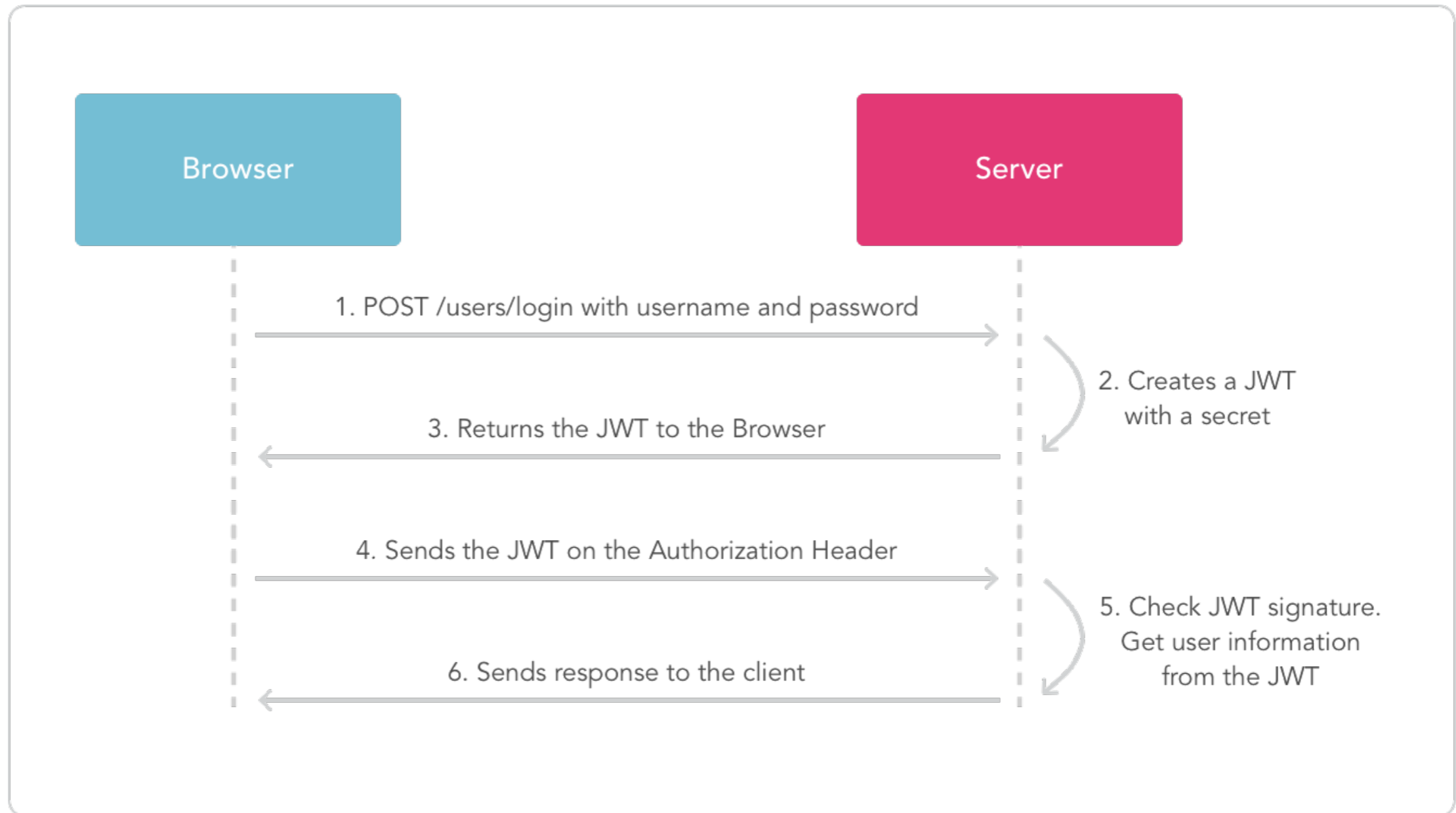
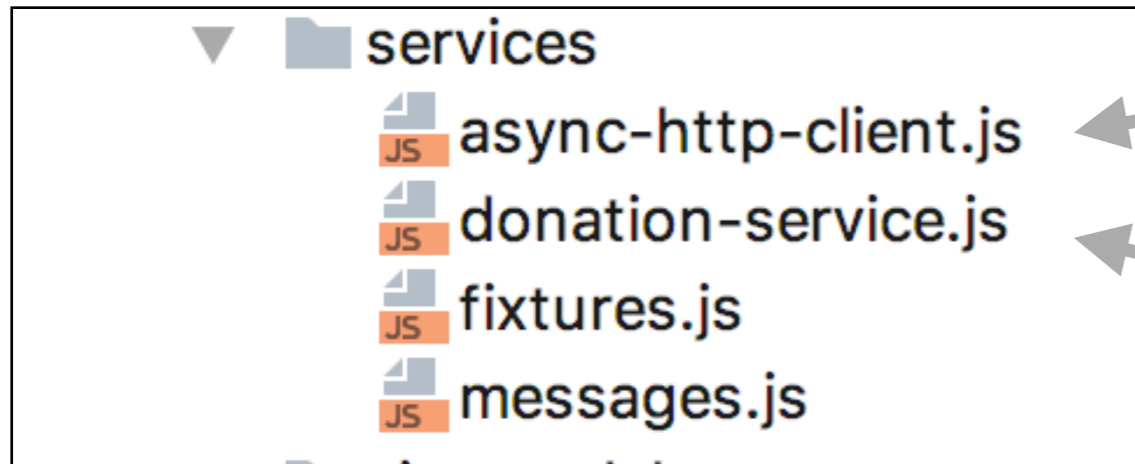


