THE UNIVERSITY OF WESTERN AUSTRALIA

Topic 15: Authentication

CITS3403 Agile Web Development

Getting MEAN with Mongo, Express, Angular and Node, Chapter 11

Semester 1, 2018

Secure web apps

THE UNIVERSITY OF WESTERN AUSTRALIA

- Security is a primary concern for anyone developing web applications.
- Data access must be controlled, passwords must be validated securely, and users just be able to trust the information presented to them.
- Complete security is very hard to achieve and beyond the scope of this unit, but basic authentication is relatively easy.
- An interesting case study of internet security is anonymous' attack on HBGary:

arstechnica.com/tech-policy/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack/

Web security basics

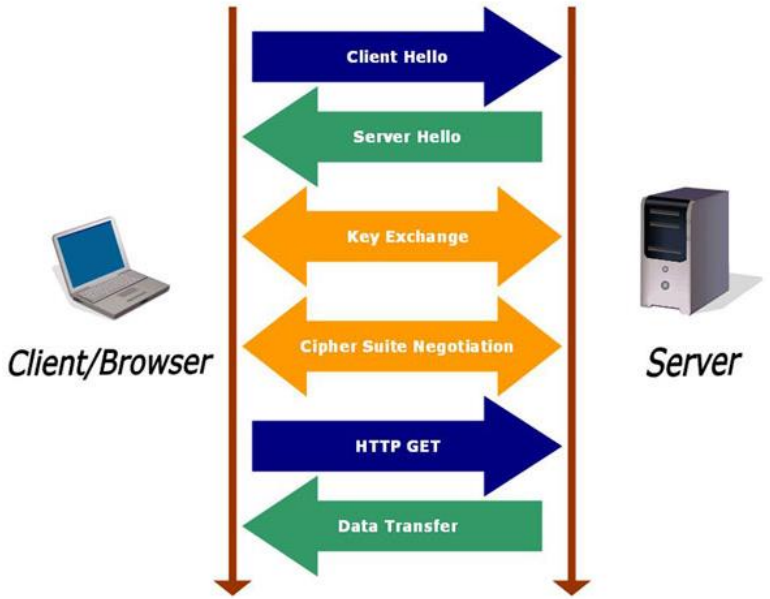
THE UNIVERSITY OF WESTERN AUSTRALIA

Web security makes use of the following basic concepts

- Public Key Encryption (eg RSA)
  - A public-private key is 2 functions *pub* and *priv* so that *x = priv(pub(x))* and given that you know *pub*, *priv* is hard to work out.
  - Public Key Encryption can be used for authentication. I can compute and publish *pub(x)* and only someone who knows *priv* can tell me what *x* is.
  - Public Key Encryption can be used for digital signatures. The pair *(x, priv(x))* can be verified by anyone, but only created by some who knows *priv.*
  - Key distribution. A random key *x* can be generated and *pub(x)* can be sent to someone who knows *priv*. Then the pair knows *x,* but no body else does (even if they have been eaves dropping
- Hashing (eg MD5)
  - Secure hashing computes a large number from a stream of data, in such a way that it's very difficult to fabricate data with a certain hash.
  - Different to hashing used for Hash tables etc.

Secure web session

- HTTP is stateless, so the server does not remember the client.
- For a secure session, every request needs to be authenticated... thankfully there are protocols to help here.
- SSL (secure sockets layer) wraps up the public key encryption process to enable a secure transaction.

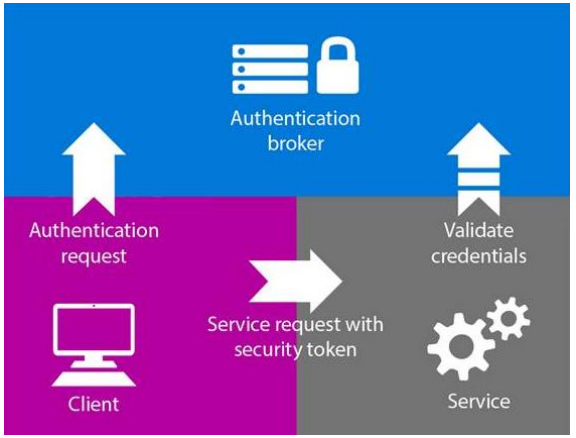


Cookies and Tokens

- Web session security is managed through cookies and tokens.
- Cookies are packets of data stored in the browser.
  - Session cookies can record a users interaction with a site, persistent remain in your browser and allow sites to track your browsing habits.
  - Cookies consist of a name, a value and a set of attribute value pairs (e.g. expiration).
  - Cookies can be created and managed through javascript: `document.cookie="trackme: false";`
  - Cookies are sent from the server to the browser:

