UCN dmai0916

3rd semester project

Group 1

PROGRAMING & TECHNOLOGY REPORT

ConQuestion Game

18th December 2017

Mirjana Erceg
Tamas Kalapacs
Zahro-Madalina Khaji
Sangey Lama
Andreas Richardsen

University College of Northern Denmark

Technology and Business

Computer Science AP Degree Programme

Programming & Technology report

dmai0916

ConQuestion Game

Project participants:

Mirjana Erceg
Tamas Kalapacs
Zahro-Madalina Khaji
Sangey Lama
Andreas Richardsen

Supervisor:

Ronni Hansen

Submission Date:

18th December 2017

Contents

Introduction

In this report we will discuss developing software using network technologies and distributed systems while servicing multiple clients (both dedicated and web based). We will discuss theoretical concepts behind these technologies, the options available to us and the reasoning for our choices for this project.

For our project we decided to make a quiz game to allow students / friends to take part online through a web or dedicated client to play or study together. Quizzes are competitive by nature and our game is no different, therefore certain problems must be addressed to make our game fair and enjoyable for everyone e.g.

- How to handle multiple clients playing in one game?
- How to handle communication between the server and client?
- How to handle security to prevent cheating?
- How to determine the winner of the game?

As you read this report you will learn how developing distributed systems allow us to make our software available to the public through web services and application programming interfaces. This allows us to expose our systems functionality for anyone to use while keeping the implementation of our system private.

Architecture deliberations

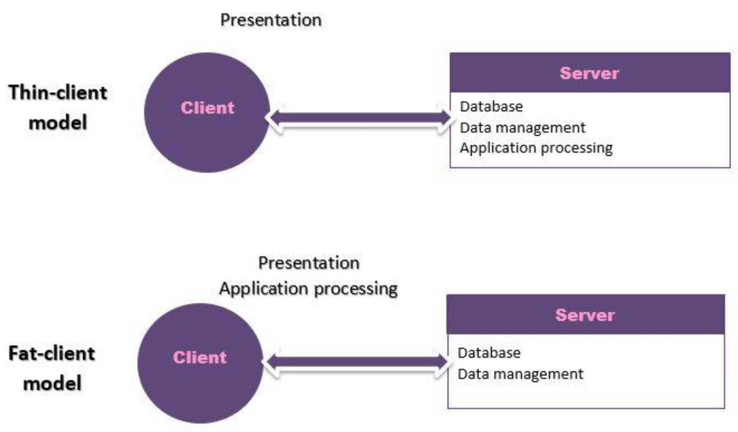
When designing distributed systems selecting an appropriate architecture is important. The choice of architecture will determine how easy it is to maintain and scale your system in the future as well as isolating sensitive information.

We look at the following types of architecture and evaluate them accordingly;

1) Two-tier architecture
   a) Thin client
   b) Fat client
2) N-tier / Multi-tier
   a) Classic client/server
   b) Classic web
   c) Service orientated
3) Peer to peer

Two-tier architecture

Two-tiered architecture consists of a client and a server. Depending on the implementation you could have thin or fat client styled architecture. [1]



Thin client

In the thin client style, the client contains only the presentation layer (user interface) while the server handles the business logic, data management and database. The major advantage of this system is that clients are easier to manage. The main drawback is that this places a heavy processing load on the server which can negatively affect performance. [1]

Fat client

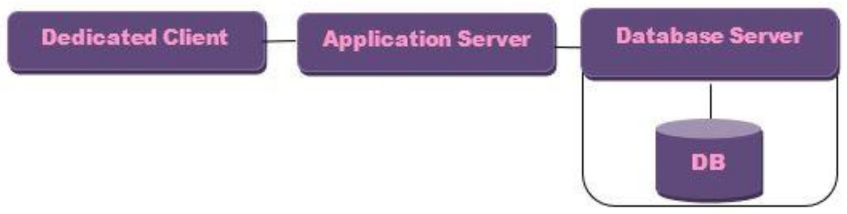
In fat client style, the client contains the presentation layer and the business logic and the server handles data management and database. This evenly distributes more of the processing load to the client's computer allowing the server to handle the database transactions. However, this leads to an update problem when functionality changes in the system in that every single client needs to be updated. [1]

Regardless if you choose the fat or thin client styles you are still dividing 4 layers (presentation, application processing, data management and database) between 2 machines. This can lead to problems with scalability and maintainability and is generally used by legacy systems or by systems that require little data management or application processing.

N-tier / Multi-tier architecture

N-tier or Multi-tier architecture addresses some of the problems of two-tiered architecture in that each logical layer runs on a separate computer/server. This means that scalability is easier to handle (add more servers as customer base increases) and that the distributed system is easier to maintain. Below are some examples:

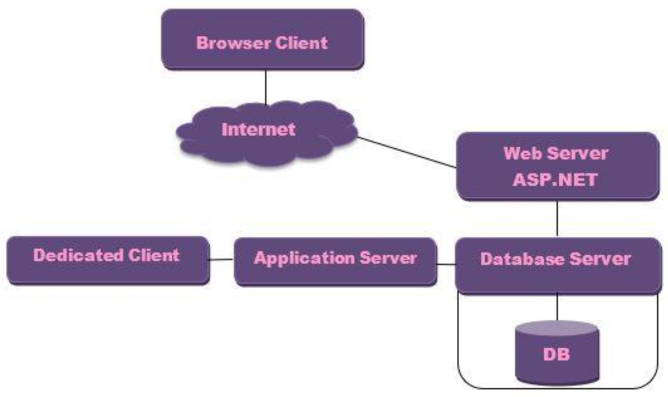
Classic client / server architecture



In this example we have the dedicated client, the application server and the database. The dedicated client contains the user interface and calls on the application server. The application server contains the business logic for our system and makes calls to the database. The database persists all the necessary data for our system.

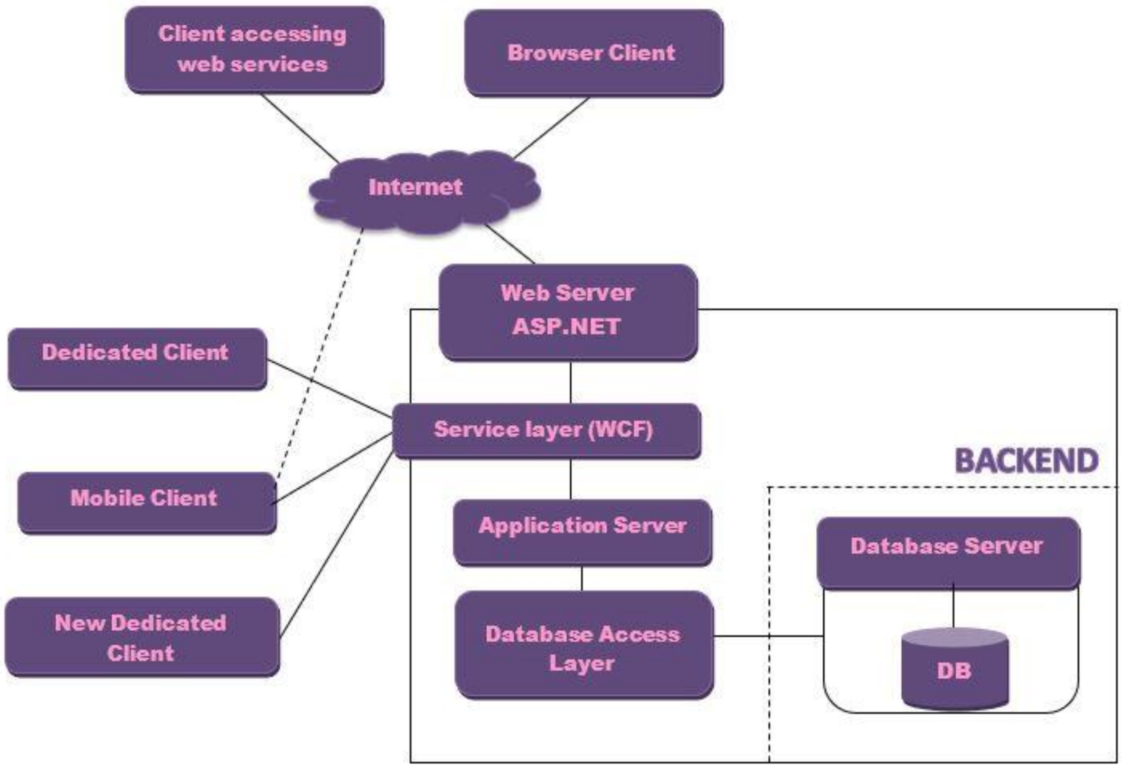
However, as we want to support web clients as well, this example is not suitable for our system.

Classic web architecture



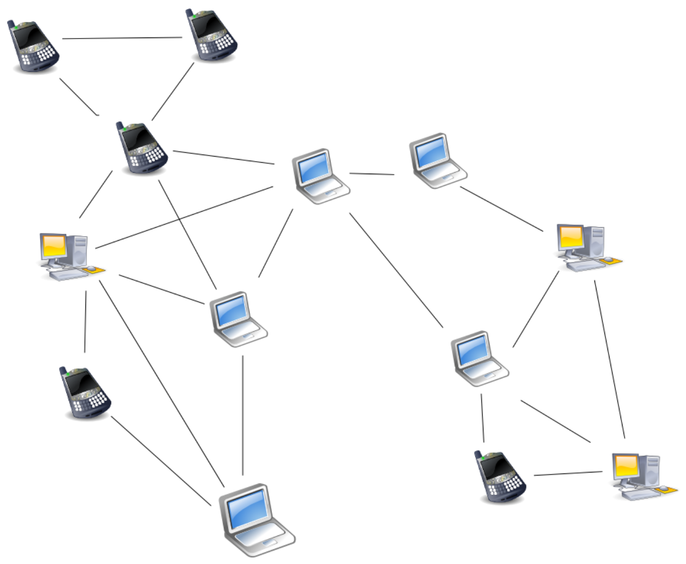
In this example we still have the dedicated client, application server, database but now also a webserver and browser clients. This solves our problem of supporting both a web client and a dedicated client however, in this example there is duplication of the code on the web server and application server which is bad for maintainability (need to change the code twice).

