# Table of Contents



This document provides technical documentation for the Student Grouping Tool application. The first part will cover the back-end, which was developed using the ASP MVC 4. The second part covers the front-end of the application, developed using JavaScript, HTML and CSS.

# Application Front-End

The front-end piece makes extensive use of jQuery for DOM manipulations and the Twitter Bootstrap front-end framework for most of the styling, as well as jQuery plug-ins for custom functionality that is not supported natively by browsers.

## Code Overview

The student grouping tool application consists of two modules:

**Group Selection module**

This module lists out all the groups that the user has access to. The user can edit groups individually or select a list of groups to edit.

**Multiple Groups Edit module**

This module allows the user to edit multiple groups simultaneously. It contains a 'workspace' that the user can add groups to. The user can add a new group to the workspace or select an existing group to add to the workspace. Once in the workspace, the group can be edited and changes can be saved back to the server.

Each of these modules contain widgets, which represent the UI components on the screen that the user interacts with. The widgets are written in JavaScript and are defined as follows:

```
widget = function (model) {
      // application logic
   this.model = model;
     // optional HTML template
   this.template = 'some HTML'
   // initialize the widget
   this.init = function(){
   }
     // returns HTML representation of this widget
   this.generateTemplate = function(){
      // manipulate template by adding/modifying DOM elements
      return this.template;
    }
}
```

A widget contains logic for handling user events. The application logic resides within the model object. Each model contains a reference to the server data and provides a facade for accessing this data. Additionally the model contains logic to keep track of state for the UI widgets.

## UI Styling

The Student Grouping Tool application leverages the Twitter Bootstrap framework for styling most of the components on the screen.

## Tooltips

Tooltips used for the UI to help guide the user are implemented using Twitter Bootstrap's tooltip plugin. These tooltips are defined in the tooltipText.js file under the Content/js directory. This file contains a JSON object, where each property key is the name of the css class to associate the tooltip with and the property's value is another JSON object that describes the placement of the tooltip and the message to display.

```
var tooltipText = {
   " .[css-class] " :
         {
         "placement" : " [left,right,top,bottom] " ,
          "message" : " [message to display] "
         }
}
```

### jQuery Plug-ins

- jquery-ui - provides the basic drag/drop functionality for the application
    - http://jqueryui.com/
- antiscroll - used to provide Mac OS X style scrollbars (alternative to the native browser scrollbar)
    - https://github.com/LearnBoost/antiscroll
- dateUtils - provides helper methods for working with date objects
    - https://github.com/JerrySievert/node-date-utils
- fileUpload - provides Ajax-style file uploading capabilities
    - http://blueimp.github.com/jQuery-File-Upload/
- jqplot - provides charting capabilities for displaying student data graphically
    - http://www.jqplot.com/