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123; Expires=Wed, 09 Jun 2021 10:18:14 GMT
…
```

- Authentication tokens allow you to store user privaleges in JWT, (JSON web tokens)

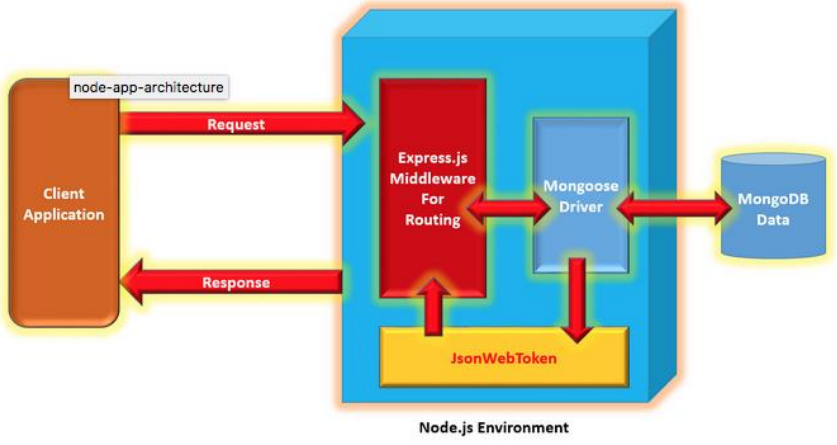


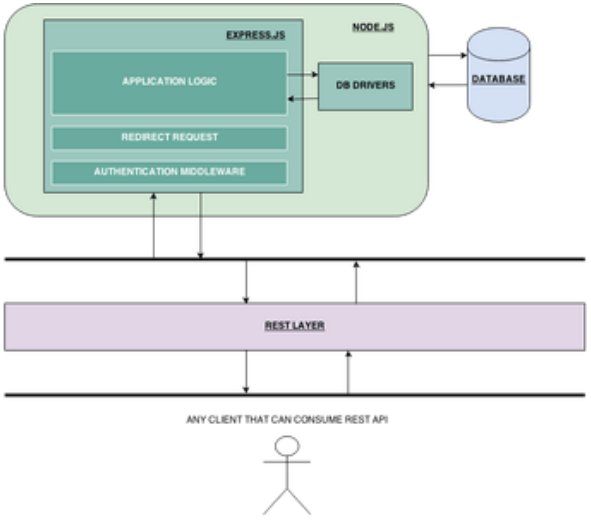
Authentication and session management

To manage users, a mongo collection can store user data and password hashes. Unverified users are required to enter login details before a secure session commences.

There are a number of good tutorials for incorporating security into your app

- http://www.dotnetcurry.com/nodejs/1302/nodejs-token-based-authentication-security
- http://thejackalofjavascript.com/architecting-a-restful-node-js-app/





Adding authentication to MEAN apps

THE UNIVERSITY OF WESTERN AUSTRALIA

- There are some useful packages to handle authentication. Passport is a popular option with many strategies, including authentication through Google and Facebook.
- We'll look at using the passport-local which stores the authentication details in a local mongo database.
- This is taken from the web tutorial http://mherman.org/blog/2015/01/31/local-authentication-with-passport-and-express-4

Elements of Web-security

THE UNIVERSITY OF WESTERN AUSTRALIA

- Web security depends on trust. There are several elements to this:

1. The web server needs to be confident that someone accessing data is authorised.
2. The user needs to know that the site they are visiting is the one they intend to.
3. Both the server and the client need to be confident that no one in the middle is accessing unauthorised data.

- 2 is typically handled by browsers, and 3 is achieved with https. We'll focus on 1.

Authentication Strategy

- To track a users identity we need to have them register so we can associate a user name with them.
- When someone uses an application a *session* is maintained via a variable held by the web-browser.
- When someone logs in they provide a password. This is salted and hashed to provide a digest which can compared to a hash in a database (keeping the password secure).
- Once the user is authenticated, they will be be served there requested pages, and their id will be a parameter of the requests.

Steps to implement security:

- Install the npom packages: `passport-local`, `express-session` and `passport-local-mongoose`
- In app.js we need to include and initialise passport.
- The express-session stores data in the backend away from the user.

