System Documentation

Richard Chan | Development Project II

RAC System

Philip Dumaresq

2018

# Summary

This document will contain sections describing how the system was implemented for future maintenance. It will cover the standards followed during development, an overview of the architecture of the system, an explanation of the data model and some details about the code.

This document should prove sufficient to learn how to navigate the RAC system.

Table of Contents

[Summary 1](#_Toc512550246)

[Table of Figures 1](#_Toc512550247)

[Introduction 2](#_Toc512550248)

[System Architecture 2](#_Toc512550249)

[Data Model 3](#_Toc512550250)

[RAC Database 3](#_Toc512550251)

[CSAdmin Database 7](#_Toc512550252)

[The Code 8](#_Toc512550253)

[Where to find it 8](#_Toc512550254)

[How to Run 8](#_Toc512550255)

[Standards 8](#_Toc512550256)

[Coding Standards 8](#_Toc512550257)

[Organizational Standards 9](#_Toc512550258)

# Table of Figures

[Figure 1: System Architecture Diagram 2](#_Toc512586819)

[Figure 2: RAC Data Model 3](#_Toc512586820)

[Figure 3: Explanation of the Tables in the RAC Data Model. 6](#_Toc512586821)

[Figure 4: Microsoft Membership Data Model 7](#_Toc512586822)

[Figure 5: Explanation of the different back end files in the system. 9](#_Toc512586823)

# Introduction

The purpose of this document is to familiarize you with the RAC system’s code. It will serve to get you to know the system so that when looking into the actual code itself, it doesn’t prove to be difficult to dig through to begin maintenance or development.

This document will contain various sections explaining the system. The first thing covered will simply be where the code is and what it does. This includes viewing it in the web client for TFS and connecting to the RAC project from visual studio to begin editing. Following that will be the standards that have been followed and will be expected to be followed to maintain the styling consistency throughout the system. It will contain the standards for not only the code style, but also for directory and document naming and organization. Following that will be a section explaining the system architecture and how MVC was used to build our system. Finally, our data model will be introduced and explained in depth, going over the purpose of each of the tables.

# System Architecture

The system was designed following the Model View Controller (MVC) architecture provided by Microsoft. As such, we’re using the default 3 layers that this provides us with, as well as some extra layers that we’ve added.

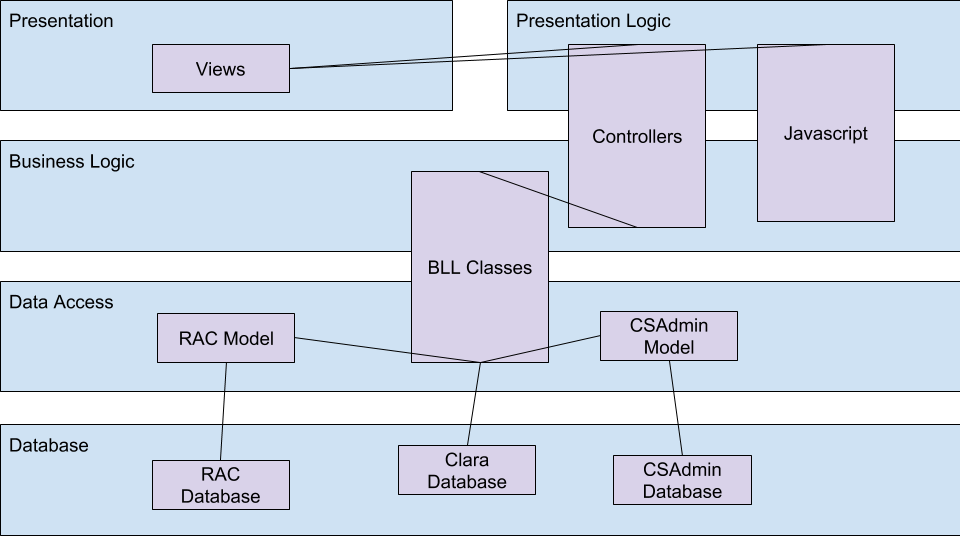


Figure 1: System Architecture Diagram

The default layers to the MVC architecture that have been used are the Presentation (View in MVC), the Presentation Logic (Controller in MVC) and the Data Access layer (Model in MVC). We have also added the Business Logic model, which handles the manipulation of data when it passes between the Presentation and the data access, and we have added the database which handles the storage of all of the data.

