Programming and Technology

Andrei Ciobanu
Delyan Iliev
Vasil Nedelchev

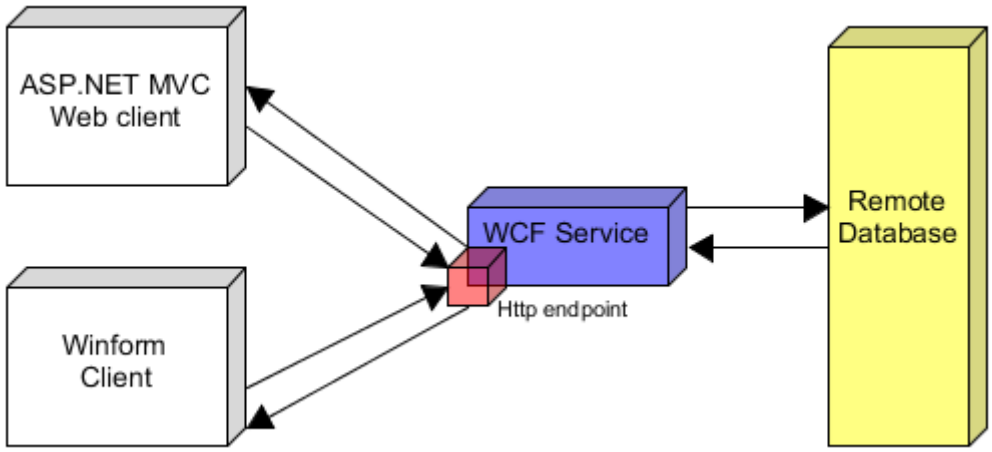
1. Introduction

The aim of this report is to provide a comprehensive guide regarding the system's architecture we made and reflect upon the various choices that have been made regarding different tools, frameworks, protocols included. Before we move on, to the application’s functionality, let’s take a look at the distributed system definition. It varies slightly across literature, but all sources agree that what makes a system distributed is the fact that multiple of its core components are hosted on multiple machines. The distributed system we have developed for third semester exam aims at establishing a platform where people can put up their goods for sale or buy them, in the form of an auction.

1.1 Problem statement

The use of distributed system started more than 50 years ago, and overtime it proved to be extremely relevant in computer communication over network, being described as a model in which components located on networked computers communicate and coordinate their actions by passing messages. This semester project application fits the best to such system type which will ensure a good communication between multiple users over the internet through a secure network. An auction system has to be highly available to access the resources , consistent, performant and fault-tolerant, characteristics successfully covered by a distributed system components.

2. Distributed system architecture



The two sides components of a distributed system are :

1. Client side - where the user has access to the program's interface, reflecting upon the design and functionality.
2. Server side - responsabile with handling most of the functionality and accessing the database, the server side plays an important role, being the intermediate between the client side and database.

The Aalborg Auction program architecture includes for the client side an ASP.NET MVC for the website and a Winform for the website administrator access, and a WCF Service, which represents the server side.

The three different solutions for both server and client side were made, to increase the accessibility through the source code as well as to show that there is no inner connection between the different parts of the application, the only communication being through the WCF service.

I. "AuctionProject" (WCF) solution:

The solution where the service lies. The service has access to the core classes and controllers which perform the business logic and make contact with the database. In order to run the other two applications, the server has to be started.

II. "AuctionProject" (Winform) solution:

The Winform client is where the managerial functionality lies. It is intended to be used by website's administrators to moderate and remove any inappropriate auctions that have been made, as well to help users who run into system difficulties or have come across bugs.

III. "AuctionWebApplication" (ASP.NET MVC) solution:

The regular user should be able to use the ASP.NET web client to its full potential and have access to most of the functionality.
The system fits the Wolfgang Emmerich's definition of a distributed system, being a" collection of autonomous components, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility", due to the following characteristics:

- **Multiple autonomous components** represented by separate clients, a service host and a database to store the user's data, granting the permission of accessing it during the application lifetime.
- **Concurrent processes** become relevant whenever multiple users try to place a bid at simultaneously.
- **The possibility of sharing resources** serves as a common access to the database, as a user, for a potentially buying process or simply informing note.

figure 3.0

3. "AuctionProject" (WCF) solution:

3.1 Architecture

As in the previous projects, our team has to ensure a layered architecture. This allows abstraction of the different layers and their specific function from one another.

In this solution we have five main projects:

- **"Core"** project – Our domain classes with the fields related to database tables lie here. The "Core" layer has reference to the "Database" layer (DAL), "WcfServiceLibraryAuction", and "BusinessLayer" layer (BL) to ensure a proper information flow from the client side to the server side and then to the database tables.
- "**Database"** project– provides communication with the data stored in the SQL Server, being connected the intermediate between the "BusinessLayer" layer and the database content.
- **"BusinessLayer"** project– here lies the business logic for the application, and it is directly accessed by the WCF server.
- **"WcfServiceLibraryAuction"** project – here is defined WCF Interface and the implementation of it. A self-hosted service has the advantage of being flexible due to its lifetime control through the Open() and Close() methods of ServiceHost<T>, easy to use and deploy.
- **"UnitTestProject1"** project - the tests made for the entire solution are in this solution

3.2 Core

In Core project there are five domain classes.

As we used Entity framework **Code - First approach** for creating the database tables we needed to mark some of the properties with attributes like:

- **[key]** – entity framework to create primary key for
- **[NotMapped]** – Denotes that a property should be excluded from database mapping
- **[StringLength(450)]** – Specifies the maximum length of characters
- **[Index(IsUnique = true)]** – indicates that the database column to which the property is mapped has an index
- **[ForeignKey("Category")]** – Denotes a property used as a foreign key because it is used Entity framework **Code - First approach** which is not ideal for the particular property. The use of WCF required us to put data contracts and data members on our classes. But we did not use it everywhere as it by default is serializing everything.

