

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS 301 MINI PROJECT

Top Level Integration Testing

Elana Kuun u12029522
Hlavutelo Maluleke u12318109
Estian Rosslee u12223426
David Breetzke u12056503
Sylvester Mpungane u11241617
Phethile Mkhabela u12097561
Renaldo van Dyk u12204359
Antonia Michael u13014171
Herman Keuris u13037618
Jaco-Louis Kruger u13025105

Github Repository Link

April 2015

Contents

1	Introduction	3
2	Testing Results	3
2.1	Top Level A Testing Results	3
2.1.1	Functional Testing	3
2.1.2	Non-functional Testing	18
2.2	Top Level B Testing Results	32
2.2.1	Functional Testing	32
2.2.2	Non-functional Testing	43
3	Conclusion	50

1 Introduction

The testing phase of the mini project required thorough testing of all the aspects of both systems developed. This included testing both the functional and non-functional aspects. The following report discusses the different aspects tested from the two different systems, A and B. For the functional testing, this includes tests performed and a discussion about whether the system's use cases are met or not. The Buzz system aimed to assist users to collaborate and share knowledge with fellow students. The Buzz System was designed in order to create modules for the various functionality required, these modules needed to be integrated to create the desired system. The testing of both systems will be discussed in the following sections.

2 Testing Results

2.1 Top Level A Testing Results

2.1.1 Functional Testing

1. Buzz-Spaces Module

- Use Cases

- createBuzzSpace

This is the service which enables a lecturer to create a Buzz space for a particular module they present.

- * Pre-conditions:

- buzzSpaceExists (should not be able to create a duplicate buzz space) (implemented, was unable to create a space that already existed)

- moduleNotActive (for the current year) (not implemented, was able to create a space for a non existent module)

- notAuthorized (only an authorized user can create a buzz space) (not implemented, was able to create a space without signing in as a lecturer)

- * Post-conditions:

- storeBuzzSpace (persist the new buzz space)(implemented,

- new buzz space was reflected in system once user is returned to home page)
- lecturer registered on the buzz space (not implemented)
- lecturer assigned as administrator of the buzz space (not implemented)
- Create welcome message as root thread for the buzz space (not implemented)

```
spaces.createBuzzSpace = function (createBuzzSpaceRequest) {
  var newBuzzSpace = new Space({
    academicYear: createBuzzSpaceRequest.academicYear,
    isOpen: createBuzzSpaceRequest.isOpen,
    moduleID: createBuzzSpaceRequest.moduleID,
    name: createBuzzSpaceRequest.name,
    adminUsers: createBuzzSpaceRequest.adminUsers
  });

  //check if the module exists
  Space.findOne({'moduleID': createBuzzSpaceRequest.moduleID}, function (err, result) {
    if (result == null) {
      console.log("Buzz space not yet created, able to insert");
      newBuzzSpace.save(function (err, cosTest) {
        if (err) return console.error(err);

        Space.find(function (err, spaces) {
          if (err) return console.error(err);
          //console.log(spaces);
        });
      });
    } else {
      console.log("Buzz space already exists");

      Space.find(function (err, spaces) {
        if (err) return console.error(err);
        //console.log(spaces);
      });
      newBuzzSpace = result;
    }
  });

  return newBuzzSpace;
};
```

```
[Express][::1][POST][DESKTOP] 2015-04-24T08:20:01.733Z /ajax/newSpace
could not authorize
Buzz space already exists
]
```

– closeBuzzSpace

This is the service used to set a buzz space to inactive.

* Pre-conditions:

- noSuchBuzzSpace (if buzz space to be closed does not exist then one can't close it) (was unable to find a non existent buzz space to test this)

[This module does not exist - Test space](#)

Year: 2015

Create new Buzz-Space

Module ID

This module does not exist

Name

Test space

Academic year

2015

Welcome to COS301

[Manage Space](#)

Threads in this space

-notAuthorized (only an admin user of a buzz space can close the buzz space) (not implemented, was able to close a space I was not an admin for)

* Post-conditions:

-buzz space no longer active (reflected upon returning to the home page)

```
spaces.closeBuzzSpace = function (closeBuzzSpaceRequest) {
  // check if space exists
  Space.findOne({'moduleID': closeBuzzSpaceRequest.moduleID}, function (err, result) {
    if (err) {
      closeBuzzSpaceRequest.callback(err);
    } else if (result == null) {
      // Module does not exist
      console.log("Did not find space that should be closed");
      //closeBuzzSpaceRequest.callback(new Error("NoSuchBuzzSpaceException"));
      closeBuzzSpaceRequest.callback("Did not find space that should be closed");
    } else {
      // check if user is an admin for buzz space
      spaces.isAdministrator({
        userID: closeBuzzSpaceRequest.userID,
        moduleID: closeBuzzSpaceRequest.moduleID,
        callback: function (error, response) {
          if (error) {
            closeBuzzSpaceRequest.callback(error);
          } else if (response == false) {
            console.log("Not an admin of this buzz space");
            //closeBuzzSpaceRequest.callback(new Error("NotAuthorizedException"));
            closeBuzzSpaceRequest.callback("Not an admin of this buzz space");
          } else {
            //else module exists, update
            console.log("Found space that should be closed");
            result.isOpen = false;

            // save changes
            result.save(function (err) {
              //res.send(result);
            });
            console.log("Space closed successfully");

            //closeBuzzSpaceRequest.callback(null, result);
            closeBuzzSpaceRequest.callback("Space closed successfully");
          }
        }
      });
    }
  });
};
```

Space closed successfully

Manage

Close Space

Add admin

userID

Add Admin

```
spaces.createBuzzSpace = function (createBuzzSpaceRequest) {
  var newBuzzSpace = new Space({
    academicYear: createBuzzSpaceRequest.academicYear,
    isOpen: createBuzzSpaceRequest.isOpen,
    moduleID: createBuzzSpaceRequest.moduleID,
    name: createBuzzSpaceRequest.name,
    adminUsers: createBuzzSpaceRequest.adminUsers
  });

  //check if the module exists
  Space.findOne({'moduleID': createBuzzSpaceRequest.moduleID}, function (err, result) {
    if (result == null) {
      console.log("Buzz space not yet created, able to insert");
      newBuzzSpace.save(function (err, cosTest) {
        if (err) return console.error(err);

        Space.find(function (err, spaces) {
          if (err) return console.error(err);
          //console.log(spaces);
        });
      });
    } else {
      console.log("Buzz space already exists");

      Space.find(function (err, spaces) {
        if (err) return console.error(err);
        //console.log(spaces);
      });
      newBuzzSpace = result;
    }
  });

  return newBuzzSpace;
};
```

```
closeBuzzSpaceRequest.callback("Did not find space that should be closed");
} else {
  // check if user is an admin for buzz space
  spaces.isAdministrator({
    userID: closeBuzzSpaceRequest.userID,
    moduleID: closeBuzzSpaceRequest.moduleID,
    callback: function (error, response) {
      if (error) {
        closeBuzzSpaceRequest.callback(error);
      } else if (response == false) {
        console.log("Not an admin of this buzz space");
        //closeBuzzSpaceRequest.callback(new Error("NotAuthorizedException"));
        closeBuzzSpaceRequest.callback("Not an admin of this buzz space");
      } else {
        //closeBuzzSpaceRequest.callback(new Error("NotAuthorizedException"));
      }
    }
  });
}
```

– registerOnBuzzSpace

This service is used to create a profile for a user on a particular buzz space.

* Pre-conditions:

- notAuthorized (user is registered for the module to which the space is associated) (not implemented, was able to register a fake user successfully)
- buzz space active (implemented, only able to access register for spaces which are active on home page)

- * Post-conditions:
 - user profile persisted to database (implemented) (screenshots provided for proof)

```
spaces.registerOnBuzzSpace = function (registerOnBuzzSpaceRequest) {
  // check if buzz space exists
  Space.findOne({"moduleID": registerOnBuzzSpaceRequest.moduleID}, function (err, result) {
    if(err) {
      registerOnBuzzSpaceRequest.callback(err);
    } else if (result == null) {
      // module does not exist
      console.log("Buzz space does not exist");
      //registerOnBuzzSpaceRequest.callback(new Error("NoSuchBuzzSpaceException"));
      registerOnBuzzSpaceRequest.callback("Buzz space does not exist");
    } else {
      // check if space is active
      if (result.isOpen === true || result.isOpen === "true") {
        // check also if profile exists
        SpaceProfile.findOne({"userID": registerOnBuzzSpaceRequest.userID}, function (err1, result1) {
          if (err1) {
            //registerOnBuzzSpaceRequest.callback(err1);
          } else if (result1 == null) {
            // create new profile
            var newSpaceProfile = new SpaceProfile({
              userNameForBuzzSpace: registerOnBuzzSpaceRequest.userNameForBuzzSpace,
              signature: registerOnBuzzSpaceRequest.signature,
              userID: registerOnBuzzSpaceRequest.userID,
              moduleID: registerOnBuzzSpaceRequest.moduleID
            });

            newSpaceProfile.save(function (err2, result2) {
              if (err) {
                //registerOnBuzzSpaceRequest.callback(err2);
                registerOnBuzzSpaceRequest.callback("Could not create profile");
              } else {
                console.log("Profile registered successfully");
                //registerOnBuzzSpaceRequest.callback(null, result2);
                registerOnBuzzSpaceRequest.callback("Profile created successfully");
              }
            });
          } else {
            console.log("Space profile already exists");
            //registerOnBuzzSpaceRequest.callback(null, result1);
            registerOnBuzzSpaceRequest.callback("Space profile already exists");
          }
        });
      } else {
        console.log("Buzz space is not active");
        //registerOnBuzzSpaceRequest.callback(new Error("BuzzSpaceNotActiveException"));
        registerOnBuzzSpaceRequest.callback("Buzz space is not active");
      }
    }
  });
});
```

```

*/
spaces.assignAdministrator = function (assignAdministratorRequest) {
  //check if the module exists
  Space.findOne({'moduleID': assignAdministratorRequest.moduleID}, function (err, result) {
    if (result == null) {
      //throw new Error("NoSuchBuzzSpaceException");
      console.log("Could not find buzz space to assign admin to");
    } else {
      console.log("Found module to assign admin to");
      // check if user is an admin for buzz space
      if (isAuthorized(assignAdministratorRequest.userID, assignAdministratorRequest.moduleID)) {
        result.adminUsers.push(assignAdministratorRequest.newAdmin);

        result.save(function (err) {
          //res.send(result);
        });
      } else {
        throw new Error("NotAuthorizedException");
      }
    }
  });
};

```


- `getProfileForUser` Simply returns the profile the user has on the buzz space
Code found but no functional implementation exists in the system that could be tested