```
1  var express = require('express');
2  var path = require('path');
3  var favicon = require('serve-favicon');
4  var logger = require('morgan');
5  var cookieParser = require('cookie-parser');
6  var bodyParser = require('body-parser');
7  //for passport
8  var passport = require('passport');
9  var LocalStrategy=  require('passport-local').Strategy;
10
11
12
13 var routes = require('./routes/index');
14 var users = require('./routes/users');
15
16 var app = express();
17
18 // view engine setup
19 app.set('views', path.join(__dirname, 'views'));
20 app.set('view engine', 'jade');
21
22 // uncomment after placing your favicon in /public
23 //app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
24 app.use(logger('dev'));
25 app.use(bodyParser.json());
26 app.use(bodyParser.urlencoded({ extended: false }));
27 app.use(cookieParser());
28 //for passport
29 app.use(require('express-session')({
30       secret: 'CITS3403',
31       resave: false,
32       saveUninitialized: false
33 }));
34 app.use(passport.initialize());
35 app.use(passport.session());
36 //end for passport
37 app.use(express.static(path.join(__dirname, 'public')));
38
39 app.use('/', routes);
40
41 //passport config
42 var Account = require('./models/account');
43 passport.use(new LocalStrategy(Account.authenticate()));
44 passport.serializeUser(Account.serializeUser());
45 passport.deserializeUser(Account.deserializeUser());
```

A model for user accounts:

- We can create a account model using mongoose. Including passport-local-mogoose will take care of hashing and salting:

```
1 var mongoose = require('mongoose');
2 var Schema = mongoose.Schema;
3 var passportLocalMongoose = require('passport-local-mongoose');
4
5 var Account = new Schema({
6       username: String,
7         password: String
8 });
9
10 Account.plugin(passportLocalMongoose);
11
12 module.exports = mongoose.model('Account', Account);
```

Adding Routes

- The login, register and index pages all require get and post actions so these need to be added to the routes.
- A controller should really be used to group the call-backs in a single js file

```
var express = require('express');
//new
var passport = require('passport');
var Account = require('../models/account');
//old
var router = express.Router();
var ctrlPerson = require('../controllers/person');

/* GET home page. */
router.get('/', ctrlPerson.personList);

/* add simple home page. */
router.post('/', ctrlPerson.newPerson);

/* delete simple */
router.get('/delete/:id', ctrlPerson.deletePerson);

router.get('/register', function(req, res) {
      res.render('register', { });
});

router.post('/register', function(req, res) {
      Account.
        register(new Account({ username : req.body.username }),
                 req.body.password,
                 function(err, account) {
                   if (err) {
                     return res.render('register', { account : account });
                   }
                   passport.authenticate('local')(req, res, function () {
                     res.redirect('/');
                     });
                 });
});

router.get('/login', function(req, res) {
      res.render('login', { user : req.user });
});

router.post('/login', passport.authenticate('local'), function(req, res) {
      res.redirect('/');
});

router.get('/logout', function(req, res) {
      req.logout();
          res.redirect('/');
});

module.exports = router;
```

Jade views

- The Jade views can now be rendered differently depending on whether a user is logged in:

```
1  extends layout
2
3  block content
4    if (!user)
5      a(href="/login") Login
6    br
7    a(href="/register") Register
8      if (user)
9        p You are currently logged in as #{user.username}
10       a(href="/logout") Logout
~
~
```