Figure 1.0

Figure 1.1 is an example from Account.cs

3.3 Database

3.3.1 Choice of method

Before we talk about the database layer and design let's talk about our choice of method.

Entity Framework (EF) is an Object/Relational Mapping framework that uses an existing infrastructure of ADO.NET (Troelsen 2012). Its main purpose is to lessen the gap between the objects and the relational data, as well as serving as an intermediate.

One of the major factors that influenced the choice of going with EF for this project, was the fact that it was a completely new technology. Up to this point, we have not ever used an object-relational mapping tool, and we saw this as an opportunity to expand our arsenal of skills and gain new experience. Ado.Net felt too much like JDBC, an external library that was used to establish the communication between the java application and the SQL database for the previous semester. Working with Ado.net felt like a carbon copy of JDBC, and using it in the third semester project would rob us of the learning experience we will get from choosing EF.

In any case, both of them have their pros and cons, and we will compare them in terms of Development speed, Flexibility, and Performance.

- Development speed

The clear winner in this category is EF. It offers the creation of a database in a code-first approach. Everything is done for you by EF, including the connection string. The same goes for all the tables and relations between them in the database. Variables in classes that contain the word "ID" are detected and automatically assigned as main key due to EF conventions.

Naturally, with all automated processes, the possibility of logical errors are always present. The relations and dependencies should always be manually

checked so that you can confirm that they adhere to the principles of normalization.

- Flexibility

It is safe to say that Ado.net offers a greater range of control when working with mapping database information to object and viceversa. Working with direct SQL statements in Ado.net allows for more precise and flexible commands to be executed by the user.

- Performance

When it comes to simple CRUD operations, both EF and ADO.NET perform nearly identical, the execution of the operations is at the same amount of time.

The clear winner for us was EF. Our program did not require that much of flexibility and we valued the development speed more than the flexibility. The development speed was not only important to us as how fast we can create the database, but also that EF requires less code to be written in order to execute the same functions. This gave us the opportunity to focus on the application and not worry about writing manual queries, which is also a lot riskier of typing errors.

3.3.1.1 Creating Context

In order to create context, which represents a session with the database, allowing us to query and save data, we define a context that derives from System.Data.Entity.DbContext and exposes a typed DbSet<TEntity> for each class in our model.The Entity Framework NuGet package provided us all the tools and services needed for the Entity Framework.

Figure 1.2 is complete listing of what Dbcontext.cs contains.

3.3.1.2 Automatic migrations

To reflect upon the changes in the database, Automatic migration is used. Via package manager console we could enable the migrations and create a folder called Configurations.cs with the functionality.AutomaticMigrationsEnabled was set to true, which mean it does not mean to use the Add-Migration command to synchronize the database with our changes. (Figure 3.3.5.1)

Figure 2.2

In DbContext.cs class's constructor it sets the database initializer to Migrate database to latest version as initializing a new instance of the MigrateDatabaseToLatestVersion class that will use context and configuration described in *Figure 2.3.*

3.3.1.3 Model builder

Model builder is used for setting true value to cascade delete, for product property (which is foreign key in database for the auction table) in the Auction class, which is not set by

convention. The code first fluent API is accessed by overriding the "OnModelCreating()" method from DbContext class.
Figure 2.4

3.3.2 Generic repository pattern

While creating our "Database" Layer, we found out that Entity Framework is quite fond of generics, so we used our knowledge over generics and patterns to implement a generic repository.
Figure 1.4

By doing so we create a base class with the basic functionality needed in a single class. The methods are virtual, allowing the classes which inherit the generic repository to override them and change the way of working. This was necessary as some classes have different requirements in the different operations, and we managed to have CRUD operations in one single class. We are really proud of that as we avoided a lot of code duplication in the "Database" layer. We provided a simple and elegant way to the server to communicate with the database as well as improved the readability and code quality.
Figure 1.5

3.4 Business Layer

"BusinessLayer" layer is the place where the application logic lies. Our BL knows both about the model classes and database classes. The BL is contacted by the WCF, execute its login and then connects with the database if required.
Figure 1.6
Having a generic repository in the DAL, we realized that we can do the same in the "BusinessLayer" Layer, so we implemented an abstract class with the basic CRUD operations. Then every other class in the BL has to extend the abstract controller class and needed to have a repository class in order to communicate with the database.

Figure 1.7 shows *a class implementing the AController<T>*

3.5 WcfServiceLibraryAuction

The WCF client is self hosted in a console application.It is the only part of the system which can directly reference all of the main components responsible for executing actions. Both of our clients, the web client and the winform client, make requests to the service, which then executes pieces of code, accessing the database, and returns a result back to the user. It is important to note that the user client side does not have **direct** implementation of the core classes and business layer. They do not know how the method AddProduct() works, with all of the controllers and repositories connected to it. They only receive minimum instructions from a proxy class which is generated by the service.
(*Figure 2.7*)

3.5.1. What is WCF and why use it ?

Windows Communication Foundation is a framework for building service-oriented applications, where data can be send asynchronously from one endpoint to another. One endpoint is a client, which makes requests to the service, another end point is the host self service which has access to the database for the user's request.
The WCF allows us to keep the business logic and database access at only one place. This is very useful, when comes to an application to have different clients such : a web client and a mobile application.
WCF also allows the business logic to be kept at one place, so the client application can stay clean from any logic, for a easier scalability rather than duplicating the code on every new system client.

3.5.2 WCF usability

In the start of the project we thought that every domain class and the database and business layer classes should have their own service. We thought that was the right way to do as we have learned about abstraction and code quality a lot. Soon we realized this is not possible and we should put every method that needed to be exposed in one big class due to the requirement of having an interface and a class which implements the needed methods for the WCF. The interface methods had to have assigned a service contract and operation contracts in order to be used by the client side. The class implements the interface then provides the implementation of the operation contracts.

