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

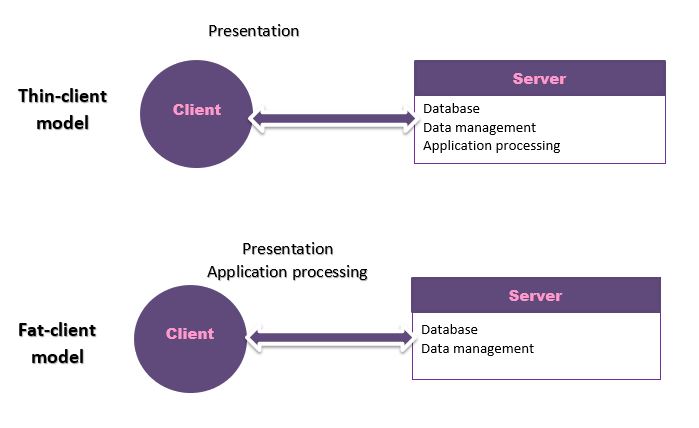
When designing distributed systems selecting an appropriate architecture is important. The choice of architecture will determine how easy it is to maintain and scale your system in the future as well as isolating sensitive information.

We take a 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. N-tier / Multi-tier 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.

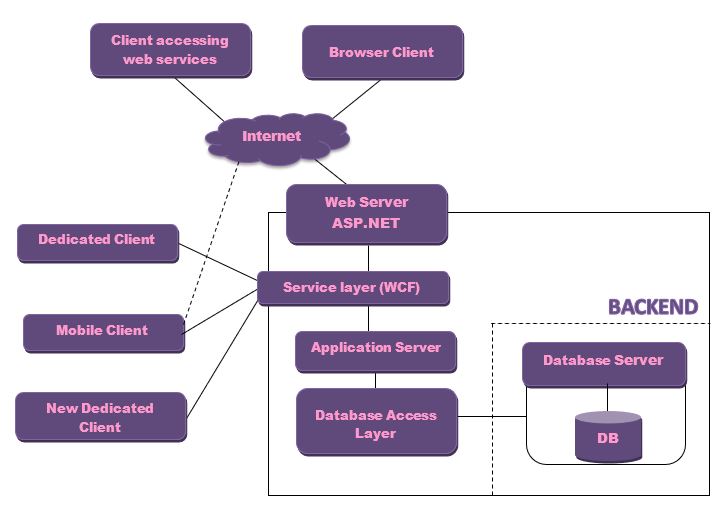
However, as we want to support web clients as well, this example is not suitable for our system.

### 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).

### N-tier / Multi-tier Service Orientated architecture



References