      <td> {{donor.firstName}} {{donor.lastName}} </td>
    </tr>
  {{/each}}
</tbody>
...
```

donationlist.hbs

Summary

- Current approach - brittle and not scalable
  - Server.bind to maintain global data
  - Store user + donation data structures
- Revised Approach
  - Migrate to more robust, cookie based session management
  - Introduce proper persistence capability (a database)

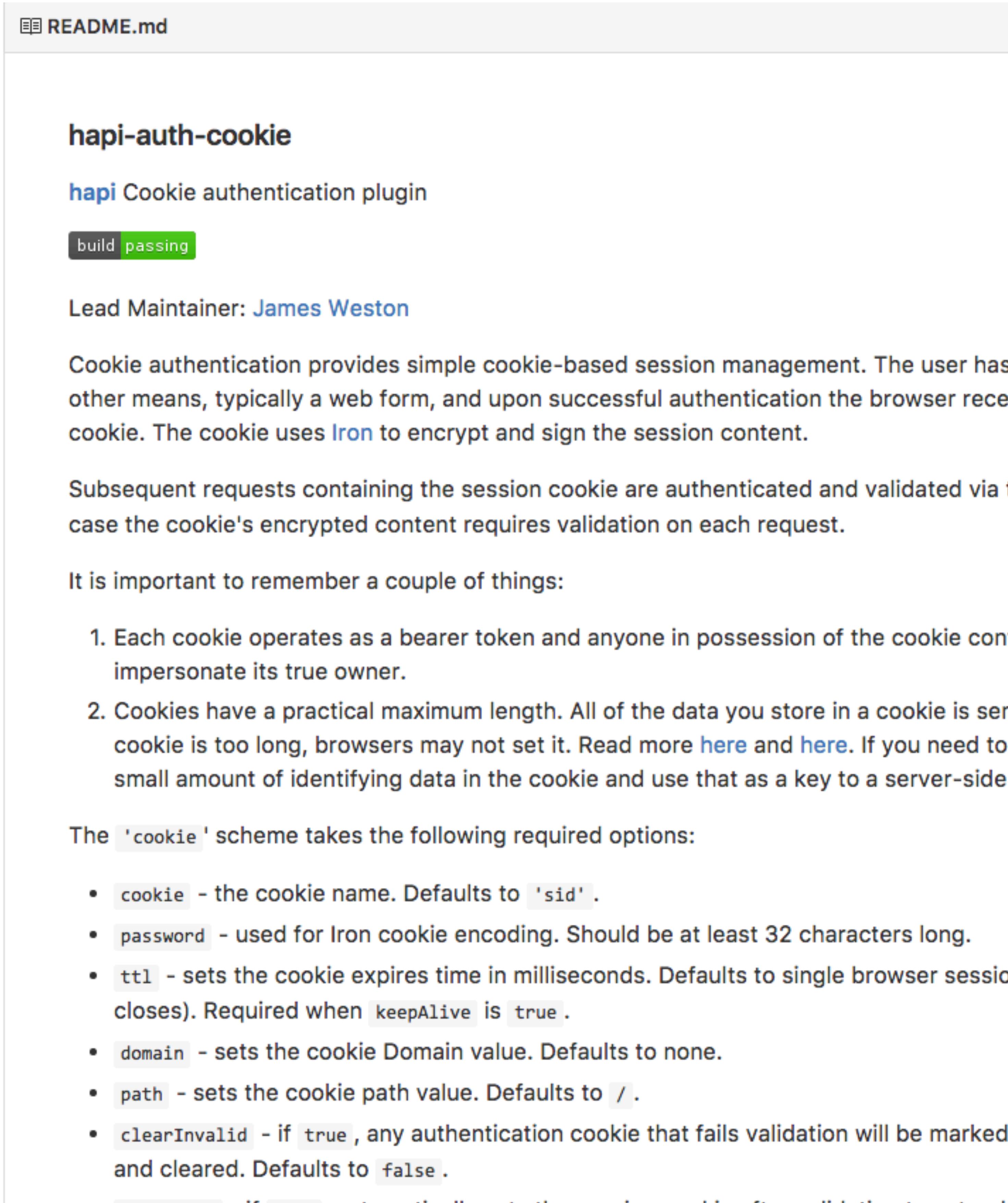
Sessions

- HTTP is described as a stateless protocol - every new request is just as anonymous as the last.
- This sounds very unhelpful for a protocol that powers websites, where users expect to be remembered as they go to page to page.
- Cookies to the Rescue:
  - A request comes to a web application with a cookie
  - using the cookie the server can look up information about the user, either from the cookie itself or from server-side storage.
  - It can then forget all about them for a while, until the next request and the same process continues over for every request.

hapi-auth-cookie

- A Hapi Plugin to manage cookie access and management.
- Must be downloaded, installed and registered (like all plugins)

https://github.com/hapijs/hapi-auth-cookie



README.md

hapi-auth-cookie

hapi Cookie authentication plugin

build passing

Lead Maintainer: James Weston

Cookie authentication provides simple cookie-based session management. The user has
other means, typically a web form, and upon successful authentication the browser rece
cookie. The cookie uses Iron to encrypt and sign the session content.

Subsequent requests containing the session cookie are authenticated and validated via
case the cookie's encrypted content requires validation on each request.

It is important to remember a couple of things:

1. Each cookie operates as a bearer token and anyone in possession of the cookie con
   impersonate its true owner.
2. Cookies have a practical maximum length. All of the data you store in a cookie is ser
   cookie is too long, browsers may not set it. Read more here and here. If you need to
   small amount of identifying data in the cookie and use that as a key to a server-side

The `'cookie'` scheme takes the following required options:

- `cookie` - the cookie name. Defaults to `'sid'`.
- `password` - used for Iron cookie encoding. Should be at least 32 characters long.
- `ttl` - sets the cookie expires time in milliseconds. Defaults to single browser sessio
  closes). Required when `keepAlive` is `true`.
- `domain` - sets the cookie Domain value. Defaults to none.
- `path` - sets the cookie path value. Defaults to `/`.
- `clearInvalid` - if `true`, any authentication cookie that fails validation will be marked
  and cleared. Defaults to `false`.

hapi-auth-cookie Installation & Registration

npm install command

