MUNSN

Master Test Plan

Document Version: 1.0

Date: March 19th 2017
Prepared by: Andrew Way

Introduction

Purpose

This document outlines the testing strategy for the MUN Social Network website. Its objective is to communicate project-wide quality standards and procedures. On a high level, the product will be tested in its ability to:

- Store a modifiable database of users, groups, and user-user and user-group relationships
- Store a modifiable database of content including:
    - Resumes
    - Pictures
    - Polls
    - Posts
    - Comments
    - Chat histories
- Modify the database of users and content
- Display a interactable user interface

Test Items

In this test plan, all modules will be independently black box tested. The interface of each module will be tested, and tests requiring the interaction of more than one module will not be included as a part of this test plan. For more information on the project, please refer to the Software Specification Document.

Features to Be Tested

1. FR001: Account Registration
2. FR002: Login
3. FR003: Profile and Group Content
4. FR004: Group Management
5. FR005: Friends
6. FR006: Guests
7. FR007: Posts
8. FR008: Suggested Friends
9. FR009: Lost & Found
10. FR010: Schedule/Calendar

11. FR011: Polls
12. FR012: Resumes
13. FR013: Instant Messenger

Note: The features will be tested as independent sub-features. Full feature testing will be done after module integration.

Project Overview

MUNSN is a social networking web application designed for students and faculty at Memorial University of Newfoundland. Users can register and communicate with other users on topics regarding course work, participate in online study groups and polls, and share professional information such as a resume and semester calendar.

Test Strategy

Objectives

The objective of our test plan is to ensure expected output and processing when the website is used as indicated in each of the use cases outlined in the Software Specifications Document. The outcome of the test is to create a fully-tested product that is able to provide all features in the Software Specifications Document and, as a by-product, provide testing scripts that can be used in post-release testing. The objective of testing in the 3rd milestone will be to ensure each module is capable of satisfying each subfeature it is designed to implement. Tests have been designed to determine whether each module satisfies this.

Test Assumptions

Key Assumptions

- Data files containing test information are available which includes:
    - User data
    - Content data

Functional Testing

- During functional testing, tests will be performed using preloaded data which is available on the system at the time of execution.

Test Principles

- Testing processes will have well defined inputs, expected outputs, and pass/fail criteria.
- Tests will test each of the functional and nonfunctional requirements outlined in the Software Specifications document
- Reusable test scripts will be created and made available for testing in future iterations
- In most tests, the tester will simply input data to the interface of each module and observe its output.

Data Approach

- Pre-loaded test data will be made available to be reusable by the test scripts at any point in time

Scope and Levels of Testing

Exploratory Testing
Purpose: Ensure that critical defects in the user-interface and user and content servers are fixed
Scope: Database modules
Testers: Back-end Development Team
Method: Informal testing without any test scripts and documentation
Timing: Parallel with independent module development

Structural Testing
Purpose: Ensure that modules integrated correctly and the entire product is functional as a whole
Scope: Across all modules
Testers: Backend and frontend development team
Method: Test scripts
Timing: After the development of all modules and parallel with module integration

Functional Testing
Purpose: Verify that each function's output matches the expected output for particular input
Scope: User interface (web page modules)
Testers: Front-end development Team
Method: Tests will be performed according to manual input
Timing: After integration of the web pages with the back-end modules
Test Acceptance Criteria:

1. Use case documents are made available (See the SRS)

2. Test cases approved and signed-off prior to start of test execution
3. Development is completed, unit tested with pass status

User Acceptance Test
Purpose: Final test by end users to ensure product fulfills project requirements
Tests: End users
Method: Black box testing
Testing: After all other levels of testing

Item Pass/Fail Criteria

A test is passed if actual output matches expected output of each step. Otherwise, it fails.

