Software Test Plan

Version 1.0

4/9/2018

Project

Tailored Tutoring Business Portal

Robert Lokken

Mentor

Ana Paula Chaves Steinmacher

Team

Alex Kahn

Jesus Garcia

Tyler Mitchell

Taylor Walker

Table of Contents

Introduction

For our client, Tailored Tutoring Co., we have a designed an online portal where both students and tutors can be in communication to receive and give help on questions and homework assignments. In order to accomplish this, student users are able to submit images of homework assignments, via our portal, and these images are then accessible via tutor users who provide a video solution walking the student through the problem-solution. In essence, our application facilitates students being able to ask questions, and tutors being able to answer these questions, all through the use of our online web-application.

The goal of this document is to outline our software testing plan of this online web application that we created. The goal here, and with software testing in general, is to test that the created final product aligns with the original intentions and plans for the software solution that was outlined in our Requirements Document. The client (Mr. Lokken, CEO of Tailored Tutoring Co.), and programmers (Business Web Solutions), set out with an initial plan to create a piece of software that solved a current problem and need for Tailored Tutoring Co. In our initial plans via the Requirements Document, we were to create an online web-application that allowed users to easily be able to upload hw assignments, with such stipulations as having Student Profiles, Tutor Profiles, a system that sent out Notifications, and more requirements that we will test for and get into further detail on later in the document.

We will be focusing on three major areas of testing: Unit Testing, Integration Testing, and Usability Testing. Unit testing will simply focus on testing our functions and code do what we intended them to do, so that there are no crashes or drastic errors when running our code for the web-application. Integration testing will be focused on proving that the different pieces, or technologies, of our web-application are in fact communicating with each other and passing around the correct data. Usability testing might take the most refactoring, as we want to assure that our web-application is “Intuitive” users (note, Intuitive is defined more in the Usability Testing section of this document). Any code refactors based off of our findings will be implemented if necessary.

We chose this testing plan because these really are the three major areas of our web-application that need to be tested and assured work correctly. For Integration Testing, we feel we have mostly proved this, since in order to get our web-application working correctly, we had to make sure data was getting passed from one technology to the other as intended. However, with the Unit Testing will help us to focus on our code and that there are no unintended or unseen side effects. While Usability Testing will focus on the users experience, so that we can make sure they enjoy using our software, since that is in fact who we built if for. All in all, the testing plan outlined in the rest of the document is designed to take a deeper dive and look at the software we created, and hopefully fix any imperfections.

Unit Testing

For unit testing our application, we plan to use the Javascript Unit Testing framework called QUnit, which offers simple setup, support for Node, and support for all browsers that support jQuery. Our plan for unit testing will include a unit test for every unique method we have throughout our Javascript files, with each one having defined inputs based on what is needed.

Some functions that we will not be unit testing is functions similar to our getData() method, which simply calls our database with some user field like an email or uID and then returns the data that is received by the HTTP get request. Since this method only does this call and returns the value, there is no functionality we can test, other than the HTTP request being sent, which we will be covering in our Integration testing.

Functions that we will be testing will be more related to our handleEvent() methods, which usually handle data within our React app, such as changing state or user object values, so we will focus on unit testing these functions. While these will be our main set of methods to test, many of these also rely heavily on database HTTP calls, so we will either stub these requests for the functions that have additional functionality that we can test, and for those functions that only use HTTP requests, we will test these with integration testing rather than unit testing.

For which functions we will be testing, below is a list of those functions, and what values we will be using as boundary conditions:

Admin.js:

- **TC1:** handleDelete (delete a user profile)
    - **Description:** an admin account can delete a user profile from the application
    - **Main Flow:**
        - Admin account enters a user account's email into the delete field, clicks Delete User
        - The user's data will be rendered, with a confirmation button
        - Admin clicks Confirm, and the account is deleted from the database
    - **Expected Outcome:** The account entered will be deleted from the database

    - **Alternative Flow:** Admin enters an email that does not exist
    - **Expected Outcome:** A report should be rendered for the admin to see the account does not exist
- **TC2:** handleGetSchool (add/remove courses for a school)
    - **Description:** an admin account can change the courses related to a school
    - **Main Flow:**
        - Admin enters a school name into the field, clicks Get School Information
        - School name and list of courses is rendered
        - Admin can add/remove courses from the school
    - **Expected Outcome:** The school will add/remove the entered name for the course
- **TC3:** handleAddTutor (account permission change)
    - **Description:** Admin can change account permission between Student and Tutor
    - **Main Flow:**
        - Admin account enters a user account's email into the changePermission field, clicks Change Permission
        - Accounts data is rendered, Admin can grant or revoke Tutor permission, or cancel

