

## Table of Contents

The inBloom Data Store Cookbook .....	4
Preface .....	4
What is in this guide.....	4
Platform notes .....	4
API versions .....	4
Installation instructions .....	4
Conventions used in this book .....	4
Set Up the inBloom Sandbox .....	5
Administer the Sandbox.....	5
Load sample data .....	6
Register an Application .....	7
The Hello Worlds.....	10
Installation Instructions .....	10
JavaScript Instructions .....	10
C# Instructions .....	12
JavaScript.....	13
Configuration File (config.js).....	14
Main Application File (app.js) .....	15
Controller Files (home.js, students.js) .....	18
View Files (layout.jade, home.jade, students.jade) .....	20
Ruby .....	20
Configuration Files .....	20
Sessions Controller .....	23
Hello Controller .....	23
Index View .....	24
C# .....	24
Authorization .....	25
Recipe Controller .....	25
The Recipes.....	27
1. Retrieving a user's unique id.....	27

Problem.....	27
Discussion.....	27
Solution .....	27
2. Determine user's educational organizations ids.....	29
Problem.....	29
Discussion.....	30
Solution .....	30
3. Create a Staff Member .....	33
Problem.....	33
Discussion.....	33
Solution .....	34
4. Associate staff with an educational organization.....	36
Problem.....	36
Discussion.....	37
Solution .....	37
5. Create an Assessment.....	40
Problem.....	40
Discussion.....	40
Solution .....	41
6. Update an Assessment.....	43
Problem.....	43
Discussion.....	43
Solution .....	44
7. Delete an Assessment.....	47
Problem.....	47
Discussion.....	47
Solution .....	48
8. Enroll a Student in a Section .....	50
Problem.....	50
Discussion.....	50
Solution .....	51
9. Find a Student by Criteria .....	54



Problem.....	54
Discussion.....	54
Solution .....	54
10. Update a Student’s Profile .....	56
Problem.....	56
Discussion.....	56
Solution .....	57
11. Assign a Grade to a Student .....	59
Problem.....	59
Discussion.....	59
Solution .....	60
12. List Parent Contact Information for an Entire Class .....	62
Problem.....	62
Discussion.....	62
Solution .....	64
Further Information .....	66
InBloom docs .....	66
Forums.....	66



# The inBloom Data Store Cookbook

## Preface

This cookbook is not meant to be a complete reference guide to inBloom's Data Store. That reference already exists in the [inBloom developer documentation](#).

This cookbook is not an introduction to programming in JavaScript, Ruby, or C#. Nor is it intended to be an introduction to working with a RESTful API.

Instead, this is a cookbook for learning the ins and outs of the inBloom Data Store. It's for people who already know their way around programming, but want to create meaningful interactions with the inBloom Data Store.

Like any other cookbook, this cookbook contains recipes. We tried hard to make the included recipes useful when read sequentially or accessed individually.

## What is in this guide

This cookbook provides examples for accessing the inBloom Data Store using three different languages: JavaScript, Ruby, and C#. All of the code samples are provided in these three languages.

The guide is divided into two main parts: "Hello Worlds" and "recipes". The Hello World examples provide the minimal code required to access the inBloom Data Store. The recipes aren't entirely self contained, they are meant to be used in conjunction with a "Hello World".

## Platform notes

### API versions

This book was developed using the inBloom API v1.2.

### Installation instructions

The installation instructions needed to run each of the Hello Worlds are provided at the end of the Hello World section.

## Conventions used in this book

resource - any meaningful concept around which a user interaction can occur. All resources share three common fields: id, entityType and metaData.

See [Perl Cookbook version](#)

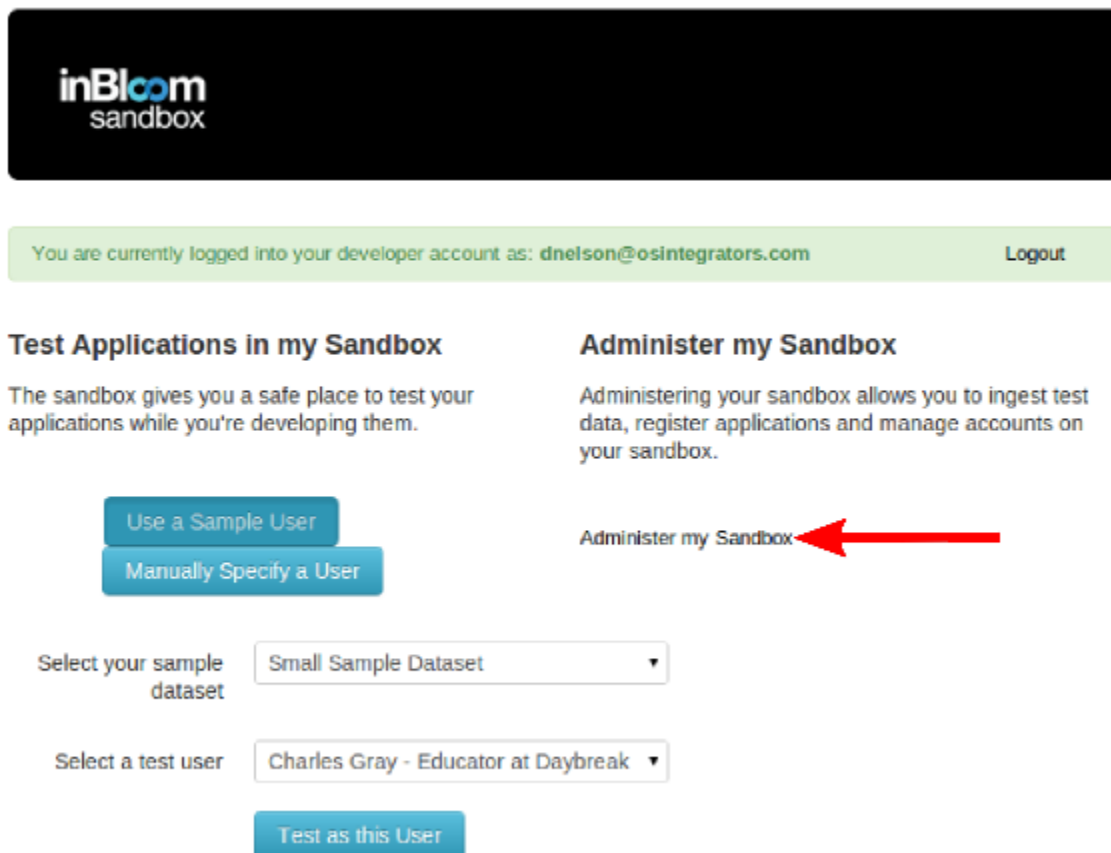


# Set Up the inBloom Sandbox

The sandbox is a simulation of a production environment. It's an independent instance of the inBloom Data Store. When you first create a sandbox, not only is there no data in it, but no applications have the right to authenticate with it. Let's fix that.

## Administer the Sandbox

First go to the [inBloom sandbox login portal](#). Login using the email and password you used to [set up your developer account](#)~You'll reach the Sandbox's main control screen. From here, go to the Administration panel, by clicking on "Administer my Sandbox". Note the convenient red arrow on the screenshot below.



The screenshot shows the inBloom sandbox interface. At the top, there's a black header with the 'inBloom sandbox' logo. Below it, a green bar indicates the user is logged in as 'dnelson@osintegrators.com' with a 'Logout' link. The main content area is divided into two columns. The left column is titled 'Test Applications in my Sandbox' and contains a description, two buttons ('Use a Sample User' and 'Manually Specify a User'), and two dropdown menus for selecting a sample dataset and a test user. The right column is titled 'Administer my Sandbox' and contains a description. A red arrow points to the 'Administer my Sandbox' link in the right column.

**inBloom**  
sandbox

You are currently logged into your developer account as: [dnelson@osintegrators.com](#) [Logout](#)

**Test Applications in my Sandbox**

The sandbox gives you a safe place to test your applications while you're developing them.

[Use a Sample User](#)  
[Manually Specify a User](#)

Select your sample dataset

Select a test user

[Test as this User](#)

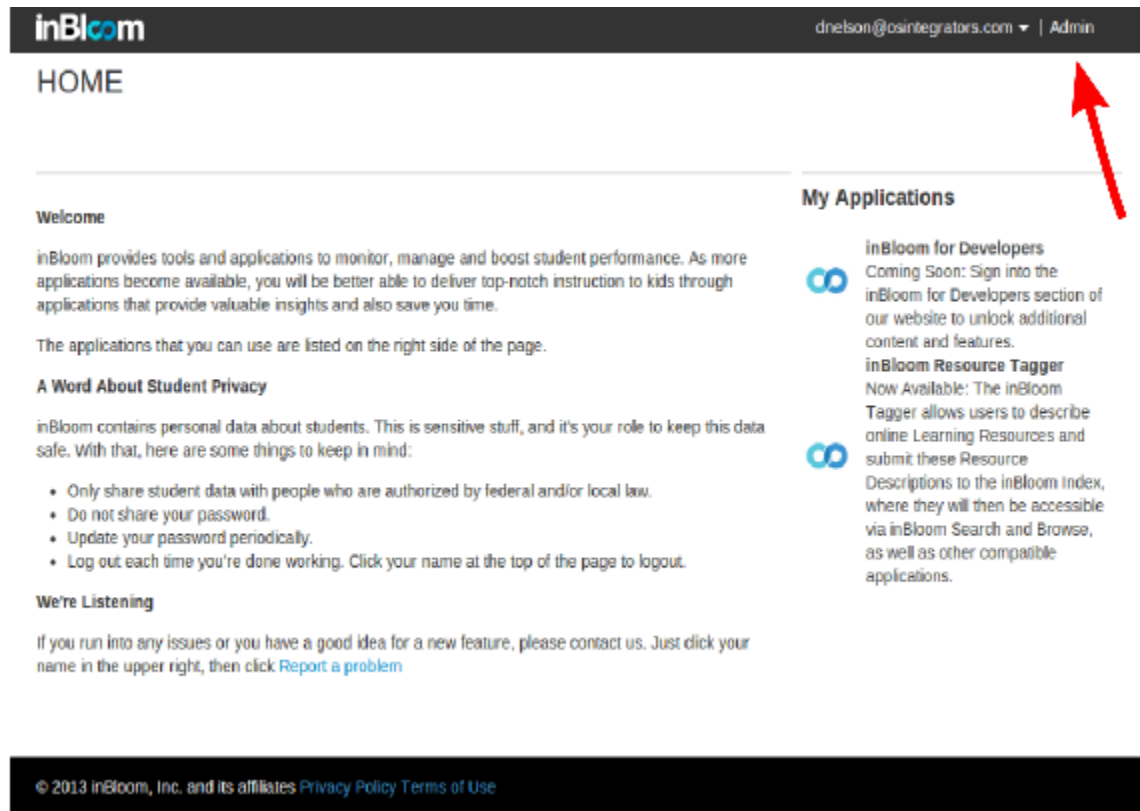
**Administer my Sandbox**

Administering your sandbox allows you to ingest test data, register applications and manage accounts on your sandbox.

