Motorbid

**Enterprise Frameworks**

**ASP.NET MVC3 Auto Auction Site**

**Cezar Oranski –x10204814**

**Conor Doherty – x11107685**

**Enterprise Frameworks**

**Diploma in Cloud Computing**

Contents

[1 Executive Summary 3](#_Toc331158971)

[2 Background Research 3](#_Toc331158972)

[3 Project Plan 3](#_Toc331158973)

[4 Software Development Methodology 4](#_Toc331158974)

[5 Wireframes 5](#_Toc331158975)

[6 Requirements 5](#_Toc331158976)

[7 Architecture 7](#_Toc331158977)

[8 Models 7](#_Toc331158978)

[9 User Interface 13](#_Toc331158979)

[10 Security 17](#_Toc331158980)

[11 Configuration 17](#_Toc331158981)

[11.1 EF Scaleability from Server CE to SQL Server 17](#_Toc331158982)

[12 Scaleability 18](#_Toc331158983)

[13 Testing Approach 19](#_Toc331158984)

[14 Conclusion 19](#_Toc331158985)

[15 References 20](#_Toc331158986)

# Executive Summary

Motorbid is a vehicle auctions website for motor dealers. The website is a 5 layer enterprise architecture built using ASP.NET MVC3 and Entity Framework. A motor dealer receives cars from clients and lists them on the website on behalf of clients. Buyers browse the car offers and either purchase a car at the reserve price by going to a shopping cart and submitting an order to the dealer or by placing a bid on a car of interest. Bidding has an expiry date. If a car has not been purchased at the reserve price, at the bidding expiry date, the top bid that meets the owner’s minimal price criterion is selected as the winning bid. The dealer administrator notifies the client by phone that the car has been successfully sold by auction bid based on bidding history that Manager has access to. The dealer administrator also notifies the buyer by email of his winning bid and/or in the case of reserve price purchase of arrangements for car delivery. At the moment only Visa credit card is accepted as form of payment, but in a real world scenario our website would connect to a secure third party payment processing facility. An attempt was made in the project to make the website more scaleable by migrating from a database bottleneck Server Compact Edition to SQL Server.

# Background Research

General web auction sites such as eBay and specialist motor vehicle auction sites such as [www.british-car-auctions.co.uk](http://www.british-car-auctions.co.uk) (BCA) have become a popular way for vehicle owners and buyers to directly transact whilst avoiding dealers commissions. Web auctions break down and remove the physical limitations of traditional auctions such as geography, presence, time, space, and a small target audience. Cezaz has a brother who has spent 20 years trading cars. He mainly buys cars in Germany ad Holland through BCA car auctions, and periodically collects batches of 6 to 8 cars every couple of weeks. His monthly turnover is around 15-20 cars during peak season. He has also established a wide range of connections with multiple dealers from whom he purchases vehicles online and in person, who might also be interested to join forces with him to start a new business if an attractive business idea arised. Cezar suggested to his brother to lauch an Auto Auction site if they can secure relevant funding and human resources. This project signifies an initial effort to create a prototype electronic car auction site with possibility to expand its functionality in the future. Modern auction sites like BCA conduct at least three different forms of Auctions, such as Live Online Auction (live streaming), Land-based Auction (where buyers are invited to come in person) and Electronic Auto Auction (like E-bay). This prototype has encouraged the possibilties that a credible car auction website could take business to the next level and if successful acquire good public exposure and substantial increase in turnover and sales. If multiple dealers could be persuaded to display their vehicles on a commercial website, this would provide a basis for successfully expanding the car trading business, such as introducing previously described forms of car auctions as well as creating new braches in new locations. ASP.NET MVC framework represents a good platform for developing and building scaleable Enterprise level web applications. We intend to expand our knowledge in pursuit of moving from a prototype to a commercial car auction web site for demonstration to potential investors and partners.

# Project Plan

Initially there were four members of the project. The approach strategy was to adopt an agile approach to design and test driven development that emphasised close communication and cooperation amongst team members whilst keeping documentation lean. The team brainstormed on possible projects and finalised on an auctions website. The next step happened quickly. 3 members of the project produced wireframes for the auction website. To get the project off the ground, the project examined the Microsoft www.asp.net website for help to learn ASP.net MVC. We discovered a music store tutorial [1]. We decided to follow the tutorial as a basis for generating motorbid auction website. We spent time learning the tutorial. The tutorial provided us with a basic understanding of how to build and administrate a web catalog with a shopping cart. We planned to modifiy the music store catalog to a motor vehicle catalog, add bidding functionality, refactor the music store into business logic and data access layers, connect it to a more scaleable database and unit test our work. Initially the group communicated via email and skype but this communication broke down. When it came time to assign development work to individuals, some were not available to commit to tasks so the project coding start date slipped. Two members of the group were committed to producing deliverables on the project and the other two members were otherwise committed. The project development schedule slipped until it came to a critical phase at a project review in class. Adrian Skehill sent out a warning email on 12/07/2012 that all members of the project had to attend a review meeting on 14/07/2012 in NCI. Two project members did not attend the meeting and Adrian Skehill decided to remove them from the project. This was a critical watershed moment for the project. Without the handoffs and excuses from the excluded two members, the remaining two members of the project were free to get on in the last two weeks of the project after a late start to deliver on website functionality. We came into the college most days for the remaining two weeks to work together all day on getting the application finished. Cezar converted initial Music Store Application into Car sales Catalog, whereas Conor joined forces later to contribute towards the bidding functionality. Both of us worked on trying to migrate the application from Server CE to SQLServer using the EntityFramework. Conor worked on building a Business Logic Layer and a Data Access Layer and documentation. 4 unit tests were implemented wuth Cezar and Conor contributing 2 of them respectively.

