National College of Ireland

MSc Web Technologies

September 2011

Name: Jonathan McCarthy

Student Number: 10205381

Email: jonathan.mccarthy@ncirl.ie

Investigation of the Effect Design Patterns have on Consumption Metrics for an Existing ASP.NET Enterprise Application running on the Azure Platform

I hereby certify that this material, which I now submit for assessment of the programme of study leading to the award of Master of Science in Web Technologies is entirely my own work and has not been taken from the work of others save and to the extent that such work has been citied and acknowledged within the text of my work.

Signed: [illegible]

Date: 14th September 2011

Student Number: 10205381

Abstract

The popularity of Cloud Computing is growing steadily, allowing businesses and organisations to host their applications and servers in the cloud on a pay-per-use model. With pay-per-use metrics on instances, throughput, CPU usage, database transactions and database size the need for a fine tuned application is evident. For a business to utilise the cloud platform effectively, it needs its enterprise application to run as efficiently as possible without impacting users' experience. If an application is designed with this in mind is it possible to reduce consumption saving the business money?

The body of this research will create an enterprise application using techniques and best practices used in industry today. This application will be used as a benchmark to show how an existing site moved to the Windows Azure Platform would perform. With the enterprise application running on Azure, it will be completely re-factored using Fowler's enterprise design patterns. The application will keep its existing look and feel, images, and content exactly the same to ensure the accuracy of the test. Performance will not be sought through reducing HTML, CSS or any other UI components. The primary goal of this research is to measure the effect re-factoring an application using design patterns has on cloud resource consumption. The patterns will be researched to find solutions to identified problems in the original application. Fowler's enterprise design patterns will be researched and implemented to try minimise the applications consumption of cloud resources.

The two applications will be migrated to the Azure Platform. This will be documented in detail showing how existing enterprise applications can be migrated to the Azure Platform. To measure the relevant performance of both applications, a testing plan will be created. This will create a load test for the applications and record their usage of the Azure pay-per-use metrics. The results will be tabularised and graphed to show if the use of design patterns reduced the consumption of the relevant cloud resources.

Acknowledgments

I would like to express my sincere thanks to my supervisor, Michael Bradford, for the continual support, guidance and advice provided throughout the project. Your dedication to the project inspired me to take my work to new levels. Michael, I could not have wished for a better supervisor, thanks for everything.

I would like to extend my appreciation to all the staff from the School of Computing. The help and support I received throughout the completion of my Masters was very much appreciated.

A special thanks to all staff at NCI, your helpfulness and support in all areas was greatly appreciated.

To my parents, John & Patricia McCarthy, thank you for the love and support you give me throughout my life.

Table of Contents

Table of Figures

Index of Tables

1. Introduction

1.1. Overview

The popularity of Cloud Computing is growing steadily, allowing businesses and organisations host their applications and servers in the cloud on a pay-per-use model. This is proving popular as it eliminates the initial up front costs of investing heavily on servers, software and equipment needed to run the IT infrastructure. A vast majority of businesses and organisations are providing customers access to their products and services through their websites. With the cloud operating on a pay-per-use model, the more web traffic a website receives, the more the company will have to pay their cloud service provider. There are a number of different factors and combinations that will affect this and these will be discussed in detail in Chapter 2 and Chapter 3.

Considering this usage model and the way software development has now enabled the quick creation of applications using drag and drop smart UI anti-patterns, the need has arisen to create applications that minimise resource consumption. There are numerous ways to fine-tune applications to run more efficiently making good use of software design principles and coding practices. This body of work will investigate the effect design patterns have on consumption metrics for an existing ASP.NET enterprise application running on the Azure Platform.

1.2. Research Problem

With cloud computing growing increasingly popular the need to properly architect an application solution is very evident. As cloud service providers charge on a pay-per-use model, the more efficient an application runs the more cost effective it will be for the business. As technologies advanced the need to properly architect an application was not always a prominent requirement, with servers having ample processing power and storage to run the application. The introduction of frameworks and superior development environments resulted in a lot of applications created using agile methods with the drag and drop smart UI anti-pattern. This development technique creates applications quickly with the developer not required to understand every piece of the application and functionality. There can be a large amount of code replication and little or no separation of concerns. With the cloud pay-per-use model there are potential savings to be made by minimising usage and structuring applications in adherence to architectural principles.

Design patterns are used as solutions to commonly recurring problems, high level blueprints for solutions rather than solutions themselves (Millett 2010). The scope of this work will research the effect re-factoring an existing ASP.NET application using recognised enterprise design patterns.

1.3. Intellectual Challenge

To test the effect design patterns will have on cloud consumption metrics, an original baseline application will be created. This application will follow industry standards, creating an application in an efficient manner to solve a business problem. The applications will be coded using ASP.NET C# and will be hosted on the Windows Azure Platform when completed. The aim is to provide an application that is capable of serving specific business needs for both the company and its customers. The application will be designed and implemented to a high quality using existing technologies and techniques to meet the business needs. The application will allow customers to view a product catalogue, add items to a shopping cart and complete the purchase order. The theme of an online golf store will be used for the enterprise application. To provide an easy reference this original application will be addressed as the legacy application for this body of work. The specific detail of the requirements and implementation will be discussed in Chapter 4 and Chapter 5.

With the original enterprise application created the process of refactoring can begin. For this project the patterns described in Martin Fowler's book 'Patterns of Enterprise Application Architecture' (Fowler 2003b) will be used. It must be noted that there are numerous authors and books written on the subject of design patterns. Fowler's patterns for enterprise applications are relevant to the application to be re-factored and for this reason the patterns in this book will be researched to find design patterns to reduce resource consumption for our legacy application. Fowler's book includes 51 design patterns (Fowler 2003b), and the requirements of the application will be taken into consideration when identifying possible patterns for use in the re-factored application. It must be noted that the quality of the re-factored application is not directly proportional to the number of design patterns in it. Fowler wrote a journal for the IEEE giving a good analogy of "...the apocryphal story of a hello world application with three decorators, two strategies, and a singleton in a pear tree" (Fowler 2003a). He states that patterns are not good or bad, they either serve a specific purpose for the application or do not. The design patterns chosen for the re-factoring will be discussed in Chapter 6.

When the legacy application and the re-factored application are complete they will be hosted on the Azure Cloud Platform (Windows Azure Platform 2011). An account will be created and the two applications will be migrated. There are a number of different subscription offers available on Azure, for the purposes of this project the pay as you go subscription will be used as the basis for measuring resource consumption used. Appendix D will detail the steps involved in getting an existing application running on the Azure Cloud.

With the two applications running on the Azure Cloud it will be necessary to test the performance and metric consumption. The following metrics will be used to measure the efficiency of the hosted applications (Azure Pricing 2011):

- Throughput
- CPU Usage – Compute Instances
- DB Transactions
- Storage

A testing plan will be created to provide a standardised mechanism to evaluate the performance of the applications. This will use a load test to simulate a high volume of traffic for the particular site and then record specific metrics. All data will be tabulated and graphed to give a direct comparison of the two applications performance. A number of different software tools and techniques will be used in the testing plans and will be covered in detail in Chapter 7.

1.4. Research Objectives

This project will first document the creation of an enterprise application for an online golf store. This will follow a standard development life-cycle process taking into consideration all necessary information to create the required application. Content for the site will be simulated to give a realistic look and feel and provide a real world testing base for the comparisons.

The re-factoring of the application will research a number of Fowler's enterprise design patterns with the goal of minimising resource consumption of cloud resources. This process will document the patterns to be used for the application and other patterns that would be of use in minimising usage but not included in the re-factored application.

The process of migrating the applications will be covered in great detail. As cloud computing becomes more popular, businesses will be faced with the decision on whether to invest further in their IT infrastructure or move to the cloud. The steps involved will be documented, showing the changes needed to the applications to get them fully functional on the Azure Platform.

With both applications hosted the testing phase of the project can begin. Load tests will be created to test the areas of the application prone to heavy usage and potential bottle necks and points of failure for the application. The test plan created in Chapter 7 will be used exactly against both applications and a series of metrics recorded. From the results of the testing plan a conclusion will be drawn on the effectiveness of design patterns to minimise resource usage on the Azure Cloud Platform.

1.5. Scope and Limitations

The scope of this research will focus on the effect design patterns will have cloud resource consumption. There are a number of different techniques used to improve efficiency and performance of applications, but these will not be used in this project. An existing enterprise application will be re-factored using Fowler's enterprise design patterns. The applications created will not be complete due to the time constraints with the lifeline of the dissertation. A subset of the application will be implemented based on the section receiving the heaviest traffic.

The data used to compare the effectiveness of the two applications will be taken from the Azure Platform. Additional metrics could have been recorded across the different sections or layers of the applications, but this research is only interested in the cloud resources consumed. These factors will be discussed extensively in the conclusion of this research.

1.6. Organisation of the Dissertation

The remainder of this dissertation will describe the research and evaluation. Chapter 2 provides a brief overview of cloud computing. There are a few myths to what cloud computing really is and this will be addressed in the following chapter. In Chapter 3 the research background will be discussed, explaining design patterns and Fowler's contribution to this knowledge area. The Windows Azure Platform will be discussed detailing all of the main components and functionality on offer. The requirements analysis for the enterprise application will be detailed in Chapter 4. This will define the problem, project scope and functional requirements for the application. UML diagrams will be used to model the systems requirements. Chapter 5 explains the main components used in the design and implementation. The re-factor of the application will be discussed in Chapter 6. This will identify where design patterns can be implemented to help reduce resource consumption on the Azure Platform. The patterns will be explained in detail showing their benefit to the application. Chapter 6 also discusses the re-factor implementation listing the main components used in the application. Appendix D outlines the main issues to consider before migrating an application to the Azure Platform. A comprehensive guide will be created to aid a migration to the Azure Cloud, detailing the exact steps. The evaluation and analysis of the research is covered in Chapter 7. This will evaluate all the data and results from the application testing. The conclusion will be discussed in Chapter 8, discussing the findings of the research and a result on the effect design patterns have on resource consumption for the Azure Platform.

2. Literature Review

2.1. *What is cloud computing?*

Cloud computing has gained in popularity over the past few years. There are many different definitions for the term *'Cloud Computing'* and these will be discussed to explain what the term really means. This chapter will explain cloud computing in detail by exploring its origins and detailing the different characteristics and models the cloud has to offer.

(Mell & Grance 2009) wrote a paper for the National Institute of Standards and Technology (NIST 2011) and defined cloud computing as *"... a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."* There have been numerous different definitions causing ambiguity as to what cloud computing really is, but this definition contains the core elements of the cloud.

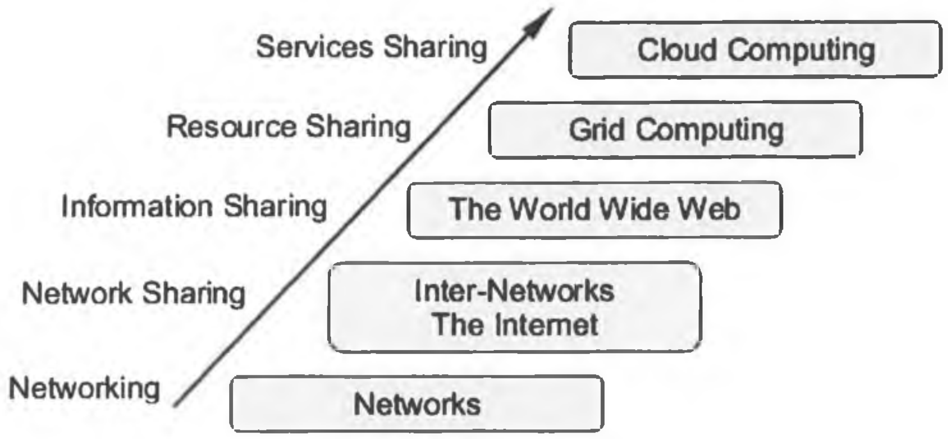


Illustration 1: Evolution of sharing on the cloud (Bojanova & Samba 2011)

Illustration 1 shows the evolution of resource sharing on the cloud (Mell & Grance 2009). The NIST definition of the cloud, its characteristics and models have been cited in a large number of papers and is considered to be the working definition for cloud computing. (Bojanova & Samba 2011) makes reference to the NIST paper by (Mell & Grance 2009) offering further detail on their characteristics of a cloud and the different types of service models. These characteristics and models will be investigated later in section 2.4.

Cloud computing has become a scalable services consumption and delivery platform, allowing different consumers share resources (L. J. Zhang et al. 2010). Sharing resources can occur at different levels, which led to three main types of cloud service models and they will be discussed in detail later in section 2.5.

2.2. Benefits of Cloud Computing

Cloud computing allows consumers access applications that reside at a location other than their own computer (T. Velte et al. 2009). Data centers are the main method of providing resources on the cloud. The main benefit of cloud computing is that it allows users to pay for only the resources they consume. There is no need for an up front investment in hardware as per the typical model. (Barr 2010) gives an excellent example showing how the cloud can provide resources to a new business venture. The venture is for an on-line business which anticipates growth for the first six months. To follow the traditional approach, based on growth predictions a server would be purchased to host the application. The server will never want to run out of capacity and to match growth figures a new server will be purchased each month for the first six months. Timing will be of the essence to ensure the new servers are received, installed and configured in a timely fashion. For the business venture capacity is vital, but getting the balance wrong will result in spending money to support users that may or may not visit the site (Barr 2010). The other problem could be too many users visit the site due to an over effective marketing campaign, the site becomes overloaded, slow and users leave unsatisfied. From the traditional sense making predictions about web traffic is very difficult and any errors made can be very costly.

Cloud computing is the solution to this problem. It allows for applications to be hosted in the cloud and only pay for the resources consumed. In the above example, if the application was hosted in the cloud, there would be no problem scaling to provide for all the extra traffic the site was getting. Using the cloud eliminates the start-up costs for hardware investment, the time and resources of commissioning the new servers and the ability to scale for any load the new site would encounter. This introduction is very high level and the components of a cloud architecture will now be discussed in greater detail.

2.3. Cloud Components

To describe the cloud from a topological view, it is comprised of clients, datacenters, and distributed servers. Using these components the cloud can be viewed as "... a coherent, large-scale, publicly accessible collection of compute, storage, and networking resources" (Barr 2010). The services on offer may typically be available via web service calls. Components of the cloud architecture are typically categorised into front-end or back-end. The front-end components will be the devices used by the client to access the cloud via the web. The back-end components will contain all the necessary infrastructure to provide all the cloud services.

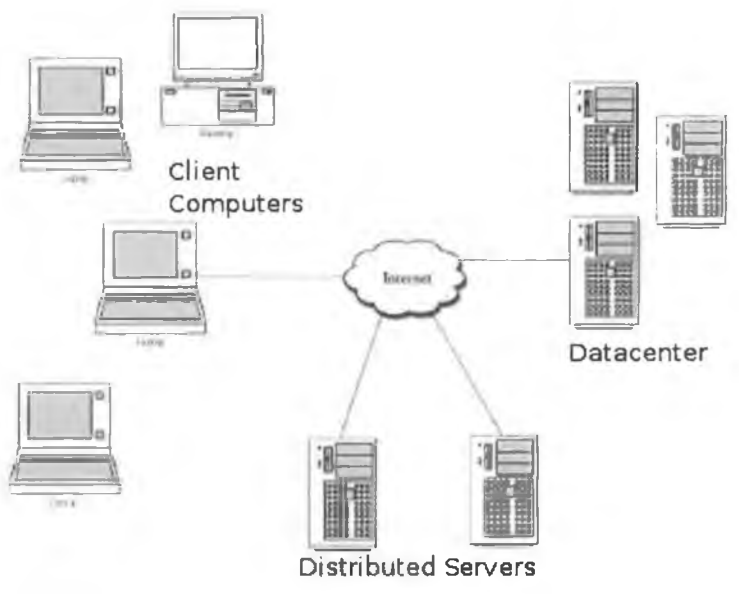


Illustration 2: Cloud Components

Illustration 2 shows a diagram of the general components that will make up a client computing solution. Contained within this are networks, servers, storage, applications, and services. The goal of the cloud is to abstract these away from the users of the cloud, and just provide them with access to the services they need.

2.3.1 Clients

In the cloud computing architecture, client machines serve the same role as they did in a typical Local Area Network (T. Velte et al. 2009). Technology has advanced over the past number of years, allowing users access the internet using laptops, thin clients, and mobile phones (Wang et al. 2008). Any device that can browse the web can be used as a client for cloud computing.

