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

|  |
| --- |
|  |

Table of Contents

[Introduction 1](#_Toc402204988)

[Problem Statement 2](#_Toc402204989)

[Discussion With Customer 2](#_Toc402204990)

[Commonality and Variability Analysis of System 4](#_Toc402204991)

[Features List 5](#_Toc402204992)

[Use Case Diagram 8](#_Toc402204993)

[Domain Analysis 9](#_Toc402204994)

[Design Pattern Selection 10](#_Toc402204995)

[Determining the Architectural Significance of Components 11](#_Toc402204996)

[Risk Analysis and Reduction 13](#_Toc402204997)

[Additional Discussion With Customer 14](#_Toc402204998)

[Design of Core Features and Use of Design Principles 15](#_Toc402204999)

[Home Page 17](#_Toc402205000)

[User Accounts 19](#_Toc402205001)

[Product Listings 19](#_Toc402205002)

[Mock Scenarios of System Interaction 23](#_Toc402205003)

[Account Registration 24](#_Toc402205004)

[Editing Account Information 25](#_Toc402205005)

[Purchasing a Product With an Account 28](#_Toc402205006)

[Purchasing a Product Without an Account 33](#_Toc402205007)

[Appendix A: References 35](#_Toc402205008)

[Source Code 36](#_Toc402205009)

[AccountController.cs 36](#_Toc402205010)

[CheckoutController.cs 44](#_Toc402205011)

[HomeController.cs 45](#_Toc402205012)

[ShoppingCartController.cs 46](#_Toc402205013)

[StoreController.cs 48](#_Toc402205014)

[AccountModel.cs 49](#_Toc402205015)

[Cart.cs 50](#_Toc402205016)

[Category.cs 51](#_Toc402205017)

[MakerStoreEntities.cs 51](#_Toc402205018)

[Order.cs 52](#_Toc402205019)

[OrderDetail.cs 53](#_Toc402205020)

[Product.cs 53](#_Toc402205021)

[SampleData.cs 54](#_Toc402205022)

[ShoppingCart.cs 55](#_Toc402205023)

[ShoppingCartRemoveViewModel.cs 59](#_Toc402205024)

[ShoppingCartViewModel.cs 59](#_Toc402205025)

[\_ChangePasswordPartial.cshtml 59](#_Toc402205026)

[\_ExternalLoginsListPartial.cshtml 60](#_Toc402205027)

[\_RemoveExternalLoginsPartial.cshtml 60](#_Toc402205028)

[\_SetPasswordPartial.cshtml 61](#_Toc402205029)

[ExternalLoginConfirmation.cshtml 61](#_Toc402205030)

[ExternalLoginFailure.cshtml 62](#_Toc402205031)

[Account/Index.cshtml 62](#_Toc402205032)

[Login.cshtml 63](#_Toc402205033)

[Manage.cshtml 64](#_Toc402205034)

[Register.cshtml 64](#_Toc402205035)

[Update.cshtml 65](#_Toc402205036)

[AddressAndPayment.cshtml 66](#_Toc402205037)

[Complete.cshtml 67](#_Toc402205038)

[About.cshtml 67](#_Toc402205039)

[Contact.cshtml 68](#_Toc402205040)

[Home/Index.cshtml 68](#_Toc402205041)

[\_Layout.cshtml 69](#_Toc402205042)

[\_LoginPartial.cshtml 70](#_Toc402205043)

[Error.cshtml 71](#_Toc402205044)

[CartSummary.cshtml 71](#_Toc402205045)

[ShoppingCart/Index.cshtml 71](#_Toc402205046)

[Browse.cshtml 73](#_Toc402205047)

[CategoryMenu.cshtml 73](#_Toc402205048)

[Details.cshtml 73](#_Toc402205049)

[Store/Index.cshtml 74](#_Toc402205050)

**Table of Figures**

[Figure 1: The SparkMacon Logo 2](#_Toc402205051)

[Figure 2: MakerStore Use Case Diagram 9](#_Toc402205052)

[Figure 3: MVC Interaction Diagram 10](#_Toc402205053)

[Figure 4: Home Page Razor Code Segment 18](#_Toc402205054)

[Figure 5: The \_Layout.cshtml Header 18](#_Toc402205055)

[Figure 6: The \_Layout.cshtml Body and Footer 19](#_Toc402205056)

[Figure 7: The Product Model 20](#_Toc402205057)

[Figure 8: The Shopping Cart Summary Method 21](#_Toc402205058)

[Figure 9: The Shopping Cart PartialView Code 21](#_Toc402205059)

[Figure 10: The Shopping Cart Migration Method 21](#_Toc402205060)

[Figure 11: ShoppingCart Index Method 22](#_Toc402205061)

[Figure 12: MakerStoreEntity.cs 22](#_Toc402205062)

[Figure 13: SampleData.cs (Categories and Users) 23](#_Toc402205063)

[Figure 14: SampleData.cs (Products) 23](#_Toc402205064)

[Figure 15: Home Page of the MakerStore 24](#_Toc402205065)

[Figure 16: The Register Button 24](#_Toc402205066)

[Figure 17: The Registration Page 25](#_Toc402205067)

[Figure 18: The Username After Creating an Account 25](#_Toc402205068)

[Figure 19: The Log In Button 26](#_Toc402205069)

[Figure 20: The Log In Screen 26](#_Toc402205070)

[Figure 21: The User Account Page 27](#_Toc402205071)

[Figure 22: The Account Information Edit Form 27](#_Toc402205072)

[Figure 23: Updated User Account Page 28](#_Toc402205073)

[Figure 24: The Store Button 28](#_Toc402205074)

[Figure 25: The Store Page 29](#_Toc402205075)

[Figure 26: The Product Listings Page 29](#_Toc402205076)

[Figure 27: The Product Listing Page 30](#_Toc402205077)

[Figure 28: The Add To Cart Button 30](#_Toc402205078)

[Figure 29: The Shopping Cart Page With Updated Cart Icon 31](#_Toc402205079)

[Figure 30: The Checkout Button 31](#_Toc402205080)

[Figure 31: The Checkout Screen 32](#_Toc402205081)

[Figure 32: The Checkout Confirmation Screen 32](#_Toc402205082)

[Figure 33: The Registration Page With the Updated Cart Icon 33](#_Toc402205083)

# Introduction

This report and the project that it corresponds to are intended to display mastery of the concepts contained in Chapters 6 - 8 of the book Head First Object-Oriented Analysis and Design by McLaughlin, Pollice, and West. These chapters build on the previous chapters by explaining how those Object-Oriented Analysis and Design (OOAD) concepts can be applied to larger, real-world problems to make them more manageable to design.

The first step offered by the text to make a large problem less intimidating to approach is to break the project into smaller, logical pieces that can be developed using the OOAD concepts provided in the previous chapters. To determine these logical pieces, techniques such as conversations with the customer and commonality and variability analysis can be used to determine the features of the system. Once the features are determined, the most architecturally significant features are determined using the three Q's of architecture and are then designed first to reduce risk. After the architectural design of the system is completed, design principles are applied to ensure that the system is not rigid, fragile, or immutable.

# 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



Figure : The SparkMacon Logo

Spark Macon reached out to us, J & T Development, Inc., to create a fully functional web store and market place web application. They sent us an email expressing their need for a web application that will contribute to and serve the Spark Macon community. In order to fully understand the needs of Spark Macon, J & T Development decided we needed to sit down with them to talk about the minutia of the project. We scheduled a meeting with Spark Macon representatives, Michael Rose and Bob Martin. We discussed the scope of the project and the features that the project would have.

**J & T Development, Inc.:** Hey guys. We saw your email and are really interested in what you are doing. Can you tell us more about your company and what it is that you want us to do for you?

**Michael Rose:** So Spark Macon is a makerspace and will have different types of people creating things, and innovating.

**Bob Martin:** Like artists and tech people.

**Michael Rose:** Yeah, and we want to help expand the local economy by setting up a place for our members to sell their products.

**Bob Martin:** But since we are a makerspace and not a flea market, we want this to take place online. But we still want to support our members.

**J & T Development, Inc.:** Okay, cool. So like, you want a web application that works like EBay?

**Bob Martin:** Sort of. We want people to be able to buy and sell. But the things must be handmade, like created through Spark Macon.

**Michael Rose:** Yes. More similar to the Etsy web application. Which stipulates that all items sold must be handmade.

**Bob Martin:** We also want this site to be specific to Macon, though.

**J & T Development, Inc.:** Okay, great. So what type of features does this application need?

**Bob Martin:** It definitely needs to have individual user accounts. We want for each user to essentially have their own mini store on the application.

**Michael Rose:** Right, I agree. Additionally, the user profile should have an "about me" section. Where they can describe themselves and their philosophies to potential buyers.

**Bob Martin:** Also, we'd like for users to have to capability of rating users and their products.

**Michael Rose:** Yes. We'd like to have some standard of integrity for sellers and buyers.

**Bob Martin:** Also, we should be able to allow potential customers to request types of items they'd like to see in the future. But I don't know. Maybe that is a bonus feature. And not something we absolutely need. But something that would be pretty cool.

**Michael Rose:** I think another feature we'd like is a products page for each profile. Where the user can display what they are selling. With pictures and the listed price. And we can also have a product review on there as well.

**Bob Martin:** And we also want a home page where we can have feature sellers.

**Michael Rose:** And products too!

**Bob Martin:** Yes. We want to create an online community as well. So we'd like to have an about me section describing our mission as Spark Macon as well as a blog type section where we keep our community updated on things going on. Like a news section.

**J & T Development, Inc.:** Yeah, that is definitely do-able. We will have to do some internal evaluation about how to approach the project from a technical standpoint. We will get back to you after our analysis.

**Bob Martin:** Thank you so much for your time. We are excited to work with you on our new venture.

**J & T Development, Inc.:**  Yeah, definitely. We are so glad that you chose us to join you on your project.

# Commonality and Variability Analysis of System

In order to ensure that this web service meets all of the customer's expectations, our team performed a commonality and variability analysis on the system. This means that we compared the proposed systems to preexisting web services and described what these systems offered or did that the customer either wanted their system to be like or not be like. In addition to ensuring that the designed system meets the customer's expectations, this commonality and variability analysis will aid our team in determining the major features of the system which will allow us to determine the most architecturally significant components.

Based on the discussion with the customer, the core functionality of the web service should be much like Etsy, an e-commerce marketplace for creators of unique items to sell their goods. The customer stressed that they wanted posted items to be actually made by the users that posted the listings, so the web service should not be like Amazon or Ebay. Since the customer is interested in strengthening the Macon community, the system should not be international or even open to anyone like Etsy, Amazon, and Ebay are. The area-based product listing system used by web services such as Craigslist is ideal for building this strengthened community of Macon makers.

# Features List

After our domain analysis we have compiled a list which conveys the complete application in a succinct and easy to digest manner. Listed below are the features that were determined as well as the requirements that make up each feature:

* User Accounts
  + User biography
  + List of products available
  + List of previously sold products
  + Product Requests
  + User Rating/ Review
* Product Listings
  + Product description
  + Product Category
  + Price
  + Product Rating/ Review
* Home Page
  + Maker of the Week
  + Featured Products
  + News
  + About
  + Link to Spark Macon
* Community Involvement
  + Community forum
  + Event coordination/ planning
  + Liking or following products or users

The User Account feature will be responsible for storing all of a user's information: biography, products, user rating, and requests. A user will be able to create an account at any time and begin selling and marketing their products. Once an account is created, the user will be able to login to the system and edit/ update their account information as well as create postings for products.

* User biography: The user biography is a place on the user’s profile where the user can talk about themselves and/or their store. The purpose of this section is so the members of SparkMacon can express themselves and connect with their fellow members.
* List of products available/Webstore: Each profile will have the option to be associated with a list of products. Those members who want to sell as well as buy, will have the option to open a personal store. The products can be categorized and sorted by price.
* List of previously sold products: There will be a section where a seller’s sold products will be listed. Unlike the list of products, previously sold products will be organized chronologically by date sold.
* Product Request: This feature allows customers to submit request for products they want to see in the future. This will allow for the sellers to have a better grasp of what their customers want, and what to produce. And this will allow for interaction between sellers and customers.
* User Rating/Review: The application will allow for users to rate users as a seller or buyer, and provide a review of the person’s habits. The purpose of this is to provide transparency between members of the community.

The Product Listing will contain all of the information for a user's product: product description, category, price, and product reviews. A user with a registered account will be able to post listings for their products. The use of the description and category will allow other users to search for products that interest them, such as robotics or art.

* Product description: Each product will have its own section or page where the seller can upload a picture of the product and fully describe the product offered.
* Product category: Each product can be assigned to a product category. On the seller’s webstore page, the products offered can be sorted by product category. Additionally, the products can be filtered by product category.
* Price: Each product will have a price associate with them.
* Product Rating/Review: Users will be able to provide ratings and reviews of products they buy from a maker. The purpose of this is to encourage sellers to produce quality products and accurate description of the product sold.

The Home Page will be the main access point of the web service. It will be used to display important pieces of information, such as the featured Maker of the Week, upcoming events, and news about the makerspace. This page will also contain information about what SparkMacon's mission is as well as provide links to their website.

* Maker of the Week: We would like to feature a new Maker each week on the SparkMacon front page. This feature is to foster a community environment. And to help Makers who are talented in their chosen art, but less schooled in marketing and business.
* Feature Products: Additionally, the SparkMacon mission aims to bring light to products that are of high quality but not easily access via any search function. This feature also serves to foster a community environment.
* News: This section of the application will be similar to a blog. All news items will be organized chronologically. News items will be news about the SparkMacon events or announcements.
* About: This section will be explain the purpose of the SparkMacon Marketplace application and how it functions.
* Link to SparkMacon: There will be a fixed section that will link to the main SparkMacon website.

There will be several components that will aim to build community involvement, including user forums, announcing local makerspace events, and the ability to like or follow other users' accounts and products.

* Community forum: There will be a place for discussion between members of SparkMacon. The forum will allow members to communicate with each other about anything maker related. Additionally, users can use this space to discuss marketing techniques.
* Event coordination/planning: This section will be open for users to plan SparkMacon events remotely.
* Liking or following products or users: SparkMacon users have the option to like or follow products. This allows for users to create a wishlist of sorts and also get notifications whenever a seller puts up a new product.

# Use Case Diagram

The users should be able to interact with the system in many different ways. These interactions include:

* Creating and editing accounts
* Creating and editing product listings
* Purchasing products
* Contributing on forums
* Navigating to SparkMacon's website
* Rating users and products.

The account interactions will be handled through registration and log in processes. A new user will have to register with the system in order to have a username and password that they will be able to log in with. Once they are registered, they will be given the ability to edit their account information, such as their biography and available products.

Only registered users will be able to create and edit product listings. Once a user is registered, they will be able to post as many products as they have to sell. Once a product is listed, only the user who posted it will be able to make edits to the listing. These edits include the product's description and price.

Registered users will also be able to contribute to the community by rating other users and their products as well as commenting on forums. If a user is interested in learning more about the SparkMacon makerspace or learning about upcoming events, they will be able to click a link that navigates to the SparkMacon site regardless of if the user is registered.

Figure 2 below visualizes the users' interactions with the system in a UML Use Case Diagram.