# Software Development Methodology

We chose an agile software development methodology. We decided to first develop wireframes to drive use case development. Next we followed the MVC3 MusicStore tutorial [1] to determine how this related to the development of an auction website. We read a C# coding book [2] for familiarisation and then reviewed an ASP.NET MVC3 reference book [3] to flesh out our understanding. Cezar performed the initial refactoring of the MusicStore models, views, and controllers to generate a vehicle auction catalog instead of a music store catalog. Cezar and Conor then pair programmed on the bidding functionality and attempted to convert the music store to connect to a SQL Server database instead of a SQL Compact Edition database for scaleability. Conor refactored a Business Logic Layer and a Data Access Layer. Cezar and Conor contributed equally to create 4 simple Unit Tests in total, covering some models and controllers. Conor did the documentation, whereas Cezar wrote chapter on Background and reviewed documentation applying occasional changes throughout this report.

# Wireframes

The first agile development step was to produce Motorbid wireframes. See Figure 1 and Figure 2.

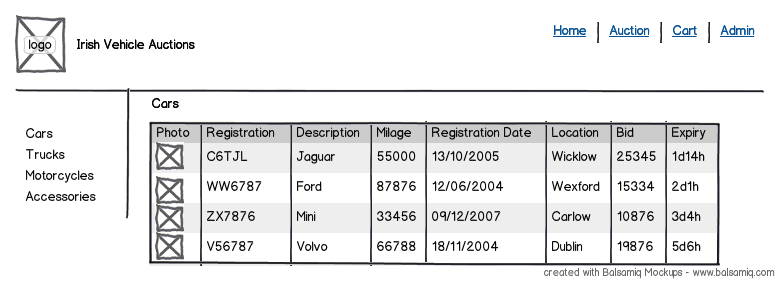


Figure 1. Motorbid Vehicle Browse Wireframe.

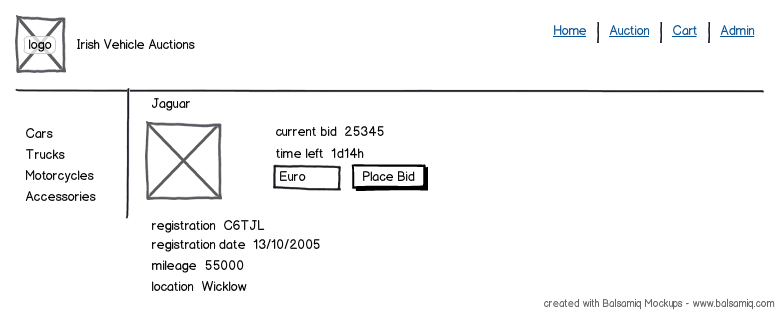


Figure 2. Motorbid Vehicle Details Wireframe.

# Requirements

The website is a functional prototype of a dealer vehicle auction site with minimal site data population and sparse look and feel. The auction model is that of a motor dealer who recieves cars from clients and lists them on the website on behalf of clients. Buyers browse the car offers and either purchase a car at the reserve price by going to a shopping cart and submitting an order to the dealer or by placing a bid on a car of interest. Bidding has an expiry date. If a car has not been purchased at the reserve price, at the bidding expiry date, the top bid that meets the owner’s minimal price criterion is selected as the winning bid. However, Administrator may choose not to sell the car if some top bid happens to be still a way below of the reserve price and not in the interest of Motor Trade Dealer to sell it. In other words, only successful bidder as a registered user is contacted directly upon Expiry Date of an auction. The dealer administrator notifies the client by phone that the car has been successfully sold by auction bid or reserve price purchase. The dealer administrator also notifies the buyer by email of his winning bid or in the case of reserve price purchase of arrangements for car delivery. Payment is currently outside the scope of the application, but in a real world scenario effort would be made to link our site to a secure third party payment processing facility that has multiple forms of payments like credit/debit cards, Paypal, Moneybookers, E-Wallet, NetTeller etc. The main Motorbid use cases are presented in Figure 3.

buyer

administrator

Figure 3. Motorbid Use Cases

# Architecture

SQLServer

**DAL**

Homedal.cs

Catalogdal.cs

CatalogManagerdal.cs

CheckOutdal.cs

ShoppingCartdal.cs

**BLL**

ShoppingCart.cs

**Controllers**

HomeController.cs

AccountController.cs

CatalogController.cs

BiddingController.cs

CatalogManagerController.cs

CheckOutController.cs

ShoppingCartController.cs

**Models**

AccountModel.cs

Auto.cs

Bid.cs

Brand.cs

Car.cs

Cart.cs

Order.cs

OrderDetail.cs

AutoAuctionEntities.cs

**Views**

Home

Account

Catalog

Bid

CatalogManager

Checkout

Shared

ShoppingCart

Figure 4. MotorBid 5 Architecture

The architecture of Motorbid is a 5 tier enterprise framework based on ASP.NET MVC3. The Presentation Layer is composed of controllers and views. The presentation Layer talks to the Business Logic Layer and the Data Access Layer. The Business Logic Layer also talks to the Data Access Layer. The Data Access Layer talks to a SQLServer database. The Data Access Layer and the Views depend on the Models. The architecture basically extends the classic Music Store MVC3 reference architecture by introducing a Business Logic Layer and a Data Access layer. Razor View Engine is used in the Views.

# Models

The following models were generated for the application.