[Administer my Sandbox](#)

Read and agree to the terms and conditions. Now you'll be at the inBloom home screen for your user account, but you still have to get to the admin panel. There's a link on the top left of the page called "Admin". Click it.





Now we're at the [admin panel](#). The section that we're going to be working with is labeled system tools on the lower left.

## Load sample data

We need to populate the Sandbox some sample data. In the inBloom documentation this process is sometimes referred to as ingestion. Click on 'Create Landing Zone'.

### System Tools

- [Register Application](#)
- [Create Custom Roles](#)
- [Create Landing Zone](#)
- [Change Password](#)
- [Manage Developer Accounts](#)

inBloom provides two fictional data sets that are ready to go. A small, generally more detailed dataset and a medium dataset has more users, students and sections, but is a bit lacking in some details. In most instances, the small dataset will be sufficient for development.



Click on 'Autoload inBloom sample data'. The 'Select' button should become clickable and the radio button should darken. The small data set should already be selected in the dropdown menu. Click 'Select' to load the small data set. This may take a few minutes, but the process will be completely in the background. In the meantime, you can continue setting up the sandbox.

It's worth noting that you can create your own data and ingest it into the sandbox, but [doing so is outside of the scope of this document](#).


## Register an Application

Now we have some data in the Sandbox, but we have not set up a path for applications to access that data. So next we're going to do that by registering an application. From the [admin panel](#), click on 'Register Application'.

### System Tools

- [Register Application](#)
- [Create Custom Roles](#)
- [Create Landing Zone](#)
- [Change Password](#)
- [Manage Developer Accounts](#)

We are now at the application manager. We will need to create a new application by clicking 'New Application'.



# Manage Applications

Register a new application and update applications you have already registered.

New Application

Name	Version	Vendor	Creation Date
------	---------	--------	---------------

The new application page is a form, with several fields. Below, we will go through the fields that must be filled out. The first three fields are mostly for your own use.

- Name: A convenient name for your application. Something like "Hello World" will work just fine.



- Description: A concise description of your application. I'd recommend keeping it short. Again "Hello World" will do.
- Version: Numerical version number. I normally use 1.

The next two fields are the URL and Redirect URL. These are used by inBloom.

- Url: This is the Url that inBloom will link to when referencing your application. In the case of the ruby hello world, this will be `http://localhost:3000/`
- Redirect Url: This is the Url to which inBloom will send the authorization code and access token during the authentication process. For demonstration purposes, the Redirect Url will not be the same as the Url to avoid confusion.

Scroll down to the bottom and click register. You will be sent back out to the application manager and your application will now be displayed, as below:

inBloom

drewnelson112@gmail.com ▾ | Admin

## Manage Applications

Register a new application and update applications you have already registered.

New Application

Name	Version	Vendor	Creation Date	Last Update	
Hello World	1	None	Thu, 25 Apr 2013 19:38:57 +0000	Thu, 25 Apr 2013 19:38:57 +0000	<div>In Progress</div> <div>Delete</div>

There are still a few more steps. You probably noticed the yellow In Progress button. You are currently in the Edit Application page. It is very similar to the New Application page, with one big difference. Scroll down to the bottom and you should see the following interface:

## Enable the districts you would like to use your application

Select a state

Please select a state ▾

Please select a state first...

Save

Cancel

Click on the 'Please select a state' drop down. Select the 'Standard State Education Agency' from the drop down. Click the checkbox next to 'Daybreak School District 4529'. Finally, click save to complete the application registration process.





If there are no options to select, the sample dataset selected earlier may not be done ingesting. If this is the case, wait until the datastore sends you a data ingestion confirmation and then return to the Manage Applications window, and click on 'In Progress' again.

**inBloom**drewnelson112@gmail.com ▾ | Admin

App was successfully updated.

## Manage Applications

Register a new application and update applications you have already registered.

New Application

Name	Version	Vendor	Creation Date	Last Update	
Hello World	1	None	Thu, 25 Apr 2013 19:38:57 +0000	Wed, 01 May 2013 15:11:27 +0000	<div>EditDelete</div>

We are now at the Manage Applications page. Click on 'Hello World' or the name of the application you just finished registering. A panel of information will appear below the application information bar. The first two entries, 'Client Id' and 'Shared Secret', have been supplied by inBloom and are very important in our application's authentication process with the inBloom servers. The client id and shared secret will be placed in a configuration file in our application. The shared secret should be kept confidential.



# The Hello Worlds

The 'Hello World' application is designed for the purpose of showing how to:

1. Authenticate with inBloom using [OAuth2 authentication](#)
2. Once authentication is successful, perform a call to receive data from the inBloom Data Store.

These are two important functions of any web application that uses inBloom's Rest API to perform operations for an end user. The Hello World applications have been designed to simply allow developers to add code to the existing structure in order to add the desired functionality and to have an application that essentially performs authentication with inBloom 'out of the box'. This document contains [twelve example recipes](#) for performing calls (GET, POST, PUT, DEL) to the inBloom Rest API to receive requested data. Each recipe can be integrated into the Hello World structure to perform the desired effect of the recipe.

The Hello World chapter is divided into two main sections:

1. Code samples for the Hello World in three different programming languages
  - a. JavaScript
  - b. Ruby
  - c. C#
2. Installation instructions
  - a. JavaScript
  - b. Ruby
  - c. C#

## Installation Instructions

### JavaScript Instructions

You will need the following installed:

- [Node.js](#)
- [Git Version Control System](#)

#### [Get the code](#)

Download the code via git:

```
git clone git@github.com:inbloom/hello-world-javascript.git...
```



Change to the application's home directory:

```
cd hello-world-javascript
```

All local Node.js modules are included in the cloned repository at the path `./node_modules`. These modules are:

- [Express](#): web application framework for Node.js
- [Restify](#): used to create client for accessing inBloom Rest API
- [Request](#): used by SLC.js to simplify HTTP requests
- [Jade](#): Node.js template engine used for creating dynamic html
- [Stylus](#): Node.js template engine used for creating css

JS Hello World uses [inBloom's JavaScript SDK](#) (SLC.js) to perform OAuth2 authentication with inBloom's authentication server.

## Ruby Instructions

### What you'll need

You will need the following installed:

```
ruby ( rvm is recommended )
```

```
rubygems
```

```
git
```

### Get the code

Download the code via git:

```
git clone git@bitbucket.org:osintegrators/inbloom-hello-world-ruby-on-rails.git
```

Change to the app directory:

```
cd inbloom-hello-world-ruby-rails
```



Install the gems with bundle:

```
bundle install
```

Let's look at what was just installed. Open the Gemfile in your text editor of choice. This is the default Rails 3.2.11 Gemfile with a few additions, namely:

```
gem 'omniauth-inbloom'  
gem 'rest-client'  
gem 'json'
```

The omniauth-inBloom gem is an OmniAuth strategy for authenticating with inBloom. It will handle the headache that OAuth2 can provide. It takes a little bit of configuration to get up and running. (See the Configuration Files section in the code sample above.)

The rest-client gem is great tool for interacting with REST apis of all sorts. However, if you're already familiar with HTTParty, that will do the job too.

The json gem provides functionality for creating and parsing JSON strings.

## C# Instructions

### What you'll need

Naturally you'll need Visual Studio. This project is based on MVC4 which is [available from Microsoft](#). Additionally you'll need several libraries. The easiest way to handle their installation and management is using [NuGet](#). You can configure NuGet to automatically find and install the required packages, but I'll leave that for the reader to explore.

In the event you are using NuGet to manage packages the required package names are in the list below.

- Microsoft.Web.Infrastructure
- Entity Framework
- Microsoft ASP.NET Web Optimization Framework



- Newtonsoft.Json (Json.net)
- RestSharp
- Input and runtime validation
- DotNetOpenAuth extensions for ASP.NET

You'll also need some version of Git for windows to clone the source from github. Either the [Microsoft's Visual Studio Tools for Git](#) or [Git for Windows](#). Directions for using Microsoft's git client are outside of the scope of these instructions, so from here on out it is assumed that you installed Git for Windows.

Get the code

Open up the Git for Windows application.

Navigate to where you wish to download the repository

Download the code via git:

```
git clone git@bitbucket.org:osintegrators/inbloom-c-sharp-hello-world.git
```

Rename the web.config.example file web.config

Open Visual Studio.

Open the hello world project.

## JavaScript

The JavaScript (JS) Hello World uses Node.js to perform server-side application processes using the JavaScript scripting language. This is not a tutorial for learning Node.js. Visit the [Node.js website](#) for information on learning and using Node.js.



The JS Hello World also uses several Node.js modules. These modules are Express, Request, Restify, Jade, and Stylus. For a brief description and links to the module's website, see [Installation Instructions for JS](#).

### Configuration File (config.js)

In order for the JS Hello World to work, there are a couple values that must be inserted into the configuration file, config.js. These are:

- Client Id
- Client Secret
- Callback URI
- Port number
- API version

When [registering the application](#), inBloom assigns every application two values, Client Id and Client Secret. These are used to authenticate with inBloom. Client Id and Client Secret will need to be assigned to client\_id and client\_secret variables, respectively (string values) in the configuration file (config.js.example).

