Cover Page

Title Page

Table of 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

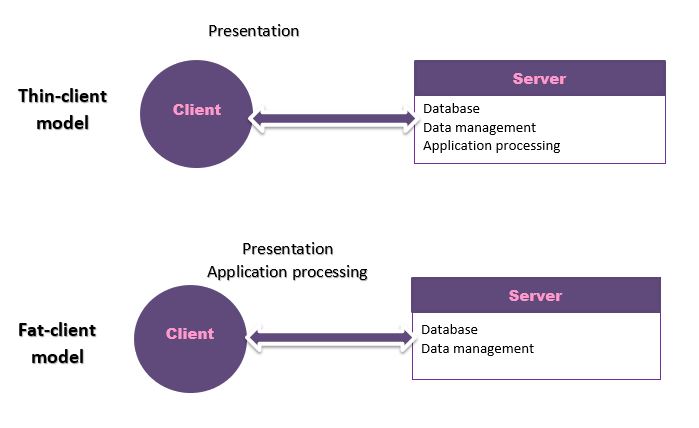
When designing distributed systems selecting an appropriate architecture is important. The choice of architecture will determine how easy it is to maintain and scale your system in the future as well as isolating sensitive information.

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

1. Two-tier architecture
   1. Thin client
   2. Fat client
2. N-tier / Multi-tier
   1. Classic client/server
   2. Classic web
   3. 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.



### 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.

### 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 more 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.

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 example

### 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.

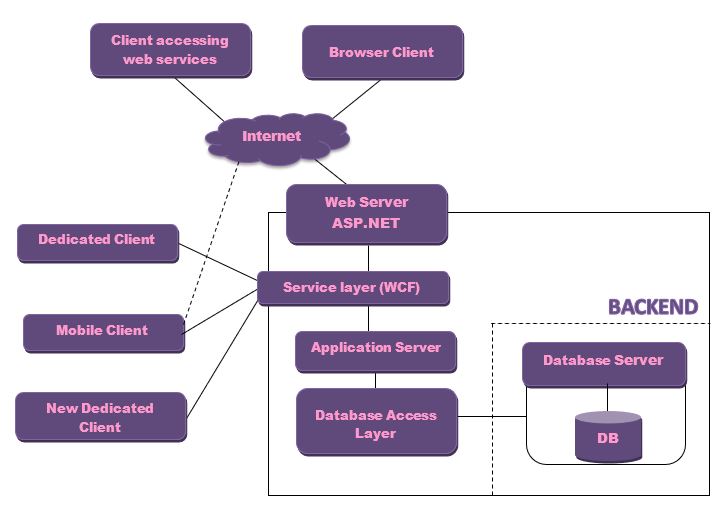
However, as we want to support web clients as well, this example is not suitable for our system.

### Classic web architecture

### 

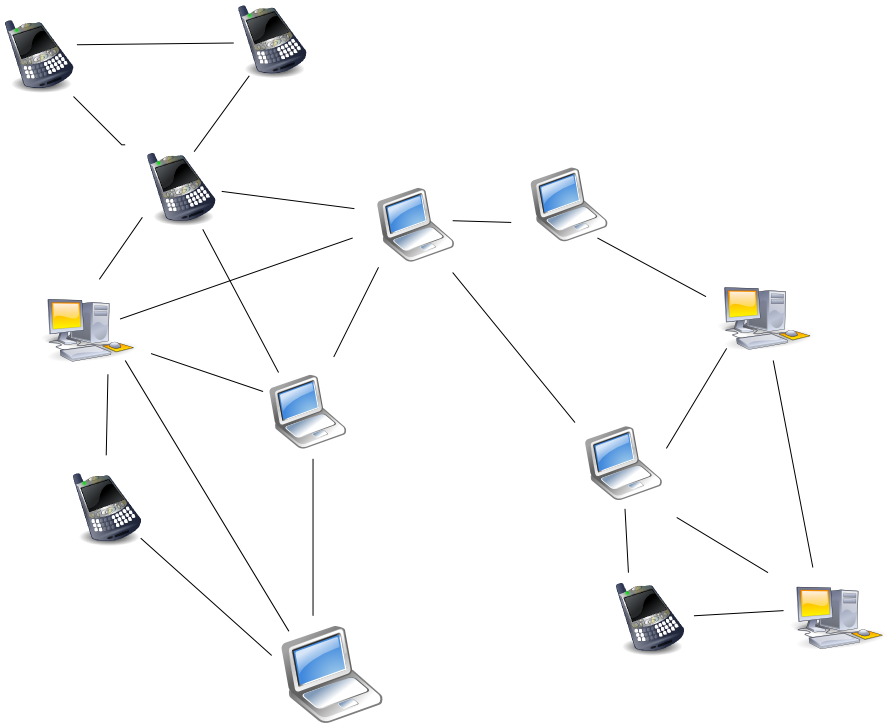
In this example we still have the dedicate client, application server, database but now also a webserver and browser clients. This solves our problem of supporting both a web client and a dedicated client however, in this example there is duplication of the code on the web server and application server 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 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.