    - **Expected Outcome:** The account entered will have an updated permission

    - **Alternative Flow:** Admin enters an email that does not exist
    - **Expected Outcome:** A report should be rendered for the admin to see the account does not exist
- **TC4:** handleAddSchool (add a new school and courses for it)
    - **Description:** Admin can add a new school, and add classes
    - **Main Flow:**
        - Admin account enters a school name to add, clicks Add School
        - A new field is rendered asking for at least one course to add for the school
        - Admin enters a course name and clicks Add School
    - **Expected Outcome:** The school entered will now have the course entered tied to it
- **TC5:** handleDelete (delete a user profile)
    - **Description:** an admin account can delete a user profile from the application
    - **Main Flow:**
        - Admin account enters a user account's email into the delete field, clicks Delete User
        - The user's data will be rendered, with a confirmation button
        - Admin clicks Confirm, and the account is deleted from the database
    - **Expected Outcome:** The account entered will be deleted from the database

    - **Alternative Flow:** Admin enters an email that does not exist
    - **Expected Outcome:** A report should be rendered for the admin to see the account does not exist

Claim.js

- **TC6:** getImageURL
    - **Description:** Method takes an array of image objects, contacts AWS S3 and returns an array of URLs for the images.
    - **Main Flow:**
        - An array of images is passed into the method
        - For each image, a params array of a bucket name, and a key are created

        - The params array is passed to an S3 call, which returns the URL, then appends it to an array for returning
    - **Expected Outcome:** Given an array of images, an array of URLs for those images will be returned

- **TC7:** getDateInformation (convert a timestamp into a string)
    - **Description:** Given a timestamp, method converts it into a string and returns
    - **Main Flow:**
        - A timestamp is passed in, and stored in a Date object
        - Each part of the timestamp is extracted, turned to strings, and concatenated for return
    - **Expected Outcome:** A string representing the timestamp is returned
- **TC8:** handleFileChange
    - **Description:** A new file will be saved in state for the claim
    - **Main Flow:**
        - Given an event, a file reader is created
        - Reader opens the file, and saves it to state
    - **Expected Outcome:** State will contain the newly added file
- **TC9:** submitVideo
    - **Description:**
    - **Main Flow:**
        - Method takes an image, and extracts user and file metadata for it
        - Verifies file extension type, sets up API parameters, and puts the image object into the database
        - Image added to images array in state, and video file added to S3
        - Email is returned to user upon submission success
    - **Expected Outcome:** Image and video files are saved in S3 and Mongo, email sent to user upon success

    - **Alternative Flow:** error returned during S3 submission
    - **Expected Outcome:** A report saying submission failed will be returned to the user
    - **Alternative Flow:** Invalid file type given to the method for video
    - **Expected Outcome:** A report saying invalid file type will be returned to the user

Dashboard.js

TC10: getVideoURL

- **Description:** Method takes an array of image objects, contacts AWS S3 and returns an array of URLs for the videos associated with them.
- **Main Flow:**
  - An array of images is passed into the method
  - For each image, a params array of a bucket name, and a key are created
  - The params array is passed to an S3 call, which returns the URL for the associated video, then appends it to an array for returning
- **Expected Outcome:** Given an array of images, an array of URLs for the associated videos will be returned

- **TC11:** check (returns the value of the school name chosen)
  - **Description:** Returns 'select' if no school name is chosen, or returns the name of the school chosen
  - **Main Flow:**
    - Checks if schoolVal is 'select', returns that if true
    - If false, returns the name of the chosen school
  - **Expected Outcome:** return the name of the school chosen, or 'select' if none

EditProfile.js

- **TC12:** schoolSelect (set state to the selected school name)
  - **Description:** Set the selected school name into state
  - **Main Flow:**
    - Given an event, the state will hold the name of the school, and the list of courses
  - **Expected Outcome:** state will hold the school name and associated courses
- **TC13:** coursesSelect
  - **Description:** selected courses will be stored in state
  - **Main Flow:**
    - Given an event, an array of courses is checked for selection

        - If selected, the courses are pushed to an array
        - Array is saved in state
    - **Expected Outcome:** state will hold an array of course names

