Test Plan Document

for

The College of Education Application Station

Version 1.0

Prepared by Molly Burns, Alec Carlyle, Mary Clark, and Hannah Zontine

UMW Computer Science Department

March 31, 2017

Table of Contents

1. Introduction

1.1. Purpose

The objective of this document is to explain the our test plan for the UMW College of Education Application Station project. This document will discuss an overview of the project itself, our test plan and testing strategy. This document explains our test plan for our client, Dr. Venita McCall, our Software Engineering professor, Dr. Karen Anewalt, and our peers who will be performing testing of the UMW College of Education Application Station.

1.2. Statement of scope

This project is a web-based application which will allow our client to manage three different applications for the UMW College of Education: the Initial Licensure Application, the application for Graduation Continuation Admission, and the application for Undergraduate Admission. The Application Station will provide a digital application management system to replace the current physical application process, which will be hosted on Domain of One's Own, which is provided by the University of Mary Washington. Any student with a UMW account will be able to register for the system and fill out any necessary application for admission into the UMW College of Education.

1.3. Overview of the remainder of the document

This document is organized into sections, which are listed on the Table of Contents (located on page 2) for reference. Each section is signified by bold text and a section number throughout the document. Additionally, each section and subsection contains specific information pertaining to the section heading.

2. System Overview

2.1. Project Description

The College of Education Application Station is a website designed to digitize the student-teaching application process at UMW. Upon initially loading the webpage, a user is able to either login to their CoEAS account or register for a new one. The application will support two classes of users: student users and admins. Both types of users have a distinct dashboard upon which they may interact with the system. Any UMW-enrolled student shall be able to register for a student user account in order to fill out 3 distinct forms. These forms include:

- Initial Licensure Internship Application
- Undergraduate 5-Year Program Application
- CoE Continuing into the Graduate Program Application

Once submitted, the above forms shall be stored inside of the database along with all of the accounts in the system. Users shall also be able to edit their submitted forms to make corrections, given that the form is still available to edit and/or submit. The admin of the system is able to set/extend the deadline for application submissions,

Faculty of the education department shall interact with the system as an admin. They shall be able to view application submissions and filter them by user name, type of application, and email address. They'll be able to sort a list of users in the database and delete inactive users. When the end of the semester comes around, the admin shall be able to archive the data in the database so that they may store it and then clear the database. They also shall be able to export application submissions into an Excel-ready .csv file format.

2.2. Requirements List

Requirements are the necessary functions and features of the CoE web application. The functional requirements highlight how the system shall behave when a user interacts with it and shall be explained using user stories. Non-functional requirements are all requirements outside of the behavior of the system.

2.2.1. Functional Requirements

2.2.1.1. Administrator

A. As an admin, I want to be able to set and/or extend the deadline for application submissions so that students may be able to either turn in or edit their applications at a date that I see appropriate.
B. As an admin, I want to sort submitted submissions by email, name, or type of form so that I may find the application that I need.
C. As an admin, I want to delete a user account or application submissions so that the database does not become too cluttered.
D. As an admin, I want to archive the database at the end of the semester to store the data before I clear the database.
E. As an admin, I want to export a series of submissions into a .csv file format so that I may view and manipulate those forms in Excel.
F. As an admin, I want to be able to login to my account so that I may perform actions unique to my account privileges.
G. As an admin, I want to view any submitted form's contents on the website so that I may see if a student is good to go on their submissions.
H. As an admin, I want to reset my password so that I can login to the website in case I forget it.
I. As an admin, I want to clear the database at the end of a semester so that I have a fresh database to work with for upcoming students.

2.2.1.2. Student User

A. As a student, I want to be able to fill out up to 3 distinct applications (forms) so that my position as a student teacher at UMW may be accepted or maintained.
B. As a student, I want to be able to register for a CoE application account so that I may be able to recover any progress on either submitted or unsubmitted forms during the entirety of my participation in the student teaching program.
C. As a student, I want to verify my account so that I can confirm my identity using my email address.
D. As a student, I want to reset my password so that I can login to the website in case I forget it.
E. As a student I want to be able to login so that I can perform actions unique to my account.

2.2.2. Non-Functional Requirements

A. The application should avoid free user input in every case where possible, and instead offer choices selected by the College of Education.
B. The application should secure student information and encrypt sensitive student

data, such as passwords and criminal records (unencrypt in admin export).

C. The application can be accessed publicly over a network.

D. The application should be hosted and on Domain of One's Own with no compatibility issues.

3. Test Plan

3.1. Testing strategy

The objective of testing is to verify that the functionality of the CoE Application Station works according to the specifications. The testing strategy mainly consists of the techniques to be used and the criteria for knowing when a test is complete. There are five items that must be tested: installation, logging in, creating an account, usability, and filling out a form.

3.2. Testing resources and staffing

The staffing for systems tests is split between two teams: the testing team and the integration team. The testing team is wholly responsible for the testing of the system and the results for each of the 5 tests. The integration team is responsible for assembling and analyzing the results of the testing team as well as for having a completed, runnable piece of code that can be tested in line with the test schedule.