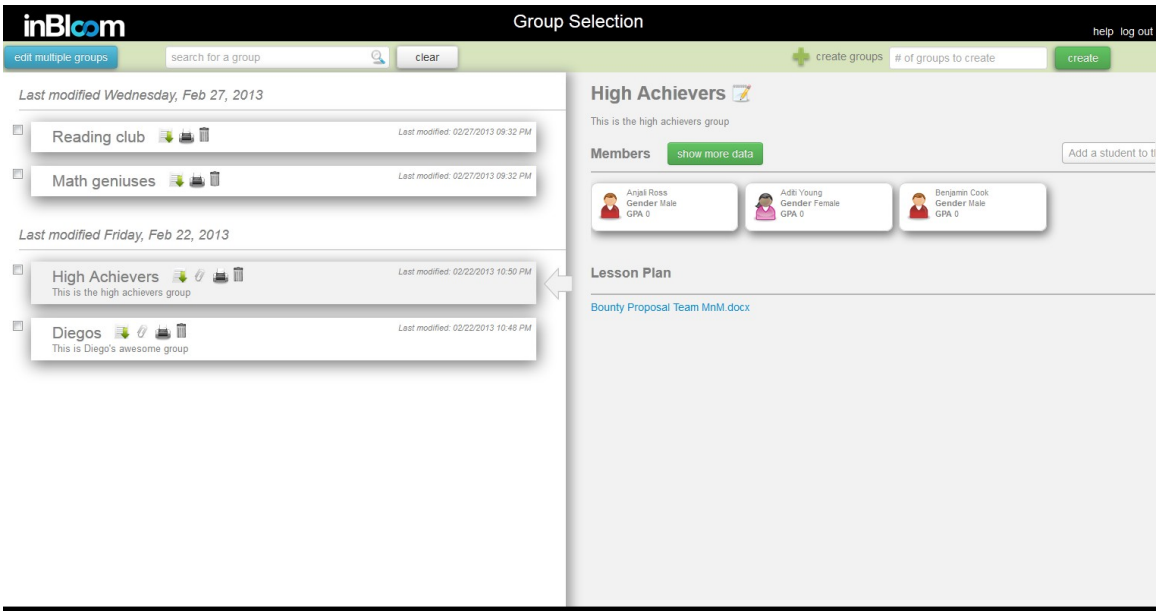
  idleTimer - used to detect idleness in order to disable the screen and prompt the user to re-authenticate after idle for a specified time period
    - http://paulirish.com/2009/jquery-idletimer-plugin/
- multidraggable - provides multi-dragging capabilities. Used for dragging multiple students into groups
    - http://www.myphpetc.com/2009/11/jquery-ui-multiple-draggable-plugin.html
- pubSub - provides a publish/subscribe notification system for communicating events between widgets and modules
    - https://github.com/federico-lox/pubsub.js
- select2 - provides a editable dropdown box for choosing between various options
    - http://ivaynberg.github.com/select2/
- spin - provides a loader animation to show the user that a request is being processed
    - http://fgnass.github.com/spin.js/
- underscore - provides utility functions for manipulating collections and arrays
    - http://underscorejs.org/

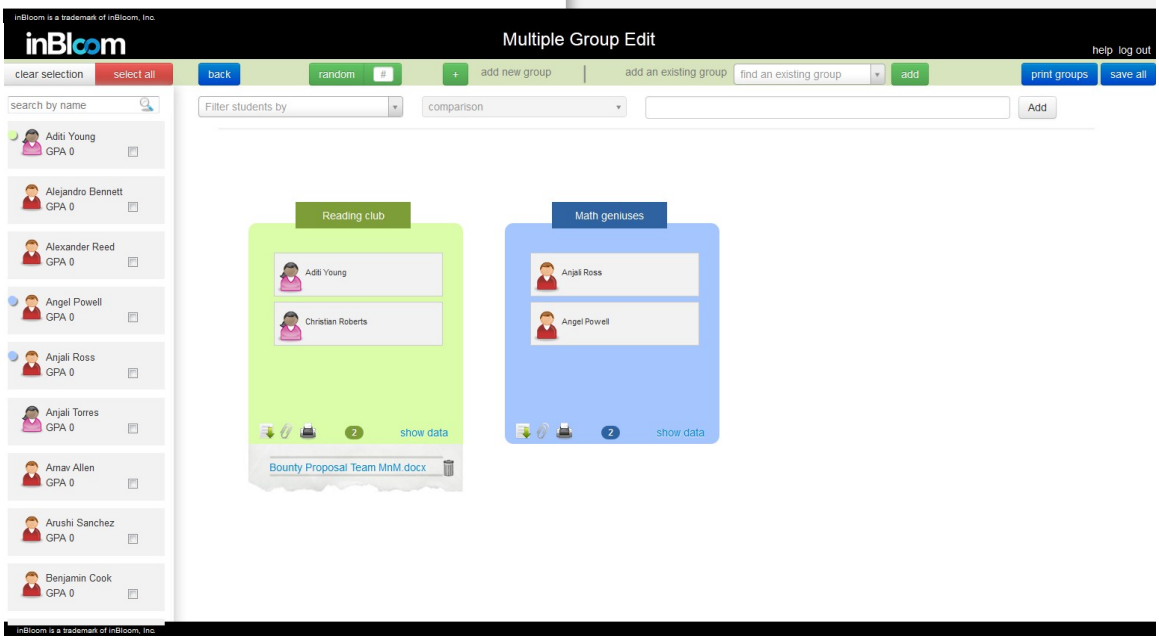
## Custom JavaScript code:

The application is split into widgets and models. The widgets display data to the user and handle user interactions. Each widget contains a model that handles the application logic. Application logic includes code for communicating with the server, such as saving to the backend or deleting the object from the backend._Models can be shared between different widgets.