Process Overview

1. Identify the features to be tested by reviewing the features in the software specification document
2. Identify the features that correspond to each module
3. Identify the tests required to test each feature in each module
4. Create test data that is adequate to verify proper operation for each test
5. Identify the expected results for each test
6. Document the test case configuration, test data, and expected results
7. Perform the tests
8. Record all observations in the test plans in this document
9. Advise the development team if expected results do not match actual results
10. Redo the tests once bugs fixed

Tests

Account Validation

AV-T001: Validate Account with Expired Link

Description: Confirm that expired links are renewed, and Confirmation Email server is called
Assumptions: Unauthenticated user exists in user table

Test data: User ID

Step	Description	Expected Result
1	Call aks.validate with expired auth key	Aks.validate called
2	Aks calls DB.Auth.find with auth key	Key is found
3	Check if key is expired	Key is found to be expired
4	Create new auth object	Auth object created
5	Call ems.resendAuthEmail with new auth object	Function called
6	Call DB.Auth.update with new auth object	Function called

Comments: None

AV-T002: Validate Account With Invalid Link

Description: Attempt to authenticate an account with a auth key that does not exist in the table
Module to test: Account Validation
Assumptions:
Test data:

Step	Description	Expected Result
1	Call aks.validate with invalid auth key	Aks.validate called
2	Aks calls DB.Auth.find with auth key	Key is not found
3	Callback with error message	Error message displayed
4	No other function calls made	No other function calls made

Comments: None

AV-T003: Validate Account With Valid Link

Description: Make request containing a valid authentication link and call user management to update the authentication status of a user
Module to test: Account Validation

Assumptions:
Test data:

Step	Description	Expected Result
1	Call aks.validate with valid auth key	Aks.validate called
2	Aks calls DB.Auth.find with auth key	Key is found
3	Check if key expired	Key is found to not be expired
4	Call DB.Auth.remove to delete auth object	Auth object from auth collection deleted
5	DB.Auth.remove calls collectionUSers.update to set auth status for user to true	Auth status for user corresponding to UID set to true in user collection

Comments: None

FST001: Account Registration

Module to test: User Management
Assumptions: NodeJS and MongoDB is presently enabled
Test data: {fName,lName,gender,dob,email,pass,address}

Step	Description	Expected Result
1	User accesses login page	Log in page delivered
2	User clicks “Register” button	Login panel changes to register panel
3	User enters personal information into text fields	All fields modifiable
4	User types in correct MUN email	Email is recognized as valid
5	User enters password	Password field modified
6	User clicks “Submit” button	Information is delivered to backend and stored in new user row in user table
6	System sends confirmation email to confirm successful creation	User receives email with authkey
7	User activates link from email	Valid key

8	User updated to authorized	Database row updated

Comments: If key is invalid then a new one is sent and you repeat from 6

User Management

UM-T001: Guest Login

Step	Description	Expected Result
1	HTTP Request for content with visibility set to “Friends”	Content is not accessible on serverside and is not given to frontend

Comments: Should not be able to view own profile or join groups.

UM-T002: Find User By Valid ID

Description: Take a user ID and search for the corresponding user entry in the users collection. Return the user object
Assumptions:
Test data: User ID

Step	Description	Expected Result
1	Get request to findUserById	Routed to DB.Users.findByID function
2	Extract user ID from request in DBUsers.findById	User ID extracted
3	Obtain user entry in collectionUsers by passing user id	User entry found
4	Return the user object corresponding to the user ID	User JSON object returned and displayed in terminal

Comments: None

UM-T003: Delete User

Description: Take a user ID and delete the corresponding user entry in the users collection.
Assumptions: User ID exists in the database
Test data: Valid User ID

Step	Description	Expected Result
1	HTTP Request to deleteUser	Routed to DB.Users.remove
2	Extract user ID from request in DB.Users.remove	User ID exists and is extracted
3	Call collectionUsers.remove	Function called
4	collectionUsers.remove deletes entry corresponding to user ID in user collection	Entry is deleted

Comments: None

UM-T004: Register User

Description: Take in a user JSON object and create a new unauthenticated user entry in the users collection.
Assumptions: Test data satisfies all input fields
Test data: User data

Step	Description	Expected Result
1	HTTP Request to registerUser	Routed to DB.Users.add
2	Extract user properties from JSON object	User properties extracted and stored in properties of “row” object
3	Call collectionUsers.insert	User entry added to user collection
4	Call DBAuth.add to initiate authentication process	Function is called and given row object

Comments: None

UM-T005: Update User

Description: Extract JSON object from request and overwrite user properties in user collection
Assumptions: User data satisfies all database fields
Test data: User properties

Step	Description	Expected Result
1	HTTP Request to updateUser	Route to DB.Users.update
2	Extract properties from JSON object	User properties extracted and stored in properties of “update” object
3	Call collectionUsers.update	Function called and given the updates object
4	Update user entry	User entry in user collection is updated with correct properties