Service orientated architecture



In this style of architecture, we add an extra service layer above the application server which in turn services various clients including the web server. The service layer acts as an abstraction of the application server only providing enough information to utilise the business logic without exposing its implementation. This removes the previous problem of code duplication as here the web server is a client of the service, therefore both the dedicated and web client make use of the same logic. There are more benefits of service oriented architecture which we will discuss later in detail, e.g. reusability, scalability, platform independence, etc. [2]

Peer-to-peer architecture



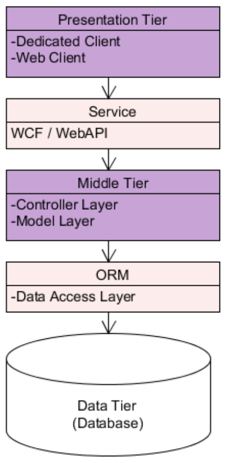
In peer to peer architecture, peers are machines interconnected to each other without need of a centralised server. This means each machine on a peer-to-peer network act both as a client and server. Peers share a portion of their computer's resources, their bandwidth, storage space or processing power. Commonly peer-to-peer networks are used for file sharing between files however other uses include multimedia streaming (Spotify) and even digital cryptocurrencies (Bitcoin). An advantage to a peer-to-peer network is robustness, meaning the network is relatively undamaged by any single peer failure. This contrasts with client/server architectures as this removes the single point of failure commonly found at the server. A drawback to this architecture is that content is completely user driven, if users do not want to share specific files they can choose to no longer share it. This means that unpopular files can be extremely hard to find on a peer-to-peer network. [1]

Our choice of architecture

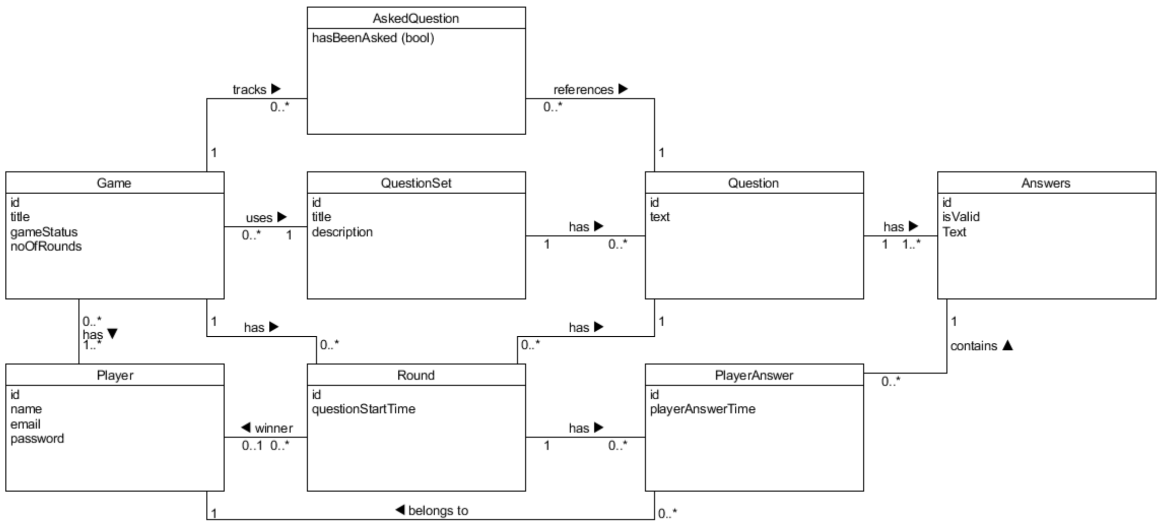
For this project we have elected to use the n-tier service orientated architecture. We have selected n-tier for our project as we need to create both a dedicated client and web based client, so we can take advantage of the reusability of code and its compatibility with multiple platforms. Other benefits such as the ease of scalability and maintainability are useful if we decide to expand the project in the future. We also will adopt a thin client style for our project due to reasons for security and cheating which could arise in multiplayer games. This means our application server will act as an authoritative server receiving game actions from players and will determine their validity.

Application architecture

For our application as we are using N-tier architecture we are dividing our project into 3 tiers, Presentation, Middle and Data. Presentation will contain our user interface for the dedicated client and web client. Middle Tier contains our business logic layers for our game logic. Data tier contains our database where we persist the data for our game state. In between these tiers we are using middlewares to assist in communication between the tiers. A service middleware between the presentation and middle tier. An ORM (Object-relational mapping tool) to handle database transactions for our system.



Domain Model



We have included the domain model we used to help us design our quiz game. We have decided to name our application Conquestion as originally, we intended the game to have a quiz round and map conquering mechanic in gameplay, but due to time constraints we scaled the project back to a simplified quiz game.

The domain model shows the domains we feel are necessary to play a quiz game namely, a Game, a QuestionSet with Questions and Answers, Players, and Rounds with PlayerAnswers. The associations show the relations between these classes and their multiplicities.

Communication and Middleware

As we are using n-tier architecture and our 3 tiers theoretically could be run on separate machines we need the various components to communicate with each other over a network.

As mentioned in our application architecture section we will be making using of middleware to assist in communication between our system components. Communication between the presentation tier and middle tier is handled by a web service middleware (SOAP or REST based). Communication between the middle tier and the database is handled by data access middleware (Entity framework or ADO.NET).

Web Services

Web services are a method of offering interoperability between systems allowing for easier communication between different platforms. [3]

Typically for web services there are two choices, SOAP (Simple Object Access Protocol) which involves the passing of envelopes through the internet or intranet or REST (REpresentation State Transfer) which involves using HTTP commands to communicate.

SOAP

SOAP is a messaging protocol typically used over HTTP to communicate with distributed systems over the web. A SOAP message is an XML document made up of an envelope with an optional header and a body containing the message being passed. The correct format of this message is described by a WSDL (Web Services Description Language).

A WSDL will contain information about the location the service is being hosted, what methods can be called and how to call them appropriately. [4]

REST

REST is an alternative to SOAP rather than using a strict protocol standard it is an architectural style which follows a set of principles. [4]

- Resources have unique addresses in the form of URIs
- These resources can be accessed and manipulated using the HTTP commands (GET, POST, PUT, DELETE).
- The protocol is stateless meaning the server doesn't store the client's context and allows for multi-layered intermediaries and caching.

SOAP vs REST

When considering which type of service to use we considered the advantages and disadvantages of both. [5]

Arguments for SOAP

- SOAP supports multiple types of Transport (REST is HTTP only)
- SOAP has HTTP standard protocols allowing for easier communication through firewalls.
- Automated code generation support in certain languages.

Arguments for REST

- REST is faster when communicating as less overhead compared to SOAP envelopes.
- REST can use more lightweight data formats e.g. JSON
- Similar design philosophy to web technologies.

Our choice

We decided to use SOAP for our project because our team was more familiar with SOAP than REST, the ability for .NET to auto generate a lot of code for SOAP making it easier to develop with. While there are some advantages for using REST in our project given the time constraints we felt more comfortable working with SOAP.

WCF

WCF (Windows Communication Foundation) is a Microsoft framework to assist in deployment of services between clients and servers. Clients can connect to services via endpoints which are made up of an address, a binding and a contract. The address specifies network address where the client can access the service, the binding specifies the transport protocol to be used and the contract is the interface the service implements. [6]

Bindings

The binding gives us more flexibility to configure the transport protocol. Our options in terms of HTTP-based bindings were: basicHttpBinding, wsHttpBinding and wsDualHttpBinding. [7]

basicHttpBinding exists mostly for backwards compatibility to support clients pre- .NET 3.0.

wsHttpBinding implements WS* specifications to support enterprise requirements for transaction management, security and reliability.

wsDualHttpBinding is "similar to wsHttpBinding but intended for use with duplex contracts (e.g., the service and client can send messages back and forth). This binding supports only SOAP security and requires reliable messaging."

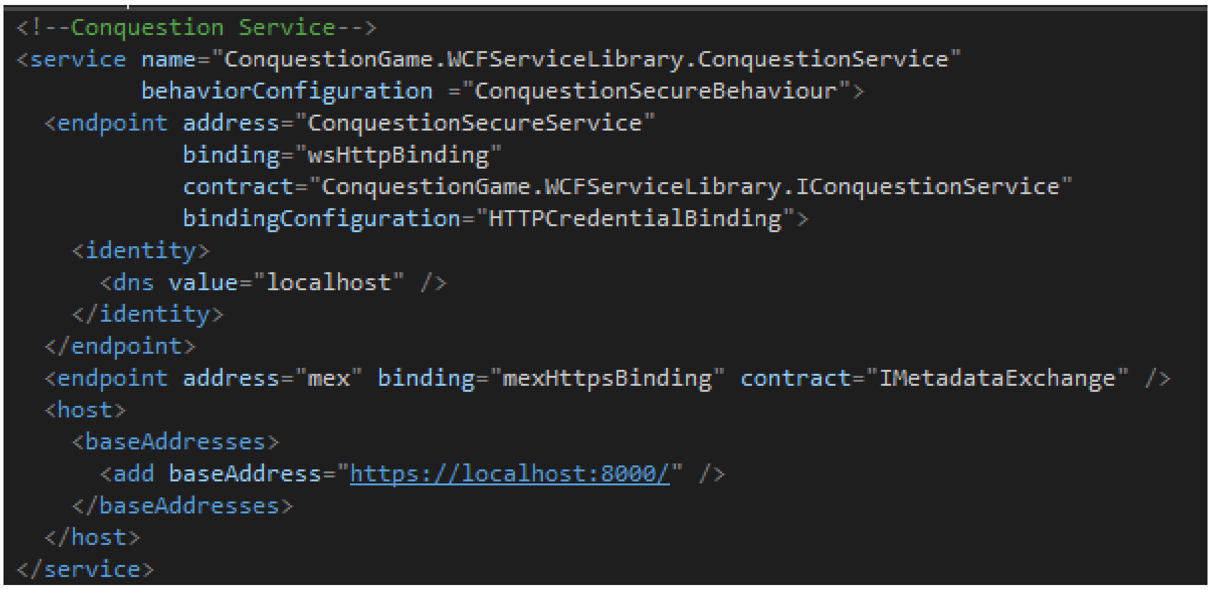
Other binding options available were: netTcpBinding, netNamedPipeBinding, netPeerTcpBinding

netTcpBinding is a binary encoded optimised binding between WCF clients and WCF services that offers the fastest binding of all the choices. However, it as a result it does not offer interoperability.

netNamedPipeBinding is generally used for secure and reliable named pipe based communication on process that run on the same machine.

netPeerTcpBinding is used for peer-to-peer computing.

Code Example

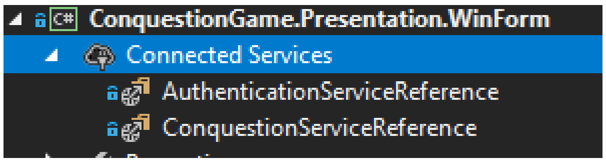


```xml
<!--Conquestion Service-->
<service name="ConquestionGame.WCFServiceLibrary.ConquestionService"
         behaviorConfiguration ="ConquestionSecureBehaviour">
  <endpoint address="ConquestionSecureService"
            binding="wsHttpBinding"
            contract="ConquestionGame.WCFServiceLibrary.IConquestionService"
            bindingConfiguration="HTTPCredentialBinding">
    <identity>
      <dns value="localhost" />
    </identity>
  </endpoint>
  <endpoint address="mex" binding="mexHttpsBinding" contract="IMetadataExchange" />
  <host>
    <baseAddresses>
      <add baseAddress="https://localhost:8000/" />
    </baseAddresses>
  </host>
</service>
```

Above is a code snippet we used from our project to configure our endpoint in WCF. The base address is located at https://localhost:8000/ showing where the service can be discovered. Here we are using a wsHttpBinding for our service with a customised binding configuration to take credentials. Finally the contract specifies where the interface for the service is located.

Proxy Class

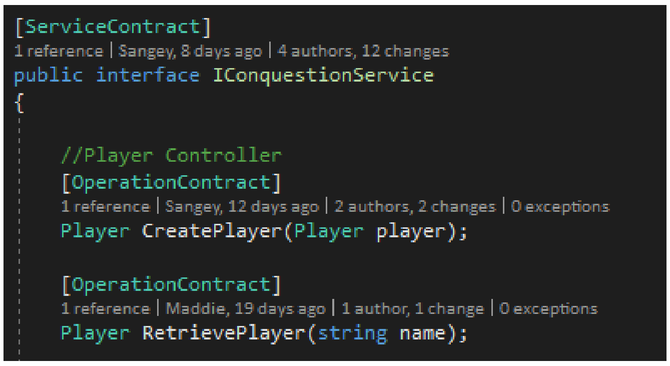
WCF allows for proxy classes to allow for the client applications to communicate with the service while at the same time encapsulating some of the service information. WCF offers the tool *Add Service Reference* which allows for auto generation of a service reference proxy class.