### Widgets

The following screenshots map out all the different widgets we developed:




## Models

- GroupModel - contains logic for adding/removing students from a group, editing the name and description, attaching/removing lesson plan and associating student data elements. Also contains the code for saving the changes back to the server.
- GroupListModel - contains logic for maintaining a list of group models, such as adding/removing and retrieving by id.
- StudentModel - contains information about a student and acts a a facade for accessing the student's properties.
- StudentFiltersModel - contains logic for filtering a list of studentModels and maintaining a list of available and selected student data filters.

- SectionModel - sections are groups of cohorts with the same last modified date.

### Multiple Drag and Drop Support

We utilized the multidraggable.js plugin, which is built on jQuery UI's draggable API, to provide drag and drop of multiple students at once into a group. Objects that are multidraggable are given the 'multidraggable' css class and initialized via $(obj).multidraggable(). Objects that are droppable are given the 'droppable' css class and initialized via $(obj).droppable().

### Communication between widgets

Communication between widgets is achieved using the Mediator pattern through the jQuery plugin pubsub.js. Widgets subscribe to events of interest through the PubSub.subscribe(event-name, callback) and publish events for other widgets through PubSub.subscribe(event-name, args)

### File Upload

In order to provide a seamless user experience, we used the jquery-file-upload plugin to upload files asynchronously to the server. The way we use the plugin is that we hook onto the 'add' event, which is triggered when the user selects a file to upload. This event provides us the 'data' parameter, which we save. This parameter is later used to invoke the upload process by calling 'data.submit()'. Right before submitting we attach the group's id to the call using the formData via 'data.formData = { id : 'id' }'. The server then sends the file over to an FTP server. The FTP server credentials are stored under the web.config file.

### IE9 Incompatibility

Currently IE9 does not support multiple file uploads, therefore when using the **Save All** option on the MultipleGroupsEdit page, if there are more than one groups with lesson plans, only the first lesson plan will be uploaded and the rest of them will not be sent to the server. For more information on browser support please visit https://github.com/blueimp/jQuery-File-Upload/issues/1173.

### User Idleness

For security purposes, the web application will log the user out after 20 minutes of idleness. User idleness is detected using the idleTimer plugin.

## Application Back-end

The back-end application that handles all the Ajax calls from the front-end is the TeamMnMGroupWebApp. This piece is written using the ASP.NET MVC4 framework. The application follows the Model-View-Controller (MVC) pattern.

The BaseController contains references to the inBloom credentials and parameters from the web.config file. The HomeController is the main controller of the application. It

extends the BaseController and provides the initial Index() landing method along with all the AJAX methods for the UI to call.

The main method in the HomeController is the Group() method. This method retrieves all the cohorts, students, filters, student data elements, sections and group colors for the UI in one AJAX call so we can minimize the number of calls we make from the UI.

The **HomeController** also provides the AJAX method calls for adding/editing/removing cohorts and data associated to those cohorts. Each AJAX call should return a **Result** object, which contains the following properties:

- completedSuccessfully - whether ALL the different components of the cohort were successfully saved (e.g student associations, lesson plan, selected student data attributes, etc)
- objectActionResult - the result of saving the cohort entity
- partialCreateSuccess - whether ALL the given students were assigned successfully to the group
- partialDeleteSuccess - whether ALL the specified students were removed successfully from the group
- failToCreateAssociations - the ids of the students that were not successfully assigned to the group
- failToDeleteAssociations - the ids of the students that were not successfully removed from the group
- customActionResult - the result of saving custom data related to a group, such as lesson plan and the associated student data attributes

## Authentication

Additionally the HomeController handles the login and authentication. Once the application starts, it will direct the user to inBloom's authentication page. The user will enter his/her credentials, and once authenticated, the web application will store the access_token in the session. The token ID will be passed to the .NET wrapper in order to make the RESTful API calls to retrieve and save data.

