Milestone 3:

Modules information and testing plan

Version 1.1

In Preparation for:

Computer Science 4770

Authored by:

Kristan Hart
Keir Strickland-Murphy
Diego Zuluaga
Meishang Chen
Aizaz Iqbal

Table of Contents

Test Plan Identifier

MUNSSN testing 1.0 MTP1
The structure of this documents is primarily based on the IEEE 829 -1983 standard for software Test Documentation.

Test Items

The scope of this testing activity will include the following modules:

1.1 Requirements--version 1.0
1.2 Server--version1.0

1.3 Route --version1.0
 Request handling(Render Template)
 Information handling(User Input)

1.4 Authentication--version 1.0
1.5 Database--version 1.0
1.6 Schema--version 1.0
1.7 Template--version 1.0

Reference:
SRS requirement Documents
https://github.com/Muroz/TestProject/blob/master/Documentation/CS4770TeamProjectRequirementDocument.pdf

Architecture Document
https://github.com/Muroz/TestProject/blob/master/Documentation/ArchitectureDocument.pdf

1. Unit testing criteria

1.1 Requirement unit test:

Tested by: Diego
Test type: Black box testing
Test case name : UTRQ01

Test case description: Test that the node modules are downloaded when they are required

Prerequisites:
1. Access to the server folder
2. Server folder without the node_modules folder
3. Access to a terminal
4. List of all the requirements used in the app

Test data:
1. Path to the node_modules folder in the server location

Test Scenario: Verify that the server is running on the given port with the output on the terminal when the server is running

Steps	Step details	Expected result	Actual result	Pass/Fail
1.	Run the npm install command on the server through the terminal	The terminal will display	As expected	Pass
2.	Check the node_modules folder	All of the listed modules should appear	As expected	Pass

1.2 Server unit test:

Tested by: Diego
Test type: Black box testing
Test case name : UTSV01

Test case description: Test that the server runs consistently on the given port

Prerequisites:
1. Access to a browser
2. Access to a terminal
3. A port available to be used by the server(port 8080 for this test)

Test data:
No test data required for this test

Test Scenario: Verify that the server is running on the given port with the output on the terminal when the server is running

Steps	Step details	Expected result	Actual result	Pass/Fail
1.	Run the server on the terminal	The command runs properly	As expected	Pass
2.	Check the output given by the terminal after the server is running	The output should be a string that says that “The server is running on port 8080”	As expected	Pass

1.3 Routes unit test:

1.3.1 Load page unit test:

Tester Name: Meishan
Test Case ID UTRV01
Test Type: Black Box

Test Case Description: Testing that the routes direct and render the page properly

Prerequisites:
1. Access to a browser
2. Server is running
3. active account

Test Data:
1. Login template
2. Signup template

Test Scenario Tester opens browser and enter host domain, click different links on index page

Step	**Step Detail**	**Expected Result**	**Actual Result**	**Pass/fail**
1	Enter application domain in browser	index page should be render	as expected	Pass
2	Click signup link	sign up page should be render	as expected	pass
3	Click login link	login page should be render	as expected	pass
4	enter username and password			
5	click submit	profile page should be render	as expected	pass

1.3.2. Sign up unit test:

Tested by: Meishang
Test type: Black box testing
Test case name : UTRV02

Test Case Description: Testing that the routes properly receive and handle the user signup input

Prerequisites:
1. access to a browser
2. a running server

Test Data:
1. First name: Meishang Last Name: Chen 2.
Username: mc2882@mun.ca
3. password: 12345678

Test Scenario: User wants to sign up on an account in MUNSSN, so user enter test data in signup form, after click signup, all information should be display on terminal

Step	Step Detail	Expected Result	Actual Result	Pass/fail
1	navigate to signup page	sign up page should be render	as expected	Pass
2	enter all test data	page should display the information	as expected	Pass
3	click submit	email confirmation message should send to user's email and test information should be display on terminal as console log	as expected	Pass

1.3.3. Picture uploading unit test:

Tested by: Diego
Test type: Black box testing
Test case name : UTRV03

Test Case Description: Testing that the routes properly save the uploaded picture on the given address

Prerequisites:
1. access to a browser
2. a running server
3. a HTML page with the capability of submitting files
4. Running terminal

Test Data:
1. Path : local address in which pictures should be saved
2. Image to be uploaded

Test Scenario: User wants to upload a picture on an account in MUNSSN

Step	**Step Detail**	**Expected Result**	**Actual Result**	**Pass/fail**
1	Open the web page	Page should render	as expected	Pass
2	Choose the picture to be uploaded	Picture can be chosen without problem	as expected	Pass
3	Click submit	The terminal should display that the picture was saved on the path given at first	as expected	Pass

