October 5, 2017

Richard Chan  
Heritage College  
RAC System

Dear Mr. Chan,

In this following memo, I will be describing the technical environment for the Computer Science Department standard for the development of the RAC system. It will describe: the **technical architecture to be used for ASP.NET**, the approaches for using the **Entity Framework**, and the **options for testing**.

There are two (2) main approaches to developing applications in *ASP.NET*. The first system is .*NET* *Web Forms*, and the second is the *Model, View, Controller (MVC)* framework. *Web Forms* are easy to deploy and understand, using drag & drop functionality, and a lot of event-driven code that abstracts the inner workings. *Web Forms* are older, which means most, if not all, .NET developers have experience working in a *Web Forms* environment. However, *Web Forms* only allows a single form per page, and restricts the frameworks the developer has access to. Lastly, the *Web Forms Life Cycle* is complicate is complicated and esoteric. MVC mitigates the negatives of *Web Forms* (many forms per page, allows using other frameworks built with *PHP, Python, Ruby,* etc.). Compared to *Web Forms*, *MVC* is relatively stateless, and minimizes the usage of *Session* storage. *MVC* logically separates the functionality of the application into one of its three core function, which leads to a cleaner code base. It also allows for cleaner connections to DBMS. However, *MVC* is still relatively new, and support isn’t as available. It also requires more knowledge of *HTML/CSS*, as opposed to the drag n’drop nature of *Web Forms.* Based on the heavy database usage for the RAC system, I would recommend choosing the *MVC* approach. Cleaner and more modern code leads to an easier to maintain codebase.

There are three (3) approaches to using the *Entity Framework* model. The first is: c*ode-first,* in which you create the classes for your entities, and the *EF* generates tables matching those classes inside your database. Secondly, there is *data-first*, in which you create your database tables inside the database itself, and the *EF* generates the classes that match those entities. Lastly, there is *model-first*, in which you design the model for the database, and *EF* generates both the classes **and** the database tables to match your model. For our system, *code-first* is the best option. Our development team has the most experience programming as opposed to maintaining a database, so it would play better to our strengths to go *code-first*, and allow the *EF* to abstract the database portion away from us.

In regards to testing, using *MVC* already makes our testing much more simple. The abstraction between the three components means we can easily point the application towards dummy versions of the components. Built in *Visual Studio* UnitTests are our best bet for regular unit testing.

These are my choices for the system. *MVC* with *Code-First Entity Framework*, and basic unit testing. This means we will have a maintainable, and modern codebase that is easily abstractable.

Warm regards,  
 Maxwell Haley