# Data Model

## RAC Database

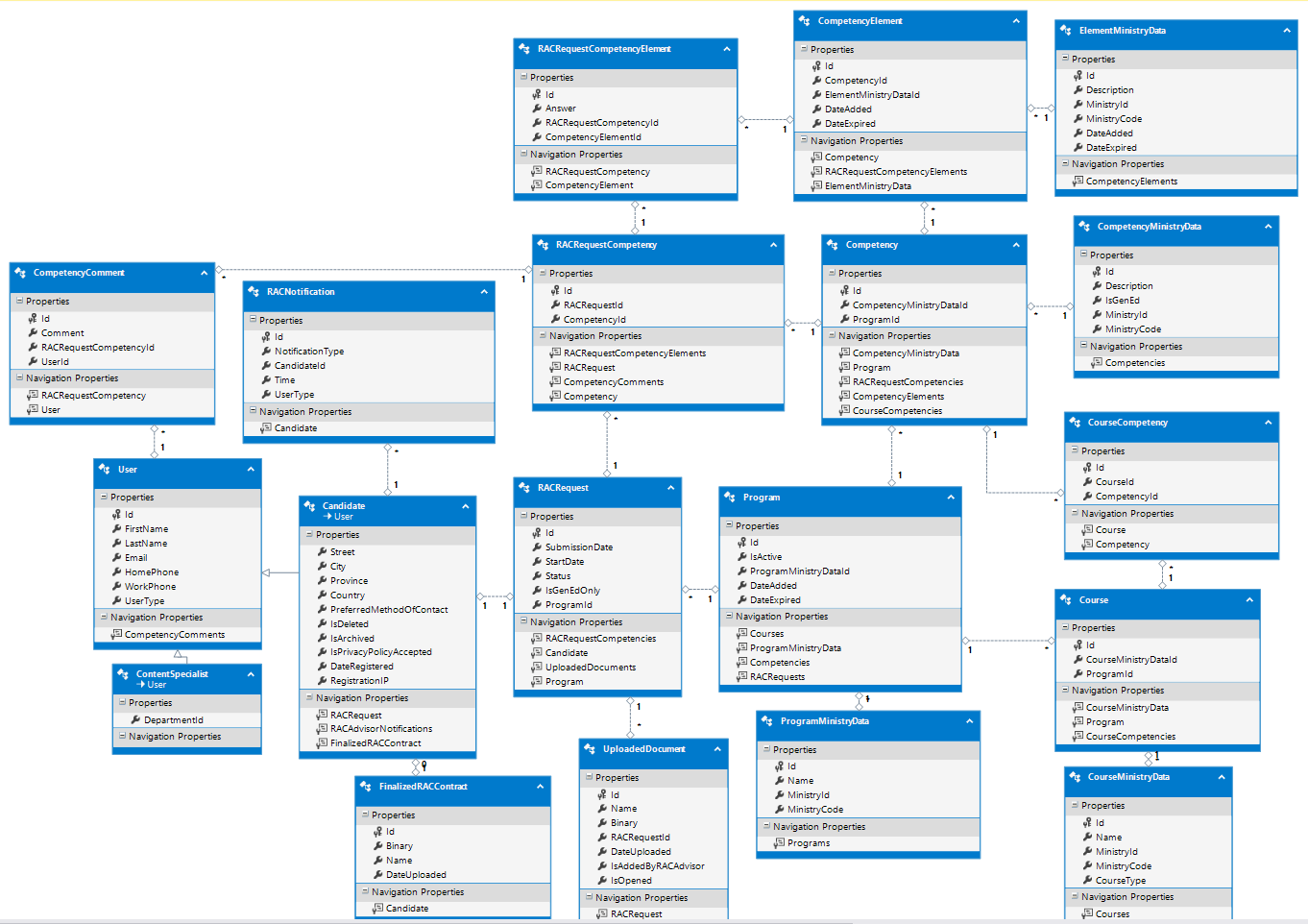
Our data model was created through dozens of modifications to an original simplistic model. This model contains only the data used for storing the tables that are part of the main business operations. There’s a second data model storing all of the user information for our system the is part of an external system. 