- **TC14:** coursesDeselect
    - **Description:** method finds which courses are deselected, updates value array, and saves value to state
    - **Main Flow:**
        - Given an event, a list of courses is created from the event, and checked for deselection
        - Deselected courses are removed from the value array
        - Value array is then saved in state
    - **Expected Outcome:** state will hold the updated list of courses after removing deselected ones

ImageUpload.js

- **TC15:** handleSubmit
    - **Description:**
    - **Main Flow:**
        - Given a submission event, metadata for the file is extracted
        - The file extension is verified, and parameters for the API are created
        - The parameters, containing the file are sent to S3, and an email is sent to the user account
        - The image object will be posted to the database
    - **Expected Outcome:** The submitted file is verified, saved to S3, and an email is sent to the user
    - **Alternative Flow:** error returned during S3 submission
    - **Expected Outcome:** A report saying submission failed will be returned to the user
    - **Alternative Flow:** file submitted contains no extension or no name
    - **Expected Outcome:** the method is ended with no submission executed

- **TC16:** _handleImageSubmit
    - **Description:** A new image will be saved in state
    - **Main Flow:**

        - Given an event, a file reader is created
        - Reader opens the file, and saves it to state
    - **Expected Outcome:** State will contain the newly added file

SignUp.js

- **TC18:** handleFormData
    - **Description:** method saves user account data in state
    - **Main Flow:**
        - Method extracts user's data to variables
        - Data saved in state
    - **Expected Outcome:** user's account data will be saved in state

Integration Testing

For integration testing, we need to make a formal system for chaining functions together. As it stands, because we are building a whole new system in itself, our entire progress relies on the fact that our modules work together, so testing has been happening as we have been developing. If our modules did not work together, we wouldn't be able to continue further progress, and as a result, would not be able to call our development complete and/or ready for testing, so what we essentially have is a retroactive integration testing suite.

More specifically relating to what would need to be integration tested would be all data being retrieved from our mongodb and AWS databases and being displayed to the user. This would be the biggest concern with our project as either of these two major parts (data retrieval and display) failing would stop the entire use of our project. By testing these two sides of the project we can ensure a quality product for our end users.

Our workflows for specific cases are outlined below. This demonstrates how our individual units work together to form the whole workflow for students and tutors.

ITC1 Signup workflow: User signs up for an account:

1. User navigates to signup page.

2. User signs up for an account using an email and password, filling out all fields
3. The user selects their college and courses they are enrolled in.
4. The system creates a document in MongoDB containing information about the user.
5. The system logs the user in from there they can resume an appropriate workflow.

Alternative Flow: 2b. Required fields aren't filled out

Expected outcome: User will be notified and account will not be created or uploaded into MongoDB, repeat step 2

Alternative Flow: 4a. Account already exists

Expected outcome: User will be notified, account will not be created or uploaded into MongoDB, return to step 1.

Alternative Flow 5a: Account is a tutor account

Expected Outcome: The tutor must wait to be granted tutor permission by an admin account to resume tutor workflow

ITC2 Student workflow: Student uploading an assignment:

1. The user signs into their account
2. The user then navigates to the Assignment Submission Page
3. The user can upload a file and select what course the image is for
4. The system ensures that the image is an allowed type (jpg, png, etc.)
5. The system creates an document in MongoDB to store information about the image
6. The System connects to AWS and stores the image there
7. The user must then wait for a video to be produced by a tutor
8. Once the tutor has uploaded the video (see tutor workflow) the user can purchase the video for a rate of 1$ per minute of video
9. The User can then purchase the video with Paypal or through a credit/debit card
10. The user enters their information and is processed, sending a confirmation to our application
11. The user can now view and download the video.

Alternative Flow: 1a. User doesn't have an account

Expected outcome: User navigates to signup page and creates an account (see signup workflow)

Alternative Flow: 4a. File is not one of the allowed types

Expected outcome: File is rejected and the user is notified, repeat step 5

Alternative Flow: 6a. AWS is not online

Expected outcome: Image is not uploaded and user is notified,go back to step 5.

Alternative Flow: 9a. User backs out of payment

Expected outcome: User is not granted access to video and cannot continue to step 11.