1.4. Authentication unit test:

Tested by: Diego
Test type: Black box testing
Test case name : UTAU01

Test case description: Test the login and session management (authentication module) of the software

Prerequisites:
1. Access to a browser
2. A running server that hosts the application
3. A login page (references in this document) and a profile page that displays the credentials passed in from the user
4. A database running in which the test data is already present (which means the user has signed up already and activated the account)
5. A user schema to define the structure of the user information in the database

Test data:
1. Username : dazc35@mun.ca
2. Password : chocOASomething

Test Scenario: Verify that the user can log in given the right information and the credentials will be kept as long as he remains online, meaning he only has to
log in once per session

Steps	Step details	Expected result	Actual result	Pass/Fail
1.	Navigate to the login page, which in this case is localhost:8080	Page should be rendered	As expected	Pass
2.	Enter the test data	Credentials can be entered	As expected	Pass
3.	Submit the test data	The user should be directed to the profile page	As expected	Pass
4.	Check the credentials	The information displayed should be the same as the test data	As expected	Pass

1.5. Database unit test:

Tested by: Kris
Test type: Black box testing
Test case name : UTDB01

Test case description: Test that the database runs correctly.

Prerequisites:
1. Connection achieved to database server
2. Save data to the database
3. Remove data from the database

Test data: The path to the data files

Test Scenario: Verify that the database is running and can save and remove data

Steps	Step details	Expected result	Actual result	Pass/Fail
1.	Run the MongoDB server	Connected completed	As Expected	Pass
2.	Insert data into the database through the terminal	Data saved successfully	As Expected	Pass
3.	Remove the data from the database through the terminal	Data removed successfully	As Expected	Pass

1.6. Schema unit test:

Tested by: Kris
Test type: Black box testing
Test case name : UTSC01

Test case description: Test that the schema runs smoothly

Prerequisites:
1. Connection achieved to server
2. Insert data successfully
3. Check the collections with a find command
4. Insert data unsuccessfully
5. Check the collections with a find command

Test data: Friend Schema

Test Scenario: Verify that the Schema is correct and users can be added into the database via a schema

Steps	Step details	Expected result	Actual result	Pass/Fail
1.	Connect to the schema	Connected completed	As Expected	Pass
2.	Insert user into the schema	Data saved successfully	As Expected	Pass
3.	Insert user unsuccessfully	Data inserted unsuccessfully	As Expected	Pass

1.7. Template unit test:

Tested by: Aizaz
Test type: Black box testing
Test case name : UTT01

Test case description: Test that the pages render correctly and function on the server

Prerequisites:
1. A functional server with the pages loaded onto it
2. A set of functional routes for the pages

Test data:

Test Scenario:

Steps	Step details	Expected result	Actual result	Pass/Fail
1.	Open the web page in a browser.	The page will be displayed correctly	As expected	Pass
2.	Check that when something is input into any given input box it works correctly	All input bars correctly take inputs	As expected	Pass
3.	Check that all buttons correctly manipulate page	All buttons correctly send information from input bars	As expected	Pass

1.7.1 Showing Friend unit test:

Tested by: Aizaz
Test type: Black box testing
Test case name : AFUT01

Test case description: Test that the page shows that the user has the ability to add friends

Prerequisites:
1. User has active account
2. There is another user to add to friends list

Test data:

Test Scenario:

Steps	Step details	Expected result	Actual result	Pass/Fail
1.	Select the friend to be added and send invitation.	The correct user will receive request	As expected	Pass
2.	The person who recived the request sees and accepts the request.	The request is displayed properly and is accepted correctly	As expected	Pass

1.7.2 Posting to timeline unit test:

Tested by: Aizaz
Test type: Black box testing
Test case name : UTPT01

Test case description: Test that the page shows that the user has the ability to add posts

Prerequisites:

1. User has active account

Test data:

Test Scenario:

Steps	Step details	Expected result	Actual result	Pass/Fail
1.	The user enters their desired text	The text is received correctly	As expected	Pass
2.	The user uploads a picture to their post	Picture uploads successfully	As expected	Pass
3.	Permissions for the post are set.	The permission is the set and shows visibility correctly	As expected	Pass
4.	Post is displayed correctly	The post is visible on the active site.	As expected	Pass