2. Buzz-Data-Sources

- Use Cases
 - login
Login - Using a provided username and password authenticates the user against details retrieved from an ldap repository
 - * Pre-conditions:
 - could connect to CS data source (LDAP) (implemented in code only, no functional way to test)
 - user exists in ldap with provided authentication details (code indicates this is checked by performing a bind to ldap using the provided details)
 - * Post-conditions:
 - userID returned (not implemented)
 - `getUsersRolesForModule`
Login - Using a provided username and password authenticates the user against details retrieved from an ldap repository
 - * Pre-conditions:
 - could connect to CS data source (LDAP) (implemented in code only, no functional way to test)
 - user exists in ldap with provided authentication details (code indicates this is checked by performing a bind to ldap using the provided details)
 - * Post-conditions:
 - userID returned (not implemented)

```

//binds using a user's dn and provided password
function auth(dn, password, callback1, callback2)
{
  //client = ldap.createClient({url: 'ldap://reaper.up.ac.za:'});
  client.bind(dn, password, function (err){
    client.unbind();
    callback1(err === null, err, callback2);
  });
}

function output(res, err, callback) {
  if (res) {
    console.log('successful bind');
    callback(true);
  } else {
    console.log('failure to bind ' + err.message);
    callback(false);
  }
}

csds.login = function(username,password,callback){

  client = ldap.createClient({url: 'ldap://reaper.up.ac.za:'});
  var opts = {
    filter: 'uid='+username,
    scope: 'sub'
  };

  client.search(base, opts, function(err,res){
    assert.ifError(err);

    res.on('searchEntry', (function (entry) {
      var user = JSON.parse(JSON.stringify(entry.object));
      if (typeof user.dn == "string") {
        setDnTest(user.dn);
        found = true;
      }
      if (found) {
        auth(entry.object.dn, password, output,callback);
      }
    }));
    res.on('error',function(err){
      console.error('error: ' + err.message);
    });
  });
  //client.unbind();
}

module.exports = csds;

```

- getUsersWithRole
 - retrieves all users with a particular role for a particular module (no functional way to test)
 - * Pre-conditions:
 - could connect to CS data source (LDAP) (implemented in code only, no functional way to test)
- getActiveModulesForYear
 - retrieves all users with a particular role for a particular module (no functional way to test)

```

var csds = new Object();
var results = new Array();

//Users role for a module
csds.getUserRolesForModules = function(memberUid,callback){
  client = ldap.createClient({url: 'ldap://reaper.up.ac.za:'});
  var opts = {
    filter: 'memberUid='+memberUid,
    scope: 'sub'
  };
  client.search(base, opts, function (err, res)
  {
    assert.ifError(err);
    res.on('searchEntry', function (entry)
    {
      results.push(JSON.parse(JSON.stringify(entry.object.cn)));
    });
    res.on('error',function(err){
      callback({'error': err.message});
    });
    res.on('end', function (result)
    {
      callback(results);
    });
  });
  client.unbind();
}

module.exports = csds;

```

- * Pre-conditions:
 - could connect to CS data source (LDAP) (implemented in code only, no functional way to test)

- getActiveModulesForYear
 - retrieves all users with a particular role for a particular module (no functional way to test)
 - * Pre-conditions:
 - could connect to CS data source (LDAP) (implemented in code only, no functional way to test)

3. Buzz Status

- Use Cases
 - AssessProfile
 - This is where the lecturers can specify different ways to assess the students. In the current system, this functionality is not present. SetStatusCalculator - This is where the status is calculated according to the number of posts the user has

```

var results = new Array();

//Users with particular role for particular module
csds.getUsersWithRole = function(role,moduleCode,callback){
  client = ldap.createClient({url: 'ldap://reaper.up.ac.za:'});
  var Role;
  switch(role){
    case 'lecturer':
    case 'Lecturer':
      Role = 'lect_';
      break;
    case 'student':
    case 'Student':
      Role = 'stud_';
      break;
    case 'teaching assistant':
    case 'Teaching Assistant':
    case 'TA':
      Role = 'teachasst_';
      break;
    case 'tutor':
    case 'Tutor':
      Role = 'tuts_';
      break;
  }

  var f = Role.concat(moduleCode);
  var opts = {
    filter: ('cn=' + f),
    scope: 'sub'
  };
  var modulesObject;
  var modulesDn;
  client.search(base,opts, function(err,res){
    assert.ifError(err);
    res.on('searchEntry', function (entry)
    {
      modulesDn = JSON.stringify(entry.object.memberUid);
      modulesObject = (JSON.parse(modulesDn));
      results.push(modulesObject);
    });
    res.on('error',function(err){
      callback({'error': err.message});
    });
    res.on('end', function (result)
    {
      callback(results);
    });
  });
  client.unbind();
}

module.exports = csds;

```

made. Again, this functionality does not exist in the sysemt and hence cannot be tested.

- getStatusForProfile

This query returns status for a specific profile. createAppraisalType - New appraisal types are created that can be reused. The current system makes use of this functionality where a user can add appraisals under the appraisal tab on the menu bar, where an option exists to add an appraisal type.

```

var csds = new Object();
var results = new Array();

//Active modules for the year
csds.getActiveModulesForYear = function(callback){
  client = ldap.createClient({url: 'ldap://reaper.up.ac.za:'});
  var opts = {
    filter: ('cn=lect_' + '*'),
    scope: 'sub'
  };
  var modulesObject;
  var modulesDn;
  //console.log('Active modules for the year: ');
  client.search(base,opts, function(err,res){
    assert.ifError(err);
    res.on('searchEntry', function (entry)
    {
      modulesDn = JSON.stringify(entry.object.cn);
      modulesObject = (JSON.parse(modulesDn));
      var res = modulesObject.substring(5, 11);
      results.push(res);
    });
    res.on('error',function(err){
      callback({'error': err.message});
    });
    res.on('end', function (result)
    {
      callback(results);
    });
  });
  client.unbind();
}

module.exports = csds;

```

The various options that exist, regarding this, is adding appraisal name, description, and adding appraisals on different levels by adding new levels with descriptors and ratings.

- activateAppraisalType
This use case activates an appraisal type for a specified period on a specific buzz space. This was not implemented.

4. Buzz-reporting

- Use Cases
 - getThreadStats
Returns statistical information of subsets of posts complying with specified restrictions (code present but no functional way to test from system)
 - getThreadAppraisal
(code not present, not implemented, not testable)

```

reporting.getThreadStats = function (set, action) {
  var answer;
  if (action == "Num") {
    answer = set.length;
  }
  else if (action == "MemCount") {
    var nameArray = [];
    for (var i = 0; i < set.length; i++) {
    }

    answer = nameArray.length;
  }
  //implemented assuming all the necessary threads/posts are in the set parameter
  else if (action == "MaxDepth") {
    var currentMax = -1;
    var id = 0;
    var parentId = 0;
    var tempCount = -1;

    for (var i = 0; i < set.length; i++) {
      id = set[i].thread_ID;
      parentId = set[i].thread_Parent;
      tempCount = 0;

      while (parentId != id) {
      }

      if (tempCount > currentMax) {
        currentMax = tempCount;
      }

    }
    answer = currentMax + 1;
  }
  //implemented assuming all the necessary threads/posts are in the set parameter
  else if (action == "AvgDepth") {
    var totDepth = 0;

    var tempDepth = 0;

    for (var i = 0; i < set.length; i++) {
    }

    answer = totDepth / set.length;
  }

  return answer;
}

```

- exportThreadAppraisal
(code not present, not implemented, not testable)
- importThreadAppraisal
(code not present, not implemented, not testable)
- exportThread
Service to backup the content of a thread (implemented in code only, no functional way to test from system)

```

reporting.exportThread = function (ThreadID) {
  var serialize = require('node-serialize');
  var thread_Collec = mongoose.model('Threads', ThreadSchema); //Defines a model f

  thread_Collec.find({}, function (err, Threads) {
    Threads.forEach(function (Thread) {
      if (Thread.thread_ID == ThreadID) {
        //convert Thread to serialized object
        var objS = serialize.serialize(Thread, true);
        typeof objS === 'string';
        var nameOfFile = 'thread' + ThreadID + '.txt';
        fs.writeFile(nameOfFile, objS, function (err) {
          if (err) {
            return console.log(err);
          }
          console.log("The file was saved!");
        });
      }
    });
  });
};
}

```

- importThread

Service to backup the content of a thread (implemented in code only, no functional way to test from system)

```

reporting.importThread = function (ThreadID) {
  var serialize = require('node-serialize');
  var nameOfFile = 'thread' + ThreadID + '.txt';
  var Thread;
  fs.readFile(nameOfFile, function (err, data) {
    if (err) throw err;
    Thread = data;
    serialize.unserialize(Thread);
  });
  return Thread;
};
return reporting;

```

5. Buzz-authorization

- Use Cases

- addAuthorizationRestriction

Adds an authorization restriction for a user role and a particular buzz space (implemented in code only, no functional way to test)

- * Pre-conditions:

- an authorization restriction does not yet exist for the specified user role and buzz space (not implemented)

-user has administrative rights (not implemented)

* Post-conditions:

-authorization restriction must be persisted (not implemented)

```
//////////////////////////////////AddAuthorizationRestrictionRequest class and functions//////////////////////////////////
var AddAuthorizationRestrictionRequest;
/**
AddAuthorizationRestrictionRequest = function(_userID, _AuthorizationRestriction)
{
    "use strict"
    console.log('in AddAuthorizationRestrictionRequest ' + _userID);

    if (this instanceof AddAuthorizationRestrictionRequest)
    {
        this.AuthorizationRestriction = _AuthorizationRestriction;
        this.userID = _userID;
    }
    else return new AddAuthorizationRestrictionRequest(_userID, _AuthorizationRestriction);
};

AddAuthorizationRestrictionRequest.prototype.getUserID=function()
{
    return this.userID;
};

AddAuthorizationRestrictionRequest.prototype.setUserID=function(_userID)
{
    this.userID=_userID;
};

```

Figure 1: Add restriction

– isAuthorized

Service used to determine which system services are available to the user. (implemented in code only, no functional way to test)

* Pre-conditions:

-buzz space is active (not implemented)

– removeAuthorizationRestriction

Removes an authorization restriction for a user role and a particular buzz space (implemented in code only, no functional way to test)

* Pre-conditions:

-user has administrator role (not implemented)


```

var isAuthorizedRequest= function(userID, ServiceIdentifier)
{
    "use strict"
    console.log('in isAuthorizedRequest ' + userID);

    if (this instanceof isAuthorizedRequest)
    {
        this.userID = userID;
        this.serviceIdentifier = ServiceIdentifier;
    }
    else return new isAuthorizedRequest(userID, ServiceIdentifier);
};

//returns the userID
isAuthorizedRequest.prototype.getUserID =function()
{
    return this.userID;
};

//returns serviceIdentifier object
isAuthorizedRequest.prototype.getIsAuthorizedRequestServiceIdentifier=function()
{
    return this.serviceIdentifier;
};

/* isAuthorizedResult class
 * @param isAuth - boolean variable
 */
var isAuthorizedResult = function(isAuth)
{
    this.isAuthorized=isAuth;
};

isAuthorizedResult.prototype.getIsAuthorized=function()
{
    return this.isAuthorized;
};