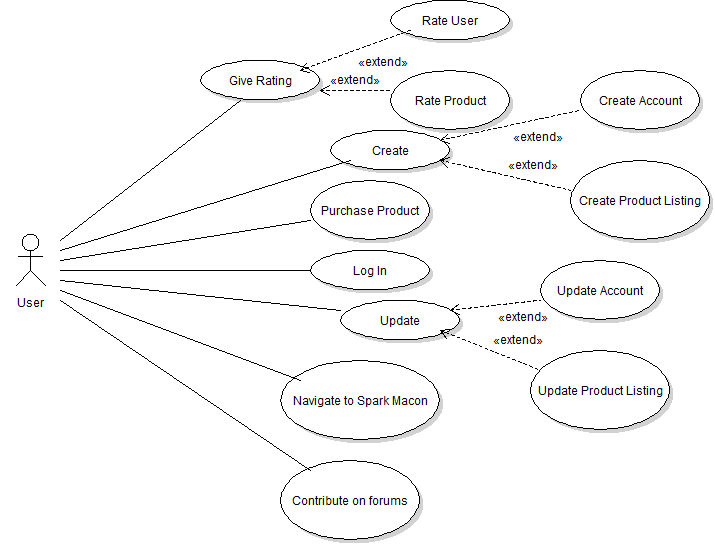


Figure : MakerStore Use Case Diagram

# Domain Analysis

We at J & T Development, Inc. want to be as transparent as possible with our clients. So while we have a detailed plan, use case and design pattern selection in place, we want to reframe the requirements of the web application in a way that our client, Spark Macon will clearly understand. J & T Development, Inc. pride ourselves on not only producing high quality products in an efficient manner, but also on our customer service and communication skills. We realize and understand, unlike our competitors, that our clients may not be able to fully digest documents such as use-cases and UML diagrams. So we go through a process of domain analysis where we study and analyze the system, and collect and organize the relevant information to the system. Then we represent it in a way that is easy to understand and digest for our clients. Our clients have on numerous occasion express appreciation for the way we present information to them. And it reduces confusion as well as ensure that both we and our clients are on the same page. And are talking about the same things.

We created a features list which allowed us to outline the application in its entirety, while expressing the functionalities of the application in a succinct manner.

Then we provide a detailed description of what each feature and sub-feature.

# 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 3 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 Architectural Significance of Components

To select which feature to begin with, our team looked at each feature and asked the Three Q's of architecture:

* Is it the essence of the system?
* What does the feature mean?
* How do you implement the feature?

A feature is deemed architecturally significant if it is either the essence of the system, the meaning of the feature is unclear to the developers, the developers are unsure of how to implement the feature, or any combination of the previous three. Neither of the Three Q's carries more significance than the other and thus asking these questions only gives us an unweighted list of the most architecturally significant features.

The features that have been determined to be the essence of the system are: the user accounts, the product listings, and the home page. User accounts are the core of the system. Without users, new content cannot be generated and Macon area makers will not be able to display their crafts. Just like a makerspace without makers is just an empty building, a makerstore without makers is just a blank website. Adding to this point, a store without items is just an empty building. This means that the product listings are also essential to the system. However, the system can still operate to a certain extent with only user accounts. Makers will still be able to describe their craft and promote themselves without the ability to post listing for their products. Finally, the home page acts as the main access point of the system. It will contain a description of the site, links to all of the other features, and a link to SparkMacon's website. This access point is especially important for users who are not familiar with what the system's goals are or the makerspace that is supporting it.

The product listing was a feature that our development team needed clarification on to get a better understanding of. Some of the components of this feature that our team is unsure of are: product categories, listing formatting, and whether the customer would want similar products to be listed. Since product categories should not be something that a user can add or remove, there needs to be a set list of available categories that a product can be. Our team will need to be given this list in order to develop the database that will store product information. Once the categories are determined, our team will need to know what format the product information should be presented in. This includes the layout of the web page as well as what information should be presented to users (such as description, posting user, or price). When performing the commonality and variability analysis of the system, our team noticed that almost every online store service implemented some form of a related products function. We will need to discuss this idea with the client to see if it would be an addition to the product feature that they would be interested in adopting.

The user account was a feature that our development team was unsure of how to implement. Since our team does not have experience with handling user authentication and protection, it is a feature that we will need to thoroughly research, experiment with, and test. User security is very important because user accounts will contain personal information, such as credit card information. Aside from the security aspects, the process for creating an account must be intuitive and not overly intrusive. Our team will research the most effective ways to facilitate the process of creating and managing accounts and account information.

After analyzing these Three Q's, our team determined that the most architecturally significant features of the system were: the user account , the product listing, and the home page. The community involvement functionalities were not deemed architecturally significant for the following reasons:

* These functionalities were regarded as nice-to-haves and thus non-essential to the system
* The meaning of the functionalities (such as the forums) were clear and obvious
* The implementation was well within our team's capabilities

Determining these architecturally significant components narrowed down the list of features to help our team find an appropriate starting place that would help to reduce the overall risk of the development.

# Risk Analysis and Reduction

With the architecturally significant features determined, our team needed to determine the order in which we developed each feature in such a way that will reduce risk. Since it was considered to be the main access point of the system, we decided to develop the home page first. Having this feature developed will allow to have a place to build off of for the remaining features. This main access point will also act as the "glue" that holds all of the remaining features together, so having this feature in place first will reduce the risk of subsequent features not merging together properly.

After the home page is set up, our team will work on the user account functionality. This will include the registration, log in, and account information functions. This feature was selected to be the next task because it enables users of the system to be able to fully interact with all of the components inherent in the system. It is also is required to have a user account system set up before setting up the product listing system because a user account will be required to post a product listing. Once the account system is developed, it will need to be integrated into the home page in such a way that a user can easily find and access it's functionalities. This feature was also selected to be the second task because of the research and experimentation required by our team to ensure a secure system. Keeping this task as close to the top of the priority list as possible will enable us to better estimate time requirements for this system which will allow us to stay on time and on budget.

Once the home page and user accounts have been developed, the product listings can be created. These listings must be available to view from both the home page and user account pages. Listings must also be creatable only by registered users and must only be editable by the registered user who created the product listing. This feature will require our team to further discuss with the client about the layout and format of the listings, so we will want to keep this task as a priority to mitigate any risk of delays or discrepancies due to miscommunications between our team and the customer.

Following these steps will ensure that risk is kept to a minimum in both design and implementation. The ordering of the tasks will be as follows: developing and designing the home page, developing and implementing the user accounts, and developing and implementing the product listings. For the reasons listed above, this task ordering will mitigate much of the risk inherent in the development and design of this system and help to ensure that our team remains on time and on budget. This risk mitigation will also allow our team to determine any potential bottlenecks or complications that we may run into in the development timeline which will allow us to plan in advance for how to deal with them.

# Additional Discussion With Customer

Michael Rose and Bob Martin from SparkMacon got to thinking, and had more ideas for their SparkMacon MakerSpace MakerStore. They called us up and wanted to discuss their new ideas.

**J & T Development**: Hey guys, what's going on? What is that you want to add to the Maker Store website?

**Michael Rose**: So we were drinking coffee at Taste and See downtown.

**Bob Martin**: On Poplar Street.

**Michael Rose**: Right. On Poplar Street. And we saw that they have really nice art work hanging on the

walls.

**Bob Martin**: The art is all done by local artists around town. And you have the option to buy any of the art work hanging around the store.

**Michael Rose**: So we starting thinking about how cool that was. And so we want to add an element of that to the website

**J & T Development**: Okay, we can help you out. What is it that you want us to do to the application?

**Michael Rose**: Well you see. We'd really like for the users of our website to focus on the actual piece itself.

**Bob Martin**: Yeah. We don't want to present our self as just a store, but also a community.

**J & T Development**: Well one way we can do this is to make sure that the photo of the item for sale is very large and featured very prominately on the sale's page.

**Michael Rose**: That sounds great.

**Bob Martin**: We also want the makers to be able to organize their wares appropiately.

**Michael Rose**: Right. Not just a huge list of product.

**Bob Martin**: But like in categories, on the side. Where people can filter by categories. Or sort all the items by the categories.

**J & T Development**: Is assigning an item to a category required?

**Michael Rose**: I'm not sure what you mean?

**J & T Development**: Well, what if makers have products that don't fit into a category. Or if they don't want to set up categories at all.

**Michael Rose**: No, we definitely want a category assigned to each and every product.

**Bob Martin**: The website can just have a default category, that houses everything until the maker moves the item out of it. We'd like for the customers to search by category, even if it just "default" or "miscellanous".

**J & T Development**: Okay, cool. We can definitely do that for you.

**Michael Rose**: We'd also like for the product page to feature more than just one picture. Of maybe like a slideshow of pictures of the product from different angles?

**J & T Development**: Yeah. No problem.

**Bob Martin**: Thanks. That's great.

**J & T Development**: Anything else?

**Michael Rose**: Yeah, we want to list the products by picture form instead of words.

**Bob Martin**: Like a thumbnail of the product.

**Michael Rose**: We also want the name of the product and just a brief, or begining of the description of the product.

**Bob Martin**: We'd like for the product listing to be organized similarly to the Etsy website. But more geared towards maker stuff. Art and technology.

**J & T Development**: Gotcha.

**Michael Rose**: Also, have similar products show up together on the "home page" of the product site. Where it is listing all the products by all of the makers.

**Bob Martin**: We also, want for the product page to allow people to "like it" or "star" it. So they can come back to it later.

**Michael Rose**: And for the makers to know that their work is appreciated, even if people do not buy their wares.

**J & T Development**: Definitely. This is all do-able

# Design of Core Features and Use of Design Principles

While developing this system, our team will implement the use of design principles such as the DRY (Don't Repeat Yourself) Principle to ensure that our system is maintainable and flexible. The main principles that will be implemented are:

* The Open-Closed Principle (OCP)
* The Don't Repeat Yourself Principle (DRY)
* The Single Responsibility Principle (SRP)
* The Liskov Substitution Principle (LSP)

The Open-Closed Principle is about the changeability of the system. The definition of the Open-Closed Principle in the words of McLaughlin, Pollice, and West is that "classes should be open for extension, and closed for modification". Simply, this means that the components of the system should be designed in such as way that the system is flexible without needing to be changed. Example implementations of this principle include inheritance of an abstract class and the use of private methods.

The Don't Repeat Yourself Principle is about avoiding repeated code. The definition provided by McLaughlin, Pollice, and West is that "avoid duplicate code by abstracting out things that are common and placing those things in a single location". In other words, the DRY Principle is about ensuring that functionality is placed in a single, logical place so that changes to this functionality will not require changes in multiple places of the system. The most common implementation of this design principle is through the use of encapsulation. However, this principle is not limited to software design and can be used in gathering requirements to ensure that no two requirements address the same topic. The use of this principle ensures that a system is easily modifiable and flexible.

The Single Responsibility Principle is closely related to the Don't Repeat Yourself Principle in that it deals with keeping functionality in a single place. McLaughlin, Pollice, and West define this principle by saying that "every object in your system should have a single responsibility, and all the object's services should be focused on carrying out that single responsibility". This means that each class in a software system should be directly related to only one task. This makes each class only have one reason to change, which reduces the effects of a change to the system. Cohesion is an example implementation of this principle, therefore software that is highly cohesive (and thus loosely coupled) is following the SRP principle.

The Liskov Substitution Principle is about the appropriate uses of inheritance and knowing when not to use inheritance. The definition of the Liskov Substitution Principle given by McLaughlin, Pollice, and West is that "subtypes must be substitutable for their base types". This means that any class that inherits from another class should be able to use the base classes methods without causing any problems. Inheritance that doesn't follow the Liskov Substitution Principle becomes hard to understand which can cause issues with implementation of the subclasses. Some alternatives that can be used when inheritance is not appropriate are: delegation, composition, and aggregation. Delegation is where functionality from another class is used to accomplish a task as opposed to extending the used class. This is useful for when the needed functionality does not need to be changed to meet the designed goals. Composition is where your class is made up of other families of classes. This is useful for when the implementation of a class may change at runtime. A side effect of this alternative is that once the composing class is destroyed, all of the composite classes that it owns are also destroyed. The last alternative takes care of this side effect. Aggregation is like composition in that it uses other families of classes to have a dynamic implementation at runtime but the composite classes will still exist outside of the context of the composing class.

## Home Page

The home page is the main access point to the web service. It contains navigation links to all of the other components of the service, including the store and user accounts. The main links available from the home page are to:

* The Store
* The About Page
* The Contact Page
* The SparkMacon Page
* The Shopping Cart Page
* The User Account Page
* The Registration Page

Through the use of Razor's partial view feature, these links as well as links back to the home page can be made available throughout all other pages on the web site. Since the home page does not store any data, it does not require a Model class. There is a Controller class as well as Razor Views and partial views. The Controller class handles serving the different web pages. For the home screen, the Controller method simply returns the corresponding view which causes the Razor file associated with the Controller method to be displayed. A segment of the Razor code can be seen in Figure 4 below.

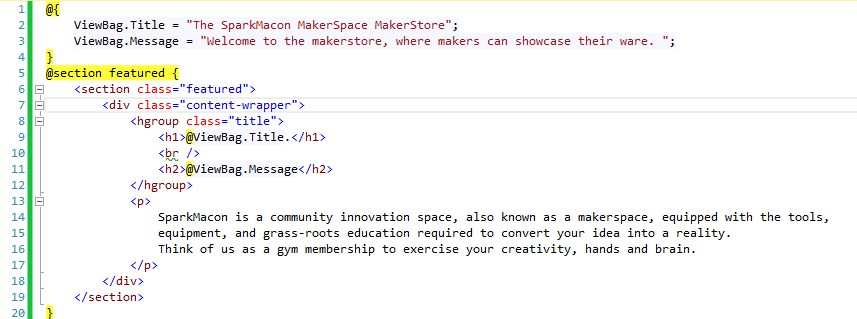


Figure : Home Page Razor Code Segment

Even though this code does not make mention of the navigation links at the top of the page explicitly, the links still appear. This is because ASP.NET MVC allows there to be an overarching layout called \_Layout.cshtml that defines all of the common views across all pages. The use of this layout file allows our development team the ability to place all of the common UI elements in a single place which follows the DRY Principle. Portions of this Razor file can be seen below in the following figures:



Figure : The \_Layout.cshtml Header

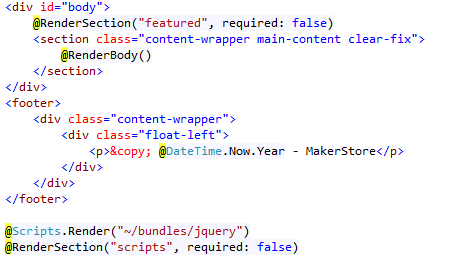


Figure : The \_Layout.cshtml Body and Footer

The Home Controller also contains the methods that reroute to the About and Contact pages. These Razor files contain mostly text and as such are not worth including in this report.

## User Accounts

Since our design team was unsure of how to implement the user account feature, we performed extensive research into methods of implementing this. It was very important that the implemented method was considered secure and nonintrusive. Ultimately, after researching various techniques our team discovered that ASP.NET MVC offers a template that adds in a user account feature. Much like how we used design principles to ensure our project was done in a proven manner, our design team decided to use this user account template to build off of. We added in additional features that our system required, such as biographies, but allowed the template to handle user registration, log in, and authentication. This template offered C# files to handle the backend, Razor files to handle the front end, and a database that would store the registered user's information. Having taken care of the user accounts early on and by discovering the user account template reduced the risk of the our development team falling behind schedule and over budget.

## Product Listings

The product listings will allow users to both post and purchase products. There is a Product Model class that stores all of the relevant information for a product listing. This class also handles making sure that the data input is correct through the use of Attributes. The Product Model is used by other Models and Controllers to display and manipulate the information for the product listings. The Product Model follows the Single Responsibility Principle because it is only concerned with the responsibilities of handing product data and behavior. This model is shown below in Figure 7.