Comments: None

Group Management

GM-T001: Group Creation

Module to test: Group Member Management
Assumptions: Group does not exist
Test data: {Userid, GroupName}

Step	Description	Expected Result
1	Request on a ‘/api/group/create’	Request received by API
2	DBGroups reads request data	A Row is prepared to be inserted
3	DBGroups Inserts row	Row is inserted and present in database
4	Response to html request	JSON result to html

Comments: None

GM-T002: Group Modification

Module to test: Group Member Management
Assumptions: Group exists
Test data: {Groupid, name, descrip}

Step	Description	Expected Result
1	Request on a '/api/group/update'	Request received by API
2	DBGroups reads request data	A Row is prepared to be updated
3	DBGroups Updates row	Row is updated and present in database
4	Response to html request	JSON result to html

Comments: None

GM-T003: Group Deletion

Module to test: Group Member Management
Assumptions: Group exists
Test data: {Groupid}

Step	Description	Expected Result
1	Request on a '/api/group/remove/'	Request received by API
2	DBGroups reads request data	A Row is prepared to be deleted
3	DBGroups Deletes row	Row is deleted and no long present in database
4	Response to html request	JSON result to html

Comments: None

Group Member Management

GMM-T001: Manipulate Table

Module to test: Group Member Management
Assumptions: Group Exists
Test data: {GroupName, UserID, GroupID }

Step	Description	Expected Result
1	Request on a relevant router path	Request received by API
2	DBGroupMembers.remove call using api/group/remove/user	Specified member is removed from group
3	DBGroupMembers.add call using api/group/add/user/	Specified user is added to group
4	DBGroupMembers.find call using api/group/users/:gid	Returns an array of all users in group

Comments: Failure will not break the system

Group Requests

GR-T001: Add New Group Request

Description: Add a new recieved Group join request to collection for recipient group and new join request to collection for sender user
Assumption: Group exists in table
Test data:

Step	Description	Expected Result
1	HTTP request to add join group	Routed to DB.GRoups.addRequest
2	Extract User ID and group ID from request body	IDs extracted

3	Find objects corresponding to user ID and group ID	Objects found in user and group collection
4	Insert new join request entry into group request collection	Entry inserted

Comments: None

GR-T002: Delete Group Request

Description: Find and return a group request entry in the group request collection corresponding to user ID
Assumption: Group request exists in collection
Test data: Group request ID

Step	Description	Expected Result
1	HTTP request to delete group request	Routed to DB.Group.removeRequest
2	Extract User ID and group ID from request body	IDs extracted
3	Call collectionGroupRequests.remove	Group object removed

Comments: None

GR-T003: Find Group Request

Description: Find and return Group request object from Group request collection
Assumptions: Group request exists in collection
Test data: Group ID

Step	Description	Expected Result
1	HTTP request to find group request	Routed to DB.Groups.findRequests
2	Extract User ID and group ID from request body	IDs extracted
3	Find objects corresponding to user ID and group ID	Objects found in user and group collections
4	Return group request object	Object returned

Comments: None

Friendship Management

FM-T001: Friendship Creation

Module to test: Friend Management
Assumptions: Friendship not exists
Test data: {userid, friendid}

Step	Description	Expected Result
1	Request on a '/api/user/add/request/'	Request received by API
2	DBFriends reads request data	Rows is prepared to be inserted
3	DBFriends Insert row	Row is inserted for both user and friend and present in database
4	Response to html request	JSON result to html
5	Server waits for user action forever	Server Awaiting and functioning normally
6	User accepts request	Row is finalized
7	Response to html request	JSON result to html

Comments: None

FM-T002: Friendship Deletion

Module to test: Friend Management
Assumptions: Friendship exists
Test data: {userid, friendid}

Step	Description	Expected Result
1	Request on a '/api/user/remove/friend/'	Request received by API
2	DBFriends reads request data	Rows is prepared to be deleted
3	DBFriends Delete row	Row is updated for both user and friend and

		present in database
4	Response to html request	JSON result to html

Comments: None

FM-T003: Suggested Friends List Creation

Module to test: Friend Management
Assumptions: Friendship exists
Test data: {userid}

Step	Description	Expected Result
1	Request on a '/api/user/friends/suggest/:uid'	Request received by API
2	DBFriends request is made	Returns array of suggested friends
4	Response to html request	JSON result to html

Comments: Failure will not break system. Should not suggest current friends.