Figure 1.8

3.5.3 Logging

Since we wanted to log all errors which occur both in the server and client side, we implemented a global error handler. This was something we did not learned in school but rather had to make our own research on it. For that we cannot guarantee that the implementation we use, is the right way to do it.

We introduced two new classes in the WCF project: "ErrorHandler" and "GlobalErrorBehaviorAttribute" class.
First class is responsible for logging all the exceptions and to provide fault for it, and the other class is extending the Attribute class and implementing the IServiceBehavior interface. Those classes are then used to change the implementation of our service class so our custom error handler can log all errors.

Figure 1.9

3.5.4 Endpoints

In order to run the other two applications, the server has to be started. To do that a service needs endpoints. All communication with WCF is done through endpoints, providing the client side the service's functionality.They follow the ABC convention:

- **A**ddress that specifies where the endpoint can be found as well as the service identity.

The address is used to locate and identify the endpoint and is specified as a Uniform Resource Identifier (URI). It is expressed as Scheme://<MachineName>[:Port]/Path.
The two addresses defined in this project are :
"net.tcp://localhost:8734/Design_Time_Addresses/WcfServiceLibraryAuction/AuctionProjectService/" and
"http://localhost:8733/Design_Time_Addresses/WcfServiceLibraryAuction/AuctionProjectService/"

- **B**inding shows how a client can communicate with the endpoint. The binding also specifies the transport protocol, message encoding and security requirements.

The WCF service exposes two endpoints: "wsHttpBinding" and "netTcpBinding" which grant the clients connection to the service.

1. wsHttpBinding is used to provide a secure and reliable connection. The transport is HTTP and the encoding is XML. We did not manage to put a security protocol as we simply did not have enough time not only to implement it but also, to make a research on the large amount of protocols available. The lack of security protocol makes the current binding similar to using the "basicHttpBinding" which does not provide any security or encryption of the connection. We define that as a big hole in our project.
2. netTcpBinding point came a little bit late in the project. We decided to introduce it as it provides a faster way of communication with applications in the .Net framework. It uses TCP for transportation and binary encoding making it significant relevant for .Net applications.

Figure 2.5

- **C**ontract outlines what functionality the endpoint exposes to the client.

Being a operation collection, it specifies what functionality the endpoint exposes to the client, or a standard way of describing what server does. In this project it consists of an interface name.

Service contracts are defined by the attributes:

- ServiceContract – is on the interface - it describes the operations a service exposes to the client.
- OperationContract – is on the method - the attribute applied inside a interface, to a method.

Figure 1.8

In case we want explicitly to decide which members to expose it is used DataContract to annotate the class and DataMember to annotate the properties otherwise without this contracts it takes all properties from System.Runtime.Serialization namespace.

(Shown in *Figure(2.6)*.

3.6 UnitTestProject1

The project was used to store all the tests run for the entire project. We started with test first for the DAL and the basic methods inside. We have test methods for almost all methods that are currently used in the application. Most of the methods use only the DAL classes as the controllers for them in the BL are not making any changes, just calling the DAL directly.

4. AuctionWebApplication(solution)

The new solution for the web application was necessary as it made things easier to navigate in and clearly separates as logic from the previous one.

4.1 What is MVC

ASP.NET MVC is a framework that utilizes the model-view-controller pattern for creating scalable and extensible Web applications. Each of these components (Model, View, Controller) are built to handle specific development aspects of an application.

The MVC framework is defined in the System.Web.Mvc assembly, and we are using the pattern for creating the "AuctionWebApplication" solution.

4.2 MVC design pattern

The main principle of this pattern is that it separates the application into three layers.

- · **Models**. The Model component corresponds to all the data related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic related data.

  In our case model folder is empty since we are using WCF Service proxy classes. Figure 3.1
- **Views**. The View component is used for all the UI logic of the application. For example, the Customer view would include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.
- **Controllers**. Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.

Figure 2.0

4.3 Use of MVC

We used MVC pattern to create "WebAucitonApplication" solution, due to its features:

The MVC pattern helps you create applications that separate the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements. The pattern specifies where each kind of logic should be located in the application. The UI logic belongs in the view. Input logic belongs in the controller. Business logic belongs in the model (in our case to the WCF Service ***Figure 3.2***). This separation helps in managing complexity when building an application, because it enables focus on one aspect of the implementation at a time. For example, you can focus on the view without depending on the business logic.

The loose coupling between the three main components of an MVC application also promotes parallel development. Thanks to that, it was possible one developer to work on the view, another one on the controller logic, and another one to focus on the business logic in the model.

5. "AuctionProject" (Winform) solution

This solution was built as a management application for the customer who owns the website. It consists of no business logic. All the operations made are through the WCF service netTcpBinding. It is mainly UI logic of how the things should be presented and evaluated. The application provides different ways of working with the database data, providing the management of supervising and manipulating data.

6. Security

As said earlier we did not manage to fulfill the security desires for the system. We had neither the time nor the knowledge of how to do it. Although we tried with the wsHTTPbinding, we did not succeed in a timely fashion and didn't have time to try different ways.

What we did to ensure security, is hashing and salting of the passwords. We put our hashing algorithms in the client applications as we didn't want to transfer clear text passwords over the network. We know that this is not best decision as for every different client we also have to put the algorithms there. This is monotonous job and adds a lot of code duplicating, which also lowers our code quality. If we had put the hashing algorithms in the WCF this would mean that we are going to need the algorithms only in one place and it is easier to change in the future, but also that clear text passwords would be transferred over the wire.
We made sure a highly proven algorithms and libraries were used when generating the salt and hash.
One of the best ways regarding the password security, which is actually highly used in real life scenarios, is represented by using an encrypted channel to transfer the password from the client to the server. To do that a security technology, also known as SSL(Secure Socket Layer) is required, for establishing an encrypted link between a web server and browser, keeping all data private and integral.
To be able to create an SSL connection, a web server requires an SSL Certificate, which includes two cryptographic keys, a Private Key and a Public Key, to ensure the encryption and decryption process.