3.2.1. Testing Team

In order to test the CoEAS, the testing team will need to clone the public InformationStation git repository (5.3.1) onto Cloud9. Each team member will need to have a number of technologies installed in order to successfully set up the website and complete each of the tests of the system. Thankfully for the testing team, all of the installations are automated in a file that they can source to run. In order to get themselves set up for testing the CoEAS, each testing team member needs to:

1. Create a new workspace on Cloud9
2. git clone the InformationStation repository (see 5.3.1)
3. In the InformationStation directory, source the installation file (. setupTest.sh)
4. In the InformationStation directory, source the run server file (. runServer.sh)
5. Click on the http link to the server in the Cloud9 terminal to view the website

The testing team will need the following technologies:

- New Cloud9 Workspace (see 5.3.2)

3.2.2. Integration Team

The integration team will be largely responsible for making sure that all of the test results are gathered. Any feedback collected from the result of this process will be used to improve usability of the software and help to squash any bugs that the Integration team may not have considered prior to conducting these tests. If any areas of the tests are unfinished by the time of testing, then the integration team will deploy all of its resources in order to make sure testing will occur as scheduled.

3.3. Test work products

Errors reported by the testing team will be fixed by the implementation team. The implementation team will prioritize the impact of the errors reported and adjust the tentative schedule accordingly.

3.4. Test record keeping

Test Type	Results
Installation	● Can you set up the application? ____________ ● Can you run the application? ____________ ● Can you access the installation instructions? ____________ ● Are the dependencies listed correct? ____________
User Interface/ Usability	● Is there mobile compatibility? ____________ ● Are there any spelling errors? ____________ ● Is there any part that needs more clarification? ____________
Create Account	● Can you create an account? ____________ ● Can you create an account with a username that already has an account? ____________ ● Can you create an account by inputting two different passwords in? ____________ ● Can you create an account with an invalid email? ____________
Log In	● Can you log in with the correct username and passwords? ____________ ● Can you log in with a username that does not exist in the system? ____________ ● Can you log in with a username that exists in the system and an incorrect password? ____________
Forms	● Can you submit a form? ____________ ● Can you submit a form with missing information? ____________

3.5. Test schedule

3.5.1. Testing team tasks

The testing team should begin by installing their own local copy of the application to ensure that the installation instructions are clear. Next, they will check the user interface and forms, which will allot the implementation team time to fully implement more subsystems for testing. Finally, available input/output testing in the login and forms will come last.

3.5.2. Implementation team tasks

During the testing team's tasks, the implementation team should be available to answer any questions and receive feedback about the application. In the meantime, they will also finish implementing unfinished areas of the system that need to be tested.

3.5.3. Schedule overview

Test Type	Date	Expected Time	Goals
Installation Testing	4/3	1 hour	**Testing Team** ● Set up and run the application in the environment of the testing group's choice for further testing. ● Check that installation instructions in README.md can be understood and followed. ● Check that dependencies listed in requirements.txt are correct.
User Interface and Usability	4/4	30 mins	**Testing Team** ● Check the available pages on several different devices and layouts for mobile compatibility. ● Check animation and layout on form. ● Look for spelling errors or areas that need clarity. **Integration Team** ● Database running. ● Complete user sign up and login.
Create Account	4/5	1 hour	**Testing Team** ● Make an account. ● Try to sign up with a bad passwords, incorrect account info. **Integration Team** ● Implement usability suggestions that both teams agree on.
Log In	4/6	30 mins	**Testing Team** ● Log into the application with an account created during sign up.
Forms	4/7	1 hour	**Testing Team** ● Make submissions using the available form. ● Make at least one submission through the UI that the system accepts. ● Otherwise, enter incorrect information, submit broken information, etc. to see if it is accepted. **Integration Team** ● Begin fixing sessions/login bugs that were discovered.

4. Test Procedure

4.1. Form Tests

The application’s forms are one of the highest priority items: they control what data can be submitted by the user, and this input is further verified on the backend. Several items are enclosed in dropdown or radio options to prevent the user from providing the extraneous data that they tend to provide on paper forms. These tests will ensure that no invalid, unnecessary, or unwanted data can be submitted to the database. The test case data can either be submitted through the form or by sending a POST request with JSON-formatted data outside of the application.

Note: an array indicates dropdown selections, where 0 is the first available index in the dropdown and sequential elements in the array are indexes in sequential dropdowns.