```

Figure 2: isAuthorized

- * Post-conditions:
 - authorization restriction for the user role must be removed from the buzz space (not implemented)

```

//RemoveAuthorizationRestrictionRequest class and functions
var RemoveAuthorizationRestrictionRequest = function (userID, _AuthorizationRestriction) {

    "use strict"
    console.log('in RemoveAuthorizationRestrictionRequest ' + userID);

    if (this instanceof RemoveAuthorizationRestrictionRequest)
    {
        this.userID = userID;
        this.AuthorizationRestriction = AuthorizationRestriction;
    }
    else return new RemoveAuthorizationRestrictionRequest(userID, AuthorizationRestriction);

    /*this.userID = userID;
    this.AuthorizationRestriction = _AuthorizationRestriction;*/
};
RemoveAuthorizationRestrictionRequest.prototype.getUserID = function () {
    return this.userID;
};
RemoveAuthorizationRestrictionRequest.prototype.getAuthorizationRestriction = function () {
    return this.AuthorizationRestriction;
};

```

Figure 3: Remove restriction

- getAuthorizationRestrictions Returns authorization restrictions for a buzz space from the database (implemented in code only,

no functional way to test)

- * Pre-conditions:

- user has administrator role for space being queried (not implemented)
- buzz space is active (not implemented)

- updateAuthorizationRestrictions

Facilitates editing of authorization restrictions for a buzz space (implemented in code only, no functional way to test)

- * Pre-conditions:

- user has administrator role for space being queried (not implemented)

- * Post-conditions:

- new authorization restriction persisted to database (implemented in code only, no functional way to test)

```
Authorization.updateAuthorizationRestriction=function(UpdateAuthReq)//The updateAuthorizationRestriction function
{
    var buzzSpace=new BuzzSpaces();
    if(!buzzSpace.isAdministrator(UpdateAuthReq.getUserID()))
    {
        return new Error("notAuthorizedException");
    }
    var auth;
    // checks if it is the first time accessing the database
    if(mongoose.models.Authorization) {
        auth = mongoose.model('Authorization');
    }else {
        auth = mongoose.model('Authorization', authSchema);
    }
    //finding the entry and updating it
    auth.findOneAndUpdate(
        { 'methodName':(UpdateAuthReq.getAuthorizationRestriction().getServiceRestriction().getServiceRestrictionServiceIdentifier().getServiceRestriction().getModuleID()), 'moduleName':(UpdateAuthReq.getAuthorizationRestriction().getServiceRestriction().getServiceRestrictionServiceIdentifier().getServiceRestriction().getModuleID()), 'roleName':(UpdateAuthReq.getAuthorizationRestriction().getServiceRestriction().getServiceRestrictionServiceIdentifier().getServiceRestriction().getRoleName()) },
        { '$set': {
            'StatusPoints': UpdateAuthReq.getAuthorizationRestriction().getServiceRestriction().getServiceRestrictionMinimumStatusPoints()
        } },
        function (err) {
            if (err) {
                console.log("Entry not found");
                return null;
            } else {
                return new UpdateAuthorizationRestrictionsResult(UpdateAuthReq.getAuthorizationRestriction());
            }
        });
};
```

Figure 4: Update restriction

6. Testing performed

//////Jaco

2.1.2 Non-functional Testing

1. Maintainability:

In order for a system to be maintainable, the system should be both flexible and extensible. For developers to continue maintenance on the system, it would require that the system should be easy to understand, the technologies used should be available for an extended period of time and developers should be able to easily add new functionality to the system.

The code tested satisfies the condition that it should be able to understand the system. The system developed is not cluttered and difficult to understand due to good extraction between layers in the Model View Controller model. An obstacle in trying to understand the system is that the code has not been thoroughly documented. This causes some difficulties when trying to gain a better context about certain functions, why they are there and what their purposes are.

The technologies chosen satisfies the requirement of having to be available for a long period of time. The technologies used (such as Node js), are relevantly new platform and therefore they are constantly being updated. This will ensure long use of the technologies, enhancing maintainability.

Developers would be able to easily maintain the code in the sense of adding new features. This is made possible for example by making use of the simple way to add a new route to the program. There might however be slight difficulties when trying to change certain aspects of the system, due to the system being developed too specific, rather than focusing on adaptability. More emphasis was put on being able to easily add to the system, rather than changing the system, though it will still be possible it would only take longer than adding to the system.

2. Scalability:

In order for a system to be scalable, the system should be able to expand and scale in order to handle a larger load or to serve more clients.

The system satisfies scalability in the sense that it is currently able to handle all the modules for the computer science department. This is however only a small portion of what the system should be capable of doing. The system should theoretically be able to scale in order to service 50 000 students. This should not be a problem for the system, due to the technologies that it are using. Node js is a lot more scalable

than Apache, even though it is only single threaded by default. The use of MongoDB, rather than MySQL also increases the scalability of the server [1]. The scalability of Node.js can be increased by using multi-core CPUs rather than a single core CPU and load balancers can be implemented. This will ensure that the server application scales more than enough in order to be able to serve the required amount of users. This can however only be done when purchasing more efficient hardware for the host machine.

3. Performance:

The performance requirements set out in the formal specification stated that all non-reporting operations should respond within less than 0.2 seconds. The specification also stated that report queries should not take any longer than 5 seconds.

The performance testing was done using Firebug for Firefox. Almost all of the non-reporting requests responded well under 0.1 seconds. The only cases where the server did not respond fast enough, was when the page was initially requested. The initial page request resulted in a response time of 1.33s which was the worst case. On average the initial request for the homepage resulted in about 305ms response time. The next case that took longer than it should, is when a page is requested that does not exist. The reason for this decrease in performance is due to the server searching for a route that does not exist. The server will then eventually just give up and return the error page. This resulted in an average response time of about 286ms.

The overall average response time for requests is about 150ms as determined by firebug. Below are some screenshots created during the testing of non-reporting requests.

The performance for the reporting use cases exceeded expectations by performing well under the recommended 5s. This might however change as the database size increases. Testing was limited to the usage of a fairly small size database.

On average the generating of a report averaged at about 310ms. Below are some screenshots that was taken during the performance testing of the report requests.

4. Security

URL	Status	Domain	Size	Remote IP	Timeline
GET file-constraints	304 Not Modified	localhost:3000	12.0 KB	127.0.0.1:3000	29ms
GET jquery-1.11.2.min.js	304 Not Modified	localhost:3000	93.7 KB	127.0.0.1:3000	19ms
GET handlebars.runtime.min.js	304 Not Modified	localhost:3000	9.1 KB	127.0.0.1:3000	6ms
GET bootstrap.min.css	304 Not Modified	localhost:3000	114.6 KB	127.0.0.1:3000	6ms
GET bootstrap-theme.min.css	304 Not Modified	localhost:3000	19.5 KB	127.0.0.1:3000	6ms
GET style.css	304 Not Modified	localhost:3000	89 B	127.0.0.1:3000	9ms
GET bootstrap.min.js	304 Not Modified	localhost:3000	35.1 KB	127.0.0.1:3000	7ms

Figure 5: Non reporting based request for constraints

URL	Status	Domain	Size	Remote IP	Timeline
GET contributors	304 Not Modified	localhost:3000	4.5 KB	127.0.0.1:3000	6ms
GET jquery-1.11.2.min.js	304 Not Modified	localhost:3000	93.7 KB	127.0.0.1:3000	20ms
GET handlebars.runtime.min.js	304 Not Modified	localhost:3000	9.1 KB	127.0.0.1:3000	19ms
GET bootstrap.min.css	304 Not Modified	localhost:3000	114.6 KB	127.0.0.1:3000	19ms
GET bootstrap-theme.min.css	304 Not Modified	localhost:3000	19.5 KB	127.0.0.1:3000	19ms
GET style.css	304 Not Modified	localhost:3000	89 B	127.0.0.1:3000	19ms
GET bootstrap.min.js	304 Not Modified	localhost:3000	35.1 KB	127.0.0.1:3000	19ms

Figure 6: Non reporting based request for contributors

URL	Status	Domain	Size	Remote IP	Timeline
GET testPost	404 Not Found	localhost:3000	4.8 KB	127.0.0.1:3000	3ms
GET jquery-1.11.2.min.js	304 Not Modified	localhost:3000	93.7 KB	127.0.0.1:3000	8ms
GET handlebars.runtime.min.js	304 Not Modified	localhost:3000	9.1 KB	127.0.0.1:3000	6ms
GET bootstrap.min.css	304 Not Modified	localhost:3000	114.6 KB	127.0.0.1:3000	13ms
GET bootstrap-theme.min.css	304 Not Modified	localhost:3000	19.5 KB	127.0.0.1:3000	14ms
GET style.css	304 Not Modified	localhost:3000	89 B	127.0.0.1:3000	13ms
GET bootstrap.min.js	304 Not Modified	localhost:3000	35.1 KB	127.0.0.1:3000	12ms

Figure 7: Non reporting based request for test Post. Results in 404 error

GET report	304 Not Modified	localhost:3000	8.1 KB	127.0.0.1:3000	30ms
GET jquery-1.11.2.min.js	304 Not Modified	localhost:3000	93.7 KB	127.0.0.1:3000	17ms
GET handlebars.runtime.min.js	304 Not Modified	localhost:3000	9.1 KB	127.0.0.1:3000	16ms
GET bootstrap.min.css	304 Not Modified	localhost:3000	114.6 KB	127.0.0.1:3000	16ms
GET bootstrap-theme.min.css	304 Not Modified	localhost:3000	19.5 KB	127.0.0.1:3000	16ms
GET style.css	304 Not Modified	localhost:3000	89 B	127.0.0.1:3000	16ms
GET bootstrap.min.js	304 Not Modified	localhost:3000	35.1 KB	127.0.0.1:3000	15ms

Figure 8: Reporting based request

GET report	304 Not Modified	localhost:3000	8.1 KB	127.0.0.1:3000	30ms
GET jquery-1.11.2.min.js	304 Not Modified	localhost:3000	93.7 KB	127.0.0.1:3000	17ms
GET handlebars.runtime.min.js	304 Not Modified	localhost:3000	9.1 KB	127.0.0.1:3000	16ms
GET bootstrap.min.css	304 Not Modified	localhost:3000	114.6 KB	127.0.0.1:3000	16ms
GET bootstrap-theme.min.css	304 Not Modified	localhost:3000	19.5 KB	127.0.0.1:3000	16ms
GET style.css	304 Not Modified	localhost:3000	89 B	127.0.0.1:3000	16ms
GET bootstrap.min.js	304 Not Modified	localhost:3000	35.1 KB	127.0.0.1:3000	15ms