During the application registration process, a 'Redirect Url' was entered for the application. The value assigned to the callback\_uri variable (string value) in the configuration file must match the 'Redirect Url' value of the application registration.

The port number in the 'Redirect Url' must also be assigned to the port variable (number value) to ensure the Express server is listening on the correct port.

The inBloom API version intended for use by the application should be specified in the configuration file. Here we use version "1.2". For more information on inBloom API versioning, consult the [inBloom Developer Documentation](#).

```
/**
 * config.js.example
 * Configuration file for inBloom API and OAuth2 information.
 *
 * Follow these steps to setup the configuration file
```



```

* 1) Rename this file to config.js
* 2) Enter the appropriate values for the following variables in this configuration file:
*     a) config.api.client_id
*     b) config.api.client_secret
*     c) config.api.callback_uri
*     d) config.app.port
* 3) Ensure the appropriate values is assigned to the following variables in this
configuration file:
*     a) config.api.api_version
*/

var config = {};

config.app = {};

// This port number (number value) must be the same port number registered in redirect url
of application's inBloom Sandbox
config.app.port = YOUR_PORT_NUMBER_HERE;

config.api = {};

// Replace value with application client id (string value), given by inBloom
config.api.client_id = 'YOUR_CLIENT_ID_HERE';

// Replace value with application client secret (string value), given by inBloom
config.api.client_secret = 'YOUR_CLIENT_SECRET_HERE';

// This callback uri (string value) must be the same as the callback uri registered in
application's inBloom Sandbox.
config.api.callback_uri = 'YOUR_CALLBACK_URI_HERE';

config.api.api_version = 'v1.2';
config.api.api_call_append = '/api/rest/';
config.api.base_url = 'https://api.sandbox.inbloom.org';
config.api.authorization_endpoint = '/api/oauth/authorize';
config.api.token_endpoint = '/api/oauth/token';

module.exports = config;

```

## Main Application File (app.js)

The app.js file is the main file for the JS Hello World. At the top of the file, the node modules and application files are imported for use. This is a good location to import other files (such as individual recipes).

```

/**
 * app.js
 * Main application setup.
 */

// Modules required for application

```



```

var express = require('express')
, restify = require('restify')
, stylus = require('stylus')
, http = require('http')
, path = require('path')
, url = require('url')
, util = require('util');

// Application files
var config = require('./config')
, SLC = require('./SLC')
, home = require('./routes/home')
, students = require('./routes/students');

```

An Express application is created. This ‘app’ obj is used to create the HTTP server and make GET and POST requests, based on received URL and routing table.

```

var app = express();

```

An instance of SLC is created to use for authentication with inBloom. The SLC.js file and additional examples of how it can be used are located at [inBloom’s JavaScript SDK](#). This instance is used to request an access token from inBloom. This access token is then passed with each request made to inBloom’s Rest API to verify authorization.

```

// Creating a instance of SLC for authentication
var inbloom = new SLC(config.api.base_url,
                      config.api.client_id,
                      config.api.client_secret,
                      config.api.callback_uri,
                      config.api.api_version);

```

A Restify JSON client is created. The configuration of the ‘client’ object is essential for successful inBloom Rest API requests. This ‘client’ object is passed to the different controllers, where the inBloom Rest API requests are made

```

// Create a JSON client using Restify module
var client = restify.createJsonClient({
  url: 'https://api.sandbox.inbloom.org',
  headers: { accept: 'application/vnd.slc+json',
              authorization: 'bearer ',
              'content-type': 'application/vnd.slc+json'
            },
  version: '*'
});

var oauth_path = url.parse(config.api.callback_uri)['pathname'];

```





```

// If the access token is active, it will allow next route handler in line to handle
// the request, if the token is expired, then it will redirect to login page
function requireToken() {
    return function(req, res, next) {
        if (req.session.accessToken) {
            next();
        }
        else {
            res.redirect('/home');
        }
    }
}

//-----Routing Table-----//

/**
 * Redirect the user to the OAuth 2.0 provider (inBloom) for authentication.
 * When complete, inBloom will redirect the user back to the
 * redirect url specified in the configuration file
 */
app.get('/auth/inbloom', function(req, res) {
    res.redirect(inbloom.getLoginURL());
});

/**
 * The OAuth 2.0 provider (inBloom) has redirected the user back to the application.
 * Finish the authentication process by attempting to obtain an access
 * token. If authorization was granted, the user will be logged in.
 * Otherwise, authentication has failed.
 */
app.get(oauth_path, function(req, res) {
    inbloom.oauth({code: req['query']['code']}, function(accessToken) {
        req.session.accessToken = accessToken;
        client['headers']['authorization'] = 'bearer ' + accessToken;
        res.redirect('/students');
    })
});

app.get('/', home.start);
app.get('/home', home.start);
app.get('/students', students.getStudents(client));
app.get('/students/:studentId', students.getStudentById(client));

// Logout and redirect to user login page.
app.get('/logout', function (req, res) {
    req.session.destroy();
    inbloom.logout(function (response) {
        util.puts("logout "+response.msg);
        res.redirect('/');
    });
});

//-----End of Routing Table-----//

```



```
http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

## Controller Files (home.js, students.js)

home.js is used to render a home page. If the user is not logged in, a link to log in is present. If the user is currently logged in, a link to logout is present.

```
/**
 * routes/home.js
 * Handler for home page requests.
 */
exports.start = function(req, res){
  var tokenExists = req['session']['accessToken'] ? true : false;

  if (tokenExists) {
    res.render('logout', { tab_name: 'Home - Logout',
                          title: 'InBloom JavaScript Hello World' }
    );
  } else {
    res.render('home', { tab_name: 'Home - Login',
                         title: 'InBloom JavaScript Hello World' }
    );
  }
};
```

students.js is used to perform a GET call for a list of all student and their associated information to the inBloom Rest API and render the student data as a web page. Notice that the configuration file was imported at the beginning of the file in order to use the api path variables in the config.js file.

In the 'getStudents' function, the 'client' Restify object is passed into the function from the app.js file. The 'client' object is used to make a GET call to the <https://api.sandbox.inbloom.org/>

[api/rest/v1.2/students](#) endpoint, assigned to the 'path' variable. The data returned by inBloom Rest API is assigned to the 'obj' object. The 'obj' object is then stringified and rendered to 'APP\_BASE\_URL/students'.

```
/**
 * routes/students.js
 * Handler for home page requests.
```



```

*/
var config = require('../config');

// Retrieving all student information.
exports.getStudents = function (client){
  return function(req, res, next){

    // Create path to 'students' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    '/students';

    client.get(path, function (err, req2, res2, obj) {

      var json_string = JSON.stringify(obj, 'false', 2);

      res.render('students', { tab_name: 'Students',
                              title: 'All Students',
                              data: json_string }

      );
    });
  };
};

```

The 'getStudent' function allows for the user to place an individual student's id into the applications url and receive associated student's information, rendered to 'APP\_BASE\_URL/students/{id}', where {id} is the student's id.

```

// Retrieving student information by student ID.
exports.getStudentById = function (client){
  return function(req, res){

    var studentId = req.params.studentId;

    // Create path to users 'home' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    '/students/' + studentId;

    // GET request to api using Restify module .get function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.get(path, function (err, req2, res2, studentJSON) {

      var studentJSON_string = JSON.stringify(studentJSON, 'false', 2);

      // Prints userUniqueID to webpage for verification
      res.render('students', { tab_name: 'Student',
                              title: 'Single Student',
                              data: studentJSON_string }

      );
    });
  };
};

```



```
};  
};
```

## View Files (layout.jade, home.jade, students.jade)

These files are used by Jade, the default Node.js template engine, to create dynamic html.

```
// layout.jade  
// Create header html page using Jade, a Node.js template engine  
doctype 5  
html  
  head  
    title= tab_name  
    link(rel='stylesheet', href='/stylesheets/style.css')  
  body  
    block content
```

```
// home.jade  
// Create home html page using Jade, a Node.js template engine  
extends layout  
  
block content  
  h1= title  
  p Welcome to #{title}  
  a(href='/auth/inbloom') Log In with inBloom
```

## Ruby

For an in depth discussion of Ruby on Rails' configuration and initializer files read the [Ruby on Rails Guide: Configuring Rails Applications](#). In short all this stuff happens before the webserver will serve any pages.

## Configuration Files

### config/initializers/session\_store.rb

```
RubyHello::Application.config.session_store :cookie_store,  
  key: '_ruby-hello-session', expire_after: 30.minutes
```



The addition of the `expire_after` value is a small but valuable change from the default ruby session store initializer. This helps make sure that your application won't think it has a valid inBloom token while the inBloom Data Store has expired that token long ago.

### **config/inbloom.example.yml**

```
development:
  client_id: 12345
  client_secret: abcdef
  redirect_uri: http://localhost:3000
  site: https://api.sandbox.inbloom.org/
  rest_url: api/rest/v1.2/

test:
  client_id: 23456
  client_secret: bcdefg
  redirect_uri: http://test-server
  site: https://api.sandbox.inbloom.org/
  rest_url: api/rest/v1.2/

production:
  client_id: 34567
  client_secret: cdefgh
  redirect_uri: http://production-server
  site: https://api.sandbox.inbloom.org/
  rest_url: api/rest/v1.2/
```

In order to get the rails app running, you'll need to make a copy of this file and rename it to "inbloom.yml". Why am I making you do this? So that my `client_id` and `client_secret` don't get uploaded into a git repository. Conveniently, this setup will also protect your sensitive information too.

You'll also need to replace the 'client\_id' value with your application's 'Client Id' from the Manage Applications page in the inBloom sandbox. You'll also need to do the same with 'client\_secret' and "Shared Secret". Once you've made those replacements, this file contains the information the omniauth-inBloom gem needs to authenticate with the inBloom data store.

Also included are the urls we will use to work with the Data Store.



## config/initializers/omniauth.rb

```
INBLOOM_CONFIG =
YAML.load_file("#{Rails.root}/config/inbloom.yml")[:Rails.env]