7. ACID principles

Whenever we deal with relational databases, especially ones that utilize processes that share resources, the use of transactions is almost mandatory. Transactions in databases need to follow the ACID principle, an acronym for Atomicity Consistency Isolation and Durability. In order to manipulate data, transactions mainly utilize the use of two actions: Commit and Rollback.

- Atomicity refers to the transaction being treated as one inseparable and undividable process. The transaction itself contains many other small processes in itself, but either they are all completed successfully, or nothing is committed and all of the information gets reverted back to its previous valid state.

- Consistency refers to being able to populate the rows in a table with valid data time after time without fail. This means that before that get inserted it should pass all validations. If one of those checks fail the whole transaction should fail.

- Isolation - Each transaction in progress should stay isolated from other transactions until is committed.

- Durability means that if for whatever reason (hardware malfunction, server downtime etc) the transaction is interrupted, no data should be lost or corrupted and everything should be returned to the state it was prior to the beginning of the transaction. Here is a snippet of code from our "AuctionRepository" class which folloes this ACID principle.

Figure 2.8
The transaction is rather simple. We need to insert two entities in the database and attaching a third one. We need a transaction as we don't want a only one of the entities to get into the database and resulting in a inconsistent data.

Rollback is not needed as Entity framework handles the case itself in the commit method. The isolation level is Read committed as the transaction does not need to read any data before being inserted.

The example in *Figure 2.9* is more complicated and longer when we want to insert a bid.

Serializable is used as transaction level as we try to prevent any dirty reads. It also means that the data used is going to be locked until the transaction is finished. This prevents that the latest bid is not going to change in the meantime while the operation is doing something different.

Before we insert a bid, we have to assure that a couple of conditions have been met. First we have to check if there are any bids for the auction, and if yes to get the latest one thus locking it so no other transaction can affect it. If there are no bids, we have to check if the bid price is valid and prepare for inserting. If there is a bit, we have to check their prices, and if the new bid price is higher we have to assign the variables and we can insert the bid.

8.Concurrency and Parallelism

Both terms appear similar and are often confused. Both are associated with taking advantage of multicore machines and making processes work faster. But in reality they are different.

Concurrency is the composition of independently executing functions.Concurrent processes can run at the same time but that is not necessary. Most likely they will run in overlapping time periods as every thread needs a CPU time. Concurrency has its threads in order to execute a function faster but that leads to the need of synchronization between them in order to avoid bottleneck. Threads do not have to wait for each other to finish in order to advance in a computation.

Parallelism is the simultaneous execution of functions. It is doing a lot of things at the same time independent of each other. Executing stuff in parallel is only possible on multicore setup.

Concurrency is important part of every modern software product. People take concurrency for granted. User expects that a single application will be able to perform a lot of different operations simultaneously with minimal or no delay. In reality handling concurrency and satisfying user's performance expectations can be a challenge.

We did not get to make any threads the old way as we don't find the need for that flexibility necessary. Instead we used the newest build in libraries. We used **Parallel** library as it provides easy and fast way of implementing parallel functions. *Figure 3.0*

We also took advantage of the **AsParallel()** method when doing heavy work. This was the case when looking for a product in our system as we wanted to check for a products as fast as possible and give them back to the user. *Figure 3.1* On the figure it can also be seen the use of the **Func** class. It is a delegate which accepts a value of one type and return a value of another.

Appendices

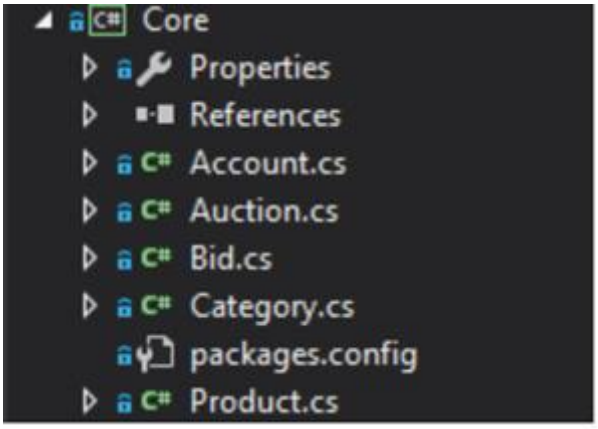


Figure 1.0

```
[StringLength(450)]
 [Index(IsUnique = true)]
public string Email { get; set; }
public string Fname { get; set; }
public string Lname { get; set; }
```

Figure 1.1

```
public class Dbcontext : DbContext
{
    public Dbcontext()
        : base("name=DbContext")
    {
        Database.SetInitializer(new
            MigrateDatabaseToLatestVersion<Dbcontext, Configuration>());
    }

    public virtual DbSet<Account> Accounts { get; set; }
    public virtual DbSet<Auction> Auctions { get; set; }
    public virtual DbSet<Bid> Bids { get; set; }
    public virtual DbSet<Product> Products { get; set; }
    public virtual DbSet<Category> Categories { get; set; }


}
```

figure 1.2

```
internal sealed class Configuration : DbMigrationsConfiguration<Dbcontext>
{
    public Configuration()
    {
        AutomaticMigrationsEnabled = true;
        AutomaticMigrationDataLossAllowed = true;
        ContextKey = "Database.DBcontext";
    }
```