Test	Requirement	Purpose	Test Case Data	Expected Results

4.1.1.	2.2.1.2.A.	Send a POST request with valid information for the continuation form.	`{"endorsementArea":[[1,3,0],[2,0]],`
			` "testRequirements" : [`
			`{"exam" : "praxis",`
			` "date" : "04222017"},`
			`{"exam" : "VCLA",`
			` "date" : "05032017"}],`
			` "graduation" : "May 2018",`
			` "continue"   : false,`
			` "reason"     : "Financial"`
			`}`
				Response Code 200 (Success)

4.1.2.	2.2.2.A.	Verify that a non-existent array index is not accepted (in the UI, the test data can be met using your browser’s developer tools)	`{"endorsementArea":[[9]],`
			` "testRequirements" : [`
			`{"exam" : "praxis",`
			` "date" : "04222017"}]`
			` "graduation" : "May 2018",`
			` "continue"   : false,`
			` "reason"     : "Financial"`
			`}`
				Response Code 400 (Bad Request)

4.1.3.	2.2.2.A.	Verify that non-given inputs are not accepted for tests (in the UI, the test data can be met using your browser’s developer tools)	`{"endorsementArea":[[1,3,0]],`
			` "testRequirements" : [`
			`{"exam" : "i hate tests",`
			` "date" : "04222017"}]`
			` "graduation" : "May 2018",`
			` "continue"   : false,`
			` "reason"     : "Financial"`
			`}`
				Response Code 400 (Bad Request)

4.1.4.	2.2.2.A.	Verify that empty data cannot be submitted.	`{"endorsementArea":[],`
			` "testRequirements" : []`
			` "graduation" : "",`
			` "continue"   : "",`
			` "reason"     : ""`
			`}`
				Response Code 400 (Bad Request)

4.2. Sign up Tests

User accounts should have distinct identification containing student information. These tests will validate that no students can have identical information in their accounts

Test	Requirement	Purpose	Test Case Data	Expected Results

4.2.1.	2.2.1.2.B.	Send a POST request with valid information to sign up.	`{"firstName" : "[tester’s f name]", "lastName" : "[tester’s l name]", "email" : "[tester’s email]", "bannerid" : "[tester’s id]", "pass" : "[password]", "pass_conf" : "[password]" }`	Response Code 200 (Success)
4.2.2.	2.2.1.2.B.	Verify that users are distinct.	`[Same information from above]`	Response Code 400 (Bad Request)
4.2.3.	2.2.1.2.B.	Verify that empty data cannot be submitted. (May also attempt to submit full requests with one field empty)	`{"firstName" : "", "lastName" : "", "email" : "", "bannerid" : "", "pass" : "", "pass_conf" : "" }`	Response Code 400 (Bad Request)

5. Appendices

5.1. Dictionary of terms

- Archive data: move data that is no longer needed/used to a separate means of storage.
- Browser Developer Tools: Tools available to anyone in browser for testing purposes, usually accessible with a unique keystroke
- Cloud9 Workspace: Development area for testing and integration teams (see 5.3.2)
- CoE: College of Education
- CoEAS: College of Education Application Station
- Dashboard: The visual set of tools available either to the student user or the admin. Each type of user has a dashboard unique to their usages of the system.
- Database: An ordered, stored arrangement of data inside of the web application. Holds onto user accounts and submitted form information.
- DoOO: Domain of One’s Own
- Form: One of 3 applications that student users will be able to submit into the application. For the purposes of this test period, only one of those forms is available
- Functional Requirements: The functions that a system shall perform
- Integration team: Responsible for having the application ready for testing according to the test plan schedule
- JSON-format: JavaScript Object Notation, a data format that allows for the exchange of information
- Non-Functional Requirements: Requirements outside of the behavior of the system
- POST request: This request in our project will allow for submissions of data to the server for storage from a user registering, logging in, submitting a form…
- Source a file: When the file script.sh is ‘sourced’ at the command line like so: . script.sh   a series of commands in the file are run at command line. This includes the installation of needed technologies and running the server
- Testing team: Responsible for completing every test in the test document and submitting the results to the Integration team
- UI: User Interface
- Usability: the ease of use of an aspect of the User Interface or other aspect of the system

5.2. Individual Contributions

This section details the individual contributions that each group member made to this Test Plan Document.

Introduction

- Purpose - Molly
- Scope - Molly
- Overview of Remainder of Document - Molly

System Overview

- Project Description - Copied from Project Plan Document (Alec)
- Requirements List - Copied from Project Plan Document (Alec)
- Non-Functional Requirements - Mary

Test Plan

- Testing Strategy - Hannah
- Testing Resources and Staffing - Alec
- Test Work Products - Hannah
- Test Record Keeping - Hannah
- Test Schedule - Mary

Test Procedure

- Form Tests - Mary
- Sign up Tests - Mary

Appendices

- Dictionary of Terms - Alec
- Individual Contributions - Molly
- Additional Information - Mary, Alec

5.3. Additional Information

5.3.1. Project Repository

The project is currently located at a public central repository on Github. The repository should be cloned from this location, and the testing team should routinely pull any changes from the master branch (which will only contain stable builds) to ensure they have the latest work from the integration team. A link to the repository is below:
https://github.com/rhodochrosiite/InformationStation

5.3.2. Cloud9 Workspace

Members of the testing team will need to create a new Cloud9 workspace for this project. Since the integration team is developing the website on Cloud9, it follows that the testing team will use the same development environment in order to reduce potential issues with compiling and running the application. A link to Cloud9 is here:
https://c9.io/