REST_URL = INBLOOM_CONFIG['site'] + INBLOOM_CONFIG['rest_url']

Rails.application.config.middleware.use OmniAuth::Builder do
  if Rails.env.development? || Rails.env.test?
    provider :inbloom,
      INBLOOM_CONFIG['client_id'],
      INBLOOM_CONFIG['client_secret'],
      :setup =>
lambda{|env|env['omniauth.strategy'].options[:client_options].site = INBLOOM_CONFIG['site']}
  else
    provider :inbloom,
      INBLOOM_CONFIG['client_id'],
      INBLOOM_CONFIG['client_secret']
  end
end
```

Loads in `config/inbloom.yml`. Then defines the `REST_URL` constant. From there we have the setup of the `omniauth-inbloom` gem. You'll note the conditional. This allows us use the sandbox ( `INBLOOM_CONFIG['site']` ) for development and testing, but let's us deploy using the Data Store proper (i.e. the default behavior of the `omniauth-inBloom` gem).

## /config/routes.rb

```
RubyHello::Application.routes.draw do

  root :to => 'hello#index'

  match "/auth/inbloom/callback" => "sessions#create"
  match "/signout" => "sessions#destroy", :as => :signout

end
```

Everybody loves routing, and we're no exception. This is the core of the Hello World.



Nevertheless it's a pretty simple routing document, but then this is a pretty simple app. Directs the root to the index action in the hello controller. Then sets up the routes for authentication with inBloom and logoff from inBloom. This is very much inspired by [Ralis Cast #241 Simple OmniAuth](#).

## Sessions Controller

```
class SessionsController < ApplicationController

  def create
    auth = request.env["omniauth.auth"]
    session[:token] = auth[:credentials][:token]
    logger.info auth.inspect

    redirect_to root_url
  end

  def destroy
    RestClient.get REST_URL + 'system/session/logout'

    session.delete(:token)

    redirect_to root_url
  end
end
```

Handles all the authentication stuff with inBloom. Again based off of [Ralis Cast #241 Simple OmniAuth](#). There's some additional bits in there. In the destroy action, we begin by logging out the inBloom session with a call to the appropriate [API endpoint](#). Then we delete our local token by removing the entry from the session hash.

## Hello Controller

```
class HelloController < ApplicationController

  def createRestClient
    # Need the session token, otherwise everything goes boom
    raise ArgumentError, 'Expecting session[:token] to exist'
  unless session[:token]

    header = {:Content_Type => 'application/vnd.slc+json',
              :Accept => 'application/vnd.slc+json',
              :Authorization => 'bearer ' + session[:token]
            }
    RestClient::Resource.new REST_URL, :headers => header
```



```

end

def index

  if session[:token]
    inBloom = createRestClient

    @rest_home = JSON.parse inBloom['home'].get
  end
end
end

```

The app uses the presence of `:token` as a proxy for successful login with the inBloom Data Store. For there we first define the appropriate headers, and then building a URL in order to make a call to the inBloom REST api.

### Index View

```

<% if session[:token] %>
  <h1><%= @endpoint %></h1>
  <%= link_to "Sign Out", signout_path %>
  <%= @rest_home %>
<% else %>
  <h1>Hello!</h1>
  <%=link_to "Sign in with Inbloom", "/auth/inbloom" %>
<% end %>

```

Again, using `:token` as a signifier for access to the Data Store. If it's there, we show the sign out link and then display the data from the API call. Alternately, we say 'Hello' and direct the user to sign in.

### C#

C#'s MVC4 includes built in support for OAuth2 authentication. We're going to leverage that and add a custom OAuth client to a more or less unchanged MVC4 project.

There are a few libraries that get used throughout this project. All of them are available via Nuget or included in the project. RestSharp is used as a handy REST client. Newtonsoft's Json.NET is used for the serialization and deserialization of c# objects to transfer them through the REST api. You'll need to install these via the Nuget package manager. Finally, the inBloomClient project is used to provide objects that correspond to the REST api.





## Rename the Web.Config files

### Authorization

#### [Authorization configuration](#)

#### /App\_Start/AuthConfig.cs

```
var _inBloomClient = new inBloomClient(  
    appId: WebConfigurationManager.AppSettings["InBloomClientId"],  
    appSecret:  
    WebConfigurationManager.AppSettings["InBloomSharedSecret"]);  
  
OAuthWebSecurity.RegisterClient(_inBloomClient, "inBloom", null);
```

When you look at the AuthConfig.cs there a list of pre-built OAuth Clients for various OAuth 2 service providers. The above code handles the creation of the inBloomClient and its registration with the OAuthWebSecurity client.

#### [inBloom OAuth2Client](#)

#### /CustomOauthClients/inBloomClient.cs

There's a lot that happens in here. The important take away is that this class implements a custom OAuth2Client for inBloom in a manner very similar to the Facebook Client provided in the DotNetOpenAuth project.

### Recipe Controller

#### /Controllers/Recipe

The recipe controller simply acts a tool





# The Recipes

In order to run the recipes in a browser, use the url:

`YOUR_SERVER_NAME/recipes/recipe`  
where 'X' represents the recipe number (i.e.  
`http://localhost:3000/recipes/recipe3`).

The source code for the individual recipes can be found in their respective project directories under:

- Javascript: `routes/recipes/`
- Ruby: `apps/controllers/recipes/`
- C# `recipes/`

---

## 1. Retrieving a user's unique id

### Problem

Obtaining a user's unique inBloom identifier.

### Discussion

inBloom assigns each individual resource a unique ID. This value is necessary for retrieving, changing and removing data from a particular resource, and the method for doing so follows the same pattern as shown here: 1) retrieve the resource 2) pull the id value out.

### Solution

#### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/getUserID', recipe_01.getUserID(client));

/**
 * routes/recipes/recipe_01.js
 * Recipe #1
 */
var config = require('../../config');

exports.getUserID = function (client){
```



```

return function(req, res){

    var recipeDescription = 'Retrieving a user\'s unique id';

    // Create path to users 'home' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+ '/home';

    // GET request to api using Restify module .get function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.get(path, function (err, req2, res2, userJSON) {

        // Use string manipulation to parse 'rel' field of associative array 'userJSON'
        // and assign value to userUniqueID
        for (var i = 0; i < userJSON['links'].length; i++) {
            if (userJSON['links'][i]['rel'] === 'self'){
                var userUniqueID =
userJSON['links'][i]['href'].substring(userJSON['links'][i]['href'].lastIndexOf('/') + 1);
                break;
            }
        }

        // Prints userUniqueID to webpage for verification
        res.render('recipes', { tab_name: 'Recipes',
                                title: 'Recipe 1',
                                description: recipeDescription,
                                data: userUniqueID }
        );
    });
};
};

```

### In Ruby

```

def getUserUID(token)
    home_url = INBLOOM_CONFIG['site'] + INBLOOM_CONFIG['rest_url'] + 'home'
    data = retrieveEndpoint(home_url, token)
    links = data['links']

    # find the self link
    self_url = links.each { |link| if link['rel'] = 'self'
                                    break link['href']
                                end }

    # pull the id field out of the self data
    data = retrieveEndpoint(self_url, token)
    uid = data['id']

    return uid
end

```



### In C#

```
namespace inBloom_c_sharp_hello_world.Recipes
{
    // Determine user's id
    public class Recipe1 : Recipe
    {
        override public string RunRecipe(string token)
        {
            return getUserId(token);
        }

        public string getUserId(string token)
        {
            var client = new
RestClient(WebConfigurationManager.AppSettings["inBloomRestURL"]);
            string homeEndpoint = "home/";

            // retrieve the home endpoint
            RestRequest homeRequest = inBloomRestRequest(token, homeEndpoint, Method.GET);
            var homeResponse = client.Execute(homeRequest);
            dynamic home = JsonConvert.DeserializeObject(homeResponse.Content);

            // find the self endpoint's url
            JArray links = home["links"];
            string selfEndpoint = findLink("self",
links).Replace(WebConfigurationManager.AppSettings["inBloomRestURL"], "");

            // retrieve the self endpoint
            RestRequest selfRequest = inBloomRestRequest(token, selfEndpoint, Method.GET);
            var selfResponse = client.Execute(selfRequest);
            dynamic self = JsonConvert.DeserializeObject(selfResponse.Content);

            // pull out the id field and return it
            return self["id"].ToString();
        }
    }
}
```

---

## 2. Determine user's educational organizations ids

### Problem

Get the all the educational organization ids associated with a user's profile



## Discussion

In order to perform certain actions with the inBloom Rest API, a user will need to be associated with the educational organization in which he/she wishes to get or modify information from. This association is part of the permission requirements necessary to access information that falls under the educational organization's authority.

In order to find the educational organizations associated with a user, a GET request is made to the user's home endpoint. The response to the GET request is then traversed to find the user's `getEducationalOrganizations` HATEOS endpoint. A second GET request is made, this time to the `getEducationalOrganizations` endpoint and the response is traversed and all of the user's educational organization ids are stored in an array.