2.3.2 Datacenter

The datacenter is a large collection of servers where the applications and services provided by the cloud are located (Armbrust et al. 2009). The datacenter will make use of server virtualisation to allow a physical server run multiple instances of virtual servers (T. Velte et al. 2009). This is a core principle of the cloud model, allowing services to scale by adding virtual server instances to balance a workload.

2.3.3 Distributed Servers

In the cloud model server distribution is used to maximise availability and uptime by replicating server instances and data among all its servers located in different geographic areas. If there is a fault in a specific region the cloud provider will be capable of providing the service or application from a different region without any disruption to the users of the service.

2.4. *Cloud Characteristics*

The Cloud model is composed of five essential characteristics (Mell & Grance 2009). These characteristics will be common to all cloud architectures and they help define what cloud computing really is. The following is a brief description of the main cloud characteristics.

2.4.1 On-Demand Service

A consumer on the cloud will appear to have access to infinite resources, e.g. server time and network storage. They have the ability to use these resources and provision for their usage without needing human interaction with the service provider. Access to resources is near instantaneous allowing consumers increase or decrease their computing capabilities as their demands dictate.

2.4.2 Broad Network Access

For cloud computing all resources are available over the Internet. Services will be accessed and managed through thin clients, thick clients, laptops, mobile phones and PDA's. All that is needed is a browser and Internet access.

2.4.3 Resource Pooling

Providers of cloud resources will have a multitude of servers and storage to meet their consumers' needs. In this model resources are pooled to serve multiple consumers, and they can share or occupy the same server or storage areas. This is known as multi-tenancy. The management application will dynamically assign different physical and virtual resources according to consumer demands. The consumer has no knowledge of where the resource is coming from (i.e. geographic location) or any of the other consumers sharing the same resources. Examples of resources include storage, processing, memory, network bandwidth, and virtual machines (Mell & Grance 2009).

2.4.4 Rapid Elasticity

Rapid elasticity provides scalability to a cloud application. The process of adding or reducing resources is usually automatic, allowing an application scale to meet the demand or load it has to deal with. To the consumer the amount of resources will appear to be unlimited and can be accessed as required.

2.4.5 Measured Service

Cloud systems have the ability to automatically manage resource use. It can optimise the resources based on metering for the particular service. Resource usage can be monitored by the consumer and provider, providing consumers with the ability to monitor and control the usage as required.

2.4.6 Business Characteristics

With the introduction of the cloud model some characteristics have changed the way businesses operate. The most obvious is that fixed costs now become variable, eliminating an initial up front investment in hardware or lengthy service contracts (Barr 2010). The cloud allows for resource consumption to change in real time in a pay-as-you-go model. The business will also gain flexibility with the ability to scale resources if there is a demand or a change in the business structure. With all services provided in the cloud, the business can focus on the development of their product or service and not spend their time on IT and support issues.

2.5. Cloud Service Models

"The term services in cloud computing is the concept of being able to use reusable, fine-grained components across a vendors network" (T. Velte et al. 2009). There are different models and they are known as '... as a service'. There are a number of common traits shares by services. They are scalable with a low entry barrier. Certain deployment models usually involve resources shared by

many users (multi-tenancy). There are a number of different deployment models and they are mainly categorised as private cloud, public cloud, community cloud, and hybrid cloud (Kundra 2011). The names are somewhat self explanatory. A private cloud is operated for a specific business and not shared with any third-parties. A public cloud is made available to the public and is owned by an organization selling cloud services. A community cloud is shared between a number of similar organisation with common interests and goals. The hybrid cloud is a mixture of both a public and private clouds used by an organisation to best suit their requirements and needs (Kundra 2011). The following subsections will explore the three main service offerings provided on the cloud today.

2.5.1 Software as a Service (SaaS)

Software as a Service (SaaS) hosts an application on the cloud for customers to access it via the Internet (Armbrust et al. 2009). With this model the hardware and software is hosted offsite by the cloud provider, the consumer will have no need to maintain or support it. All of this is taken care of by the provider. This is ideal for both the application provider and the consumer (Kundra 2011). In a traditional model the application provider may have to support the customer in installing the application on their hardware (servers etc) and there are a wide variety of variables that can effect the operation of the application. Different versions of operating systems, web servers, platforms, language versions, patches etc can impact the application in different ways which may lead to software tweaks to get the application running in a particular environment. For the provider version management and upgrades can become unmanageable. This will also have an impact on testing and support issues as it may be difficult to replicate different system environments (Barr 2010).

For the customer SaaS works straight out of the box. There are minimal configuration and maintenance issues. With the SaaS model comes a new pricing structure for the consumer. Costs for SaaS are continual, the more you use it, the more you pay! This differs from the standard up-front one off cost of purchasing the software. SaaS will transform the acquisition and delivery of enterprise business applications from a procurement and ownership model to a subscription and outsourced services model (online-crm.com 2011).

2.5.2 Platform as a Service (PaaS)

Platform as a Service (PaaS) is an application delivery model, providing all the resources required to build applications and services completely on the Internet. PaaS services include application design, development, testing, deployment, and hosting. One draw-back for PaaS is the lack of interoperability and portability among providers. It can be quite difficult and costly to move providers (T. Velte et al. 2009).

PaaS allows businesses to develop and deploy applications without having to make an up front investment in hardware and software (Barr 2010). All of the infrastructure required to develop the application is available on the providers cloud. This may include testing, deployment, hosting, security, storage etc. PaaS is closely linked to SaaS, with the developed application on PaaS being offered to customers as SaaS.

2.5.3 Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) provides hardware to the consumer so they can do what they want with it! From a server perspective, the consumer will have full control over the server instance and can install software and configure it any way they choose. This model provides dynamic scalability and operates as a utility computing billing paying for the resources used (T. Velte et al. 2009).

2.6. Resource Consumption

As described earlier in this chapter cloud computing is a pay-per-use model. With most businesses and organisations currently considering if a move to the cloud is economically viable, resource consumption can play an important role in making this decision for the business in question. Resource consumption is based on the amount of services and data used by the application on the cloud platform. As this research is using the public Azure Cloud Platform the pricing model for Azure will be used to explain resource consumption. Appendix D lists all the standard rates for service consumption on the Azure Platform. It is broken up into four distinct categories, Windows Azure, SQL Azure, AppFabric, and Data Transfers. Resource consumption is discussed from the Azure perspective in Chapter 3.

2.7. Design Patterns

A design pattern offers a solution to software design problems that occur frequently in real world application development (dofactory.com 2011). The origin of design patterns is attributed to Christopher Alexander, who worked as an architect in the field of building construction and introduced the concept of design patterns as a common vocabulary for design discussions. Alexander described a pattern as a problem that frequently occurs which has an associated core solution to this problem. This solution can be reused every time this problem occurs, and may deviate slightly depending on the problem being solved.

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core solution to that problem, in such a way that you can use the solution a million times over, without ever doing it the same way twice" (Alexander et al. 1977).

Even though Alexander's discussions on patterns were from a specific architectural perspective, it is completely relevant for object orientated design patterns (Gamma et al. 1995). The Gang of Four have built on Alexander's concepts and show that a pattern is a solution to a problem in a specific context.

2.8. Makeup of a Design Pattern

The Gang of Four have specified in great detail the makeup of a design pattern (Gamma et al. 1995). From a general perspective, a pattern has four essential elements:

- Pattern Name
- Problem definition
- Solution
- Consequences

The pattern name is a self explanatory title used to describe the design problem. The naming of a pattern is important, its name must describe the problem, solution, and consequences in a word or two (Gamma et al. 1995). The pattern name allows developers discuss patterns by name, without having to specifically talk about the specifics of an implementation. The problem definition describes when the pattern should be used. It can describe specific design problems, give class or object structures and offer a list of conditions that must be met before it makes sense to apply the pattern. The solution describes the elements that make up the design. This will offer information on the relationships and responsibilities the elements may have. The solution is more like a template that can be applied in many different situations. The pattern will give an abstract description of a design problem and a general solution to it (Gamma et al. 1995). The consequences will list the trade-offs of applying the pattern.

The method of describing design patterns has followed the template in the Gang of Four's book 'Design Patterns, Elements of Reusable Object-Oriented Software'. Although this book was first published in 1995, the concepts described in it still hold true today.

2.9. Fowlers Patterns for Enterprise Design

Fowler's book 'Patterns of Enterprise Application Architecture' specifically deals with the design and implementation of enterprise applications. Fowler describes them as "...enterprise applications dealing with the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data" (Fowler 2003b). Examples of enterprise applications include reservation systems, financial systems, online stores and other systems used to run a particular business. Fowler uses design patterns to aid in the architectural issues in building enterprise applications and allow developers communicate more effectively on the development process.

Enterprise applications deal with a large volume of persistent data with concurrent access. It will contain a lot of user interfaces and may be required to integrate with other enterprise applications. Enterprise applications will contain complex business logic which must be catered for to allow the day to day operations of the business.

The patterns detailed in Fowlers book will be researched in Chapter 6 when the legacy enterprise golf store application is re-factored before its migration to the Azure Platform.

2.10. Conclusion

This chapter has given a brief overview of cloud computing. The definition of cloud computing was discussed with a detailed explanation of what cloud computing really is. The benefits of using the cloud model from a business perspective was analysed showing the factors that must be considered for a migration to the cloud. The main components of a cloud architecture were examined along with the five essential cloud characteristics. Resource consumption was introduced and this will be discussed in detail in Chapter 3.

Design patterns were introduced giving a brief history of their origins. Fowler's patterns for enterprise design were introduced showing how Fowler's book will be researched to re-factor the legacy enterprise application.

The next chapter will discuss the research background expanding on some of the topics in this chapter. Windows Azure components and features will be examined along with Fowler's approach to enterprise design patterns.

3. Research Background

3.1. Introduction

This chapter will explain the research question and the main components involved in this research area. This project's main goal is to investigate the effect design patterns have on consumption metrics for an existing ASP.NET Enterprise Application running on the Azure Platform. To do this the Azure Platform will be discussed in detail, explaining the architecture and deployment models. The benefits of hosting an application on Azure will be discussed, showing the cost implications for certain setups. The re-factor of the original enterprise application will be researched, identifying design patterns and architectures to help minimise resource consumption. The creation of a testing plan will be researched, to simulate a realistic load for the enterprise applications.

3.2. Problem Definition

The popularity of cloud computing is increasing, allowing businesses eliminate large investment in hardware infrastructure. Cloud computing operates a pay-per-use model, charging only for the resources consumed. For a business moving their enterprise application to the cloud, the implementation of their existing application may not be best suited this environment. Current ASP.NET applications are developed to run on Windows Server 2008 and the only considerations are application performance on both the client and server side. An enterprise application may be well designed, but there are many design patterns that are not typically implemented in a solution. This may be due to time constraints or the development team's unfamiliarity with the patterns. This research will re-factor an existing enterprise application to improve the system architecture and implement patterns to introduce proper design principles. The two applications will be migrated to the Azure Platform and their performance will be tested to measure their resource consumption.

3.3. *Windows Azure Platform*

3.3.1 What is Windows Azure?

"The Windows Azure platform is an Internet-scale computing and services platform hosted in Microsoft data centers. The Windows Azure platform includes the foundation layer of Windows Azure as well as a set of developer services which can be used individually or together"(MSDN 2011). Illustration 3 shows the products and components that are part of the Windows Azure platform.

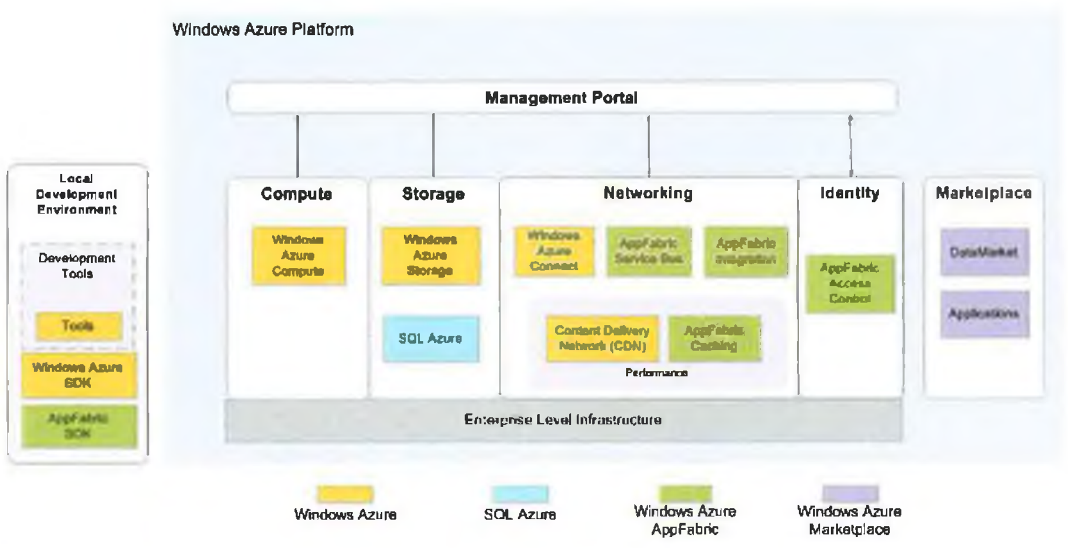


Illustration 3: Components of the Windows Azure Platform

3.3.2 Local Development Environment

The Windows Azure SDK (Azure SDK 2011) gives developers the ability to create applications locally before they are deployed to the cloud. This is an important feature as any debugging can be done locally. The SDK integrates seamlessly with Visual Studio IDE and simulates storage and compute instances, allowing developers a realistic local development environment. The Azure SDK also allows for the deployment of applications directly to the cloud, with authentication to the management portal to automise the entire deployment process (MSDN 2011).

3.3.3 Windows Azure Compute

Windows Azure Compute provides applications with three types of roles to perform different tasks. These are web roles, worker roles, and VM roles. A web role is a VM that hosts your application within IIS. A worker role is the same as a web role, but without IIS. It's intended for typical back-end processing workloads. A VM role enables you to define the configuration and updates of the operating system for the virtual machine. While a web role and a worker role run in a virtual machine, the VM role is the virtual machine, which gives you full control of operations (MSDN 2011).

For an Azure application, it must use at least one type of role. Azure uses Virtual Machines to achieve separation of services across physical servers (Hay & Prince 2010). The VM is a base installation of Windows Server 2008 with some minor modifications. Every instance of a service is installed onto a separate VM. Even though the service is running on a VM, this is abstracted away and the role instance is only visible.

3.3.4 Microsoft SQL Azure

"Microsoft SQL Azure Database is a cloud-based relational database service that is built on SQL Server technologies and runs in Microsoft data centers on hardware that is owned, hosted, and maintained by Microsoft" (Microsoft MSDN 2011b). It operates the same as a standard SQL Server instance. Applications require no changes to use SQL Azure. As SQL Azure is a cloud database, Microsoft is responsible for the physical administration. Access to the database is managed through the Azure Management Portal (Windows Azure Platform 2011).

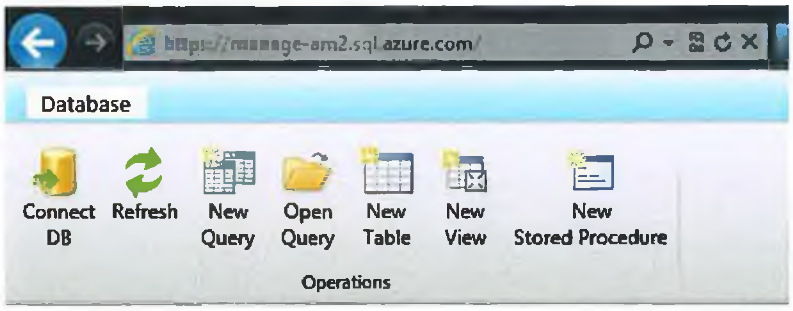
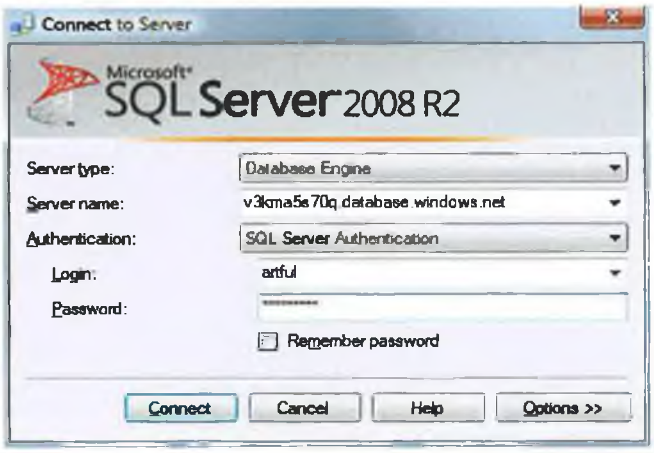


Illustration 4: SQL Azure Management Portal

Illustration 4 shows the main features of the SQL Azure Management Portal. This gives the interface to create tables, run queries and perform all the general operations associated with managing database content. Management if the SQL Azure database is also available through SQL Server Management Studio.