figure 1.3

```
9 references | Delyan Antonov Iliev, 5 days ago | 2 authors, 12 changes
public abstract class ARepository<T> where T : class
{
    18 references | Delyan Antonov Iliev, 48 days ago | 1 author, 1 change
    public Dbcontext context { get; set; }
    public DbSet<T> Set;

    5 references | Delyan Antonov Iliev, 48 days ago | 1 author, 1 change
    public ARepository(Dbcontext Context)
    {
        this.context = Context;
        Set = context.Set<T>();
    }

    5 references | Delyan Antonov Iliev, 48 days ago | 1 author, 1 change
    public ARepository()
    {
        context = new Dbcontext();
        Set = context.Set<T>();
    }
```

figure 1.4

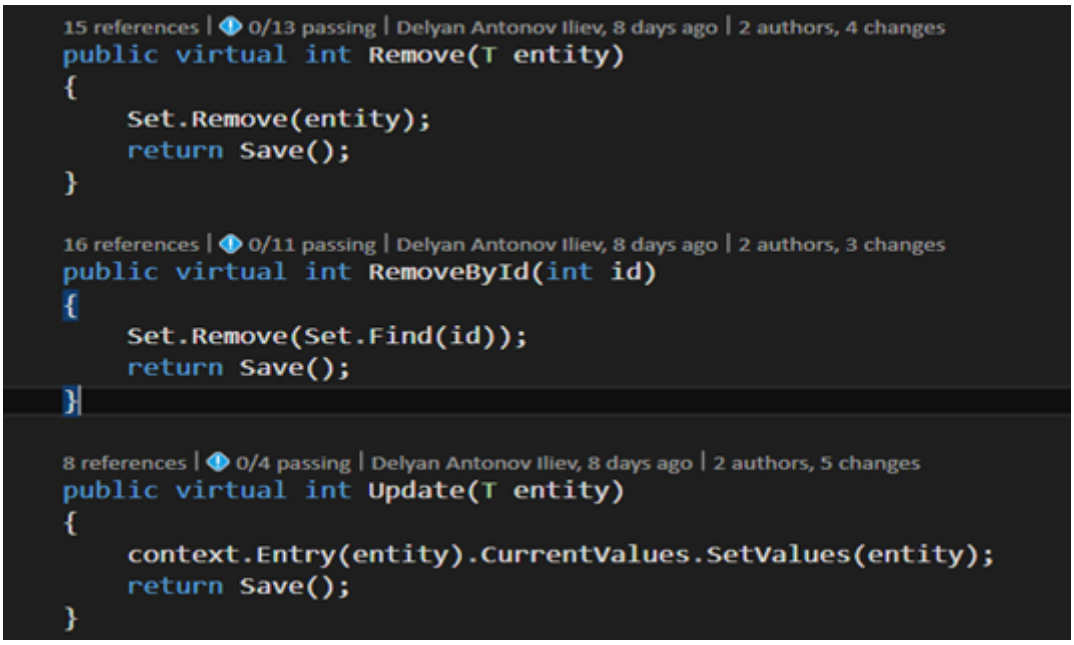


figure 1.5

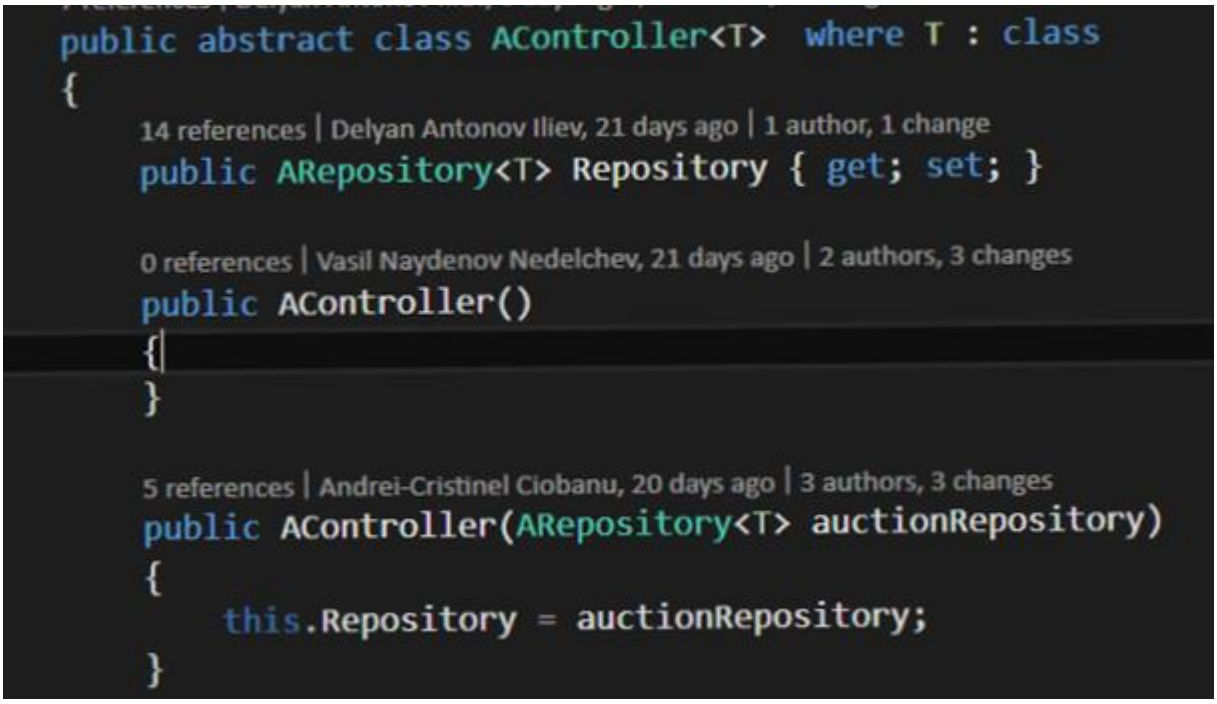


figure 1.6

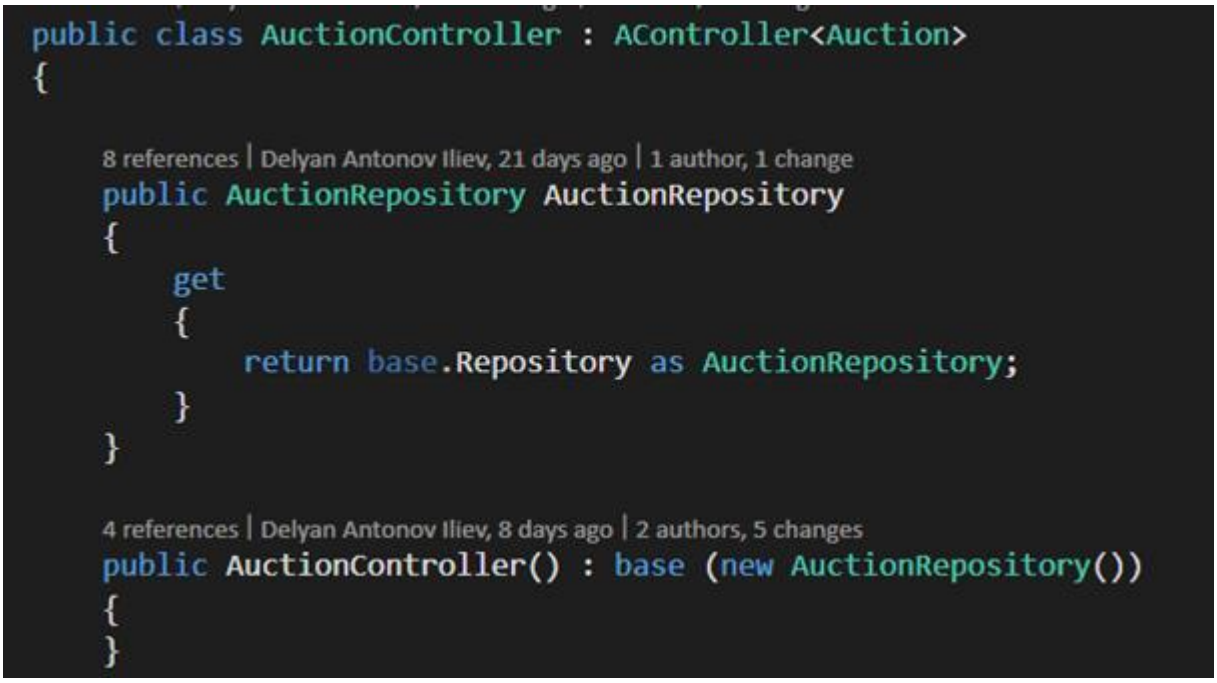


figure 1.7

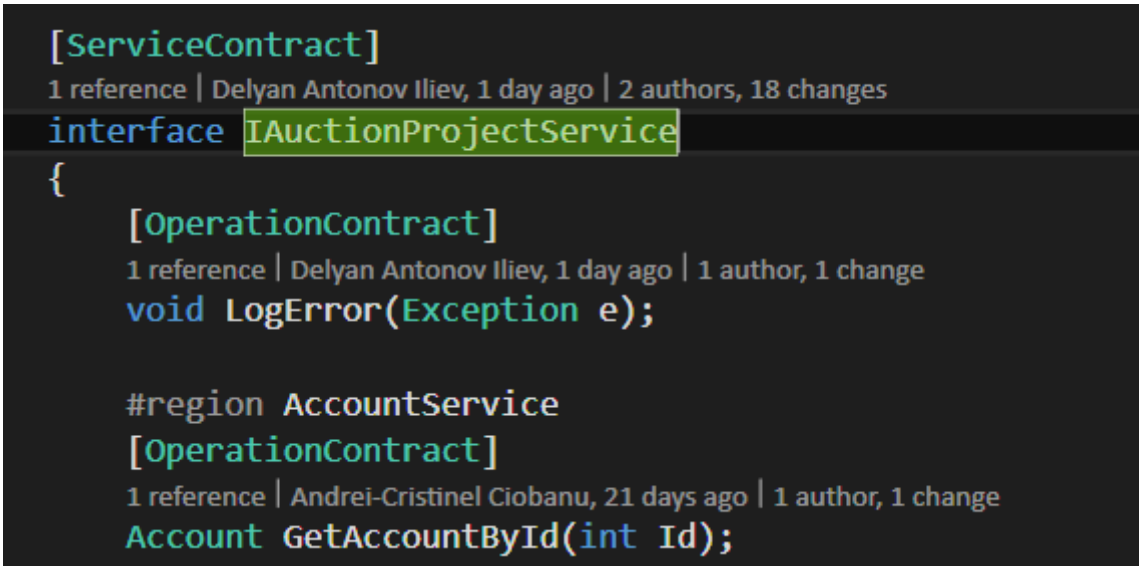


figure 1.8

Figure 1.9