### Endpoints

home endpoint:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-home](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-home)

staff/{id}/staffEducationOrgAssignmentAssociations/educationOrganizations endpoint:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-staff-id-staffEducationOrgAssignmentAssociations-educationOrganizations](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-staff-id-staffEducationOrgAssignmentAssociations-educationOrganizations)

### Entities

Staff definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Staff](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Staff)

Name definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Name](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Name)

## Solution

### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/getUserEdOrgIDs', recipe_02.getUserEdOrgIDs(client));
```



```

/**
 * routes/recipes/recipe_02.js
 * Recipe #2
 */
var config = require('../../config');

exports.getUserEdOrgIDs = function (client){
  return function(req, res){

    var recipeDescription = 'Determine users\'s educational organizations ids';

    // Create path to users 'home' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+ '/home';

    // GET request to api using Restify module .get function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.get(path, function (err, req2, res2, userJSON) {

      // Searches 'userJSON' for object containing 'getEducationOrganizations'
      // and assigns api endpoint to 'edOrgsPath'
      for (var i = 0; i < userJSON['links'].length; i++) {
        if (userJSON['links'][i]['rel'] === 'getEducationOrganizations'){
          var edOrgsPath = userJSON['links'][i]['href'];
          break;
        }
      }

      client.get(edOrgsPath, function (err, req3, res3, userEdOrgsJSON) {
        var edOrgsIDs = [];

        // Searches all objects in 'userEdOrgsJSON' for 'id' field
        // and pushes 'id' values to an array
        for (var i = 0; i < userEdOrgsJSON.length; i++) {
          edOrgsIDs.push(userEdOrgsJSON[i]['id']);
        }

        // Prints userEdOrgsIDs to webpage for verification
        res.render('recipes', { tab_name: 'Recipes',
                                title: 'Recipe 2',
                                description: recipeDescription,
                                data: edOrgsIDs }

        );
      });
    });
  };
};

```



### In Ruby

```
def getUserEdOrgs
  inBloom = setupRestClient
  schools = JSON.parse inBloom['schools'].get
  schools.map do |school|
    school['id']
  end
end
```

### In C#

```
namespace inBloom_c_sharp_hello_world.Recipes
{
    // Determine user's educational organization ids
    public class Recipe2 : Recipe
    {
        /// Returns a concatenated string of all of the IDs of the Education Organizations
        the user is associated with
        override public string RunRecipe(string token)
        {
            string ret = "";

            IEnumerable<EducationOrganization> eduOrgs = getEducationalOrgs(token);

            // pull out the id fields and return it
            foreach (EducationOrganization eduOrg in eduOrgs)
            {
                ret += eduOrg.id + "\t";
            }
            return ret;
        }

        public IEnumerable<EducationOrganization> getEducationalOrgs(string token)
        {
            var client = new
            RestClient(WebConfigurationManager.AppSettings["inBloomRestURL"]);
            string homeEndpoint = "home/";

            // retrieve the home endpoint
            RestRequest homeRequest = inBloomRestRequest(token, homeEndpoint, Method.GET);
            var homeResponse = client.Execute(homeRequest);
            dynamic home = JsonConvert.DeserializeObject(homeResponse.Content);

            // find the self endpoint's url
            JArray links = home["links"];
            string eduOrgsEndpoint = findLink("getEducationOrganizations",
            links).Replace(WebConfigurationManager.AppSettings["inBloomRestURL"], "");

            // retrieve the self endpoint
            RestRequest eduOrgsRequest = inBloomRestRequest(token, eduOrgsEndpoint,
            Method.GET);
            var eduOrgsResponse = client.Execute(eduOrgsRequest);
        }
    }
}
```





```
        return  
        JsonConvert.DeserializeObject<List<EducationOrganization>>(eduOrgsResponse.Content);  
    }  
}
```

---

### 3. Create a Staff Member

#### Problem

Create a staff member.

#### Discussion

In order to create a staff member, a POST request must be sent to the staff endpoint. The request body for POST must include the required fields defined in the [data model](#). To form the POST request body, package all required and other desired fields into a conventional JSON document.

The user needs the right permissions to create objects. Most roles cannot, by default, create staff members. If a user without the required privileges tries to create a staff member, inBloom will respond with a “403 Forbidden HTTP” status code. For any action creating a new object, handling a possible 403 error is prudent.

#### Endpoints

You’ll need the location value from the `response` to access the newly created staff member.

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-staff](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-staff)

#### Entities

Staff definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Staff](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Staff)

Name definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Name](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Name)



## Solution

### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/createStaffMember', recipe_03.createStaffMember(client));

/**
 * routes/recipes/recipe_03.js
 * Recipe #3
 */
var config = require(' ../../config');

exports.createStaffMember = function (client){
  return function(req, res){

    var recipeDescription = 'Create a staff member';

    // Create path to 'staff' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
    + '/staff';

    var staff = {
      sex: 'Male',
      staffUniqueStateId: 'ksoze',
      hispanicLatinoEthnicity: false,
      name: {
        lastSurname: 'Soze',
        firstName: 'Keyser' },
      highestLevelOfEducationCompleted: 'No Degree'
    };

    // POST request to api using Restify module .post function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.post(path, staff, function (err, req2, res2, obj) {

      var newStaffEndpoint = res2.headers['location'];

      // Prints userEdOrgsIDs to webpage for verification
      res.render('recipes', { tab_name: 'Recipes',
        title: 'Recipe 3',
        description: recipeDescription,
        data: newStaffEndpoint }

      );
    });
  };
};
```



### In Ruby

```
new_staff = staff("jdoe", name("John", "Doe"), "Male", "No Degree")

url = REST_URL + 'staff'
response = RestClient.post url, new_staff.to_json, headers
location = response['location']

def name(first, last, optional ={} )
  name_hash = {
    :firstName => first,
    :lastName => last
  }
  name_hash.update optional
end

def staff( state_id, name, sex, edu_level, optional = {} )
  staff_hash = {
    :staffUniqueId => state_id,
    :name => name,
    :sex => sex,
    :highestLevelOfEducationCompleted => edu_level
  }
  staff_hash.update optional
end
```

### In C#

```
namespace inBloom_c_sharp_hello_world.Recipes
{
    // Create a staff member
    public class Recipe3 : Recipe

    {
        override public string RunRecipe(string token)
        {
            Name name = createName(
                firstName: "John",
                lastName: "Doe"
            );

            Staff staff = createStaff(
                staffUniqueId: "jDoe",
                name: name,
                sex: SexType.Male,
                highestLevelOfEducationCompleted: LevelOfEducationType.NoDegree
            );
        }
    }
}
```



```

        RestResponse response = postStaff(token, staff);

        // RestSharp recommended error code
        if (response.Exception != null)
        {
            const string message = "Error retrieving response. Check details in
exception for more info.";
            var inBloomException = new ApplicationException(message,
response.Exception);
            throw inBloomException;
        }

        // Find Location in headers
        if (response.ResponseStatus == ResponseStatus.Completed)
        {
            foreach (Parameter header in response.Headers)
            {
                if (header.Name == "Location")
                {
                    return header.Value.ToString();
                }
            }
            return "No location value in completed response. Did the post succeed work?";
        }
        else
        {
            return "Post was not successful: " + response.ResponseStatus.ToString();
        }
    }

    // See /Recipes/Recipe3.cs for the details of the various helper functions
}
}

```

---

## 4. Associate staff with an educational organization

### Problem

Once a staff member has been created, the staff member needs to be associated with an Educational Organization before very much can be done with the staff member. Once the association is made, it is easier to access the user in the future and more associations can be created involving the user and various other inBloom entities.



## Discussion

A Staff Education Organization Association requires four pieces of data to get created: a staff id (staff\_id), an education organization id (edu\_org\_id), a staff classification (staff\_class), and a beginning date(begin\_date).

## Entities

StaffEducationOrganizationAssociation:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-StaffEducationOrganizationAssociation](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-StaffEducationOrganizationAssociation)

StaffClassificationType: [https://inbloom.org/sites/default/files/docs-developer/data\\_model-enums.html#type-StaffClassificationType](https://inbloom.org/sites/default/files/docs-developer/data_model-enums.html#type-StaffClassificationType)

## Solution

### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/createStaffEdOrgAssoc', recipe_04.createStaffEdOrgAssoc(client));

/**
 * routes/recipes/recipe_04.js
 * Recipe #4
 */
var config = require('../././config');

exports.createStaffEdOrgAssoc = function (client){
  return function(req, res){

    var recipeDescription = 'Associate staff with an educational organization';

    // Assign existing education organization ID and staff ID for test purposes. This
    // ID will most likely be passed into the function in a functional application.
    var educationOrganizationId = ED_ORG_ID_HERE;
    var staffId = STAFF_ID_HERE;

    // Create path to 'staffEducationOrgAssignmentAssociations' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    ' /staffEducationOrgAssignmentAssociations';

    var staffEdOrgAssoc = {
```



```

        'beginDate': '1999-12-31',
        'staffClassification': 'Principal',
        'educationOrganizationReference': educationOrganizationId,
        'staffReference': staffId
    };

    // POST request to api using Restify module .post function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.post(path, staffEdOrgAssoc, function (err, req2, res2, obj) {

        var newStaffEdOrgAssocEndpoint = res2.headers['location'];

        // Prints new Staff/EducationOrganization association endpoint to webpage for
        verification
        res.render('recipes', { tab_name: 'Recipes',
                                title: 'Recipe 4',
                                description: recipeDescription,
                                data: newStaffEdOrgAssocEndpoint }

        );
    });
};
};
};

```

### In Ruby

```

staff = JSON.parse RestClient.get staff_url, headers
staff_id = staff['id']

edu_org = JSON.parse RestClient.get REST_URL + "educationOrganizations", headers
edu_org_id = edu_org[0]['id']

staff_class = "Other"
begin_date = "2013-05-01"

assoc = staff_education_organization_association( staff_id, edu_org_id, staff_class,
begin_date)

url = REST_URL + "staffEducationOrgAssignmentAssociations"

response = RestClient.post url, assoc.to_json, headers
return response.headers[:location]

def staff_education_organization_association( staff_id, edu_org_id, staff_class,
begin_date, optional = {} )
  association_hash = {
    :staffReference => staff_id,
    :educationOrganizationReference => edu_org_id,
    :staffClassification => staff_class,
    :beginDate => begin_date
  }
  association_hash.update optional
end

```



## In C#

```
namespace inBloom_c_sharp_hello_world.Recipes
{
    public class Recipe4 : Recipe
    {
        override public string RunRecipe(string token) { return "This actual won't work
without two other parameters."; }

        public string RunRecipe(string token, string staffId, string eduOrgId)
        {
            StaffClassificationType staffClass = StaffClassificationType.Other;
            DateTime beginDate = new DateTime(year: 2013, month: 03, day: 15);

            StaffEducationOrganizationAssociation assoc =
createStaffEducationOrganizationAssociation(
                staffId: staffId,
                educationOrganizationReference: eduOrgId,
                staffClassification: staffClass,
                beginDate: beginDate
            );

            RestResponse response = postStaffEducationOrganizationAssociation(token, assoc);