Figure 2: RAC Data Model

The purpose of each of the above tables is as follows:

|  |  |  |
| --- | --- | --- |
| **Table Data** | | Purpose |
| **Name** | **Columns** |
| RACRequestComptencyElement | * ID * Answer * RACRequestCompencyId * CompetencyElementId | This table is for the instances of a competency element appearing in a RAC request competency. This table stores the answers to a candidate’s self-evaluations. |
| CompetencyElement | * Id * CompetencyId * ElementMinistryDataId * DateAdded * DateExpired | The CompetencyElement table is a table that stores all the competency elements that are stored in the system. |
| ElementMinistryData | * Id * Description * MinistryId * MinistryCode * DateAdded * DateExpired | This table stores the ministry of education’s data for the competency element. This is so that the data doesn’t get repeated for each time the element is stored in a program. They can be added to the system and then re-used. |
| ComptencyComment | * Id * Comment * RacRequestCompetencyId * UserId | This table is used to store a user’s comment on a candidate’s RAC request. This is it’s own table to handle the potential future implementation of RAC advisors or Content Specialists making comments on the Candidate’s self-evaluation. |
| RACNotification | * Id * NotificationType * CandidateId * Time * UserType | This is for storing notifications for both the candidate and the RAC advisor. They are shown to the UserType stored, and display notifications based off the NotificationType – which is ultimately an enum stored in our code base. |
| RACRequestCompetency | * Id * RACRequestId * CompetencyId | This table stores the competency used in a RAC request. Ultimately it acts as an associative entity to provide the competency description and comments to the candidate and RAC advisor. |
| Competency | * Id * CompetencyMinistryDataId * ProgramId | This table is used to store all of the competencies in the system for each version of a program that uses it. It’s an associative entity tying the ministry data, the elements and the program. This doesn’t store the description themselves to avoid duplicate data. |
| CompetencyMinistryDate | * Id * Description * IsGenEd * MinistryId * MinistryCode | This table serves to normalize the database, as with the ElementMinistryData table. This table stores the bulk of the information for the competency and is then tied in with an associative entity. |
| User | * Id * FirstName * LastName * Email * HomePhone * WorkPhone * UserType | The purpose of the user table is to store all of the information for users in our system that isn’t stored in the Microsoft membership tables. This entity is also inherited by Candidate and ContentSpecialist, so if ContentSpecialist never ends up getting implemented, this can be combined into the Candidate table. |
| Candidate | * Street * City * Province * PreferreredMethodOfContact * IsDeleted * IsArchived * IsPrivacyPolicyAccepted * DateRegistered * RegistrationIp | The candidate table serves to store all of the information that is applicable only to the Candidate users and not to the Content specialist. The Candidate table is where the RACRequest is tied into the user and has a 1-1 relationship with the RACRequest table. This can get combined with the User table if content specialist does not get implemented. |
| ContentSpecialist | * DepartmentId | This table is currently NOT USED at all. This is in here due to changing requirements and never ended up getting removed. If the ContentSpecialist workflow ever gets implemented, it will be relevant. |
| RACRequest | * Id * SubmissionDate * StartDate * Status * IsGenEdOnly * ProgramId | This is the table that ties the entire process to the candidate. The program chosen by the candidate is stored in this table and all of the RACRequestCompetency elements will tie into this table to store all of the information for a user’s RAC reques.t |
| Program | * Id * IsActive * ProgramMinistryDateId * DateAdded * DateExpired | The program table serves to store a record of all of the programs that the college offers RAC for. This table will include multiple versions of the same program because every time a program revision is made, this table will have a new row added with the old one kept for data integrity. |
| Course | * Id * CourseMinistryData * ProgramId | The course table serves to store all of the courses offered by a program in the system. The table is used in every version of a program that uses it, and as such acts more as an associative entity so that you can pull data from the CourseMinistryData table. |
| CourseCompetency | * Id * CourseId * CompetencyId | This table allows the RACAdvisor to map the competencies in the system to the courses. This is so that in the end, the candidate can get a sense of how many courses they will be able to get credited. While the competencies can be mapped in the system, this feature to show that to the user is not currently implemented. |
| FinalizedRACContract | * Id * Binary * Name * DateUploaded | This table is also not currently being implemented in the system and is made redundant by the IsAddedByRacAdvisor field in the UploadedDocument table. This was for RAC advisors to upload the final RAC contracts to Candidates. |
| UploadedDocuement | * Id * Name * Binary * RACRequestId * DateUploaded * IsAddedByRacAdvisor * IsOpened | This table allow candidates and RAC advisors to uploaded files to the system. This is so candidates can upload resumes, job contracts, etc., for the RAC advisor to see, and the RAC advisor can upload contracts and other documents for the candidates. |
| ProgramMinistryData | * Id * Name * MinistryId * MinistryCode | This table, as with the other XMinistryData tables, server to normalize the database by storing the program data only once. This way the program table doesn’t need to have this repeat for every different profile of a given program. |
| CourseMinistrydata | * Id * Name * MinistryId * MinistryCode * CourseType | Same as the rest of the XMinistryData tables, this one stores the courses so they aren’t duplicated for every different program profile. |