Figure 9: Reporting based request

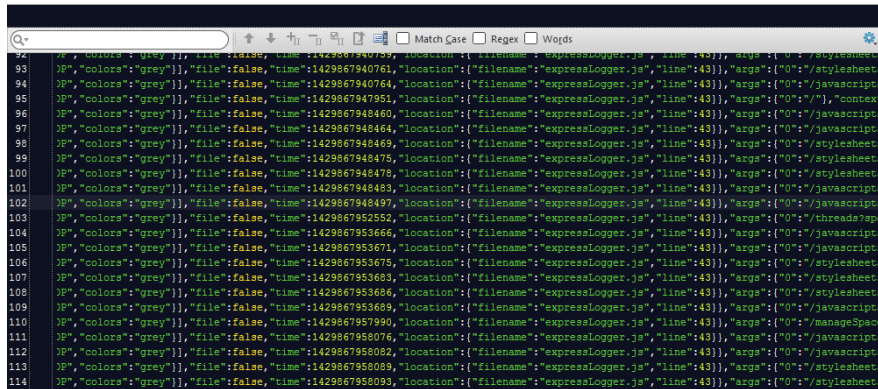
GET report	304 Not Modified	localhost:3000	8.1 KB	127.0.0.1:3000	30ms
GET jquery-1.11.2.min.js	304 Not Modified	localhost:3000	93.7 KB	127.0.0.1:3000	17ms
GET handlebars.runtime.min.js	304 Not Modified	localhost:3000	9.1 KB	127.0.0.1:3000	16ms
GET bootstrap.min.css	304 Not Modified	localhost:3000	114.6 KB	127.0.0.1:3000	16ms
GET bootstrap-theme.min.css	304 Not Modified	localhost:3000	19.5 KB	127.0.0.1:3000	16ms
GET style.css	304 Not Modified	localhost:3000	89 B	127.0.0.1:3000	16ms
GET bootstrap.min.js	304 Not Modified	localhost:3000	35.1 KB	127.0.0.1:3000	15ms

Figure 10: Reporting based request

The system is not secure in all areas. It requires a username and password in order for a user to login. The system successfully performs authentication against the LDAP repository. It does not allow users without the necessary authorisation to delete buzz spaces, but anyone can create a buzz space whether they are logged in or not. Only authorised users can add or remove admins. The system is also somewhat

configurable, MIME types can be added, edited, and removed, but the system does not take authorisation level or whether a user is logged in into account for these actions, anyone can perform these actions. Admin users cannot configure access to certain services.

The log in user interface:



```

92 X", "colors": "grey", "file": false, "time": 1429867940759, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
93 X", "colors": "grey", "file": false, "time": 1429867940761, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
94 X", "colors": "grey", "file": false, "time": 1429867940764, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
95 X", "colors": "grey", "file": false, "time": 1429867940795, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
96 X", "colors": "grey", "file": false, "time": 1429867940860, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
97 X", "colors": "grey", "file": false, "time": 1429867940864, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
98 X", "colors": "grey", "file": false, "time": 1429867940868, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
99 X", "colors": "grey", "file": false, "time": 1429867940875, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
100 X", "colors": "grey", "file": false, "time": 1429867940878, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
101 X", "colors": "grey", "file": false, "time": 1429867940883, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
102 X", "colors": "grey", "file": false, "time": 1429867940897, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
103 X", "colors": "grey", "file": false, "time": 1429867952552, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
104 X", "colors": "grey", "file": false, "time": 1429867953666, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
105 X", "colors": "grey", "file": false, "time": 1429867953671, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
106 X", "colors": "grey", "file": false, "time": 1429867953675, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
107 X", "colors": "grey", "file": false, "time": 1429867953683, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
108 X", "colors": "grey", "file": false, "time": 1429867953686, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
109 X", "colors": "grey", "file": false, "time": 1429867953689, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
110 X", "colors": "grey", "file": false, "time": 1429867953690, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
111 X", "colors": "grey", "file": false, "time": 1429867958076, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
112 X", "colors": "grey", "file": false, "time": 1429867958082, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
113 X", "colors": "grey", "file": false, "time": 1429867958089, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/
114 X", "colors": "grey", "file": false, "time": 1429867958093, "location": {"filename": "expressLogger.js", "line": 43}, "args": [{"0": "/stylesheets/

```

Determine if the user is authorised to close the buzz space:

```

closeBuzzSpaceRequest.callback("Did not find space that should be closed");
} else {
    // check if user is an admin for buzz space
    spaces.isAdministrator({
        userID: closeBuzzSpaceRequest.userID,
        moduleID: closeBuzzSpaceRequest.moduleID,
        callback: function (error, response) {
            if (error) {
                closeBuzzSpaceRequest.callback(error);
            } else if (response == false) {
                console.log("Not an admin of this buzz space");
                //closeBuzzSpaceRequest.callback(new Error("NotAuthorizedException"));
                closeBuzzSpaceRequest.callback("Not an admin of this buzz space");
            } else {

```

Assign an admin to a BuzzSpace:

5. Auditability:

The system is auditable. The system keeps a log of all the requests and responses. The requests does not have an id, but it does have a user id, a date/time stamp, it includes the request that was made, and it does not store any sensitive information.

For the responses, there is not an id for the log entry or a corresponding request entry, there is a date/time stamp, and there is no sensitive

```

*/
spaces.assignAdministrator = function (assignAdministratorRequest) {
  //check if the module exists
  Space.findOne({'moduleID': assignAdministratorRequest.moduleID}, function (err, result) {
    if (result == null) {
      //throw new Error("NoSuchBuzzSpaceException");
      console.log("Could not find buzz space to assign admin to");
    } else {
      console.log("Found module to assign admin to");
      // check if user is an admin for buzz space
      if (isAuthorized(assignAdministratorRequest.userID, assignAdministratorRequest.moduleID)) {
        result.adminUsers.push(assignAdministratorRequest.newAdmin);

        result.save(function (err) {
          //res.send(result);
        });
      } else {
        throw new Error("NotAuthorizedException");
      }
    }
  });
};

```

information included. Because the response entries does not have a way of indicating to which request it responded it can be difficult to make sense of the date or to trace errors.

The audit log can be accessed by humans and the system.

The screenshot shows a web application interface displaying an audit log. The log contains multiple entries, each with a timestamp, a file path, and a location. The entries are listed in a table-like format with line numbers on the left. The log shows various operations being performed, such as 'assignAdministrator' and 'manageSpace'.

An example of responses that are logged: A buzzSpace does not exist:

Could not find space to remove:

Request to manage space COS9:

Response that states a space was closed successfully:

6. Testability: Only some modules included unit tests. For example, threads and spaces: Unit testing for threads:

Integration tests are also performed in some instances.

The screenshot displays a web application interface with three main sections:

- LOGS - DATES:** A table listing log entries. The table has columns for ID, DATE, METHOD, STATUS, TAG, and URL. The logs are filtered by DATE (FRI APR 24 2015) and METHOD (EXPRESS). The table shows a list of log entries, including a detailed view of a specific log entry (ID 224) and a list of log entries (ID 216).
- Log Entry Details:** A detailed view of a log entry (ID 224) showing the URL `/manageSpace?space=C059` and the status `200`.
- Code Editor:** A code editor showing unit tests for the application. The code is written in JavaScript and includes a `describe` block for "Unit Tests for Threads" and an `it` block for "Test Post".

```

describe('Unit Tests for Threads', function() {
  var MockBuzz;

  var trigger = function() {
    throw new Error('Whoops!');
  };

  it('Test Post', function() {
    console.log("");
    console.log("Test Post:");

    console.log("Create MockBuzz");
    MockBuzz = {
      buzzSpaceId: "MockBuzzSpace",
      rootThread: new Thread(new Post("text", "Root thread content", Date.now(), "Root Thread Title",
        "admin_user_post", "information"), "MockBuzzSpace")
    };
    MockBuzz.rootThread.saveThread();

    test
    .object(MockBuzz).hasProperty("buzzSpaceId", "MockBuzzSpace")
    .object(MockBuzz.rootThread).hasProperty("_id", MockBuzz.rootThread._id)
    .object(MockBuzz.rootThread.post).hasProperty("mimeType", "text")
    .object(MockBuzz.rootThread.post).hasProperty("content", "Root thread content")
    .object(MockBuzz.rootThread.post).hasProperty("title", "Root Thread Title")
    .object(MockBuzz.rootThread.post).hasProperty("submitter", "admin_user_post")
    .object(MockBuzz.rootThread.post).hasProperty("postType", "information")
    .error(trigger)
    .hasMessage('Whoops!');
  });
});

```

In this instance authorisation is checked outside of the module that is was created in.


```

function checkIfAuthorizedUser(domain, service, objectIntercepted)
{
    if(objectIntercepted.userId === undefined)
    {
        console.log("could not authorize")
    }
    else
    {
        var userId = objectIntercepted.userId;
        var serviceIdentifier = new authorization.ServiceIdentifier(domain, service);
        var isAuthorizedRequest = new authorization.IsAuthorizedRequest(userId, serviceIdentifier);
        var isAuthorizedResult = new authorization.Authorization.IsAuthorized(isAuthorizedRequest);
        if(isAuthorizedResult.isAuthorized === true)
        {
            //console.log("Authorized");
        }
        else
        {
            //console.log("Not Authorized");
            throw {'status':401,'message':'Unauthorized Access Restricted'};
        }
    }
}

```

For both the user tests and the integration tests it is verified that all the pre-conditions are met

The system is thus only testable in some cases, but there is a lot of potential to add further test cases as the system is set up in such a way that makes testing more convenient.

7. Usability:

The system is usable because firstly the necessary actions that a user can take appear in a navigation bar at the top of the screen. Thus it is easy for a novice user to be able to know what to click on and navigate the website.

The interface is not cluttered, and only basic functionality is displayed on the home screen, making the system more learnable. The buttons are labelled with text rather than with graphical icons, and the text on the button is quite explanatory, which makes their purpose more clear.

Larger headings are used to label the different sections, for example under the Manage Constraints tab there are large headings to indicate the Existing constraints section and the Add new constraint section. This again contributes to ease of use for the novice first year user.

Through these mechanisms the system is memorable hence it is also be understandable.

8. Integratibility:

Buzz++@UP

A Discussion Board Created For The COS301 Mini Project 2015

[Home](#)
[Post](#)
[Manage Constraints](#)
[Login](#)
[Appraisal](#)
[Reports](#)
[Contributors](#)

Active Buzz Spaces

789 - Tonia
Year: 2020

111 - Module111
Year: 2000

3 - 3
Year: 2000

489 - 4897
Year: 2000

78 - 7887
Year: 78

000 - COS000
Year: 2014

frk111 - FRK111
Year: 2015

COS314 - Artificial Intelligence
Year: 3

COS332 - Networks
Year: 3

Buzz++@UP

A Discussion Board Created For The COS301 Mini Project 2015

[Home](#)
[Post](#)
[Manage Constraints](#)
[Login](#)
[Appraisal](#)
[Reports](#)
[Contributors](#)

Welcome to COS 332

[Manage Space](#)

Threads in this space

Title: Why wont my TARDIS work

like teh troll

Follow

Reply

Title: Ummmmmmmm

hope the lecturers are happy

Follow

Reply

Title: Write thread title here

rolling is fun!!!!