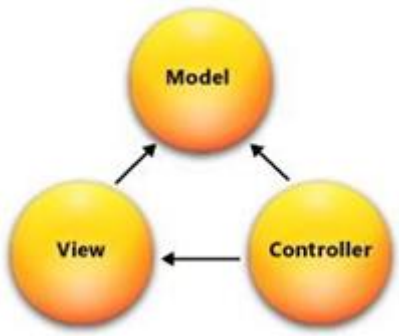


Figure 2.0

```
2 references | Delyan Antonov Iliev, 9 days ago | 1 author, 1 change | 0 exceptions
private Account HashAndSaltPassword(Account acc)
{
    // creates salt with random vales
    byte[] saltBytes = new byte[32];
    using (var provider = new RNGCryptoServiceProvider())
        provider.GetNonZeroBytes(saltBytes);
    acc.Salt = Convert.ToBase64String(saltBytes);
    // Create the Rfc2898DeriveBytes and get the hash value
    acc.Password = ComputeHash(acc.Salt, acc.Password);
    return acc;
}
2 references | Delyan Antonov Iliev, 9 days ago | 1 author, 1 change | 0 exceptions
private static string ComputeHash(string salt, string password)
{
    var saltBytes = Convert.FromBase64String(salt);
    using (var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, saltBytes, 1000))
        return Convert.ToBase64String(rfc2898DeriveBytes.GetBytes(256));
}
```

