**Technical Documentation**

Online Tool for Fieldwork Planning and Risk Assessment

Group M – CITS3200

October 2017

Table of Contents

[**HTML DOCUMENTATION** 5](#_Toc496224292)

[1.1 form1 5](#_Toc496224293)

[1.2 form1Print 5](#_Toc496224294)

[1.3 form7 5](#_Toc496224295)

[1.3.1 Date pickers 5](#_Toc496224296)

[1.3.2 Adding Rows 6](#_Toc496224297)

[1.3.3 Toggling new info 6](#_Toc496224298)

[1.3.4 Validation and tickboxes() 6](#_Toc496224299)

[1.3.5 Delete 6](#_Toc496224300)

[1.3.6 HTML 7](#_Toc496224301)

[1.4 form7Review 7](#_Toc496224302)

[1.4.1 Viewing files 7](#_Toc496224303)

[1.4.2 Submit Form 7](#_Toc496224304)

[1.4.3 EJS and HTML 7](#_Toc496224305)

[1.5 form7StudentReview 8](#_Toc496224306)

[1.5.1 EJS and HTML 8](#_Toc496224307)

[1.6 Form7Print 8](#_Toc496224308)

[1.6.1 EJS and HTML 8](#_Toc496224309)

[1.7 riskassForm 8](#_Toc496224310)

[1.7.1 Listeners for putting risk score 9](#_Toc496224311)

[1.7.2 Adding Risks 9](#_Toc496224312)

[1.7.3 Adding Controls 9](#_Toc496224313)

[1.7.4 Deleting a control 9](#_Toc496224314)

[1.7.5 Deleting a Risk 10](#_Toc496224315)

[1.7.6 Validate Forms 10](#_Toc496224316)

[1.7.7 EJS and HTML 10](#_Toc496224317)

[1.8 Risk Assessment Form review 10](#_Toc496224318)

[1.8.1 Submit forms 10](#_Toc496224319)

[1.8.2 EJS and HTML 11](#_Toc496224320)

[1.9 riskAssFormStudentReview 11](#_Toc496224321)

[1.9.1 EJS and HTML 11](#_Toc496224322)

[1.10 riskAssFormPrint 12](#_Toc496224323)

[1.10.1 EJS and HTML 12](#_Toc496224324)

[**Style Guide Documentation** 13](#_Toc496224325)

[2.1 Fonts and Colours 13](#_Toc496224326)

[2.1.1 Font 13](#_Toc496224327)

[2.1.2 Colours 13](#_Toc496224328)

[2.2 CSS Package 13](#_Toc496224329)

[2.3 Page specifics 14](#_Toc496224330)

[2.3.1 The Header 14](#_Toc496224331)

[2.3.2 Form Content Styling 14](#_Toc496224332)

[2.3.3 Dashboard Content Styling 15](#_Toc496224333)

[2.3.4 Half-width Styling 15](#_Toc496224334)

[**Back-end Technical Documentation** 16](#_Toc496224335)

[3.1 Models 16](#_Toc496224336)

[3.1.1 account.js 16](#_Toc496224337)

[3.1.2 form1.js 16](#_Toc496224338)

[3.1.3 form7.js 16](#_Toc496224339)

[3.1.4 riskAssForm.js 16](#_Toc496224340)

[3.2 Routes 16](#_Toc496224341)

[3.2.1 admin.js 16](#_Toc496224342)

[3.2.2 changePass.js 17](#_Toc496224343)

[3.2.3 dashboard.js 17](#_Toc496224344)

[3.2.4 forgot.js 17](#_Toc496224345)

[3.2.5 form1.js 17](#_Toc496224346)

[3.2.6 form1Print.js 17](#_Toc496224347)

[3.2.7 form7.js 17](#_Toc496224348)

[3.2.8 form7Print.js 17](#_Toc496224349)

[3.2.9 form7Review.js 18](#_Toc496224350)

[3.2.10 form7StudentReview.js 18](#_Toc496224351)

[3.2.11 index.js 18](#_Toc496224352)

[3.2.12 register.js 24](#_Toc496224353)

[3.2.13 reset.js 24](#_Toc496224354)

[3.2.14 riskAssForm.js 24](#_Toc496224355)

[3.2.15 riskAssFormPrint.js 24](#_Toc496224356)

[3.2.16 riskAssFormReview 24](#_Toc496224357)

[3.2.17 riskAssFormStudentReview.js 24](#_Toc496224358)

[3.2.18 viewForms.js 25](#_Toc496224359)

[3.3 Application 25](#_Toc496224360)

[3.3.1 Database 25](#_Toc496224361)

# **HTML DOCUMENTATION**

## **1.1 form1**

The technical content within form1 are relatively straight forward. Within the JavaScript there are four main functions:

* checkpolicies()
* checktraining()
* checkmedical()
* checkgeneral()

These simply provide a way to ensure that all boxes that need to be checked have been checked. Within each one it will go through the names of each checkbox in the relevant section and check if it has not been ticked it will return false and provide an alert.