2 Testing code

2.1 Server/Requirements Test Code:

```
//set up

// require all the tools needed
var express = require('express');
var app = express();
var port = 8080;
var mongoose = require('mongoose');
var passport = require('passport');
var flash = require('connect-flash');
var path = require('path');

var morgan = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var session = require('express-session');

var configDB = require('./config/database.js');

// configuration



mongoose.connect(configDB.url); // connect to our database

require('./config/passport')(passport); // pass passport for configuration

//set up the express application

app.use(express.static(path.join(__dirname,'public')));

app.use(morgan('dev')); // log every request to the console
app.use(cookieParser()); // read cookies (needed for auth)
app.use(bodyParser()); // get information from html forms

app.set('view engine', 'ejs'); // set up ejs for templating

// required for passport
app.use(session({ secret: 'ilovevodkavodkavodkavodka' })); // 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

//routes
require('./app/routes.js')(app,passport); // load our routes and pass in our app and fully configure
passport

// launch
app.listen(port);
console.log('The magic happens on port ' + port);
```

2.2 Authentication Test Code:

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

// load up the user model
var User            = require('../app/model/user').User;

// expose this function to our app using module.exports
module.exports = function(passport) {

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

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

    // used to deserialize the user
    passport.deserializeUser(function(id, done) {
        User.findById(id, function(err, user) {
            done(err, user);
        });
    });

    //
=========================================================================
=
    // LOCAL SIGNUP
============================================================
   //
==========================================================================
=
   // we are using named strategies since we have one for login and one for signup
   // by default, if there was no name, it would just be called 'local'

         passport.use('local-login', new LocalStrategy({
      // by default, local strategy uses username and password, we will override with email
      usernameField : 'email',
      passwordField : 'password',
      passReqToCallback : true // allows us to pass back the entire request to the callback
   },
   function(req, email, password, done) { // callback with email and password from our form

      // find a user whose email is the same as the forms email
      // we are checking to see if the user trying to login already exists

      User.findOne({ 'local.email' :  email }, function(err, user) {
         // if there are any errors, return the error before anything else
         if (err)
            return done(err);

         // if no user is found, return the message
         if (!user)
            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 (!user.validPassword(password))
            return done(null, false, req.flash('loginMessage', 'Oops! Wrong
password.'+user.local.active)); // create the loginMessage and save it to session as flashdata

         // if the user's account is not activate
         if(!user.local.active)
             return done(null,false,req.flash('loginMessage', 'Account is not actve'));

         // all is well, return successful user
         return done(null, user);
      });

   }));
   function checkPassword(str)
  {
```

```
  var re = /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}$/;
  return re.test(str);
};

function checkUserPassword(username,password)
{
  if(username == "") {
    //alert("Error: Username cannot be blank!");
    return false;
  }
  re = /^\w+$/;
  if(!re.test(username)) {
    //alert("Error: Username must contain only letters, numbers and underscores!");
    //form.username.focus();
    return false;
  }
  if(password != "") {
    if(!checkPassword(password)) {
      //alert("The password you have entered is not valid!");
      return false;
    }
  } else {
   // alert("Error: Please check that you've entered and confirmed your password!");
   // form.pwd1.focus();
    return false;
  }
  return true;
};
  passport.use('local-signup', new LocalStrategy({
      // by default, local strategy uses username and password, we will override with email
      usernameField : 'email',
      passwordField : 'password',
      passReqToCallback : true // allows us to pass back the entire request to the callback
  },
  function(req, email, password, done) {

      // asynchronous
      // User.findOne wont fire unless data is sent back
      process.nextTick(function() {

      // find a user whose email is the same as the forms email
      // we are checking to see if the user trying to login already exists
      User.findOne({ 'local.email' :  email }, function(err, user) {
            // if there are any errors, return the error
            if (err)
                return done(err);

            // check to see if theres already a user with that email
            if (user) {
                return done(null, false, req.flash('signupMessage', 'That email is already taken.'));
            } else {

                // if there is no user with that email
                // create the user
                var newUser            = new User();
               // if(checkUserPassword(email,password)){
                // set the user's local credentials
                newUser.local.email    = email;
                newUser.local.password = newUser.generateHash(password);

                // save the user
                newUser.save(function(err) {
                    if (err)
                        throw err;
                    return done(null, newUser);
                });

            // else{
            //     return done(null, false, req.flash('signupMessage', 'The password you have entered
is not valid!'));
            // }
            }


        });

        });

    }));