ConquestionGame.Presentation.WinForm
Connected Services
AuthenticationServiceReference
ConquestionServiceReference

The above image shows two reference proxy classes added to our dedicated client in our project.

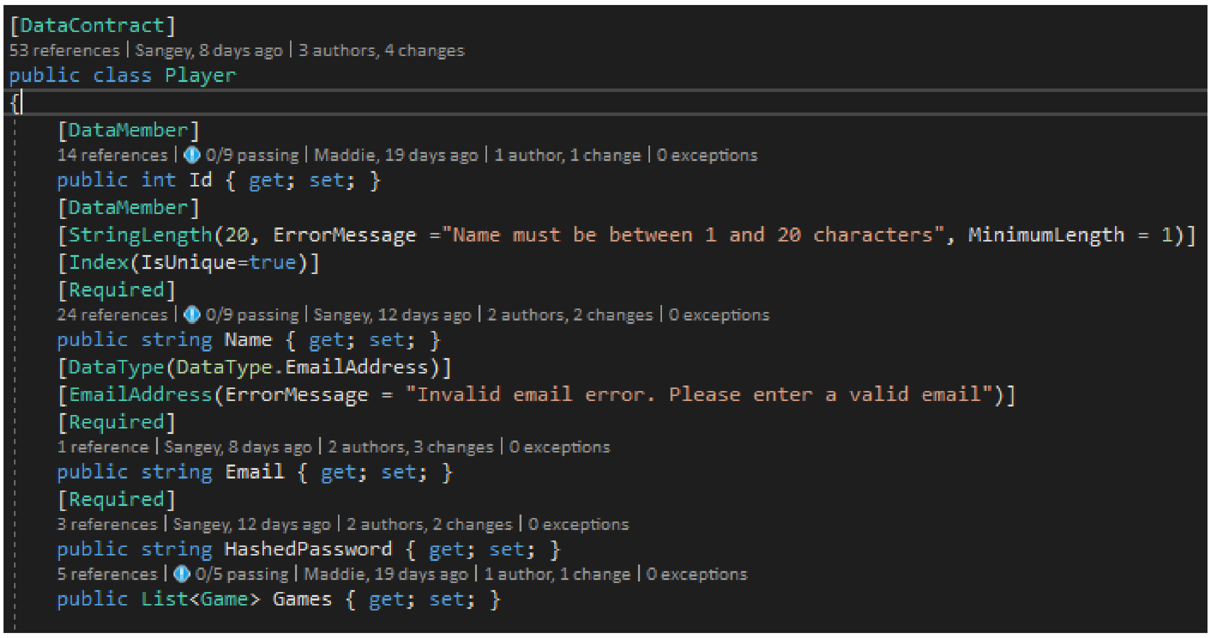
Service contracts and serialisation

When making calls to service methods, objects need to be serialised into a byte stream to then be communicated over the net and then rebuilt once received. WCF achieves this through use of service operation contracts, data contracts and the component DataContractSerializer. [8]



```
[ServiceContract]
1 reference | Sangey, 8 days ago | 4 authors, 12 changes
public interface IConquestionService
{

    //Player Controller
    [OperationContract]
    1 reference | Sangey, 12 days ago | 2 authors, 2 changes | 0 exceptions
    Player CreatePlayer(Player player);

    [OperationContract]
    1 reference | Maddie, 19 days ago | 1 author, 1 change | 0 exceptions
    Player RetrievePlayer(string name);
```

Above we have one of our service contracts, the RetrievePlayer operation takes a string “name” and returns a player object. The DataContractSerializer can convert primitive types into XML for transport however for User-defined types Data contracts are needed for serialisation.



```
[DataContract]
53 references | Sangey, 8 days ago | 3 authors, 4 changes
public class Player
{
    [DataMember]
    14 references | 0/9 passing | Maddie, 19 days ago | 1 author, 1 change | 0 exceptions
    public int Id { get; set; }
    [DataMember]
    [StringLength(20, ErrorMessage ="Name must be between 1 and 20 characters", MinimumLength = 1)]
    [Index(IsUnique=true)]
    [Required]
    24 references | 0/9 passing | Sangey, 12 days ago | 2 authors, 2 changes | 0 exceptions
    public string Name { get; set; }
    [DataType(DataType.EmailAddress)]
    [EmailAddress(ErrorMessage = "Invalid email error. Please enter a valid email")]
    [Required]
    1 reference | Sangey, 8 days ago | 2 authors, 3 changes | 0 exceptions
    public string Email { get; set; }
    [Required]
    3 references | Sangey, 12 days ago | 2 authors, 2 changes | 0 exceptions
    public string HashedPassword { get; set; }
    5 references | 0/5 passing | Maddie, 19 days ago | 1 author, 1 change | 0 exceptions
    public List<Game> Games { get; set; }
```

Above is an example of a data contract for our player class. Data members specify what information is exposed to the WCF client proxy classes in this case the Player Id and Name.

Data Access Middleware

ADO.NET

In the .NET stack to access and modify data in the database ADO.NET is used to handle communication between the database and application server. ADO.NET is composed of two main components .NET Framework Data Providers and DataSets. [9]

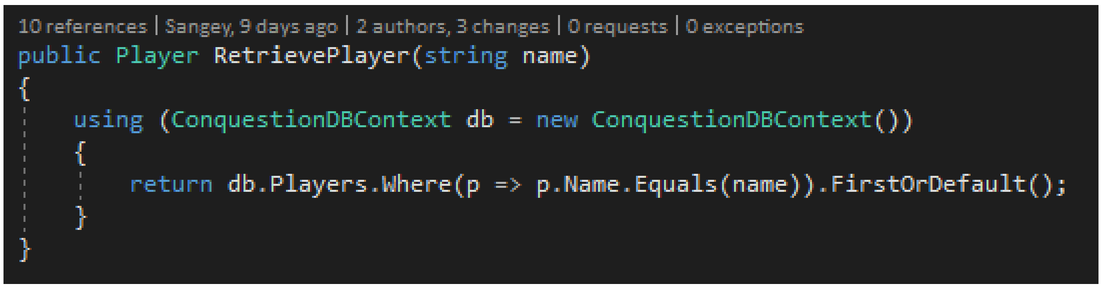
The .Net Framework Data Providers are responsible for connection to the database, and issuing database commands to retrieve or modify data within the database.

DataSets are a way to cache database information locally in the application accessing them. They are populated using Data Readers and Data Adapters and contain all necessary information regarding the row data in a DataTable object. This allows for extensive processing on data without needing an open connection to the database.

Entity Framework

Entity Framework is an ORM (Object-Relational Mapping tool) which allows developers to work with domain-specific objects such as players or questions in our project rather than the database tables and columns. This means developers spend less time writing ADO.NET code and more time on business logic coding. [10]

Another big benefit is the use of LINQ to write queries to the database which can simplify the querying process significantly.



```
10 references | Sangey, 9 days ago | 2 authors, 3 changes | 0 requests | 0 exceptions
public Player RetrievePlayer(string name)
{
    using (ConquestionDBContext db = new ConquestionDBContext())
    {
        return db.Players.Where(p => p.Name.Equals(name)).FirstOrDefault();
    }
}
```

An example of a simple LINQ query to retrieve a Player object from our database. We simply use our DBContext to access the Players table and using a where clause with a lambda expression to find the first or default row with a matching name.

However, as Entity Framework is built on top of ADO.NET this does mean there is more overhead for a complex tool resulting in reduced performance than pure ADO.NET.

Entity Framework supports 3 development approaches: Code First, Model First, Database First. For this project we will compare Code First and Database First.

Code First

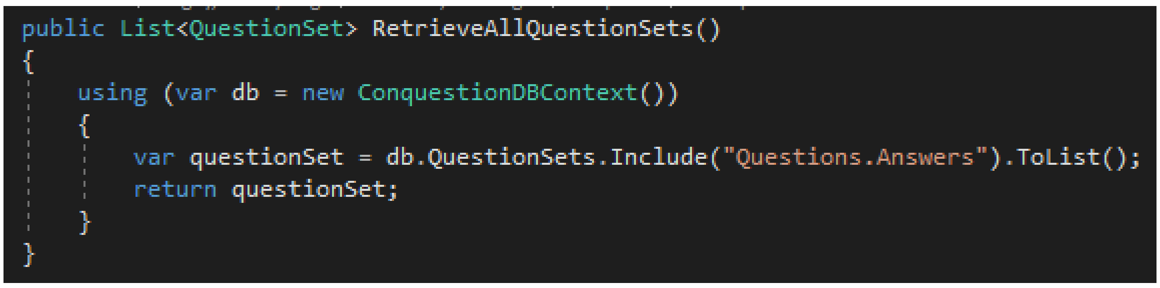
In code first approach you create your domain model classes and generating a database based on these domain model classes. This means very little to no manual control of the database creation using scripts is needed in this approach, entity framework will generate and modify the database as your domain changes making things very simple. However, a downside is that any manual changes to the database could be lost. [10]

Database First

In database first entities are generated from an already existing database automatically which can in turn be mapped in domain model classes. This approach allows for manual changes for the database and auto-updating of the model classes accordingly. Database first is great if you have an already existing database you want to base your project on. [10]

Lazy Loading vs Eager loading

When loading a main entity with relations to other entities from the database there are two approaches lazy loading and eager loading. Eager loading is when the main entity is loaded the related entities are loaded at the same time using the Include() method. [10]



```
public List<QuestionSet> RetrieveAllQuestionSets()
{
    using (var db = new ConquestionDBContext())
    {
        var questionSet = db.QuestionSets.Include("Questions.Answers").ToList();
        return questionSet;
    }
}
```

Lazy loading is loading the main entity without the relations at first, and only loading them when they are access via a new SQL query. This is achieved by marking properties on domain model classes as virtual.

Eager loading is good in situation where you know you will need the related data in the process being carried out e.g. loading a Question object with all the related Answer objects. Lazy loading is better for situations where the related data in accessed infrequently.

Eager loading generally will result in heavier calls to the database with table joins while lazy loading has the possibility of producing several SQL queries.

Our Choice

We choose to use Entity Framework using the code first approach. Given the time constraints and the amount of new technologies we were working with this time on this project we opted for Entity framework as we believed it would save us time in the long run.

After a short initial research spike and adjustment period getting used to EF we were able to write, refactor our code and database more easily and quickly. It saved a lot of time that would've been spent on writing database scripts and ADO.NET code.

Also we decided to use MS SQL server as our database for development for multiple reasons.

- Compatibility with Microsoft environment technologies (.NET)
- Ability to use Entity Framework and LINQ without needing 3rd party tools
- Previous experience working with MS SQL from the majority of our project member and familiarity with the technology.

Other Technologies Used

In this section we will briefly discuss some of the other technologies we used for our project.

Clients

Our system is being designed as a service oriented architecture to service multiple clients concurrently. Given the framework being used (WCF) for our service and its ability to service different platforms we are building two separate clients, a dedicated client and a web based client.