Unfollow

Reply

© 2015 Buzz++

The system is able to address future integration requirements by providing access to its services using widely adopted public standards such as firstly having separate npm packages for all the different modules. The

26

packages are stored on Synopia. Also, electrolyte is being used in the server to provide a dependency injection. The HandleBars server is the main server that needs to be used to test and integrate all the modules on. The routes/index.js, routes/infrastructure.js and routes/content.js files use express to route the different hbs files for the different modules in order to integrate the infrastructure and content subsystems into the main system.

```

15 var querystring = require('querystring');
16 module.exports = function (router, database, authentication, csds, spaces, notification) {
17   var mongoose = database.mongoose;
18
19   function FindUserModules(memberId, result) {
20     // Find memberId - get from session
21     csds.findUserModules(memberId, function (res) {
22       try {
23         result.render('./infrastructure', {modules: res});
24       } catch (err) {
25         console.log(err);
26       }
27     });
28   }
29
30   function GetActiveModulesForYear() {
31     csds.getActiveModulesForYear(function (res) {
32       try {
33         console.log(res);
34       } catch (err) {
35         console.log(err);
36       }
37     });
38   }
39
40   /**
41    * Handles login based on username and password.
42    * @param username (String)
43    * @param password (String)
44    * @param res res from calling middleware
45    * @param req
46    * @param lastPage (String) URL of page that needs to be opened after login request
47    */
48   function login(username, password, req, res, lastPage) {
49     if (app.get('env') !== 'production' && username === 'u12345678') {
50       var session = req.session;
51       session.userId = username; //this is assuming we only log in with our userid, which are our student n
52       if (lastPage) {
53         res.redirect(lastPage);
54       } else {
55         res.redirect("/");
56       }
57     } else {
58       csds.login(username, password, function (data) {
59         if (data === true) {
60

```

A separate file is used to establish the connection to the database to avoid having this done in all the separate files. Also, global variables are now used such as the global password and username for example.

A document has been provided via email and a README file has been provided to specify important standards and regulations that must be followed.

The functional code in the separate packages must be placed in an exportable function taking parameters such as the database or settings, and this is done as part of the electrolyte dependency injection.

Exports are also used in the different files to make the code accessible to the other files.

```

9 | <div class="form-group">
10 |   <input type="text" value="Create Space">
11 |   <input type="text" value="Academic Year">
12 |   <input type="button" value="Submit" />
13 | </div>
14 |
15 | <script>
16 |   $(document).ready(function() {
17 |     // Create new Buzz Space
18 |     $('#submit').click(function() {
19 |       // Create new Buzz Space
20 |       $.ajax({
21 |         url: 'ajaxNewSpace',
22 |         method: 'POST',
23 |         success: function(data) {
24 |           // Assuming threadID of thread which has been updated will be received in jsonObj
25 |           // Query database to find user(s) who are subscribed to the thread and send email to each
26 |           // var dataset = nonopb model('Notifications_Thread', new Schema({
27 |             notificationFollowing: Boolean,
28 |             notificationThreadID: String
29 |           }));
30 |           dataset.find({'notificationThreadID': jsonObj.threadID}, function (err, data) {
31 |             if (err == null) {
32 |               info = data;
33 |               console.log("User has been registered to get moved thread notifications - " + info);
34 |             } else {
35 |               console.log("Not found in database.");
36 |               return false;
37 |             }
38 |           });
39 |           return true;
40 |         },
41 |         error: function(err) {
42 |           console.log(err);
43 |         }
44 |       });
45 |     });
46 |   });
47 |
48 |   notification.appraisalRegister = function (jsonObj) {
49 |     new appraisals({
50 |       notificationStudentID: jsonObj.studentID,
51 |       notificationAppraisalType: jsonObj.appraisalType
52 |     }).save(function (err, doc) {
53 |       if (err != null) {
54 |         console.log(err);
55 |       } else {
56 |         console.log("User has been registered to appraisal notifications - " + doc);
57 |       }
58 |     });
59 |   });
60 |
61 |   notification.appraisalDeregister = function (jsonObj) {
62 |     appraisals.remove({
63 |       notificationStudentID: jsonObj.studentID,
64 |       notificationAppraisalType: jsonObj.appraisalType
65 |     }, function (err) {
66 |       if (err != null) {
67 |         console.log(err);
68 |       } else {
69 |         console.log("User has been removed from table");
70 |       }
71 |     });
72 |   });
73 |
74 |   return false;
75 | });
76 |
77 |

```

The following screenshot indicates the systems' integratibility because it is clear that the following function is able to accept standards such as json strings for objects. This is intergrateble because the interface does not know which functions are sending the aforementioned objects. Therefore it allows the interface to be connected to any backend system.

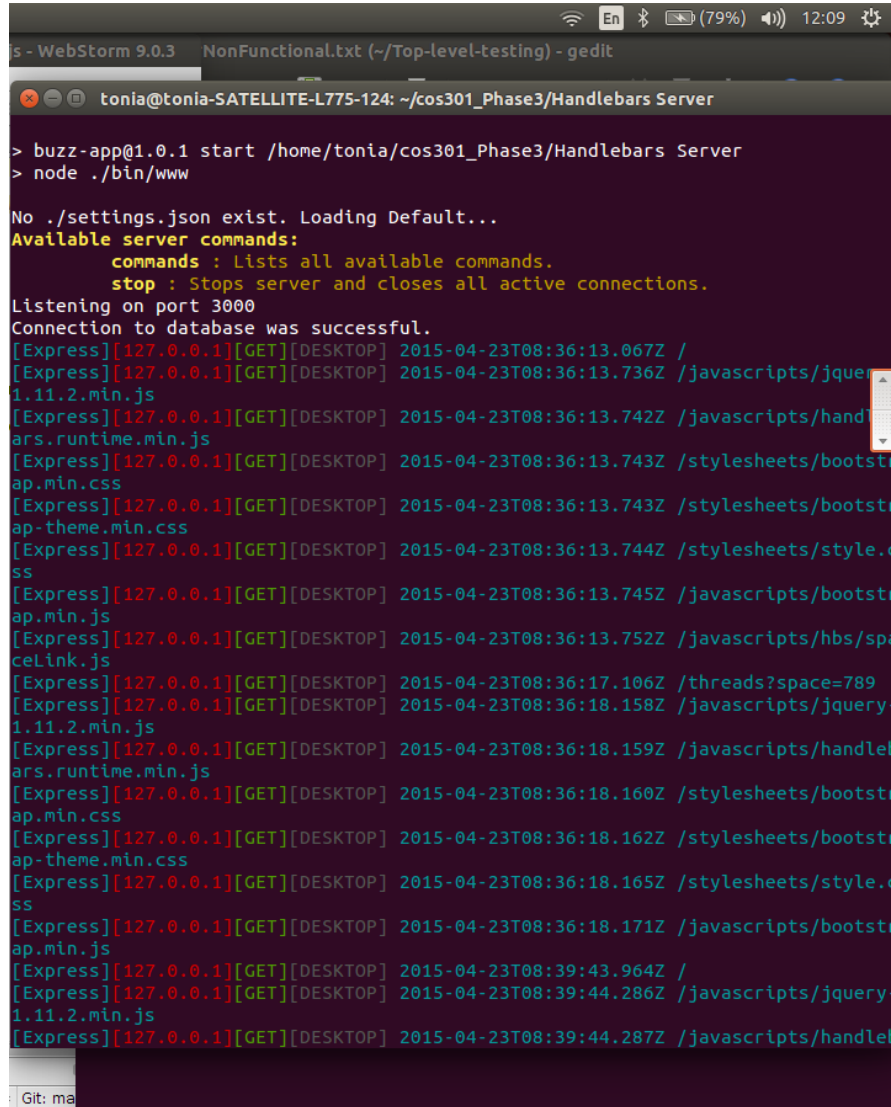
```

241 | //Querying database to find user(s) who need to be sent emails,
242 | //Assuming threadID of thread which has been updated will be received in jsonObj
243 | //Query database to find user(s) who are subscribed to the thread and send email to each
244 | //var dataset = nonopb model('Notifications_Thread', new Schema({
245 |   notificationFollowing: Boolean,
246 |   notificationThreadID: String
247 | }));
248 | dataset.find({'notificationThreadID': jsonObj.threadID}, function (err, data) {
249 |   if (err == null) {
250 |     info = data;
251 |     console.log("User has been registered to get moved thread notifications - " + info);
252 |   } else {
253 |     console.log("Not found in database.");
254 |     return false;
255 |   }
256 | });
257 | return true;
258 |
259 |
260 |
261 | notification.appraisalRegister = function (jsonObj) {
262 |   new appraisals({
263 |     notificationStudentID: jsonObj.studentID,
264 |     notificationAppraisalType: jsonObj.appraisalType
265 |   }).save(function (err, doc) {
266 |     if (err != null) {
267 |       console.log(err);
268 |     } else {
269 |       console.log("User has been registered to appraisal notifications - " + doc);
270 |     }
271 |   });
272 |
273 |   notification.appraisalDeregister = function (jsonObj) {
274 |     appraisals.remove({
275 |       notificationStudentID: jsonObj.studentID,
276 |       notificationAppraisalType: jsonObj.appraisalType
277 |     }, function (err) {
278 |       if (err != null) {
279 |         console.log(err);
280 |       } else {
281 |         console.log("User has been removed from table");
282 |       }
283 |     });
284 |   });
285 |
286 |   return false;
287 | });
288 |
289 |

```

9. Deployability:

The system is deployable on Linux servers as we have run it using Ubuntu 14.04 Linux and the system was able to run. The following screenshot shows the system running on a Linux server:

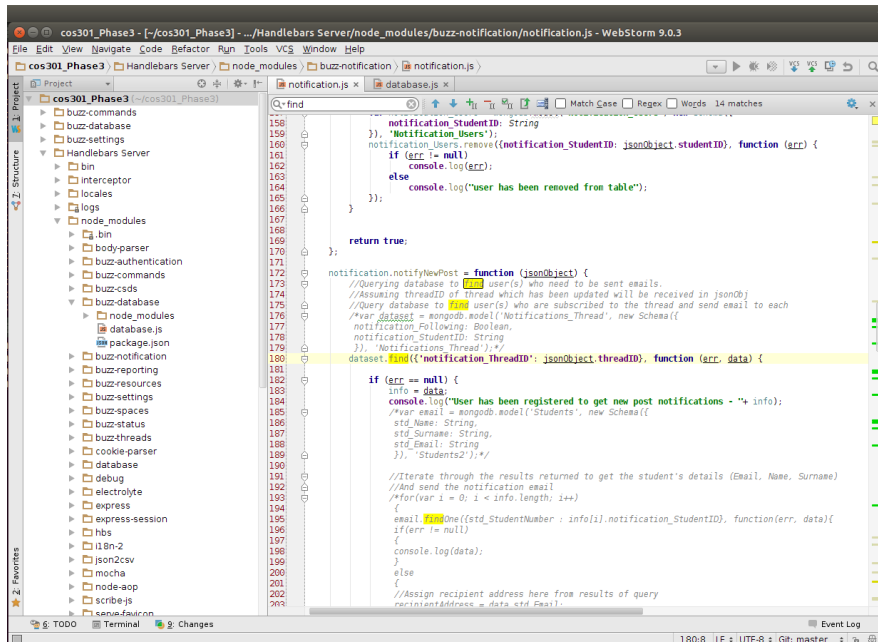


```
s - WebStorm 9.0.3  NonFunctional.txt (~~/Top-level-testing) - gedit
tonia@tonia-SATELLITE-L775-124: ~/cos301_Phase3/Handlebars Server
> buzz-app@1.0.1 start /home/tonia/cos301_Phase3/Handlebars Server
> node ./bin/www

No ./settings.json exist. Loading Default...
Available server commands:
  commands : Lists all available commands.
  stop : Stops server and closes all active connections.
Listening on port 3000
Connection to database was successful.
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:13.067Z /
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:13.736Z /javascripts/jquery
1.11.2.min.js
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:13.742Z /javascripts/handle
ars.runtime.min.js
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:13.743Z /stylesheets/bootst
ap.min.css
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:13.743Z /stylesheets/bootst
ap-theme.min.css
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:13.744Z /stylesheets/style.
ss
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:13.745Z /javascripts/bootst
ap.min.js
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:13.752Z /javascripts/hbs/sp
ceLink.js
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:17.106Z /threads?space=789
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:18.158Z /javascripts/jquery
1.11.2.min.js
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:18.159Z /javascripts/handle
ars.runtime.min.js
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:18.160Z /stylesheets/bootst
ap.min.css
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:18.162Z /stylesheets/bootst
ap-theme.min.css
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:18.165Z /stylesheets/style.
ss
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:36:18.171Z /javascripts/bootst
ap.min.js
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:39:43.964Z /
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:39:44.286Z /javascripts/jquery
1.11.2.min.js
[Express][127.0.0.1][GET][DESKTOP] 2015-04-23T08:39:44.287Z /javascripts/handle
ars.runtime.min.js
```

The system is deployable on an environment using different databases for persistence of the Buzz database because the Handlebars server contains a folder called node_modules, and it contains the buzz_database

package. This package can easily be swapped out and an alternative database package can be plugged in, that the system can use due to the flexibility of this server. As long as the new package has the same name so that the files that require the database do not need to be changed, no major changes will need to be made.



The system is deployable in environments where the user authentication credentials and roles are sourced from different repositories. The following screenshot indicates that the Handlebars server contains a folder called `node_modules`, and it contains the `buzz_csd`s package.

This package can easily be swapped out and an alternative data source package with different credentials and roles can be used, due to the pluggability of the system and dependency injection employed by the top level team. The new package must just be named the same as the old one to avoid errors where the file is required from the package in the code.

The screenshot shows the WebStorm 9.0.3 IDE with the `cos301_Phase3` project open. The file explorer on the left shows the project structure, with `node_modules/buzz-database/database.js` selected. The main editor displays the contents of `database.js`, which is a module that connects to a MongoDB database using Mongoose. The code includes comments, imports, and a function to connect to the database. The status bar at the bottom indicates the file is at line 23, column 1, in UTF-8 encoding, and is part of the `master` branch in a Git repository.

```
1  /**
2   * @param settings
3   * @returns {db: (mongoose.connection), mongoose: Mongoose}
4   */
5  module.exports = function(settings) {
6    /**
7     * @type {Mongoose}
8     */
9    var mongoose = require('mongoose');
10    mongoose.connect(settings.database);
11
12    var db = mongoose.connection;
13    db.on('error', console.error.bind(console, 'connection error:'));
14    db.once('open', function(callback) {
15      console.log("Connection to database was successful.");
16    });
17
18    return {db: db, "mongoose": mongoose};
19  };
20
21 module.exports['singleton'] = true;
22 module.exports['require'] = ['buzz-settings'];
23
```

The screenshot shows the WebStorm 9.0.3 IDE with the `cos301_Phase3` project open. The file explorer on the left shows the project structure, with `node_modules/buzz-csds/csds.js` selected. The main editor displays the contents of `csds.js`, which is a module that exports various functions for interacting with the CSDS database. The code includes comments, imports, and a function to connect to the database. The status bar at the bottom indicates the file is at line 17, column 1, in UTF-8 encoding, and is part of the `master` branch in a Git repository.

```
1  /**
2   * Created by collins on 2015/03/24.
3   */
4  module.exports = function(database) {
5    var csds = new Object();
6    csds.login = require('./CSDS_Modules/Login').login;
7    csds.findUserModules = require('./CSDS_Modules/FindUserModules').findUserModules;
8    csds.getUserRolesForModules = require('./CSDS_Modules/GetUserRolesForModules').getUserRolesForModules;
9    csds.getActiveModulesForYear = require('./CSDS_Modules/GetActiveModulesForYear').getActiveModulesForYear;
10    csds.getUsersWithRole = require('./CSDS_Modules/GetUsersWithRole').getUsersWithRole;
11
12    return csds;
13  };
14
15 module.exports['require'] = ['database'];
16 module.exports['literal'] = false;
17
```

2.2 Top Level B Testing Results

2.2.1 Functional Testing

1. Buzz Resources Module

Description

The system suppose to integrate a resource module that is used to upload and manage resources like media files and documents that can either linked or embedded to a post.

Preconditions

- Size constraints must be met
- The resource mimetype must be detected
- The resource to must be supported

Post-conditions

- Url for resource must be generated
- Resource must persist

Test Results

The Buzz resources module works when test in isolation through unit testing but fails on automated integration testing, it was not integrated as required by system specification.

2. Buzz Notification Module

Description

This module is focused on registering to receive notification messages for submitted threads and post of a particular user and any notification one receive for their posts via email.

Preconditions

- User must have registered to receive notifications

Post-conditions

- User should receive all notification based on the domain (receive from all threads or a particular thread) the user registered to receive notifications.

Test Results

The Buzz notification was not integrated for this system, no email notifications are sent when a thread or post is submitted.

3. Buzz-Status module

The Buzz-Status module was expected to at least implement these 6 functional processes:

(a) assessProfile:

- **Objective:** Serve as a general interface through which lecturers can choose a pluggable implementation with which to assess user profiles.
- **Pre-conditions:** Have some form of administrative and general users. Have profiles for users of the Buzz system.
- **Post-conditions:** ProfileAssessor can return queried information about a users profile according to various criteria but must, as a minimal requirement, at least be able to calculate the user's status (i.e. the user's rating).
- **Testing:** Although both pre-conditions were met, no actual implementation for this functional requirement is present in the Buzz system (meaning post-conditions not met).

(b) setStatusCalculator:

- **Objective:** An interface through which a StatusCalculator can be assigned. The StatusCalculator is used to assign a status value to a user according to specific criteria. The default StatusCalculator should be NumPostAssesor which assigns a status directly proportional to the number of posts a user has made.
- **Pre-conditions:** Have some form of administrative and general users. Have profiles for users of the Buzz system. Have a space where users can post content (this is a pre-condition for the default StatusCalculator, i.e. NumPostAssessor).
- **Post-conditions:** The assigned StatusCalculator must be able to assign a status value to all users.
- **Testing:** Only the first two pre-conditions are met. Although there is a space where users can post content (i.e.

<http://buzz-codechat.rhcloud.com/spaces/>) it does not fulfil any of its required functionality (i.e. can't successfully post content, no thread handling). There is no implementation of a StatusCalculator present in the Buzz system, therefore no post-conditions were met.

(c) `getStatusForProfile`:

- **Objective:** A simple query which returns the user's status.
- **Pre-conditions:** Have profiles for users of the Buzz system. Each user has a status.
- **Post-conditions:** The user's status is returned.
- **Testing:** Only the first pre-condition is met (i.e. user's do not have statuses). There is no way of storing a user's status in the Buzz system, it therefore fails to meet the post-condition.

(d) `createAppraisalType`:

- **Objective:** Be able to create a data structure by which posts can be graded/appraised (e.g. `FunnyAppraisalType` allows user to mark a post as Hilarious, Funny or Boring).
- **Pre-conditions:** Have a space where posts can be displayed.
- **Post-conditions:** Have a user created data structure which can then be used to appraise/rank a post.
- **Testing:** The pre-condition is not met as although a space for posts exists, none of the required functionality exist (e.g. being able to post new content in a thread). The post-conditions are not met as no functions in the Buzz system implements the required functionality.

(e) `activateAppraisalType`:

- **Objective:** Assign an appraisal type (which was created by `createAppraisalType`) to be used in a specific Buzz space for a specific period of time.
- **Pre-conditions:** Have a space where posts can be displayed. Have a function(s) which implements `createAppraisalType`.
- **Post-conditions:** Be able to use user created appraisal types in a certain buzz space for a specific period of time.
- **Testing:** Neither pre-conditions are met as the Buzz space does not function properly and no version of `createAppraisalType`

was implemented. This function was not implemented which means no post-conditions were met.

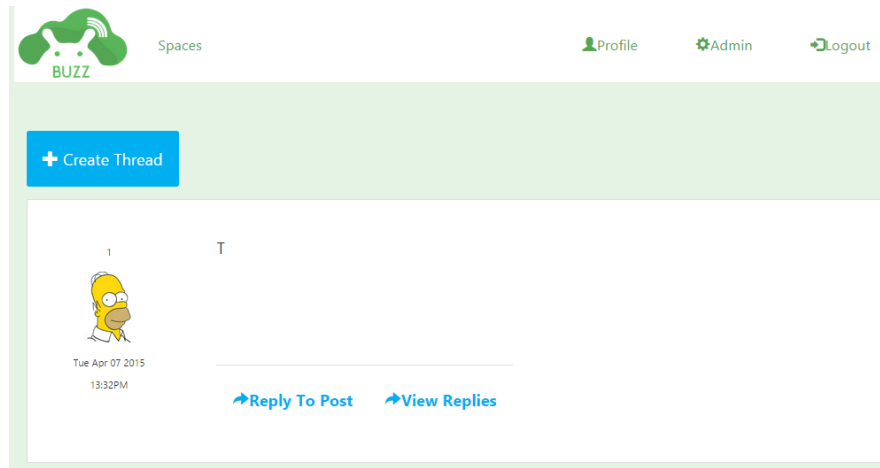


Figure 11: There are no appraisal icons with which to "upvote" posts

(f) assignAppraisalToPost:

- **Objective:** Allows the user to assign an appraisal to a post (i.e. rate a post).
- **Pre-conditions:** Have a space where posts can be displayed.
- **Post-conditions:** The ranking/status of both the post and the user who posted the post will change.
- **Testing:** The pre-condition is not met as the Buzz space does not function properly. This function was not implemented which means no post-conditions were met.

4. Threads module

The Threads module was expected to at least implement these 6 functional processes:

(a) submitPost:

- **Objective:** Used to submit a post to a thread by creating a child thread.