All four of these functions must return true for the file to be submitted. This is checked in the validateForm() function.

The HTML is straightforward and any changes that need to be made should be relatively simple.

## **1.2 form1Print**

Form1 print is again a relatively straight forward piece of code that shouldn’t cause too many issues. The major issue within this form is the use of ejs in the code. As a result a working knowledge of ejs is required to change code within this form. In particular ensuring that on if statements code is duplicated in both the if and else statements. Also ensure that all code that is to be posted on the page starts with the <%= tag.

## **1.3 form7**

The following section will outline all functions in form7.

### **1.3.1 Date pickers**

All functions that start with $(function () {$( "#id" ).datepicker are functions that run a jQuery script in jQuery UI that creates a date picker for that particular input field. Relevant changes that can be made to this date picker can be found in the jQuery UI documentation on their website.

### **1.3.2 Adding Rows**

Functions that start with either $(".addContact").click(function(){ or $(".addRow").click(function(){ are functions that dynamically add HTML to the page when adding fields such as UWA emergency contacts or itinerary details. This essentially is done as such:

* add 1 to variable that controls the id of said part of form (eg. rowNumItin).
* Create variable *row* that generates the HTML as a string.
* Find the correct table to append *row* to and call $(#tableid).append(row).

Please ensure that if adding listeners or other JavaScript objects that they are placed on every new HTML object that is created

### 1.3.3 Toggling new info

The function that starts with $(".togglable").click(function(){ is a function to toggle the extra info sections within the HTML. This code is relatively simple and is set to show be default in the css.

### **1.3.4 Validation and tickboxes()**

The validateForm() function runs several functions that set the form up to be submitted to the database. The first function that it runs is checkboxes(). This function simply checks that the agreement and permission boxes have been ticked within the form and it will return false if they have not. The form will not submit until this function returns true.

The next function that validateForm runs is a function called tickboxes(). What this code does is that on a successful submit it will go through and check all boxes in the declaration table. This is done as it is required within the mongoose database to have a value on every box otherwise displaying the values in the other forms will not work. As a result if a box is unticked it will be ticked and post a value of false to the database if a box has already been ticked then it will be posted with the value specified in its value tag in HTML.

### **1.3.5 Delete**

The function that starts with $(".delRow").click(function(){ is a function that deletes rows from the dynamic parts of form 7. First of all the function cheks if the id of the button that was pressed was itin or declaration. It then checks if the tables have more than one row. If they do it then removes the last row in the table and decreases the row counter by 1 e.g. *rowNumItin.*

### **1.3.6 HTML**

The HTML in this page is relatively simple the only thing of note is the required tags within the form and also the ejs tags at the top of the form. If u want to remove a field from being required please remove its required tag from that particular input. The EJS code at the start of form7 simply creates a date field. Also, to remove read only in the permission to proceed simply remove the read-only tag in the input. Also important to note is that any input with the same name tag will be posted into the database as an array and any individual names will be posted as single entries into the database.

## **1.4 form7Review**

See form7 for similar functions. The following section will include any functions not found in form7.

### **1.4.1 Viewing files**

The Code immediately below $(document).ready(function(){ that starts with <%for (var i = 0; i < form7.file.length; i ++) {%> is code that allows you to view the pdf that a person has uploaded. What it does is it takes the binary data in the file[] field in the database, changes it into base64 encoding before turning this base64 into a BLOB (Binary Long Object) and displaying that BLOB.

### **1.4.2 Submit Form**

This function located at the end of the script tag and named function submitForm(action) is a function that changes the action of the post within the form based on which button was clicked to submit the form. If the button of type submit is clicked then the form will post with the action /form7Review/<%=form7.\_id%> if the button of type button with class submit is clicked then the form will change the action of the form to "/form7Forward" + "/<%=form7.\_id%>". This is done to get the flow of forms to work correctly based on whether the form needs to be sent back for correction or is approved by the admin

### **1.4.3 EJS and HTML**

Within the EJS the major parts to be aware of are, firstly for for loops located at tables UWAcontact, itin and declaration. These loops allow all dynamically added HTML values to be rendered properly. Also it is important to understand how selecting checkboxes works in the declaration table. Within this each box is checked if the particular field within the database has a value that was posted in the input field e.g. “decVolunteer”. If the database has a value different from that value, e.g. false then it will uncheck the box.