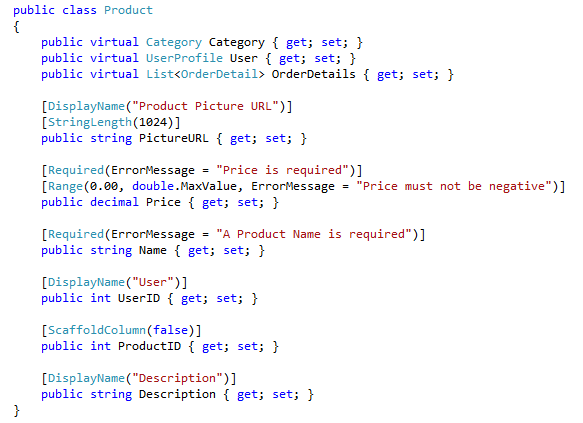


Figure : The Product Model

To allow users to find products to purchase from other users, a store page was added to the system. The first page of the store lists out the available categories that have products listed. Once a category is selected, a listing of products that belong to this category are listed through the Browse StoreController Method. Selecting a product that has been listed will open the Details StoreController Method which contains information about the product as well as the ability to add the product to the user's shopping cart.

When a user is ready to purchase a product, they can add the product to their shopping cart. The Shopping Cart Model is responsible for keeping track of the items that a user is planning on purchasing. It also keeps a running total of items and prices. The Shopping Cart Model follows the Liskov Substitution Principle because it delegates data storage to Cart objects instead of incorrectly inheriting from the Cart Model. To help users remember how many items are stored in their shopping cart, a partial view (seen in the figures below) was used to give the user a visual that contained the number of items in the cart. This partial view follows the DRY Principle.

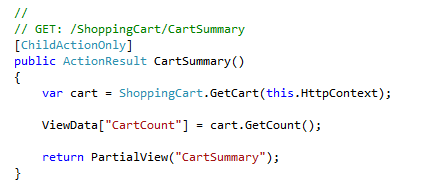


Figure : The Shopping Cart Summary Method



Figure : The Shopping Cart PartialView Code

The Shopping Cart also allows users to fill their cart without being registered or logged in by having the ability to migrate carts between users. The cart for an unregistered user is stored in the browser's session data so that it can be accessed when the user logs in or registers. The method that handles migrations is found in Figure 10 below.

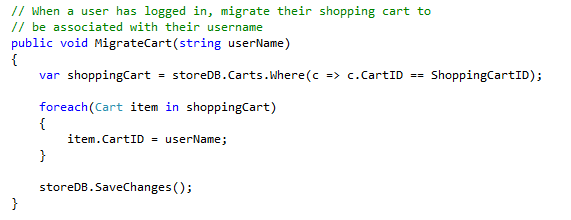


Figure : The Shopping Cart Migration Method

When a user is ready to check out, they can navigate to the shopping cart page to view the items that have been added to the cart. The user will also be presented with a price breakdown as well as a grand total. From the shopping cart page, they will be able to navigate to the checkout screen and enter their billing and shipping information. The Controller method for the shopping cart page is shown in Figure 11 below.

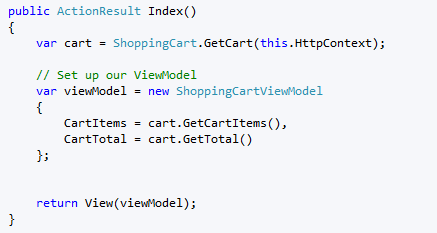


Figure : ShoppingCart Index Method

To enable our team to test the system and ensure that it was operating as intended, we created a dummy database using the Entity Framework's Code First development to create a database. The Code First database requires two classes: an Entity class (MakerStoreEntities.cs) and a class to store the data that will be used to initialize the database (SampleData.cs). The MakerStoreEntities class defines the tables that will be created in the database. The SampleData class defines the data that will populate those tables. The code for these classes can be seen in the following figures.

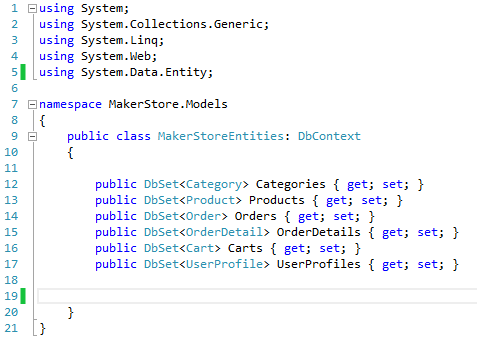


Figure : MakerStoreEntity.cs



Figure : SampleData.cs (Categories and Users)

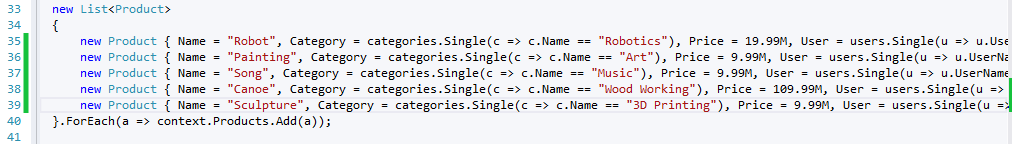


Figure : SampleData.cs (Products)

# Mock Scenarios of System Interaction

To ensure that the system operates as intended, out team created a set of common scenarios that the average user would go through. We then stepped through these scenarios and paid great attention to the results to compare the actual results to the expected results. If there were any discrepancies, we quickly remedied the situation and corrected the interaction. The common scenarios that we tested were:

* Account Registration
* Editing Account Information
* Creating a Product Listing
* Editing a Product Listing
* Purchasing a Product

To test these interactions, our team created step by step instructions on how to accomplish the tasks that were targeted by each scenario. In the following subsections are the scenarios, the steps to accomplish them, and the expected results of each scenario.

## Account Registration

In order to begin using the web service, a user must create an account. This scenario steps through the process of creating this account.

1. Navigate to the home page URL of the website through an internet browser. (Note: the domain hasn't been set yet, so when running these simulations from Visual Studio the URL of the home page is localhost:[port] where port is the port number assigned by Visual Studio at build time.)

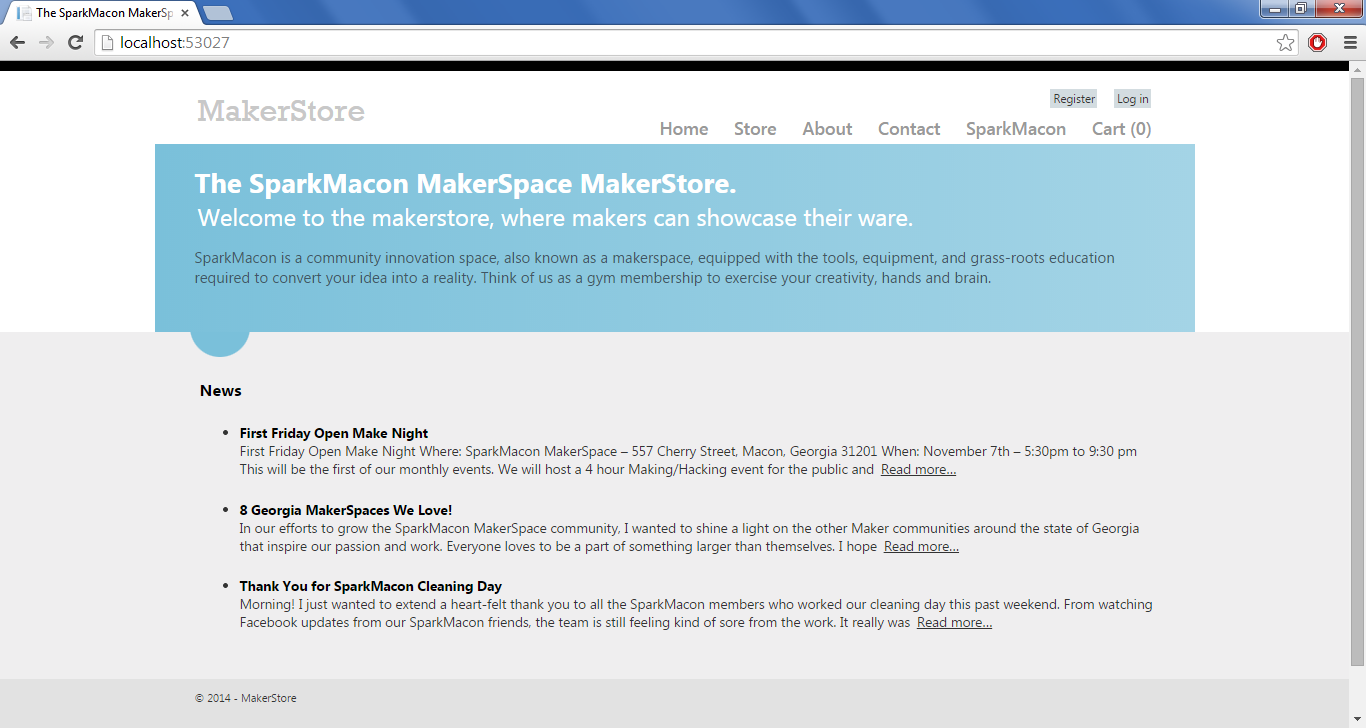


Figure : Home Page of the MakerStore

1. Click the Register button at the top right of the screen.



Figure : The Register Button

1. From the registration page, fill in the requested information.

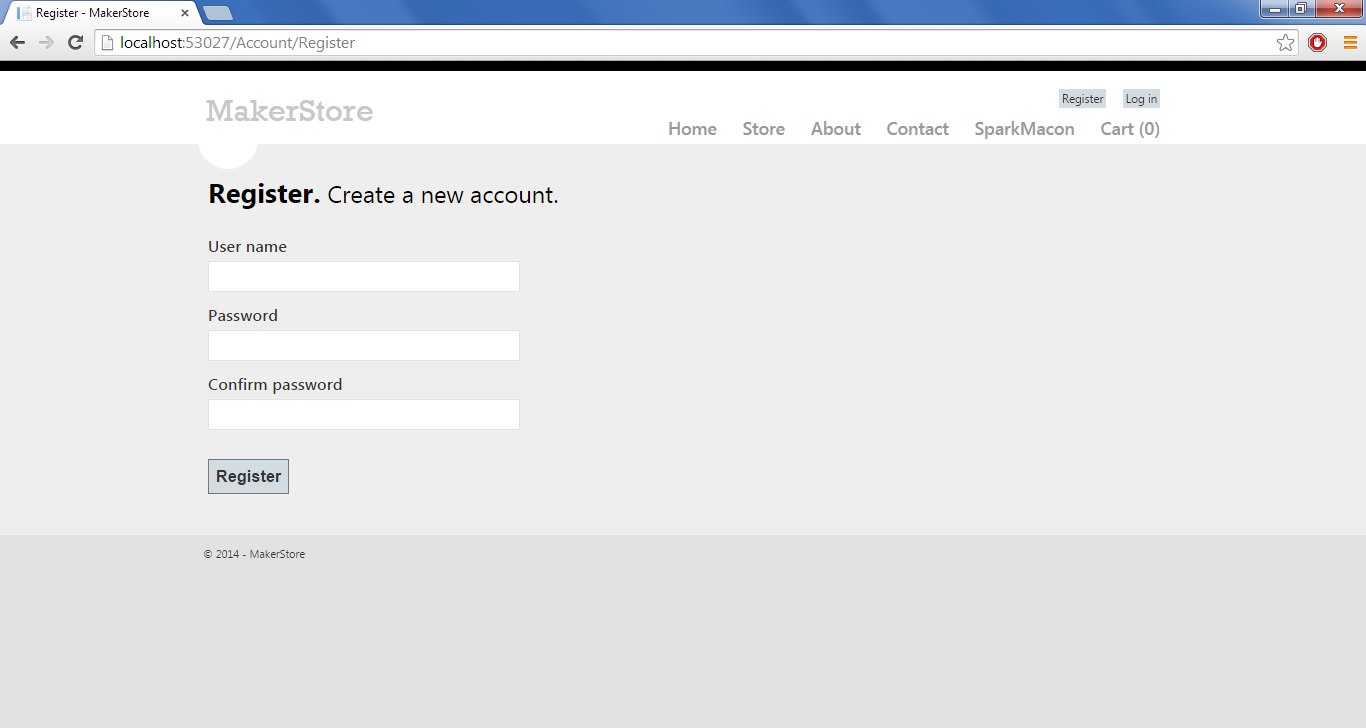


Figure : The Registration Page

1. Select Continue. You will be returned to the home page and the selected username will appear in the top right of the screen.



Figure : The Username After Creating an Account

## Editing Account Information

After an account has been created, a user will have the ability to edit their account's information. This will include details such as their biography and list of available products. This scenario can only be completed after the previous scenario has been completed.

1. Navigate to the home page of the website.
2. Select Log In from the top right of the screen.



Figure : The Log In Button

1. Enter the username and password associated with the account created in the first scenario.

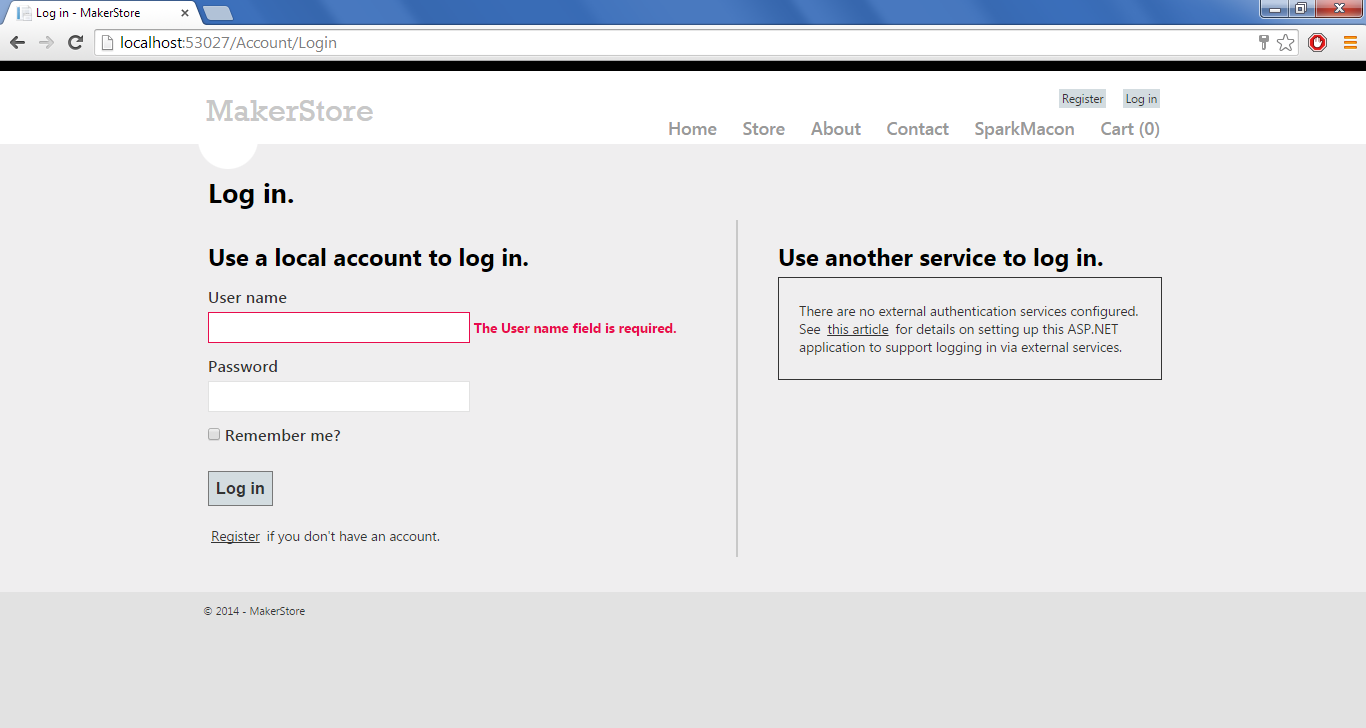


Figure : The Log In Screen

1. Press Continue. You will be returned to the home page. Select your username from the top right of the screen.
2. Click on the username. From the user account page, select Update Account Information.