Server type: Database Engine
Server name: v3kma5e70q.database.windows.net
Authentication: SQL Server Authentication
Login: artful
Password: **********
Remember password

Illustration 5: SQL Server Management Studio Authentication

This allows the database administrator interact with the databases, content, security and logins, and other management issues. Connecting to the database using the SQL Server Management Studio is a great asset, as moving a new or existing application to the Azure Platform will require the setup of a new database. A new or existing database can be converted into script format and used to create the new database on Azure. With SQL Azure it is very easy to create and manage relational database solutions. The key benefits of SQL Azure include manageability, high availability, scalability, a familiar development model, and a relational data model (Microsoft MSDN 2011a).

3.3.5 Windows Azure Management Portal

The Windows Azure Platform Management Portal provides access to service deployment and management tasks as well as at-a-glance status information that lets you know the overall health of your deployments and accounts (Microsoft MSDN 2011b). The Management Portal organizes the components of the Windows Azure deployments with constantly refreshed information that's easy to discover and understand.

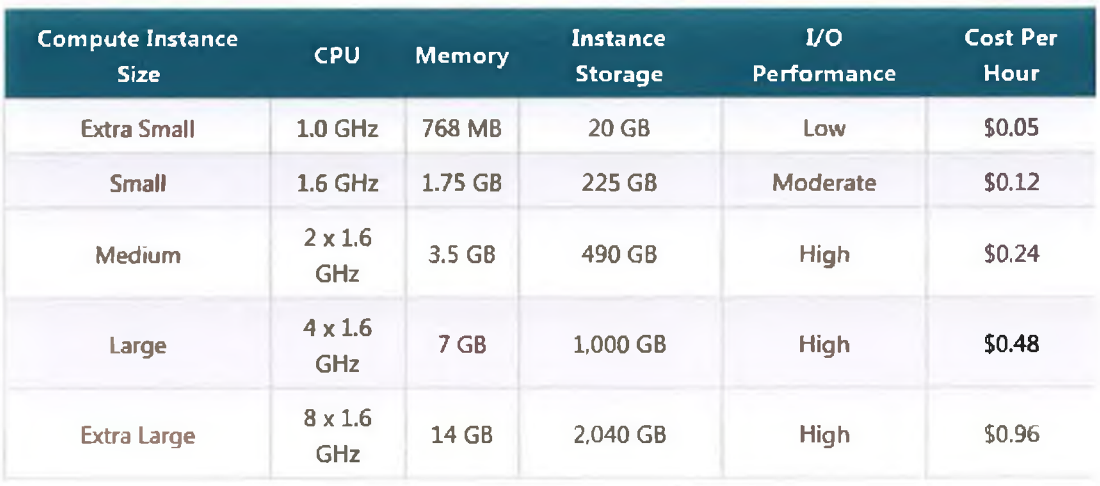
3.4. Benefits of hosting in the Cloud

Hosting an application on the cloud eliminates the concern of how much disk space, RAM and processing power will be needed for the application to function (Barr 2010). In the traditional server model careful planning is required to anticipate future growth for the application. Using cloud services allows an application to scale as required. If the site is under a substantial load additional instances can be spawned, providing a consistent service for all users. The main advantage of this is that services can be scaled back when the demand on the application decreases. With costs based on consumption the cost of the cloud hosting should be considerably lower than a hardware investment. With the cloud model additional resources can be added and removed as needed (T. Velte et al. 2009).

3.5. Azure Pricing

The Windows Azure pricing model charges for a number of different services which are divided into four distinct categories. The categories are Windows Azure, SQL Azure, AppFabric, and Data Transfers and a comprehensive list is all services and their associated prices are listed in Appendix E. For this research the pricing for the Windows Azure Compute Instances, SQL Azure and Data Transfers will be used.

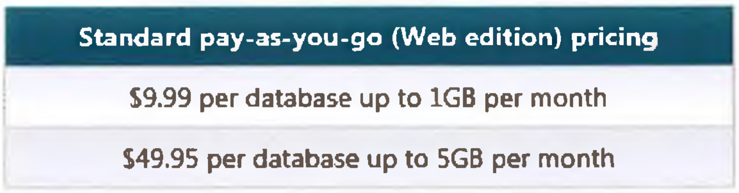
Compute Instances are divided into five different options each offering different levels of performance to the application. The process of hosting an application on the Azure Platform must be carefully considered, as the cost implications for getting it wrong can be high. Illustration 6 shows the Azure pricing guide for compute instances. These range from extra small to extra large. To run an application on a single extra small instance for one year would cost $438. To run the application on a single extra large instance for one year would cost $8409.60 (Azure Pricing 2011). There is a considerable difference in the high and low ends of the pricing spectrum. For a business it is imperative they select the compute instance size to best meet their business needs. There is no point hosting the application on an extra large instance if the application is only going to use a small fraction of the resources on a daily basis. With the cloud model scalability is the key, adding additional resources as required. To maximise performance the application should make full use of an instance to deal with an average daily load. If additional traffic is received Azure can add more compute instances to deal with this load. This will help maximise resource usage from a cost perspective.



Compute Instance Size	CPU	Memory	Instance Storage	I/O Performance	Cost Per Hour
Extra Small	1.0 GHz	768 MB	20 GB	Low	$0.05
Small	1.6 GHz	1.75 GB	225 GB	Moderate	$0.12
Medium	2 x 1.6 GHz	3.5 GB	490 GB	High	$0.24
Large	4 x 1.6 GHz	7 GB	1,000 GB	High	$0.48
Extra Large	8 x 1.6 GHz	14 GB	2,040 GB	High	$0.96

Illustration 6: Windows Azure Compute Instances

SQL Azure pricing used two different pricing plans, web and business editions. The charges are per database and set a limit of the amount of storage. Illustration 7 shows the two pricing plans for the web edition.



Standard pay-as-you-go (Web edition) pricing
$9.99 per database up to 1GB per month
$49.95 per database up to 5GB per month

Illustration 7: SQL Azure Price Plan (Web Edition)

This research will use the web edition of SQL Azure. The size of the database and its content is minimal and the cost implications of running the business edition is too high. The web edition of SQL Azure will be capable of serving both enterprise applications.

The final pricing metric to be discussed is data transfers. Data transfer metrics measures throughput between the application and other services. The amount of data being sent and received by the application is billable on the Azure Platform. The charges for data transfers are minimal and are listed in Appendix E. Data transfers between the application and SQL Azure cost significantly more than regular data transfers. If SQL Azure provides the application with a large volume of data, it will be added to the SQL Azure usage charges. For an enterprise application a high volume of data will be sent and received from the database. This metric will be carefully observed in the testing and analysis phase.

3.6. Design Patterns and Architecture

The development of both applications will use good system architecture with some of the most popular design patterns used in industry today. The legacy application will use a system architecture and design patterns to develop an enterprise application as quickly as possible meeting all of the functional requirements. The re-factored application will concentrate on the best architectural solution using Fowler's enterprise design patterns to deliver the required functionality. The design and architecture of both applications will be detailed later in this research.

Fowler's research identifies six architectural issues to be considered when building enterprise applications (Fowler 2003b). These are listed in Table 1.

Layering of enterprise applications
Structuring domain (business) logic
Structuring a web user interface
Linking in-memory modules (particularly objects) to a relational database
Handling session state in stateless environments
Principles of distribution

Table 1: Fowler's Architectural Issues

Enterprise applications usually involve persistent data which will contain a lot of data. Access to this data will be concurrent, with a large multitude of users accessing the system at the same time. A user interface will be required to allow users interact with the system. Business logic will contain the business rules of the system, detailing how certain business processes will operate. Fowler has discussed these issues in his definition of what an enterprise application is.

The development of the legacy application will consider Fowler's concepts as listed above, but the primary goal is to deliver a functional application in a timely fashion. Chapter 4 and 5 will discuss the design and implementation of the legacy application.

The re-factored application will deliver the functionality required using the best design and architecture following Fowler's principles. Time is not a factor in this development and the main goal is a perfectly designed system. Chapter 6 will explain the implementation and design patterns used in the development of the re-factored application.

3.7. Area of Proposed Contribution

With the increasing popularity of Cloud Computing businesses will consider the possibility of moving to the cloud. For a current business with an existing enterprise application serving customers needs the design of the application may not run efficiently on a cloud environment. With cloud resources available on a pay-per-use model an efficiently designed application will help minimise costs when using the cloud model.

This research will identify if re-factoring an existing enterprise application using proper architecture and design patterns will produce a more efficient application running on a cloud environment. It will also detail the steps and processes needed to migrate an ASP.NET enterprise application to the Windows Azure Platform. This research will provide an informed process on changing an existing enterprise application to run in a cloud environment and the benefits of changing the existing architecture to minimise resource usage. The steps involved in creating a testing plan will give the process needed to test any ASP.NET applications running on Azure.

3.8. Conclusion

This chapter discussed the research background for this project. The main body of this research area was detailed in the problem definition showing the issues relating to resource consumption on the cloud and how design patterns could be used to try minimise resource usage. The Windows Azure Platform was introduced detailing all of its main components and services. Fowler's research into enterprise application development was discussed, listing the main architectural issues to be considered in creating a new application. The next chapter will introduce the golf store case study and the requirements analysis phase before the system development commences.

4. Enterprise Application Requirements Analysis

4.1. Introduction – Golf Store Case Study

An existing golf business has recently seen a decrease in business via traditional sales channels and needs to reach a new customer base. The owner has decided to create a new online store offering cutting edge golf products. The business owner has sourced a completely new product line with many novelty items. It is thought the most cost effective route to market is an online store.

4.2. Existing problem

The business wants to sell directly to the customer at a very competitive price. For the customer this will offer considerable savings when compared to existing golf stores. The range of stock available in most golf stores is limited. The golf market is large with the potential for novelty sales on Father's Day, Birthdays and Christmas. The new online store will cater for all types of golfers taking into account their needs and price range. Existing customers of competitors' sites have noticed their site has become slow at peak times, which may lead to customers leaving the site and moving to a competitor's site. The new site will be positioned to take business and sales from existing competitors through excellent service and strategic pricing.

4.3. Project Scope

The scope of this project requires the creation of an enterprise online golf store. The new site must be cutting edge, capable of standing out from any existing golf sites and have a professional quality. The online store must embrace the following features:

- Online catalogue with categories and brands
- User friendly for customers
- Shopping cart
- Online order processing

The golf business considered all options and the only alternative is to invest in new hardware or move the site to the cloud. A dedicated server could be purchased and added to a server rack with the current server provider. This would provide the site with a dedicated web server and storage. The main issue with this solution is the up front investment costs for the hardware and software licenses. The additional traffic to the site is seasonal, usually between March and September. Taking this into consideration an additional up front investment in hardware may not be the most cost effective solution. The new server will depreciate within 3 years and a new investment may be required.

The developer maintaining the website suggested that a move to the Windows Azure Platform may be a viable solution. Azure offers customers access to cloud computing as Platform as a Service (PaaS). This will allow the business to deploy applications without having to spend the money to buy the servers on which to house them. The business will only pay for the services used and will allow the application to scale as the demand needs.

After carefully considering both options the business decided to go with the Azure Platform. This decision was based on the pay-per-use usage model and the ability Azure has to deal with increasing demand as required. The existing application will be modified to run on the Azure platform.

4.4. Functional Requirements

4.4.1 Overview

The functional requirements are defined by the stakeholder and the type of online business that will be run. For the online golf store customers must have the ability to browse the catalogue, add items to the shopping cart and complete the transaction with a credit card online payment. From the business administration perspective, staff must be able to manage the catalogue, order processing and customer accounts.

4.4.2 Site Sign-up

The new site will offer customers the opportunity to sign-up to the site and receive email information on upcoming events and special offers.

4.4.3 Product Catalogue

The sites main functionality is the product catalogue. This allows the customers to view all products available for sale on the site. The catalogue will list items by category and brand. The category will be used to group products based on their similarity. Table 2 shows the proposed list of categories for the golf products on the site.

Drivers	Balls
Irons	Ladies Section
Fairway Woods	Junior Golf
Clothing	Putters
Trollies	Rainwear

Table 2: Product Categories

This will allow customers view products based on the categories listed above. Customers interested in a particular brand will be able to view a list of products for a particular brand name.

The initial product list viewed by the customer will only contain a minimal set on information. If the customer requires more detail for a specific product, they can click the details link to view a larger picture of the product with a detailed description of the product and its features. The catalogue is linked with the shopping cart, allowing site customers add items to their cart.

4.4.4 Shopping Cart

The new site will allow online shopping customers to accumulate a list of items for purchase by placing items in the shopping cart. Upon checkout the software will calculate a total for the order, including shipping charges and other charges as applicable.

4.4.5 Functional Requirements for Administrators

- Login
- Manage Accounts (Admin and Customer)
- CRUD functionality for catalogue
- CRUD Special Offers
- Manage Orders

4.4.6 Operational Requirements

- Cross Browser Site
- Professional look and feel
- To be hosted on IIS 7.0,
- Developed using C#

4.5. *Use Case Diagrams*

4.5.1 View Product Items (Product Catalogue)

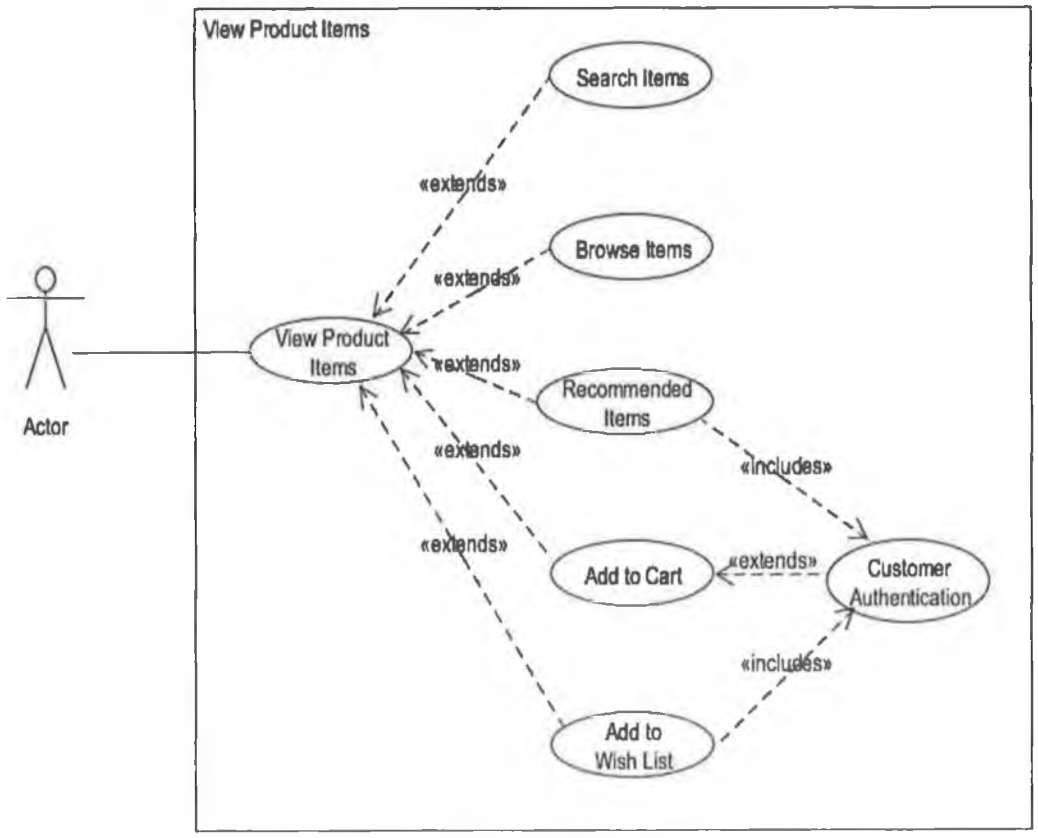


Illustration 8: Use Case - View Product Items

Illustration 8 shows the Use Case diagram for the product items. The product catalogue contains detailed information on the product items for sale on the site.