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Globalization;

using System.Web.Mvc;

using System.Web.Security;

namespace Mvc3ToolsUpdateWeb\_Default.Models

{

public class ChangePasswordModel

{

[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 LogOnModel

{

[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]

[DataType(DataType.EmailAddress)]

[Display(Name = "Email address")]

public string Email { 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; }

}

}

Figure 5. AccountModels.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace MvcAutoAuction.Models

{

public class Auto

{

public int AutoId { get; set; }

public string Name { get; set; }

}

}

Figure 6. Auto.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace MvcAutoAuction.Models

{

public partial class Brand

{

public int BrandId { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public List<Car> Cars { get; set; }

}

}

Figure 8. Brand.cs

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Web.Mvc;

namespace MvcAutoAuction.Models

{

[Bind(Exclude = "BidId")]

public partial class Bid

{

[ScaffoldColumn(false)]

public int BidId { 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 = "Car Brand and Year")]

[DisplayName("Car Details")]

[StringLength(160)]

public string AutoInfo { 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 is not valid.")]

[DataType(DataType.EmailAddress)]

public string Email { get; set; }

[Required(ErrorMessage = "Please enter Your Bid in digits")]

[DisplayName("Your Bid")]

public decimal BidAmount { get; set; }

[ScaffoldColumn(false)]

public decimal Total { get; set; }

public virtual Car Car { get; set; }

}

}

Figure 7. Bid.cs

using System;

using System.Linq;

using System.Web;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Web.Mvc;

using System.Collections.Generic;

namespace MvcAutoAuction.Models

{

[Bind(Exclude = "CarId")]

public class Car

{

[ScaffoldColumn(false)]

public int CarId { get; set; }

[DisplayName("Brand")]

public int BrandId { get; set; }

[DisplayName("Auto")]

public int AutoId { get; set; }

[Required(ErrorMessage = "Car Title is required")]

[StringLength(16)]

public string Title { get; set; }

[Required(ErrorMessage = "Price")]

[Range(100.00, 100000.00,

ErrorMessage = "Price must be between 100.00 and 100000.00")]

public decimal Price { get; set; }

[DisplayName("Image URL")]

[StringLength(1024)]

public string Image { get; set; }

public virtual Brand Brand { get; set; }

public virtual Auto Auto { get; set; }

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

}

}

Figure 9. Car.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.ComponentModel.DataAnnotations;

namespace MvcAutoAuction.Models

{

public class Cart

{

[Key]

public int RecordId { get; set; }

public string CartId { get; set; }

public int CarId { get; set; }

public int Count { get; set; }

public System.DateTime DateCreated { get; set; }

public virtual Car Car { get; set; }

}

}

Figure 10. Cart.cs

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Web.Mvc;

namespace MvcAutoAuction.Models

{

[Bind(Exclude = "OrderId")]

public partial class Order

{

[ScaffoldColumn(false)]

public int OrderId { get; set; }

[ScaffoldColumn(false)]

public System.DateTime OrderDate { 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 = "County is required")]

[StringLength(40)]

public string County { 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 is not valid.")]

[DataType(DataType.EmailAddress)]

public string Email { get; set; }

[ScaffoldColumn(false)]

public decimal Total { get; set; }

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

}

}

Figure 11. Order.cs

namespace MvcAutoAuction.Models

{

public class OrderDetail

{

public int OrderDetailId { get; set; }

public int OrderId { get; set; }

public int CarId { get; set; }

public int Quantity { get; set; }

public decimal UnitPrice { get; set; }

public virtual Car Car { get; set; }

public virtual Order Order { get; set; }

}

}

Figure12. OrderDetail.cs

using System.Linq;

using System.Web;

using System.Data.Entity;

namespace MvcAutoAuction.Models

{

public class AutoAuctionEntities : DbContext

{

public DbSet<Car> Cars { get; set; }

public DbSet<Brand> Brands { get; set; }

public DbSet<Auto> Autos { get; set; }

public DbSet<Bid> Bids { get; set; }

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

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

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

}

}

Figure13. AutoAuctionEntities.cs

# User Interface

Screenshots are provided below for the key functionality of Motorbid.