## Caching

Caching is used throughout the Student Grouping Tool to reduce the number of API calls and provide the user with a more responsive user interface. Caching is done using the HttpContext.Current.Cache, which caches objects in memory across all sessions. The objects that are currently being cached are all the DisplayObjects:

- CohortDisplayObject - contains the cohort entity along with the students assigned to the cohort and the custom object associated with the cohort
- SectionDisplayObject - contains the details of a section: id, courseTitle, courseDescription, subjectArea, courseLevel
- StudentDisplayObject - contains the details about a student, including all the available student data attributes
- GroupingDisplayObject - consolidated display object containing the list of all cohorts, students, sections, filters, colors, and dataElements.

These objects are passed to the UI through the AJAX calls. Every time there is an update to a cohort, that cohort will be removed from the cache.

### Filters

The filters for filtering the student list on the Multiple Group Edit screen are configured via the **FilterHelper**. This helper class provides static methods for retrieving the filters related to the student data such as GPA, Gender, Disabilities, etc. Each filter has the following properties:

- attributeId : matches the name of the student data to filter on
- attributeName: the text to display for the filter
- operators: the type of comparison to use for the filter operation
  values: the selectable values to compare to when filtering

You can add, remove or edit the student filters for the Multiple Group Edit screen by modifying the FilterHelper.cs class. This static helper class contains references to all the different inBloom types, which are defined as enums in the inBloom Client library, as well as all the comparison operators that can be used for each type (e.g =, >, <).
The InitializeFilters() method in FilterHelper.cs returns a list of Filter objects, which are used in the UI to allow the user to filter through the list of students. Each Filter consists of the following properties:

- name of the student data (must match the field on the Student entity)
- title of the filter
- list of operators for comparison
- list of values that the user can select from (i.e. the different grade levels, disabilities, etc)

### Group Colors

The colors used to distinguish between different groups in the Multiple Group Edits page are defined in the **Colors.txt** file under the root/Data directory. The **GlobalHelper.cs** class provides static helper methods for retrieving a list of colors to pass to the UI.

To add, remove or modify the group colors used to represent the different cohorts that the user has added to the workspace in the Multiple Group Edit screen, refer to Data/Colors.txt. This file contains the list of colors. Each color has two properties:

- **background** : defines the background color of the cohort object on the screen
- **title**: defines the background color of the cohort's title div

## inBloom Sandbox Account

### Credentials and Configuration

The credentials and authentication endpoints for accessing the inBloom APIs are defined in the web.config file inside the TeamMnMGroupingWebApp project. The settings are:

- InBloomClientId – unique id assigned by inBloom for accessing the APIs

- InBloomSharedSecret – secret code assigned by inBloom for accessing the APIs. To be used with the clientId
- InBloomRedirectUrl – the url to redirect to upon successful login
- InBloomSandboxLogin – the oauth authorization URL
- InBloomOAuthUrl – the oauth token URL

The web.release.config file contains the credentials to use on the production environment. When publishing the application the user can choose which environment to use as shown in the figure below:

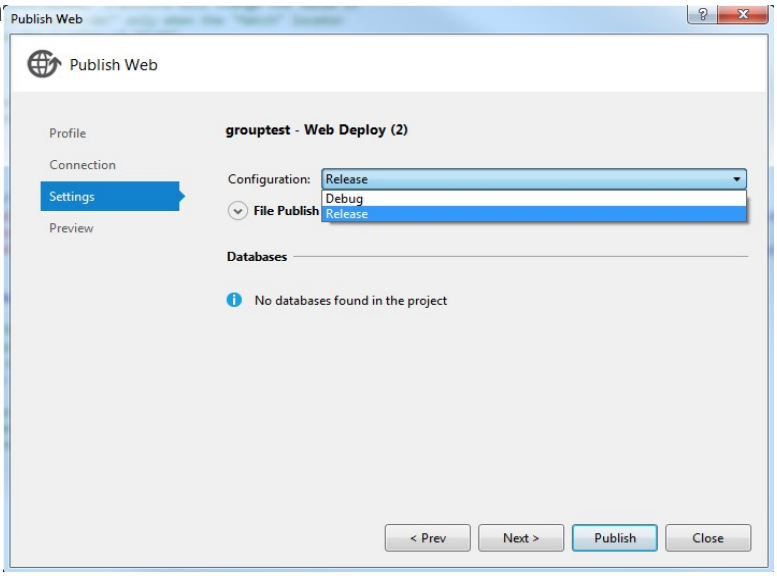



Figure - Web.config deploy

The publish process will use the original web.config file regardless of which configuration file is chosen. However, it will copy the settings from the select configuration file (i.e. web.release.config) and replace those settings in the web.config file. This allows us to use different login credentials for different environments (i.e. dev vs uat vs prod).

We show the configuration for 2 different environments above but you can configure as many environments as needed by using the Configuration Manager in **Build -> Configuration Manager…** as shown in Figure 2. Specify the appropriate credentials in the new config file. When publishing remember to choose the right publish environment.

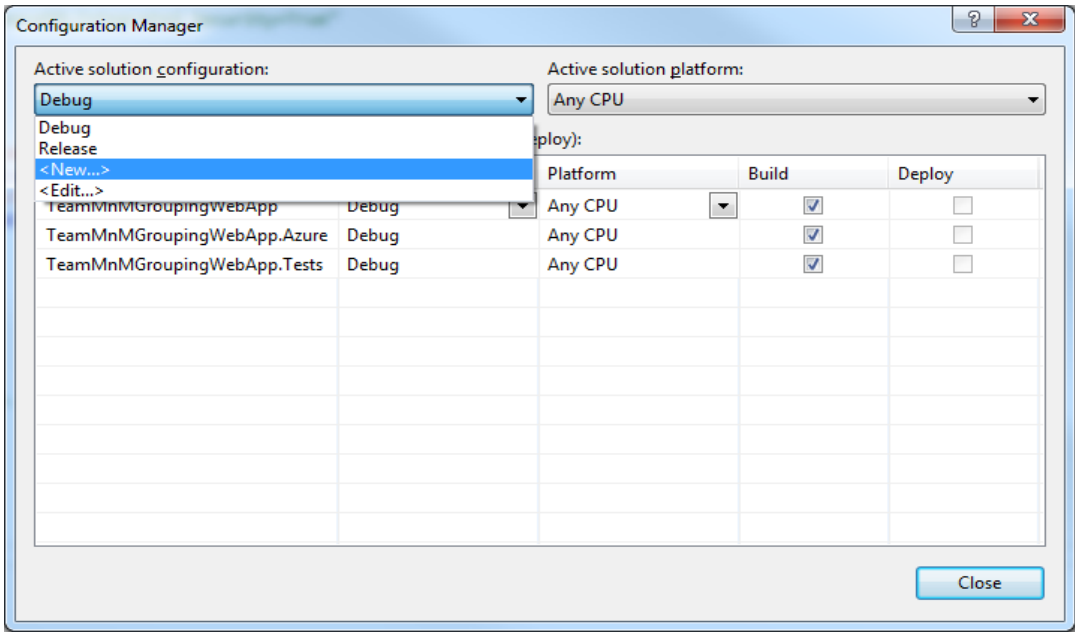



**Figure - Web.Config manager**

### Educational Organization ID

The Educational Organization ID (EdOrgId) is required to create cohorts. Because we are using the sample data set, there is no EdOrgId associated with the existing users, therefore we hard-code this ID for the user Amy Kopel (akopel) in the web.config file. However, the code for retrieving the EdOrgId associated with the current authenticated user has already been implemented. If the code cannot find the associated EdOrgId, then it will fall back to using the hard-coded one for akopel.