Customers can search the site for a list of specific items. The option to browse product items can be sorted by both product category and brand name. The product list will first display a summary with the option to view a detailed description if needed. The catalogue contains a link to add the product item to the shopping cart. There is a requirement for the customer to sign in to the site to complete an order and use the shopping cart.

4.5.2 Shopping Cart

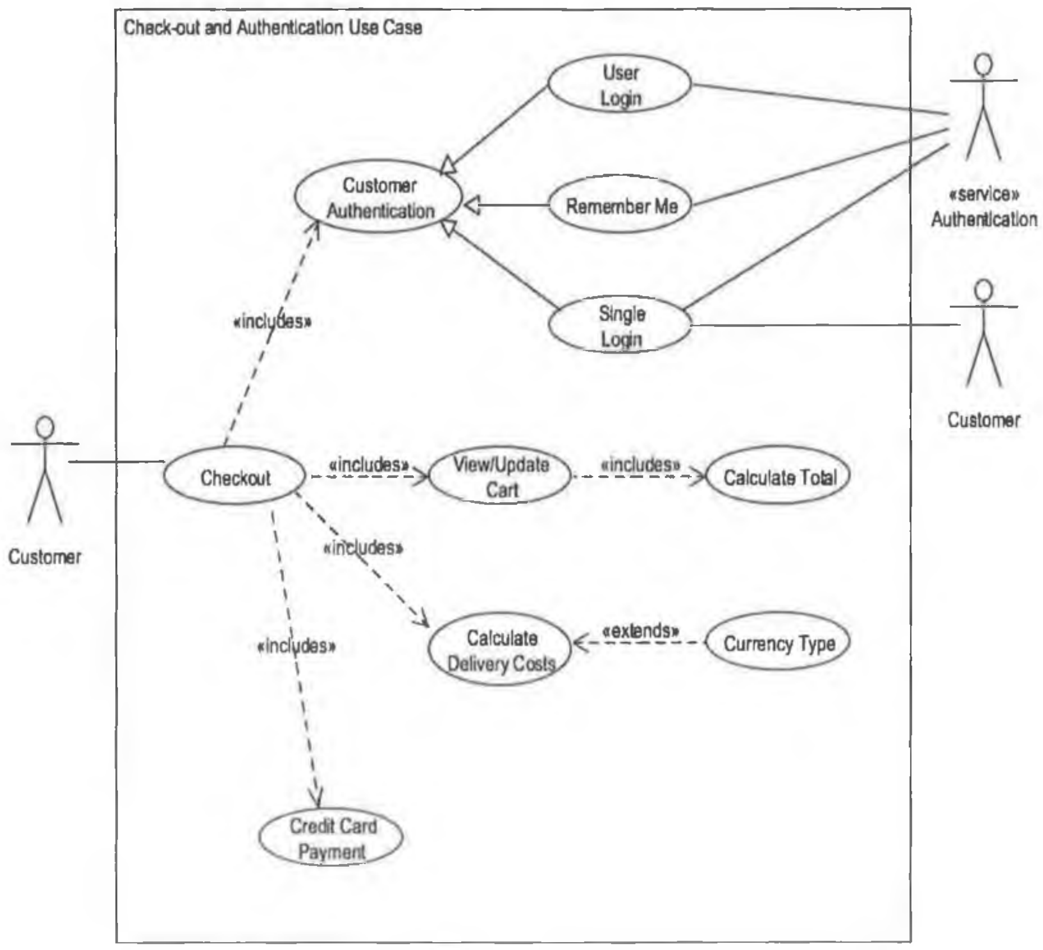


Illustration 9: Use Case - Check-out and Authentication

Illustration 9 shows the Use Case diagram for the applications shopping cart check-out and authentication. The shopping cart allows users to add items they wish purchase as the browse the product catalogue. The customer can view all the items in their shopping cart at any time. Any items can be removed from the shopping basket if required. The shopping cart will also specify any delivery or related charges and give the customer the total price before they proceed to the payment page. There is a requirement the customer must be logged in to use this functionality.

4.5.3 Order Processing

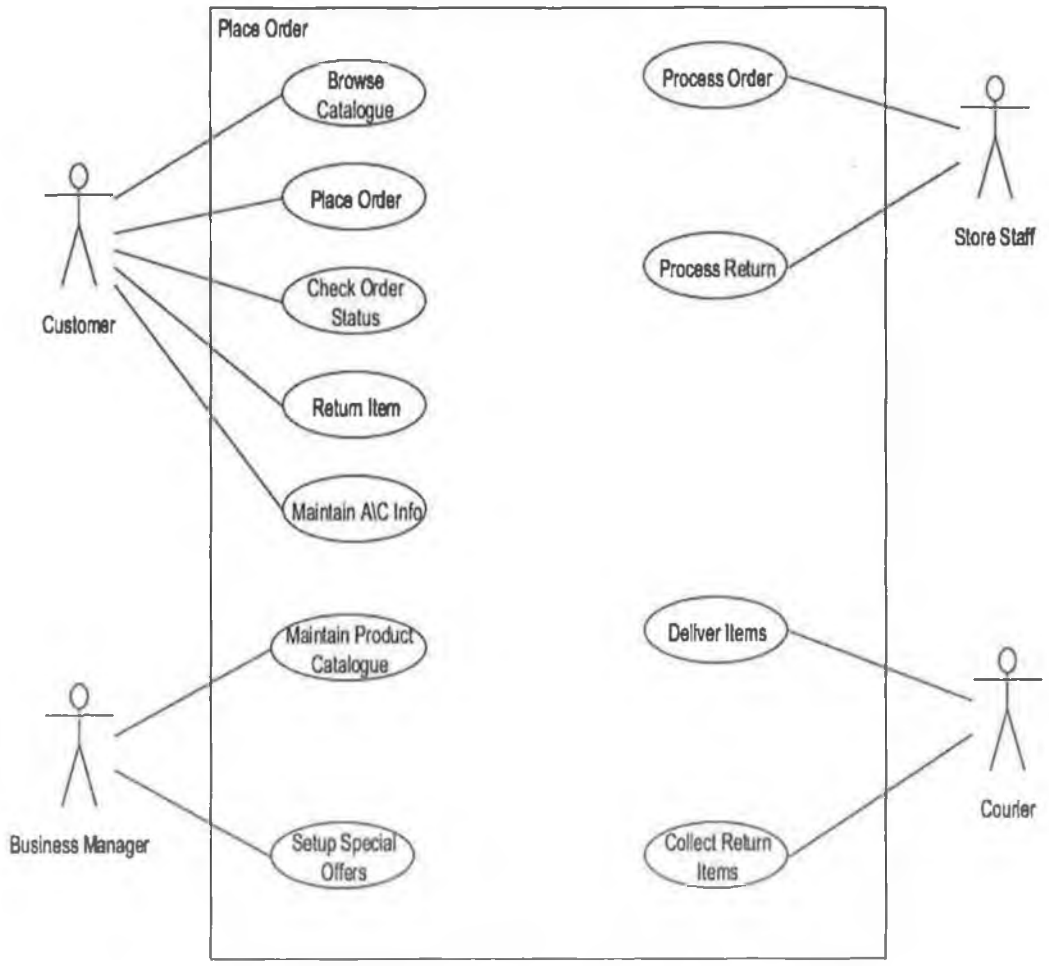


Illustration 10: Use Case - Place Order

Illustration 10 shows the Use Case diagram for the place order functionality. Order processing allows customers to purchase all the items added to their shopping cart. This will require the customer to enter their credit card details to complete the order process. The customer will have the option to manage their own account information and check the current status of their orders. The business staff can view a list of all new orders to enable completion and shipping of orders in a timely manner.

4.6. Class Diagrams

The class diagrams for the legacy application are shown in Appendix F. The diagrams were created using the information gathered from the business owner in the requirements capture phase of the project. The three class diagrams show a static view of the classes for the new Domain Model. The three main diagrams are product, orders, and shopping cart. The relationships between the different classes are also shown. The class diagrams will be used as a map to create the objects for the new applications Domain Model, specifying their relationships and adding any required business logic. The relational database will strictly follow the class diagram as the implementation will follow a Data Driven Design (DDD).

4.7. Conclusion

The requirements analysis was used to gather all relevant data about the business to facilitate the creation of the enterprise application. The capture followed the process as a case study for an existing golf store. The problem definition identified the need for a new enterprise application and the project scope showed the functionality the business wanted in the new system. The functional requirements were obtained from meetings with the business owner. UML Use Case and Class diagrams were created to document the functionality and requirements of the new application and as a method of communicating and portraying this to the business owner. The next phase is the coding and development of the new enterprise application and this will be covered in Chapter 5.

5. Enterprise Application Design and Implementation

5.1. Introduction

The first step in the development of the application was the creation of the user interface. The quality of the user interface was a specific requirement of the business owner, and this must be completed and signed off on by the business owner before the functionality is added. This step ensures the business owner is satisfied with the look and feel of the site. Any changes to the user interface after this will be managed through change requests. Any additional changes may result in the project deadline being pushed out.

The user interface was created using HTML, CSS, JavaScript, and jpeg images. Based on the UML Class and Use Case diagrams it was possible to determine the data needed to display product items and menu options. There is no functionality added to the user interface and this will be used as a template when the application is being coded. Minor changes can be made to the user interface in the development process but the general look and feel of the user interface must be incorporated into the finished application. The next section will detail the main elements of the user interface and their main functions.

5.2. Header and Site Logo

The header was created to incorporate the logo of the business owner. As the main purpose of this site is to sell golf products there was a requirement to incorporate a golf theme to the header and site logo. The graphic design for the header used a green theme to associate with the grass on a golf course and the golf balls and an integral part of the business logo and the faded into the background. Illustration 11 shows the completed header and site logo.

Illustration 11: Header and Site Logo

5.3. Latest Products

The latest product section will display all the products to the user. This will first be presented to the customer when they first visit the main page of the site, and the products will be sorted on date, the newest displaying first. Illustration 12 shows the layout for the product display.

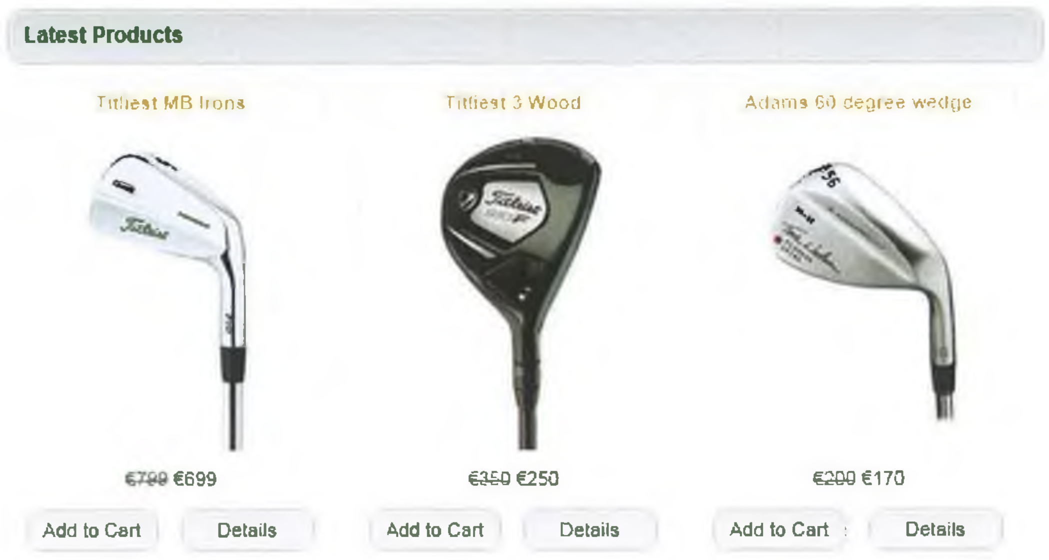


Illustration 12: Product Display

The product display is only a summary or overview of the product. The summary will show the product name, an image of the product and the price of the item. Links will be provided to allow customers view detailed information on the product or add it directly to their shopping cart. Products will be displayed in rows of three depending on how many products are returned from the customers search.

5.4. Category Display

The Categories menu will display all the categories of products on the site, as shown in Illustration 13. The range of golf products for sale is extensive, and the business owner requested that the products be assigned to categories to allow customers view similar products and find what they are looking for. This has been catered for in the Use Case and Class diagrams.

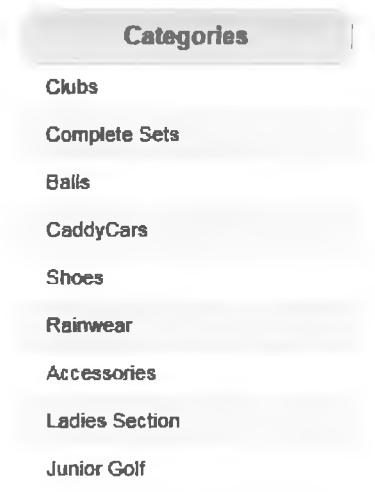


Illustration 13: Category Display

Each product is assigned to a category when it is added to the site. When a customer clicks one of the Categories links, all products associated with this category will be displayed in the format shown for Latest Products as shown in Illustration 12 above.

5.5. Brand Display

The Brand menu will display all the particular product brands available for the site and is shown in Illustration 14. This caters for customers interested in a specific brand. The business owner knows the importance of brand loyalty among customers, and wants to make it as easy as possible for customers to find their desired golf items. Each product has a brand name. When a customer clicks one of the Brand links, all products associated with this brand will be displayed is the format shown for Latest Products above.

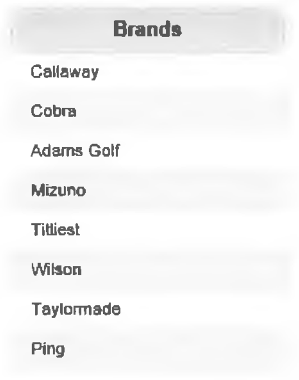


Illustration 14: Brand Menu

5.6. Site Menu

The site menu contains links to all of the functionality the site has to offer. It allows new customers navigate to the signup page, and existing customers can browse the product catalogue and login to view their cart and complete their orders.

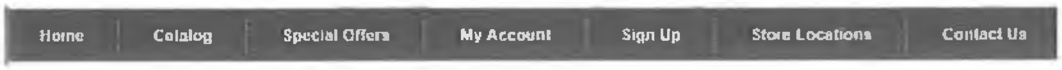


Illustration 15: Site Menu

All the menu options are shown in Illustration 15. They are not explained in detail as they are straightforward and self explanatory.

5.7. *Main components of the Legacy Application*

5.7.1 Development Platform

ASP.NET is a web application framework developed by Microsoft (Microsoft ASP.NET 2011). It facilitates the creation of dynamic web applications and services. It was released in January 2002 with version 1.0 of the .NET platform. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language. C# is a modern object-oriented programming language that works with the Common Language Runtime (CLR) (Troelsen 2010).

5.7.2 SQL Server Database

The database for this application will be a SQL Server Database. As the completed application will be hosted on the Azure Platform, using SQL Server will allow the database to be migrated over to SQL Azure. The database will be created and populated using SQL Server Management Studio. The completed database can be transferred over to SQL Azure with minimal problems. This will be covered later in this section.

5.7.3 ASP.NET MVC

In October 2007 Microsoft announced the new MVC web development platform (Sanderson 2010). It is designed to run on the core ASP.NET platform and is a solution to the limitations of Web Forms. ASP.NET MVC offers improved application development with improved separation of concerns. The MVC pattern is not a new concept, it originated in 1978 in Smalltalk (Sanderson 2010). Microsoft made their version on ASP.NET MVC open source to keep in line with other competitive alternatives. For the Golf Store enterprise application ASP.NET MVC will be used. This will allow the application development to be well-architected, offering fast download speeds and cross-browser compatibility.