The HTML in this form is essentially the same as that in form7 except for the ejs that gets data from the database. As a result please see form7 HTML documentation. Also, in order to work around needing buttons that firstly, are not changeable and secondly post data to the database all button that require this have the tag onclick=”return false;” this stops people from changing the values plus it stops allows the value to be posted to the database.

## **1.5 form7StudentReview**

See form7 and form7Review for all functions.

### **1.5.1 EJS and HTML**

The only change between this form and the other two form7s is that it includes a section for the admins comments to be displayed. These comments are called the same as all other fields that are created from the database. Otherwise the ejs that applied to form7 review applies to this form.

As mentioned above the only difference between this form and the other form7s is that it has places for comments. In particular most of the CSS and styling for this section was grabbed off of the jqueryui website and uses their highlight class. As a result any changes that are to be made to the comment sections must take this into account.

## **1.6 Form7Print**

### **1.6.1 EJS and HTML**

This form is roughly the same as the forms that have been put above it. The major difference between this form and the forms above is that all EJS is printed into <p> tags. Otherwise the EJS is the same as in the other form7s.

The major difference between this form and the other form7 is that most of the extra info that was put in the forms for filling out purposes has been removed. This includes all the extra info. If adding this code in is something that is needed in the print forms then that code can be found and copied in form7.ejs.

## **1.7 riskassForm**

The following section contains content regarding the risk assessment form.

### **1.7.1 Listeners for putting risk score**

All functions that start with $('#tableid').change(function(e) { are functions that are listening to a change in risk scores and then printing out the total score for that risk. Essentially the way in which this code works is that it firstly checks that the item that was clicked was a select button. It then checks whether the id has a number in the 1st or 2nd and 1st spot. If it is in the 2nd and 1st spot that means that it should update the Residual risk p tag and scores if it is a single number then it should update the identified risk p tag.

Within the function the parameter e is the object that calls the event.

### **1.7.2 Adding Risks**

Any function that starts with, $(".addtypeofrisk").click(function() { is a function that adds a risk to one of the tables. The way that this code works is that it increases the count of the number of risks in that table e.g. *genericcount*. It also resets the number of control that are within that particular risk to one e.g. *genericrcount.* It then creates a variable *row* that creates a string of the HTML to be added to that table. Finally, it gets the correct table and appends *row* to that table using $('#tableid').append(row);

### **1.7.3 Adding Controls**

Any function that starts with $("#tableid ").click(function(e) { is a function that adds a control to the risk directly above it. To do this it first checks whether the element that called the function is the add control button. It then adds one to the count of the number of controls on that risk. It then adds a variable *row* that creates a string of the HTML to be entered. It then gets the table row which the button was attached to by calling js.parent().parent(). Finally it then adds this code to the fourth table row above it with jx.prev().prev().prev().prev().after(row);

### **1.7.4 Deleting a control**

These functions start with the same call as the add controls function e.g. $('#tableid').click(function(e) { however, it differs by firstly checking if there is more than one control currently on the particular risk. It then checks whether the element that was selected was a delete control button by checking the classes. It then runs a for loop to delete the next three rows above the table row that the button is in.

### **1.7.5 Deleting a Risk**

Any function that starts with $('.deleteriskX').click(function() { is a function that deletes a risk from that particular table. The funcitons works by firstly checking if there is more than one risk in the table. E.g. genericcount > 1. If it is it creates a while loop that will delete rows until it reaches a row that has the class ‘lastrow’. It will then finally remove this class and then say that it has removed this risk by updating the risk counters to 1 less.

### **1.7.6 Validate Forms**

The validate forms function has only one thing to validate before the form is submitted. That is to check the date of expiry to ensure that it is within five years of the current date. This is done in the checkdates() function and simply gets the current date adds 5 years to it and then checks the expDate against this new date and return false if expDate is greater.

### **1.7.7 EJS and HTML**

The only EJS in this form is a simple check on whether to render the page or not. What it does is it checks the date of all form1s that a user has submitted and checks whether they have filled in a form 1 within the last year. If they have not then the page will not render.