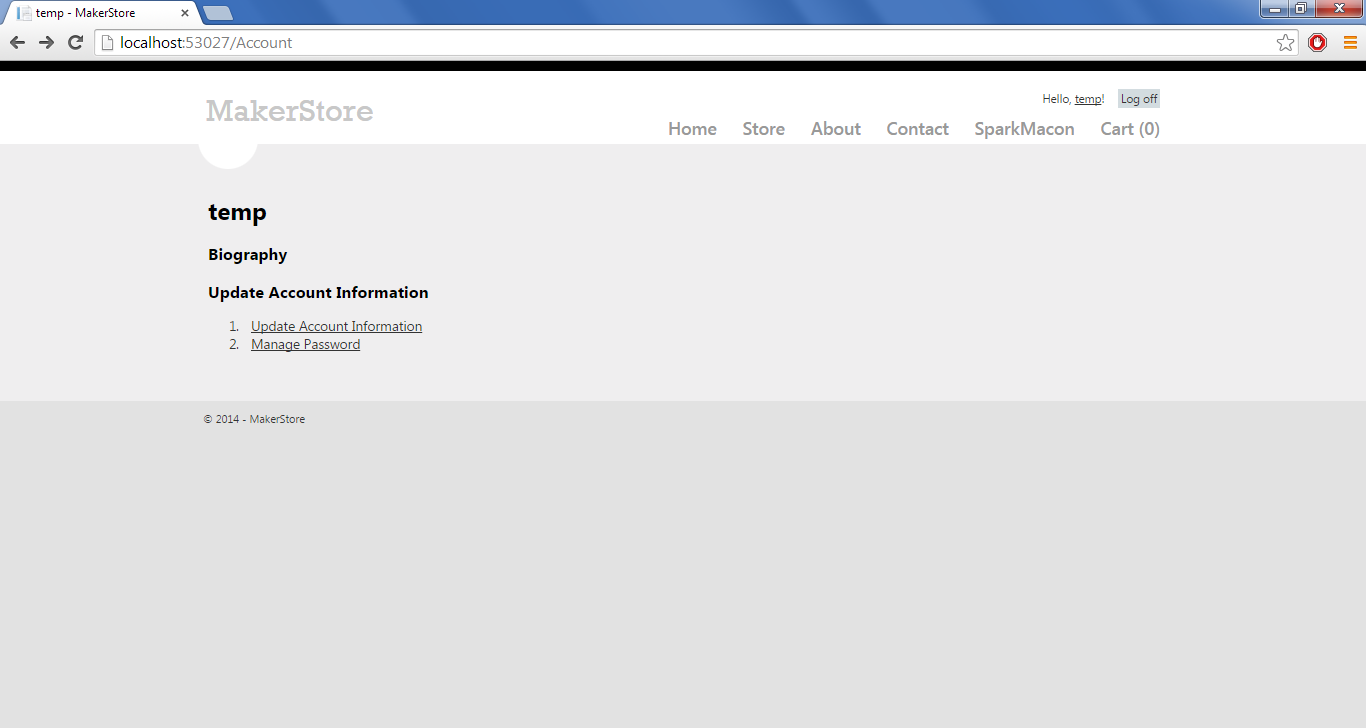


Figure : The User Account Page

1. Input the updated information. Click Continue.

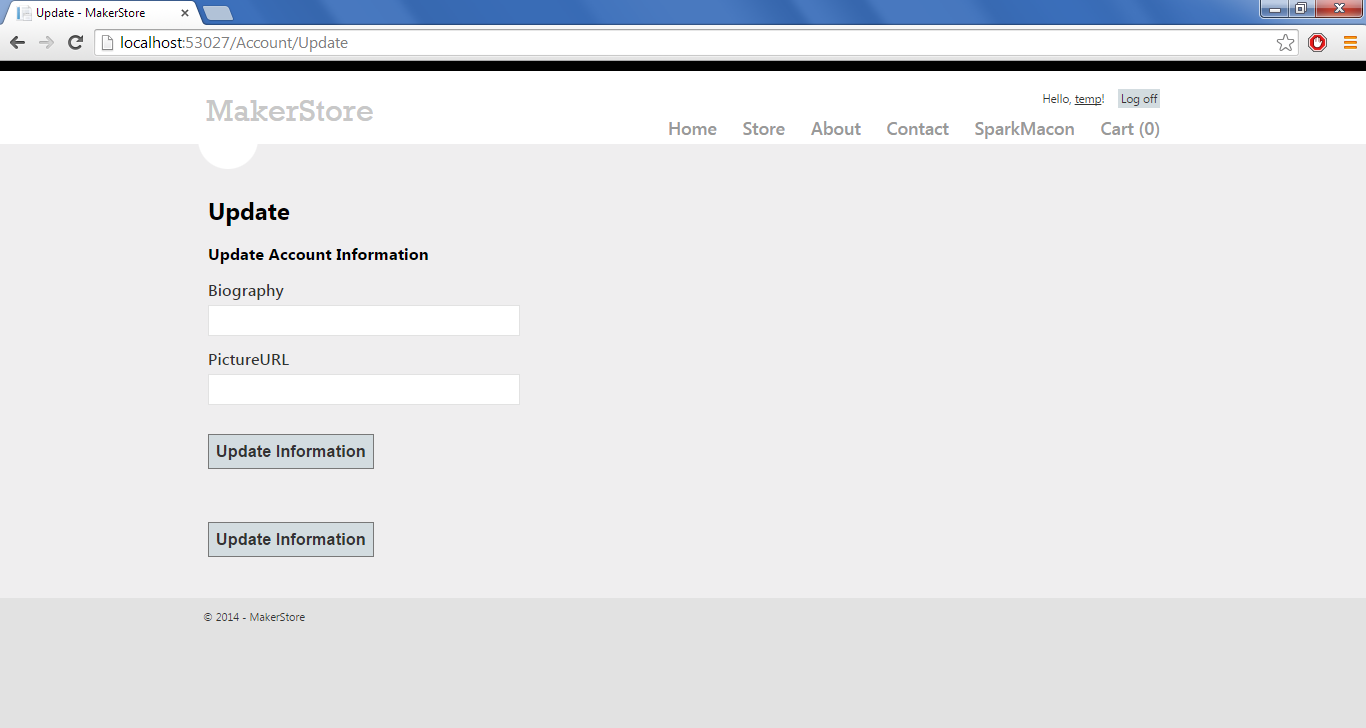


Figure : The Account Information Edit Form

1. The website will redirect you to an updated user account page.

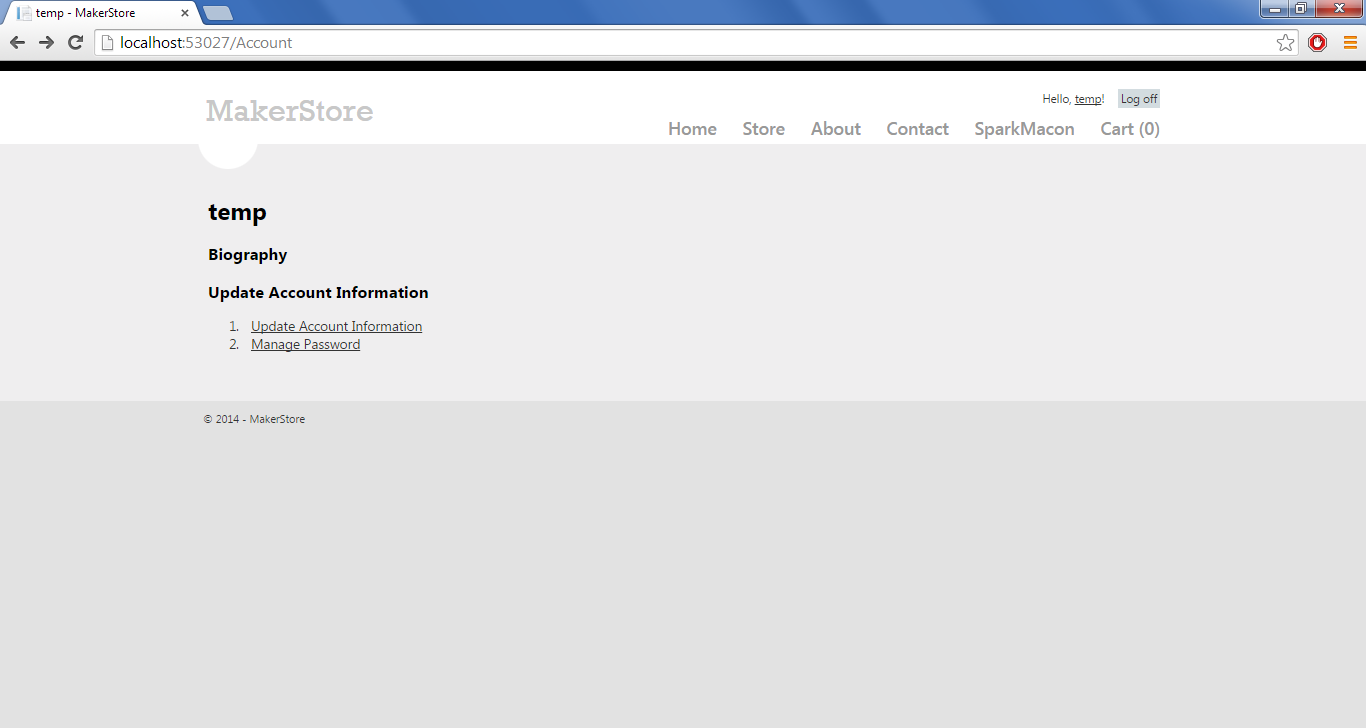


Figure : Updated User Account Page

## Purchasing a Product With an Account

After an account has been created, a user will have the ability to edit purchase a product. This scenario can only be completed after the first scenario has been completed.

1. Navigate to the home page of the web site.
2. Select Log In from the top right of the screen.
3. Enter the username and password associated with the account created in the first scenario.
4. You will be redirected to the home page. Click the Store button in the top right.



