Recommendations for RAC system design

Cegep Heritage college | CITE DES JEUNES

ASP.NET, Entity Framework and testing

Philip dumaresq

2017

Contents

[Introduction 2](#_Toc494992139)

[Technical architecture for asp.net 2](#_Toc494992140)

[webforms 2](#_Toc494992141)

[mvc 2](#_Toc494992142)

[Recommendation 3](#_Toc494992143)

[Entity framework 3](#_Toc494992144)

[Model first 3](#_Toc494992145)

[Data first 4](#_Toc494992146)

[Code first 4](#_Toc494992147)

[Recommendation 4](#_Toc494992148)

[Testing 4](#_Toc494992149)

[Conclusion 5](#_Toc494992150)

[References 5](#_Toc494992151)

# Introduction

The purpose of this report is to discuss technical recommendations for the development of the RAC system. The report will cover the system design architecture, covering ASP.NET WebForms and the MVC framework. It will then cover an analysis of entity framework as an ORM for accessing the data and recommend an approach to use for designing in this framework. Finally, the report will cover different options for testing the system. Unit testing of any business logic will be covered, as well as the possibility of doing automated UI testing.

# Technical architecture for asp.net

In ASP.NET there are two approaches commonly used for building web application – ASP.NET WebForms and the MVC framework approach. There are several advantages and disadvantages to each one that will be covered.

## webforms

ASP.NET Web Forms is an approach that utilizes ASPX to create HTML-like documents that each have C# server pages attached to them. Web forms is a very simple model and makes accessing your elements very easy. Web forms allows you a good deal of freedom when designing as well because you don’t have to conform to the use of MVC or the Entity Framework, which will be discussed late on. It allows you to create your own data access layer and define all your business objects yourself, which gives you a lot of freedom in terms of where you want chunks of your code to be. You also have the freedom to ultimately decide how you want to organize your solutions. You can choose different tiered architectures to use, whether you have it all in one project or if you want to bring in external DLLs with code from other compiled solutions. This also brings in some disadvantages though because you need to enforce the organization of your code a lot more. You need to establish a lot more standards to follow for your system as well.

## mvc

The MVC framework in ASP.NET is a framework used to break your code down into 3 separate logical roles. The first layer in the MVC pattern is called the Model, which controls all of your business logic. Then there’s the View, which controls your display, or what your end user is actually seeing. Finally, you have the controller, which handles any user input and things such as the firing of events from the user doing things in the front end. The MVC model provides a very power alternative to Web Forms and makes the life of the developer much easier. One big benefit to MVC is that is it goes back to HTML and not ASPX that web forms uses. Because all your elements aren’t run on the server side now the way they are in Web forms it allows you to create much more dynamic web pages. Another benefit is because MVC uses the entity framework there’s a big abstraction between your database and your code. However, this also results in your code being much more tightly coupled to the two frameworks, which gives you less flexibility to change architectures in the future should you ever chose to do so.

## Recommendation

For the RAC system, I would recommend using the MVC framework. One of the big user requirements gathered was that the system must be user friendly, and MVC supports that a lot more by creating dynamic web pages. Giving the user just that slightly smoother user experience could provide a lot of value to the system. The college standard also locks us into using the Entity Framework, which ties together with MVC very well, which makes our lives as developers much easier and allows the team to focus on the key things for the project and not need to worry as much about how some of the data is handled. This system is also going to need to be maintained by future students, and so developing this system with that in mind, there will likely be less surprises for future students going with an MVC approach since they’ll won’t need to dig through our layers understanding how everything works.

# Entity framework

When architecting your data using the Entity Framework there’s 3 different approaches that you can take. The first approach is a model first approach. This is where you begin by designing the data model and then generating a database and classes out of that model. Another more traditional approach is a data first approach, where you begin by creating your data tables and then generating a data model and your classes out of that. Finally, there’s the code first approach, where you code your classes and then generate data tables and a data model out of those classes.