ITC3 Tutor workflow: Tutor uploading a video for an assignment:

1. Tutor signs into their account
2. The tutor navigates to the dashboard page
3. The tutor claims an image to upload a video for
4. The system adds tutor information to the associated image document in MongoDB
5. The system updates the status for the associated image document in MongoDB
6. The tutor navigates to the claims page
7. The tutor uploads a video for an image and submits
8. The system ensures that the video is an allowed type (mp4, wmv, etc)
9. The system adds video information to the associated image document in MongoDB
10. The system connects to AWS and stores the video there
11. The system sends a notification to the image uploader

Alternative Flow: 1a. Tutor doesn't have an account

Expected outcome: Tutor navigates to signup page and creates an account (see signup workflow)

Alternative Flow: 3a. There are no images to claim

Expected outcome: Tutor waits for images to be uploaded by students.

Alternative Flow: 8a. File is not one of the allowed types

Expected outcome: File is rejected and the user is notified, repeat step 5.

Alternative Flow: 10a. AWS is not online

Expected outcome: Image is not uploaded and user is notified, return to step 10.

Usability Testing

For usability testing, our main focus is on the functionality of how users interact with our web-application. In our requirements document, we outlined a set of functional requirements that our web-application should provide to users. Examples of these are: a user should be able to log-in or sign-up, upload images (for Student Users) or upload videos (for Tutor Users). Also in our requirements document, we wanted our web-application to be Intuitive to the users.

Our main goal here in usability testing, is to verify that our web-application is in fact Intuitive. In order to do so, we have to define this term a bit better, and then outline a plan to test for it. In our requirements document, we defined Intuitive as giving the users a specific tasks, such as Log-In, and assessing whether they could perform that within 5 clicks. Also, we want to measure how long it takes the user to perform the said task. We assume the amount of clicks, and the time taken to perform said task will inversely correlate with Intuitiveness of our web-application. For example, the easier it is to perform a task, the less time it should take the user to perform it, and the less clicks to accomplish it (meaning they don't have to use the back button anywhere).

One other form of testing we will use here is a questionnaire. Although it may be a simple form of testing, using a 1-5 rating system and asking the user "about ease of use" or frustration will help us verify our findings using the number of clicks, and timing measurements. So to recap, we will use a questionnaire, number of clicks, and timing to accomplish each task as our measurement values for testing. Below is an example of some tests that we will be performing, as well as maximum/minimum measurements we predict will showcase Intuitiveness of our system (adjusting these numbers if needed, based on our questionnaire responses).

Tests we will be performing are outlined in Table 1 below:

Task	# of Clicks	Time Taken (mins)
User can create an Account	5	2
Student User uploads a HW problem/image	4	2
Tutor User uploads a Video solution	4	2
Student can edit their profile	3	2

Table 1: Tasks to be tested, and their maximum threshold of measurements

Questions that will be asked to Users after they have performed a few of these Tasks (on a scale of 1 to 5, 5 being strongly agree):

1. Would you say the web-application was intuitive?
2. Could you easily perform your designated task/s?
3. Was the web-application easy to navigate?
4. Were the buttons and actions clearly identifiable, as well as what they did?
5. Did you enjoy your overall experience using the web-application?

We will test, at minimum, at least 3 specific Student Users and 2 specific Tutor Users. This will give us feedback from both sides of the user-base, with emphasis on Students, as they will be the primary business and fiscal users of the product. Based on our results by both analyzing the tests measurements as well as our findings using the questionnaires (which we want a minimum average of a 3), we will refactor certain processes or user-interface layout to make things clearer.

We realize this is version 1.0 of our web-application, and naturally as the software gets used more and more feedback is provided, changes will be made as necessary to simplify or update the user-interface and make it better. However, our main goal here is to have an intuitive enough system so that users can accomplish what they need to without being deterred by the software. With our testing metrics in place, we feel we will be able to both provide and prove that our software meets these requirements.

Conclusion

As stated in the previous paragraph, we feel confident in the testing plans we have put together. In fixing any side-effects, or errors, that we encounter, and using the test as feedback; we will be able to verify and provide that our software accomplishes and provides what we originally set out to build. Testing, like any software-development step, is there to keep the project and developers on track with the bigger picture of the project.

With that in mind, designing the tests and looking back at the requirements helped remind us of what we were trying to build and accomplish with this software. We were building a platform to provide a convenient place for students to submit images of homework assignments, and then receive online video-solutions for qualified tutors.

Thus making the tutoring aspect, and getting help with homework, a much simpler and convenient process for students.

Being close to the end, and seeing a somewhat final-version of our web-application, we are excited to present this piece of software to Mr. Lokken and the Tailored Tutoring Co. With these testing methods in place, we will be able to put the final touches on our web-application, with hopes that our software provides an easy to use interface and online-destination where both students and tutors can interact with each other to receive and provide help on their studies.