};
```

2.3 Route Test Code:

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


  var friends = require("mongoose-friends");
  var Status = require("mongoose-friends").Status;
  var User = require('../app/model/user').User;
  var Post =  require('../app/model/posts').Post;
  var nodemailer = require("nodemailer");
  var smtpTransport = require("nodemailer-smtp-transport")
  var mailOptions,host,link;
  /* SMTP server */



  var smtpTransport = nodemailer.createTransport(smtpTransport({
    host: "smtp.gmail.com",
    secureConnection : false,
    port: 587,
    auth:{
        user: "multiculturalteam67@gmail.com",
        pass: "qwe12332"
      }
    }));

  //home page
  app.get('/', function(req, res) {
    res.render('index.ejs'); // load the index.ejs file
  });

  //login
```

```
    // show the login form
    app.get('/login', function(req, res) {
        // render the page and pass in any flash data if it exists
        res.render('login.ejs', { message: req.flash('loginMessage') });
    });

    // process the login form
    // app.post('/login, do all our passport stuff here);

    //signup

    //show the signup form
    app.get('/signup', function(req,res){

        //render the page and pass in any flash data if it exists
        res.render('signup.ejs', { message: req.flash('signupMessage') });
    });


app.post('/requestFriend', isLoggedIn, function(req, res){
            var friendToRequest = req.body.email;
            var currentUserId = req.user;
            currentUserId.friendRequest(friendToRequest, function (err, request) {

                console.log('request', request);

            });
        });

        app.post('/acceptRequest', isLoggedIn, function (req, res){
            var friendToAdd = req.body.email;
            var currentUserId = req.user;
```

```
                currentUserId.acceptRequest(friendToAdd, function (err, request) {

                    console.log('request', request);
                });
        });

        app.post('/denyRequest', isLoggedIn, function (req, res){
                var friendToDeny = req.body.email;
                var currentUserId = req.user;
                currentUserId.denyRequest(friendToDeny, function (err, denied) {

                    console.log('denied', denied);
                    // denied 1
                });
        });

app.post('/post', isLoggedIn, function(req,res, done){
    console.log(req.body.message);
    console.log(req.body.email);
    User.findOne({'local.email':req.body.email}, function(err, u){
    console.log('its here');
    var date = new Date();
    var current_date = date.getDate();
    var newPost = new Post();
    newPost.postby = req.user._id;
    newPost.postto = u._id
    newPost.body = req.body.message;
    newPost.date = current_date;
    newPost.save(function(error) {
    if (!error) {
        res.redirect(req.get('referer'));
       return done(null, newPost);
  }
```

```
        });});

    });



        // process the signup form
        // app.post('/signup', do all our passport stuff here);



        // profile section

        // 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.ejs',{
                user: req.user // get the user out of session and pass to template
            });
        });



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

        app.get('/profile-temp', isLoggedIn, function(req,res){

            if(req.user.local.active){
                res.render('profile.ejs', {
                });}
                else {
                    res.render('profile-temp.ejs', {});
                host = req.get('host');
                link ="http://"+req.get('host')+"/verify?id="+req.user._id;
                mailOptions ={
                  from:"multiculturalteam67@gmail.com",
                  to :req.user.local.email,
                  subject:"Please confirm your Email Account",
                  html:"Hello, <br> Please Click on the link to verify your email. <br><a href
="+link+">Click here to verify</a>"
                 }
                 console.log(mailOptions);
                 smtpTransport.sendMail(mailOptions,function(error, resoonse){
                  if (error){
                    console.log(error);
                  }else{
                    console.log("Message sent: " + res.message);
                  }
                 });
                }
              });


  app.get('/verify',function(req,res,done){
  console.log(req.protocol+":/"+req.get('host'));
  console.log("Domain is matched. Information is from Authentic email");
    User.findById(req.query.id, function(err, user) {

       if(err){
         console.log(err)
        res.end("<h1>Request is from unknown source");
       }

      user.local.active = true
      console.log("<h1>Email "+user.local.email+" is been Successfully verified");
```

```
            user.save(function(err) {
                    if (err)
                        throw err;
                    return done(null, user);
                });
            res.render('index.ejs', {});
        });

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

    // process the login form
    app.post('/login', passport.authenticate('local-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 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('/');
}
```