Jwt in Aurelia

JWT Base Authentication



services



Simple wrapper around aurelia-http-client

Uses async-http-client for all API access

+ refactor these classes to accept a JWT on successful authentication

+ ... and transmit the token on every subsequent api access

Refactoring AsyncHttpClient & DonationService

AsyncHttpClient

- Introduce new 'authenticate()' method into AsyncHttpClient
- Retrieve JWT from server (if correct credentials sent)
- Store the token, and send with each subsequent api request
- Clear the Token on logout

DonationService

- Simplify approach
- No longer retrieve list of users
- just invoke 'authenticate()' AsyncHttpClient
- Rely on AsyncHttpClient to generate 'loggedIn/ loggedOut' events

AsyncHttpClient Constructor

- Import
LoginStatus
- Inject
EventAggregator

```
import {inject} from 'aurelia-framework';
import {HttpClient} from 'aurelia-http-client';
import Fixtures from './fixtures';
import {EventAggregator} from 'aurelia-event-aggregator';
import {LoginStatus} from './messages';

@Inject({
  HttpClient,
  EventAggregator
})
export default class AsyncHttpClient {

  constructor(httpClient, fixtures, ea) {
    this.http = httpClient;
    this.http.configure(http => {
      http.withBaseUrl(fixtures.baseUrl);
    });
    this.ea = ea;
  }
  ...
}
```

AsyncHttpClient authenticate()

- Post user credentials to donation-service
- If success, recover JWT and store in localStorage
- Set Authorization header to include JWT for subsequent api calls
- Publish LoginStatus event

```
authenticate(url, user) {
  this.http.post(url, user).then(response => {
    const status = response.content;
    if (status.success) {
      this.http.configure(configuration => {
        configuration.withHeader('Authorization',
                                'bearer ' + response.content.token);
      });
    }
    this.ea.publish(new LoginStatus(status));
  }).catch(error => {
    const status = {
      success: false,
      message: 'service not available'
    };
    this.ea.publish(new LoginStatus(status));
  });
}
```

AsyncHttpClient clearAuthentication()

- Clear the Authorization header

```
clearAuthentication() {  
    this.http.configure(configuration => {  
        configuration.withHeader('Authorization', '');  
    });  
}
```

DonationService - Constructor

donation-client

```
export default class DonationService {  
  
  donations = [];  
  methods = [];  
  candidates = [];  
  users = [];  
  total = 0;  
  
  constructor(data, ea, ac) {  
    this.methods = data.methods;  
    this.ea = ea;  
    this.ac = ac;  
    this.getCandidates();  
    // this.getUsers();  
  }  
}
```