The HTML in this form is relatively simple and is similar to code in the other forms. The major things to note within this is to change the required tags within each input that you do, or do not want to have to be submitted. Also if you would like to change the read only status on the Implementation, then need to change the read-only tags in the input. Finally, any input tags with the same name will be posted to the database as an array and any input tags with different names will be posted as their own separate values. Also each new control is given a hidden value that is associated with it. In particular the last control in a risk is given a value of 1 and all other risks have a value greater than 1.

## **1.8 Risk Assessment Form review**

The following section contains content regarding the risk assessment form review. Any functions that are similar can be found in the section above, 1.7 riskAssForm.

### **1.8.1 Submit forms**

This function runs similar to the function in form7 review with the same name so if the use select the button with the value send back to user, the normal action /riskAssFormReview/<%=riskform.\_id%> will be posted however, if they select the button then the function submitForm(action) will be run this will change the action value on the form to action","/RiskAssForward" + "/<%=riskform.\_id%> and then submit the form. This runs through the same usual checks including validateForm as the other submit does.

### **1.8.2 EJS and HTML**

The EJS in this section is the largest and most complicated in the whole document the major thing to note are the for loops that run through each of the four risks based on the number of risks that are there. This is based on the *Xname* arraylength. The next thing to note is the while loop on the four forms that will correctly display the number of controls that are associated with each risk. This done by giving each control a value. If this value is 1 then it is the last control for a particular risk. So the while loop will run until it hits a 1 it will then render that last control outside of the loop as it includes the risk scores as well.

Finally, looking at the rendering of values, what happens is that a switch statement is run for all select statements that will get the value of that select statement from the database and then it will create a select option with the correct option selected based on the database value. Apart from these things most of the ejs is just simply getting data from the database and displaying it.

The HTML in this form is the same as that in the risk assessment form except for the selecting of different options in select statements based on, what value the database gives that select statement.

## **1.9 riskAssFormStudentReview**

The following section contains content regarding the risk assessment form student review. All functions can be found in the sections above, 1.7 riskAssForm and 1.8 riskAssFormReview.

## **1.9.1 EJS and HTML**

The only change between this form and the other two risk assessment forms is that it includes a section for the admins comments to be displayed. These comments are called the same as all other fields that are created from the database. Otherwise the ejs that applied to risk assessment form review applies to this form.

As mentioned above the only difference between this form and the other risk assessment forms is that it has places for comments. In particular most of the CSS and styling for this section was grabbed off of the jQuery UI website and uses their highlight class. As a result, any changes that are to be made to the comment sections must take this into account.

## **1.10 riskAssFormPrint**

### **1.10.1 EJS and HTML**

This form is roughly the same as the forms that have been put above it. The major difference between this form and the forms above is that all EJS is printed into <p> tags. Otherwise the EJS is the same as in the other risk assessment forms.

The major difference between this form and the other form7 is that most of the extra info that was put in the forms for filling out purposes has been removed. This includes all the definition tables. If adding this code in is something that is needed in the print forms then that code can be found and copied from the riskForm EJS.

# **Style Guide Documentation**

## **2.1 Fonts and Colours**

The system uses UWA-approved fonts and colours to maintain consistency with other internal and external applications, these guidelines are correct as of October 2017, however they may change in the future.

### **2.1.1 Font**

The system uses Source Sans Pro.

This font is freely available at: <https://fonts.googleapis.com/css?family=Source+Sans+Pro>

### **2.1.2 Colours**

The colours utilized by the system are listed below:

**UWA Blue, Hex code #27348b**

**UWA Gold, Hex code #e2b600**

**UWA Silver, Hex code #ececec**

**Cream, Hex code #** **#f2edd9**

## **2.2 CSS Package**

The system uses Pure.css (<https://purecss.io/>) to automatically style many of the pages, Pure has many features that improve the look of HTML forms, hence it was chosen. The system uses Pure’s responsive grid

## **2.3 Page specifics**

Each page follows roughly the same design. The code is listed below with some comments explaining the intention of each line:

### **2.3.1 The Header**

*Pure grid system:*

<div class="pure-g">

*Header enclosure:*

<div class="pure-u-1" id="header">

*Header image:*

<img src="./images/backgroundDefault.jpg" style="top: -80px; position:relative; z-index: -1;">