```
npm install hapi-auth-cookie -save
```

package.json updated

```
{
  "name": "donation-web",
  "version": "1.0.0",
  "description": "an application to host donations for candidates",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "handlebars": "^4.0.5",
    "hapi": "^14.1.0",
    "hapi-auth-cookie": "^6.1.1",
    "inert": "^4.0.1",
    "vision": "^4.1.0"
  }
}
```

Register in index.js

```
server.register([require('inert'), require('vision'), require('hapi-auth-cookie')], err => {
```

hapi-auth-cookie Configuration

- Set an auth ‘strategy’ before application starts
- Specifies range or parameters, including:
  - password for securing cookie
  - cookie name
  - security level
  - time to live (expiry)

```
...
server.auth.strategy('standard', 'cookie', {
  password: 'secretpasswordnotrevealedtoanyone',
  cookie: 'donation-cookie',
  isSecure: false,
  ttl: 24 * 60 * 60 * 1000,
});
...
```

hapi-auth-cookie Configuration

- By default hapi-auth-cookie will only allow the cookie to be transferred over a secure TLS/ SSLconnection.
- This may not be convenient during development so you can set the isSecure option to false.
- Set ‘standard’ as the default strategy for all routes

```
...
server.auth.strategy('standard', 'cookie', {
  password: 'secretpasswordnotrevealedtoanyone',
  cookie: 'donation-cookie',
  isSecure: false,
  ttl: 24 * 60 * 60 * 1000,
  redirectTo: '/login',
});

server.auth.default({
  strategy: 'standard',
});

...
```

Annotating Routes

- All routes are now 'guarded' by default, cookie based authentication mechanism

```
...

server.auth.default({
  strategy: 'standard',
});

...
```

- Any attempt to visit a route will be rejected unless valid cookie detected.

- Some routes need to be available (to signup or login for instance)

- These routes must specifically disable auth mechanism

```
...
exports.signup = {
  auth: false,
  handler: function (request, reply) {
    reply.view('signup', { title: 'Sign up for Donations' });
  },

};

exports.login = {
  auth: false,
  handler: function (request, reply) {
    reply.view('login', { title: 'Login to Donations' });
  },

};
...
```

Setting the Cookie

- Set the cookie if correct user credentials presented.

```
request.cookieAuth.set({
  loggedIn: true,
  loggedInUser: user.email,
});
```

```
...
exports.authenticate = {
  auth: false,
  handler: function (request, reply) {
    const user = request.payload;
    if ((user.email in this.users) && (user.password === this.users[user.email].password)) {
      request.cookieAuth.set({
        loggedIn: true,
        loggedInUser: user.email,
      });
      reply.redirect('/home');
    } else {
      reply.redirect('/signup');
    }
  },

};
...
```

Reading the Cookie

- If cookie set, it can be read back in any handler

```
request.cookieAuth.set({
  loggedIn: true,
  loggedInUser: user.email,
});
```

- We are storing logged in users email in this example

```
    const donorEmail = request.auth.credentials.loggedInUser;
```

- Use this email to look up user details in some storage infrastructure (database).