            // RestSharp recommended error handling
            if (response.ErrorException != null)
            {
                const string message = "Error rectrieving response. Check details in
exception for more info.";
                var inBloomException = new ApplicationException(message,
response.ErrorException);
                throw inBloomException;
            }

            // Find Location in headers
            if (response.ResponseStatus == ResponseStatus.Completed)
            {
                foreach (Parameter header in response.Headers)
                {
                    if (header.Name == "Location")
                    {
                        string location = header.Value.ToString();
                        string assocId = location.Replace(response.ResponseUri.ToString(),
""");

                        return assocId;
                    }
                }
                return "No location value in completed response. Did the post succeed?";
            }
            else
            {

```



```

        return "Post was not successful: " + response.ResponseStatus.ToString();
    }
}

// See /Recipes/Recipe4.cs for the details of the various helper functions
}
}

```

---

## 5. Create an Assessment

### Problem

Create an assessment.

### Discussion

In order to create an assessment, a POST request must be sent to the assessments endpoint. The request body for POST must include the required fields defined in the data model. To form the POST request body, package all required and other desired fields into a conventional JSON document.

One of the required fields for creating an assessment is `assessmentIdentificationCode`. This field is an array of JSONs and must be formatted as such for successful creation of an assessment.

### Endpoints

Assessment

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-assessments](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-assessments)

### Entities

Assessment

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Assessment](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Assessment)

`AssessmentIdentificationCode`





[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-AssessmentIdentificationCode](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-AssessmentIdentificationCode)

AssessmentIdentificationSystemType

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-enums.html#type-AssessmentIdentificationSystemType](https://inbloom.org/sites/default/files/docs-developer/data_model-enums.html#type-AssessmentIdentificationSystemType)

## Solution

### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/createAssessment', recipe_05.createAssessment(client));

/**
 * routes/recipes/recipe_05.js
 * Recipe #5
 */
var config = require('../../config');

exports.createAssessment = function (client){
  return function(req, res){

    var recipeDescription = 'Create an assessment';

    // Create path to 'assessments' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    '/assessments';

    var assessment = {
      'academicSubject': 'Reading',
      'assessmentIdentificationCode': [{
        'assigningOrganizationCode': 'Enter assigning orgs code or ID here',
        'ID': 'Enter assessment ID here',
        'identificationSystem': 'Other'
      }],
      'assessmentTitle': '1984',
      'gradeLevelAssessed': 'Tenth grade',
      'version': 42
    };

    // POST request to api using Restify module .post function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.post(path, assessment, function (err, req2, res2, obj) {
```



```

var newAssessmentEndpoint = res2.headers['location'];

// Prints new Assessment endpoint to webpage for verification
res.render('recipes', { tab_name: 'Recipes',
                        title: 'Recipe 5',
                        description: recipeDescription,
                        data: newAssessmentEndpoint }
);
});
};
};

```

### In Ruby

```

new_assessment = assessment(
  "rand1",
  [assessment_identification_code("Other", "Some Identification")],
  1
)

url = REST_URL + 'assessments'
response = RestClient.post url, new_assessment.to_json, headers
location = response.headers[:location]

def assessment( assessment_title, array_of_assessment_identification_codes, version,
optional ={} )
  assessment_hash = {
    :assessmentTitle => assessment_title,
    :assessmentIdentificationCode => array_of_assessment_identification_codes,
    :version => version
  }
  assessment_hash.update optional
end

def assessment_identification_code( identification_system, id, optional ={} )
  assessment_identification_code = {
    :identificationSystem => identification_system,
    :ID => id
  }
  assessment_identification_code.update optional
end

```



### In C#

```
namespace inBloom_c_sharp_hello_world.Recipes
{
    public class Recipe5 : Recipe
    {
        override public string RunRecipe(string token)
        {
            // Create the assessment it's required objects
            AssessmentIdentificationCode idCode = createAssesmentIdentifcationCode(
                identificationSystem: AssessmentIdentificationSystemType.Other,
                ID: "Unique Alphanumeric Identifier"
            );

            List<AssessmentIdentificationCode> idCodes = new
List<AssessmentIdentificationCode>();
            idCodes.Add(idCode);

            Assessment assessment = createAssessment(
                assessmentTitle: "Title",
                assessmentIdentificationCode: idCodes,
                version: 1);

            // return location or error
            return postAssessment(token, assessment);
        }

        // See /Recipes/Recipe5.cs for the details of the various helper functions
    }
}
```

---

## 6. Update an Assessment

### Problem

Updating an existing assessment.

### Discussion

An update is performed as a HTTP PUT request. The request body for PUT must include the data for the standard fields of the entire resource, not just the fields you're updating. To form the PUT request body, we recommend that you start with a GET response body for the resource you're updating, remove the metadata and links, and edit the values for each field you want to update.



There are certain key fields that cannot be modified after the entity is created. These key fields are noted in the entity definition found in the inBloom Developer Documentation. A link to the Assessment Entity can be found below.

Note that trying to update a resource that does not exist results in the “404 Not Found” response code.

### Endpoints

Assessments/{id} endpoint:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-assessments-id](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-assessments-id)

### Entities

Assessment definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Assessment](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Assessment)

## Solution

### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/updateAssessment', recipe_06.updateAssessment(client));
```

```
/**
 * routes/recipes/recipe_06.js
 * Recipe #6
 */
var config = require('../../config');

exports.updateAssessment = function (client){
  return function(req, res){

    var recipeDescription = 'Update an assessment';
```



```

// Assign existing assessment ID for test purposes. This ID will most likely
// be passed into the function in a functional application.
var assessmentId = ASSESSMENT_ID_HERE;

// Create path to 'assessments/{id}' endpoint
var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    '/assessments/' + assessmentId;

// An assessment's key fields (academicSubject, assessmentIdentificationCode,
// assessmentTitle, gradeLevelAssessed, version) cannot be modified once created.
var updatedValuesForAssessment = {
    'maxRawScore': 100,
    'minRawScore': 0
}

// GET request to api using Restify module .get function.
// 'client' object is passed into function (Created in 'Hello World')
client.get(path, function (err, req2, res2, assessment) {

    var assessmentToUpdate = {};

    // Tranverses 'assessment' object and copies field/values to 'assessmentToUpdate'
    // object if the fields are not metadata.
    for (var child in assessment) {
        if (child === 'id' || child === 'links' || child === 'entityType') {
            continue;
        } else if (updatedValuesForAssessment[child]) {
            assessmentToUpdate[child] = updatedValuesForAssessment[child];
            delete updatedValuesForAssessment[child];
        } else {
            assessmentToUpdate[child] = assessment[child];
        }
    }
    // If fields to be updated are not set currently in 'assessment' object,
    // enter fields in 'assessmentToUpdate' object now
    for (var child in updatedValuesForAssessment) {
        if (child === 'id' || child === 'links' || child === 'entityType') {
            continue;
        } else {
            assessmentToUpdate[child] = updatedValuesForAssessment[child];
        }
    }

    // PUT request to api using Restify module .put function.
    client.put(path, assessmentToUpdate, function (err, req3, res3, obj) {

        var updatedAssessment = req3['body'];

        // Prints new Assessment endpoint to webpage for verification
        res.render('recipes', { tab_name: 'Recipes',
                                title: 'Recipe 6',
                                description: recipeDescription,
                                data: updatedAssessment }

```



```

    );
  });
};
};
};

```

### In Ruby

```

an_assessment = JSON.parse RestClient.get endpoint, headers

an_assessment[:maxRawScore] = 100
RestClient.put endpoint, an_assessment.to_json, headers

```

### In C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Configuration;
using Newtonsoft.Json;
using RestSharp;
using InBloomClient.Entities;

namespace inBloom_c_sharp_hello_world.Recipes
{
    public class Recipe6 : Recipe
    {
        override public string RunRecipe(string token) { return ""; }

        public string RunRecipe(string token, string id)
        {
            Assessment assessment = getAssessmentById(token, id);

            assessment.minRawScore = 42;

            RestResponse putResponse = putAssesement(token, id, assessment);

            // RestSharp recommended error code
            if (putResponse.Exception != null)
            {
                const string message = "Error rectrieving response. Check details in
exception for more info.";
                var inBloomException = new ApplicationException(message,
putResponse.Exception);

```



```

        throw inBloomException;
    }
    return assessment.id;
}

private Assessment getAssessmentById(string token, string id)
{
    var client = new
RestClient(WebConfigurationManager.AppSettings["inBloomRestURL"]);
    var endpoint = "assessments/" + id;

    var request = inBloomRestRequest(token, endpoint, Method.GET);

    RestResponse response = (RestResponse)client.Execute(request);
    Assessment assessment =
JsonConvert.DeserializeObject<Assessment>(response.Content);
    return assessment;
}

private RestResponse putAssesment(string token, string id, Assessment assessment)
{
    var client = new
RestClient(WebConfigurationManager.AppSettings["inBloomRestURL"]);
    var endpoint = "assessments/" + id;

    var request = inBloomRestRequest(token, endpoint, Method.PUT);
    request.AddBody(assessment);

    RestResponse response = (RestResponse)client.Execute(request);

    return response;
}
}
}

```

---

## 7. Delete an Assessment

### Problem

Deleting an existing assessment.

### Discussion

A delete is performed as a HTTP DELETE request. There is no request body for DELETE.

Note that trying to delete a resource that does not exist results in the “404 Not Found” response code.