Figure : The Store Button

1. You will be redirected to the Store Page.

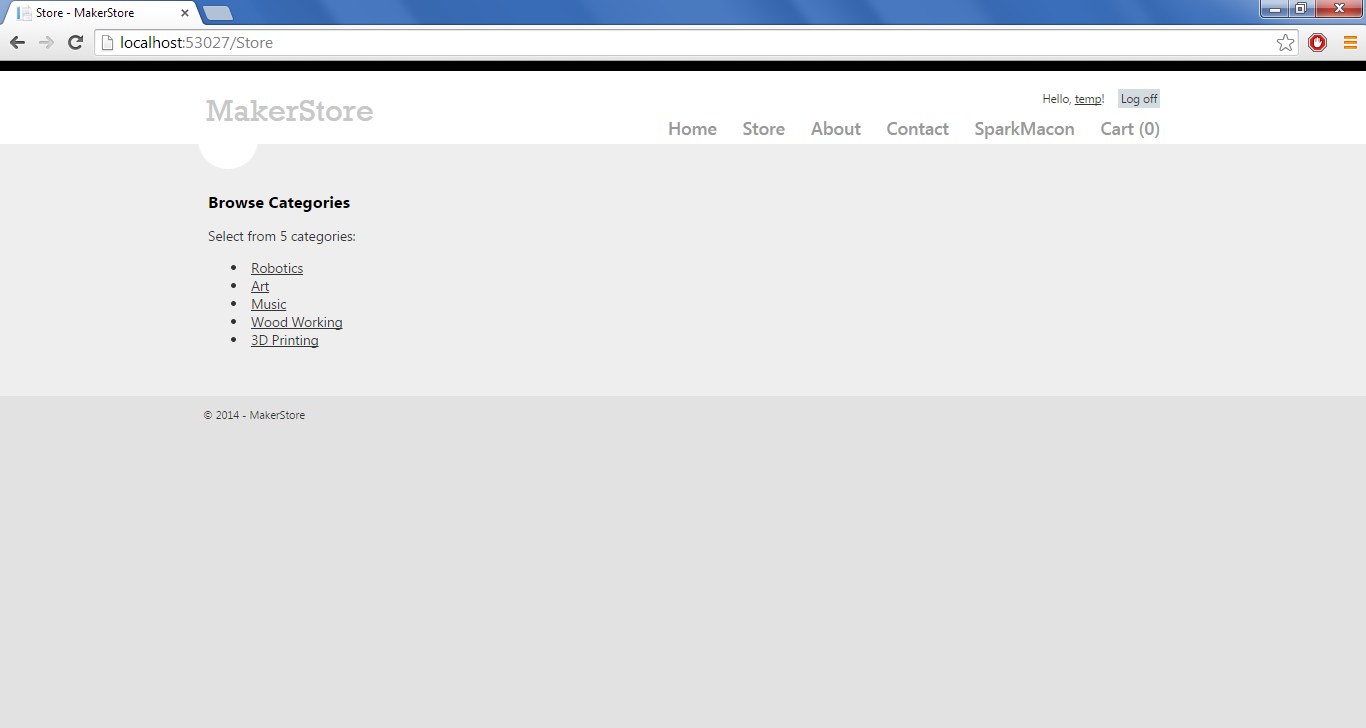


Figure : The Store Page

1. From the Store Page, select a category.
2. You will be redirected to a list of products from that category.

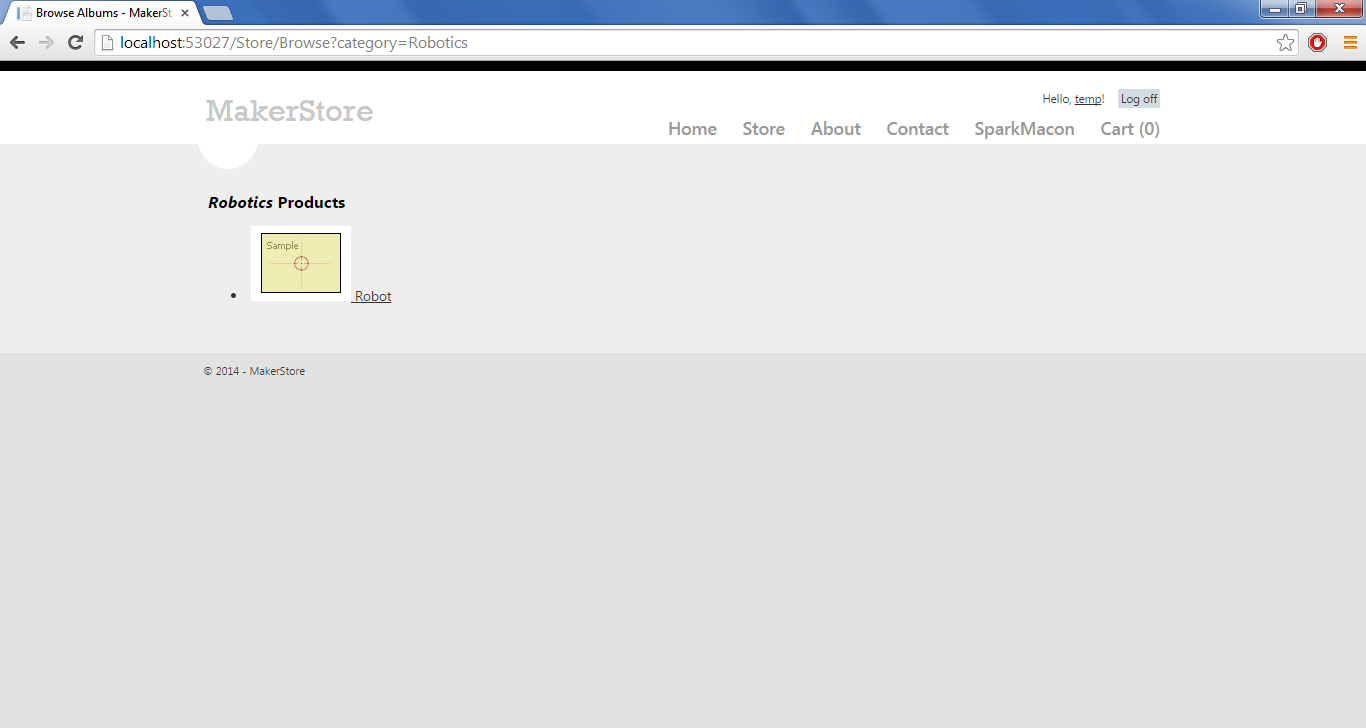


Figure : The Product Listings Page

1. Select a product.
2. You will be redirected to the listing page of the product.

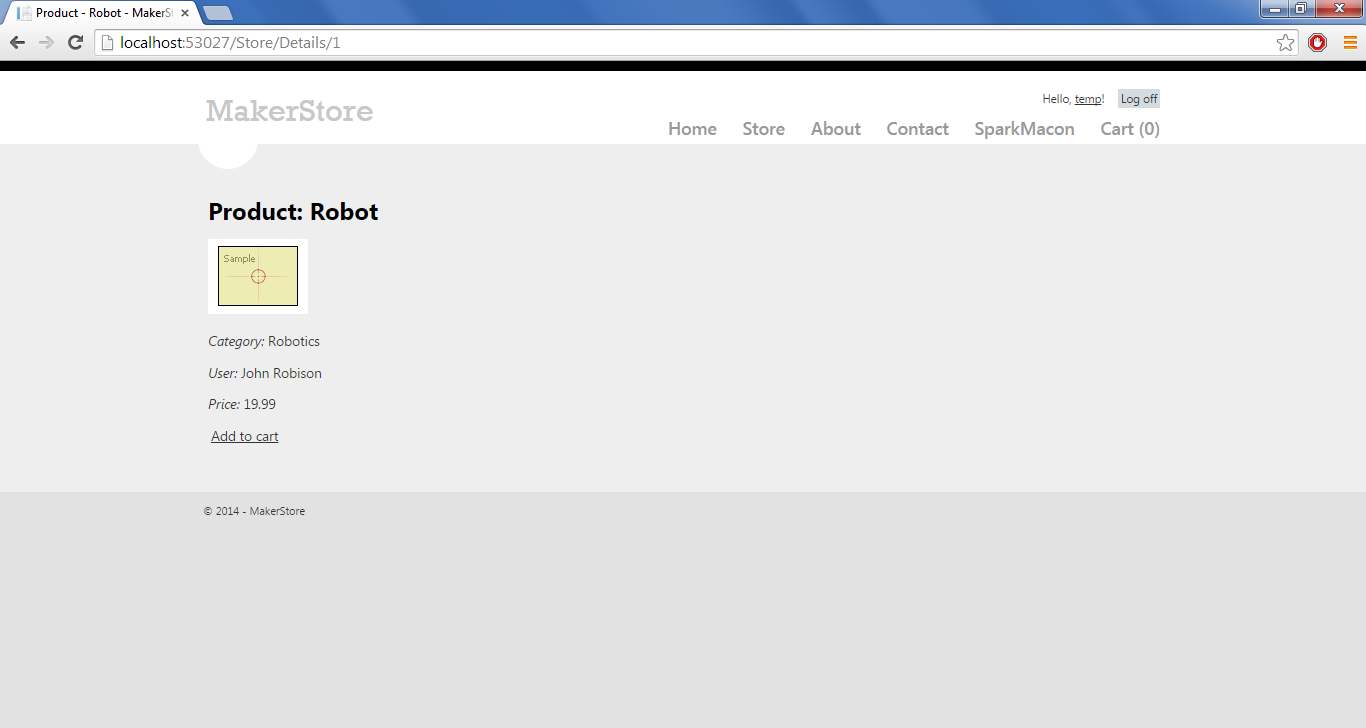


Figure : The Product Listing Page

1. From the Product Listing Page, click Add To Cart.

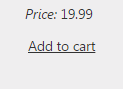


Figure : The Add To Cart Button

1. You will be redirected to the Shopping Cart Page. The Cart icon in the top right of the screen will be updated to display the number of items in the cart.

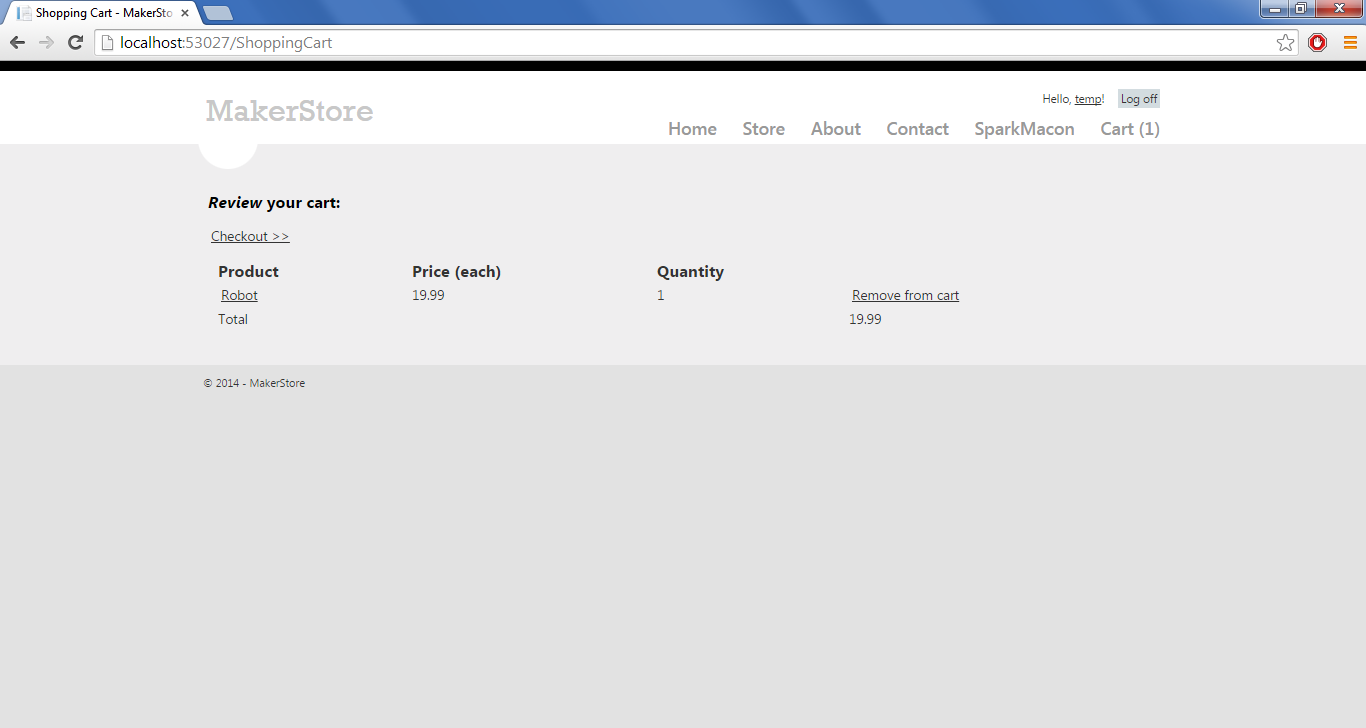


Figure : The Shopping Cart Page With Updated Cart Icon

1. Click Checkout.

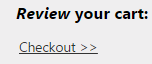


Figure : The Checkout Button

1. Verify that the billing and shipping information is correct. Click Submit Order.

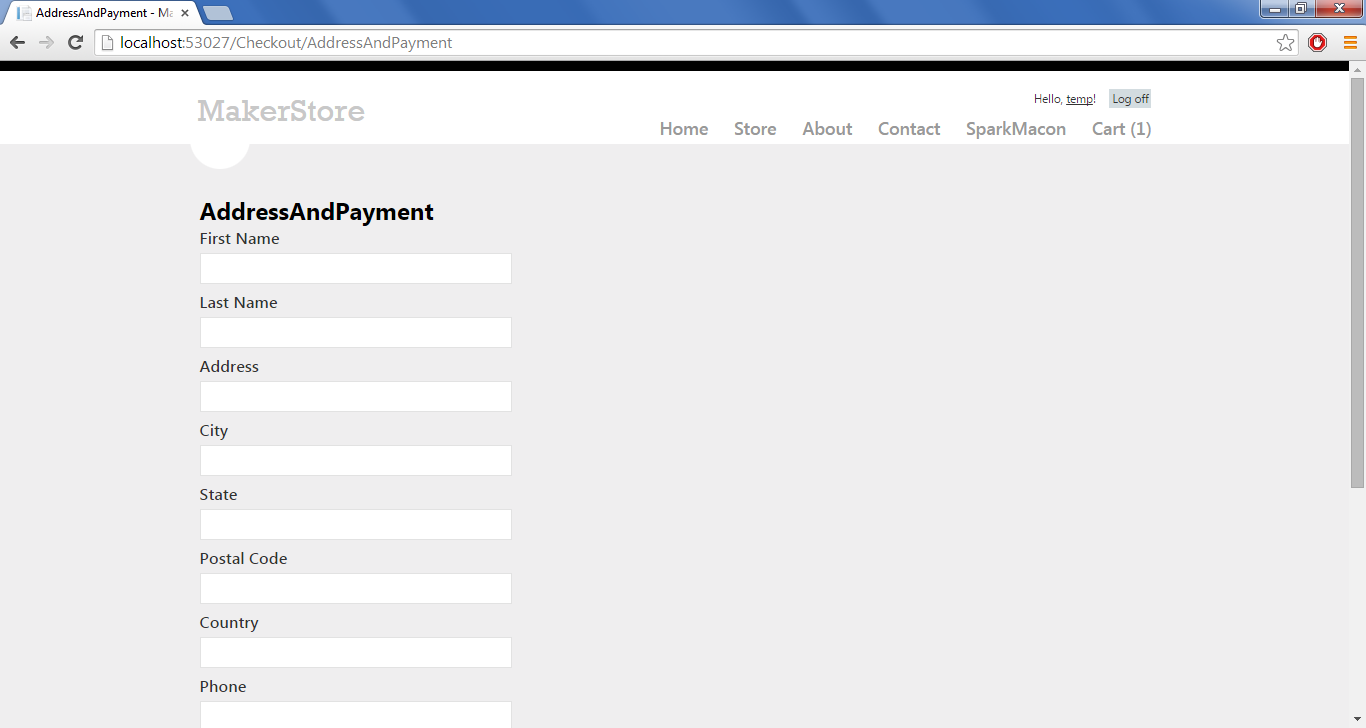


Figure : The Checkout Screen

1. You will be redirected to a Checkout Confirmation Screen. The Cart icon will be reset to the original configuration.

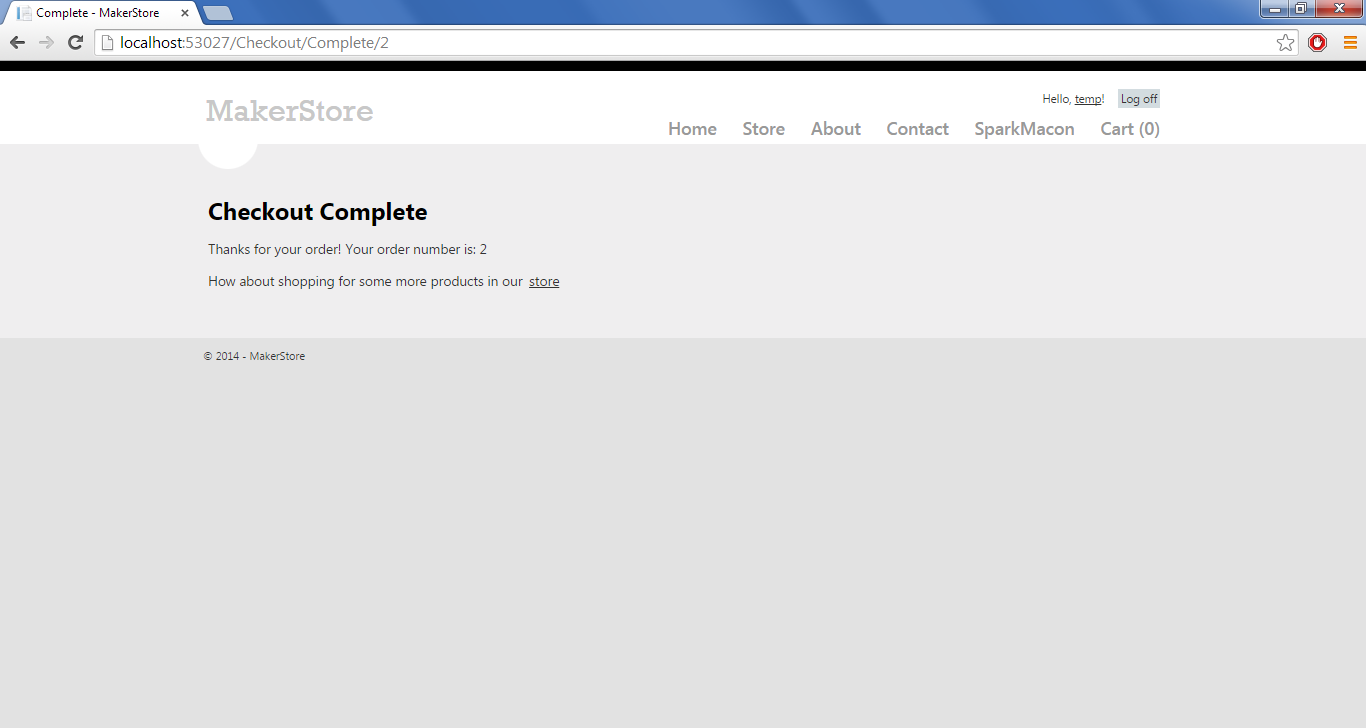


Figure : The Checkout Confirmation Screen

## Purchasing a Product Without an Account

Products can be purchased by users who do not currently have an account. Products can be added to a temporary Shopping Cart until the user is ready to check out and then they will be required to register an account. This scenario does not require any previous scenarios to have been completed.

1. Navigate to the home page of the web site.
2. Click the Store button in the top right.
3. You will be redirected to the Store Page. From the Store Page, select a category.
4. From the category page, select a product.
5. You will be redirected to the listing page of the product.
6. From the Product Listing Page, click Add To Cart.
7. You will be redirected to the Shopping Cart Page. The Cart icon in the top right of the screen will be updated to display the number of items in the cart.
8. Click Checkout.
9. You will be redirected to the Registration Page. The Registration Page will still contain the updated Cart icon.

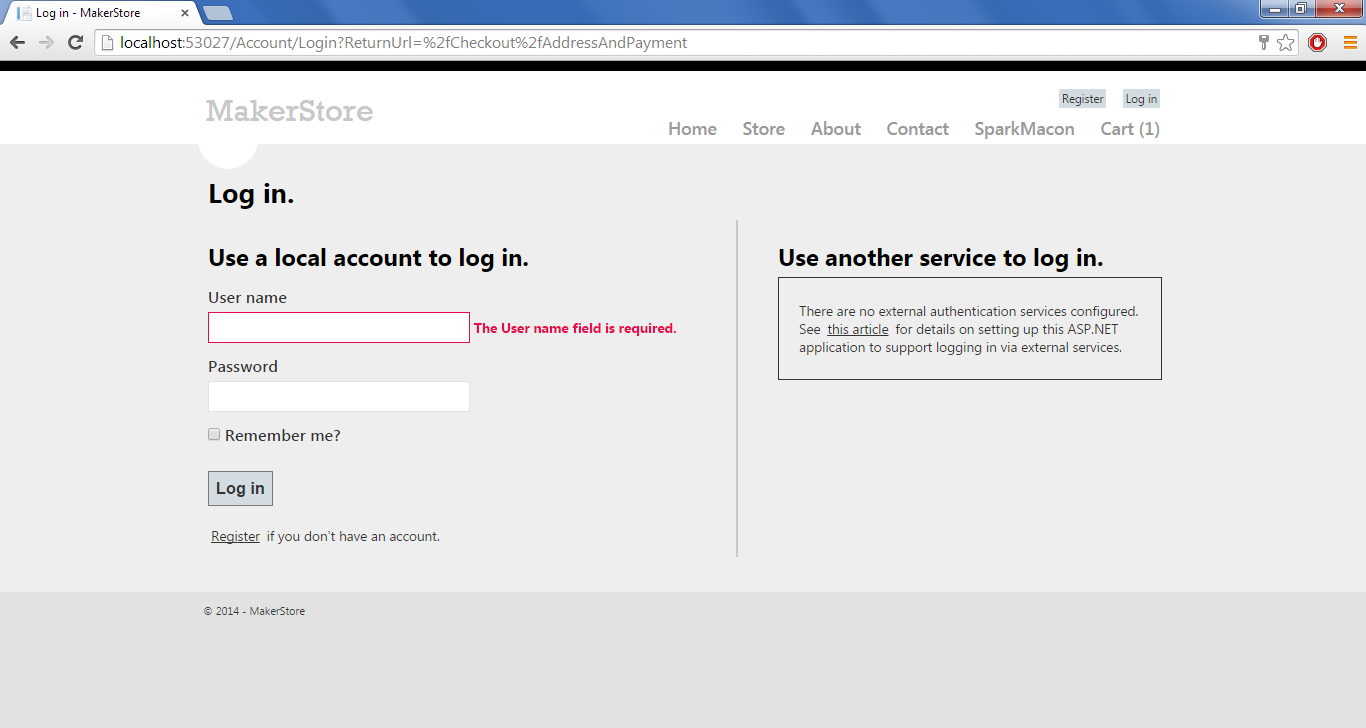


Figure : The Registration Page With the Updated Cart Icon

1. Fill in the required information and click Continue. You will be redirected to the Checkout Page.
2. Verify that the billing and shipping information are correct. Click Continue.
3. You will be redirected to a Checkout Confirmation Screen. The Cart icon will be reset to the original configuration.