The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller (ASP.NET 2010). MVC is a standard design pattern and this web application will benefit from the MVC framework. The MVC framework includes the following components (ASP.NET 2010):

- Model objects are the parts of the application that implement the logic for the applications data domain.
- Views are the components that display the applications user interface (UI).
- Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI.

5.7.4 Domain Model

The Domain Model in an application can be describes as "...a conceptual layer that represents the domain you are working in" (Millett 2010). Items in the domain model will have relationships to other things, and in the Golf Store Application these are products, orders, and categories. These items have data and behaviour and it will contain all the data associated with the item and all the business logic needed for its operations. All logic relating to the items should reside in the domain model. This will separate logic code from presentation code, creating a more maintainable application.

For this project a separate domain model will be used to encapsulate the business logic. This technique helps decouple the domain objects from the other functions of the system (S. Sanderson & D. Sanderson 2010). From the start of the project it was decided to use the domain model to provide this separation.

Following the information gained in the requirements capture the domain model was constructed. The UML class diagram shown in Appendix F was used to model the classes for the business objects. Any functionality related to the business logic will be kept in the domain model and this will be discussed later in this section.

5.7.5 Data Access Layer

A data access layer is used for a system to store and retrieve information from a database. The domain model will host the data access functionality and repositories will be created to provide this functionality. The repositories are nothing more than object-oriented representations of the underlying database acting as a façade over the real implementation (ASP.NET 2010). A separate repository was created for each of the objects in the project.

5.7.6 LINQ to SQL

LINQ to SQL is an ORM tool designed to provide strongly typed views of database information (Microsoft MSDN 2007). It was decided to use this in the project as it quick to implement and the project can deliver quicker for a small scale application. The requirements analysis showed there was no complicated database relations needed to provide the client with all the functionality they required. Entity Framework was investigated as a possible strong contender, but LINQ to SQL was favoured as it is pre-built into Visual Studio 2010.

5.7.7 Dependency Injection – Ninject

Dependency Injection (DI) is a technique shown in Martin Fowler's article Inversion of Control Containers and the Dependency Injection Pattern (Fowler 2004). Dependency Injection (DI) facilitates the injection of objects into a class rather than relying on the class to create the object itself. The use of a factory class is one common way to implement DI. Dependency Injection aims to reduce the amount of boilerplate wiring and infrastructure code that you must write (MSDN Magazine 2005).

Ninject is a software component that can be added to an ASP.NET MVC project to provide Dependency Injection. “Ninject is a lightning-fast, ultra-lightweight dependency injector for .NET applications. It helps you split your application into a collection of loosely-coupled, highly-cohesive pieces, and then glue them back together in a flexible manner. By using Ninject to support your software's architecture, your code will become easier to write, reuse, test, and modify” (Ninject 2010).

5.7.8 Unit Testing

NUnit is a unit-testing framework for all .Net languages (NUnit 2010). This was chosen as it widely used across industry for unit-testing applications. Moq is a mocking library for .NET developed from scratch to take full advantage of .NET 4.0. It supports mocking interfaces as well as classes. Its API is extremely simple and straightforward, and doesn't require any prior knowledge or experience with mocking concepts (Moq 2010).

The first part of the project to be developed was the product catalogue, allowing for all the items to be displayed on the site and the option to sort them by category. Unit testing and mock data were used in the coding of this functionality. The unit tests allowed for certain functional requirements to be added to the tests and these will not pass if the functionality is not correctly coded into the project. The menu items required data from the database, Moq was initially used to hard code some details to test the online menu. When everything was working correctly, the Moq data was replaced with the database repository.

Unit testing and test driven development was only used for the Product Catalogue. It involves a fair bit of extra code, which was impacting on the time available to complete the project. Unit tests are a very fast, focused, and precise way to define specific behaviours and then verify that your implementation matches them (NUnit 2010).

5.8. Implementation of the Golf Store

5.8.1 Initial Architecture Setup

Step 1: Create a blank Visual Studio solution.

Step 2: Add a blank C# class library project

Name this GolfStore.Domain

This will contain all the business logic and the database repository

Step 3: Add an empty MVC 2 Web Application

Name this GolfStore.Web.UI

This will contain the web interface and any related code

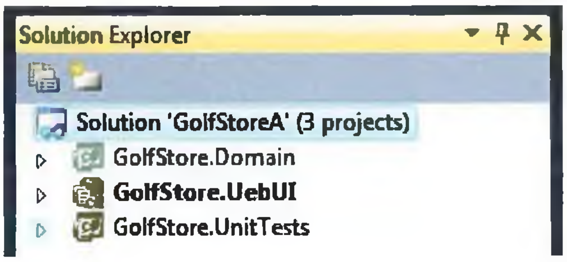
The Views and Controllers for the MVC Pattern are contained here.

Step 4: Add a C# class library project

This class project will be used to hold all the unit tests for the application

The three projects contained in the same Visual Studio solution offers good separation for concerns for the application. The GolfStore.Domain project is the Domain Model used to separate out all the business logic from the application. This allows the web pages remain lightweight and not contain complicated logic.

In the solution explorer right click the empty MVC 2 Web Application and select "Set as Startup Project".


Solution Explorer

Solution 'GolfStoreA' (3 projects)
GolfStore.Domain
GolfStore.UebUI
GolfStore.UnitTests

Illustration 16: Set as startup project

5.8.2 Setup the Domain Model

The Domain Model will be divided into a number of different sections. The first section to be created is Entities. Add a new folder named Entities to the Domain project. All Business entity classes for the application will be contained within this folder. The first entity to create is Product. The UML class diagrams will give the properties needed to create the product class.

```
namespace GolfStore.Domain.Entities
{
    public class Product
    {
        public int ProductID { get; set; }
        public int FKCategory { get; set; }
        public string ProductName { get; set; }
        public decimal OriginalPrice { get; set; }
        public decimal OfferPrice { get; set; }
        public string ProductImage { get; set; }
        public int FKBrand { get; set; }
        public string ProductSummary { get; set; }
        public string AdditionalInfo { get; set; }
        public int ProductVisible { get; set; }
        public int OutOfStock { get; set; }
        public byte[] ImageData { get; set; }
        public string ImageMimeType { get; set; }
    }
}
```

5.8.3 Database Design

The database for the Golf Store was created using Microsoft's SQL Server Management Studio. This allows full control over the database creation and the relationships between the different tables. The information for the products was obtained from the business owner in Excel spreadsheets. These were parsed in Excel to generate insert statements for the table data. The entire database schema is listed in Appendix x.

```
PRINT 'Create Table Products'
CREATE TABLE Products
(
                ProductID int identity(1,1),
                FKCategory int,
                ProductName nvarchar(255) NOT NULL,
                OriginalPrice money default 0.00,
                OfferPrice money default 0.00,
                ProductImage nvarchar(100) NOT NULL,
                FKBrand int,
                ProductSummary text NOT NULL,
                AdditionalInfo text,
                ProductVisible int default 1,
                OutOfStock int default 0,
                ImageName nvarchar (100),
                ImageData varbinary(MAX),
                ImageMimeType nvarchar (45),
                Primary Key(ProductID),
                FOREIGN KEY (FKCategory) REFERENCES
ProductCategory(ProductCategoryID),
                FOREIGN KEY (FKBrand) REFERENCES Brands(BrandID)
)
```

5.8.4 Database Repository

To retrieve information from the database, the Repository Pattern will be implemented. This pattern acts like an in-memory collection, completely isolating business entities from the underlying data infrastructure. This pattern works very well with the Domain Model Pattern, utilising the plain old common language runtime object (POCO) and the persistent ignorant (PI) objects. As this is a domain-driven design (DDD) methodology, a Repository will be created for each entity within the Domain Model. The concrete implementations of the repository will be abstracted away from the entities using interfaces. This will promote loose coupling and could offer the flexibility to replace the Data Access layer without the need to re-factor the whole application.

Create an interface class named IProductsRepository in the Abstract folder

```
namespace GolfStore.Domain.Abstract
{
    public interface IProductsRepository
    {
        IQueryable<Product> Products { get; }
    }
}
```

Create a concrete implementation of the SQL Product Repository.

```
namespace GolfStore.Domain.Concrete
{
    public class SQLProductsRepository : IProductsRepository
    {
        private Table<Product> productsTable;

        public SQLProductsRepository(string connectionString)
        {
            productsTable = (new DataContext(connectionString)).GetTable<Product>();
        }

        public IQueryable<Product> Products
        {
            get { return productsTable; }
        }
    }
}
```

The next step is to wire up LINQ to SQL to provide direct mapping between the products database table and the entity class in the domain model. This will require the Product entity class to be updated with the LINQ to SQL attributes. These will create an association between the Product class and the Products database table. It specifies the database table name and the Columns that are to be mapped.

```
namespace GolfStore.Domain.Entities
{
    [Table(Name = "Products")]
    public class Product
    {
        [Column(IsPrimaryKey = true, IsDbGenerated = true, AutoSync = AutoSync.OnInsert)]
        public int ProductID { get; set; }

        [Column] public int FKCategory { get; set; }
        [Column] public string ProductName { get; set; }
        [Column] public decimal OriginalPrice { get; set; }
        [Column] public decimal OfferPrice { get; set; }
        [Column] public string ProductImage { get; set; }
        [Column] public int FKBrand { get; set; }
        [Column] public string ProductSummary { get; set; }
        [Column] public string AdditionalInfo { get; set; }
        [Column] public int ProductVisible { get; set; }
        [Column] public int OutOfStock { get; set; }
        [Column] public byte[] ImageData { get; set; }
        [Column] public string ImageMimeType { get; set; }
    }
}
```

5.8.5 Web Interface

The web interface was created using HTML, CSS and JavaScript as a non-functional prototype to give the business owner the opportunity to sign off on the design before the application development commenced. The created interface will be incorporated into the Site.Master file that was created earlier. This will act as a site template, containing the common elements all pages should contain.

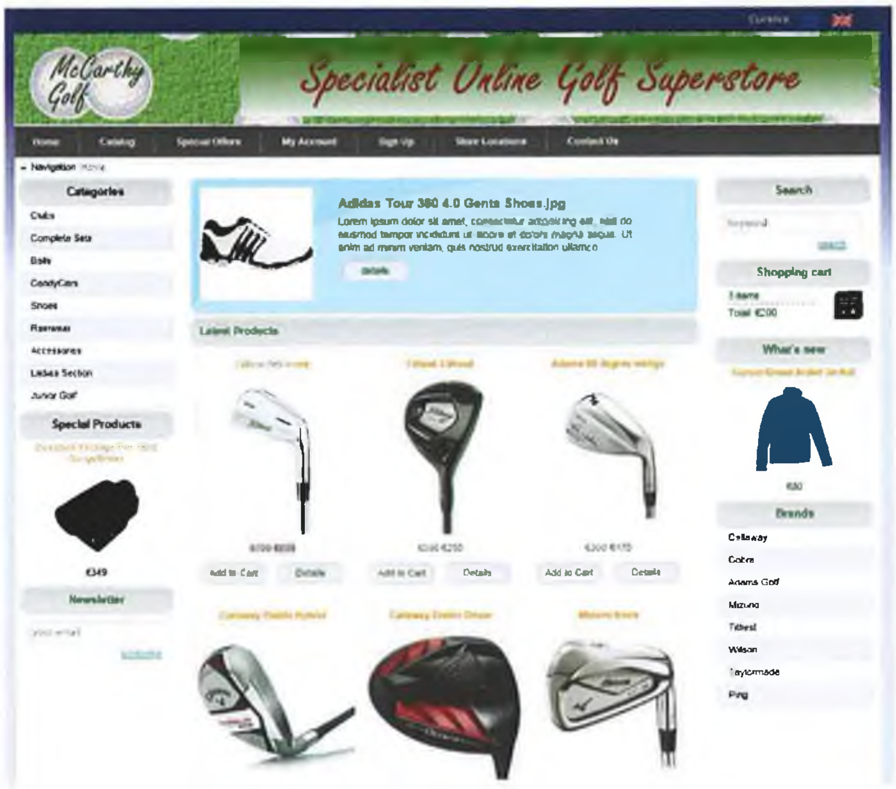


Illustration 17: Golf Store Site Template

The header, menu and secondary menus and the site footer will all be contained in the Site.Master file. The views in the MVC Pattern will use this file as a template into which the views content will be added. This reduces the amount of unnecessary code and replication in all the view pages.

5.8.6 MVC Controllers

The controllers will interact between the views and the domain model for this application. The Product Controller will retrieve data from the Repository and pass the results to the view for display.

```
public class ProductsController : Controller
    {
        private IProductsRepository productsRepository;
        private ICategoryRepository categoryRepository;
        private IBrandRepository brandRepository;

        public ProductsController(IProductsRepository productsRepository,
                                       ICategoryRepository categoryRepository,
                                    IBrandRepository brandRepository)
        {

            this.productsRepository = productsRepository;
            this.categoryRepository = categoryRepository;
            this.brandRepository = brandRepository;
        }
```

The Controllers constructor initialises the interfaces to the Repository. To list all products available:

```
public ViewResult List(string category, int page = 1)
        {
            var catID = 0;
            if (category != null)
            {
                var categoryID = from c in categoryRepository.Categories
                where c.CategoryName == category select c.ProductCategoryID;
                catID = categoryID.First();
            }
            var productsToShow = (category == null)
                ? productsRepository.Products
                : productsRepository.Products.Where(x => x.FKCategory == catID);

            return View(productsToShow.ToList());
        }
```

5.9. Conclusion

This section showed the architecture used in setting up the Enterprise Golf Store. This was only an overview of the code and techniques used in implementing the application. There are a number of domain entities used in this application and it is not feasible to document the entire application line by line. Chapter 6 will begin the re-factor of the application, listing the design patterns and system architecture used by following Fowler's patterns for enterprise design.

6. Re-factor the Application using Design Patterns

6.1. Introduction

The existing legacy application will be re-factored following the design techniques in Fowler's book on enterprise design (Fowler 2003b). Scott Millett's book Professional ASP.NET Design Patterns (Millett 2010) was used to find practical ASP.NET examples of the techniques detailed in Fowler's book. The re-factored application draws heavily on the examples from Millett's book. The level of architecture, design and separation of concerns is quite impressive. The time spent implementing the re-factor was considerable when compared to the time spent creating the legacy application.

The infrastructure for the application will be separated into a single project class library. Within this project the different functionality offered will be kept in their own individual folder. The infrastructure will add the following functionality to the application:

- Domain Layer Supertype
- Unit of Work Pattern
- Query Object Pattern
- Application Configuration Settings
- Logging
- Helper Classes

The infrastructure project uses folders to separate the different functionality in the domain model. The project contains the folders listed in Table 3.

Authentication	Helpers
Configuration	Logging
CookieStorage	Payments
Domain	Querying
Email	Unit of Work

Table 3: Infrastructure Folders

When completed the infrastructure project could be renamed and have it as a separate dependency that can be used in all new projects.

6.2. *Improved Domain Model*

Fowler defines a Domain Model as "... an object model of the domain that incorporates both behaviour and data" (Fowler 2003b). Business logic is complex and different rules and logic can be used to describe different behaviour. A Domain Model creates a web of interconnected objects and each object will represent a meaningful individual (Fowler 2003b).

To use the Domain Model in an application a whole layer of objects will be created in the application. The objects will model the business area the project relates to. Objects will mimic the data and rules in the business. An object orientated domain model will be very similar to the database model, but there will be differences. The Domain Model will mingle data and processes, has multivalued attributes with complex associations (Fowler 2003b). Fowler identifies two styles of Domain Model used in industry today:

- Simple Domain Model
- Rich Domain Model

A simple domain model is very similar to a database with one domain object for each database table. This can use Active Record for database mapping. A rich domain can differ from the database design using inheritance, strategies and other design patterns (Fowler 2003b). The rich domain is better for dealing with complex logic. To facilitate database mapping the rich domain will use a Data Mapper.

Business data and logic is prone to change, and from a development and maintenance perspective it is important to be able to modify and test this layer. To enable this the Domain Model will require minimum coupling with other layers in the system. This is a guiding force of many layering patterns. The key is to limit the number of dependencies between the system and the domain Model.

Fowler's preference is to use a Data Mapper for the database interaction using the Domain Model. This keeps the Domain Model independent of the database. Fowler also notes when the Domain Model is implemented the Service Layer could be used to give the Domain Model a more distinct API.