## Endpoint

Assessments/{id} endpoint:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-assessments-id](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-assessments-id)

## Entities

Assessment definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Assessment](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Assessment)

## Solution

### In JavaScript

```
/**
 * Routing code Located in app.js
 */
app.get('/recipes/deleteAssessment', recipe_07.deleteAssessment(client));

/**
 * routes/recipes/recipe_07.js
 * Recipe #7
 */
var config = require('../../config');

exports.deleteAssessment = function (client){
  return function(req, res){

    var recipeDescription = 'Delete an assessment';

    // Create path to 'assessments/{id}' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    '/assessments/{id}';
    // DELETE request to api using Restify module .del function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.del(path, function(err, req2, res2) {

      // Prints previous endpoint of deleted assessment to webpage for verification
      res.render('recipes', { tab_name: 'Recipes',
                             title: 'Recipe 7',
```





```

        description: recipeDescription,
        data: path }
    });
};
};

```

### In Ruby

```
RestClient.delete endpoint, headers
```

### In C#

```

namespace inBloom_c_sharp_hello_world.Controllers
{
    public class Recipe7 : Recipes.Recipe
    {
        public string RecipeMessage()
        {
            return "Assessment deleted at: ";
        }

        override public string RunRecipe(string token) { return ""; }

        public string RunRecipe(string token, string id)
        {
            return deleteAssessmentById(token, id);
        }

        private string deleteAssessmentById(string token, string id)
        {
            var client = new RestClient("https://api.sandbox.inbloom.org/api/rest/v1.2/");

            var endpoint = "assessments/" + id;

            var request = inBloomRestRequest(token, endpoint, Method.DELETE);

            RestResponse response = (RestResponse)client.Execute(request);

            // RestSharp recommended error code
            if (response.Exception != null)
            {
                const string message = "Error retrieving response. Check details in
exception for more info.";
                var inBloomException = new ApplicationException(message,
response.Exception);
                throw inBloomException;
            }

            if (response.ResponseStatus == ResponseStatus.Completed)

```



```

    {
        return response.ResponseStatus.ToString();
    }
    else
    {
        return "Delete was not successful: " + response.ResponseStatus.ToString();
    }
}
}
}

```

---

## 8. Enroll a Student in a Section

### Problem

Create an association between an existing student and an existing section

### Discussion

In order to enroll a student in a section, a Student Section Association must be created. This is done by sending a HTTP POST request with the association information in the request body.

The information required to create a Student Section Association are the start date of the enrollment (beginDate), the student Id (studentId), and the section Id (sectionId).

### Endpoints

Student Section Associations endpoint:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-studentSectionAssociations](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-studentSectionAssociations)

### Entities

Student Section Associations definition:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-studentSectionAssociations](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-studentSectionAssociations)



## Solution

### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/createStudentSectionAssoc', recipe_08.createStudentSectionAssoc(client));

/**
 * routes/recipes/recipe_08.js
 * Recipe #8
 */
var config = require('../../config');

exports.createStudentSectionAssoc = function (client){
  return function(req, res){

    var recipeDescription = 'Enroll a student in a section';

    // Assign existing section ID and student ID for test purposes. This ID will most
    likely // be passed into the function in a functional application.
    var sectionId = SECTION_ID_HERE;
    var studentId = STUDENT_ID_HERE;

    // Create path to 'studentSectionAssociations' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    '/studentSectionAssociations';

    var studentSectionAssoc = {
      'beginDate': '2012-12-31',
      'sectionId': sectionId,
      'studentId': studentId
    };

    // POST request to api using Restify module .post function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.post(path, studentSectionAssoc, function (err, req2, res2, obj) {

      var newStudentSectionAssocEndpoint = res2.headers['location'];

      // Prints new Staff/EducationOrganization association endpoint to webpage for
      verification
      res.render('recipes', { tab_name: 'Recipes',
                             title: 'Recipe 8',
                             description: recipeDescription,
                             data: newStudentSectionAssocEndpoint }

      );
    });
  });
};
```



```
};
};
```

### In Ruby

```
student = JSON.parse RestClient.get student_endpoint, headers
student_id = student['id']

section = JSON.parse RestClient.get section_endpoint, headers
section_id = section['id']

begin_date = Time.now.strftime("%Y-%m-%d")

assoc = student_section_association(student_id, section_id, begin_date)
logger.info "Association: " + assoc.to_json

url = REST_URL + "studentSectionAssociations"
response = RestClient.post url, assoc.to_json, headers
location = response.headers[:location]

def student_section_association( student_id, section_id, begin_date, optional ={} )
  student_section_association_hash = {
    :studentId => student_id,
    :sectionId => section_id,
    :beginDate => begin_date
  }
  student_section_association_hash.update optional
end
```

### In C#

```
namespace inBloom_c_sharp_hello_world.Controllers
{
  class Recipe8 : Recipe
  {
    override public string RunRecipe(string token)
    {
      // get a student
      List<Student> students = (List<Student>)getStudents(token);
      Student student = students.First();

      // get a section
      List<Section> sections = (List<Section>)getSections(token);
      Section section = sections.First();

      // create a date
      DateTime begindate = DateTime.Now;
```



```

// create a Student section association
StudentSectionAssociation assoc = createStudentSectionAssociation(
    studentId: student.id,
    sectionId: section.id,
    beginDate: begindate
);

// Post the association to the database
RestResponse response = postStudentSectionAssociation(token, assoc);

// RestSharp recommended error handling
if (response.ErrorException != null)
{
    const string message = "Error rectrieving response. Check details in
exception for more info.";
    var inBloomException = new ApplicationException(message,
response.ErrorException);
    throw inBloomException;
}

// return the Location of the new association
if (response.ResponseStatus == ResponseStatus.Completed)
{
    foreach (Parameter header in response.Headers)
    {
        if (header.Name == "Location")
        {
            string location = header.Value.ToString();
            string assocId = location.Replace(response.ResponseUri.ToString(),
""");

            return assocId;
        }
    }
    return "No location value in completed response. Does this resource already
exist?";
}
else
{
    return "Post was not successful: " + response.ResponseStatus.ToString();
}
}
// See /Recipes/Recipe8.cs for the details of the various helper functions
}
}

```



## 9. Find a Student by Criteria

### Problem

Find a student by searchable criteria.

### Discussion

There are times when a user will need to find a student based on the information located in the student's profile. The inBloom Rest API allows a user to perform a query using the students endpoint and adding student field names, values and [query operators](#). If the query matches multiple students, all of the matching students will be returned by the query.

### Endpoints

Students endpoint:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-students](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-students)

### Entities

Student definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Student](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Student)

## Solution

### In JavaScript

```
/**
 * Routing code Located in app.js
 */
app.get('/recipes/getStudentByQuery', recipe_09.getStudentByQuery(client));

/**
 * routes/recipes/recipe_09.js
 * Recipe #9
 */
var config = require('../../config');

exports.getStudentByQuery = function (client){
```



```

return function(req, res){

    var recipeDescription = 'Find a student by criteria';
    var query = 'name.firstName=Daniel';

    // Create path to 'students' endpoint including queries
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
        '/students?' + query;

    // GET request to api using Restify module .get function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.get(path, function (err, req2, res2, studentsQueryJSON) {

        var studentsQueryJSON_string = JSON.stringify(studentsQueryJSON, 'false', 2);

        // Prints userUniqueID to webpage for verification
        res.render('recipes', { tab_name: 'Recipes',
                                title: 'Recipe 9',
                                description: recipeDescription,
                                data: studentsQueryJSON_string }

        );
    });
};

```

## In Ruby

```

url = REST_URL + "students?name.firstName=Damon"

@json = JSON.parse RestClient.get url, headers

```

## In C#

```

namespace inBloom_c_sharp_hello_world.Recipes
{
    public class Recipe9 :Recipe
    {
        public override string RunRecipe(string token)
        {
            Dictionary<string, string> queries = new Dictionary<string,string>();

            queries.Add("name.firstName", "Mi-Ha");

            List<Student> students = (List<Student>)getStudents(token, queries);

            string ret = "";

```



```

        foreach (Student student in students)
        {
            ret += student.name.firstName + "\t" + student.id + "\n";
        }
        return ret;
    }

    // See /Recipes/Recipe9.cs for the details of the various helper functions
}

```

---

## 10. Update a Student's Profile

### Problem

Updating an existing student's profile.

### Discussion

An update is performed as a HTTP PUT request. The request body for PUT must include the data for the standard fields of the entire resource, not just the fields you're updating. To form the PUT request body, we recommend that you start with a GET response body for the resource you're updating, remove the metadata and links, and edit the values for each field you want to update.

There are certain key fields that cannot be modified after the entity is created. These key fields are noted in the entity definition found in the inBloom Developer Documentation. A link to the Student Entity can be found below.

Note that trying to update a resource that does not exist results in the 404 Not Found response code.

### Endpoints

Students/{id} endpoint:





[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-students-id](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-students-id)

### Entities

Student definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Student](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Student)

## Solution

### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/updateStudent', recipe_10.updateStudent(client));

/**
 * routes/recipes/recipe_10.js
 * Recipe #10
 */
var config = require('../../config');

exports.updateStudent = function (client){
  return function(req, res){

    var recipeDescription = 'Update a student\'s profile';

    // Assign existing student ID for test purposes. This ID will most likely
    // be passed into the function in a functional application.
    var studentId = STUDENT_ID_HERE;

    // Create path to 'students/{id}' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    '/students/' + studentId;

    // An student's key fields (studentUniqueStateId) cannot be modified once created.
    var updatedValuesForStudent = {
      'otherName': [{
        'firstName': 'Mark',
        'lastSurname': 'Twain',
        'otherNameType': 'Alias'
      }]
    }

    // GET request to api using Restify module .get function.
```



```

// 'client' object is passed into function (Created in 'Hello World')
client.get(path, function (err, req2, res2, student) {

    var studentToUpdate = {};

    // Traverses 'student' JSON, removing links and metadata, copies fields/values
    being updated
    // from 'updatedValuesForStudent' JSON and copies fields/values not being updated
    from
    // 'student' JSON and places these fields/values in 'studentToUpdate' JSON.
    for (var child in student) {
        if (child === 'id' || child === 'links' || child === 'entityType') {
            continue;
        } else if (updatedValuesForStudent[child]) {
            studentToUpdate[child] = updatedValuesForStudent[child];
            delete updatedValuesForStudent[child];
        } else {
            studentToUpdate[child] = student[child];
        }
    }
    for (var child in updatedValuesForStudent) {
        if (child === 'id' || child === 'links' || child === 'entityType') {
            continue;
        } else {
            studentToUpdate[child] = updatedValuesForStudent[child];
        }
    }
}

// PUT request to api using Restify module .put function.
client.put(path, studentToUpdate, function (err, req3, res3, obj) {

    var updatedStudent = req3['body'];

    // Prints new Assessment endpoint to webpage for verification
    res.render('recipes', { tab_name: 'Recipes',
                           title: 'Recipe 10',
                           description: recipeDescription,
                           data: updatedStudent }

    );
});
});
};
};
};

```

### In Ruby

--Insert Ruby code here--



### In C#

```
namespace inBloom_c_sharp_hello_world.Recipes
{
    public class Recipe10 : Recipe
    {
        override public string RunRecipe(string token)
        {
            // get a student to work with
            List<Student> students = (List<Student>)getStudents(token);
            Student student = students.First();

            OtherName alias = createOtherName(
                firstName: "Mark",
                lastSurname: "Twain",
                otherNameType: OtherNameType.Alias
            );

            if (student.otherName == null)
            {
                student.otherName = new List<OtherName>();
            }
            student.otherName.Add(alias);

            RestResponse response = putStudent(token, student);

            return response.StatusCode.ToString();
        }

        // See /Recipes/Recipe10.cs for the details of the various helper functions
    }
}
```

---

## 11. Assign a Grade to a Student

### Problem

Assign a grade to an existing student who is enrolled in an existing section.