Figure 3: Explanation of the Tables in the RAC Data Model.

## CSAdmin Database

The second data model used by our system is stored in a different database. This is the database used by the Microsoft Membership framework for user authentication, account management, email validations, password resets, etc. This data model can be found in the CSAdmin database.

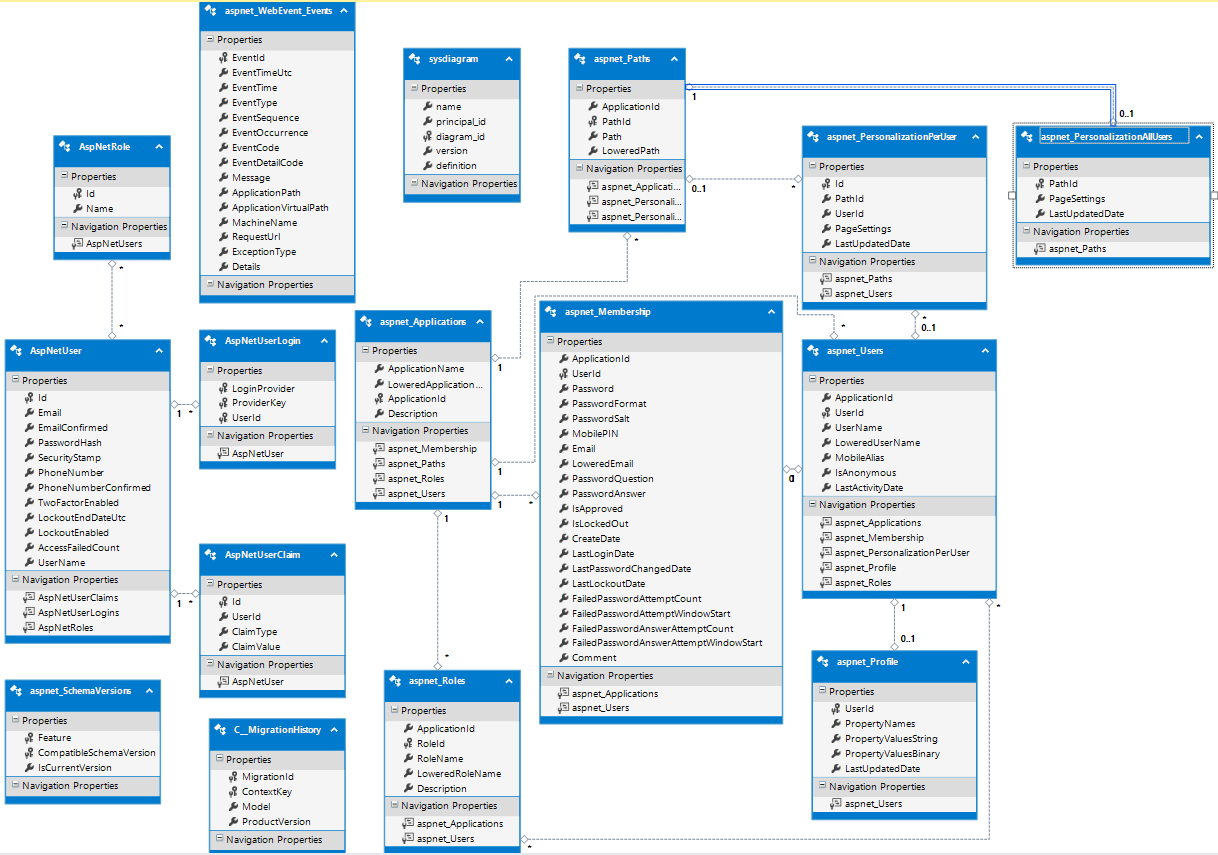


Figure 4: Microsoft Membership Data Model

These tables are all default tables from the Microsoft Membership framework. This framework handles everything to do with user accounts in our system. It handles things like password recovery, email confirmation, roles, etc. More about the Microsoft Membership framework can be found in [online documentation](https://msdn.microsoft.com/en-us/library/yh26yfzy.aspx) provided by Microsoft.

This table is currently stored in the RAC database, however will likely be changing to the CSAdmin database prior to deployment.

# The Code

The code is divided up into our various different layers in the system architecture. The main places that the code can be found are the BusinessLogic, Controllers, RACModels, RACModels/Validation, Scripts and Views folders. This document will contain descriptions of the most important files, which are mostly those that handle the back end logic.

## Where to find it

|  |  |
| --- | --- |
| **File Name** | **Purpose** |
| **./RAC/RAC/BusinessLogic/** | |
| CandidateBLL.cs | This class handle everything to do with the management of candidates. It holds methods for registration, updating candidates, deleting and archiving candidates, server side validation, getting users, changing programs, etc.. |
| CompetencyBLL.cs |  |
| DbContext.cs | The purpose of this file is to provide a singleton of the database context to the system. While this doesn’t scale well to a larger system, it simplifies the code greatly for the RAC system. |
| DocumentList.cs | The document list is also provided to the system as a singleton. This class also handles the management of files in the system, saving and removing documents as well as reading from and writing to the DB. |
| Notifications.cs | Handles the sending of email notifications and sending notifications to candidate or RAC advisor home pages. It saves notifications to the DB and removes them as well. |
| ProgramBLL.cs | This file handles the CRUD operations on everything to do with programs. It had various different ways to extract programs from Clara, as well as the competencies and courses for a program. This will also the management of Competency elements within the system. |