Friend Requests

FR-T001: Add New Friend Request

Description: Add a new recieved friend request to collection for recipient user and new sent friend request to collection for sender user
Assumption: Sender and recipient users exist in database
Test data: Sender and recipient user friend requests

Step	Description	Expected Result
1	HTTP request to add new friend request	Routed to DB.Friends.addRequest
2	Extract User ID and friend ID from request body	IDs extracted
3	Find user objects corresponding to user ID and friend ID	Objects found in user collection

4	Insert new friend request entry into friend request collection	Entry inserted

Comments: None

FR-T002: Delete Friend Request

Description: Find and return a friend request entry in the friend request collection corresponding to user ID
Assumption: Friend request exists in the database
Test data: Sender and recipient user IDs

Step	Description	Expected Result
1	HTTP request to delete friend request	Routed to DB.Friends.removeRequest
2	Extract User ID and friend ID from request body	IDs extracted
3	Call collectionFriendRequests.remove	Friendship object removed

Comments: None

FR-T003: Find Friend Request

Description: Find and return friend request object from friend request collection
Assumption: Friend Request exists in database
Test data: Friend request ID

Step	Description	Expected Result
1	HTTP request to find friend request	Routed to DB.Friends.findRequests
2	Extract User ID and friend ID from request body	IDs extracted
3	Find user objects corresponding to user ID and friend ID	Objects found in user collection
4	Return friendship request object	Object returned

Comments: None

Comments

C-T001: Create Comment

Description: Create new comment, store into the comment database and associate with the comment a parent post id.
Test data: Comment text information and other property data

Step	Description	Expected Result
1	HTTP Request to create new comment	Route to DB.Comments.findByPostId
2	Extract userID from request body	User ID extracted
3	Search collectionComments using postID as key	Comments associated with a post are found
4	Add new comment with parent ID as postID	New comment is added to comment collection and has reference to parent ID

Comments: None

C-T002: Find Comments

Description: Search the comment collection using a post ID and return the comments associated with that post.
Assumption: Comment exists in database
Test data: Post ID

Step	Description	Expected Result
1	HTTP Request to find comments	Route to DB.Comments.findByPostId
2	Extract userID from request body	User ID extracted
3	Search collectionComments using postID as key	Comments associated with a post are found
4	Respond with JSON object of	JSON object constructed and transferred

	comments	

Comments: None

C-T003: Delete Comment

Description: Find a comment in the post collection and delete it.
Assumption: Comment exists in database
Test Data: Comment ID and post ID

Step	Description	Expected Result
1	HTTP Request to find comment	Route to DB.Comments.removeByID
2	Extract postID and commentID from request body	IDs extracted and saved in variables
3	Search collectionComments using postID and commentID	Comment associated with a post are found
4	Delete the comment	Comment is removed from comment collection

Comments: None

C-T004: Update Comment

Description: Create new comment, store into the comment database and associate with the comment a parent post id.
Assumption: Comment exists in collection and comment information satisfies comment properties
Test Data: New comment information, post ID, and comment ID

Step	Description	Expected Result
1	HTTP Request to update comment	Route to DB.Comments.update
2	Extract comment text from request	Text extracted
3	Call commentCollection to add new comment to comment history associated with comment id	Comments associated with a post are found

Comments: None

Posts

P-T001: Post Creation

Module to test: Posts
Assumptions: None
Test data: {authorid, origin, dataType, data}

Step	Description	Expected Result
1	Request on a '/api/post/add/timeline/' OR '/api/post/add/group/'	Request received by API
2	DBPosts reads request data	Rows is prepared to be inserted
3	DBPosts Inserts row	Row is Inserted and present in the database
4	Response to html request	JSON result to html

Comments: None

P-T002: Post Modification

Module to test: Posts
Assumptions: None
Test data: {postID, postData, postUpdates}

Step	Description	Expected Result
1	Request on a '/api/post/update/'	Request received by API
2	DBPosts reads request data	Rows is prepared to be updated
3	DBPosts Updates row	Row is updated and present in the database
4	Response to html request	JSON result to html