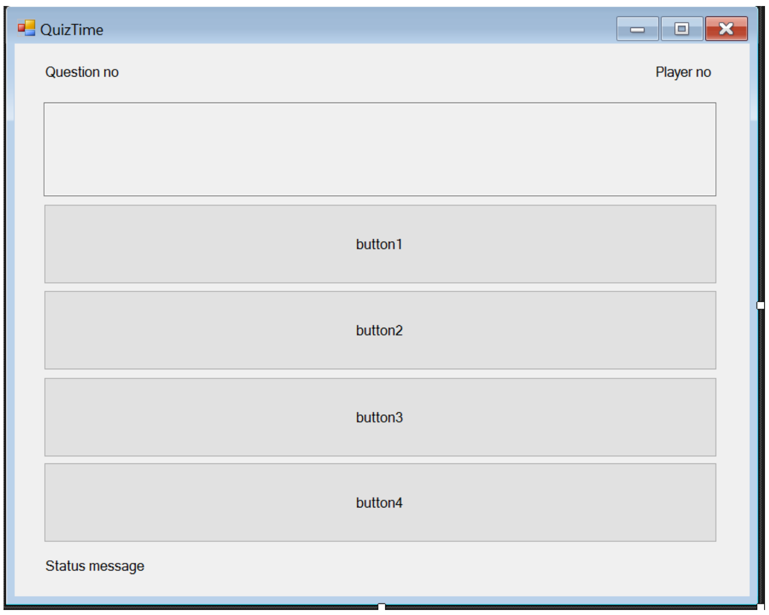
Dedicated Client

Before starting to work on our dedicated client we had to choose what technology to use. We had a pool of choices, but our major contenders were WPF and Winforms.

Winforms is Microsoft's own graphical class library and it is part of the .NET framework. This technology is less powerful than other options, however since it has a rich history, it is more tried and tested than anything else.

WPF (Windows Presentation Foundation) was also made by Microsoft, however it was only added in the .NET framework 3.0 and uses DirectX technology. It hides much more possibility than e.g. Winforms in designing the interface itself thanks to it using styles coded in XAML. This attribute of WPF can be a disadvantage as well. Making the Interface look as one would really want it to could be really time consuming and it can have a big learning curve as well.

We felt like that Winforms would suit us the best because our limited time and that our Interface was not the most important thing it this product. Furthermore, all of the members of our group already had experience with Winforms and we were more interested in learning how to use ASP.NET MVC.



Web Client

For building our web client we had the option to choose between ASP.NET Web Forms and ASP.NET MVC. Here are our reasons for choosing MVC over Web Forms: MVC was introduced after Web Forms to compensate some of its weaknesses. At a time when web development was in its inception many developers struggled with becoming familiar with the HTTP protocol and the paradigm shift from desktop applications to web applications. A major difficulty was the fact that unlike traditional desktop applications, HTTP is a stateless protocol which means according to [11] that: "as soon as the web server sends a response to the client browser, everything about the previous session is forgotten.". Web Forms offered to solution to this problem though it's viewstate. With this and many other features Web Forms became an essential tool meant to help developers with their web development projects. Over time the developers grew more familiar with HTTP and other web technologies and their needs changed. According to [11]: "These developers needed the bridging technologies less and less and wanted more and more control of the rendered views.". And so, to satisfy their demands Microsoft came up with ASP.NET MVC. Some of the advantages of MVC are: the MVC (Model-View-Controller) pattern secures the separation of concerns and therefore it is easier to manage complexity. In addition, it has better support for test-driven development, it's ideal for distributed and large teams and it provides a high degree of control over the application behaviour.

For the proper use of ASP.NET MVC developers need to be quite experienced with HTTP, HTML, client-side scripting languages like JavaScript and possibly CSS and the Bootstrap Framework if they want their web pages to look very nice. MVC allows developers to build dynamic views with Razor. According to [12]: "Razor is not a new programming language itself, but uses C# syntax for embedding code in a page without the ASP.NET delimiters: <%= %>.". Basically, it allows us to combine C# code with HTML and once one becomes familiar with using it, creating dynamic content for MVC Views will be much easier. However, the backbone of every web page is HTML. According to [11]: "HTML is a standard markup language used to describe how literal text, images, external links, and various HTML controls are to be rendered within the client-side browser.". It is considered by many to be the skeleton of a web page. While you can have web pages build only with HTML, CSS is usually added to the mix to provide a more user friendly interface. CSS (Cascading Style Sheets) is concerned with the look of a web page and it can be used to change the appearance of HTML elements depending on whatever style we would like them to be displayed as. In ASP.NET we use Bootstrap, a HTML, CSS and JavaScript Framework to ensure that our rendered views are also responsive meaning the look of the web page changes depending on the size of the browsers screen. Once one is content with the look and content of a web page, JavaScript and possibly jQuery is necessary to give it additional functionality. Continuing the metaphor, while HTML is the skeleton of the web page and CSS its outer appearance, JavaScript can be thought of like the muscles on the bones as it allows our page to "do something", specifically whatever we can program it to do.

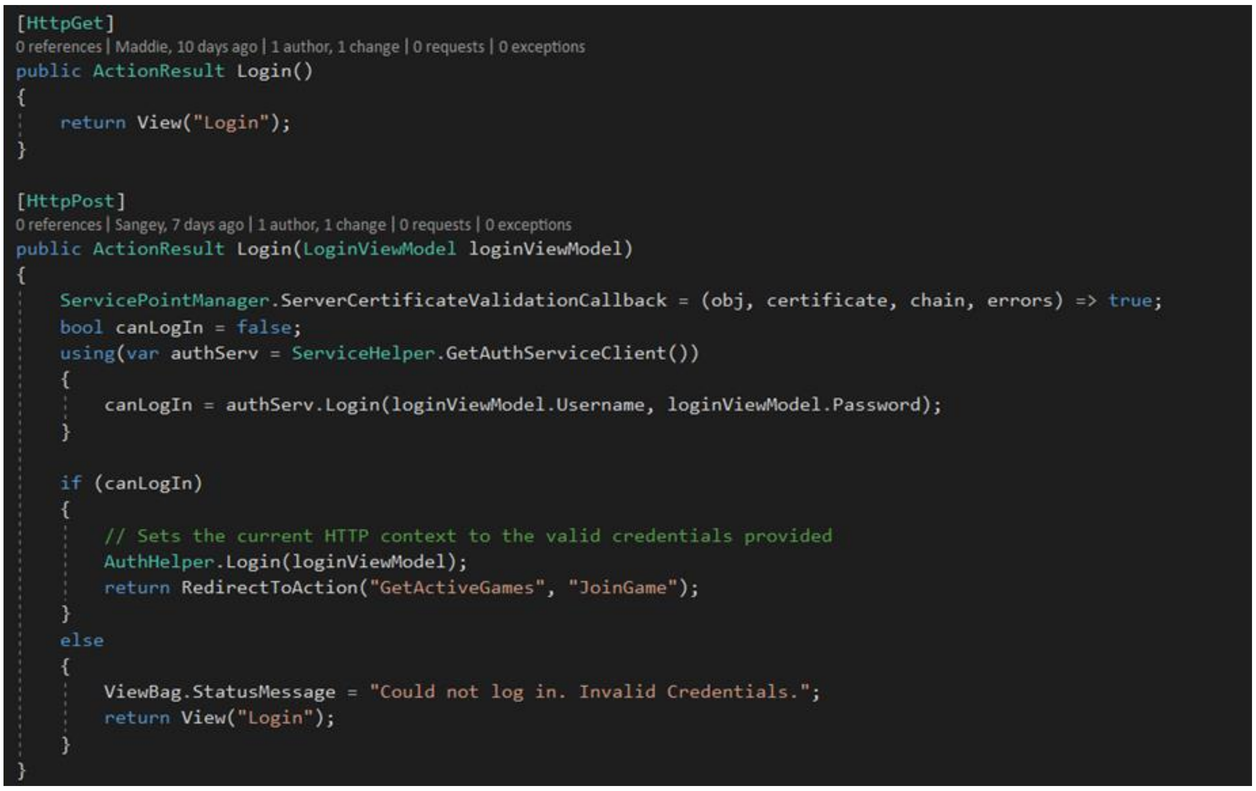
The ConquestionGame web client uses services to implement our most important use stories.

The projects default opening page is the Login View from the Account Controller.

The Controllers folder contains five distinct classes called: Account, JoinGame, Lobby and Quiz controller.

The Account Controller is responsible for managing the players Log In, Log Out and the registration of a new player.

Fig.1 Demonstrates our implementation for the "As a player I want to log in with my credentials." User story. The first Login ActionResult method is a GET method as only displays the Login View with an empty form for the player to input his credentials in. Once the player presses the Login button on the View then the POST Login ActionResult method is called. This method takes the players inputs, his username and password and then checks if they are valid by calling the service using an AuthServiceClient. If the player's credentials are valid then he/she is redirected to the GetActiveGames View from the JoinGame Controller, else the player is shown a warning that the login was unsuccessful.

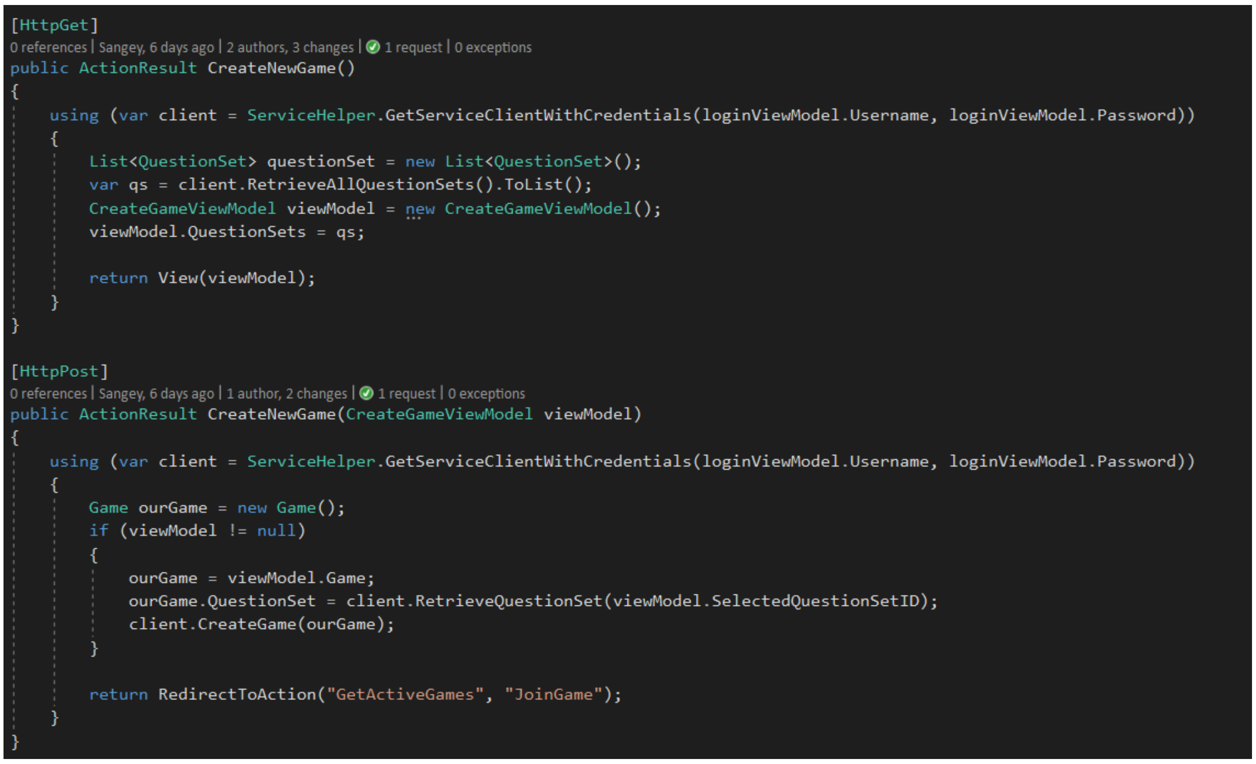


```
[HttpGet]
0 references | Maddie, 10 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public ActionResult Login()
{
    return View("Login");
}

[HttpPost]
0 references | Sangey, 7 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public ActionResult Login(LoginViewModel loginViewModel)
{
    ServicePointManager.ServerCertificateValidationCallback = (obj, certificate, chain, errors) => true;
    bool canLogIn = false;
    using(var authServ = ServiceHelper.GetAuthServiceClient())
    {
        canLogIn = authServ.Login(loginViewModel.Username, loginViewModel.Password);
    }

    if (canLogIn)
    {
        // Sets the current HTTP context to the valid credentials provided
        AuthHelper.Login(loginViewModel);
        return RedirectToAction("GetActiveGames", "JoinGame");
    }
    else
    {
        ViewBag.StatusMessage = "Could not log in. Invalid Credentials.";
        return View("Login");
    }
}
```

Fig. 1 Player Login code example

The JoinGame Controller is managing the display of active games done with GetActiveGames, the creation of a new game with CreateNewGame and the joining a lobby with the JoinGame ActionResult method.