2.4 Database Test Code:

2.4.1 Database Unit Test:

2.4.1.1 Making the initial connection to the Mongo database:

```
mongod --dbpath /home/kris/workspace/TestDatabase/test
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten] MongoDB starting : pid=18347
port=27017 dbpath=/home/kris/workspace/TestDatabase/test 64-bit host=kris-ThinkPad-T420
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten] db version v3.4.1
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten] git version:
5e103c4f5583e2566a45d740225dc250baacfbd7
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.2g
1 Mar 2016
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten] allocator: tcmalloc
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten] modules: none
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten] build environment:
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten]     distmod: ubuntu1404
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten]     distarch: x86_64
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten]     target_arch: x86_64
2017-03-08T17:46:10.096-0330 I CONTROL  [initandlisten] options: { storage: { dbPath:
"/home/kris/workspace/TestDatabase/test" } }
2017-03-08T17:46:10.121-0330 I -        [initandlisten] Detected data files in
/home/kris/workspace/TestDatabase/test created by the 'wiredTiger' storage engine, so setting
the active storage engine to 'wiredTiger'.
2017-03-08T17:46:10.121-0330 I STORAGE  [initandlisten]
2017-03-08T17:46:10.121-0330 I STORAGE  [initandlisten] ** WARNING: Using the XFS
filesystem is strongly recommended with the WiredTiger storage engine
2017-03-08T17:46:10.121-0330 I STORAGE  [initandlisten] **          See
http://dochub.mongodb.org/core/prodnotes-filesystem
2017-03-08T17:46:10.121-0330 I STORAGE  [initandlisten] wiredtiger_open config:
create,cache_size=4429M,session_max=20000,eviction=(threads_max=4),config_base=false,st
atistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(
close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] ** WARNING: Access control is not
enabled for the database.
```

```
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] **          Read and write access to
data and configuration is unrestricted.
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] **        We suggest setting it to
'never'
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] **        We suggest setting it to
'never'
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
2017-03-08T17:46:10.426-0330 I FTDC     [initandlisten] Initializing full-time diagnostic data
capture with directory '/home/kris/workspace/TestDatabase/test/diagnostic.data'
2017-03-08T17:46:10.426-0330 I NETWORK  [thread1] waiting for connections on port 27017
2017-03-08T17:46:14.010-0330 I NETWORK  [thread1] connection accepted from
127.0.0.1:34992 #1 (1 connection now open)
2017-03-08T17:46:14.011-0330 I NETWORK  [conn1] received client metadata from
127.0.0.1:34992 conn1: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB
Internal Client", version: "3.4.1" }, os: { type: "Linux", name: "LinuxMint", architecture: "x86_64",
version: "18.1" } }
2017-03-08T17:46:27.636-0330 I NETWORK  [thread1] connection accepted from
127.0.0.1:35006 #2 (2 connections now open)
2017-03-08T17:46:27.642-0330 I NETWORK  [conn2] received client metadata from
127.0.0.1:35006 conn2: { driver: { name: "nodejs", version: "2.2.24" }, os: { type: "Linux", name:
"linux", architecture: "x64", version: "4.4.0-53-generic" }, platform: "Node.js v7.3.0, LE,
mongodb-core: 2.1.8" }
2017-03-08T17:46:27.665-0330 I -        [conn2] end connection 127.0.0.1:35006 (2 connections
now open)
2017-03-08T17:46:51.306-0330 I -        [conn1] end connection 127.0.0.1:34992 (1 connection
now open)
2017-03-08T17:46:52.093-0330 I NETWORK  [thread1] connection accepted from
127.0.0.1:35016 #3 (1 connection now open)
2017-03-08T17:46:52.093-0330 I NETWORK  [conn3] received client metadata from
127.0.0.1:35016 conn3: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB
Internal Client", version: "3.4.1" }, os: { type: "Linux", name: "LinuxMint", architecture: "x86_64",
version: "18.1" } }
2017-03-08T17:47:31.060-0330 I -        [conn3] end connection 127.0.0.1:35016 (1 connection
now open)
2017-03-08T17:47:34.392-0330 I NETWORK  [thread1] connection accepted from
127.0.0.1:35026 #4 (1 connection now open)
```

```
2017-03-08T17:47:34.393-0330 I NETWORK  [conn4] received client metadata from
127.0.0.1:35026 conn4: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB
Internal Client", version: "3.4.1" }, os: { type: "Linux", name: "LinuxMint", architecture: "x86_64",
version: "18.1" } }
2017-03-08T17:47:39.352-0330 I NETWORK  [thread1] connection accepted from
127.0.0.1:35030 #5 (2 connections now open)
2017-03-08T17:47:39.358-0330 I NETWORK  [conn5] received client metadata from
127.0.0.1:35030 conn5: { driver: { name: "nodejs", version: "2.2.24" }, os: { type: "Linux", name:
"linux", architecture: "x64", version: "4.4.0-53-generic" }, platform: "Node.js v7.3.0, LE,
mongodb-core: 2.1.8" }
2017-03-08T17:47:39.380-0330 I -        [conn5] end connection 127.0.0.1:35030 (2 connections
now open)
```