- candidates 'open' route
- getUser 'closed' route

donation-web

```
exports.find = {  
  
  auth: false,  
  
  handler: function (request, reply) {  
    Candidate.find({}).exec().then(candidates => {  
      reply(candidates);  
    }).catch(err => {  
      reply(Boom.badImplementation('error accessing db'));  
    });  
  },  
}
```

```
exports.find = {  
  
  auth: {  
    strategy: 'jwt',  
  },  
  
  handler: function (request, reply) {  
    User.find({}).exec().then(users => {  
      reply(users);  
    }).catch(err => {  
      reply(Boom.badImplementation('error accessing db'));  
    });  
  },  
};
```


DonationService - login/logout

- Login defers to asyncHttpClient
- Logout asks asks asyncHttpClient to clear the JWT, and then broadcasts new status

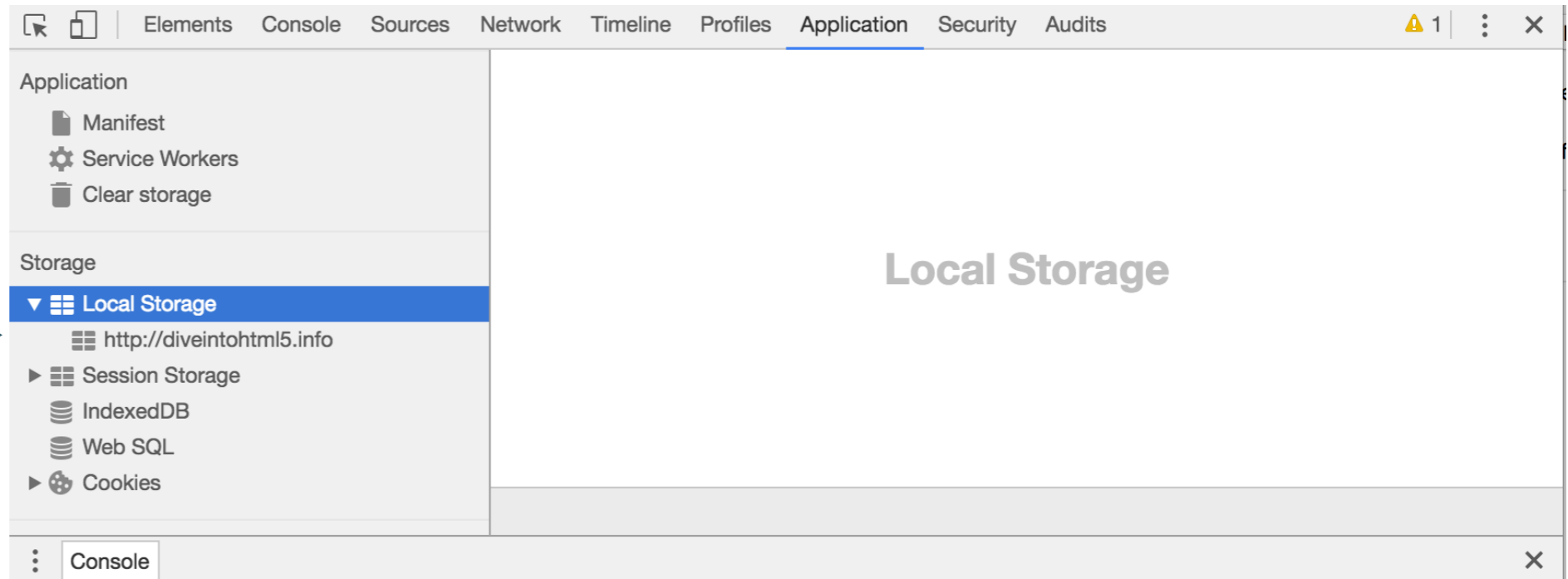
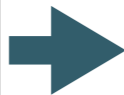
```
login(email, password) {
  const user = {
    email: email,
    password: password
  };
  this.ac.authenticate('/api/users/authenticate', user);
}

logout() {
  const status = {
    success: false,
    message: ''
  };
  this.ac.clearAuthentication();
  this.ea.publish(new LoginStatus(new LoginStatus(status)));
}
```