## Peer-to-peer architecture

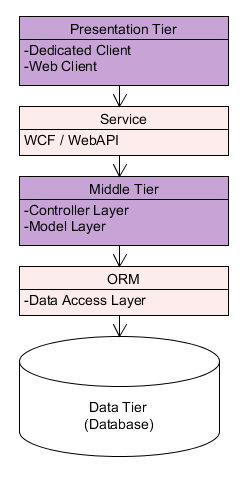


In peer to peer architecture, peers are machines interconnected to each other without need of a centralised server. This means each machine on a peer-to-peer network act both as a client and server. Peers share a portion of their computer’s resources, their bandwidth, storage space or processing power. 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, 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.

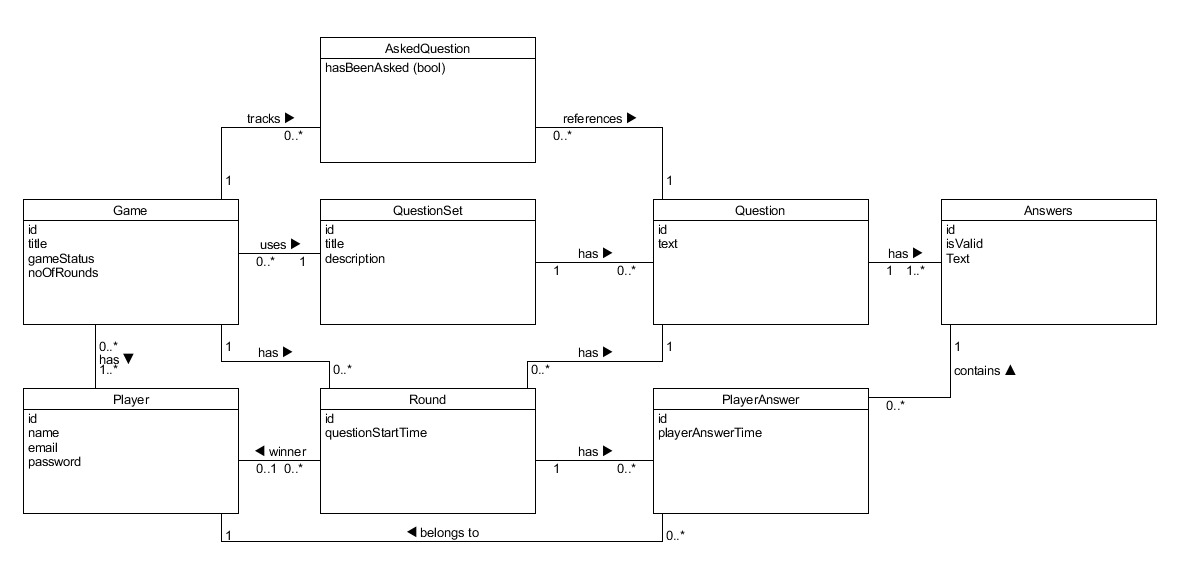
## Our choice of architecture

For this project we have elected to use the n-tier service orientated architecture. We have selected n-tier as for our project as we need to create both a dedicated client and web based client 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 dedicate client and web client. Middle Tier contains our business logic layers for our game logic. Data is 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

# 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.

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.