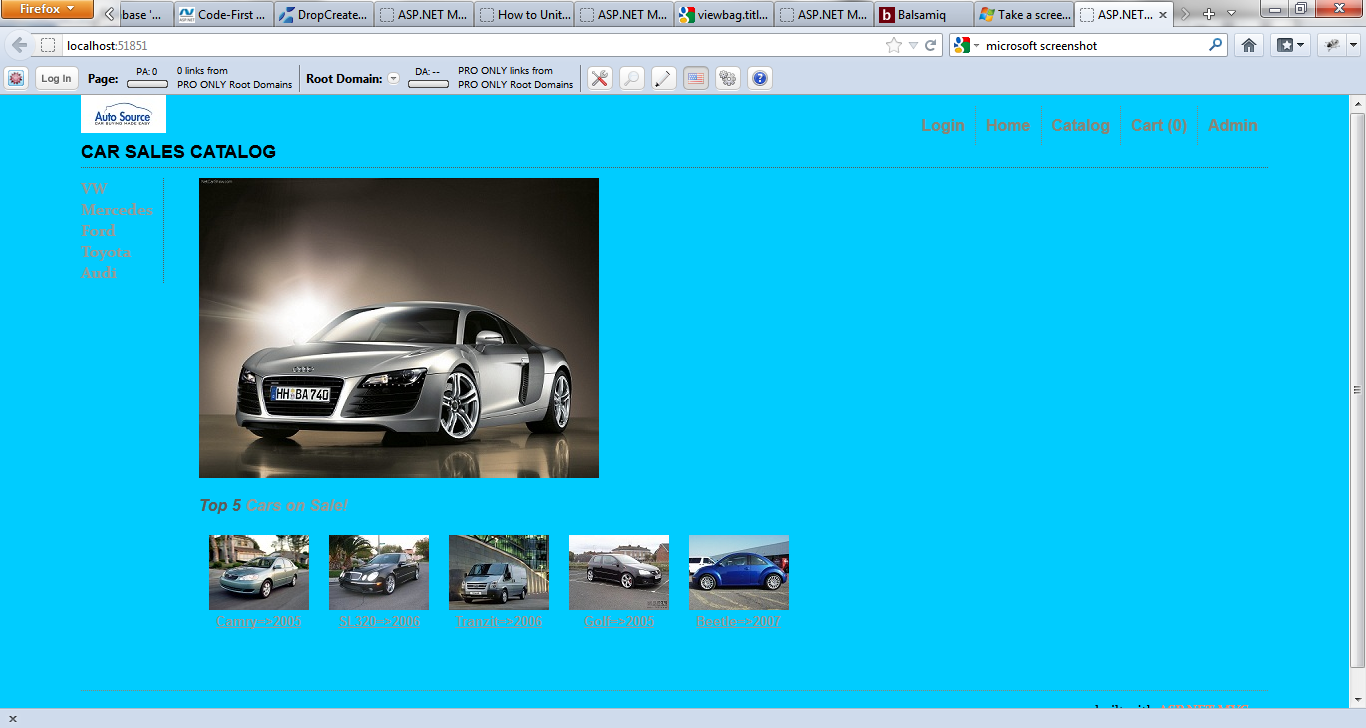


Figure 8. Motorbid Homepage

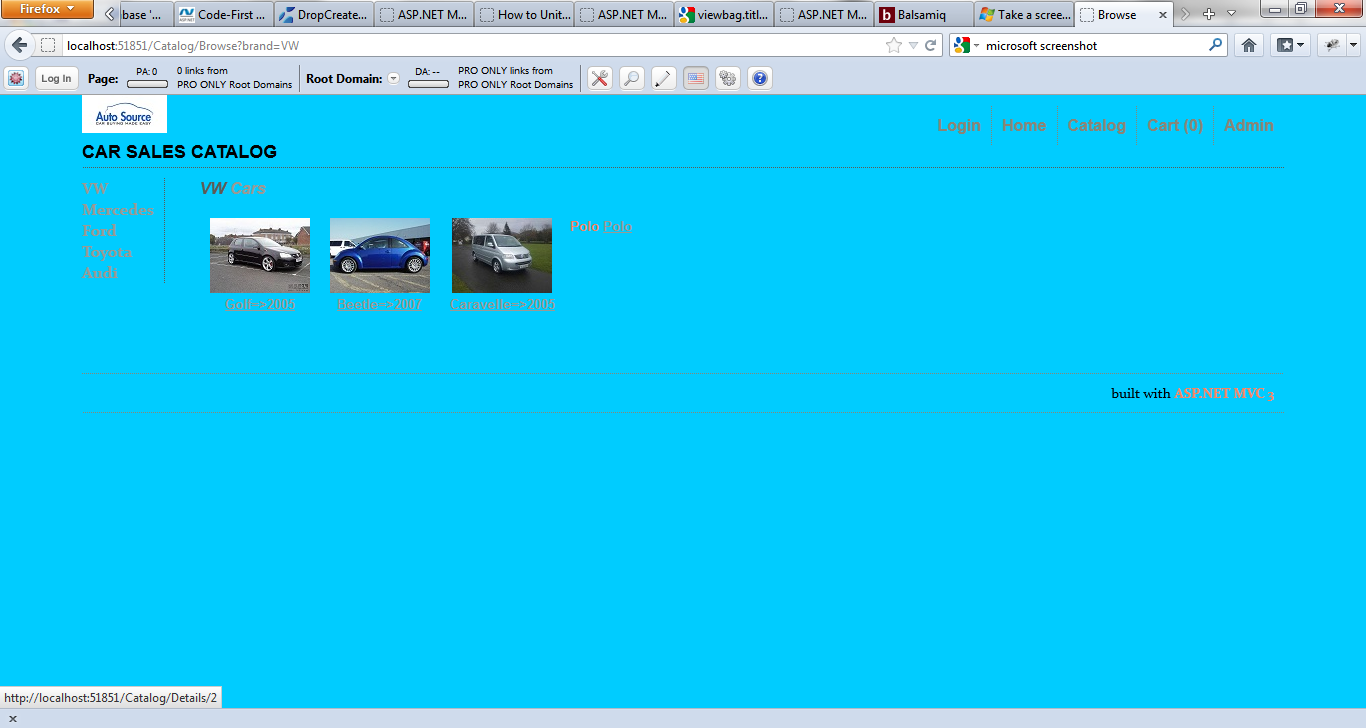


Figure 9. Motorbid Catalog

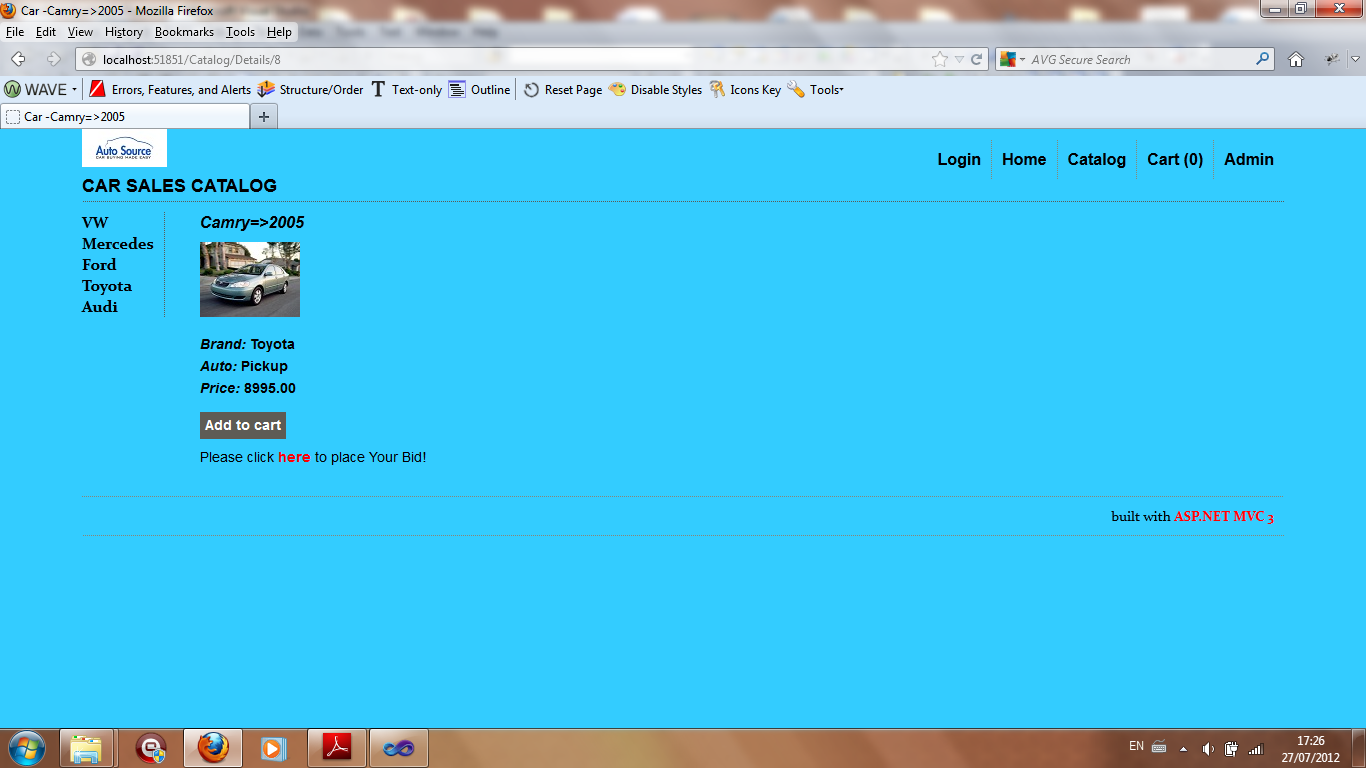


Figure 10. Motorbid Vehicle Detail

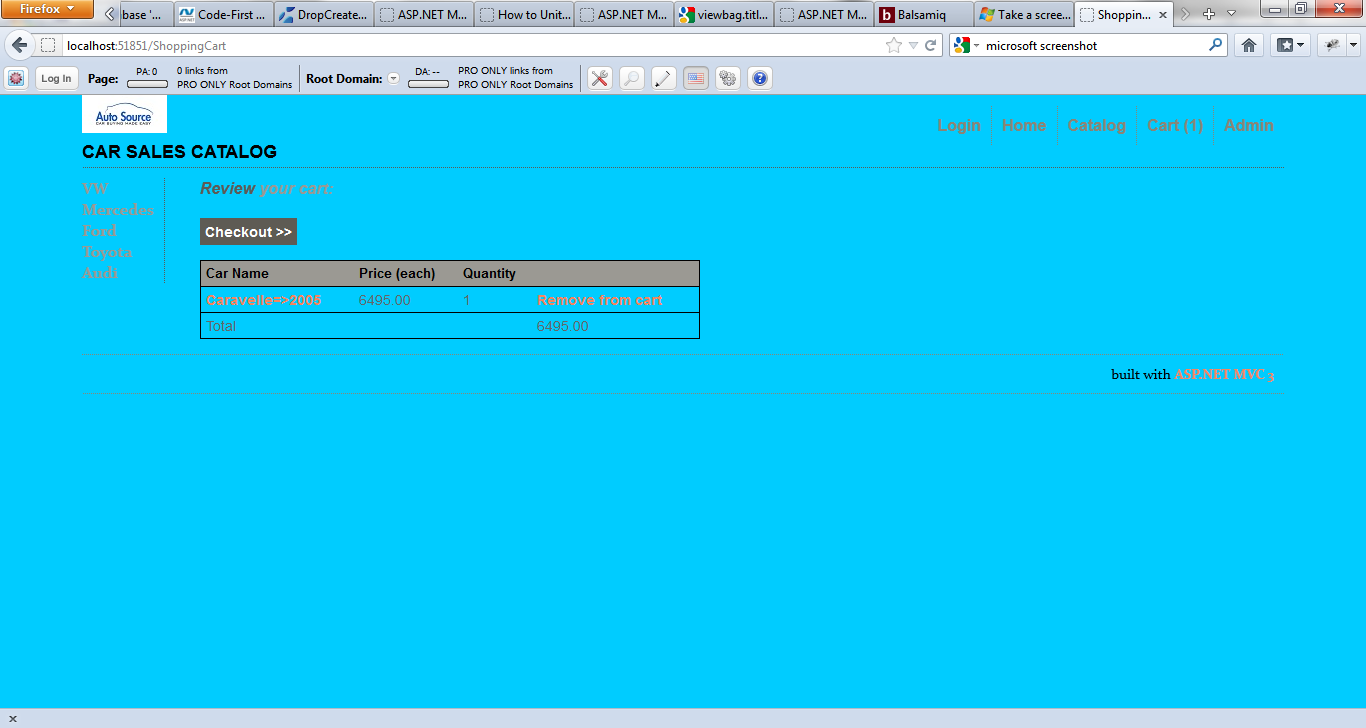


Figure 11. Motorbid Shopping Cart

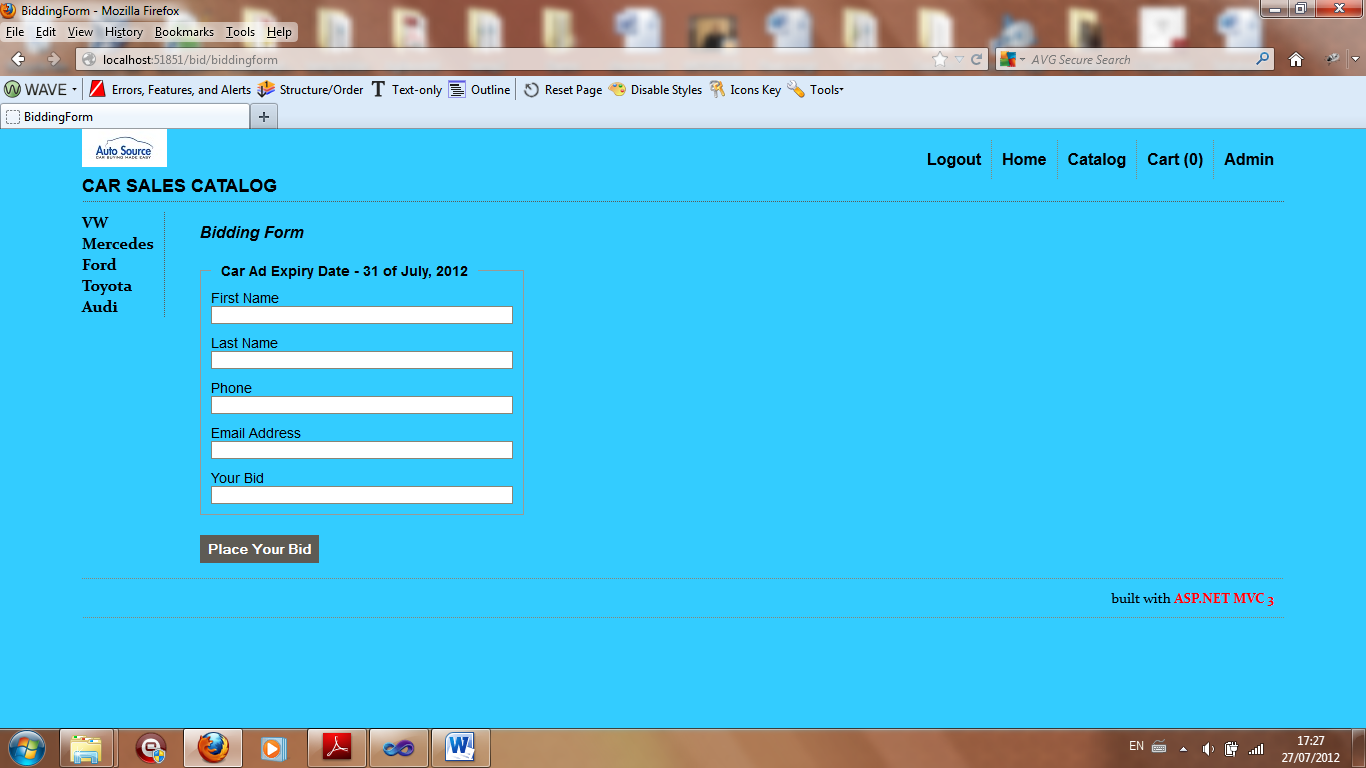


Figure 12. Motorbid Enter Bid

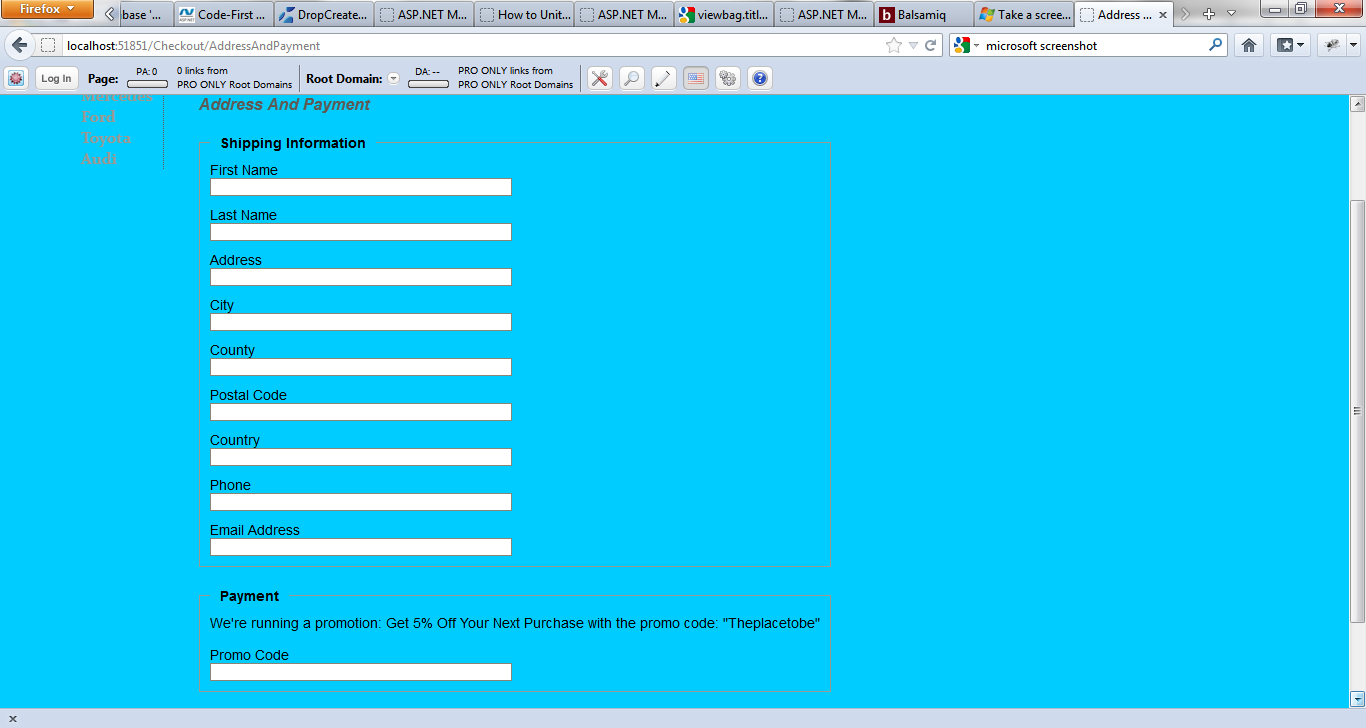


Figure 13. Motorbid Checkout



Figure 14. Motorbid Administration



Figure 15. Bidding History

# Security

Motorbid has user registration, authentication, and authorisation features for buyers and an administrator. ASP.net configuration website was used to set up an administrator role and password. Role restricted access for this administrator was implemented for the CatalogManagerController. Buyers must register and login before they can checkout from the shopping cart and place an order and also before they place a bid. Cookies are used to store user information. The website does not explicitly deal with security as a cross cutting concern between the five layers using advanced patterns such as aspect oriented programming.

