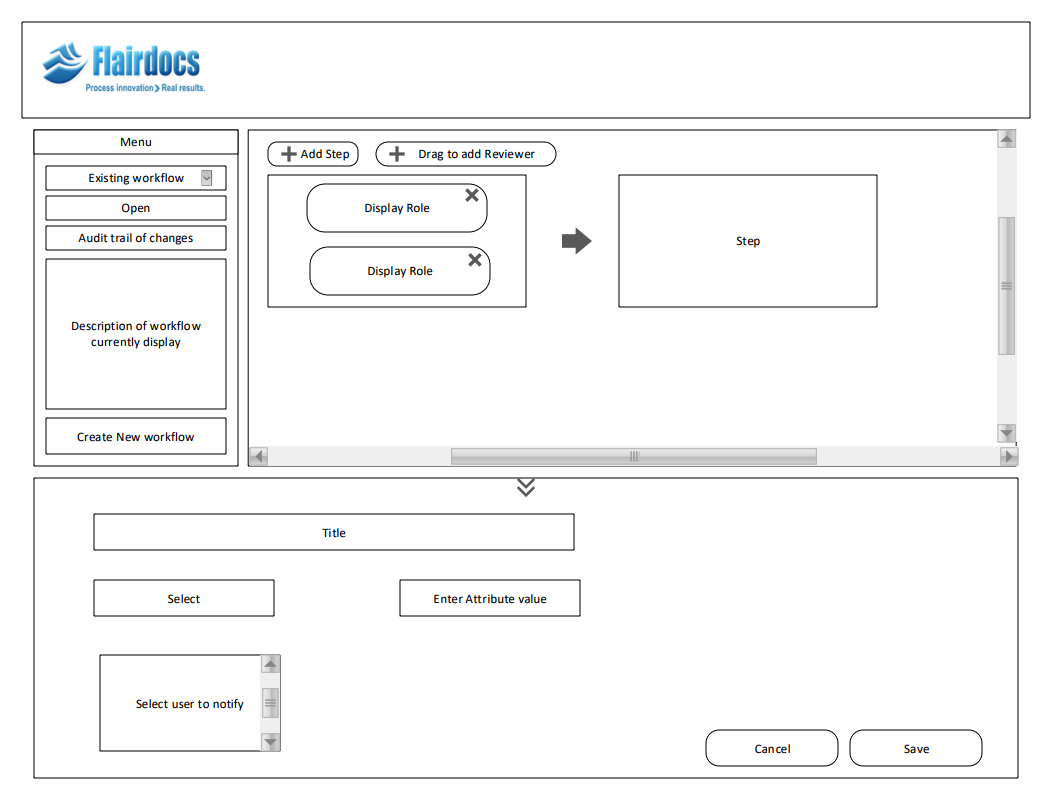
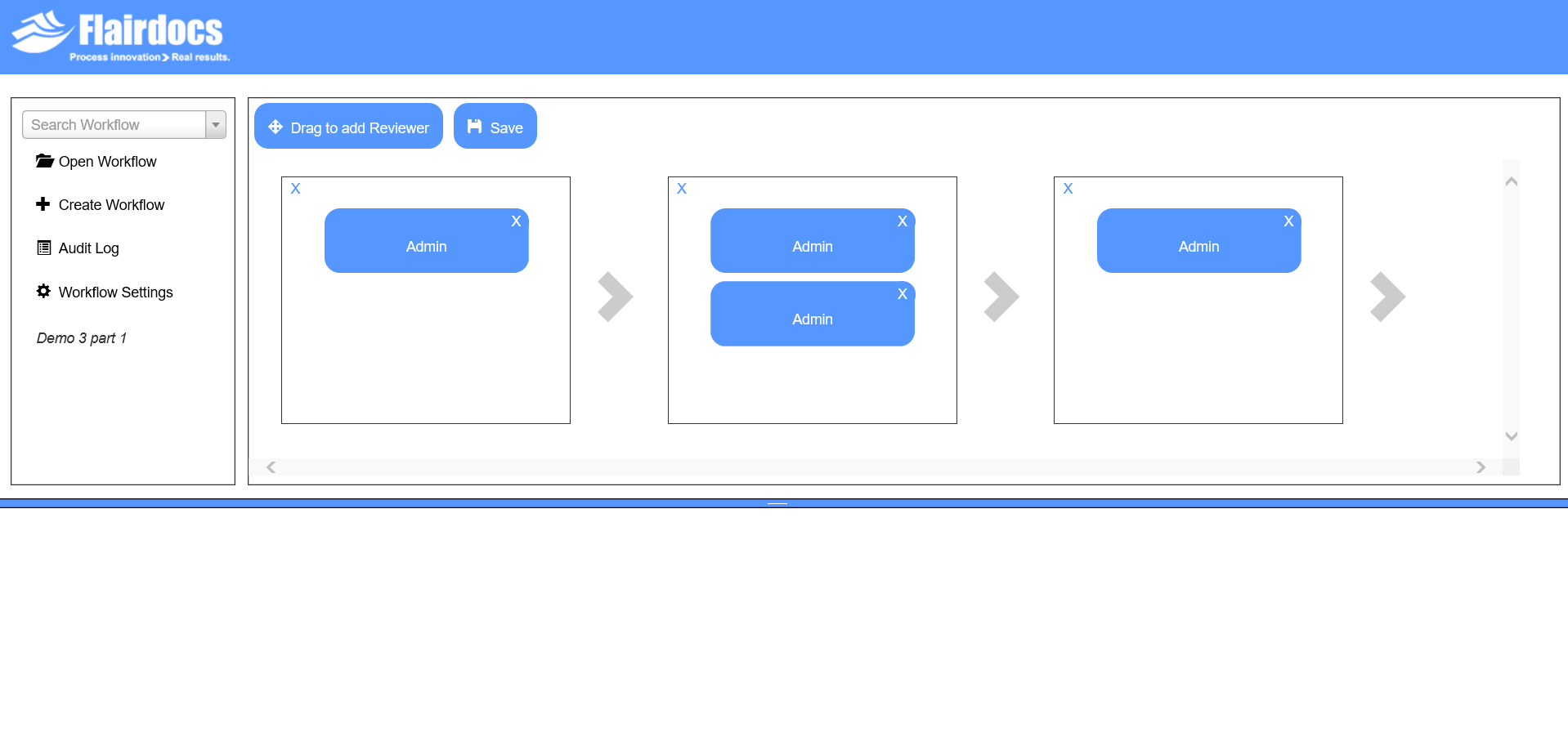
**1. Workflow Designer Tool Initial Design**