Figure 2.1

```
internal sealed class Configuration : DbMigrationsConfiguration<Dbcontext>
{
    public Configuration()
    {
        AutomaticMigrationsEnabled = true;
        AutomaticMigrationDataLossAllowed = true;
        ContextKey = "Database.DBcontext";
    }
```

Figure 2.2

```
public class Dbcontext : DbContext
{
    public Dbcontext()
        : base("name=DbContext")
    {
        Database.SetInitializer(new
            MigrateDatabaseToLatestVersion<Dbcontext, Configuration>());
    }

    public virtual DbSet<Account> Accounts { get; set; }
```

Figure 2.3

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Entity<Auction>()
    .HasRequired(p => p.Product)
    .WithMany()
    .WillCascadeOnDelete(true);
}
```

Figure 2.4

```
<service name="WcfServiceLibraryAuction.AuctionProjectService">
  <clear />
  <endpoint address="BidService" binding="wsHttpBinding" name="secure"
    contract="WcfServiceLibraryAuction.IAuctionProjectService" />

  <endpoint address="net.tcp://localhost:8734/Design_Time_Addresses/WcfServiceLibraryAuction/AuctionProjectService/"
    binding="netTcpBinding" bindingConfiguration="" name="binary"
    contract="WcfServiceLibraryAuction.IAuctionProjectService" />

  <host>
    <baseAddresses>
      <add baseAddress="net.tcp://localhost:8734/Design_Time_Addresses/WcfServiceLibraryAuction/AuctionProjectService/" />
      <add baseAddress="http://localhost:8733/Design_Time_Addresses/WcfServiceLibraryAuction/AuctionProjectService/" />
    </baseAddresses>
```

Figure 2.5

```
[DataContract(IsReference = true)]
public class Bid
{
    public Bid()
    {
    }
    public Bid(double price,DateTime time,Account bidOwner,int auctionId)
    {
        Price = price;
        BidTime = time;
        BidOwnerId = bidOwner.Id;
        BidOwner = bidOwner;
        AuctionId = auctionId;
    }
    [DataMember]
    public int Id { get; set; }
    [DataMember]
    public double Price { get; set; }
    [DataMember]
    public DateTime BidTime { get; set; }
    [DataMember]
    public int BidOwnerId { get; set; }
    [DataMember]
    public Account BidOwner { get; set; }
    [DataMember]
    public int AuctionId { get; set; }
```

Figure 2.6

```
class Program
{
    static void Main(string[] args)
    {
        Uri baseAddress = new Uri("http://localhost:8733");
        using (ServiceHost host = new ServiceHost(typeof(AuctionProjectService), baseAddress))
        {
            // Enable metadata publishing.
            ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
            smb.HttpGetEnabled = true;
            smb.MetadataExporter.PolicyVersion = PolicyVersion.Policy15;
            host.Description.Behaviors.Add(smb);

            // Open the ServiceHost to start listening for messages. Since
            // no endpoints are explicitly configured, the runtime will create
            // one endpoint per base address for each service contract implemented
            // by the service.
            host.Open();

            Console.WriteLine("The service is ready at {0}", baseAddress);
            Console.WriteLine("Press <Enter> to stop the service.");
            Console.ReadLine();

            // Close the ServiceHost.
            host.Close();
        }
    }
```

Figure 2.7

```
13 references | 0/5 passing | Delyan Antonov Iliev, 2 days ago | 2 authors, 7 changes
public override int Add(Auction auc)
{
    int i = 0;
    using (var dbContextTransaction = context.Database.BeginTransaction(System.Data.IsolationLevel.ReadCommitted))
    {
        context.Products.Add(auc.Product);
        context.Accounts.Attach(auc.Seller);
        Set.Add(auc);
        i = context.SaveChanges();
        dbContextTransaction.Commit();
    }
    return i;
}
```

Figure 2.8

```
using (var dbContextTransaction = context.Database.BeginTransaction(System.Data.IsolationLevel.Serializable))
{
    int auctionId = bid.AuctionId;
    var latest = GetLatestBidForAucion(auctionId);
    bool condition;
    Auction auction;
    if (latest != null)
    {
        condition = CheckValidBidPrice(latest, bid);
        auction = latest.Auction;
    }
    else
    {
        AuctionRepository repo = new AuctionRepository(context);
        auction = repo.getByIdWithObjects(auctionId);
        condition = CheckValidBidPrice(bid, auction);
    }
    if (!condition)
        return 0;
    else
        AssignVariables(bid, auction);

    context.Auctions.Attach(bid.Auction);
    context.Entry(bid.Auction).State = EntityState.Modified;
    Set.Add(bid);
    i = context.SaveChanges();
    dbContextTransaction.Commit();
```

Figure 2.9

```
1 reference | Delyan Antonov Iliev, 3 days ago | 1 author, 2 changes
public IEnumerable<Auction> GetAllAuctionsForBids(IEnumerable<Bid> bids)
{
    HashSet<Auction> set = new HashSet<Auction>();
    object obj = new object();
    Parallel.ForEach(bids, (item) =>
    {
        lock (obj)
        {
            set.Add(item.Auction);
        }
    });
    return set;
}
```

Figure3.0