- **Pre-conditions:** The buzz space must be active (i.e. exist). The user submitting the post must exist and be a registered user on that buzz space.
- **Post-conditions:** Either create a post and append it as a child to the current thread or return the relevant error message should something go wrong.
- **Testing:** Both pre-conditions are met. Although there is a function to generate threads in the Buzz system (createNewThread) it does not function as the submitPost functional requirements dictate it should. createNewThread allows for the creation of threads without necessarily appending it as a child to a parent thread (which submitPost does not allow). Furthermore createNewThread is hard-coded with dummy info and is clearly used for simple display purposes (i.e. to show how a thread might have looked if the thread handling functions had been fully implemented). Since no proper implementation of submitPost is present in the Buzz system it therefore fails the post-conditions.

(b) markPostAsRead:

- **Objective:** Create a data object when a user reads a post. The data object will contain information about the reading event such as which user read the post at what time.
- **Pre-conditions:** This function requires a post and a user to read the post so that it can save the relevant information regarding the reading event.
- **Post-conditions:** A data object containing information about the reading event is stored or an error message is returned.
- **Testing:** All pre-conditions are met but there is no function in the buzz system which implements this functional requirement, it therefore fails the post-conditions.

(c) closeThread:

- **Objective:** This closes a thread and prevents it from being further manipulated by the user. It also gives the user to either manually or autonomously summarise the thread.

```

//This is what the functional integration teams had to write
function createNewThread(subject, post, parentThread, module) {

    var Thread = ds.models.thread;
    //Find the level of the thread whose object_id is the same as the parent thread's ID (i.e. find the parent's level)
    if (parentThread == null) {
        var level = 1;
        //Construct a new Thread
        var newThreadJSON = {
            "ParentID": null,
            "UserID": "u12345678",
            "NumChildren": 0,
            "Closed": false,
            "Hidden": false,
            "Level": level,
            "Post": post,
            "Subject": subject,
            "Module": module
        };
        //Post it to the Database
        postThreadToDatabase(newThreadJSON);
    }
    else {
        Thread.findOne({'_id': parentThread}, function (err, parentThread) {
            if (err) {
                console.log("ERR: " + err);
            }
            else {
                //Set the child thread's level
                var parentLevel = parentThread.level;
                var parentID = parentThread._id;
                var level = 0;
                level = Number(parentLevel) + 1;
                //Construct a new Thread
                var newThreadJSON = {
                    "ParentID": parentID,
                    "UserID": "u12345678",
                    "NumChildren": 0,
                    "Closed": false,
                    "Hidden": false,
                    "Level": level,
                    "Post": post,
                    "Subject": subject,
                    "Module": module
                };
                //Post it to the Database
            }
        });
    }
}

```

Figure 12: Implementation of the createNewThread function

- **Pre-conditions:** A thread to close and a user which has administrator priviledges.
- **Post-conditions:** The thread is closed and made inaccessible to general users. If the summariser is implemented then a summary is made.
- **Testing:** All pre-conditions are met but there is no function in the buzz system which implements this functional requirement, it therefore fails the post-conditions.

(d) moveThread:

- **Objective:** Be able to detach a sub-tree of threads from one thread and add it to another.
- **Pre-conditions:** A minimum of two threads (one to move and one from which it can be detached or attached to).
- **Post-conditions:** The threads sub-tree is either successfully

moved or an error message is displayed.

- **Testing:** All pre-conditions are met but there is no function in the buzz system which implements this functional requirement, it therefore fails the post-conditions.

(e) `hideThread`:

- **Objective:** This is used to mark a thread as hidden. Hidden threads (and all their child threads) should be made inaccessible to the users and should not be rendered by the user interface.
- **Pre-conditions:** This simply requires a thread to mark as hidden (a user is not required as this function is not directly accessed by users).
- **Post-conditions:** The thread (and all its descendants) are marked as hidden and are not displayed on the user's interface.
- **Testing:** All pre-conditions are met but there is no function in the buzz system which implements this functional requirement, it therefore fails the post-conditions.

(f) `queryThread`:

- **Objective:** This function should return subsets of posts according to a range of user restrictions.
- **Pre-conditions:** This function requires the user to specify the following restrictions for their query: `startDateTime` (return posts after this `dateTime`), `endDateTime` (return posts before this `dateTime`), `maxLevel` (return posts which are, at most, at this depth relative to the current post), `minLevel` (return posts which are, at least, at this depth relative to the current post), `userGroup` (only return posts which were posted by these specific users) and `phraseSet` (only return posts which contain all of these phrases).
- **Post-conditions:** The query should return specific information of all the posts which conform to the user specified restrictions. The information to be returned include: `ParentID`, `Author`, `TimeStamp`, `Content` and `Status` (e.g. closed or hidden).
- **Testing:** All pre-conditions are met but there is no function

in the buzz system which implements this functional requirement, it therefore fails the post-conditions

5. Buzz data source Module:

This module is responsible for sourcing data from the external cs database (Read Access) and using it for authentication and representation purposes.

Use Cases

- Login and Administrative user

This is the service that will authenticate against the cs data source where authentication credentials are stored

- Pre-conditions:

- could connect to CS data source (LDAP) (Not implemented-The system does not connect or acquire data from the computer science database)

- user exists in ldap with provided authentication details (Implemented-The system has users that can connect to the system but are not in the cs data source, dummy users were created to provide the login functionality)

- Post-conditions:

- userID returned (not implemented-Code was found but no functionality)

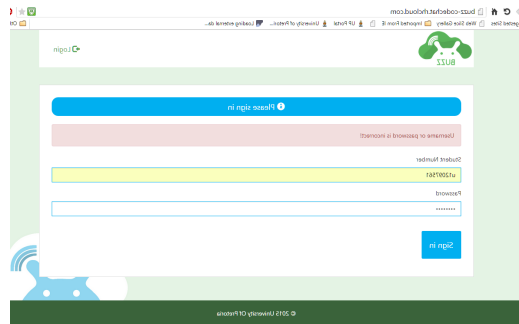


Figure 13: result to a user from the cs database trying to login

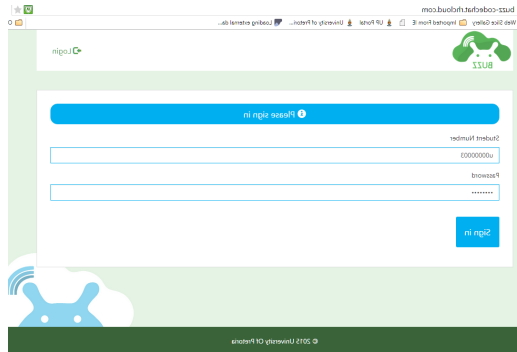


Figure 14: result to a dummy user not in the cs database trying to login

- getUsersRolesForModule

This service is to query the user roles for a particular user

- Pre-conditions:
 - could connect to CS data source (LDAP) (not implemented- No modules from cs data source where found , only one hard coded one)
 - user exists in ldap with provided authentication details (Not Implemented)
- Post-conditions:
 - userID returned (not implemented)

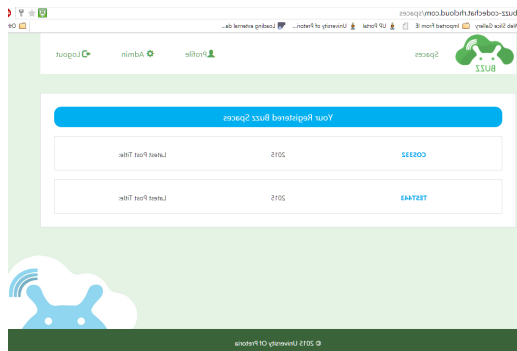


Figure 15: result to a dummy user not in the cs database trying to login

- `GetUsersWithRole`
This service is required to retrieve all the users which have a role for a particular module
- Pre-conditions:
-could connect to CS data source (LDAP) (not implemented-
Not Implemented) No code found and no administrative users on site

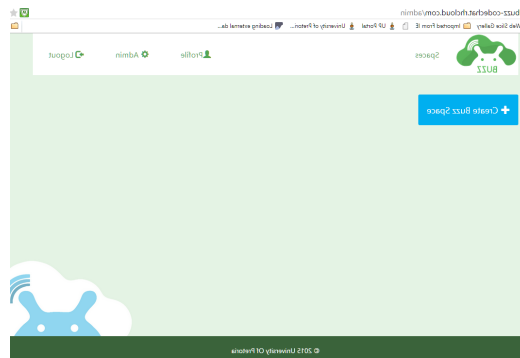


Figure 16: result to a dummy user not in the cs database trying to login

6. The Web Module:

The module provides a portal based, web responsive front end that allows a user to get access to the application facets.

Sections

- Login/out
The buzz system does allow user name `u00000000n` (with `n` being any single number starting from 1) and password "password" to login and out of the buzz system.
- personalization and configuration
A user cannot configure a dashboard type of view or buzz space, no functionality or options of such where provided.

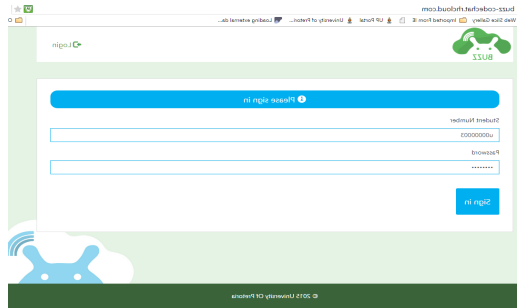


Figure 17: login format

- Accessibility of functionality
Access to a service does not depend on a user's role on a buzz space or a user's status on that space. There is no form of grouping, all users are the same and have the same privileges and authorisations, basically all users have access to everything in the system.