</div>

</div>

### **2.3.2 Form Content Styling**

<div id="excludingHeader">

<div class="pure-g">

*All pages contain sidebars that are designated as follows:*

*Left-side sidebar:*

<div class="pure-u-lg-1-5" id="sidebar"> </div>

<div id="formArea">

*Form title goes within the following div:*

<div id="formTitle"> Form 7 - Planning </div>

*Sections are enclosed in the following container*

<div class="sectionContainer">

*Section titles are enclosed in the following container*

<div class="sectionTitle"> Fieldwork Description </div>

*Form content is enclosed in the following container*

<div class="input">

Content content content

</div>

</div>

</div>

*Right-side sidebar*

<div class="pure-u-lg-1-5" id="sidebar"> </div>

</div>

</div>

### **2.3.3 Dashboard Content Styling**

*Full-width section:*

*This class is used for a full-width container*

<div class="pure-u-1 pure-u-lg-2-3">

*This div is used for encapsulating a column of a form*

<div id="formArea">

*This div is used for encapsulating a section, similar to forms*

<div class="sectionContainer">

<div class="sectionTitle"> Welcome </div>

*This div is used for encapsulating content*

<div class="dashboardContainer">

*This div is used to add a scrollbar to the container*

<div class="scrollContainer" style="height:400px;">

Content Content

</div>

</div>

</div>

</div>

</div>

### **2.3.4 Half-width Styling**

Same as above, except the initial Pure container class is replaced with the following:

*This class is used for a half-width container*

<div class="pure-u-1 pure-u-lg-1-3">

# **Back-end Technical Documentation**

## **3.1 Models**

The models folder contains the schemas for the collections in the database. All the fields described in a model are the fields of that particular collection.

### **3.1.1 account.js**

The account model defines all the fields in an account for our system. The fields are defined and then a type is assigned to that particular field. In the case of the account model, the admin also has a default value set to false so all users of the system start out without having admin privileges.

### **3.1.2 form1.js**

The form1 model defines all the fields that are visible and need to be incorporated from Form 1. All of the fields have their names defined as well as the type that will be accepted into the database upon a post request.

### **3.1.3 form7.js**

The form7 model defines all the fields that are visible and need to be incorporated in Form 7. All of the fields have their names defined as well as the type that will be accepted into the database upon a post request.

### **3.1.4 riskAssForm.js**

The riskAssForm model defines all the fields that are visible and need to be incorporated from the risk assessment form. All of the fields have their names defined as well as the type that will be accepted into the database upon a post request.

## **3.2 Routes**

The routes folder contains routes for each view in the system. The routing determines how the application responds to the clients requests.

### **3.2.1 admin.js**

When the client requests the admin view, the router searches the database for all accounts who are not administrators and passes them into the render of the admin view.

### **3.2.2 changePass.js**

When the client request changePass, the router renders the changePass view, passing in the currently logged in user’s information.

### **3.2.3 dashboard.js**

When the client requests the dashboard, the database is queried for all the form 1’s, form 7’s and risk assessment forms that the logged in user has submitted, as well as all the form 1’s, form 7’s and risk assessment forms where the admin field matches the currently logged in user’s username. This information is then passed into the render of the dashboard.

### **3.2.4 forgot.js**

When the client requests the forgot password page the page is rendered with nothing passed into it, as no user will be logged in as they have forgotten their password.

### **3.2.5 form1.js**

When the client requests form 1, the form is rendered with the user’s information passed in. This is so the form can be tied with the currently logged in user.

### **3.2.6 form1Print.js**

When the client requests the form1Print view, there will be the id of the form the user clicked on passed into the router call. This id is then queried in the database and the form found is passed into the render of form1Print. When the print button on the form1Print page is pressed, the router posts and creates a pdf which is viewed in the browser. Once the pdf has been created and opened for viewing it is deleted from the file system.

### **3.2.7 form7.js**

When the client requests form 7, the database is queried for a form 1 by the same user and then the result is passed into the render of form1. This result is rendered in because if a user is to complete any other forms (form7 and risk assessment form) they must first complete form1.

### **3.2.8 form7Print.js**

When the client requests the form7Print view, there will be the id of the form the user clicked on passed into the router call. This id is then queried in the database and the form found is passed into the render of form7Print. When the print button on the form7Print page is pressed, the router posts and creates a pdf which is viewed in the browser. Once the pdf has been created and opened for viewing it is deleted from the file system.

### **3.2.9 form7Review.js**

When the client request form7Review, the form selected has its id queried in the database and searches for a form 7 that matches the id. The form that is returned is passed into the render of form7Review. This review is for administrators only.

### **3.2.10 form7StudentReview.js**

When the client requests form7StudentReview, the form selected has its id queried in the database as is it passed in, as a parameter, and searches for a form 7 that matches the id. The form that is returned is passed into the render of form7StudentReview. This review is for the user who submitted the form to edit any fields they may have entered incorrectly or forgotten.