# Configuration

## EF Scaleability from Server CE to SQL Server

We decided to use the Entity Framework [4] as an Object Relational Mapper. The Entity Framework is a set of technologies in ADO.NET that support the development of data-oriented software applications. Developers must model the entities, relationships, and logic of the business problems they are solving, and they must also work with the data engines used to store and retrieve the data. The Entity Framework enables developers to work with data in the form of domain-specific objects and properties without having to concern themselves with the underlying database tables and columns where this data is stored.Vehicle entries that include description and images are added and deleted directly on client-side from DataBase–SQL Compact Edition, using CRUD Form functionality available within Admin section of the Enterprise Level Web Application.  .Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework’s ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals.

We had major problems refactoring the Music Store tutorial to get it to work with SQL Server instead of its default Microsoft SQL Server Compact Edition (CE) in order to increase scaleability. Microsoft SQL Server Compact (SQL CE) is a compact relational database for applications that run on mobile devices and desktops. SQL Server Compact shares a common API with the other Microsoft SQL Server editions It also includes ADO.NET providers for data access using ADO.NET APIs, and built-in synchronization capabilities, as well as support for LINQ and Entity Framework. Unlike other editions of Microsoft SQL Server, SQL CE runs in-process with the application which is hosting it. It has a disk footprint of less than 2 MB and a memory footprint of approximately 5 MB.

We should have tried to get our application working with SQL Server instead of SQL Server CE as our first refactoring move to verify that we had a scaleable MVC three layer - client, presentation, and scaleable database architecture. Instead we waited till we had a five layer architecture before trying to refactor our application from SQL CE to SQL Server. Refactoring turned out to be a major challenge that was very time consuming.

First Conor tried variations of connection strings in web.config to access SQL Server instead of SQL CE. Then we tried a code first approach using our models to generate a SQL Server datatbase, tables, and relationships using Entity Framework ORM. We succeded in creating a database, but cound not generate tables and relationships, and therefore couldn’t couple it with our 5 layer Enterprise Level Application.

Next Cezar created and handcoded tables and relationships in our Entity Framework DBcontext database in SQLServer and tried various connection strings to connect this up to our Entity Framwork models. This also proved fruitless.

Finally Cezar consulted [5] on ASP.NET MVC 4 on Entity Framework connection to a database. He created an EFAdapter, a EF Repository, an IRepository and a Ninject Controller Factory (DI Tool) and tried to bind to SQL Server. This complex approach that gave us some exposure to dependency injection pattern in the DAL also did not work.

## GitHub Repositories

We created three github repositories for our project

1. <https://.github.com/DublinerNCI1/ent_frame_project.git>
2. <https://github.com/DublinerNCI1/autoauction.git>
3. <https://github.com/DublinerNCI1/AAuctionV1Demo.git>

We uploaded the music store tutorial for our study to 1. We created 2. to coordinate most of our work. At the end of the project there were github synchronisation issues with 2. So we created 3. to showcase our final working demo code.

# Scaleability

If the application is deployed to Azure, Azure can provide user configured scaleout of the IIS and ASP.NET MVC web worker roles until the database becomes the bottleneck.

The first version of our application connected to SQL CE is not very scalable given the disk footprint of 2MB and in memory footprint of 5 MB.

The second version of our application connected to SQL Server would be much more scaleable and could be clustered if it became a bottleneck for our application.