| RACAdvisorBLL.cs | Since the RAC advisor comes out of CSAdmin and not our system, there’s no CRUD to perform. Instead this BLL class handles sending notifications to the RAC Advisor and mapping ViewModels. |
| RACRequestBLL.cs | CRUD on RACRequests in the DB. It handles changing programs, filling out the form, and validating the self-evaluation form. |
| UploadedDocumentBLL.cs | This handles the reading and writing of files from and to the database. It will also handle the notifications on documents, setting the indication on whether or not they have been read yet by a candidate or a RAC advisor. |
| **./RAC/RAC/Controllers/** | |
| AccountController.cs | This controller handles the presentation logic for everything in the “./RAC/RAC/Views/Account/” folder. This includes things such as the ChangeProgram, Register and ResetPassword views. |
| CandidateController.cs | This controller will be handling everything within the “./RAC/RAC/Views/Candidate/” folder. This will be views such as Home, SelfEvaluation, ViewAccount, etc. |
| HomeController.cs | The home controller will only handle the views that are available without being logged in. This includes the About, Help, ReleaseNotes, etc. |
| RACAdvisorController.cs | This will handle all of the views that the RAC advisor can access in the “./RAC/RAC/Views/RACAdvisor” folder. This will include the views for listing all candidates, managing programs, etc. |
| UploadedDocumentController.cs | This controller handles only partial views that are inserted into pages for handling documents. It will serve the \_UploadDocuments and \_ViewDocuments partial views to the self-evaluation, Candidate home, Candidate details, and various other views throughout the system. |
| **./RAC/RAC/RACModels/Validation/** | |
| CandidateVal.cs | These files are all pipeline validation classes for the Candidate, User, and Password. Since we’re using the Model first approach in Entity Framework, our model gets reset whenever we make a change to our database architecture. Due to this, we’ve written validation directly into these 3 files.  Whenever a Candidate or a User is passed around within the front end, it’s a CandidateVal or UserVal model being passed, and is then mapped to a Candidate or User before being used in the backend. These files contain validation on rules on all of the attributes on the 3 classes including annotations for how they should be formed.  For example, the Province attribute in CandidateVal contains the annotation:  [Required(ErrorMessage = “Please enter a province”)] |
| ForgotPasswordVal.cs |
| UserVal.cs |