### **3.2.11 index.js**

Relevant models and node packages are defined at the beginning of the file for use later.

Get homepage - When the client requests the index (landing) page, the router gets the call and renders the index view.

Get register - renders the register view

Post register - When the client fills in the form on the register page, this post is called. The post registers a new Account into the database, getting the relevant information provided from the body of the view. I then authenticates the user with passport and redirects the user to the dashboard.

Post login - When the client logs in, the login post is called. This checks the database using passport.authenticate for the user, and if the user has a valid account, it is authenticated and the client gets redirected to the dashboard.

Post form1 - When the client submits form1, the form1 post is called. This post creates a new Form1 and sets the appropriate fields from what the client entered in the body of the form1 view. The owner field is set to the currently logged in user and there is no admin field as form1 does not need to be reviewed by any admins. The forms state is set to complete. The date filled field is filled in from the variable ‘today’ which calculates the current date at the date of form submission. The client is then redirected to the dashboard after successful submission, or shown an error message if not successful.

Post form7 - When the client submits form7, the form7 post is called. This post creates a new Form7 and sets the appropriate fields from what the client entered in the body of the form7 view. For form7, formidable body parsing was used so that the user can upload pdf files (uses multipart/form-data encoding type) in their form7. From the parser call, fields becomes an array containing data from the posted form, data is identified by keys set to their names in the input tags within the form, the files parameter is set to an array of the files uploaded. The admin field for creating Form 7 is set to the fieldwork coordinator specified by the user submitting the form. The state of the form is set to pending so that the administrator can review the form, and the forwarded field is set to 0 to let the application know that this form has not been signed off on and forwarded yet. After the fields are added, the post goes on to handle file input. If the length of the variable files.filestoupload (filestouplad is the name of the file input in form 7) has type “undefined” there were either 0 or 1 files uploaded, otherwise multiple files were uploaded and are all stored in an array. If no files were uploaded the file will not be able to be loaded and the code will do nothing. If only one file was uploaded the file’s name is checked so that it’s last 4 characters are “.pdf”. If not the code skips to the end of the if-else statement. If the extension indicates it’s a pdf, the file’s address in the temp folder is found and the file is read then converted to a binary object. At this point the file is checked to be between 0 and 10MB, if it is outside of these parameters the file is ignored. If it is of correct size, the newly created form 7 document in this post has it’s “file” field updated and the file is pushed to the back of the form 7 file field (which is initially empty). If the files.filestoupload variable is not null, it is the number of uploaded files. In this event the procedure for one file is repeated for each file and every correct size pdf file is stored in the form 7 file field. For file troubleshooting refer the formidable github documentation (<https://github.com/felixge/node-formidable>). Easy ways to check file attributes include file.fileinputname.length() to check a files length or whether a file was found at all and referring to errors logged by the console after callback functions (normally of form: if(err) console.log(err)). The client is then redirected to the dashboard after successful submission, and shown an error message if the submission was unsuccessful.

Post form7Review - When an administrator submits their review of a form7 back to the user, the form7Review post is called. This post is called with the id of the form that is being reviewed, as a parameter. This id is then queried in the Form7 collection of the database and the resulting form has all of its fields updated from what the client has entered (or left the same) in the body of the form. The admin field is set to the fieldwork coordinator to begin the forwarding process again. The state of the form is set to “Reviewed by Admin” and the forwarded field is set to 0 to reset the flow of the forms to go back to the fieldwork coordinator again after the user submits their fixed version of the form. The client is then redirected to the dashboard after successful submission, and shown an error message if the submission was unsuccessful.

Post form7StudentReview - When the user submits their form 7 after it has already been reviewed by an admin, the form7StudentReview post is called. This post is called with the id of the form that is being reviewed, as a parameter. This id is then queried in the Form7 collection of the database and the resulting form has all of its fields updated from what the client has entered (or left the same) in the body of the form. The state of the form is set to “Reviewed by User”. After the fields are added, the post goes on to handle file input. The way files are processed here are the exact same as within the post form 7 except now the file field in the form 7 document is unset before update (this only happens if more files are uploaded). This ensures the code does not just add new files to the file field but instead resets the field to only be whatever new files were uploaded. The client is then redirected to the dashboard after successful submission, and shown an error message if the submission was unsuccessful.