```
exports.donate = {

  handler: function (request, reply) {
    let data = request.payload;
    const donorEmail = request.auth.credentials.loggedInUser;
    data.donor = this.users[donorEmail];
    this.donations.push(data);
    reply.redirect('/report');
  },

};
```

Clearing the Cookie

- Cookie deleted
- Any attempt to access protected routes rejected

```
exports.logout = {
  auth: false,
  handler: function (request, reply) {
    request.cookieAuth.clear();
    reply.redirect('/');
  },

};
```

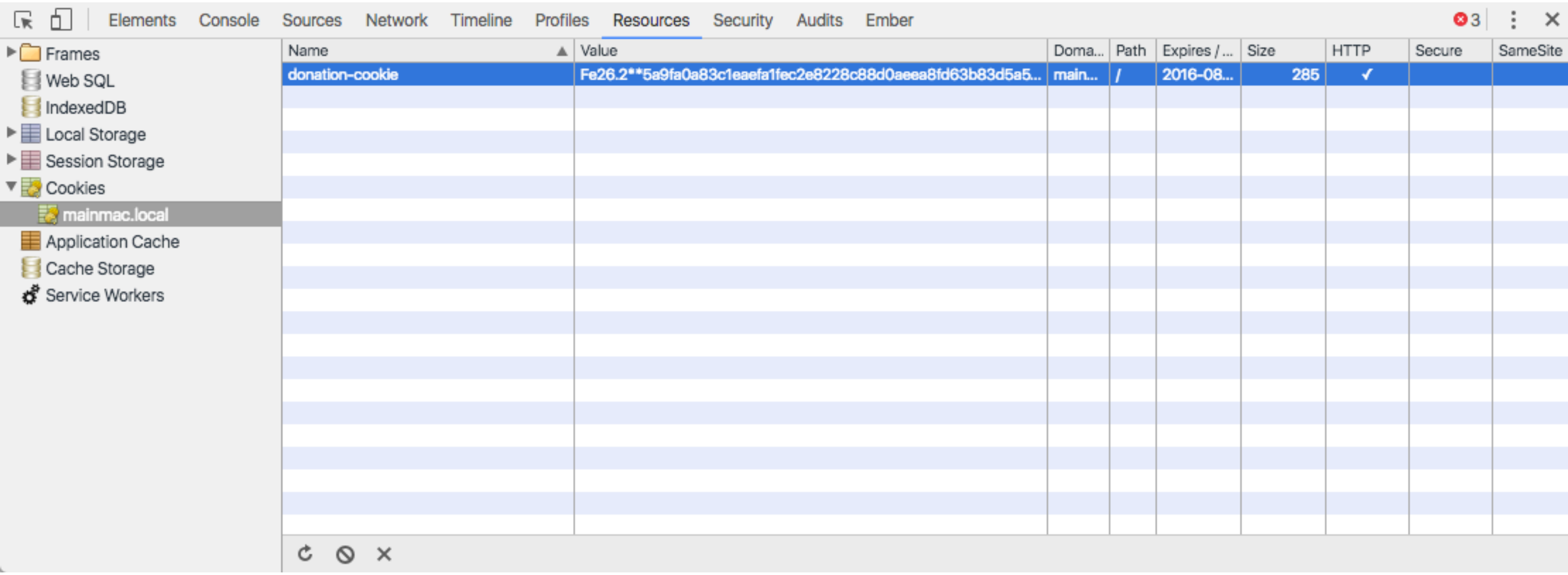
Redirects

If route protected, and cookie deleted/ timeout

Redirect to login

```
server.auth.strategy('standard', 'cookie', {
  password: 'secretpasswordnotrevealedtoanyone',
  cookie: 'donation-cookie',
  isSecure: false,
  ttl: 24 * 60 * 60 * 1000,
  redirectTo: '/login',
});
```

Cookies can be Inspected in Browser



Name	Value	Doma...	Path	Expires / ...	Size	HTTP	Secure	SameSite
donation-cookie	Fe26.2**5a9fa0a83c1eaefa1fec2e8228c88d0aeea8fd63b83d5a5...	main...	/	2016-08...	285	✓		