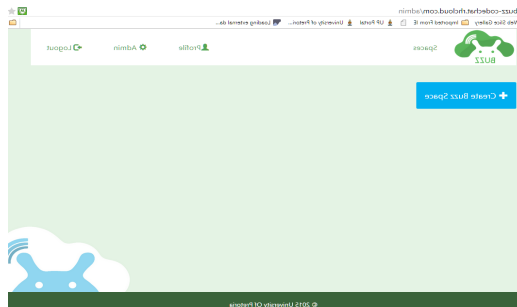


Figure 18: User roles in the system

- spaces
In the buzz space, users are able to view the buzz spaces that are available, no clear distinction of modules registered for and modules they have not yet registered for is presented
- UI Considerations around threads
Threads are posted as a flat list of posts showing the contents of each post, no user can post; only view posts hard coded in to the system



Figure 19: threads on the space

- posts
Posts have header, content and footer area, in the buzz space the header contains the user name, the date/time and a profile picture of the user who submitted the post. The footer contains the submission buttons, no appraisal values, no appraisal types and no posts can be submitted, created or replied to

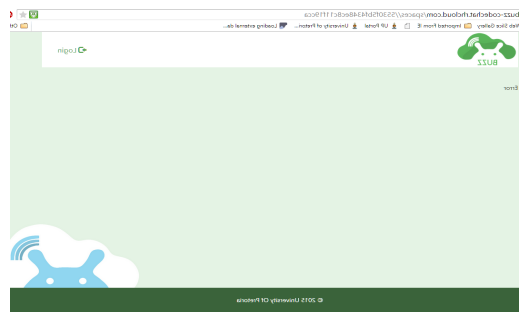


Figure 20: error received when user tries to post

- profile information
A user can view their own profile picture and are given an option to upload a new one yet that functionality does not work, a user cannot view their user name, role of the user or status and the number of posts/appraisals made/received by the user

2.2.2 Non-functional Testing

1. Maintainability:

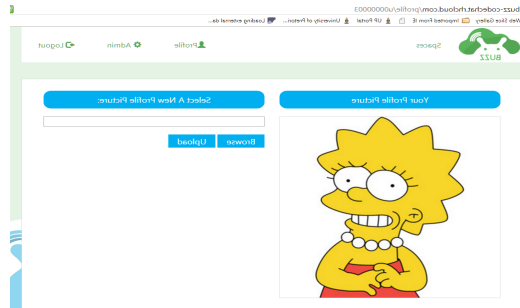


Figure 21: Dummy Profile data

The Buzz(B) system is maintainable in the following aspects:

- Understandable by future developers:
Every subsystem or 'module' of the system is implemented as a separate functioning module. Standalone modules are then plugged into the system. New developers can change modules independently.
- Technologies used is current and expected to be available for a long time:
Node.js makes up the core of the buzz system. It is a widely used open-source platform released in Q1 of 2009. Popularity has grown tremendously and is not expected to stop growing in the near future.
MongoDB is used as a database structure of the Buzz system and is highly compatible with Node.js. Also released in 2009 and is now the fourth most popular type of database management system. MongoDB is expected to grow even more in the coming years which makes it a good choice for the Buzz system in terms of maintainability.
- The system is easily updated
New functionality can be added by adding modules to the system. Functionality can be changed by changing independent modules without having to rebuild the whole system.

2. Scalability

The Buzz(B) system is partially scalable in the following aspects:

- Able to host buzz spaces for all Computer Science modules:
The Buzz(B) system failed this scalability test. It does not support functionality of adding a new buzz space thus cannot host buzz spaces for all CS modules.
- University-wide, servicing in the order of 50 000 students:
The Buzz(B) system failed this scalability test. It does not support functionality of adding a new user or student. Thus one default user exists as apposed to 50 000.

3. Auditability:

Description

The system needs log all requests and all responses for all user services provided by the system.

Test Results

The figure below shows that the Buzz Space system is partially auditable, only the requests for the user serves of the system are logged. The logged request only contains:

- User serves requested.
- Request object stringified as JSON

There are no log responses that are provided by the system, and the serves required to extract information from the audit logs is also not available not provided.

```
Fri Apr 24 2015 01:14:42 GMT-0700 (Pacific Daylight Time):
Node server started on 127.0.0.1:80 ...
Connected to mongo server.
GET / 200 103.927 ms - 4844
GET /stylesheets/style.css 304 13.117 ms - -
GET /stylesheets/Logo.png 304 27.288 ms - -
GET /js/View.js 304 30.342 ms - -
GET /stylesheets/bg.gif 304 30.193 ms - -
Validating password : password
POST /login 302 1902.391 ms - 70
In function isLoggedInIn: REQ:
{ _id: 552f6eae78c9203005711800,
  profile_pic: 'u00000004.png',
  status_value: 7,
  post_count: 0,
  username: 'Marge',
  user_id: 'u00000004',
  __v: 0,
  modules: [ 'COS301', 'COS332', 'COS314' ],
  roles:
    [ { _id: 552f6eae78c9203005711803,
        module: [Object],
        role_name: [Object] },
      { _id: 552f6eae78c9203005711802,
        module: [Object],
        role_name: [Object] },
      { _id: 552f6eae78c9203005711801,
        module: [Object],
        role_name: [Object] } ] }
GET /spaces 304 736.758 ms - -
GET /stylesheets/style.css 304 0.986 ms - -
GET /js/View.js 304 1.281 ms - -
GET /stylesheets/Logo.png 304 1.119 ms - -
```

Figure 22: Buzz Space B Auditability Test

4. Testability:

Description

A system needs testabl through:

- Unit testing components in isolation
- Integration tests where the components are integrated to the actual environment

Test Results

The Buzz system is divided into manageable components (modules) for each use case. Breaking the system into these components makes make the system testable through unit testing using mock objects. The Buzz system is a modular system, the system components are pluggable and makes automated integration testing simple as the component is integrated in the entire system.

5. Performance requirements:

Requirements for the Buzz system were for non-reporting operations to preferably not last longer than 0.2 seconds and for report queries to be processed in no more than 5 seconds. To test this we used Firebug, an open source web browser extension of Firefox used for webpage monitoring. (Version used: Firebug 2.0.7). The Buzz system met these requirement:

- Non-reporting operations such as returning to a previous page (i.e. operations that relied on cached copies of a webpage) completed relatively close to the target time limit (e.g. returning to the main Buzz Spaces page after viewing the user’s profile took only 0.286 seconds, very close to the target time of 0.2 seconds).
- Report query operations (i.e. which had to communicate and receive some information from the system’s database) also completed within their desired time limit of 5 seconds (e.g. accessing the test module “TEST443”’s discussion space took only 1.24 seconds). Even some of the report query operations which took comparatively long to complete stayed within the 5 second limit (e.g. accessing module “COS332” took 4,81 seconds and accessing the Admin tab took 4,96 seconds).



Figure 23: Non-reporting operation test

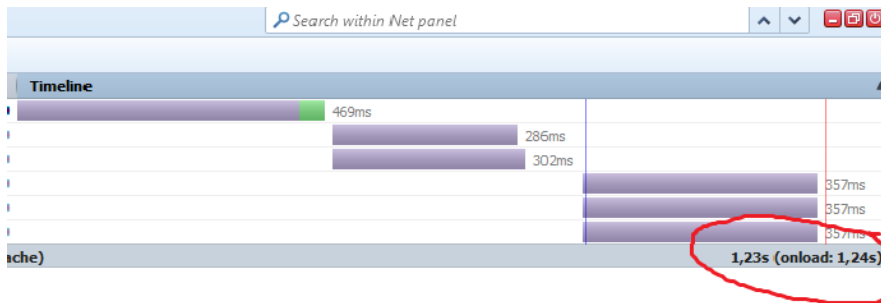


Figure 24: Reporting operation test

6. Reliability and availability:

- **textbfAvailability:** The Buzz system can be accessed at the following URL (<http://buzz-codechat.rhcloud.com/>) at any time of day on any of the widely used web browsers (Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Safari and Opera). The Buzz system even functions and displays correctly on the web browsers of mobile devices to ensure maximum availability.
- **textbfReliability:** The system performs various error checks to ensure that no user action could cause the Buzz system to fail/crash (e.g. returning error messages if the user tries to log on with incorrect information , Figure 3). Even in cases where the system's functionality was not fully implemented procedures are in place to rather warn the user of incomplete operations instead of blindly

carrying out a harmful request (e.g. an error message is displayed when a user tries to upload a new profile picture, Figure4). The reason why the Buzz system can still function (to an extent) even though much of its functional procedures are incomplete/missing is due to the modular fashion in which the system was created. Even if one module of the system fails it is appropriately separated from the other modules to prevent a complete system failure.

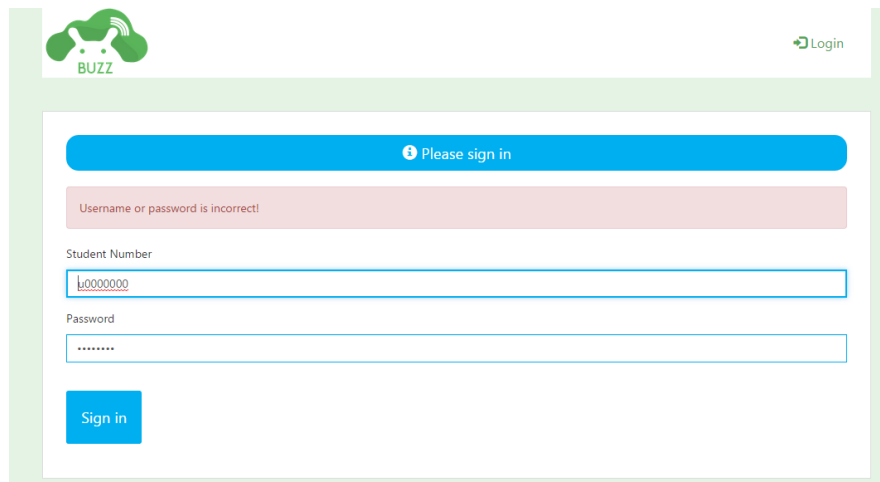


Figure 25: Login error

7. Integratibility:

The system uses separate npm packages for each module which make it easy to share, update and reuse the code and hence integrate-able.

8. Deployability:

The System is Deployable on both Linux and windows servers as it was ran and worked fine in both cases, the following screen shot shows the system running on a windows server.

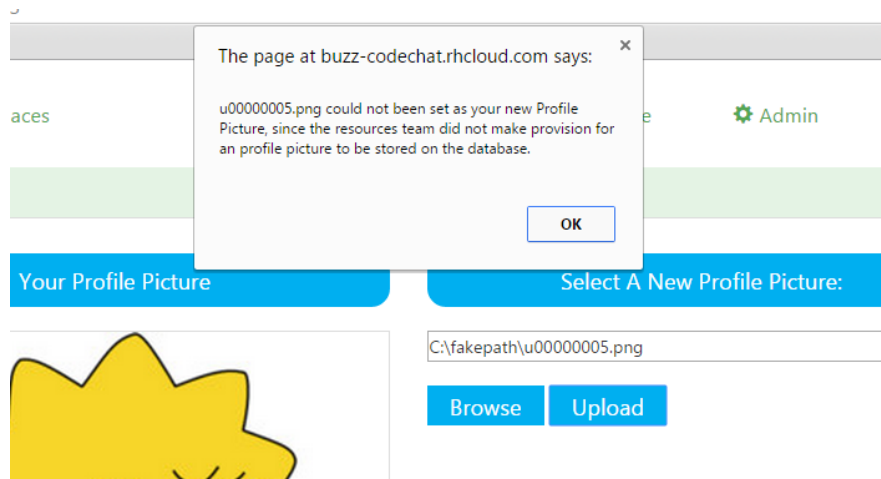


Figure 26: Error when uploading new profile picture

3 Conclusion

The Buzz systems had been tested, evaluated and compared to the original specification that defined how the system should be implemented. The evaluation of both implementations have brought up several interesting facets. It has been noted that both systems function to an extent, but none of the two implementations fulfilled the requirements completely. The systems implemented showed some promise in terms of some modules when viewed in isolation, while some modules did not function completely. Some modules failed to satisfy testing requirements such as unit tests, while others excelled at their unit tests. The integration of several related modules led to great success while others failed. At the top level some implementations delivered a well-functioning system, but lacked in terms of security and completeness. The other implementation led to a well functioning interfaced system, but lacked in terms of functionality. The testing of both systems led to the conclusion that the systems both worked to some extent, but failed to satisfy the complete requirements of the original specification of the system.