### REST

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

* 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.

Arguments for SOAP

* SOAP is support 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.

#### 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.

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.”

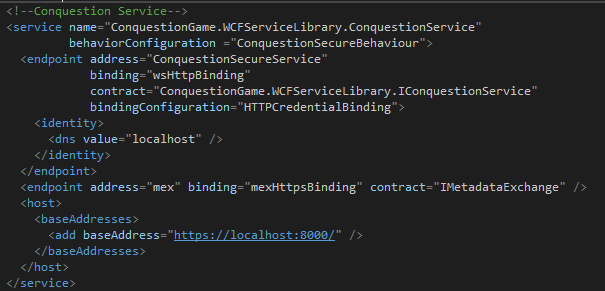
Other binding options available were: netTcpBinding, netNamedPipeBinding, netPeerTcpBinding

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.

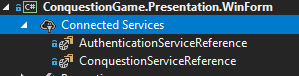
#### Code Example



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

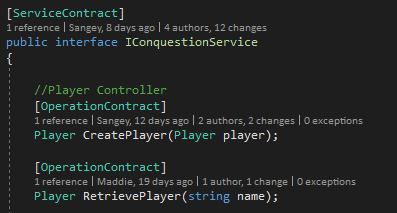
WCF allows for proxies classes to allow for the client applications to communicate with the service while at the same time encapsulating some of the service information. WCF offers the tool *Add Service Reference* which allows for auto generation of a service reference proxy class.



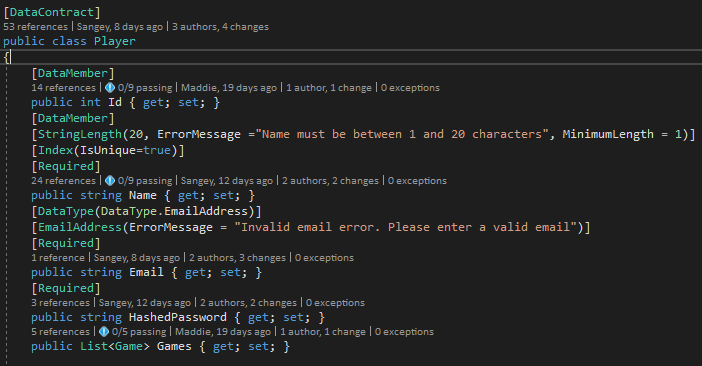
The above image shows two reference proxy classes added to our dedicated client in our project.

#### 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 service operation contracts, data contracts and the component DataContractSerializer.



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.



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.

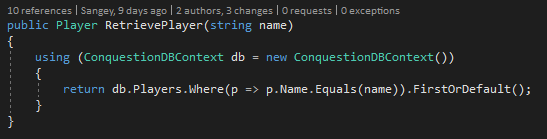
The .Net Framework Data Providers are responsible for connection to the database, and issuing database commands to retrieve or modify data within the database.

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.

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



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

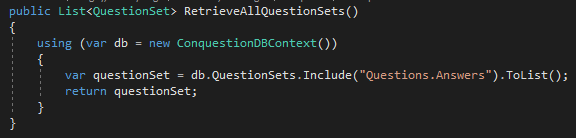
In code first approach you create your domain model classes and generating a database based on these domain model classes. This means very little to no manual control of the database creation using scripts is needed in this approach, entity framework will generate and modify the database as your domain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

#### 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.

#### 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.



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.

References

<https://www.w3schools.com/xml/xml_soap.asp>

<http://www.c-sharpcorner.com/uploadfile/pjthesedays/wcf-service-binding-explained/>

https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/specifying-data-transfer-in-service-contracts