Scott Millett uses Fowlers enterprise patterns in his book “Professional ASP.NET Design Patterns”. Millett uses the Domain Model and also adds additional techniques used in industry today. The Domain Model is a conceptual layer that represents the domain the business is in. Objects that exist in the Domain Model will have relationships with things in the real world. An example of this would be a basket or an order for a store. These things will have both data and behaviour. “The closer your domain model represents the real domain the better” (Millett 2010). It will make it easier to model complex business logic and its associated rules. As seen in Fowlers work, the Domain Model has no knowledge of persistence. To persist a business object in the domain model the Repository pattern must be used. This will use a data mapper to map the business entities to the entities in the data model.

As an addition to Fowlers description of the Domain Model, Millett introduces the Domain Driven Design methodology. This utilises the Domain Model pattern. “It is a way of thinking and a set of priorities, aimed at accelerating software projects” (Domain Language, Inc. 2011). This technique models the real domain first. Domain Driven Design is not a framework, it offers a set of building blocks to be incorporated into the solution (Millett 2010). The major components of Domain Driven Design will be briefly described to give an insight of how it operates in the Domain Model.

6.2.1 Value Objects

Value objects have no identity and are only used for their attributes. Value objects are generally attributes of an entity. These will be discussed in more detain in the aggregates section below.

6.2.2 Aggregates and Aggregate Roots

For large enterprise systems the domain could contain hundreds of entity and value objects which have complex relationships. To deal with this complexity the domain model needs to manage these associations. Entities and value objects will need to define an interface to let other objects interact with them.

An aggregation is a grouping of logical entities and value objects. The aggregate root is an entity and is the only member of an aggregation grouping that can hold a reference to an external object. This is used to ensure data integrity within the Domain Model.

6.3. Domain Layer Supertype

The domain layer supertype acts as the supertype for all types in its layer. This will create a Domain Object superclass for all the domain objects in the Domain Model (Fowler 2005). The layer supertype pattern defines an object that will act as the base class for all types in its layer. Its purpose is to centralise business logic shared among objects. To implement this the domain-driven design principles will be used. The aggregate root will be used as the interface for a group of related domain entities. The aggregate roots main function is to ensure the aggregation remains in a constant valid state (Millett 2010). The entity assigned as the aggregate root will also have a repository to deal with data persistence. An interface was created in the Domain folder as follows:

```
namespace GolfStore.FrontEnd.Infrastructure.Domain
{
    public interface IAggregateRoot
    {
    }
}
```

All business entities will inherit from this supertype. The sypertype will also use a simple framework to check if an entity class is valid. As the application is using the domain-driven model not all objects will be modelled as entities. As seen in the section on domain driven design the application will be comprised of entities and value objects. Value objects do not have an identity and are usually attributes of an entity. For the golf store there are two sets of aggregations. There will be entities that can be retrieved and persisted and entities that can only be retrieved (read only). This is setup by creating two different interfaces. The first will be the read only repository, only allowing the implementation of read requests. The second interface will allow the implementation of CRUD functionality for the entity in question.

6.4. Unit of Work

Fowler defines that the Unit of Work pattern "...maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems" (Fowler 2003b). If an application uses a database, it is important to know what data has been requested and what has been changed. For a simple system this may not be a problem, but more complex systems may have thousands of users that may request and change the data at the same time. To avoid users overwriting updated data or viewing out of date data, a mechanism is required to deal with this issue. It is possible to change the database with each change in the application, but this will lead to lots of small database calls, which will slow down the system. The Unit of Work pattern will keep track of all changes made in a business transaction. When everything has been completed with the current task the Unit of Work pattern will make the appropriate changes to update the database with all the appropriate changes.

The Unit of Work will keep track of any changes made to objects in the application. In the application every time an object is created, updated or deleted the Unit of Work must be informed. Database reads must also be passed to the Unit of Work, to ensure no inconsistencies occur within this transaction. When the transaction is complete the Unit of Work will make the appropriate changes to the database.

This pattern has been identified to help reduce cloud resource consumption. The Azure cloud charges per database transaction metrics, and if an application is making small changes to the database on-going throughout the business transaction this will lead to a very high metric for the per database transaction. Introducing the Unit of Work will help minimise database transactions by keeping all database update to the end of the transaction.

For the implementation of the Unit of Work pattern in the application will require all repositories in the solution to implement the IunitOfWork Repository.

```
namespace GolfStore.FrontEnd.Infrastructure.UnitOfWork
{
    public interface IUnitOfWorkRepository
    {
        void PersistCreationOf(IAggregateRoot entity);
        void PersistUpdateOf(IAggregateRoot entity);
        void PersistDeletionOf(IAggregateRoot entity);

    }
}
```

The Unit of Work contract is kept outside the repository project as the concrete implementation is of no concern to the domain services that will use it (Millett 2010).

```
namespace GolfStore.FrontEnd.Infrastructure.UnitOfWork
{
    public interface IUnitOfWork
    {
        void RegisterAmended(IAggregateRoot entity,
                             IUnitOfWorkRepository unitOfWorkRepository);
        void RegisterNew(IAggregateRoot entity,
                             IUnitOfWorkRepository unitOfWorkRepository);
        void RegisterRemoved(IAggregateRoot entity,
                             IUnitOfWorkRepository unitOfWorkRepository);
        void Commit();
    }
}
```

6.5. Query Object Pattern

The query object pattern uses an object that represents a database pattern (Fowler 2003b). SQL can be a difficult language and many developers are not familiar with it. Large database schema requiring complex queries can be difficult to incorporate in to applications and manage as the system changes. A Query Object is based on the original Interpreter pattern from the Gang of Four. It is a structure of objects that can form SQL queries. This is achieved by referencing to classes and fields rather than tables and columns (Fowler 2003b).

To implement the query object pattern in the re-factored application the query will be constructed within the domain service layer and passed to the repository to be satisfied (Millett 2010). This acts as a query translator by converting the object pattern into a form the persistence storage can understand. The query object for this application will deal with general queries and sub-queries. The main benefit of using the query object pattern is that it can grow as the application grows. Additional query operators can be added as needed and the query object could be moved to its own project solution. This could be re-used for any new projects.

6.6. *Application Configuration Settings*

The re-factored application will have a large number of configuration settings in comparison to the original application. All settings are stored in the Web.Config file in the MVC project, but this could change. To facilitate any future changes a component will be created to abstract these settings away from the implementation.

6.7. *Helper Classes*

Helper classes will be added to the application to abstract formatting issues away from the application. For the product catalogue the prices will be formatted to Euro using helper classes.

```
namespace GolfStore.FrontEnd.Infrastructure.Helpers
{
    public static class PriceHelper
    {
        public static string FormatMoney(this decimal price)
        {
            return String.Format("€{0}", price);
        }
    }
}
```

The FormatMoney method is an extension method to a decimal and will allow any variable of type variable to access this method and format the pricing for the product catalogue.

6.8. *Creating the Product Catalogue*

The requirements for the product catalogue remain the same as for the original application. A number of improvements will be added to the usability of the site using RIA techniques. The database will also be re-factored to help optimise the applications performance. The following sections will detail the creation of the new Domain Model, the database Repository using NHibernate, and the use of AutoMapper to provide object to object mapping in the service layer for use with the view models. JSON techniques will also be introduced to provide AJAX functionality for the product catalogue views.

6.9. Software used in the Application

6.9.1 NHibernate

“NHibernate is a mature, open source object-relational mapper for the .NET framework. It's actively developed, fully featured, and used in thousands of successful projects” (NHibernate 2011). For the re-factor of the application NHibernate will be used as the Object Relational Mapper (ORM). NHibernate supports persistence ignorance therefore the applications business objects won't have to inherit from base classes or implement framework interfaces. It will allow the application to query the database and add, update and delete items as needed. To facilitate mapping business objects to the database tables NHibernate uses XML configuration files. NHibernate has been chosen as the ORM for the re-factor of the application as LINQ to SQL is typically designed for more simple data structures and is difficult to implement sub queries and more complex queries. LINQ to SQL tables are mapped strictly on a 1:1 basis. For the original application there were issues with sub-queries sorting products on categories and brands. This will be addressed in the re-factor of the application.

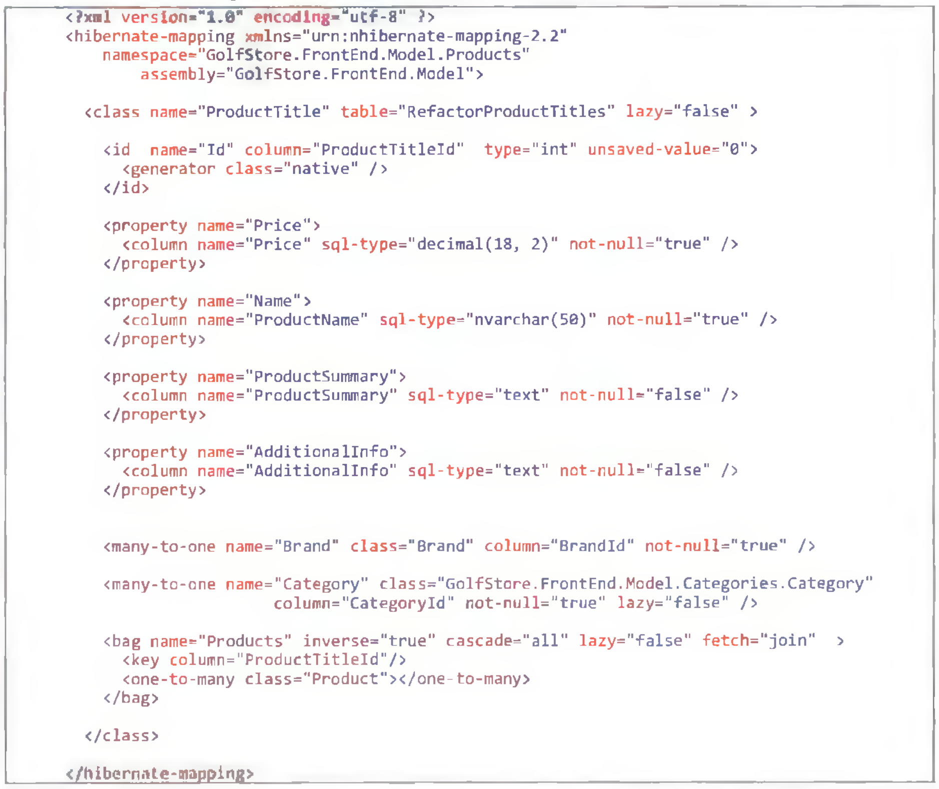
6.9.2 AutoMapper

“AutoMapper is an object-object mapper. Object-object mapping works by transforming an input object of one type into an output object of a different type” (AutoMapper 2011). This will be used in the application to manage the mapping to the ViewModels. AutoMapper is open source.

6.10. Repository Layer

The repository pattern is used in Domain Driven Design as an in-memory collection or repository for business entities. It will completely abstract the underlying data structure. This keeps the Domain Model free of any infrastructure issues.

The database for the re-factored application follows the layout of the class diagram in Appendix F. The repository will be used to retrieve product information and map between the data model and the business model. NHibernate will be the object relational mapper (ORM) in the Repository Layer and offers many built-in enterprise patterns (Millett 2010). The Repository Layer is a separate project in the Visual Studio solution. The first step in implementing the repository using NHibernate is to map between .NET-Classes (Objects) and relational data (Database-Tables) (Codegod 2006). These mappings will be defined using XML documents. The NHibernate DLL must be added to the projects references. For each object in the domain model a XML mapping file will be created to associate with the corresponding database table.



```xml
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
    namespace="GolfStore.FrontEnd.Model.Products"
        assembly="GolfStore.FrontEnd.Model">

  <class name="ProductTitle" table="RefactorProductTitles" lazy="false" >

    <id  name="Id" column="ProductTitleId"  type="int" unsaved-value="0">
      <generator class="native" />
    </id>

    <property name="Price">
      <column name="Price" sql-type="decimal(18, 2)" not-null="true" />
    </property>

    <property name="Name">
      <column name="ProductName" sql-type="nvarchar(50)" not-null="true" />
    </property>

    <property name="ProductSummary">
      <column name="ProductSummary" sql-type="text" not-null="false" />
    </property>

    <property name="AdditionalInfo">
      <column name="AdditionalInfo" sql-type="text" not-null="false" />
    </property>


    <many-to-one name="Brand" class="Brand" column="BrandId" not-null="true" />

    <many-to-one name="Category" class="GolfStore.FrontEnd.Model.Categories.Category"
                      column="CategoryId" not-null="true" lazy="false" />

    <bag name="Products" inverse="true" cascade="all" lazy="false" fetch="join"  >
      <key column="ProductTitleId"/>
      <one-to-many class="Product"></one-to-many>
    </bag>

  </class>

</hibernate-mapping>
```

The example above shows the NHibernate mapping file for the ProductTitle class. The xml file must be added to the project as an embedded resource to ensure it is available for use within the application. With all of the XML mapping files created the repository can be completed.

The repository will implement the Unit of Work pattern as setup in the Infrastructure project of the solution. A session storage container is used to store and retrieve NHibernate sessions. Additional containers can be added to serve different media, for example a smart phone. A Factory class will be used to supply a valid session container. Using the Unit of Work pattern no changes will be made to the database until a transaction is committed. One of the main reasons NHibernate was used in the re-factor of the enterprise application is its Identity Map functionality. The Identity Map will maintain a single instance of a business entity in the session no matter how many times it is retrieved (Millett 2010). The repository also implements the Query Object pattern. This is used to create SQL queries the database will understand.

6.11. Service Layer

The application service layer sits on top of the domain model to coordinate application activity. It does not contain any business logic or preserve and state. The application service layer can be used to provide an API into the domain model using the Request-Reply messaging pattern.

The product services layer in the re-factored solution manages the retrieval and persistence of business entities. The service layer acts as an intermediary between the controllers and the domain model. The controller will make specific requests to the service layer for updates or to get a specific view of the domain. All views in the application are strongly typed views and these are contained in the service layer.

6.12. Controllers

The controllers for the re-factored application are contained in a separate Visual Studio project and are strongly typed to the corresponding .ASPX views. The controllers have also incorporated JSON objects to allow product category pages be refined using AJAX. JSON objects are converted to .NET objects using the MVC ModelBinder.

6.13. Inversion of Control

"StructureMap is a Dependency Injection / Inversion of Control tool for .Net that can be used to improve the architectural qualities of an object oriented system by reducing the mechanical costs of good design techniques" (StructureMap 2011). In the re-factored application StructureMap is used to enable loose coupling between classes and their dependencies.

6.14. Summary

This chapter identified the enterprise application design patterns to be used in the re-factor of the application. The architecture and design patterns were sourced in Fowler's book 'Patterns of Enterprise Application Architecture' and the implementation method was modelled using Millett's book 'Professional ASP.NET Design Patterns'. The different architectures and patterns were discussed describing its background and how it proposes to minimise resource consumption on the cloud platform. The software used in the application was listed and an explanation was given for its use in the re-factored enterprise application. Some code examples were shown on how the re-factored application was implemented. The next chapter will migrate the two enterprise applications to the Windows Azure Platform.

7. Evaluation and Analysis

7.1. Overview

The evaluation phase of this project will test the legacy application and the re-factored application running on the Azure Platform. The main goal of the project is to evaluate if a re-factored application using enterprise design patterns will help reduce resource consumption on the cloud platform. Chapter 3 detailed the billable consumption metrics for the Windows Azure platform. The main metrics are throughput, compute hours, and database throughput. To test the applications a testing plan will be created for both applications. The plan will test the sites main functionality, which is the product catalogue. This forms the main functional component of the site with customers spending a lot of time viewing potential purchases and items of interest.

7.2. Migrate Applications to Azure

Before any testing can commence both applications needed to be migrated to the Azure Platform. The first application to be migrated was the legacy application. Microsoft offers a Windows Azure SDK which operates seamlessly with Visual Studio (Azure SDK 2011). Section 3.3.2 gives a detailed description of the features and functionality of the Azure SDK. Using the Azure SDK the applications will be prepared for their migration to the Windows Azure Platform.

The applications cannot be uploaded straight to Azure as they currently stand, minor modifications to both applications will be required (Hay & Prince 2010). A cloud service project must be added to the solution to enable the application run in the cloud environment. The existing MVC Web UI will be set as the web role in the cloud service project. Any references to library files must be added to the cloud service project. With these changes the application can now be run and tested locally. If there are any outstanding issues these will be flagged by the debugger. Running the application locally using the Azure SDK allows developers test before deploying on Azure.