Local Storage

- A HTML5 standard way for web pages to store named key/value pairs locally, within the client web browser.
- Like cookies, this data persists even after you navigate away from the web site, close your browser tab, exit your browser.
- Unlike cookies, this data is never transmitted to the remote web server (unless you send it manually)

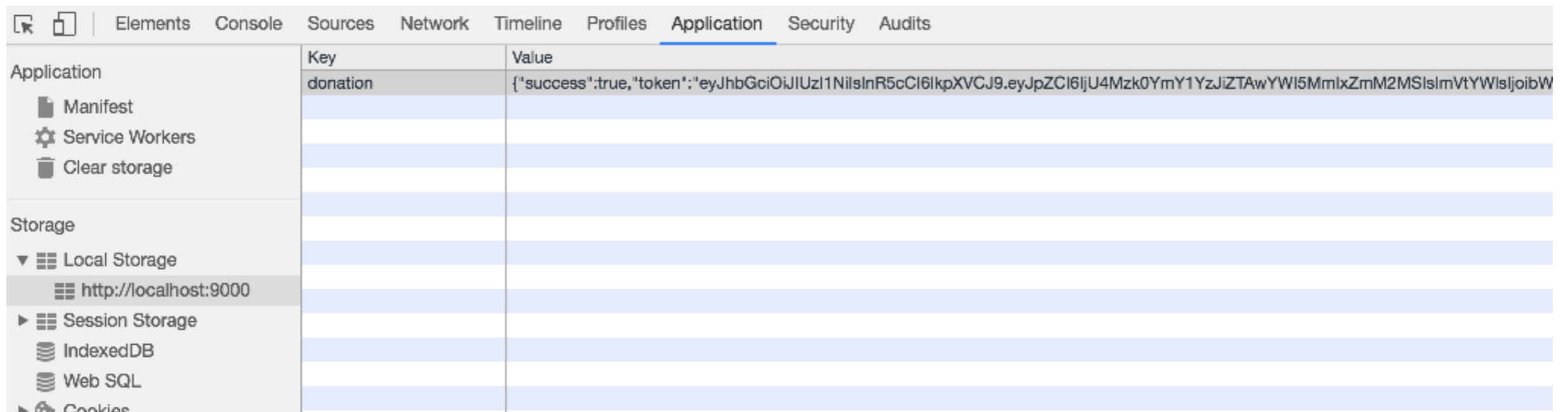
Inspect it using developer tools



Storing Tokens in Local Storage

- donation
name
value pair
created
in local
storage

```
authenticate(url, user) {  
  this.http.post(url, user).then(response => {  
    const status = response.content;  
    if (status.success) {  
      localStorage.donation = JSON.stringify(response.content);  
      this.http.configure(configuration => {  
        configuration.withHeader('Authorization',  
                                'bearer ' + response.content.token);  
      });  
    }  
  });  
  ...  
}
```



The screenshot shows the Chrome DevTools Application tab. The left sidebar is expanded to 'Local Storage' for the URL 'http://localhost:9000'. The main area displays a table with the following data:

Key	Value
donation	{"success":true,"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjU4Mzk0YmY1YzJiZTAwYWI5Mm1xZmM2MSIsImVtYWlsIjoibW"}

Check LocalStorage for Tokens

- If token is found:
- set the token as an 'Authorization' header for subsequent api requests

```
isAuthenticated() {  
  let authenticated = false;  
  if (localStorage.donation !== 'null') {  
    authenticated = true;  
    this.http.configure(http => {  
      const auth = JSON.parse(localStorage.donation);  
      http.withHeader('Authorization', 'bearer ' + auth.token);  
    });  
  }  
  return authenticated;  
}
```

On App Startup...

- Check to see if token is present...
- ... if it is, bypass login/signup routes and go straight to 'home' router

```
export class App {  
  ...  
  
  attached() {  
    if (this.ds.isAuthenticated()) {  
      this.au.setRoot('home').then(() => {  
        this.router.navigateToRoute('dashboard');  
      });  
    }  
  }  
}
```