Fig.2 This code example shows our implementation for creating a game. The CreateNewGame GET action method displays an empty form on the CreateNewGame View. The form has a dropdown list for selecting a Question Set so the items of that list are provided to the view by this method. The CreateNewGame POST action method creates a new game using the players inputs.



```
[HttpGet]
0 references | Sangey, 6 days ago | 2 authors, 3 changes | 1 request | 0 exceptions
public ActionResult CreateNewGame()
{
    using (var client = ServiceHelper.GetServiceClientWithCredentials(loginViewModel.Username, loginViewModel.Password))
    {
        List<QuestionSet> questionSet = new List<QuestionSet>();
        var qs = client.RetrieveAllQuestionSets().ToList();
        CreateGameViewModel viewModel = new CreateGameViewModel();
        viewModel.QuestionSets = qs;

        return View(viewModel);
    }
}

[HttpPost]
0 references | Sangey, 6 days ago | 1 author, 2 changes | 1 request | 0 exceptions
public ActionResult CreateNewGame(CreateGameViewModel viewModel)
{
    using (var client = ServiceHelper.GetServiceClientWithCredentials(loginViewModel.Username, loginViewModel.Password))
    {
        Game ourGame = new Game();
        if (viewModel != null)
        {
            ourGame = viewModel.Game;
            ourGame.QuestionSet = client.RetrieveQuestionSet(viewModel.SelectedQuestionSetID);
            client.CreateGame(ourGame);
        }

        return RedirectToAction("GetActiveGames", "JoinGame");
    }
}
```

Fig. 2. CreateNewGame code example

The Lobby Controller shows the game lobby with the help of the DisplayLobby action method. In this method there is a call to the private method Setup that passes information about the game and the question set to the view. The CheckIfLobbyHost method called in the Setup ensures that only the player who is the lobby host is shown the button which starts the game for all the other players. The players that are not hosts are informed with a message instead.

The Quiz Controller is concerned with the game play and with displaying the questions and their correct answers. The StartQuiz action method is called once at the beginning of the quiz to initialize it and then it redirects the player to the UniversalQuiz action which only displays the current question. If there are questions left the UniversalQuiz is called again and again else if there are no questions left then the player is redirected to the EndScreen. After the 30 seconds timer expires or after a player answers a question then he/she is redirected to the ShowCorrectAnswers action method. This method shows the player the correct answers for 5 seconds then calls the UniversalQuiz action and then repeats the cycle. When a player answers a question by selecting an answer then the AnswerSelected action is called with the answer id as a parameter. This method records the players answer and then redirects us to ShowCorrectAnswers.

According to [13]: “In ASP.NET MVC, ViewModels allow you to shape multiple entities from one or more data models or sources into a single object, optimized for consumption and rendering by the view.”. We use four different viewModels to display the Views with all the information they need.

The web client and the dedicated client have mostly achieved the same tasks despite their different implementations. One thing that the web client does not have however, is a check after the players have answered that all the players have answered. This check makes it so that after a player has answered, the timer goes on but you are not redirected to the ShowCorrectAnswers view yet. The players are all supposed to be shown the correct answers at the same time. Because of our lack in knowledge regarding MVC and JavaScript and lack of time we were not able to implement this feature.

The web client EndScreen and the dedicated clients one also differs in the fact that web client only displays the name of the winner of the game but no statistics like the dedicated client does. The statistics were not a priority, but they may be implemented in a future sprint given the chance.

Another difference is that unlike the dedicated client, the web clients list of players and games in the Views does not update constantly. Instead it is only loaded once when the View is called to display. Therefore, for example even if a new player joins the lobby, the player using the web client will not be shown the new players name, if they open the lobby page before the new player enters. The player would have to manually refresh the page or exit and re-enter the lobby for the updated list to be shown.

In the making of the Views we used a load of web technologies. HTML was often rendered using HTML helpers which was a significant help and a useful feature of MVC. In terms of the CSS, we created a minimal number of new styles and instead used Bootstrap and the default CSS style that the page had. JavaScript was used for implementing the timers on the UniversalQuiz and ShowCorrectAnswers view. jQuery was used in the ShowCorrectAnswers view to change the CSS property of color depending if the answers displayed were correct or not. Razor was used to dynamically display lists of items when they were needed and in if statements to display a different message to the view depending on the case.

Handling of simultaneity problems

For this project we must address the issue of handling concurrent clients connecting and using our service and especially for a multiplayer game. In this section we will talk about the problems in this area and how we tried to minimise their affects on our system.

Some of these problems include:

- Areas of our system where concurrency control is required
- Starting a game simultaneously for all clients in a game
- Presenting the same information to all players
- Checking if all players have answered
- Creating a new rounds

Concurrency Control

For concurrency we shall discuss two key types; optimistic and pessimistic concurrency control. [14]

Pessimistic Concurrency Control

In pessimistic concurrency control the system will assume that two clients will try to edit row data at the same time possibly resulting in negative concurrency effects (Lost updates, Dirty Reads etc). To prevent this pessimistic concurrency control locks are applied by a user to prevent other users editing data until they release the locks. [14]

Optimistic Concurrency Control

Optimistic concurrency control differs from pessimistic in that it doesn't use locks but rather compares the data from when it was initially read, to when an update is ready to be committed and checks if the data has changed in the interim. In a situation where the data has changed since the initial read, the transaction typically will be rolled back and an error message will appear. [14]

Transactions

We will briefly recap transactions and ACID properties and their importance in concurrency control. ACID properties are intended to guarantee the success of a logical operation in a database transaction, Atomicity, Consistency, Isolation, Durability.

- Atomicity – "All or nothing" the idea that if any part of the transaction step fails that the entire transaction is rolled back and no changes are saved to the database.
- Consistency – Guarantees that the transaction will result in a valid database state (constraints, keys, cascades are not violated) however this doesn't guarantee the validity of the transaction itself.
- Isolation – ensures the concurrent execution of transactions as if they were executed in isolation of each other (transactions should not interfere with each other).
- Durability – Guarantees after a transaction has been committed that it persists in the database in the event of database failure (power loss, crashes etc.) [15]

Isolation is important when we discuss concurrency control as it is the property of transaction to ensure transactions do not affect each other and that concurrency errors do not arise. In a database various levels of isolation can be set with varying properties. These levels control what types of locks are applied to the data and what operations can or can't be done by other transactions on the locked data (Reads or Writes).

In MS SQL server the levels available are from highest to lowest; Serializable, Repeatable read, Read committed, Read uncommitted. [16]

For this project we will just compare the highest isolation level (Serializable) and SQL's default level (Read committed).

- Serializable – Read and write locks are acquired on selected data and are only released at the end of the transactions and range-locks must be acquired if a ranged WHERE clause is used. The range locks are important in preventing a case of "phantom reads" where a range of data changes (new inserted data or deleted data) between the initial transaction read and the final commit.
- Read committed – Write locks are acquired and only released at the end of the transaction while read locks are released immediately after SELECT queries are processed. This prevents dirty reads (reading of uncommitted changes) however, it does not prevent phantom reads or non-repeatable reads (changes to data since the initial read).

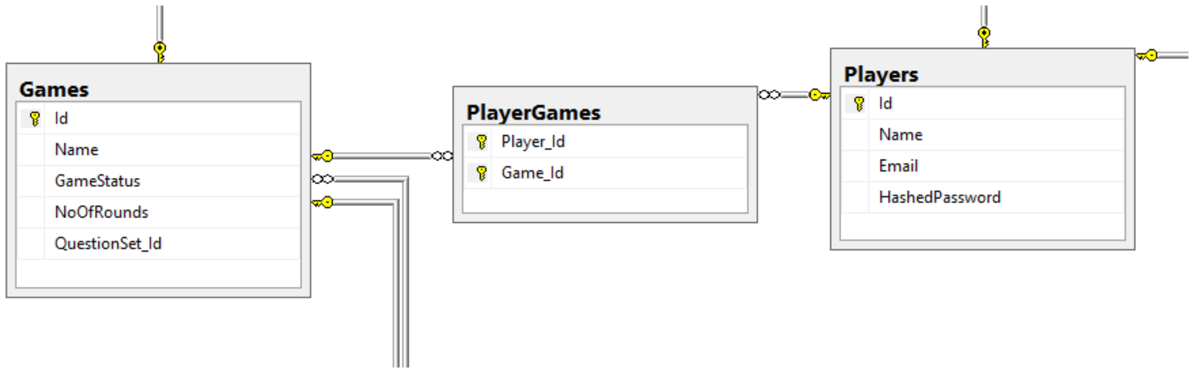
A question one might ask why not use the highest level of isolation all the time? There are some reasons to relax the isolation level in the database such as deadlocking and overhead costs in locking. Acquiring locks causes overhead in the database, resulting in slower performance while the likelihood of deadlocks increases (where two transactions are waiting for the other transaction's lock). [17]

Project examples

Pessimistic Concurrency

In our project we identified some key areas where concurrency between multiple clients could be problematic. For example, in an early user stories we agreed on a rule that a max of 4 players could play at once.

To handle the max number of players joining a game "lobby" (setting up before the game starts) we need to take a look at our database architecture.