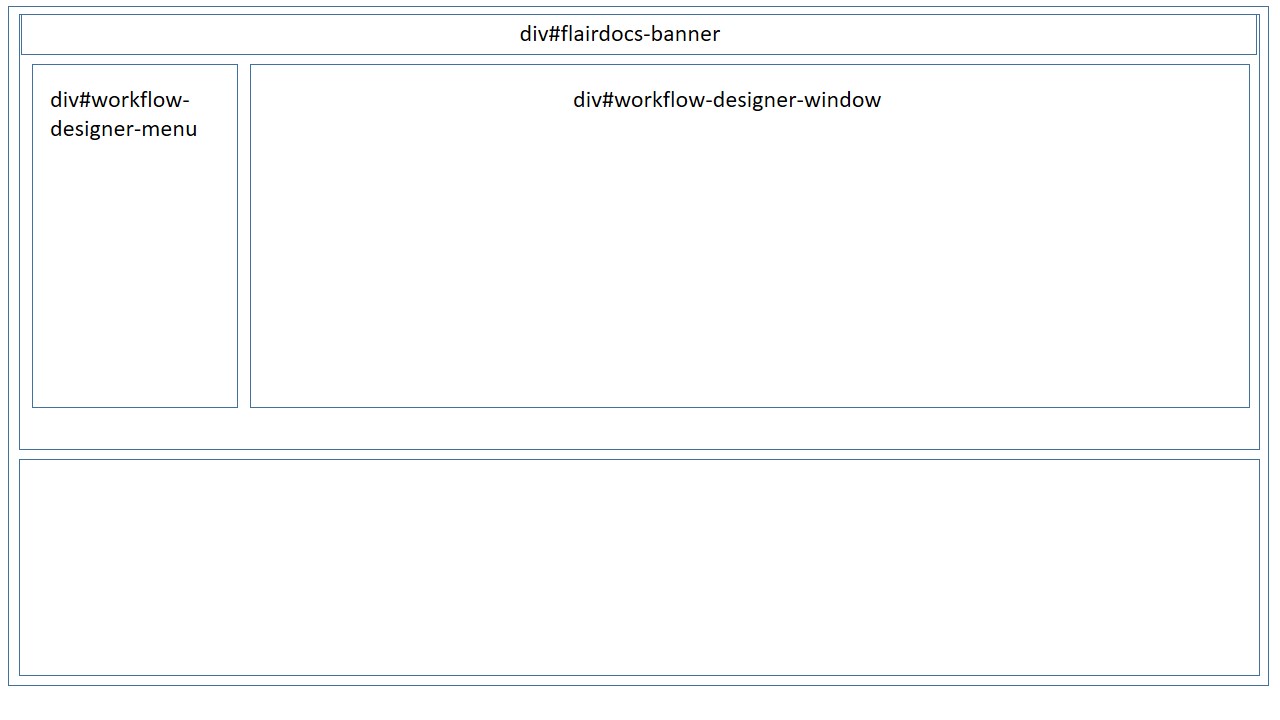


**2. Workflow Designer Tool as of 11/29/2017**

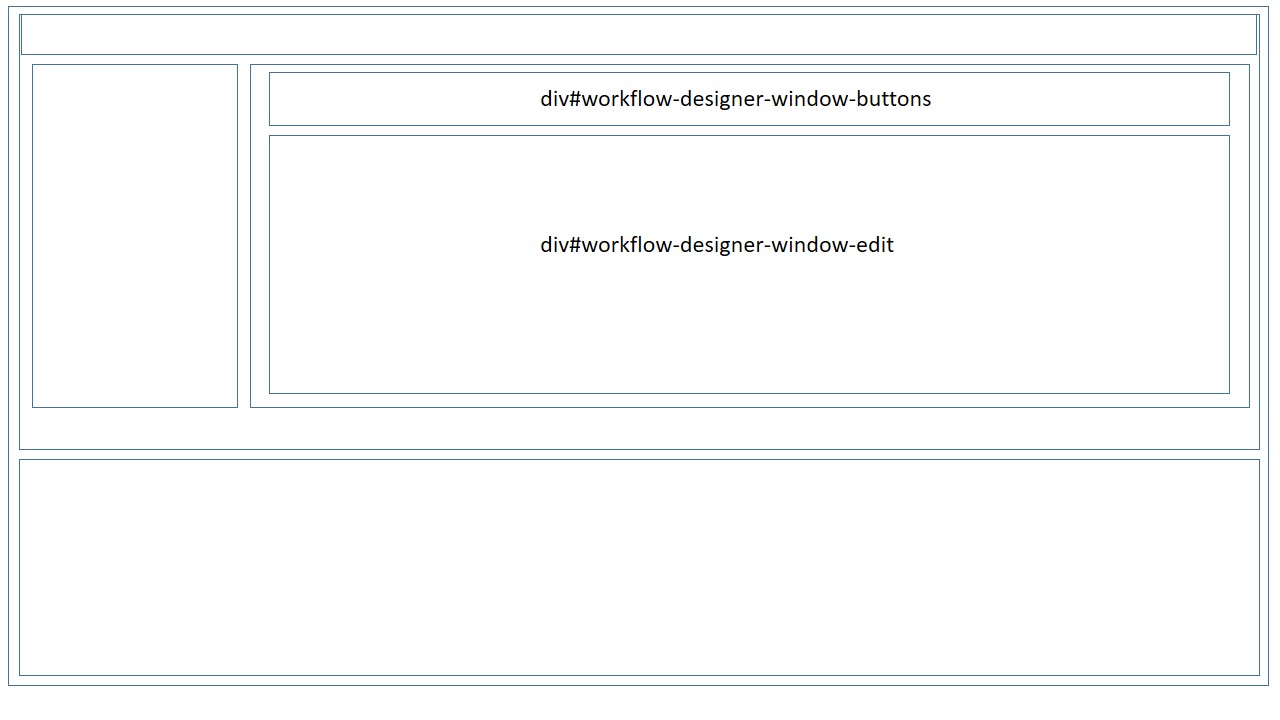


**3. Workflow Designer Tool Layout (HTML Overview)**

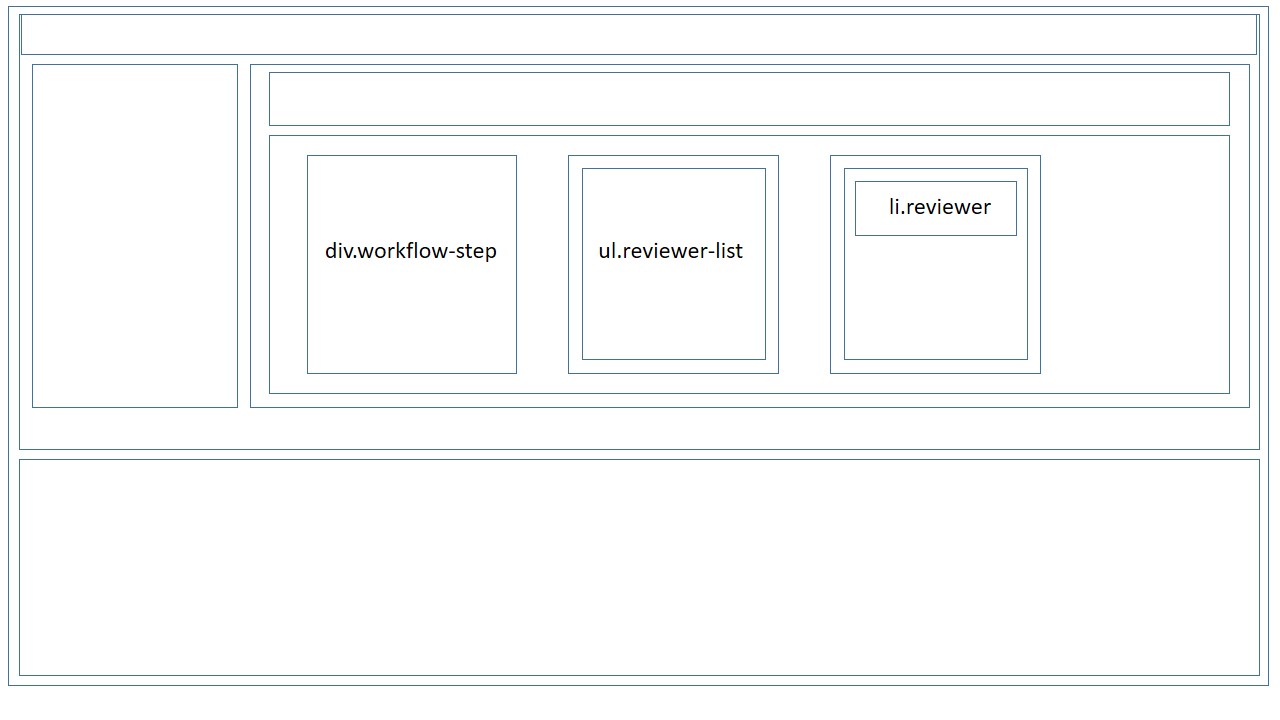
Supporting the workflow designer tool is a toolkit called Dojo. The workflow design tool uses a feature from the Dojo library called BorderContainer, which allows for the content of the webpage to be divided into two separate panels with a live splitter. The top panel contains the menu as well as the workflow design window. The bottom panel, called the center panel by dojo convention, contains information regarding a specific instance of a reviewer selected from the workflow in the top panel. This project focuses almost exclusively on the features contained in the top panel. Below we will illustrate the contents of the top panel at a high level.



Above is the base layout in the top panel. However surrounding the workflow-designer-menu and the workflow-designer-window is a workflow-designer-top-wrapper which is not