2.4.1.2 Inserting a document into the database:

```
var mnogodb = require('mongodb');

//"MongoClient" interface connecting to a mongodb server.
var MongoClient = mongodb.MongoClient;

// Connection URL.
var url = 'mongodb://localhost:27017/test';

// Use connect method to connect to the Server
MongoClient.connect(url, function (err, db) {
  if (err) {
    console.log('Unable to connect to the mongoDB server. Error:', err);
  } else {
    console.log('Connection established to', url);

    // do some work here with the database.
    var collection = db.collection('test');
    var doc1 = {'hello':'doc1'};
    var doc2 = {'world':'doc2'};

    collection.insert(doc1);

    collection.insert(doc2, {w:1}, function(err, result) {});

    //Close connection
    db.close();
  }
});
```

2.4.1.3 Using Mongo shell to view insertions:

```
MongoDB shell version v3.4.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.1
Server has startup warnings:
2017-03-08T17:46:10.121-0330 I STORAGE  [initandlisten]
2017-03-08T17:46:10.121-0330 I STORAGE  [initandlisten] ** WARNING: Using the XFS
filesystem is strongly recommended with the WiredTiger storage engine
2017-03-08T17:46:10.121-0330 I STORAGE  [initandlisten] **          See
http://dochub.mongodb.org/core/prodnotes-filesystem
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] ** WARNING: Access control is not
enabled for the database.
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] **          Read and write access to
data and configuration is unrestricted.
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] **        We suggest setting it to
'never'
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten] **        We suggest setting it to
'never'
2017-03-08T17:46:10.421-0330 I CONTROL  [initandlisten]
> use test
switched to db test
> show collections
test
> db.test.find()
{ "_id" : ObjectId("58c074f3904386486746600e"), "hello" : "doc1" }
{ "_id" : ObjectId("58c074f3904386486746600f"), "world" : "doc2" }
```

2.4.1.4 Removing the documents from the database

```
MongoClient.connect(url, function (err, db) {
  if (err) {
    console.log('Unable to connect to the mongoDB server. Error:', err);
  } else {
    console.log('Connection established to', url);

    // do some work here with the database.
    var collection = db.collection('test');
    collection.remove({});

    //Close connection
    db.close();
  }
});
```

2.4.1.5 Check the successful removal using the mongo terminal

```
> use test
switched to db test
> show collections
test
> db.test.find()
>
```

2.4.2 Schema Unit Test

2.4.2.1 Schema Connected

```
var mongoose = require('mongoose');
var bcrypt = require('bcrypt-nodejs');
var ObjectId = mongoose.Schema.Types.ObjectId;
//define the schema for the user model

var userSchema = mongoose.Schema({

        local   : {
                email: {
                        type: String,
                        unique: true,
                },
                password: String,
        },
        // shift this to profile after testing
        friends:[{type : ObjectId, ref: 'User' }]
});

var profileSchema = mongoose.Schema({
        name: {
                first: { type: String},
                last: { type: String}
        },
        address: {type: String},
        birthDate: Date,
        image: {
                type: String,
                default: 'images/user.png'
        },

        friends: {
                accepted: [{type: ObjectId, ref: 'User'}],
                pending: [{type: ObjectId, ref: 'User'}]
        },
        login : [{type: ObjectId, ref: 'userSchema'}]
});

// methods
// generating a hash

userSchema.methods.generateHash = function(password) {
        return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null);
};

// checking if password is valid
userSchema.methods.validPassword = function(password){
        return bcrypt.compareSync(password, this.local.password);
};

// create the model for users and expose it to the app

var User = mongoose.model('User', userSchema);

module.exports = {User: User};
```

2.4.2.2 Schema add friend

```
GET /profile 200 12.273 ms - 3201
POST /addfriend 302 9.386 ms - 60
GET /profile 200 15.764 ms - 3251
```

2.4.2.3 Successfully added friend

```
> db.users.find()

{ "_id" : ObjectId("58a4718928899c2429f4d344"),
"friends" : [ ObjectId("58a4718928899c2429f4d344") ],
 "image" : "images/user.png",
"local" : { "password" :
"$2a$08$iJaR.T0moXMCB2OiqJMXbelU.V50IuPbt98f43DMtdMc8WINsRk1O",
"email" : "dsad" }, "__v" : 0 }

{ "_id" : ObjectId("58a471d128899c2429f4d345"),
"friends" : [ ObjectId("58a4718928899c2429f4d344"), ObjectId("58a471d128899c2429f4d345")
],
"image" : "images/user.png",
"local" : { "password" :
"$2a$08$SiWemBb5WXVHJZFEOKU/X.TKvdaCqir7ARWZjBltQxBcGlNmoxnEi",
"email" : "dsadss" }, "__v" : 0 }
```