```
2 references | Delyan Antonov Iliev, 5 days ago | 1 author, 4 changes
public ParallelQuery<Product> GetAllProductsWithName(string name)
{
    if(name.Contains(' '))
    {
        return null;

    }
    var NameEquals = Set.AsParallel().Where(prd => prd.Name.Equals(name, StringComparison.OrdinalIgnoreCase));
    var NameContains = Set.AsParallel().Where(NameFunc(name));
    return NameEquals.Union(NameContains).Union(GetProductsWithDescription(name));
}

2 references | Delyan Antonov Iliev, 5 days ago | 1 author, 2 changes
public Func<Product, bool> DescriptionFunc(string description)
{
    return pr => pr.Description.IndexOf(description, StringComparison.OrdinalIgnoreCase) >= 0;
}

2 references | 1 author, 2 changes
public Func<Product, bool> NameFunc(string name)
{
    return pr => pr.Name.IndexOf(name, StringComparison.OrdinalIgnoreCase) >= 0;
}
```

Figure 3.1

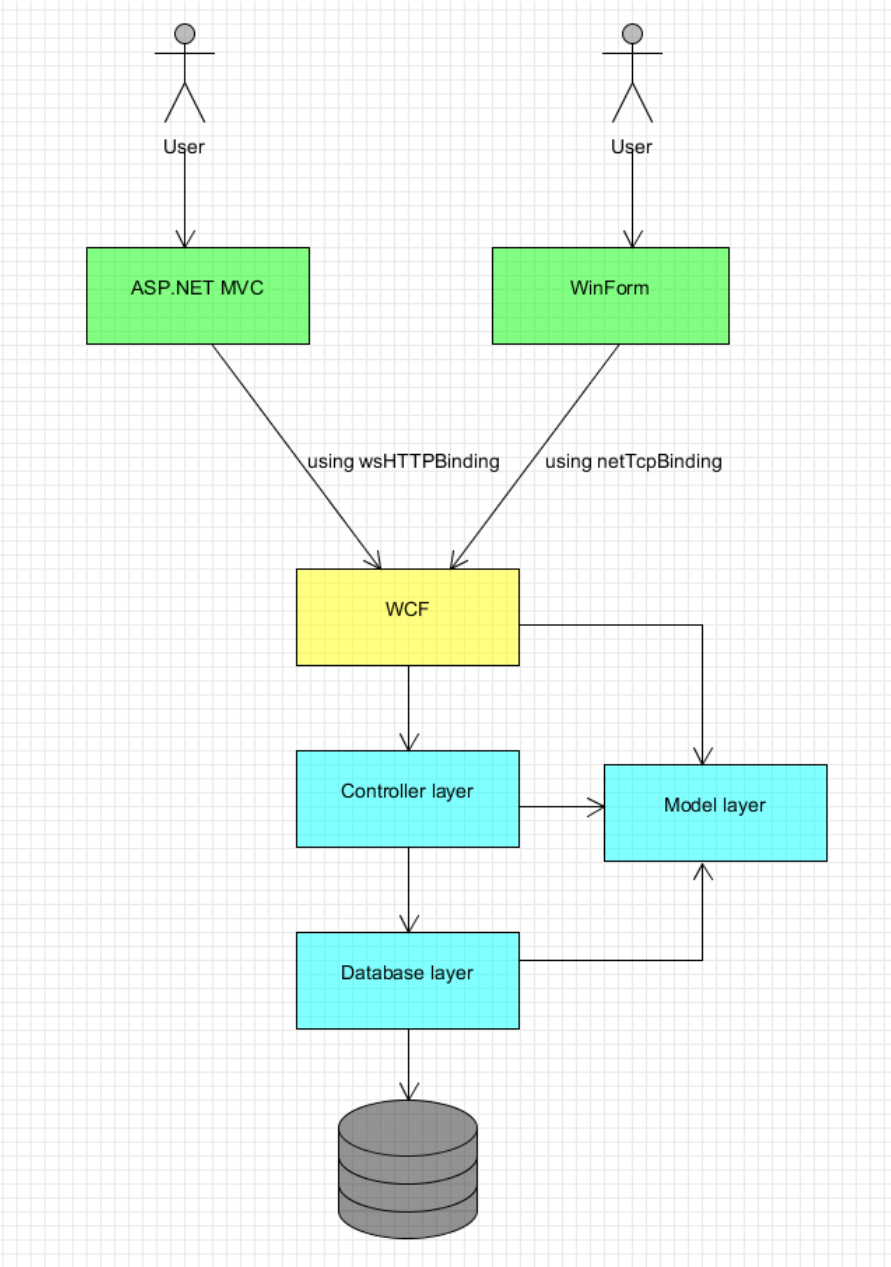


figure 3.0

```
public ActionResult Details(int id)
{
    AuctionWebApplication.AuctionService.Account account = AccService.GetAccountById(id);
    if (account == null)
    {
```

figure 3.1

```
if (ModelState.IsValid)
{
    account = AccService.GetAccountByEmail(account.Email);
    return RedirectToAction("Edit", "Auction");
}
```

figure 3.2

Bibliography

Books:

Pro C# and .NET 4.5 framework sixth Edition – Andrew Troelsen

Online Sources:

Distributed Systems – Prof. Wolfgang Emmerich – UCL Computer Science
http://www0.cs.ucl.ac.uk/staff/ucacwxe/lectures/ds98-99/dsee3.pdf
https://msdn.microsoft.com/en-us/library/ms731299(v=vs.110).aspx
https://msdn.microsoft.com/en-us/library/ms733107.aspx
https://msdn.microsoft.com
https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx
http://www.entityframeworktutorial.net/what-is-entityframework.aspx
http://www.hpcs.cs.tsukuba.ac.jp/~tatebe/lecture/h23/dsys/dsd-tutorial.html
http://info.ssl.com/article.aspx?id=10241