# 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>.

# Source Code

## AccountController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Transactions;

using System.Web;

using System.Web.Mvc;

using System.Web.Security;

using DotNetOpenAuth.AspNet;

using Microsoft.Web.WebPages.OAuth;

using WebMatrix.WebData;

using MakerStore.Filters;

using MakerStore.Models;

namespace MakerStore.Controllers

{

[Authorize]

[InitializeSimpleMembership]

public class AccountController : Controller

{

//

// GET: /Account/

public ActionResult Index()

{

return View();

}

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Login(LoginModel model, string returnUrl)

{

if (ModelState.IsValid && WebSecurity.Login(model.UserName, model.Password, persistCookie: model.RememberMe))

{

return RedirectToLocal(returnUrl);

}

// If we got this far, something failed, redisplay form

ModelState.AddModelError("", "The user name or password provided is incorrect.");

return View(model);

}

//

// POST: /Account/LogOff

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

WebSecurity.Logout();

return RedirectToAction("Index", "Home");

}

//

// GET: /Account/Register

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Register(RegisterModel model)

{

if (ModelState.IsValid)

{

// Attempt to register the user

try

{

WebSecurity.CreateUserAndAccount(model.UserName, model.Password);

WebSecurity.Login(model.UserName, model.Password);

return RedirectToAction("Index", "Home");

}

catch (MembershipCreateUserException e)

{

ModelState.AddModelError("", ErrorCodeToString(e.StatusCode));

}

}

// If we got this far, something failed, redisplay form

return View(model);

}

//

// POST: /Account/Disassociate

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Disassociate(string provider, string providerUserId)

{

string ownerAccount = OAuthWebSecurity.GetUserName(provider, providerUserId);

ManageMessageId? message = null;

// Only disassociate the account if the currently logged in user is the owner

if (ownerAccount == User.Identity.Name)

{

// Use a transaction to prevent the user from deleting their last login credential

using (var scope = new TransactionScope(TransactionScopeOption.Required, new TransactionOptions { IsolationLevel = IsolationLevel.Serializable }))

{

bool hasLocalAccount = OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

if (hasLocalAccount || OAuthWebSecurity.GetAccountsFromUserName(User.Identity.Name).Count > 1)

{

OAuthWebSecurity.DeleteAccount(provider, providerUserId);

scope.Complete();

message = ManageMessageId.RemoveLoginSuccess;

}

}

}

return RedirectToAction("Manage", new { Message = message });

}

//

// GET: /Account/Update

public ActionResult Update()

{

return View();

}

//

// GET: /Account/Manage

public ActionResult Manage(ManageMessageId? message)

{

ViewBag.StatusMessage =

message == ManageMessageId.ChangePasswordSuccess ? "Your password has been changed."

: message == ManageMessageId.SetPasswordSuccess ? "Your password has been set."

: message == ManageMessageId.RemoveLoginSuccess ? "The external login was removed."

: "";

ViewBag.HasLocalPassword = OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

ViewBag.ReturnUrl = Url.Action("Manage");

return View();

}

//

// POST: /Account/Manage

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Manage(LocalPasswordModel model)

{

bool hasLocalAccount = OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

ViewBag.HasLocalPassword = hasLocalAccount;

ViewBag.ReturnUrl = Url.Action("Manage");

if (hasLocalAccount)

{

if (ModelState.IsValid)

{

// ChangePassword will throw an exception rather than return false in certain failure scenarios.

bool changePasswordSucceeded;

try

{

changePasswordSucceeded = WebSecurity.ChangePassword(User.Identity.Name, model.OldPassword, model.NewPassword);

}

catch (Exception)

{

changePasswordSucceeded = false;

}

if (changePasswordSucceeded)

{

return RedirectToAction("Manage", new { Message = ManageMessageId.ChangePasswordSuccess });

}

else

{

ModelState.AddModelError("", "The current password is incorrect or the new password is invalid.");

}

}

}

else

{

// User does not have a local password so remove any validation errors caused by a missing

// OldPassword field

ModelState state = ModelState["OldPassword"];

if (state != null)

{

state.Errors.Clear();

}

if (ModelState.IsValid)

{

try

{

WebSecurity.CreateAccount(User.Identity.Name, model.NewPassword);

return RedirectToAction("Manage", new { Message = ManageMessageId.SetPasswordSuccess });

}

catch (Exception)

{

ModelState.AddModelError("", String.Format("Unable to create local account. An account with the name \"{0}\" may already exist.", User.Identity.Name));

}

}

}

// If we got this far, something failed, redisplay form

return View(model);

}

//

// POST: /Account/ExternalLogin

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider, string returnUrl)

{

return new ExternalLoginResult(provider, Url.Action("ExternalLoginCallback", new { ReturnUrl = returnUrl }));

}

//

// GET: /Account/ExternalLoginCallback

[AllowAnonymous]

public ActionResult ExternalLoginCallback(string returnUrl)

{

AuthenticationResult result = OAuthWebSecurity.VerifyAuthentication(Url.Action("ExternalLoginCallback", new { ReturnUrl = returnUrl }));

if (!result.IsSuccessful)

{

return RedirectToAction("ExternalLoginFailure");

}

if (OAuthWebSecurity.Login(result.Provider, result.ProviderUserId, createPersistentCookie: false))

{

return RedirectToLocal(returnUrl);

}

if (User.Identity.IsAuthenticated)

{

// If the current user is logged in add the new account

OAuthWebSecurity.CreateOrUpdateAccount(result.Provider, result.ProviderUserId, User.Identity.Name);

return RedirectToLocal(returnUrl);

}

else

{

// User is new, ask for their desired membership name

string loginData = OAuthWebSecurity.SerializeProviderUserId(result.Provider, result.ProviderUserId);

ViewBag.ProviderDisplayName = OAuthWebSecurity.GetOAuthClientData(result.Provider).DisplayName;

ViewBag.ReturnUrl = returnUrl;

return View("ExternalLoginConfirmation", new RegisterExternalLoginModel { UserName = result.UserName, ExternalLoginData = loginData });

}

}

//

// POST: /Account/ExternalLoginConfirmation

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLoginConfirmation(RegisterExternalLoginModel model, string returnUrl)

{

string provider = null;

string providerUserId = null;

if (User.Identity.IsAuthenticated || !OAuthWebSecurity.TryDeserializeProviderUserId(model.ExternalLoginData, out provider, out providerUserId))

{

return RedirectToAction("Manage");

}

if (ModelState.IsValid)

{

// Insert a new user into the database

using (MakerStoreEntities db = new MakerStoreEntities())

{

UserProfile user = db.UserProfiles.FirstOrDefault(u => u.UserName.ToLower() == model.UserName.ToLower());

// Check if user already exists

if (user == null)

{

// Insert name into the profile table

db.UserProfiles.Add(new UserProfile { UserName = model.UserName });

db.SaveChanges();

OAuthWebSecurity.CreateOrUpdateAccount(provider, providerUserId, model.UserName);

OAuthWebSecurity.Login(provider, providerUserId, createPersistentCookie: false);

return RedirectToLocal(returnUrl);

}

else

{

ModelState.AddModelError("UserName", "User name already exists. Please enter a different user name.");

}

}

}

ViewBag.ProviderDisplayName = OAuthWebSecurity.GetOAuthClientData(provider).DisplayName;

ViewBag.ReturnUrl = returnUrl;

return View(model);

}

//

// GET: /Account/ExternalLoginFailure

[AllowAnonymous]

public ActionResult ExternalLoginFailure()

{

return View();

}

[AllowAnonymous]

[ChildActionOnly]

public ActionResult ExternalLoginsList(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return PartialView("\_ExternalLoginsListPartial", OAuthWebSecurity.RegisteredClientData);

}

[ChildActionOnly]

public ActionResult RemoveExternalLogins()

{

ICollection<OAuthAccount> accounts = OAuthWebSecurity.GetAccountsFromUserName(User.Identity.Name);

List<ExternalLogin> externalLogins = new List<ExternalLogin>();

foreach (OAuthAccount account in accounts)

{

AuthenticationClientData clientData = OAuthWebSecurity.GetOAuthClientData(account.Provider);

externalLogins.Add(new ExternalLogin

{

Provider = account.Provider,

ProviderDisplayName = clientData.DisplayName,

ProviderUserId = account.ProviderUserId,

});

}

ViewBag.ShowRemoveButton = externalLogins.Count > 1 || OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

return PartialView("\_RemoveExternalLoginsPartial", externalLogins);

}

#region Helpers

private ActionResult RedirectToLocal(string returnUrl)

{

if (Url.IsLocalUrl(returnUrl))

{

return Redirect(returnUrl);

}

else

{

return RedirectToAction("Index", "Home");

}

}

public enum ManageMessageId

{

ChangePasswordSuccess,

SetPasswordSuccess,

RemoveLoginSuccess,

}

internal class ExternalLoginResult : ActionResult

{

public ExternalLoginResult(string provider, string returnUrl)

{

Provider = provider;

ReturnUrl = returnUrl;

}

public string Provider { get; private set; }

public string ReturnUrl { get; private set; }

public override void ExecuteResult(ControllerContext context)

{

OAuthWebSecurity.RequestAuthentication(Provider, ReturnUrl);

}

}

private static string ErrorCodeToString(MembershipCreateStatus createStatus)

{

// See http://go.microsoft.com/fwlink/?LinkID=177550 for

// a full list of status codes.

switch (createStatus)

{

case MembershipCreateStatus.DuplicateUserName:

return "User name already exists. Please enter a different user name.";

case MembershipCreateStatus.DuplicateEmail:

return "A user name for that e-mail address already exists. Please enter a different e-mail address.";

case MembershipCreateStatus.InvalidPassword:

return "The password provided is invalid. Please enter a valid password value.";

case MembershipCreateStatus.InvalidEmail:

return "The e-mail address provided is invalid. Please check the value and try again.";

case MembershipCreateStatus.InvalidAnswer:

return "The password retrieval answer provided is invalid. Please check the value and try again.";

case MembershipCreateStatus.InvalidQuestion:

return "The password retrieval question provided is invalid. Please check the value and try again.";

case MembershipCreateStatus.InvalidUserName:

return "The user name provided is invalid. Please check the value and try again.";

case MembershipCreateStatus.ProviderError:

return "The authentication provider returned an error. Please verify your entry and try again. If the problem persists, please contact your system administrator.";

case MembershipCreateStatus.UserRejected:

return "The user creation request has been canceled. Please verify your entry and try again. If the problem persists, please contact your system administrator.";

default:

return "An unknown error occurred. Please verify your entry and try again. If the problem persists, please contact your system administrator.";

}

}

#endregion

}

}

## CheckoutController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using MakerStore.Models;

namespace MakerStore.Controllers

{

[Authorize]

public class CheckoutController : Controller

{

MakerStoreEntities storeDB = new MakerStoreEntities();

const string PromoCode = "FREE";

//

// GET: /Checkout/AddressAndPayment

public ActionResult AddressAndPayment()

{

return View();

}

//

// POST: /Checkout/AddressAndPayment

[HttpPost]

public ActionResult AddressAndPayment(FormCollection values)

{

var order = new Order();

TryUpdateModel(order);

try

{

if (string.Equals(values["PromoCode"], PromoCode, StringComparison.OrdinalIgnoreCase) == false)

{

return View(order);

}

else

{

order.UserName = User.Identity.Name;

order.OrderDate = DateTime.Now;

// Save Order

storeDB.Orders.Add(order);

storeDB.SaveChanges();

// Process the order

var cart = ShoppingCart.GetCart(this.HttpContext);

cart.CreateOrder(order);

return RedirectToAction("Complete", new { id = order.OrderID });

}

}

catch

{

// Invalid - Redisplay with errors

return View(order);

}

}

//

// GET: /Checkout/Complete

public ActionResult Complete(int id)

{

// Validate that customer owns this order

bool isValid = storeDB.Orders.Any(o => o.OrderID == id

&& o.UserName == User.Identity.Name);

if (isValid)

{

return View(id);

}

else

{

return View("Error");

}

}

}

}

## HomeController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MakerStore.Controllers

{

public class HomeController : Controller

{

public ActionResult Index()

{

ViewBag.Message = "Where Makers Prosper";

return View();

}

public ActionResult About()

{

ViewBag.Message = "Your app description page.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

}

}

## ShoppingCartController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using MakerStore.Models;

using MakerStore.ViewModels;

namespace MakerStore.Controllers

{

public class ShoppingCartController : Controller

{

MakerStoreEntities storeDB = new MakerStoreEntities();

//

// GET: /ShoppingCart/

public ActionResult Index()

{

var cart = ShoppingCart.GetCart(this.HttpContext);

// Set up our ViewModel

var viewModel = new ShoppingCartViewModel

{

CartItems = cart.GetCartItems(),

CartTotal = cart.GetTotal()

};

return View(viewModel);

}

//

// GET: /ShoppingCart/AddtoCart/5

public ActionResult AddToCart(int id)

{

// Retrieve the album from the database

var addedAlbum = storeDB.Products.Single(product => product.ProductID == id);

// Ad it to the shopping cart

var cart = ShoppingCart.GetCart(this.HttpContext);

cart.AddToCart(addedAlbum);

// Go back to the main store page for more shopping

return RedirectToAction("Index");

}

//

//AJAX: /ShoppingCart/RemoveFromCart/5

[HttpPost]

public ActionResult RemoveFromCart(int id)

{

// Remove the item from the cart

var cart = ShoppingCart.GetCart(this.HttpContext);

// Get the name of the album to display confirmation

string albumName = storeDB.Carts.Single(item => item.ProductID == id).Product.Name;

// Remove from cart

int itemCount = cart.RemoveFromCart(id);

// Display the confirmation message

var results = new ShoppingCartRemoveViewModel

{

Message = Server.HtmlEncode(albumName) + " has been removed from your shopping cart.",

CartTotal = cart.GetTotal(),

CartCount = cart.GetCount(),

DeleteId = id

};

return Json(results);

}

//

// GET: /ShoppingCart/CartSummary

[ChildActionOnly]

public ActionResult CartSummary()

{

var cart = ShoppingCart.GetCart(this.HttpContext);

ViewData["CartCount"] = cart.GetCount();

return PartialView("CartSummary");