2.4.2.4 Schema unsuccessful add friend without name

```
POST /signup 302 7.297 ms - 58
GET /signup 200 5.295 ms - 1234
```

2.4.2.5 No change in schema

```
> db.users.find()

{ "_id" : ObjectId("58a4718928899c2429f4d344"),
"friends" : [ ObjectId("58a4718928899c2429f4d344") ],
 "image" : "images/user.png",
"local" : { "password" :
"$2a$08$iJaR.T0moXMCB2OiqJMXbelU.V50luPbt98f43DMtdMc8WlNsRk1O",
"email" : "dsad" }, "__v" : 0 }

{ "_id" : ObjectId("58a471d128899c2429f4d345"),
"friends" : [ ObjectId("58a4718928899c2429f4d344"), ObjectId("58a471d128899c2429f4d345")
],
"image" : "images/user.png",
"local" : { "password" :
"$2a$08$SiWemBb5WXVHJZFEOKU/X.TKvdaCqir7ARWZjBltQxBcGlNmoxnEi",
"email" : "dsadss" }, "__v" : 0 }
```

2.4.3 Adding relationships to users

```
        "_id" : ObjectId("58c8a07c9a74716036de32c0"),
        "friend" : [
                {
                        "added" : ISODate("2017-03-15T02:01:52.782Z"),
                        "status" : "accepted",
                        "_id" : ObjectId("58c8a08c9a74716036de32c1")
                },
                {
                        "added" : ISODate("2017-03-15T02:02:03.250Z"),
                        "status" : "requested",
                        "_id" : ObjectId("58c8a07c9a74716036de32c0")
                },
                {
                        "added" : ISODate("2017-03-15T02:02:03.250Z"),
                        "status" : "requested",
                        "_id" : ObjectId("58c8a07c9a74716036de32c0")
                },
                {
                        "added" : ISODate("2017-03-15T02:41:15.720Z"),
                        "status" : "pending",
                        "_id" : ObjectId("58c8a86393bfa96f66c9c486")
                },
                {
                        "added" : ISODate("2017-03-15T03:54:55.728Z"),
                        "status" : "pending",
                        "_id" : ObjectId("58c8bb0831bc5219da4dbcf7")
                }
        ],
        "local" : {
                "password" :
"$2a$08$6XQPCHBj0sVMvl.R5aJAmeaXwXzY6BrbSJjNhBOgQ0zSdNIvDJKOy",
                "email" : "test1",
                "active" : true
        },
        "__v" : 0
```

2.4.3 Adding posts to the database

```
> db.posts.find().pretty()
{
        "_id" : ObjectId("58cbdef8d07788517baf4d3a"),
        "date" : ISODate("1970-01-01T00:00:00.017Z"),
        "body" : "Hello Shawn, this is a great little posting system that you have set up here.",
        "comments" : [ ],
        "postto" : [
                ObjectId("58c8bb0831bc5219da4dbcf7")
```

```
	],
	"postby" : [
		ObjectId("58c8bb0831bc5219da4dbcf7")
	],
	"__v" : 0
}
```

2.5 Template Test Code

2.5.1 Code

2.5.1.1 HTML

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Untitled Document</title>
<link href="CSS/signCSS.css" rel="stylesheet" type="text/css">
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.min.js"></script>
<script src="JS/navbarJS.js"></script>
</head>
<div id="banner">
   <h2>Test Information</h2>
   <p>There will be a banner here</p>
</div>
<nav id="nav_bar">
   <ul class="nav_links">

         <li><form action="/action_page.php">
                Email: <input type="text" name="em" value="">
                Password: <input type="text" name="pw" value="">
                <input type="submit" value="Log In">
</form>
   </ul>
</nav>