depicted above. The purpose of the workflow-designer-top-wrapper is to set up a flex layout so that the workflow-designer-menu and the workflow-designer-window can be displayed horizontally rather than vertically. The workflow-designer-menu houses basic operations like searching, opening and creating a workflow. It also includes endpoints to view an individual workflow’s audit log and change an individual workflows’ settings. At the heart of the top container is the workflow-designer-window, which is where the user can view and manipulate a workflow through the GUI.



The workflow-designer-window has two direct children, workflow-designer-window-buttons and workflow-designer-window-edit. The workflow-designer-window-buttons houses the buttons used to manipulate and save a workflow. The workflow-designer-window-edit reflects the current state of a workflow graphically and allows the user to make changes to the state.



Within the workflow-designer-window-edit is the visual representation of a workflow. It consists of some series of workflow steps ordered left to right. Additionally each step will have some number of reviewers. A reviewer is an entity with some responsibilities to fulfill at a given step in the workflow. The reviewers are housed within a reviewer-list and each step in the workflow contains a reviewer list. In the illustration above, each workflow-step contains every element layed out, but the image has been broken apart to demonstrate the layering.

**4. Libraries and Dependencies**

This project has made use of several different libraries to support implementation of various workflow designer features.

***Dojo***

At the base of the html for the workflow designer tool is the Dojo Toolkit. Dojo allows for having two separate content panels that can be resized and manipulated easily. To create this behavior designer tool uses a BorderContainer with two ContentPanes. The top ContentPane houses the workflow designer menu and the workflow designer window. The bottom ContentPane, denoted by region:center, contains information about a single reviewer instance.