}

}

}

## StoreController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using MakerStore.Models;

namespace MakerStore.Controllers

{

public class StoreController : Controller

{

MakerStoreEntities storeDB = new MakerStoreEntities();

//

// GET: /Store/

public ActionResult Index()

{

var categories = storeDB.Categories.ToList();

return View(categories);

}

//

// GET: /Store/Browse?category=robotics

public ActionResult Browse(string category)

{

// Retrieve Category and its Associated Products from database

var categoryModel = storeDB.Categories.Include("Products").Single(c => c.Name == category);

return View(categoryModel);

}

//

// GET: /Store/Details/5

public ActionResult Details(int id)

{

var product = storeDB.Products.Find(id);

return View(product);

}

//

// GET: /Store/CategoryMenu

[ChildActionOnly]

public ActionResult CategoryMenu()

{

var categories = storeDB.Categories.ToList();

return PartialView(categories);

}

}

}

## AccountModel.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity;

using System.Globalization;

using System.Web.Mvc;

using System.Web.Security;

namespace MakerStore.Models

{

[Table("UserProfile")]

public class UserProfile

{

[Key]

[DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]

public int UserId { get; set; }

public string UserName { get; set; }

public List<Product> Products { get; set; }

public string Biography { get; set; }

public string PictureURL { get; set; }

}

public class RegisterExternalLoginModel

{

[Required]

[Display(Name = "User name")]

public string UserName { get; set; }

public string ExternalLoginData { get; set; }

}

public class LocalPasswordModel

{

[Required]

[DataType(DataType.Password)]

[Display(Name = "Current password")]

public string OldPassword { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "New password")]

public string NewPassword { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm new password")]

[Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

public class LoginModel

{

[Required]

[Display(Name = "User name")]

public string UserName { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[Display(Name = "Remember me?")]

public bool RememberMe { get; set; }

}

public class RegisterModel

{

[Required]

[Display(Name = "User name")]

public string UserName { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

public class ExternalLogin

{

public string Provider { get; set; }

public string ProviderDisplayName { get; set; }

public string ProviderUserId { get; set; }

}

}

## Cart.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.ComponentModel.DataAnnotations;

namespace MakerStore.Models

{

public class Cart

{

[Key]

public int RecordID { get; set; }

public string CartID { get; set; }

public int ProductID { get; set; }

public int Count { get; set; }

public System.DateTime DateCreated { get; set; }

public virtual Product Product { get; set; }

}

}

## Category.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace MakerStore.Models

{

public class Category

{

public int CategoryID { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public List<Product> Products { get; set; }

}

}

## MakerStoreEntities.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

namespace MakerStore.Models

{

public class MakerStoreEntities: DbContext

{

public DbSet<Category> Categories { get; set; }

public DbSet<Product> Products { get; set; }

public DbSet<Order> Orders { get; set; }

public DbSet<OrderDetail> OrderDetails { get; set; }

public DbSet<Cart> Carts { get; set; }

public DbSet<UserProfile> UserProfiles { get; set; }

}

}

## Order.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

namespace MakerStore.Models

{

[Bind(Exclude = "OrderID")]

public class Order

{

[ScaffoldColumn(false)]

public int OrderID { get; set; }

[ScaffoldColumn(false)]

public string UserName { get; set; }

[Required(ErrorMessage = "First Name is required")]

[DisplayName("First Name")]

[StringLength(160)]

public string FirstName { get; set; }

[Required(ErrorMessage = "Last Name is Required")]

[DisplayName("Last Name")]

[StringLength(160)]

public string LastName { get; set; }

[Required(ErrorMessage = "Address is required")]

[StringLength(70)]

public string Address { get; set; }

[Required(ErrorMessage = "City is required")]

[StringLength(40)]

public string City { get; set; }

[Required(ErrorMessage = "State is required")]

[StringLength(40)]

public string State { get; set; }

[Required(ErrorMessage = "Postal Code is required")]

[DisplayName("Postal Code")]

[StringLength(10)]

public string PostalCode { get; set; }

[Required(ErrorMessage = "Country is required")]

[StringLength(40)]

public string Country { get; set; }

[Required(ErrorMessage = "Phone is required")]

[StringLength(24)]

public string Phone { get; set; }

[Required(ErrorMessage = "Email Address is required")]

[DisplayName("Email Address")]

[RegularExpression(@"[A-Za-z0-9.%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}", ErrorMessage = "Email is not valid")]

[DataType(DataType.EmailAddress)]

public string Email { get; set; }

[ScaffoldColumn(false)]

public decimal Total { get; set; }

[ScaffoldColumn(false)]

public System.DateTime OrderDate { get; set; }

public List<OrderDetail> OrderDetails { get; set; }

}

}

## OrderDetail.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace MakerStore.Models

{

public class OrderDetail

{

public int OrderDetailID { get; set; }

public int OrderID { get; set; }

public int ProductID { get; set; }

public int Quantity { get; set; }

public decimal UnitPrice { get; set; }

public virtual Product Product { get; set; }

public virtual Order Order { get; set; }

}

}

## Product.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

namespace MakerStore.Models

{

public class Product

{

public virtual Category Category { get; set; }

public virtual UserProfile User { get; set; }

public virtual List<OrderDetail> OrderDetails { get; set; }

[DisplayName("Product Picture URL")]

[StringLength(1024)]

public string PictureURL { get; set; }

[Required(ErrorMessage = "Price is required")]

[Range(0.00, double.MaxValue, ErrorMessage = "Price must not be negative")]

public decimal Price { get; set; }

[Required(ErrorMessage = "A Product Name is required")]

public string Name { get; set; }

[DisplayName("User")]

public int UserID { get; set; }

[ScaffoldColumn(false)]

public int ProductID { get; set; }

[DisplayName("Description")]

public string Description { get; set; }

}

}

## SampleData.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

namespace MakerStore.Models

{

public class SampleData : DropCreateDatabaseIfModelChanges<MakerStoreEntities>

{

protected override void Seed(MakerStoreEntities context)

{

var categories = new List<Category>

{

new Category { Name = "Robotics"},

new Category { Name = "Art"},

new Category { Name = "Wood Working"},

new Category { Name = "3D Printing"},

new Category { Name = "Music"}

};

var users = new List<UserProfile>

{

new UserProfile { UserName = "Admin"},

new UserProfile { UserName = "Test"},

new UserProfile { UserName = "Micheal Rose", Biography = "I am very commited to pushing the Macon maker's movement. My passion is to create excellent software. In my spare time I create classical music.", PictureURL = "/Images/placeholder.gif"},

new UserProfile { UserName = "Bob Martin", Biography = "I create hand crafted canoes from wood.", PictureURL = "/Images/placeholder.gif"},

new UserProfile { UserName = "Tanya Do", Biography = "I create oil paintings of landscapes.", PictureURL = "/Images/placeholder.gif"},

new UserProfile { UserName = "John Robison", Biography = "I create robotic systems.", PictureURL = "/Images/placeholder.gif"},

new UserProfile { UserName = "Paul MacNeil", Biography = "I create 3D printed sculptures.", PictureURL = "/Images/placeholder.gif"}

};

new List<Product>

{

new Product { Name = "Robot", Category = categories.Single(c => c.Name == "Robotics"), Price = 19.99M, User = users.Single(u => u.UserName == "John Robison"), PictureURL = "/Images/placeholder.gif"},

new Product { Name = "Painting", Category = categories.Single(c => c.Name == "Art"), Price = 9.99M, User = users.Single(u => u.UserName == "Tanya Do"), PictureURL = "/Images/placeholder.gif"},

new Product { Name = "Song", Category = categories.Single(c => c.Name == "Music"), Price = 9.99M, User = users.Single(u => u.UserName == "Micheal Rose"), PictureURL = "/Images/placeholder.gif"},

new Product { Name = "Canoe", Category = categories.Single(c => c.Name == "Wood Working"), Price = 109.99M, User = users.Single(u => u.UserName == "Bob Martin"), PictureURL = "/Images/placeholder.gif"},

new Product { Name = "Sculpture", Category = categories.Single(c => c.Name == "3D Printing"), Price = 9.99M, User = users.Single(u => u.UserName == "Paul MacNeil"), PictureURL = "/Images/placeholder.gif"}

}.ForEach(a => context.Products.Add(a));

}

}

}

## ShoppingCart.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MakerStore.Models

{

public class ShoppingCart

{

MakerStoreEntities storeDB = new MakerStoreEntities();

string ShoppingCartID { get; set; }

public const string CartSessionKey = "CartId";

public static ShoppingCart GetCart(HttpContextBase context)

{

var cart = new ShoppingCart();

cart.ShoppingCartID = cart.GetCartId(context);

return cart;

}

// Helper method to simplify shopping cart calls

public static ShoppingCart GetCart(Controller controller)

{

return GetCart(controller.HttpContext);

}

public void AddToCart(Product Product)

{

// Get the matching cart and album instances

var cartItem = storeDB.Carts.SingleOrDefault(

c => c.CartID == ShoppingCartID

&& c.ProductID == Product.ProductID);

if(cartItem == null)

{

// Create a new cart item if no cart item exists

cartItem = new Cart

{

ProductID = Product.ProductID,

CartID = ShoppingCartID,

Count = 1,

DateCreated = DateTime.Now

};

storeDB.Carts.Add(cartItem);

}

else

{

// If the item does exist in the cart, then add one to the quantity

cartItem.Count++;

}

// Save changes

storeDB.SaveChanges();

}

public int RemoveFromCart(int id)

{

// Get the cart

var cartItem = storeDB.Carts.Single(

cart => cart.CartID == ShoppingCartID

&& cart.ProductID == id);

int itemCount = 0;

if(cartItem != null)

{

if(cartItem.Count > 1)

{

cartItem.Count--;

itemCount = cartItem.Count;

}

else

{

storeDB.Carts.Remove(cartItem);

}

// Save changes

storeDB.SaveChanges();

}

return itemCount;

}

public void EmptyCart()

{

var cartItems = storeDB.Carts.Where(cart => cart.CartID == ShoppingCartID);

foreach(var cartItem in cartItems)

{

storeDB.Carts.Remove(cartItem);

}

// Save changes

storeDB.SaveChanges();

}

public List<Cart> GetCartItems()

{

return storeDB.Carts.Where(cart => cart.CartID == ShoppingCartID).ToList();

}

public int GetCount()

{

// Get the count of each item in the cart and sum them up

int? count = (from cartItems in storeDB.Carts

where cartItems.CartID == ShoppingCartID

select (int?)cartItems.Count).Sum();

// Return 0 if all entries are null

return count ?? 0;

}

public decimal GetTotal()

{

// Multiply album price by count of that album to get

// the current price for each of those albums in the cart

// sum all album price totals to get the cart total

decimal? total = (from cartItems in storeDB.Carts

where cartItems.CartID == ShoppingCartID

select (int?)cartItems.Count \* cartItems.Product.Price).Sum();

return total ?? decimal.Zero;

}

public int CreateOrder(Order order)

{

decimal orderTotal = 0;

var cartItems = GetCartItems();

// Iterate over the items in the cart, adding the order details for each

foreach(var item in cartItems)

{

var orderDetail = new OrderDetail

{

ProductID = item.ProductID,

OrderID = order.OrderID,

UnitPrice = item.Product.Price,

Quantity = item.Count

};

// Set the order total of the shopping cart

orderTotal += (item.Count \* item.Product.Price);

storeDB.OrderDetails.Add(orderDetail);

}

// Set the order's total to the orderTotal count

order.Total = orderTotal;

// Save the order

storeDB.SaveChanges();

// Empty the shopping cart

EmptyCart();

// Return the OrderId as the confimation number

return order.OrderID;

}

// We're using HttpContextBase to allow access to cookies

public string GetCartId(HttpContextBase context)

{

if (context.Session[CartSessionKey] == null)

{

if(!string.IsNullOrWhiteSpace(context.User.Identity.Name))

{

context.Session[CartSessionKey] = context.User.Identity.Name;

}

else

{

// Generate a new GUID using System.Guid class

Guid tempCartId = Guid.NewGuid();

// Send tempCartId back to client as a cookie

context.Session[CartSessionKey] = tempCartId.ToString();

}

}

return context.Session[CartSessionKey].ToString();

}

// When a user has logged in, migrate their shopping cart to

// be associated with their username

public void MigrateCart(string userName)

{

var shoppingCart = storeDB.Carts.Where(c => c.CartID == ShoppingCartID);

foreach(Cart item in shoppingCart)

{

item.CartID = userName;

}

storeDB.SaveChanges();

}

}

}

## ShoppingCartRemoveViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace MakerStore.ViewModels

{

public class ShoppingCartRemoveViewModel

{

public string Message { get; set; }

public decimal CartTotal { get; set; }

public int CartCount { get; set; }

public int ItemCount { get; set; }

public int DeleteId { get; set; }

}

}

## ShoppingCartViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using MakerStore.Models;

namespace MakerStore.ViewModels

{

public class ShoppingCartViewModel

{

public List<Cart> CartItems { get; set; }

public decimal CartTotal { get; set; }

}

}

## \_ChangePasswordPartial.cshtml

@model MakerStore.Models.LocalPasswordModel

<h3>Change password</h3>

@using (Html.BeginForm("Manage", "Account")) {

@Html.AntiForgeryToken()

@Html.ValidationSummary()

<fieldset>

<legend>Change Password Form</legend>

<ol>

<li>

@Html.LabelFor(m => m.OldPassword)

@Html.PasswordFor(m => m.OldPassword)

</li>

<li>

@Html.LabelFor(m => m.NewPassword)

@Html.PasswordFor(m => m.NewPassword)

</li>

<li>

@Html.LabelFor(m => m.ConfirmPassword)

@Html.PasswordFor(m => m.ConfirmPassword)

</li>

</ol>

<input type="submit" value="Change password" />

</fieldset>

}

## \_ExternalLoginsListPartial.cshtml

@model ICollection<AuthenticationClientData>

@if (Model.Count == 0)

{

<div class="message-info">

<p>There are no external authentication services configured. See <a href="http://go.microsoft.com/fwlink/?LinkId=252166">this article</a>

for details on setting up this ASP.NET application to support logging in via external services.</p>

</div>

}

else

{

using (Html.BeginForm("ExternalLogin", "Account", new { ReturnUrl = ViewBag.ReturnUrl }))

{

@Html.AntiForgeryToken()

<fieldset id="socialLoginList">

<legend>Log in using another service</legend>

<p>

@foreach (AuthenticationClientData p in Model)

{

<button type="submit" name="provider" value="@p.AuthenticationClient.ProviderName" title="Log in using your @p.DisplayName account">@p.DisplayName</button>

}

</p>

</fieldset>

}

}

## \_RemoveExternalLoginsPartial.cshtml

@model ICollection<MakerStore.Models.ExternalLogin>

@if (Model.Count > 0)

{

<h3>Registered external logins</h3>

<table>

<tbody>

@foreach (MakerStore.Models.ExternalLogin externalLogin in Model)

{

<tr>

<td>@externalLogin.ProviderDisplayName</td>

<td>

@if (ViewBag.ShowRemoveButton)

{

using (Html.BeginForm("Disassociate", "Account"))

{

@Html.AntiForgeryToken()

<div>

@Html.Hidden("provider", externalLogin.Provider)

@Html.Hidden("providerUserId", externalLogin.ProviderUserId)

<input type="submit" value="Remove" title="Remove this @externalLogin.ProviderDisplayName credential from your account" />

</div>

}

}

else

{

@: &nbsp;

}

</td>

</tr>

}

</tbody>

</table>

}

## \_SetPasswordPartial.cshtml

@model MakerStore.Models.LocalPasswordModel

<p>

You do not have a local password for this site. Add a local

password so you can log in without an external login.