## inBloom Client (.NET Wrapper)

The web application communicates with the inBloom Data Stores via the inBloom Client project. The inBloom Client project serves as the .NET wrapper for accessing the inBloom REST APIs. It contains classes that represent the different inBloom entities and associations, as well as services for communicating with the inBloom APIs. This project retrieves the data from the inBloom Data Store, parses the results and maps them into the C# wrapper objects that are defined in this project.

The inBloom Client project contains an app.config file that defines the following URLs for accessing the inBloom API endpoints:

- InBloomSandboxUrl – root inBloom sandbox URL
- InBloomApiSandboxUrl – inBloom API Sandbox URL used to make the API calls agains
- InBloomApiSandboxSessionUrl – URL used to retrieve the user info, such as the user id, whether the user is authenticated, and the user's roles.

To utilize this .NET wrapper in a .NET project, just add a reference to this library and utilize the service classes under the services directory to retrieve/modify data from the inBloom Data Stores. An access token, which is provided by the inBloom OAuth mechanism upon successful login, is needed to make the API calls.

## Structure

The inBloom Client project is split into the following sections (folders within the source code):

- Constants - contains the hardcoded URIs for the inBloom Data Store entities and associations, categorized into different classes representing different entities (e.g Cohort.cs, Student.cs). New API URIs should be added to this section.
- Entities - contains C# classes for all the inBloom Data Store entities. New entities should be added into this section.
- Enums - contains C# classes for all the inBloom Data Store static types (e.g CohortType, DisabilityType).
- Services - contains code for communicating with the inBloom APIs and mapping the objects returned from the APIs into the entities we have defined in the inBloom Client project. New API calls should be added to this section

## JSON.NET

The JSON.NET library is used to serialize/de-serialize data retrieved from the inBloom. This allows us to easily map inBloom entities to the C# entities defined in the inBloom Client library.

Each Enum has two attributes associated with it: **Description** and **EnumMember**. Each attribute contains the actual Enumeration string value from inBloom. Even though they share the same string value, they serve different purposes. **Description** is used by the web application to get the string value of the enum to construct student data attribute filters (for more details refer to the **Filters** section). **EnumMember** is used by the JSON.NET serializer/deserializer to parse the string Enum value between this wrapper and the inBloom Data Store.

**ShouldSerialize** + *[property name]* is the syntax used to tell JSON.NET whether it should serialize a property based on a specified condition (e.g. whether a value is null or not).
For more information visit http://james.newtonking.com/projects/json-net.aspx.

## ELMAH

All exceptions thrown by the application are logged using the elmah library. The error logs can be accessed through the path root/elmah.axd. For security purposes remote access has been disabled, meaning that the logs can only be accessed locally (i.e. localhost) or through FTP by manually downloading the log files.

The elmah configuration can be found in the web.config file. You can specify which directory to save the log files to, as well as the path to access the elmah error logs screen. For more information on elmah please visit http://code.google.com/p/elmah/wiki/DotNetSlackersArticle.

If the web application is hosted on the cloud, such as Windows Azure, then instead of accessing the logs locally, the logs can be downloaded and consolidated into reports using the elmah Log Download tool at http://code.google.com/p/elmah/wiki/ErrorLogDownloadApplications.

## Nuget Package Manager

The Student Grouping Tool's .NET external dependencies (third-party libraries and tools) are handled by the **Nuget Package Manager**. When the source code is downloaded and opened in Visual Studio for the first time, these required dependencies might not be installed in the system. To install the necessary dependencies right-click on the project solution and click on **Manage Nuget Packages for Solution**. This will bring up **the Nuget Package Manager**. Select the required dependencies and click on Install to install them.

The following is a list of all the project's .NET dependencies:

- ELMAH – used for error-logging
- Json.NET – used for serializing JSON objects into C# objects
- HTTP Client Libraries – used to make HTTP requests to REST endpoints
- ASP NET Web Pages 2 – core runtime required for ASP websites
- ASP NET Web Infrastructure – runtime libraries required for ASP MVC websites
- ASP NET MVC 4 – main runtime assemblies for ASP MVC websites
- ASP NET Razor 2 – runtime assemblies for the Razor HTML rendering engine
- ASP NET Web API – package for building HTTP services