<div id="body_div">
   <div id = "profLeft">
         <p>Lorem ipsum dolor sit amet, vel te tibique percipitur efficiantur, dictas recteque mel
ad. Nam modo honestatis disputationi te, elaboraret delicatissimi ea pri, ne pri impedit
mediocrem petentium. Mel in fabulas electram, fabellas prodesset definitiones ei his. In mea
dicta dicam dolores, ea legere mandamus cum. Debitis voluptaria dissentiunt an quo, putant
definitiones nec ne, ut eam solet epicurei.
```

Et pro wisi dicunt ponderum, vis in simul recusabo, eu per idque aliquando. Cum quaeque omnesque ut, cu nihil malorum sea, eu purto tincidunt has. Qui propriae definitiones in. Eripuit vocibus nec te, persius saperet adipisci in nec, cu pri maiorum sententiae. Invidunt inimicus in nam. Cum everti equidem an.

Albucius prodesset rationibus ea sea, malis doctus definitionem mei ei. Solum clita nam no, cum impetus laoreet ad. Cum constituto interpretaris ne. Ponderum accusamus vel no, an pri utamur intellegat. Ea cum liber omittam, dicit albucius pri ad, habeo tritani labitur ei sit.

Usu latine corpora offendit et. Ne unum commune vix, et qui corrumpit dissentiunt. Omnes eirmod principes ea sea, rebum officiis urbanitas ut duo. Vis nominavi recusabo te, duo eu facer tempor moderatius, mea aeterno veritus perfecto an.

Iriure corrumpit mediocritatem eum te, unum graeco viderer est no. Mea option sanctus eu, probo adhuc signiferumque et qui. Minimum posidonium pri eu. No alii mandamus est. Ex vix putent singulis volutpat, his aperiam veritus at.

Has at qualisque voluptaria. Eam ei enim officiis philosophia, sit ad dicat volumus, nihil tibique ad his. Et est sint graeci, qui vero vidisse ea, amet suavitate in sit. Falli malorum vel cu. Epicuri invidunt eum an, aperiam civibus sea et, vis summo affert sapientem at. Mea no ullum minim. Pri ut quaeque suscipiantur deterruisset.

Vix ex appetere indoctum. Ne elit discere mediocrem vis. Est cu omnis latine, sed ex repudiandae contentiones. Quo eros periculis et. At justo feugait invidunt mea, qui ne vocent latine, sit nemore inermis torquatos no. Ei alia modus patrioque ius.</p>

```
   </div>
   <div id = "profRight">
         <h1>Sign Up</h1>
                        <form class="signForm" action="/action_page.php">
                                <input class="fName" type="text" name="fName"
placeholder="First Name">
                                <input class="lName" type="text" name="lName"
placeholder="Last Name"></br></br>
                                <input class="iBox" type="text" name="Email"
placeholder="Email"></br></br>
                                <input class="iBox" type="text" name="EmailC"
placeholder="Email Confirm"></br></br>
                                <input class="iBox" type="text" name="Password"
placeholder="Password"></br></br>
                                <input class="iBox" type="text" name="PasswordC"
placeholder="Password Confirm"></br></br>
```

```
                                    <input type="submit" value="Submit">
                           </form>
   </div>

</div>

<body>
</body>
</html>
```

2.5.1.2 CSS

```
@charset "utf-8";
/* CSS Document */
/* Not everything in this CSS document is relevant for the signup page */
.signForm input {
   color:darkgray;
   line-height: normal;
}

.iBox {
   width:100%;
}

.fName {
   width: 35%;
}

.lName {
   width: 60%;
   float: right;
}

body {
   line-height: 0px;
   margin: 0;
   padding: 0;
   width: 100%;
   height: 100%;
}
```

```
.navbar-fixed {
    top: 0;
    z-index: 100;
    position: fixed;
    width: 100%;
}

#body_div {
    vertical-align: top;
    top: 0;
    height: 100%;
    margin: 0 5% 0 5%;
    line-height: normal;

}

#profLeft {
    width:44%;
    float:left;
    padding: 3%;
}

#profRight {
    background-color: maroon;
    width: 44%;
    padding: 3% ;
    float: right;
    border-radius: 0px 0px 15px 15px;
    color: white;
}

#friendsBox {
    background-color: maroon;
    width: 54%;
    padding: 3% ;
    float: right;
    border-radius: 15px;
    margin-top: 10px;
    margin-bottom: 10px;
}

#banner {
    width: 100%;
    height: 273px;
    background-color: grey;
    overflow: hidden;
}

#nav_bar{
    border: 0;
    background-color:maroon;
    border-radius: 0px;
    margin: 0 ;
    line-height: normal;
    z-index: 100;
    width: 100%;

}

.nav_links {
    margin: 0 5% 0 0;
}

.nav_links li {
    display: inline-block;
    padding: 4px;
    margin: 0;
    background-color: maroon;
}

.nav_links li a {
    padding: 0 15.5px;
    color:white;
    text-decoration: none;
}

p {
    margin: 0;
}
```

2.5.1.3 JScript

```
$(document).ready(function() {
  "use strict";
  $(window).scroll(function () {
        console.log($(window).scrollTop());
        if ($(window).scrollTop() > $('#banner').height()) {
        $("#nav_bar").addClass("navbar-fixed");
        }
        if ($(window).scrollTop() <= $('#banner').height()) {
        $("#nav_bar").removeClass("navbar-fixed");
        }
  });
});
```