With the applications now running locally using the Azure SDK, the database can now be moved to SQL Azure. As the proposed research is to migrate an existing enterprise application with a SQL Server database, SQL Server Management Studio can be used to script the entire database as a text file.

A new SQL Azure database can be created using the Azure Management Portal. Once created, the connection details will be available, allowing connections using the SQL Server Management Studio. The database can be created on SQL Azure by executing the SQL script file.

The final step is to deploy the applications to Azure. Visual Studio will manage the complete process. Appendix D details a complete migration step by step from start to finish. The process used in Appendix D can be used for any ASP.NET enterprise application as a cloud migration strategy.

7.3. Testing Plan

Visual Studio Team System 2008 will be used to create the testing plan. This software facilitates the creation of a web testing plan, which consists of the urls to be used. This will simulate a customer using the website. It is important the web tests created are identical for both applications. As this test is comparing the effectiveness of two applications against each other, the simulated testing plans must match exactly. The page sequence is shown in Table 4.

1	Home Page
2	List Drivers
3	Details Mizuno FastTrack MP630
4	List Drivers 2
5	Details Titleist 910 D2 Driver
6	List Drivers 3
7	Detail Titleist 910 D3 Driver
8	List Titliest from Brands Menu
9	List Nike from Brand Menu
10	Detail Nike SQ Machspeed Black Driver
11	List Irons from Category Menu
12	Detail Titleist AP2 710 Irons
13	List Titliest from Brand Menu
14	Detail Titleist MB 710 Irons

Table 4: Web Testing Sequence

To ensure a proper and realistic test is conducted, a testing station will be setup on the client side with an internet connection to run the tests. This will be a standard PC hosting Visual Studio Team System 2008 to simulate specific concurrent loads. Visual Studio Team System will facilitate the creation of specific testing plans, detailing which pages to visit and in what sequence. It will also record specific metrics on the performance of the application on the client side. Throughput will not be recorded using this software.

Using the above pages a testing plan was created for both applications. The testing plan consists of a list of pages to be visited in a specific sequence. Table 4 shows the testing sequence for both testing plans. Using these a load test was created to simulate a high volume of traffic for the site concurrently. For this test the load consisted of 25 concurrent users browsing the 14 links listed above and was repeated in a loop of 10 iterations.

The tests will be run a number of times for both applications, to ensure there is an even distribution for all result sets. The metrics will be taken from the Azure Management Portal. This portal keeps track of the metrics and consumption rates for all the applications and instances running on the Azure Platform.

Data Transfer Usage Charges
SQL Azure Usage Charges
Windows Azure Usage Charges

Table 5: Windows Azure Usage Charges

Table 5 shows the three main categories of usage charges for the Azure Platform. These metrics will be used to measure the efficiency of the two applications. It would have been possible to record metrics on the client side and also within the application itself, on a layer by layer basis. This would give invaluable information on the performance of the application generally, but the main aim of this body of work is to examine the applications consumption of cloud resources. On the Azure Platform the metrics consumed are displayed and the cost incurred for this usage is also available. To keep the results as accurate as possible the metrics will be taken from the Azure Management Portal. For each test of the application all results will be tabularised and graphed. Each test will be conducted a number of times to ensure for an accurate dispersal of results.

As this research is primarily interested in the effect design patterns have on Azure consumption metrics, the main metrics for consideration will be from Azure. The re-factored application has made primary changes to the Repository, Domain Model and the introduction of a Service Layer to manage messaging and eliminate requesting the same information multiple times. To measure the effectiveness of both applications, the usage data from SQL Azure and Windows Azure will be examined. The data from SQL Azure will be used to measure the performance of the Repository Layer for both applications. The throughput data from Windows Azure will be used to measure the effectiveness of the changes made to the Domain Model and the Service Layer of the re-factored application.

7.4. Data Acquisition

Gathering usage data from the Azure platform is not straightforward or intuitive. The Azure site offers documentation that runs through the headings offered in the billing portal but the exact calculations on how figures are calculated is not evident. To ensure there were no misinterpretations made in the data analysis a number of calls were made to Azure Support to gain a better insight into the exact operation of SQL Azure and Windows Azure from a data metrics perspective. The metrics recorded are based on data transfers in, data transfers out, database consumption in GB and Compute hours. The efficiency of the applications is measured on their usage of the Azure cloud metrics. As the applications and their associated SQL database are hosted on the Azure Platform, the applications will have metrics for data in and out.

“Data transfer rates are determined by the region in which your solution is deployed, whether the transfer is inbound or outbound, and whether the transfer is peak or off-peak. Data transfers between Windows Azure platform services located within the same sub region are not subject to charge. Data transfers between sub regions are charged at normal rates on both sides of the transfer” (Azure Platform Bills 2011).

Metric usage will occur when the application makes a request to the database for record information, the database information retrieved for the application and the delivery of the information to the client requesting the information.

The main area for testing was the product catalogue. For this subsection of functionality the application focused on the client requesting information on a specific brand or product. This request was made to the application and the application retrieved all necessary information from the database to form a response. The following sub sections detail the main metrics gathered for this research.

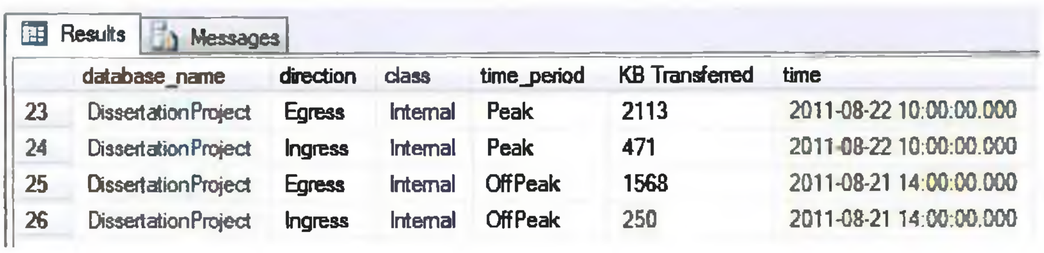
7.4.1 SQL Azure Metrics

The first metrics to be gathered were from SQL Azure. On the Windows Azure Management Portal the billing section there is a section for SQL Azure Usage Charges. This offers no useful data and only displays an average of the resources that are available for consumption. For this research the SQL Azure database was a web edition with 1GB per month limit. The management portal reported daily usage at 32.258MB per day, which is just 1GB divided by 31days! This offers no real insight into any real usage.

The real usage data for the database is stored in the master database. The data is accessible using database views and a query analyser. The view used to get the throughput in and out of the database is sys.bandwidth_usage (Microsoft MSDN 2011c). The select statement is shown below.

```
SELECT database_name, direction, class, time_period,
    quantity AS [KB Transferred], [time]
FROM sys.bandwidth_usage
ORDER BY [database_name],[time] DESC;
```

To execute the query the Microsoft SQL Server Management Studio or the Azure Database Manager can be used. For this project the Microsoft SQL Server management Studio was used. When the query is executed against the database the results will be as shown in Illustration 18.



Results | Messages

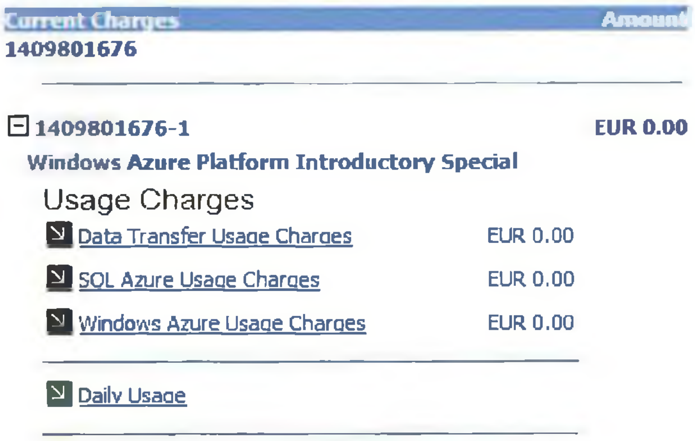
	database_name	direction	class	time_period	KB Transferred	time
23	DissertationProject	Egress	Internal	Peak	2113	2011-08-22 10:00:00.000
24	DissertationProject	Ingress	Internal	Peak	471	2011-08-22 10:00:00.000
25	DissertationProject	Egress	Internal	OffPeak	1568	2011-08-21 14:00:00.000
26	DissertationProject	Ingress	Internal	OffPeak	250	2011-08-21 14:00:00.000

Illustration 18: SQL Azure Metrics

For a particular date and time (measured to the nearest hour) the throughput in and out of the SQL Azure database is given. Egress is throughput out and Ingress is throughput in. For the load tests run using Visual Studio Team System 2008 there will be a corresponding Egress and Ingress data reading. This will give the exact throughput generated by the load test on the SQL Azure database. The data will be gathered over a number of tests for both the legacy application and the re-factored application.

7.4.2 Windows Azure Metrics

The metrics for Windows Azure were gathered using the billing section of the management portal (Microsoft Online Customer Portal 2011). This allows customers to view their consumption of the Azure cloud resources as shown in Illustration 19.



Current Charges — Amount

1409801676

1409801676-1 — EUR 0.00

Windows Azure Platform Introductory Special

Usage Charges

Data Transfer Usage Charges	EUR 0.00
SQL Azure Usage Charges	EUR 0.00
Windows Azure Usage Charges	EUR 0.00

Daily Usage

Illustration 19: Windows Azure Portal Metrics

The Data Transfer Usage Charges section gives an overview of the throughput in and out for the application for the previous billing day. It must be noted that this will not update on a continuous basis and cannot be refreshed to get more up to date information at a later stage. The data in this section is a snapshot of the data usage for the previous day, metrics are only updated on a daily basis. The SQL Azure Usage Charges were explained in detail in the previous section. The Windows Azure Usage Charges gives the total compute hours used by the application. For this research the instance limit was set to 1 to ensure accurate comparisons between the two applications. The compute hours would be 24 if one application was hosted continually on the Azure Platform. These three data sections only give an overview of the data used. For a comprehensive listing of all data metrics this can be obtained from the Daily Usage section. This section contains the functionality to export all the usage metrics an Excel spreadsheet. This spreadsheet has exact details on throughput in and out for each application at the exact time the resource was consumed. These metrics will be used to analyse the performance of the two enterprise applications.

7.4.3 Client Metrics

Metrics will also be gathered on the testing PC hosting the Visual Studio Team System 2008 Team Suite. Visual Studio will run the load test and record information on the performance of the test. This will be used to measure the performance of the applications from the client perspective. These metrics will have no direct impact on cloud resource consumption but the performance of the applications is vital.

The throughput of data in and out on the client PC will also be measured, to give an indication of the expected throughput that should be showing on the Azure Management Portal. Appendix G shows all the data recorded from the Client PC and the Azure Portal.

7.5. Test Results

Ten tests were conducted for both enterprise applications. For each application five tests were hosted on an extra small instance and five were hosted no a small instance. Each test was conducted on a separate day to ensure no ambiguity occurred with the data gathering from the Azure Platform. The applications were both hosted on Azure at the same time. The only traffic the applications received was from the execution of the testing plans. All data was gathered on a daily basis and tabularised in an Excel spreadsheet.

All of the metrics recorded from the Azure Portal are throughput. Throughput inbound can be defined as requests made to the application to supply specific content and data. Requests made from the browser will generate throughput inbound on Windows Azure. The clearest example would be a user clicking a link on a web page to view specific content. Clicking the link generates throughput in as the request is made for a different page or content, throughput outbound is created in serving the page or data to the users' browser.

The following sections displays the findings of the data recorded for all Azure metrics. The analysis of these findings will be discussed later in this chapter

7.5.1 Windows Azure Throughput Out

Illustration 20 shows the graph of the throughput out for the Azure applications. To generate this data a load test with identical page requests was created using Visual Team Server 2008. The legacy application had an average throughput out of 20.9MB for the load tests used. The re-factored application had an average throughput out of 21.5MB.

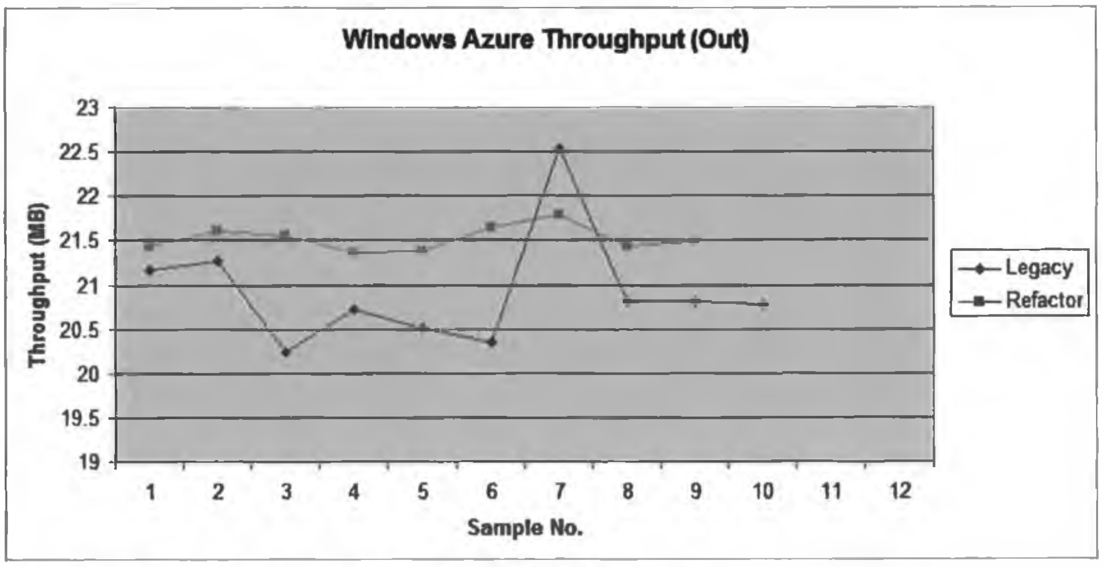


Illustration 20: Windows Azure Throughput Out

The graph clearly shows the legacy application has lower throughput out that the re-factored application. The average difference is 0.6MB, which is not a considerable amount of data but the size of the load test is small. In percentage terms the legacy application uses 2.79% less on the throughput out. If this difference was observed on an enterprise application receiving a high volume of traffic on a daily basis, the difference measured on a yearly basis could be considerable.

7.5.2 Windows Azure Throughput In

The throughput in for the legacy application was higher than the re-factored application. The average for the legacy application was 1.82MB versus the re-factored applications 1.54MB.

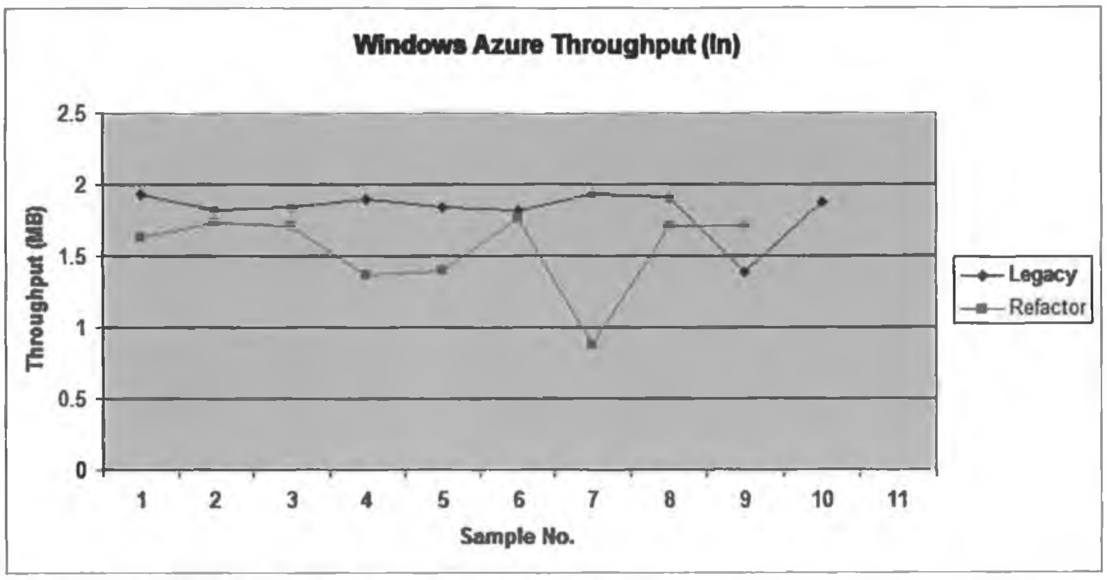


Illustration 21: Windows Azure Throughput In

The re-factored application was more efficient with throughput in, with an average difference of 0.28MB.