***Sweet Alerts***

To create aesthetic alert boxes the designer tool makes use of the sweet alerts library. The alert boxes created using sweeting alerts can be found when creating a workflow, or deleting a step or reviewer.

***Select2***

In consideration of search, we wanted to have a ‘live search’ ability that could narrow down the user’s query options with every character entered into the search box. Creating such a feature from scratch would be rather tedious. Instead the designer tool uses a library called select2 to create the ‘live search’ feature.

***jquery UI***

Drag-and-drop is a very important feature of the designer tool in order to have a tool that is ‘user friendly’. Manipulating a workflow using drag-and-drop is challenging for the developer, but essential for any user who wants an intuitive tool. We were able to implement the drag-and-drop functionality with the help of jquery UI, which allows us to create sortable lists and easily create the JavaScript for dragging and dropping elements.

***Selenium***

For GUI testing, we wanted something different than Microsoft Unit Testing. Selenium was a great option because of the large community and ample web drivers that can be found and installed through NuGet package manager in Visual Studio.

**5. GUI Elements**

Described in section 3 was the general layout of the html the makes up the workflow designer gui. However it is also important to know all of the additional buttons and view objects.

***select.workflow-live-search***: Search box for workflows within the system.

***button#open-workflow-button***: Button to open the workflow that is stored in the search box.

***button#create-workflow-button***: Button to create a new workflow.

***button#workflow-audit-log-button***: Button to view an audit log for the current workflow.

***button#workflow-settings-button:*** Button to show the current workflows settings

***div#add-reviewer-button***: Button used to drag and drop a reviewer into the workflow.

***button#save-workflow-button***: Button to save the current state of the workflow, not including reviewer attributes.

***div.add-step***: Button to add a step into the workflow, the new step will be added directly to the right of the particular instance clicked.

***div.workflow-step***: Div to represent a step in the workflow.

***div.reviewer-list***: An unordered list to store all of the reviewers of a given workflow-step.

***li.reviewer***: A list item that represents a single reviewer in the workflow. These can be drag and dropped to any reviewer-list in the workflow.

***button.remove-step-button***: Button to remove a particular step in the workflow, upon confirmation the selected step will be deleted from the workflow.

***button.remove-reviewer-button***: Button to remove a particular reviewer in the workflow, upon confirmation the selected step will be deleted from the workflow.

**6. Controller Interaction**

The workflow designer tool would be rather useless if the user cannot save, delete, and view workflows. In order to perform these actions the GUI must be able to communicate with the controller. In it’s current design, the workflow designer tool communicates to the controller via JavaScript and XMLHttpRequests. Additionally, the controller routing is created using Razor syntax within the cshtml file. This solution is better than hard-coding routes within the JavaScript, as these routes may change over time. Here is an example of using JavaScript, XMLHttpRequests, and Razor syntax to communicate with the controller to open a workflow:

function open\_workflow() {  
 if ($('.workflow-live-search').select2('data')[0].text != "") {  
 var xhttp = new XMLHttpRequest();  
 xhttp.open("POST", '@Url.Action("WorkflowSearch", "Home")?title=' +  
 $('.workflow-live-search').select2('data')[0].text, true);  
 xhttp.send();  
 xhttp.onreadystatechange = function () {  
 if (xhttp.readyState == 4 && xhttp.status == 200 && xhttp.response != null) {  
 var url = '@Url.Action("Workflow", "Home")?id=' + xhttp.response;  
 window.location.href = url;  
 }  
 }  
 }  
 }

In the code above we create a new XMLHttpRequest with var xhttp = new XMLHttpRequest();

We direct the routing of this request using Razor syntax using a URLAction. The resulting statement is '@Url.Action("WorkflowSearch", "Home")’. This statement will direct the http request to the WorkflowSearch method within the Home controller. To complete this statement we add in parameters to the routing ?title=' + $('.workflow-live-search').select2('data')[0].text, true);

We can then send this message and wait for a response from the server. Once we receive a response, we can choose to respond in whatever way is appropriate. In this scenario, we would like to redirect to the webpage for the given workflow.

**7. Model Interaction**

All interaction with the model should be done through the controller. We have created an object to reference our database called ‘db’. Use this object to communicate with the modeling objects.

**8. Connecting to a database**

The project is currently connected to an Azure Web Server provided by the Flairdocs team. To change the database/server you will need to modify the connectionString found in the Web.config file. Changing this may result in the need to create a new database schema with all of the necessary tables. To create these tables you will need to run the WorkflowDesigner.edmx.sql file, and make sure the USE command is set to the appropriate database context.