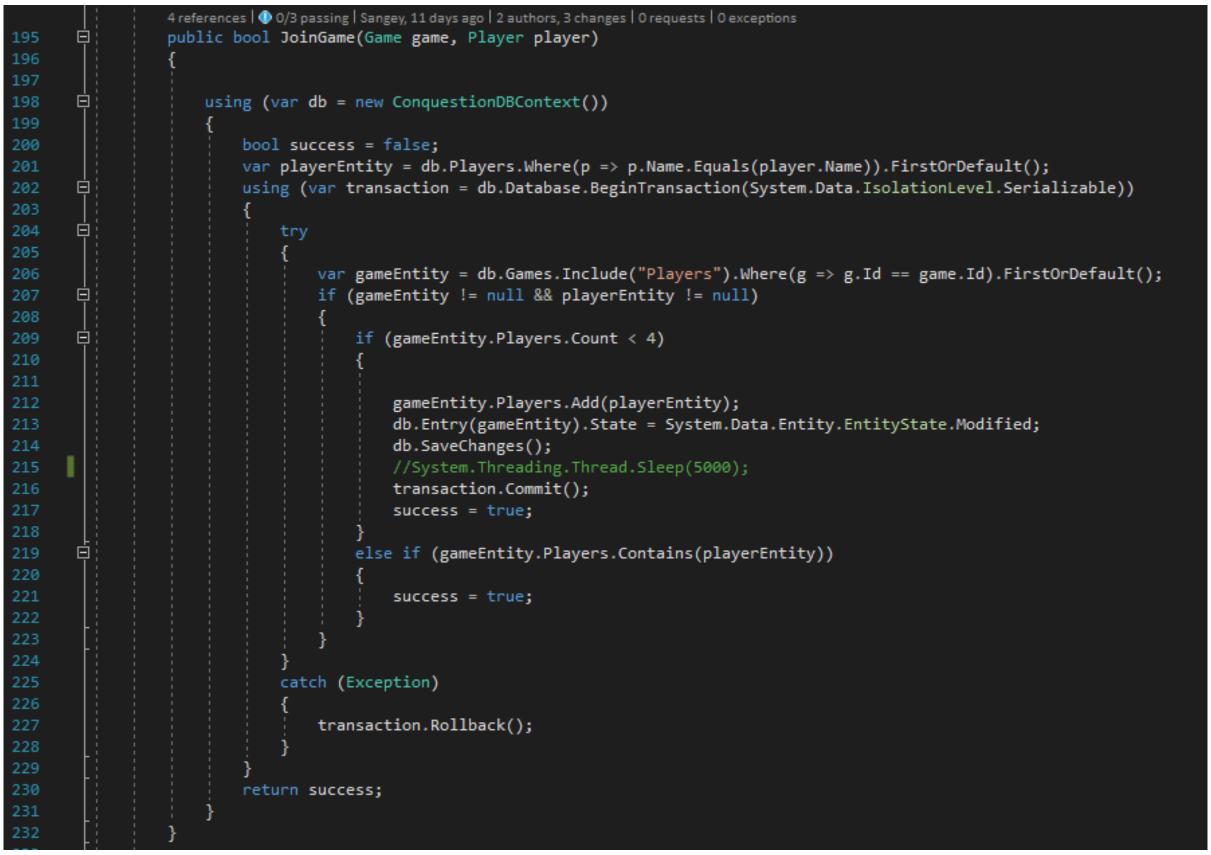


Here is a diagram from our MS SQL database showing how we persist this information. From our domain model we knew a game could have multiple players, but a player could also be associated with multiple games (games played previously). Due to this many-to-many relation we needed to map which players were currently associated to a game using a junction table "PlayerGames".

The concurrency problem occurs when we have a lobby with 3 players and 2 extra players try to join at the same time. There is a possibility that both player could successfully join meaning 5 players in a game (phantom read error).

To prevent this situation, we must use pessimistic concurrency control in our database, optimistic is not viable as we are inserting new row data and can't check against an already existing data for changes.

As there is a possibility for phantom reads in this situation we are using the highest isolation level serializable to acquire range locks for our data.



```
                4 references | 0/3 passing | Sangey, 11 days ago | 2 authors, 3 changes | 0 requests | 0 exceptions
195             public bool JoinGame(Game game, Player player)
196             {
197
198                 using (var db = new ConquestionDBContext())
199                 {
200                     bool success = false;
201                     var playerEntity = db.Players.Where(p => p.Name.Equals(player.Name)).FirstOrDefault();
202                     using (var transaction = db.Database.BeginTransaction(System.Data.IsolationLevel.Serializable))
203                     {
204                         try
205                         {
206                             var gameEntity = db.Games.Include("Players").Where(g => g.Id == game.Id).FirstOrDefault();
207                             if (gameEntity != null && playerEntity != null)
208                             {
209                                 if (gameEntity.Players.Count < 4)
210                                 {
211
212                                     gameEntity.Players.Add(playerEntity);
213                                     db.Entry(gameEntity).State = System.Data.Entity.EntityState.Modified;
214                                     db.SaveChanges();
215                                     //System.Threading.Thread.Sleep(5000);
216                                     transaction.Commit();
217                                     success = true;
218                                 }
219                                 else if (gameEntity.Players.Contains(playerEntity))
220                                 {
221                                     success = true;
222                                 }
223                             }
224                         }
225                         catch (Exception)
226                         {
227                             transaction.Rollback();
228                         }
229                     }
230                     return success;
231                 }
232             }
```

Above is an example of us using Serializable isolation in our join game method on our game controller. Usually when we use entity framework when we call db.SaveChanges() EF will automatically wrap any changes made in a transaction for us to the database, however in this case because the database default isolation level is Read committed and we want to use Serializable we have to manually set the isolation level. [18]

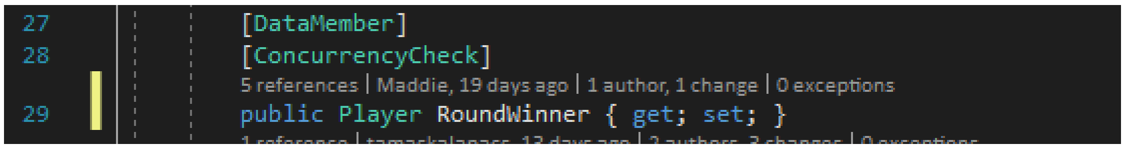
We set the isolation level on line 202 using Entity framework's BeginTransaction() method which allows us to specify an isolation level to be used as a parameter. We set the parameter to Serializable and wrap the logic in a using statement using the transaction variable. [18]

This should prevent phantom reads from occurring, committing the changes and return true if there are no errors or rolling back the transaction in the case exceptions occur.

Optimistic Concurrency

Our main concurrency control situation for our quiz was determining the winner of a round. A RoundWinner property on our Round entities determined which player answered the quickest and answered correctly. However if we use the default database settings there is a chance for a lost update error.

To prevent this we used optimistic concurrency control on the RoundWinner property.



```
        [DataMember]
        [ConcurrencyCheck]
        5 references | Maddie, 19 days ago | 1 author, 1 change | 0 exceptions
        public Player RoundWinner { get; set; }
        1 reference | tamaskalapacs, 13 days ago | 2 authors, 2 changes | 0 exceptions
```

Entity framework allows us to use DataAnnotations such as [Concurrency Check] to allow us to mark properties for optimistic concurrency control. With this annotation entity framework knows to compare the read value on RoundWinner and use it when updating the database (this is done by including the read value as part of the where clause as part of the SQL update query). [19]

As this is optimistic concurrency control that means we don't need to use locking strategies, so we don't need to manually configure transactions in our round controller.

Managing multiple clients

As our project is a multiplayer game with a time sensitive element when it comes to answering questions we had to address an issue of how to signal to all the clients that the game had begun.

We had two options, either we get our clients to continuously pull from our server to check if the game had started (approx. once a second) or the server can push to all our clients signalling something has changed.

Ideally we would like to implement a push strategy, as multiple clients constantly pulling from the server could cause significant performance issues or even overload the server if the number of clients becomes too high. Unfortunately for this project and once again with time constraints as learning the use of a new push technology (e.g. SignalR) would have added too much time to our already tight schedule. Perhaps in a future iteration we could implement push technology.

Extra Implementation

In this section we will talk about some other aspects of our system such as security we implemented, measures we took to counter act cheating, and how we authenticate users and user commands.

Security

While security was not one our Must have topics for this project we found the topic interesting and wanted to implement some security functions to our game as it can be relevant in multiplayer games in regards to cheating.

TLS/SSL

TLS (Transport Layer Security)/SSL (Secure Socket Layer) are cryptographic protocols designed to encrypt data to keep any sensitive data being transmitted private. These can be used to help prevent man in the middle attacks which try to intercept messages between clients and server. [20]

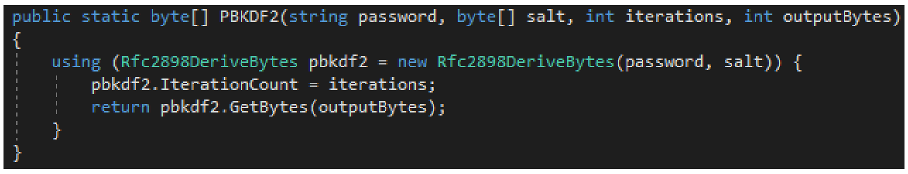
Secure password storage

Hashing/Salt:

Hashing algorithm: makes an amount of data into a string of fixed size which is made so you cannot invert it and find out what the data is by the hash.

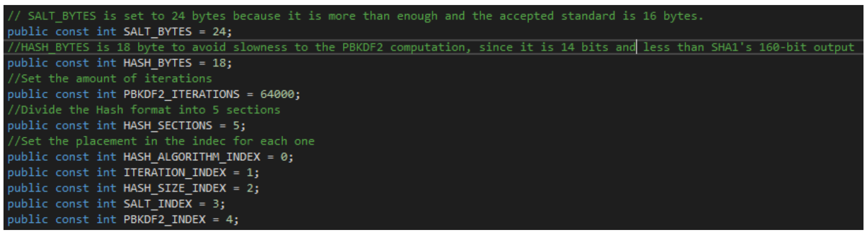
Salt: is a random data that is used together with the data before it is hashed. They use the concept of nonce. The use of salt is to make passwords that are similar to have different salts, so you do not know what ones password is if you know someone else's, to protect against rainbow table (a precomputed table for reversing cryptographic hash functions) and dictionary attacks (try multiple combination of pre-determined words). Salts is never to be used twice (if you change password a new salt is used) and is not to be too short (so the attacker will not be able to look up all the possible combinations of salt).

In cryptography you cannot store a password in plain text or even if you just hash it, it could still be cracked using rainbow table or dictionary attacks. Because of that you need to use salt. The salt makes the hash harder to crack against dictionary attacks. To make it even harder to crack the password you can use key stretching which is a CPU intensive hash function like the PBKDF2, this value determines how slow the hash function will be.



```
public static byte[] PBKDF2(string password, byte[] salt, int iterations, int outputBytes)
{
    using (Rfc2898DeriveBytes pbkdf2 = new Rfc2898DeriveBytes(password, salt)) {
        pbkdf2.IterationCount = iterations;
        return pbkdf2.GetBytes(outputBytes);
    }
}
```

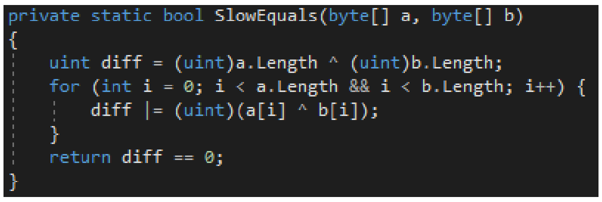
We use a hash size of 18Bytes, a salt size of 24 Bytes and a PBKDF2 iteration of 64 000. And we store the algorithm, iterations, hash size, salt, hash information in the database and we do not use multiple algorithms since we go by Kerckoffs's principle, which assumes that whoever is attacking us will know exactly what we use making extra algorithms only a minor inconvenience. [21]



```
// SALT_BYTES is set to 24 bytes because it is more than enough and the accepted standard is 16 bytes.
public const int SALT_BYTES = 24;
//HASH_BYTES is 18 byte to avoid slowness to the PBKDF2 computation, since it is 14 bits and less than SHA1's 160-bit output
public const int HASH_BYTES = 18;
//Set the amount of iterations
public const int PBKDF2_ITERATIONS = 64000;
//Divide the Hash format into 5 sections
public const int HASH_SECTIONS = 5;
//Set the placement in the indec for each one
public const int HASH_ALGORITHM_INDEX = 0;
public const int ITERATION_INDEX = 1;
public const int HASH_SIZE_INDEX = 2;
public const int SALT_INDEX = 3;
public const int PBKDF2_INDEX = 4;
```

When a user log in to our system we use a method to compare the hash in the database with the password that is used to log in. The format (algorithm, iterations, hash size, salt and hash) is retrieved from the database split into sections and the salt, no of iterations and the hash length are used to hash the password provided, it is then compared to the password hash stored in the database and if they match the user will be authenticated.