## Model first

The model first approach is the way that our class first learned how to do it. You start by defining what the tables that you need are, then defining the attributes or columns of each of those tables, then defining the relationships between tables, then solving any traps and normalizing the database. The model first approach is good because it gives you time to make sure that you have a good database and it graphically shows all of your dependencies and relationships between tables. You can figure out paths through your data much more quickly before ever coding it and noticing flaws. However, it’s also a disadvantage because every time you need to make a change you need to update your data model first, and then update all of your tables, which is slightly more difficult to keep in sync with your database then it is doing a data first or code first approach. With a mode first approach you also have to design your database graphically with a UI and not from being in the code, which a lot of people dislike. A model first approach is good for prototyping a database, but implementing isn’t the best approach for actually implementing the database.

## Data first

A data first approach is really good because you can very easily make changes to the database that you’ve generated. All you need to do to make a change is run a couple alter table commands and you don’t have to worry about updating anything else. If your code is designed well, then your code shouldn’t break by adding or removing any columns from your database, although it might not react will to removing an entire table. Once you’ve made changes to your data, reflecting those changes is as simple as regenerating your classes and you don’t need to worry about anything else. One downside the data first approach is that you need to have a team that’s really strong in databases, which is harder to find than a team of good coder. Data first approach is the most efficient and arguably most powerful way of doing it, but you need people who really know databases.

## Code first

A code first approach is definitely the simplest approach. Designing a series of classes and then generating data tables out of them is very quick to do and it will get your system up and running faster than either other approach. With a code first approach you also don’t need to be as good with databases because the Entity framework will take care of a lot of the work for you. You need a team of people who are good at coding and don’t need to worry about how the database is working quite as much. One disadvantage to a code first design is that you don’t have the ability to customize the database quite as much as you do with a data first approach. You don’t get as much control over keys and relationships. However, with a code first approach you also know that it will work, even if it’s not very robust and it is kind of sloppy.

## Recommendation

Our team is full of very strong coders, and I think there’s only one person on the team whose particularly comfortable with designing the database from a data first approach. Because of this, I think that the best recommendation for our team is to use a code first approach. This will simplify the defining of our data and make changing our data much easier if we need to modify our tables.

# Testing

For testing, the main approach we need to take is to do automated unit testing. Automated unit testing ensures that all of our business logic is properly functioning and that we’re always getting proper data back from the database. To make our development go smoother and to try and avoid ever encountering major bugs in the system, the system should follow a test driven development where we’re constantly cycling through development of business logic and business objects and writing test cases for them. This way we can keep the development of the system running smoothly and can make sure a method Is working entirely before including that functionality into the system.

If possible, it would also be nice to write UI tests for this system. UI testing is very hard to do however and there’s no good tools for doing it. However, since this system is going to be used by a very wide variety of different types of people with different backgrounds and different levels of skill using computers, we don’t know the kind of things they could do to potentially break the UI. So if possible, automated UI testing would also be of huge value to this system.

# Conclusion

In conclusion, this report has gone over the advantage and disadvantages of several different approaches to the design of this system. The different approaches discussed are: the ASP.NET Webforms vs the MVC framework, concluding that the MVC approach would be best suited for this system; model first, data first and code first approaches to defining your data, deciding that code first would be the most suitable approach for this system and for the team; and finally different approaches to testing and discussing that unit testing and a test driven development should be followed, and if possible it would be very beneficial to include automated unit tests.

# References

<https://en.wikipedia.org/wiki/ASP.NET_MVC>

<https://www.asp.net/mvc>

<https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started>

<http://www.tutorialsteacher.com/mvc/asp.net-mvc-tutorials>

<http://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx>

<https://www.itworld.com/article/2700195/development/3-reasons-to-use-code-first-design-with-entity-framework.html>

<http://www.entityframeworktutorial.net/model-first-with-entity-framework.aspx>

<http://www.entityframeworktutorial.net/database-first-with-entity-framework.aspx>