</p>

@using (Html.BeginForm("Manage", "Account")) {

@Html.AntiForgeryToken()

@Html.ValidationSummary()

<fieldset>

<legend>Set Password Form</legend>

<ol>

<li>

@Html.LabelFor(m => m.NewPassword)

@Html.PasswordFor(m => m.NewPassword)

</li>

<li>

@Html.LabelFor(m => m.ConfirmPassword)

@Html.PasswordFor(m => m.ConfirmPassword)

</li>

</ol>

<input type="submit" value="Set password" />

</fieldset>

}

## ExternalLoginConfirmation.cshtml

@model MakerStore.Models.RegisterExternalLoginModel

@{

ViewBag.Title = "Register";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

<h2>Associate your @ViewBag.ProviderDisplayName account.</h2>

</hgroup>

@using (Html.BeginForm("ExternalLoginConfirmation", "Account", new { ReturnUrl = ViewBag.ReturnUrl })) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>Association Form</legend>

<p>

You've successfully authenticated with <strong>@ViewBag.ProviderDisplayName</strong>.

Please enter a user name for this site below and click the Confirm button to finish

logging in.

</p>

<ol>

<li class="name">

@Html.LabelFor(m => m.UserName)

@Html.TextBoxFor(m => m.UserName)

@Html.ValidationMessageFor(m => m.UserName)

</li>

</ol>

@Html.HiddenFor(m => m.ExternalLoginData)

<input type="submit" value="Register" />

</fieldset>

}

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## ExternalLoginFailure.cshtml

@{

ViewBag.Title = "Login Failure";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

<h2>Unsuccessful login with service.</h2>

</hgroup>

## Account/Index.cshtml

@model MakerStore.Models.UserProfile

@{

ViewBag.Title = @User.Identity.Name;

}

<h2>@User.Identity.Name</h2>

<h3>Biography</h3>

<h3>Update Account Information</h3>

<ol>

<li>

@Html.ActionLink("Update Account Information", "Update", "Account")

</li>

<li>

@Html.ActionLink("Manage Password", "Manage", "Account")

</li>

</ol>

## Login.cshtml

@model MakerStore.Models.LoginModel

@{

ViewBag.Title = "Log in";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

</hgroup>

<section id="loginForm">

<h2>Use a local account to log in.</h2>

@using (Html.BeginForm(new { ReturnUrl = ViewBag.ReturnUrl })) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>Log in Form</legend>

<ol>

<li>

@Html.LabelFor(m => m.UserName)

@Html.TextBoxFor(m => m.UserName)

@Html.ValidationMessageFor(m => m.UserName)

</li>

<li>

@Html.LabelFor(m => m.Password)

@Html.PasswordFor(m => m.Password)

@Html.ValidationMessageFor(m => m.Password)

</li>

<li>

@Html.CheckBoxFor(m => m.RememberMe)

@Html.LabelFor(m => m.RememberMe, new { @class = "checkbox" })

</li>

</ol>

<input type="submit" value="Log in" />

</fieldset>

<p>

@Html.ActionLink("Register", "Register") if you don't have an account.

</p>

}

</section>

<section class="social" id="socialLoginForm">

<h2>Use another service to log in.</h2>

@Html.Action("ExternalLoginsList", new { ReturnUrl = ViewBag.ReturnUrl })

</section>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Manage.cshtml

@model MakerStore.Models.LocalPasswordModel

@{

ViewBag.Title = "Manage Account";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

</hgroup>

<p class="message-success">@ViewBag.StatusMessage</p>

<p>You're logged in as <strong>@User.Identity.Name</strong>.</p>

@if (ViewBag.HasLocalPassword)

{

@Html.Partial("\_ChangePasswordPartial")

}

else

{

@Html.Partial("\_SetPasswordPartial")

}

<section id="externalLogins">

@Html.Action("RemoveExternalLogins")

<h3>Add an external login</h3>

@Html.Action("ExternalLoginsList", new { ReturnUrl = ViewBag.ReturnUrl })

</section>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Register.cshtml

@model MakerStore.Models.RegisterModel

@{

ViewBag.Title = "Register";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

<h2>Create a new account.</h2>

</hgroup>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

@Html.ValidationSummary()

<fieldset>

<legend>Registration Form</legend>

<ol>

<li>

@Html.LabelFor(m => m.UserName)

@Html.TextBoxFor(m => m.UserName)

</li>

<li>

@Html.LabelFor(m => m.Password)

@Html.PasswordFor(m => m.Password)

</li>

<li>

@Html.LabelFor(m => m.ConfirmPassword)

@Html.PasswordFor(m => m.ConfirmPassword)

</li>

</ol>

<input type="submit" value="Register" />

</fieldset>

}

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Update.cshtml

@model MakerStore.Models.UserProfile

@{

ViewBag.Title = "Update";

}

<h2>Update</h2>

<h3>Update Account Information</h3>

@if (Request.IsAuthenticated)

{

<fieldset>

<legend>Update Account Information</legend>

<ol>

<li>

@Html.LabelFor(m => m.Biography)

@Html.TextBoxFor(m => m.Biography)

</li>

<li>

@Html.LabelFor(m => m.PictureURL)

@Html.TextBoxFor(m => m.PictureURL)

</li>

</ol>

<input type="submit" value="Update Information" />

</fieldset>

}

else

{

<p>You are not logged in.</p>

}

<fieldset>

<legend>Update Account Information</legend>

<ol>

<li>

</li>

<li>

</li>

<li>

</li>

</ol>

<input type="submit" value="Update Information" />

</fieldset>

## AddressAndPayment.cshtml

@model MakerStore.Models.Order

@{

ViewBag.Title = "AddressAndPayment";

}

<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>

<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")" type="text/javascript"></script>

@using (Html.BeginForm())

{

<h2>AddressAndPayment</h2>

<fieldset>

<legend>Shipping Information</legend>

@Html.EditorForModel()

</fieldset>

<fieldset>

<legend>Payment</legend>

<p>We're running a promotion: all music is free with the promo code: "FREE"</p>

<div class="editor-label">

@Html.Label("Promo Code")

</div>

<div class="editor-field">

@Html.TextBox("PromoCode")

</div>

</fieldset>

<input type="submit" value="Submit Order" />

}

## Complete.cshtml

@model int

@{

ViewBag.Title = "Complete";

}

<h2>Checkout Complete</h2>

<p>Thanks for your order! Your order number is: @Model</p>

<p>How about shopping for some more products in our @Html.ActionLink("store", "Index", "Home")</p>

## About.cshtml

@{

ViewBag.Title = "About SparkMacon";

}

<hgroup class="title">

<h1>@ViewBag.Title</h1>

</hgroup>

<article>

<p>

We believe that supporting creativity and connection between the

arts and technology is how we bring the next great ideas to life.

</p>

<p>

We are a group of government, business and community leaders

spent almost 9 months exploring the idea of putting a MakerSpace

in Macon. From that larger group the small group of “Founders”

you see in our team section took the project on.

These “Founders” represent many aspects of our community and the

desire to bring together and create.

</article>

<aside>

<h3>Links</h3>

<ul>

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul>

</aside>

## Contact.cshtml

@{

ViewBag.Title = "Contact SparkMacon!";

}

<hgroup class="title">

<h1>@ViewBag.Title</h1>

</hgroup>

<section class="contact">

<header>

<h3>Phone</h3>

</header>

<p>

<span class="label">Main:</span>

<span>478-569-6475</span>

</p>

</section>

<section class="contact">

<header>

<h3>Email</h3>

</header>

<p>

<span><a href="mailto:Support@example.com">info@sparkmacon.com</a></span>

</p>

</section>

<section class="contact">

<header>

<h3>Address</h3>

</header>

<p>

557 Cherry Street<br />

Macon, Georgia 31201

</p>

</section>

## Home/Index.cshtml

@{

ViewBag.Title = "The SparkMacon MakerSpace MakerStore";

ViewBag.Message = "Welcome to the makerstore, where makers can showcase their ware. ";

}

@section featured {

<section class="featured">

<div class="content-wrapper">

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

<br />

<h2>@ViewBag.Message</h2>

</hgroup>

<p>

SparkMacon is a community innovation space, also known as a makerspace, equipped with the tools,

equipment, and grass-roots education required to convert your idea into a reality.

Think of us as a gym membership to exercise your creativity, hands and brain.

</p>

</div>

</section>

}

<h3>News </h3>

<ul class="round">

<li class="one">

<h5>First Friday Open Make Night</h5>

First Friday Open Make Night Where: SparkMacon MakerSpace – 557 Cherry Street, Macon,

Georgia 31201 When: November 7th – 5:30pm to 9:30 pm This will be the first of our

monthly events. We will host a 4 hour Making/Hacking event for the public and

<a href="http://go.microsoft.com/fwlink/?LinkId=245151">Read more…</a>

</li>

<li class="two">

<h5>8 Georgia MakerSpaces We Love!</h5>

In our efforts to grow the SparkMacon MakerSpace community, I wanted to shine a light on the other

Maker communities around the state of Georgia that inspire our passion and work.

Everyone loves to be a part of something larger than themselves. I hope

<a href="http://go.microsoft.com/fwlink/?LinkId=245153">Read more…</a>

</li>

<li class="three">

<h5>Thank You for SparkMacon Cleaning Day</h5>

Morning! I just wanted to extend a heart-felt thank you to all the SparkMacon members who worked

our cleaning day this past weekend.

From watching Facebook updates from our SparkMacon friends, the team is still

feeling kind of sore from the work. It really was

<a href="http://go.microsoft.com/fwlink/?LinkId=245157">Read more…</a>

</li>

</ul>

## \_Layout.cshtml

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<title>@ViewBag.Title - MakerStore</title>

<link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />

<meta name="viewport" content="width=device-width" />

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

<header>

<div class="content-wrapper">

<div class="float-left">

<p class="site-title">@Html.ActionLink("MakerStore", "Index", "Home")</p>

</div>

<div class="float-right">

<section id="login">

@Html.Partial("\_LoginPartial")

</section>

<nav>

<ul id="menu">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("Store", "Index", "Store")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

<li><a href="http://www.sparkmacon.com">SparkMacon</a></li>

<li>@{Html.RenderAction("CartSummary", "ShoppingCart");}</li>

</ul>

</nav>

</div>

</div>

</header>

<div id="body">

@RenderSection("featured", required: false)

<section class="content-wrapper main-content clear-fix">

@RenderBody()

</section>

</div>

<footer>

<div class="content-wrapper">

<div class="float-left">

<p>&copy; @DateTime.Now.Year - MakerStore</p>

</div>

</div>

</footer>

@Scripts.Render("~/bundles/jquery")

@RenderSection("scripts", required: false)

</body>

</html>

## \_LoginPartial.cshtml

@if (Request.IsAuthenticated) {

<text>

Hello, @Html.ActionLink(User.Identity.Name, "Index", "Account", routeValues: null, htmlAttributes: new { @class = "username", title = "Manage" })!

@using (Html.BeginForm("LogOff", "Account", FormMethod.Post, new { id = "logoutForm" })) {

@Html.AntiForgeryToken()

<a href="javascript:document.getElementById('logoutForm').submit()">Log off</a>

}

</text>

} else {

<ul>

<li>@Html.ActionLink("Register", "Register", "Account", routeValues: null, htmlAttributes: new { id = "registerLink" })</li>

<li>@Html.ActionLink("Log in", "Login", "Account", routeValues: null, htmlAttributes: new { id = "loginLink" })</li>

</ul>

}

## Error.cshtml

@model System.Web.Mvc.HandleErrorInfo

@{

ViewBag.Title = "Error";

}

<hgroup class="title">

<h1 class="error">Error.</h1>

<h2 class="error">An error occurred while processing your request.</h2>

</hgroup>

## CartSummary.cshtml

@Html.ActionLink("Cart (" + ViewData["CartCount"] + ")", "Index", "ShoppingCart", new { id = "cart-status" })

## ShoppingCart/Index.cshtml

@model MakerStore.ViewModels.ShoppingCartViewModel

@{

ViewBag.Title = "Shopping Cart";

}

<script type="text/javascript" src="http://code.jquery.com/jquery-1.7.1.min.js"></script>

<script type="text/javascript">

$(function () {

// Document.ready -> link up remove event handler

$(".RemoveLink").click(function () {

// Get the id from the link

var recordToDelete = $(this).attr("data-id");

if (recordToDelete != '') {

// Perform the ajax post

$.post("/ShoppingCart/RemoveFromCart", { "id": recordToDelete },

function (data) {

// Successful requests get here

// Update the page elements

console.log(data.ItemCount)

if (data.ItemCount == 0) {

$('#row-' + data.DeleteId).fadeOut('slow');

}

else {

$('#item-count-' + data.DeleteId).text(data.ItemCount);

}

$('#cart-total').text(data.CartTotal);

$('#update-message').text(data.Message);

$('#cart-status').text('Cart (' + data.CartCount + ')');

});

}

});

});

function handleUpdate()

{

// Load and deserialize the returned JSON data

var json = context.get\_data();

var data = Sys.Serialization.JavaScriptSerializer.deserialize(json);

// Update the page elements

if(data.ItemCount == 0)

{

$('#row-' + data.DeleteId).fadeOut('slow');

}

else

{

$('#item-count-' + data.DeleteId).text(data.ItemCount);

}

$('#cart-total').text(data.CartTotal);

$('#update-message').text(data.Message);

$('#cart-status').text('Cart (' + data.CartCount + ')');

}

</script>

<h3><em>Review</em> your cart:</h3>

<p class="button">

@Html.ActionLink("Checkout >>", "AddressAndPayment", "Checkout")

</p>

<div id="update-message"></div>

<table>

<tr>

<th>

Product

</th>

<th>

Price (each)

</th>

<th>

Quantity

</th>

<th></th>

</tr>

@foreach (var item in Model.CartItems)

{

<tr id="row-@item.RecordID">

<td>

@Html.ActionLink(item.Product.Name, "Details", "Store", new { id = item.ProductID }, null)

</td>

<td>

@item.Product.Price

</td>

<td id="item-count-@item.RecordID">

@item.Count

</td>

<td>

<a href="#" class="RemoveLink" data-id="@item.RecordID">Remove from cart</a>

</td>

</tr>

}

<tr>

<td>

Total

</td>

<td></td>

<td></td>

<td id="cart-total">

@Model.CartTotal

</td>

</tr>

</table>

## Browse.cshtml

@model MakerStore.Models.Category

@{

ViewBag.Title = "Browse Albums";

}

<div class="genre">

<h3><em>@Model.Name</em> Products</h3>

<ul id="album-list">

@foreach (var product in Model.Products)

{

<li>

<a href="@Url.Action("Details", new { id = product.ProductID})">

<img alt="@product.Name" src="@product.PictureURL" />

<span>@product.Name</span>

</a>

</li>

}

</ul>

</div>

## CategoryMenu.cshtml

@model IEnumerable<MakerStore.Models.Category>

<ul id="categories">

@foreach (var category in Model)

{

<li>@Html.ActionLink(category.Name, "Browse", "Store", new { Category = category.Name }, null)</li>

}

</ul>

## Details.cshtml

@model MakerStore.Models.Product

@{

ViewBag.Title = "Product - " + Model.Name;

}

<h2>Product: @Model.Name</h2>

<p>

<img alt="@Model.Name" src="@Model.PictureURL" />

</p>

<div id="product-details">

<p>

<em>Category:</em>

@Model.Category.Name

</p>

<p>

<em>User:</em>

@Model.User.UserName

</p>

<p>

<em>Price:</em>

@String.Format("{0:F}", Model.Price)

</p>

<p class="button">

@Html.ActionLink("Add to cart", "AddToCart", "ShoppingCart", new { id = Model.ProductID }, null)

</p>

</div>

## Store/Index.cshtml

@model IEnumerable<MakerStore.Models.Category>

@{

ViewBag.Title = "Store";

}

<h3>Browse Categories</h3>

<p>Select from @Model.Count() categories:</p>

<ul>

@foreach (var category in Model)

{

<li>@Html.ActionLink(category.Name, "Browse", new { category = category.Name })</li>

}

</ul>