Post form7Forward - When the administrator signs off on a form 7, the form7Forward post is called. This post is called with the id of the form that is being reviewed, as a parameter. This id is then queried in the Form7 collection of the database and the resulting form has all of its fields updated from what the client has entered (or left the same) in the body of the form. If the number of times the form has been forwarded is 0, forwarded gets set to 1, the state of the form gets set to “Forwarded by Admin” and the admin the form goes to is the admin that was specified on the form as being the “Supervisor”. The client is then redirected to the dashboard. If the number of times the form has been forwarded is 1, forwarded gets set to 2, the state of the form gets set to “Forwarded by Admin” and the admin the form goes to is the admin that was specified on the form as being the “Head of School”. The client is then redirected to the dashboard. If the number of times the form has been forwarded is 2, then the forwarded value gets set to 3 and the state of the form is “Complete”. The client is then redirected to the relevant form 7 print page and has the option to print as a pdf.

Post riskAssForm - When the client submits the risk assessment form, the riskAssForm post is called. This post creates a new Risk Assessment Form and sets the appropriate fields from what the client entered in the body of the riskAssForm view. The admin field for the creation of the risk assessment form is set to the fieldwork coordinator set by the user submitting the form. The state of the form is set to pending so that the administrator can review the form, and the forwarded field is set to 0 to let the application know that this form has not been signed off on and forwarded yet. The client is then redirected to the dashboard after successful submission, and shown an error message if the submission was unsuccessful.

Post riskAssFormReview - When an administrator submits their review of a risk assessment form back to the user, the riskAssFormReview post is called. This post is called with the id of the form that is being reviewed, as a parameter. This id is then queried in the RiskAssForm collection of the database and the resulting form has all of its fields updated from what the client has entered (or left the same) in the body of the form. The admin field is set to the fieldwork coordinator specified by the user who submitted the form previously, so that the sign off chain is reset. The state of the form is set to “Reviewed by Admin” and the forwarded field is set to 0 to reset the flow of the forms to go back to the fieldwork coordinator again after the user submits their fixed version of the form.

Post riskAssFormStudentReview - When the user submits their risk assessment form after it has already been reviewed by an admin, the riskAssFormStudentReview post is called. This post is called with the id of the form that is being reviewed, as a parameter. This id is then queried in the RiskAssForm collection of the database and the resulting form has all of its fields updated from what the client has entered (or left the same) in the body of the form. The state of the form is set to “Reviewed by User”. The client is then redirected to the dashboard after successful submission, and shown an error message if the submission was unsuccessful.

Post riskAssForward - When the administrator signs off on a risk assessment form, the riskAssForward post is called. This post is called with the id of the form that is being reviewed, as a parameter. This id is then queried in the RiskAssForm collection of the database and the resulting form has all of its fields updated from what the client has entered (or left the same) in the body of the form. If the number of times the form has been forwarded is 0, forwarded gets set to 1, the state of the form gets set to “Forwarded by Admin” and the admin the form goes to is the admin that was specified on the form as being the “Supervisor”. The client is then redirected to the dashboard. If the number of times the form has been forwarded is 1, forwarded gets set to 2, the state of the form gets set to “Forwarded by Admin” and the admin the form goes to is the admin that was specified on the form as being the “Head of School”. The client is then redirected to the dashboard. If the number of times the form has been forwarded is 2, then the forwarded value gets set to 3 and the state of the form is “Complete”. The client is then redirected to the relevant risk assessment form print page and has the option to print as a pdf.

Post makeAdmin - When an administrator clicks on the button to make a user admin on the admin control page, the makeAdmin post is called. This post is called with the id of the user to get admin privileges as a parameter. This id is then queried in the Accounts collection of the database and the resulting account gets their admin field set to “true”. The account changes are then saved and the client is redirected to the dashboard.

Post changePass - When the user submits to change their password on the changePass view, the changePass post is called. This post queries the Account collection in the database with the currently logged in user and then the users password is set to whatever the user had in the body of the view. The user’s details are then saved and the user is redirected to the dashboard.

Post forgot - When the client submits their email address on the forgot password page, the forgot post is called. A token is then assigned to the user so they can reset their password with it later, and the Account collection of the database is queried with the given email address (username) the client entered. The tokens are then set in the user and is set to expire in one hour from the submission of the forgot password form. The user’s token and expiry are then saved and an email is sent to the user, containing a link containing the token they need to reset their password. The email is sent from a custom Gmail account created for this purpose. The email and password for the account are found in this post.