```
1  extends layout
2
3  block content
4    h1 Simple
5
6    if (!user)
7      a(href="/login") Login
8    br
9    a(href="/register") Register
10
11   if (user)
12     p You are currently logged in as #{user.username}
13     a(href="/logout") Logout
14
15     mixin email(addr)
16       a(href='mailto://'+ addr)= addr
17
18     table
19       tr
20         th Name
21         th Age
22         th Email
23         th Delete?
24       each p in people
25         tr
26           td #{p.name}
27           td #{p.age}
28           td
29             - var e = p.email;
30             +email(e)
31           td
32             a(href='/delete/'+p._id) Delete
33
34      form(action='/', method='post')
35        label(for='name') Name
36        input(id = 'name' , type='text', name='name')
37        label(for='age') Age
38        input(id='age', type ='number', name='age')
39        label(for='email') Email
40        input(id='email', type ='email', name='email')
41        input(type='submit', value='Submit')
~
```

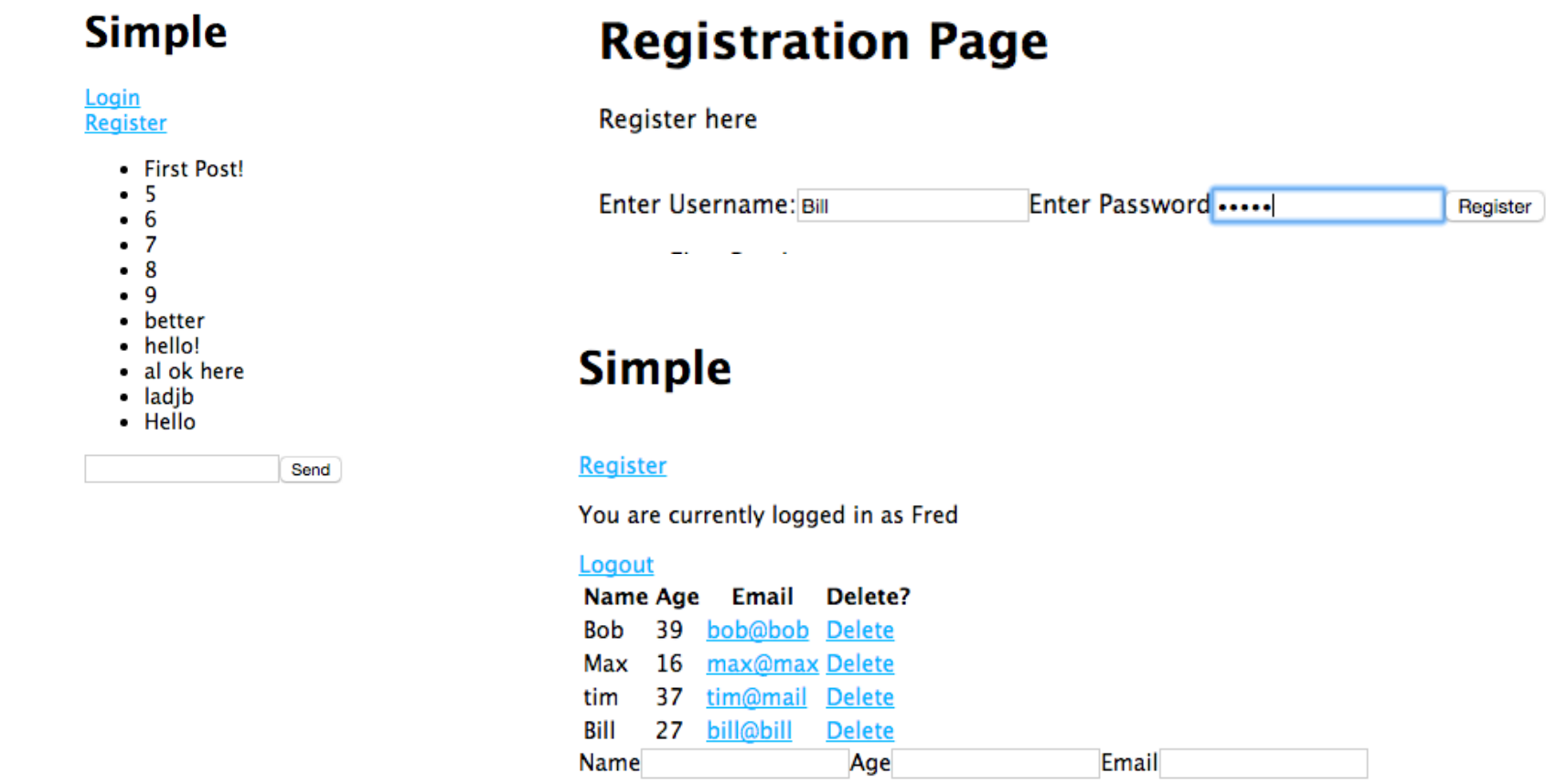
Login and register pages

- These can now be implemented as simple forms

