|  |
| --- |
| Mercer university |
| Project 2 |
| SSE 657 - Object Oriented Project Methods |
|  |
| **Tanya Do & John Robison** |
| **10/27/2014** |

|  |
| --- |
|  |

Table of Contents

[Problem Statement 4](#_Toc401694488)

[Discussion With Customer 4](#_Toc401694489)

[Commonality Comparison of System 4](#_Toc401694490)

[Features List 4](#_Toc401694491)

[Domain Analysis 4](#_Toc401694492)

[Use Case Diagrams 4](#_Toc401694493)

[Design Pattern Selection 5](#_Toc401694494)

[Determining the Architecturally Significance of Components 6](#_Toc401694495)

[Risk Analysis and Reduction 6](#_Toc401694496)

[Use of Design Principles 6](#_Toc401694497)

[Mock Scenarios of System Interaction 6](#_Toc401694498)

[Appendix A: References 7](#_Toc401694499)

**Table of Figures**

[Figure 1: MVC Interaction Diagram 5](#_Toc401694500)

# Problem Statement

With the opening of SparkMacon, Macon's own Maker Space, there is now a need for a web service that will allow makers of all trades and skill sets to advertise their products and promote their business. This service will allow makers to create a profile that contains: a biography describing their craft, previously completed or sold projects, products currently for sale, user reviews of the maker's products, and a place for users to request unlisted or new products. The people behind SparkMacon are devoted to building a strong maker community in the Macon area and want a system that helps to push their makers further. To build this community that SparkMacon aspires to, the web service will need features that promote community involvement: featured makers (i.e. Maker of the Week), community forums, the ability to like or follow products and users, etc.

# Discussion With Customer

# Commonality Comparison of System

# Features List

# Domain Analysis

# Use Case Diagrams

# Design Pattern Selection

The designed system will be built on the ASP.NET MVC Framework. The use of this framework utilizes the MVC (Model-View-Controller) design pattern which encourages designs to be loosely coupled through a separation of concerns. This separation is achieved by using the three main components: Models which handle the data and logic of the system, Views which display the data in a way that makes sense to the user, and Controllers which are responsible for dealing with user inputs by updating the models and view. Below in Figure 1 is a diagram that visualizes the interactions between these components.



Figure : MVC Interaction Diagram

The ASP.NET framework builds on the MVC design pattern by relying on a "convention over configuration" approach which reduces the amount of code required for a project. This approach requires certain design decisions, such that items are placed in the correct locations (i.e. Controllers must be in the Controller folder) and that they are named appropriately (i.e. Controllers will be named ending with "Controller"). Not only does this convention based approach reduce code requirements, it also aids in overall readability of the system.

Due to the loose coupling offered by the MVC design pattern, our team chose ASP.NET MVC as the framework for this system because it allowed for code reuse and parallel development. Since the models, views, and controllers are separated, they can easily be implemented into another application. An example of this would be if the customer requested a mobile application since the model and controller classes could be reused. The only major change in this example is that new views would have to be created specifically for the mobile platform. This separation also allows components to be designed and built separately. One developer can build the store user interface in the views while another developer designs the underlying business logic in the models.

In addition to all of the benefits of MVC's separation of concerns listed above, our team decided to develop with the ASP.NET MVC Framework for its use of .NET languages (C# and Visual Basic) which we are familiar with as well as the use of its easy to learn view engine, Razor. C#, our .NET language of choice, is extremely powerful and easy to develop with due to its number of available libraries and the use of Microsoft's Intellisense, which can auto-fill code for the developer such as variable and method names. The Razor view engine is a combination of HTML and either C# or Visual Basic, which makes it very natural for .NET developers to write and learn. The use of C# in the markup allows for very powerful dynamic web pages. Razor also allows the use of layouts which enables developers to have a single Razor file act as a template for all other views. These layouts reduce duplicate code by encapsulating common view elements into the layout. The final advantage of Razor that will aid in development is the ability to use Intellisense to quickly write the markup and code contained in the Razor files.

# Determining the Architecturally Significance of Components

# Risk Analysis and Reduction

# Use of Design Principles

# Mock Scenarios of System Interaction

# Appendix A: References

* McLaughlin, Brett, Gary Pollice, and David West. *Head First Object-oriented Analysis and Design*. Sebastopol, CA: O'Reilly, 2007. Print.
* "Easy Intro to ASP.NET MVC." *Easy Intro to ASP.NET MVC*. N.p., n.d. Web. 21 Oct. 2014. <http://www.beansoftware.com/ASP.NET-Tutorials/Intro-ASP.NET-MVC.aspx>.