Comments: None

P-T003: Post Deletion

Module to test: Posts
Assumptions: Post Exists
Test data: {postID, postData}

Step	Description	Expected Result
1	Request on a '/api/post/remove/'	Request received by API
2	DBPosts reads request data	Rows is prepared to be deleted
3	DBPosts Deletes row	Row is deleted and not present in the database
4	Response to html request	JSON result to html

Comments: None

P-T004: Lost & Found Item Post

Module to test: Lost & Found
Assumptions: None
Test data: {}

Step	Description	Expected Result
1	Request on "/api/lost/add"	New lost and found post object stored in collection with reference to picture, coordinates, and text info

Comments: None

Schedule

S-T001: Schedule

Module to test: Schedules
Assumptions: User has google account and has calendar
Test data: Google account credentials

Step	Description	Expected Result

1	User signs into google account	User signed in
2	Request to google server for calendar information	Request sent
3	Update calendar with google calendar contents	Calendar updated

Comments: None

Instant Messenger

IM-T001: Instant Messenger

Description: Facilitate instant messaging between two or more users
Assumptions: User Exists
Test data:

Step	Description	Expected Result
1	User enters chat	Chat information displayed
2	User enters text information into text field	Text is displayed in the text field
3	User clicks submit	Chat object is submitted to server
4	Server stores chat message in chat log	Message stored in chat log
5	Refresh user's chat windows	Users chat windows refreshed

Comments: None

Confirmation Email Server

CMS-T001: Emailing

Description: Test creating an email body based on an auth object and send to email address

Assumptions:
Test data: General Text data for email body

Step	Description	Expected Result
1	Store ems.js reference	Reference stored
2	Call EMS.sendAuthEmail	Auth object received
3	Create email body and link	Email body and link made
4	sendEmail(email,callback)	Email sent to recipient

Comments: None

File Transfer

FT-T001: Image Upload

Module to test: File Transfer
Assumptions: NodeJS is presently enabled
Test data: Image File {*.png; *.jpg}

Step	Description	Expected Result
1	Request on a relevant router path	User has file to upload
2	NodeJS receives file	File is downloading or downloaded
3	File is saved	File exists on the path

Comments: Existing files are overwritten automatically.

FT-T002: Find Files

Module to test: File Transfer
Assumptions: Folder Structure exists
Test data: Filter String, File Path

Step	Description	Expected Result
1	Path exists to file	Path is found
2	Read files in Path	Files are accumulated
3	Next File	Info for file is retrieved
4	File matches filter	File is returned

Pass/Fail:
Comments: Repeat from 3 until there are no more files

FT-T003: Download File

Module to test: File Transfer
Assumptions: File Exists
Test data: File Path

Step	Description	Expected Result
1	Request on a router path	Request received by API
2	utils.findFiles call	First result is the path on the filesystem
3	Call to utils.download	Response containing requested file
4	Download file	File is downloaded on client side

Comments: Failure does not break the system

FT-T004: Content Update

Module to test: File Transfer
Assumptions: Content is present to be updated
Test data: Some various data

Step	Description	Expected Result

1	Request on a relevant router path	Request received by API
2	utils.findFiles call	First result is the path on the filesystem
3	Call to utils.download	Response containing requested file
4	Download file	File is downloaded on client side

Comments: All content updates just replace existing files outright

FT-T005: Content Deletion

Module to test: File Transfer
Assumptions: File Exists
Test data: File Path

Step	Description	Expected Result
1	Request on a relevant router path	Request received by API
2	utils.findFiles call	First result is the path on the filesystem
3	Call to utils.delete	Server executes deletion
4	Deleted file	File is deleted on server side

Comments: Failure does not break the system

FT-T006: Resume Upload

Module to test: File Transfer
Assumptions: User Exists
Test data: Resume PDF file

Step	Description	Expected Result
1	Request POST on a ‘/content/resume/user/’	Request received by Content Router
2	Router reads request data	File is prepared to be received
3	Server receives file data	File is saved to server

4	Response to html request	JSON result to html

Comments: None

FT-T007: Resume Deletion

Module to test: File Transfer
Assumptions: User Exists and Resume exists
Test data: None

Step	Description	Expected Result
1	Request DELETE on a ‘/content/resume/user/’	Request received by Content Router
2	Router reads request data	File is prepared to be deleted
3	Server deletes file data	File is deleted from server
4	Response to html request	JSON result to html

Comments: None