```
1 extends layout
2
3 block content
4   .container
5     h1 Login Page
6     p Login to the simple app here.
7     br
8     form(action="/login",method="post")
9       label(for='username') Enter username:
10      input(id='username', type = 'text', name = 'username')
11      label(for='password') Enter password:
12      input(id='password', type = 'password', name = 'password')
13      input(type='submit', value='Submit')
```

```
1 extends layout
2
3 block content
4   .container
5     h1 Registration Page
6     p Register here
7     br
8     form(action="/register",method="post")
9       label(for='username') Enter Username:
10      input(id='username', type = 'text', name = 'username')
11      label(for='password') Enter Password
12      input(id='password',type ='password', name='password')
13      input(type='submit', value='Register')
~
```

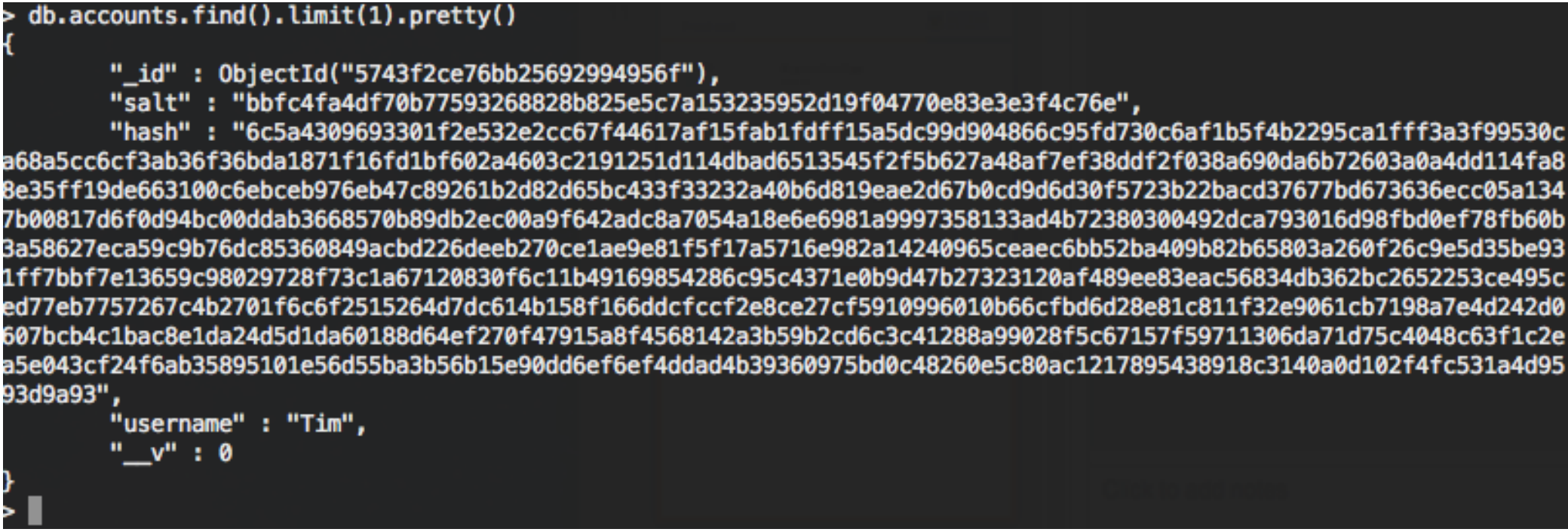
Front end:


Simple
Login
Register
First Post!
5
6
7
8
9
better
hello!
al ok here
ladjb
Hello
Send
Registration Page
Register here
Enter Username: Bill
Enter Password
Register
Simple
Register
You are currently logged in as Fred
Logout
Name Age Email Delete?
Bob 39 bob@bob Delete
Max 16 max@max Delete
tim 37 tim@mail Delete
Bill 27 bill@bill Delete
Name
Age
Email

Mongo Backend

THE UNIVERSITY OF WESTERN AUSTRALIA

- You can see how the account data is held via the Mongo Shell:



Next lecture….

THE UNIVERSITY OF WESTERN AUSTRALIA

- Testing and Agile development….