The method that compares the two hashes need to take the same amount of time so an attacker cannot make use of timing attacks. To understand how SlowEquals work see [22].

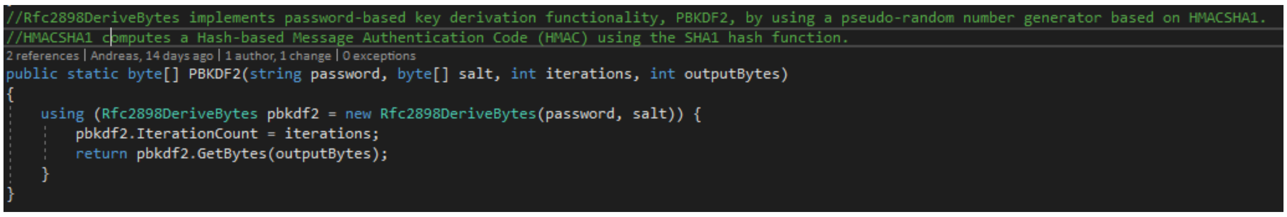


```
private static bool SlowEquals(byte[] a, byte[] b)
{
    uint diff = (uint)a.Length ^ (uint)b.Length;
    for (int i = 0; i < a.Length && i < b.Length; i++) {
        diff |= (uint)(a[i] ^ b[i]);
    }
    return diff == 0;
}
```

SHA-1:

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that makes data into a 160-bit (20-byte) hash value (hexadecimal). Since 2005 SHA-1 have been considered unsecure against brute force attacks with a lot of resources, [23] it has been recommended to swap to and use SHA-2 (Secure Hash Algorithm 2) or SHA-3 (Secure Hash Algorithm 3). Microsoft, Google, Apple and Mozilla have all stated that they will no longer support SHA-1 SSL certificates by 2017, and any web page using it will be marked as not secure. [24] In 2017 CWI and Google were succeeded in finding a collision attack publishing two similar PDF files which created the same hash with SHA-1. [25]

In our project we used SHA-1 because we had found a well written and documented code for it [26] and it is only a security threat/not secure enough if the attacker is well funded, since our program is not large scale and no one with a lot of money and resources will attack it and with PBKDF2 it is considered to be secure. In the future we would like to implement the password security with SHA-2 so it is considered more secure.



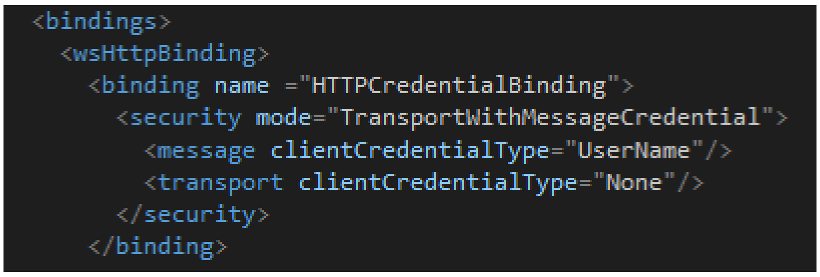
```
//Rfc2898DeriveBytes implements password-based key derivation functionality, PBKDF2, by using a pseudo-random number generator based on HMACSHA1.
//HMACSHA1 computes a Hash-based Message Authentication Code (HMAC) using the SHA1 hash function.
2 references | Andreas, 14 days ago | 1 author, 1 change | 0 exceptions
public static byte[] PBKDF2(string password, byte[] salt, int iterations, int outputBytes)
{
    using (Rfc2898DeriveBytes pbkdf2 = new Rfc2898DeriveBytes(password, salt)) {
        pbkdf2.IterationCount = iterations;
        return pbkdf2.GetBytes(outputBytes);
    }
}
```

SQL Injection

SQL Injection: Is when a user is inserting malicious SQL code into a string field to be executed e.g. the user could input the code “Player1'; drop table Players—“ where the comment marks ends the string for the user and the “--“ makes all the text after to be ignored. [14] In our code we do not protect against SQL injections because we are using LINQ which passes data to the database via SQL parameters safeguarding against SQL injections. [27]

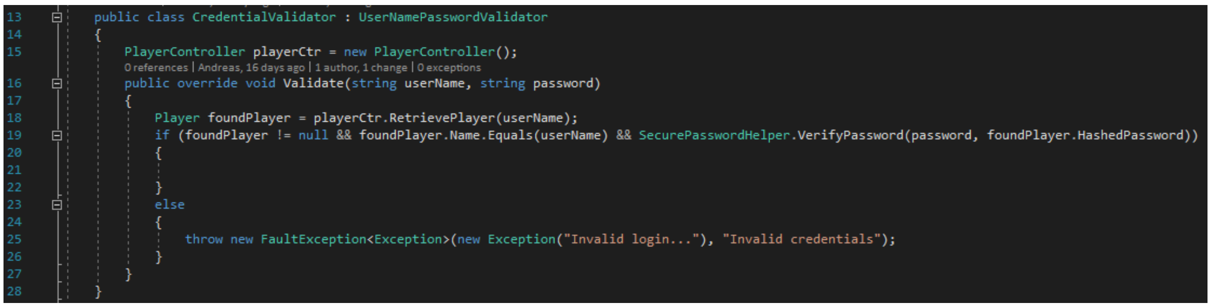
Authentication

We authenticate our users by customising our conquestion service endpoint requiring any message sent also provide ClientCredentials in the form of a user name and password.



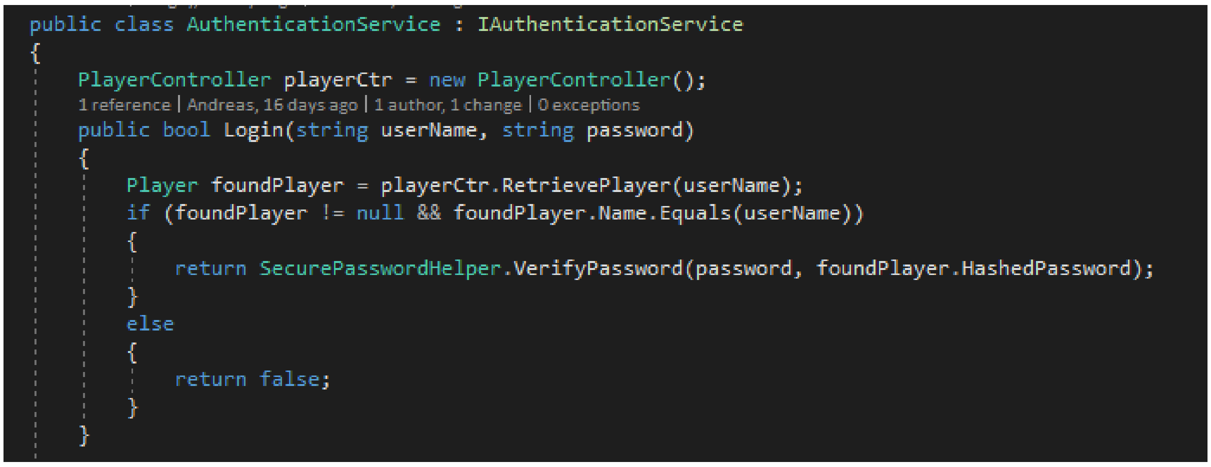
```
<bindings>
  <wsHttpBinding>
    <binding name ="HTTPCredentialBinding">
      <security mode="TransportWithMessageCredential">
        <message clientCredentialType="UserName"/>
        <transport clientCredentialType="None"/>
      </security>
    </binding>
```

The service will then use our customer validator class to validate the credentials provided and if no fault exceptions are thrown it is considered an authenticated message.



```
public class CredentialValidator : UserNamePasswordValidator
{
    PlayerController playerCtr = new PlayerController();
    0 references | Andreas, 16 days ago | 1 author, 1 change | 0 exceptions
    public override void Validate(string userName, string password)
    {
        Player foundPlayer = playerCtr.RetrievePlayer(userName);
        if (foundPlayer != null && foundPlayer.Name.Equals(userName) && SecurePasswordHelper.VerifyPassword(password, foundPlayer.HashedPassword))
        {

        }
        else
        {
            throw new FaultException<Exception>(new Exception("Invalid login..."), "Invalid credentials");
        }
    }
}
```

However this would cause a problem with logging in initially, as you would need to provide valid log in credentials to attempt to log in. To get around this problem we create a separate service that doesn't require credentials to make calls to the service. This Authentication service allows user to attempt to login and upon successful login their credentials are passed and used in Conquestion Service.

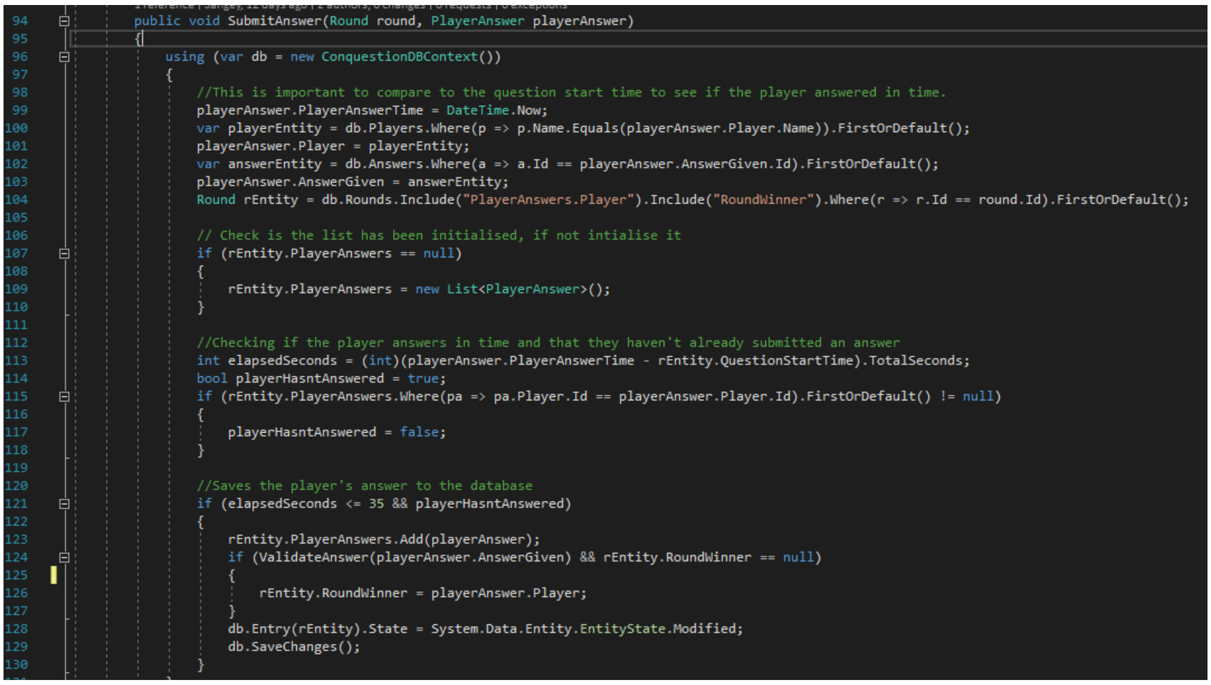


```
public class AuthenticationService : IAuthenticationService
{
    PlayerController playerCtr = new PlayerController();
    1 reference | Andreas, 16 days ago | 1 author, 1 change | 0 exceptions
    public bool Login(string userName, string password)
    {
        Player foundPlayer = playerCtr.RetrievePlayer(userName);
        if (foundPlayer != null && foundPlayer.Name.Equals(userName))
        {
            return SecurePasswordHelper.VerifyPassword(password, foundPlayer.HashedPassword);
        }
        else
        {
            return false;
        }
    }
```

Cheating

Cheating in a game was an important area for us to discuss as our project is online multiplayer based, cheating in single player doesn't ruin anyone else's experience of the game. Therefore we took steps to try and limit players ability to cheat in our game.