### Discussion

In order to assign a grade to a student, the student needs to exist as an entity and be enrolled in a section. By default, this means a Student Section Association must also exist. Assigning a grade to a student is done by sending a HTTP POST request with the association information in the request body.



The necessary information required to create a Grade is the type of grade (gradeType), the school year the grade was taken (schoolYear), the student Id (studentId), the section Id (sectionId), and the student section association Id (studentSectionAssociationId).

### Endpoints

Grades endpoint:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-grades](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-grades)

### Entities

Grade definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Grade](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Grade)

## Solution

### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/createGradeForStudent', recipe_11.createGradeForStudent(client));

/**
 * routes/recipes/recipe_11.js
 * Recipe #11
 */
var config = require('../././config');

exports.createGradeForStudent = function (client){
  return function(req, res){

    var recipeDescription = 'Assign a grade to a student';

    // Assign existing section ID, student ID, and student/section association ID for
    test purposes.
    // These IDs will most likely be passed into the function in a functional
    application.
    var sectionId = SECTION_ID_HERE;
    var studentId = STUDENT_ID_HERE;
```



```

var studentSectionAssociationId = STUDENT_SECTION_ASSOC_ID_HERE;

// Create path to 'grades' endpoint
var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    '/grades';

var grade = {
    'gradeType': 'Exam',
    'letterGradeEarned': 'A+',
    'numericGradeEarned': 99,
    'schoolYear': '2011-2012',
    'sectionId': sectionId,
    'studentId': studentId,
    'studentSectionAssociationId': studentSectionAssociationId
};

// POST request to api using Restify module .post function.
// 'client' object is passed into function (Created in 'Hello World')
client.post(path, grade, function (err, req2, res2, obj) {

    var newGradeEndpoint = res2.headers['location'];

    // Prints new Assessment endpoint to webpage for verification
    res.render('recipes', { tab_name: 'Recipes',
                           title: 'Recipe 11',
                           description: recipeDescription,
                           data: newGradeEndpoint }

    );
});
};
};

```

### In Ruby

--Insert Ruby code here--

### In C#

```

namespace inBloom_c_sharp_hello_world.Recipes
{
    public class Recipe11 : Recipe
    {
        public override string RunRecipe(string token)
        {
            List<StudentSectionAssociation> assocs = getStudentSectionAssociation(token);
            StudentSectionAssociation assoc = assocs.First();

```



```

        string studentSectionAssociationId =
"264c2a4b468a418b3fe6005263192e2957b9497e_id8d843440104f871e2717f38759c10c89c3742092_id";
        string sectionId = "264c2a4b468a418b3fe6005263192e2957b9497e_id";
        string studentId = "ea99ef20bbb347149565f4eb8318f7f9d16265c4_id";

        /*string studentSectionAssociationId = assoc.id;
        string sectionId = assoc.sectionId;
        string studentId = assoc.studentId;*/

        Grade grade = createGrade(
            gradeType: GradeType.Exam,
            schoolYear: SchoolYearType.Y20122013,
            sectionId: sectionId,
            studentId: studentId,
            studentSectionAssociation: studentSectionAssociationId);

        RestResponse response = postGrade(token, grade);
        return response.StatusCode.ToString();
    }

    // See /Recipes/Recipe11.cs for the details of the various helper functions
}
}

```

---

## 12. List Parent Contact Information for an Entire Class

### Problem

Create a list that includes the parent contact information for each student enrolled in a class.

### Discussion

Creating a list of parent contact information for a section of students requires obtaining the section id. Once the section id is found, two separate GET requests are processed and the responses are combined to form the parent contact JSON for the entire section.

The first GET request is made to the sections/{id}/studentSectionAssociations/students/studentParentAssociations endpoint. The JSON returned in the response contains a very important field for each Student Parent Association, contactRestrictions. The contactRestrictions field holds a string value of any contact restrictions the parent may have with the associated child.

The second GET request is made to the sections/{id}/studentSectionAssociations/students/



studentParentAssociations/parents endpoint. The JSON returned in the response contains information for each parent of the students in the section. Parent contact information is included in this information.

Before rendering the parent contact information from the second GET request, the contactRestriction field from the first GET request is added to the associated parent.

### Endpoints

sections/{id}/studentSectionAssociations/students/studentParentAssociations endpoint:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-sections-id-studentSectionAssociations-students-studentParentAssociations](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-sections-id-studentSectionAssociations-students-studentParentAssociations)

sections/{id}/studentSectionAssociations/students/studentParentAssociations/parents endpoint:

[https://inbloom.org/sites/default/files/docs-developer/rest\\_api\\_resources-data-v1.2.html#sect-v1.2-sections-id-studentSectionAssociations-students-studentParentAssociations-parents](https://inbloom.org/sites/default/files/docs-developer/rest_api_resources-data-v1.2.html#sect-v1.2-sections-id-studentSectionAssociations-students-studentParentAssociations-parents)

### Entities

Student Parent Associations definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-StudentParentAssociation](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-StudentParentAssociation)

Parent definition:

[https://inbloom.org/sites/default/files/docs-developer/data\\_model-entities.html#type-Parent](https://inbloom.org/sites/default/files/docs-developer/data_model-entities.html#type-Parent)



## Solution

### In JavaScript

```
/**
 * Routing code located in app.js
 */
app.get('/recipes/getParentsForSection', recipe_12.getParentsForSection(client));

/**
 * routes/recipes/recipe_12.js
 * Recipe #12
 */
var config = require('../././config');

exports.getParentsForSection = function (client){
  return function(req, res){

    var recipeDescription = 'List parent contact information for an entire class';

    // Assign existing section ID for test purposes. This ID will most likely be passed
    into
    // the function in a functional application.
    sectionID = SECTION_ID_HERE;

    // Create path to 'sections/{id}/studentSectionAssociations/students
    // /studentParentAssociations/parents' endpoint
    var path = config.api.base_url + config.api.api_call_append + config.api.api_version
+
    '/sections/' + sectionID +
    '/studentSectionAssociations/students/studentParentAssociations';

    // GET request to api using Restify module .get function.
    // 'client' object is passed into function (Created in 'Hello World')
    client.get(path, function (err, req2, res2, studentParentAssocJSON) {

      var parentContactRestrictions = [];

      // Traverses 'studentParentAssocJSON' object and stores contactRestrictions
      // field in parentContactRestriction array
      for (var i = 0; i < studentParentAssocJSON.length; i++) {
        parentContactRestrictions[i] = {};
        parentContactRestrictions[i]['parentId'] =
studentParentAssocJSON[i]['parentId'];
        if (studentParentAssocJSON[i]['contactRestrictions']) {
          parentContactRestrictions[i]['contactRestrictions'] =
            studentParentAssocJSON[i]['contactRestrictions'];
        } else {
          parentContactRestrictions[i]['contactRestrictions'] = 'none';
        }
      }
    })
  }
}
```





```

path = path + '/parents';

// GET request to api using Restify module .get function.
client.get(path, function (err, req3, res3, parentJSON) {

    // Traverses parentJSON and adds the contactRestriction field/value to the
    // associated parent.
    for (var i = 0; i < parentJSON.length; i++) {
        for (var j = 0; j < parentContactRestrictions.length; j++) {
            if (parentJSON[i]['id'] === parentContactRestrictions[j]['parentId'])
        {
            parentJSON[i]['contactRestrictions'] =
                parentContactRestrictions[j]['contactRestrictions'];
            break;
        }
    }

    var parentJSON_string = JSON.stringify(parentJSON, 'false', 2);

    // Prints userEdOrgsIDs to webpage for verification
    res.render('recipes', { tab_name: 'Recipes',
                            title: 'Recipe 12',
                            description: recipeDescription,
                            data: parentJSON_string }

    );
});
});
};
};

```

### In Ruby

--Insert Ruby code here--

### In C#

```

namespace inBloom_c_sharp_hello_world.Recipes
{
    public class Recipe12 : Recipe
    {
        public override string RunRecipe(string token)
        {
            // Get section to work with
            List<Section> sections = getSections(token);

```



```

        Section section = sections.First();
        string sectionId = section.id;

        List<StudentParentAssociation> assocs =
getStudentParentAssociationsBySection(token, sectionId);
        Dictionary<string, string> contactRestrictions = findContactRestrictions(assocs);

        List<List<object>> stuff = new List<List<object>>>();
        foreach(StudentParentAssociation assoc in assocs)
        {
            List<object> temp = new List<object>();
            temp.Add(assoc.parentId);
            temp.Add(getParentById(token, assoc.parentId));
            foreach(KeyValuePair<string, string> kvp in contactRestrictions)
            {
                if (kvp.Key == assoc.parentId) {
                    temp.Add(kvp.Value);
                }
            }
            stuff.Add(temp);
        }

        return stuff.ToString();
    }

    // See /Recipes/Recipe12.cs for the details of the various helper functions
}
}

```

---

## Further Information

### InBloom docs

[inBloom Developer Documentation](#)

### Forums

[inBloom Developer Forum](#)