The use of a 5 layer enterprise framework provides software development scaleability by separation of concerns between the layers. This makes it easier for teams of developers to more independently work on different layers to interfaces between layers. For instance if advanced patterns are used such as dependency injection using ninject, DAL developers can develop plugins for different databases without interfering with interfaces to the BLL.

# Testing Approach

We generated four passing unit tests for HomeController, HomeDAL, CatalogDAL, and CatalogManagerDAL. It was necessary to comment out some Visual Studio Code generation in the test stubs to get the tests to run on IIS.

[TestMethod()]

//[HostType("ASP.NET")]

//[AspNetDevelopmentServerHost("C:\\autoauction5\\MvcAutoAuction", "/")]

//[UrlToTest("http://localhost:51851/")]

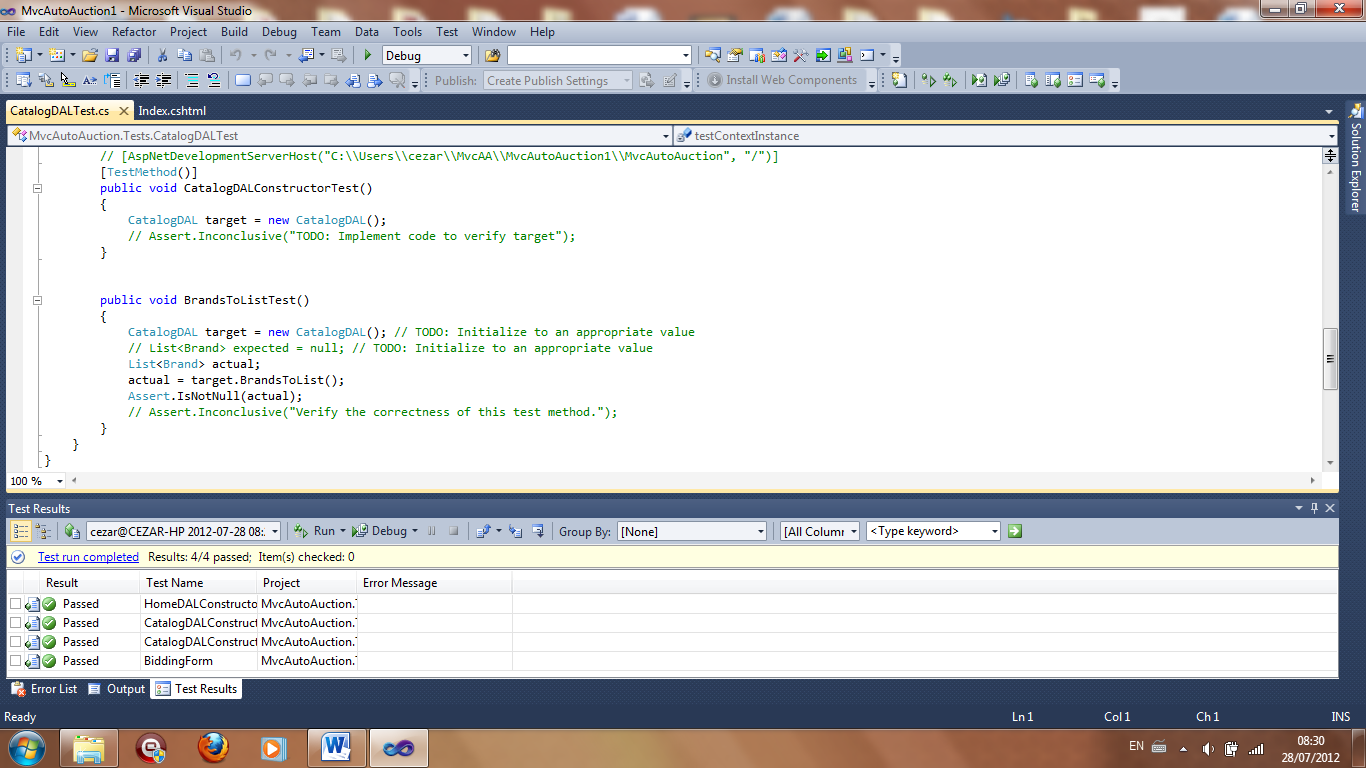


Figure 16. 4 Passing Unit Tests

# Conclusion

We learnt the basics of enterprise frameworks using ASP.NET MVC3 and C# using [1][2][3][4][5][6][7]. The Motorbid website is an interesting prototype for a motor dealer auction website that fulfills the criteria of a simple enterprise framework used to build a 5 layer architecture based on ASP.NET MVC3 with Entity Framework. The website prototype is far from an enterprise website however. A second version of the website could be extended to incorporate more complex vehicle descriptions, search, and electronic payment. It could also permit private sellers to directly list vehicles without intermediation of a dealer for viewing by private buyers. It could also incorporate live streaming of car viewings at our auctions (and associated auctions from dealers to other dealers or private buyers). At a technical level, more design patterns could be succcessfully incorporated such as dependency injection using popular Dependency Injection Tool like Ninject.

# References

[1] <http://www.asp.net/mvc/tutorials/mvc-music-store>

[2] Andrew Stellman and Jennifer Greene (2010). Head First C#. O’Reilly.

[3]Adam Freeman and Stephen Sanderson (2011). Pro ASP.NET MVC3 Framework. Apress.

[4] <http://msdn.microsoft.com/en-us/library/bb399572>

[5] Applied ASP.NET 4 in Context (2011). Adam Freeman. Apress.

[6] <http://stackoverflow.com>

[7] <http://ASP.NET.com>