Figure 5: Explanation of the different back end files in the system.

## How to Run

To run the solution, you need to ensure that the RAC project is set as the startup project in the solution. The solution also contains a project called “RAC Tests” which contains all of the unit tests that have been written for our BLL layer.

# Standards

Following standards presents several benefits to the development of a system. To start off, it allows us to have a uniform look to all source code. This allows for easier parsing by the developers or maintainers. There is no chance of a surprise line-wrap, or an unexpected variable name that requires investigation. Consequently, this makes the hunting of bugs not just easier, but faster.

## Coding Standards

The coding standards that we use in the RAC system are heavily based off of various code style guides provided by various organizations. Our standards vary depending on the language being used. The following are the standards that we use:

* C#, LINQ: [C# Coding Conventions](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions), by Microsoft
* JavaScript: [Google JavaScript Style Guide](https://google.github.io/styleguide/jsguide.html), by Google
* HTML, CSS: [Google HTML/CSS Style Guide](https://google.github.io/styleguide/htmlcssguide.html), by Google
* SQL (if needed): [SQL Style Guide](http://www.sqlstyle.guide/), by Simon Holywell

In addition to these standards, we have made some minor modifications that overrule anything found in these four web pages.

* Indentation will be **four spaces** for new block-level elements (methods, functions, if, loops, etc.)
* Indentation will be **eight spaces** for line wrapping
* Line wrapping will happen at **100 characters.**

Our system has a ReSharper style guide that can be found in the root of the solution folder called [RAC.sln.DotSettings](http://cstfs.cegep-heritage.qc.ca:8080/tfs/RACv2/RACApplication/_versionControl#path=%24%2FRACApplication%2FRAC%2FRAC.sln.DotSettings&_a=contents).

The ReSharper style XML file is also embedded here for you to view:



## Organizational Standards

Our organization standards build on top of the default MVC directory structure built by Visual Studio 2015. All of our files – with the exception of Visual Studio made files such as App\_Start and App\_Data follow UpperCamelCase naming. The C# files in the system also follow the same UpperCamelCase naming standard, however the JavaScript and CSS break off from this to keep standard with bootstrap and jQuery file naming. The JavaScript and CSS files both follow a lowercase-with-dashes naming standard. This is to ensure our naming standard matches up with jQuery and bootstrap so we don’t need to rename any of those files.

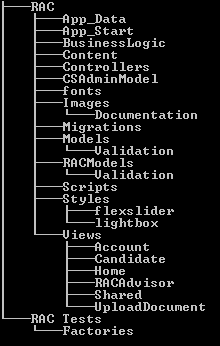


Figure 6: Folder structure for our solution.

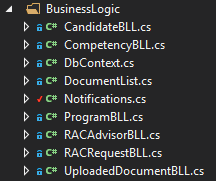


Figure : C# file naming standards

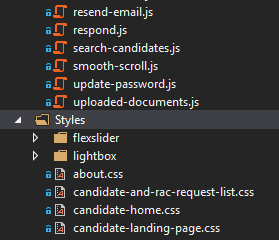


Figure 8: JavaScript and CSS file naming standards