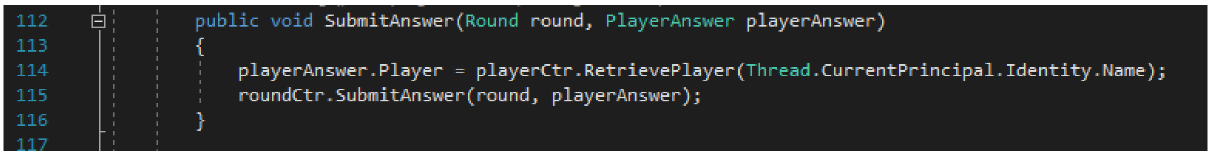
One area we identified was possibly players could alter their local computer's system clock or try to spoof their messages with a modified DateTime to when they answered. To counter this, rather than use DateTime sent from the client the server would overwrite the DateTime to when it received a PlayerAnswer. The time would then be compared to the round's QuestionStartTime to see if it submitted in time.



```
public void SubmitAnswer(Round round, PlayerAnswer playerAnswer)
{
    using (var db = new ConquestionDBContext())
    {
        //This is important to compare to the question start time to see if the player answered in time.
        playerAnswer.PlayerAnswerTime = DateTime.Now;
        var playerEntity = db.Players.Where(p => p.Name.Equals(playerAnswer.Player.Name)).FirstOrDefault();
        playerAnswer.Player = playerEntity;
        var answerEntity = db.Answers.Where(a => a.Id == playerAnswer.AnswerGiven.Id).FirstOrDefault();
        playerAnswer.AnswerGiven = answerEntity;
        Round rEntity = db.Rounds.Include("PlayerAnswers.Player").Include("RoundWinner").Where(r => r.Id == round.Id).FirstOrDefault();

        // Check is the list has been initialised, if not intialise it
        if (rEntity.PlayerAnswers == null)
        {
            rEntity.PlayerAnswers = new List<PlayerAnswer>();
        }

        //Checking if the player answers in time and that they haven't already submitted an answer
        int elapsedSeconds = (int)(playerAnswer.PlayerAnswerTime - rEntity.QuestionStartTime).TotalSeconds;
        bool playerHasntAnswered = true;
        if (rEntity.PlayerAnswers.Where(pa => pa.Player.Id == playerAnswer.Player.Id).FirstOrDefault() != null)
        {
            playerHasntAnswered = false;
        }

        //Saves the player's answer to the database
        if (elapsedSeconds <= 35 && playerHasntAnswered)
        {
            rEntity.PlayerAnswers.Add(playerAnswer);
            if (ValidateAnswer(playerAnswer.AnswerGiven) && rEntity.RoundWinner == null)
            {
                rEntity.RoundWinner = playerAnswer.Player;
            }
            db.Entry(rEntity).State = System.Data.Entity.EntityState.Modified;
            db.SaveChanges();
        }
```

Another area we identified could that player objects being passed from the client to the service could be altered to pretend to be another player. E.g. answer incorrectly for another player before they have a chance to answer.

This was a little more difficult to deal with as we still needed to pass player objects to our service. Our solution was to use the System.Security.Principal to retrieve the identity used in the validated credentials used to call the service. We used the name property from this identity and used that to retrieve our player objects rather than a provided player object.

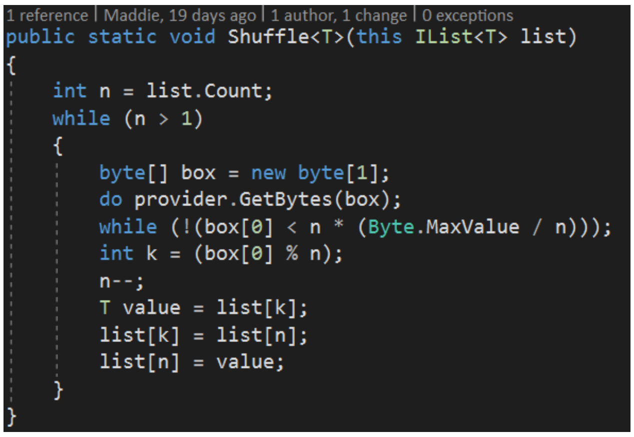


```
public void SubmitAnswer(Round round, PlayerAnswer playerAnswer)
{
    playerAnswer.Player = playerCtr.RetrievePlayer(Thread.CurrentPrincipal.Identity.Name);
    roundCtr.SubmitAnswer(round, playerAnswer);
}
```

Extension Class

We decided to write an extension method and to help use randomise the order of lists when creating a list of random questions to be used in a game.

To achieve the shuffling we used an algorithm called Fisher-Yates shuffle, with a few changes that was made by Richard Durstenfeld to make it more efficient. Below you can find our implementation of the shuffle. [28]



```
1 reference | Maddie, 19 days ago | 1 author, 1 change | 0 exceptions
public static void Shuffle<T>(this IList<T> list)
{
    int n = list.Count;
    while (n > 1)
    {
        byte[] box = new byte[1];
        do provider.GetBytes(box);
        while (!(box[0] < n * (Byte.MaxValue / n)));
        int k = (box[0] % n);
        n--;
        T value = list[k];
        list[k] = list[n];
        list[n] = value;
    }
}
```

Conclusion

For this project we set out to build a distributed system that provides a service that could be consumed by two different clients. We aimed to build a multiplayer quiz game that could be played with other people online. We also set out to learn more about technologies behind distributed systems and considerations needed when developing them.

Overall we feel we succeeded in these goals, our system is not perfect by any means but it is a working example of a distributed system and we can play a game to completion with multiple classes. We also were able to familiarise ourselves with many new technologies:

- C#
- Entity Framework
- WCF
- ASP.NET MVC
- SOAP

Given more time and resources we would have liked to expand the scope of our game and included a map and conquering mechanics we original envisioned. Also work on optimizing our game making better use of server push technologies such as SignalR or implementing better security functionality for the web based client.

We all learned the fundamentals of building distributed systems, the challenges involved in developing them, their benefits and their downsides. We also have an understanding on underlying communication principles between distributed systems and measures that can be taken to secure them. Finally we considered the implications of managing concurrent clients in a distributed system and the issues that can arise from concurrency control and performance.

References

[1] I. Sommerville, Software Engineering 10th Edition, Pearson, 2016.

[2] “Chapter 3: Architectural Patterns and Styles,” [Online]. Available: https://msdn.microsoft.com/en-us/library/ee658117.aspx#ServiceOrientedStyle. [Accessed December 2017].

[3] “Web Services Explained,” [Online]. Available: https://www.service-architecture.com/articles/web-services/web_services_explained.html. [Accessed December 2017].

[4] W3Schools, [Online]. Available: https://www.w3schools.com. [Accessed December 2017].

[5] “Understanding SOAP and REST Basics and Differences,” [Online]. Available: https://blog.smartbear.com/apis/understanding-soap-and-rest-basics/. [Accessed December 2017].

[6] “What is windows communication foundation,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf. [Accessed December 2017].

[7] “WCF Service Binding Explained,” [Online]. Available: http://www.c-sharpcorner.com/uploadfile/pjthesedays/wcf-service-binding-explained/. [Accessed December 2017].

[8] “Data transfer in service contracts,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/specifying-data-transfer-in-service-contracts. [Accessed December 2017].

[9] “Microsoft ActiveX Data Objects,” [Online]. Available: https://docs.microsoft.com/en-us/sql/ado/microsoft-activex-data-objects-ado. [Accessed December 2017].

[10] “Entity Framework (EF) Documentation,” [Online]. Available: https://msdn.microsoft.com/en-us/library/ee712907(v=vs.113).aspx. [Accessed December 2017].

[11] A. Troelsen, Pro C# 6.0 and the .NET 4.6 Framework 7th Edition, Apress, 2015.

[12] “Getting started with razor view engine,” [Online]. Available: http://www.c-sharpcorner.com/UploadFile/3d39b4/getting-started-with-razor-view-engine-in-mvc-3/ . [Accessed December 2017].

[13] “Use ViewModels to manage data & organize code in ASP.NET MVC applications,” [Online]. Available: http://rachelappel.com/use-viewmodels-to-manage-data-amp-organize-code-in-asp-net-mvc-applications/. [Accessed December 2017].

[14] “Types of Concurrency Control,” [Online]. Available: https://technet.microsoft.com/en-us/library/ms189132(v=sql.105).aspx. [Accessed December 2017].

[15] "ACID Properties," [Online]. Available: https://msdn.microsoft.com/en-us/library/aa480356.aspx. [Accessed December 2017].

[16] "Isolation levels in the database engine," [Online]. Available: https://technet.microsoft.com/en-us/library/ms189122(v=sql.105).aspx. [Accessed December 2017].

[17] "Understanding Isolation levels," [Online]. Available: https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-isolation-levels. [Accessed December 2017].

[18] "Entity Framework Working With Transactions (EF6 Onwards)," [Online]. Available: https://msdn.microsoft.com/en-us/library/dn456843(v=vs.113).aspx. [Accessed December 2017].

[19] "Handling Concurrency with the Entity Framework 6 in a ASP.NET MVC 5 Application," [Online]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/handling-concurrency-with-the-entity-framework-in-an-asp-net-mvc-application. [Accessed December 2017].

[20] "What is SSL, TLS and HTTPS," [Online]. Available: https://www.symantec.com/page.jsp?id=ssl-information-center. [Accessed December 2017].

[21] "Kerckhoffs's principle," [Online]. Available: http://www.crypto-it.net/eng/theory/kerckhoffs.html . [Accessed December 2017].

[22] "Salted password hashing - doing it right," [Online]. Available: https://crackstation.net/hashing-security.htm. [Accessed December 2017].

[23] "Cryptanalysis of SHA-1," [Online]. Available: https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html. [Accessed December 2017].

[24] "Windows enforcement of SHA1 Certificates," [Online]. Available: [4] https://social.technet.microsoft.com/wiki/contents/articles/32288.windows-enforcement-of-sha1-certificates.aspx. [Accessed December 2017].

[25] "CWI, Google announce first collision for industry security standard SHA-1," [Online]. Available: https://phys.org/news/2017-02-cwi-google-collision-industry-standard.html. [Accessed December 2017].

[26] "Secure Password Storage v2.0," [Online]. Available: https://github.com/defuse/password-hashing/tree/a00bdf9b6d7861bff2fb9ea95de4bd209b3252fe. [Accessed December 2017].

[27] "Security Considerations (Entity Framework)," [Online]. Available: https://msdn.microsoft.com/en-us/library/cc716760(v=vs.100).aspx. [Accessed December 2017].

[28] D. E. Knuth, The Art of Computer Programming, Addison-Wesley, 1968.