Post reset - When the client enters their new password from the reset page, the reset post is called. The reset post has the token created in the forgot post so that the system is able to find the correct account to change the password on. The token is queried in the Accounts collection of the database and the user found gets their password set to whatever the client entered into the view. The user is then saved and the token and token expiry are reset to be undefined. An email is sent to the user that the password was changed for confirming their password change. The user is then redirected to the login page.

Post logout - When the client clicks on logout on the dashboard, the logout post is called. The user is logged out and is then redirected back to the login page.

Post fileupload1 - When the client submits a new first aid pdf from the dashboard the fileupload1 post is called. The post will parse the file with formidable and then check it has a pdf extension and it is between 0MB and 10MB. If these checks fail, nothing is entered into the database. Otherwise the file data is read into the server and converted to binary data for storage. The document with the same username as the currently logged in user (the owner of this dashboard) is then found and it’s firstAid field is set to a buffer object containing the file data. Whether the upload goes through or not the page will reload (redirect to itself). As node operates asynchronously the redirect often occurs before the file has been uploaded to the database and so even though the page is reloaded the file will not yet be in the system. Another refresh should fix this.

Post fileupload2 - When the client submits a new offroad/4WD pdf from the dashboard the fileupload2 post is called. This post is identical to fileupload1 but instead of updating the first aid field it updates the off field of an Account.

Post fileupload3 - When the client submits a new licence pdf from the dashboard the fileupload3 post is called. This post is identical to fileupload1 but instead of updating the first aid field it updates the licence field of an Account.

### **3.2.12 register.js**

When the client requests the register page, the register page is rendered as is.

### **3.2.13 reset.js**

When the client requests reset it will be through a link that was sent to their email. The reset page is where the client will reset their password if they have forgotten it. The router gets the page with a token parameter. This token is then searched through the accounts in the database’s tokens and when one matched the correct account has been found. If no user is found the client gets redirected to the landing page. If a user is found then the reset page is rendered with the correct user passed in so they can reset their password.

### **3.2.14 riskAssForm.js**

When the client requests the risk assessment form, the database is queried for a form 1 by the same user and then the result is passed into the render of form1. This result is rendered in because if a user is to complete any other forms (form7 and risk assessment form) they must first complete form1.

### **3.2.15 riskAssFormPrint.js**

When the client requests the riskAssFormPrint view, there will be the id of the form the user clicked on passed into the router call. This id is then queried in the database and the form found is passed into the render of riskAssFormPrint. When the print button on the riskAssFormPrint page is pressed, the router posts and creates a pdf which is viewed in the browser. Once the pdf has been created and opened for viewing it is deleted from the file system.

### **3.2.16 riskAssFormReview**

When the client request riskAssFormReview, the form selected has its id queried in the database ,as the id is passed in as a parameter, and the database searches for a risk assessment form that matches the id. The form that is returned is passed into the render of riskAssFormReview This review is for administrators only.

### **3.2.17 riskAssFormStudentReview.js**

When the client requests riskAssStudentReview, the form selected has its id queried in the database ,as is it passed in as a parameter, and searches for a risk assessment form that matches the id. The form that is returned is passed into the render of riskAssStudentReview. This review is for the user who submitted the form to edit any fields they may have entered incorrectly or forgotten.

### **3.2.18 viewForms.js**

When the client requests viewForms, the database searches for all form 1’s, form 7’s and risk assessment forms of the username that is passed in as a parameter. The forms that are found are then passed into the render of viewForms. It also passes in the account information of that person.

## **3.3 Application**

The file app.js in our system defines the application and tells the application where certain files are, as well as handling the connection to the database and user authentication. At the beginning of the file, important node packages that are required are defined for use later. Next, all of the routes for all of the views in our system are defined and the application is told where it can find all of these files. The application then sets up the view engine to accept embedded javascript files as the view files and then node packages are setup including the logger, bodyParser, cookieParser, and passport. The application then defines all of the views in the system as it was told earlier where they were located in the file system. Passport is then setup using the account model we defined earlier. Finally, the application connects to the database we have set up using correct mongo authentication to connect to the users database. Lastly, an error handler is set up and the application listens on port 3000. The listening is mainly for localised testing of our application.

### **3.3.1 Database**

The database used in this application is a Mongo database. To access the database you will need the mongo shell installed. For instructions on how to do this, please visit the documentation for Mongo.

The account with administrative access in the database:

Username: usersAdmin

Password: cits3200project

To access the database through the mongo shell:

Enter the command:

>mongo 130.95.176.159/users -u usersAdmin -p cits3200project

Or:

>mongo 130.95.176.159

>use users

>db.auth(“usersAdmin”, “cits3200project”)