7.5.3 SQL Azure Throughput Out

The outbound throughput graph for SQL Azure is shown in Illustration 22. The legacy application had an average of 1555kb per load test. The re-factored application was significantly higher at 2127kb.

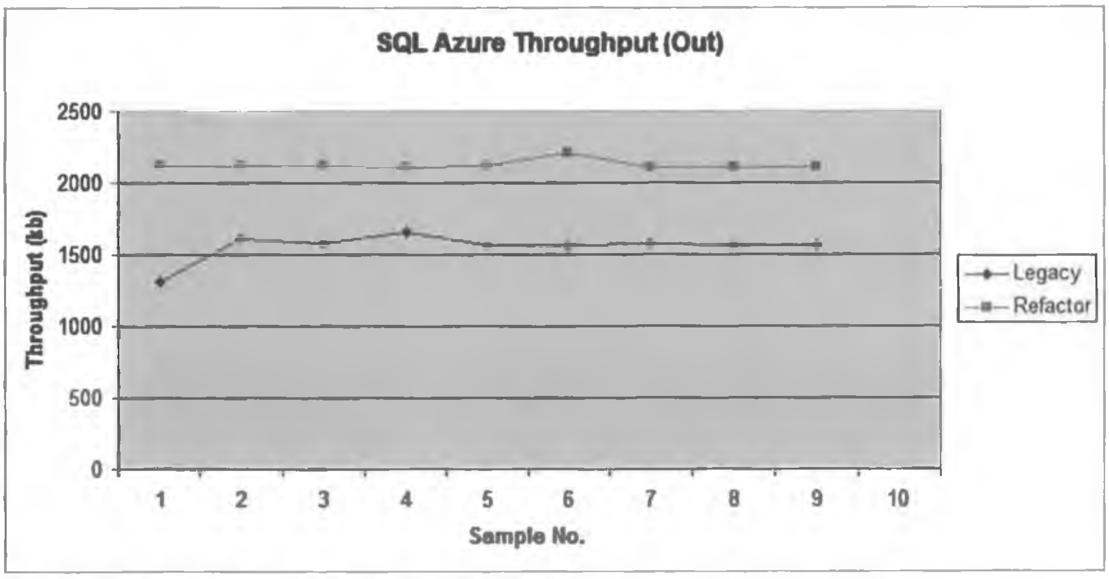


Illustration 22: SQL Azure Throughput Out

The throughput outbound from SQL Azure is the data needed to generate the required web pages. Both the legacy and the re-factored application used the same testing plan, and used the same database information to generate the pages. With a constant amount of data required to generate the pages from the testing plan, the performance difference can be attributed to the effectiveness of the ORM and the repository layer. The legacy application was more efficient using the SQL Azure database. The graph shows the results were very consistent for both sets of testing creating near straight line graphs.

7.5.4 SQL Azure Throughput In

The inbound throughput graph for SQL Azure is shown in Illustration 23. The legacy application had an average of 244kb per load test. The re-factored application was significantly higher at 472kb.

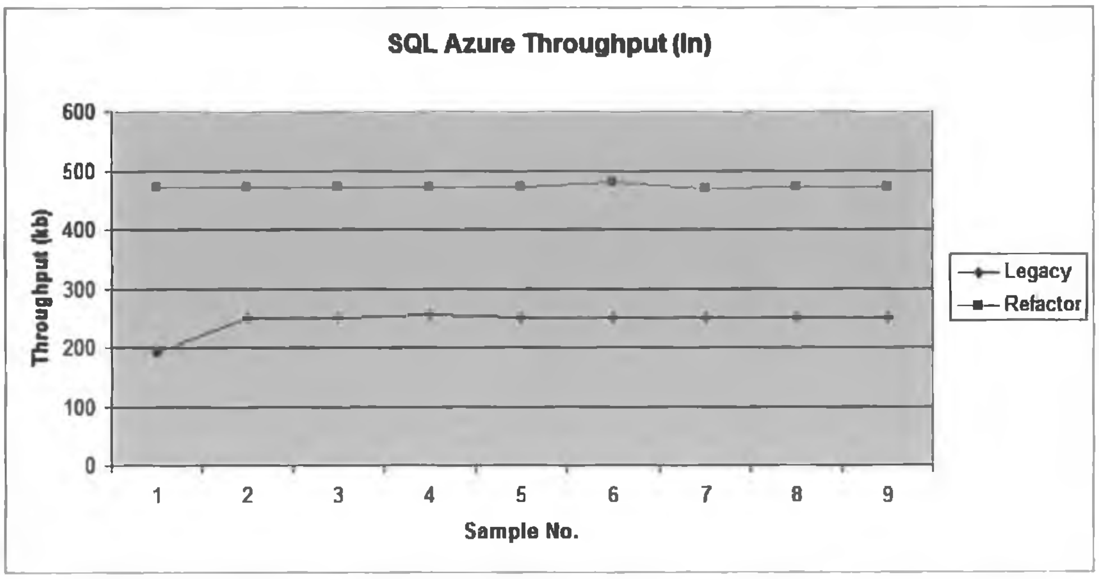


Illustration 23: SQL Azure Throughput In

This figure is the data size of the requests made to the database. These are select statements generated in the repository layer to get information relating to the product catalogue. The difference observed between the two figures can be attributed to the effectiveness of the database repository. The re-factored application implemented the Query Object pattern and this may have let to inefficient query generation. For the SQL Azure throughput inbound the legacy application was nearly twice as efficient as the re-factored application.

7.5.5 Client Data Traffic

The client download volume was recorded as a baseline to compare to the figures recorded from Windows Azure. The legacy application had an average download volume of 20.68MB and the re-factored application averaged 21.73MB.

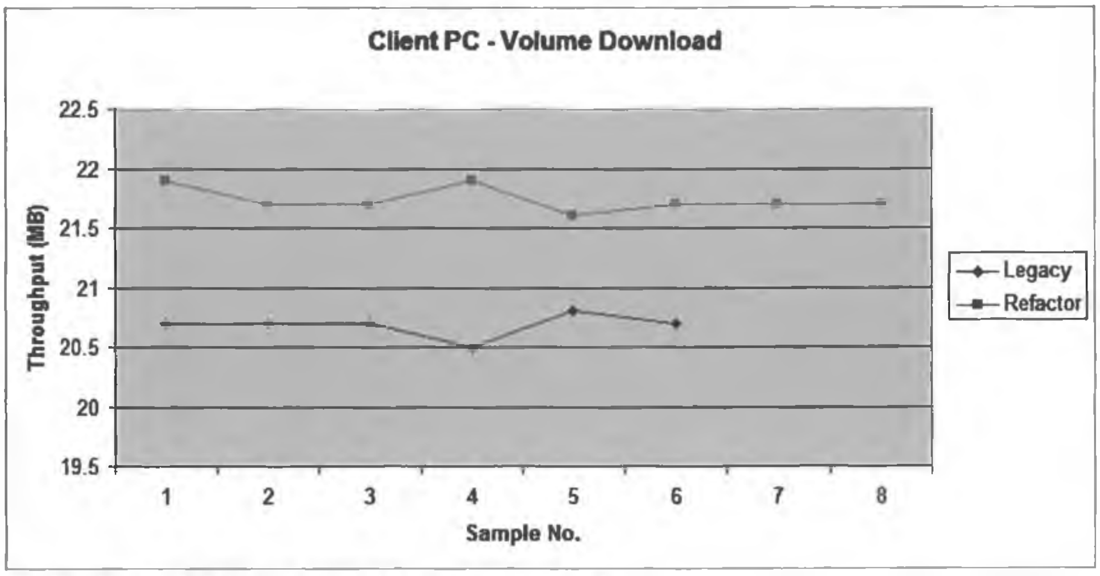


Illustration 24: Client PC - Volume Download

Comparing these figures to the values of the Windows Azure Throughput Out the throughput figures nearly match exactly.

The client volume upload graph is shown in Illustration 25. The legacy application had an average of 1.7MB and the re-factored application had an average of 1.5MB.

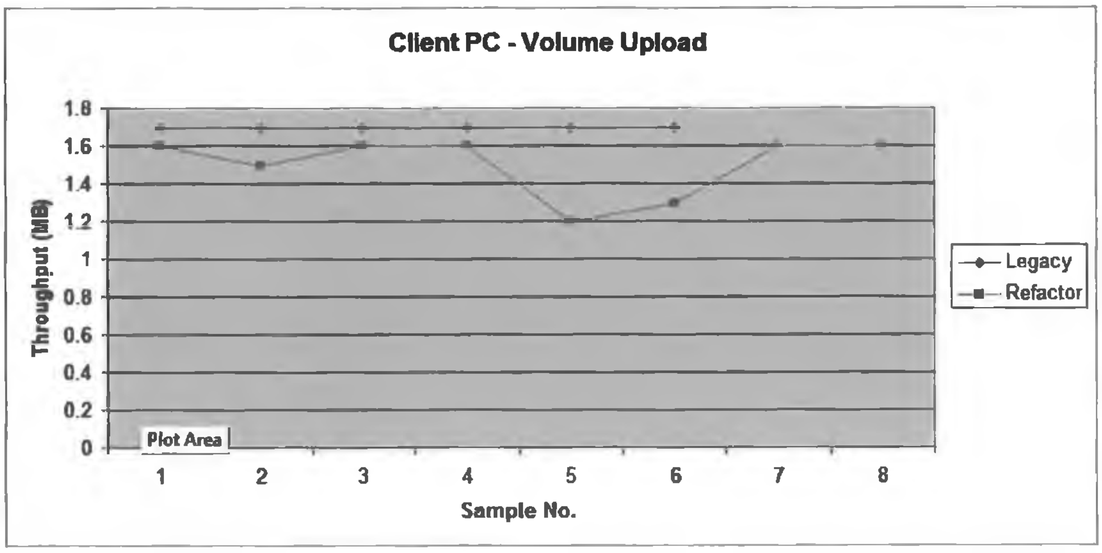


Illustration 25: Client PC - Volume Upload

The upload volume of the Client PC was very similar to the Windows Azure Throughput figures. The throughput (inbound and outbound) should match the upload and download volume of the Client PC. Windows Azure generates throughput outbound to meet the client's requests. This will be mirrored on the client side, with the client receiving this amount of data for the requests made. This test was used to ensure both Azure and the Client PC were operating correctly and no anomalies occurred.

7.6. Analysis of Results

From the analysis of the results in Section 7.5 the re-factored application was not as efficient when compared with the legacy application. The two main areas of concentration were the Windows Azure throughput and the SQL Azure throughput. The Windows Azure throughput is a measure of how effective the application is at receiving requests and issuing responses. As this research is only interested in cloud metric consumption no comparisons were made on application performance. The data in Appendix A is available to show the performance of both applications.

The re-factored application introduced a domain driven design model, service layer, and improved infrastructure to organise the application. The re-factored application also upgraded the repository layer to deal with database requests. This used the Unit of Work pattern, the Query Object pattern, and NHibernate with the built in Identity Map functionality. The changes introduced complete separation of concerns a well layered application.

The results of the tests show the re-factored application is not as efficient as the legacy application. The throughput out is higher after the re-factor. The throughput inbound is better, this can be attributed to the service layer and the messaging system used. The throughput out may be contributed to the strongly typed view-models, but the same images, CSS, and HTML was used for both applications.

The metrics from SQL Azure are the most contrasting of all the data gathered. The legacy application was twice as efficient as the re-factored application. The re-factored application uses Unit of Work, Query Object, and NHibernate in a complete re-code of the repository. The legacy application used Linq to SQL with one to one mapping, with a little grey logic in the controller classes.

8. Conclusion

This chapter provides an overall perspective of the research conducted in this project and summarises its main contributions.

8.1. Overview

The services offered with Cloud Computing has allowed businesses and organisations host their applications and servers in the cloud on a pay-per-use model. Using this model eliminates the initial up front costs of investing heavily on servers, software and equipment needed to run the IT infrastructure. With the cloud operating on a pay-per-use model, the more web traffic a website receives, the more the company will have to pay their cloud service provider.

The design and efficiency of an application is an important factor when using the cloud model, as services are provided on a pay-per-use basis. If an application has been designed or implemented inefficiently it may potentially cost more to run this application on the cloud. There are numerous ways to fine-tune applications to run more efficiently making good use of software design principles and coding practices. This research investigated the possibility of re-factoring an existing enterprise application to try minimise resource consumption on the Windows Azure Platform.

8.2. Research Definition & Research Overview

This research project investigated the effect design patterns have on consumption metrics for an existing ASP.NET enterprise application running on the Windows Azure Platform. An existing enterprise application was re-factored using Fowler's design patterns to introduce solid design principles and try minimise resource consumption.

The project started with a comprehensive literature review, researching cloud computing and design patterns in detail. The exact definition of cloud computing was examined, trying to find out what cloud computing really is. The benefits of cloud computing was looked at showing how this model could be used by businesses to minimise investment costs in quickly depreciating fixed assets. The main components of a cloud environment were discusses explaining how the cloud characteristics can be utilised by businesses on a pay-per-use basis. The concept of design patterns was introduced showing how a design pattern can offer a solution to software design problems that occur frequently in real world application development. The history of design patterns was examined showing how Christopher Alexander introduced to concept of design patterns from an architectural perspective. The work of the Gang of Four in the area of design patterns was addressed by introducing the essential elements and makeup of a design pattern. Martin Fowler's patterns of enterprise design were briefly introduced showing how they will be used to re-factor the existing legacy application.

Following on from the literature review the research background explained the research question and the main components involved in this research area. Cloud computing operates a pay-per-use model, charging only for the resources consumed. For a business moving their enterprise application to the cloud, the implementation of their existing application may not be best suited this environment. The Windows Azure Platform was discussed in great detail explaining the main components of the platform, the benefits of hosting in the cloud, and a breakdown of the Azure pricing plan. The patterns identified in Fowler's book 'Patterns of Enterprise Application Architecture' were listed giving an overview of how they would be used to re-factor the legacy application.

With the research and background complete the requirements analysis for the enterprise application could begin. The enterprise application was designed from the perspective of a golf store case study. This case study was followed to give a realistic business approach to the architecture, design and implementation of the enterprise application. The existing problem was defined along with the project scope. The functional requirements were documented and UML Use Case diagrams and Class diagrams were created to give a graphical representation to the requirements capture and ensure the system accurately met the business needs.

The next phase in the project was the design and implementation of the legacy application. Chapter 5 shows the layout of the user interface and details the main components used to create the application. The implementation of the application is also discussed explaining the architecture and any design principles used.

Chapter 6 shows how the application was re-factored using Fowler's enterprise design patterns. The patterns identified earlier in this research are discussed in detail, explaining how Fowler intends these patterns should be used. The implementation of the applications used ASP.NET. Scott Millett's book 'Professional ASP.NET Design Patterns' gives examples of Fowler's enterprise design patterns using real world examples. The re-factored application has drawn on the experience contained within this book.

With the two applications completed the migration to the Windows Azure Platform could begin. The applications required specific architectural modification to allow them run on the Azure Platform. The complete process for migrating the applications is listed in Appendix D. With both applications now operational on Azure the testing plan was created. The testing plan consisted of 14 page requests that are common to both the legacy and re-factored applications. A load test was created using Visual Studio Team System which simulated a load of 25 concurrent users and was looped 10 times. All results were recorded and graphed to show the effectiveness of both the legacy application and the re-factored application running on the Windows Azure Platform.

8.3. Contribution to the Body of Knowledge

This research projects primary was to reduce resource consumption on the Windows Azure Platform. With cloud computing becoming a more economically viable option for most businesses and organisations, the possibility of further cost savings could be obtained from the efficient operation of the enterprise application.

The first contribution to this knowledge area was the creation of a migration strategy giving detailed steps to move an existing ASP.NET application to the Azure Platform. The steps involved are comprehensive and should serve as a solid guide for any future migrations.

The testing plan created could be used as a basis to evaluate the performance of an application running on the Azure Platform. It facilitated the inclusion of specific pages allowing certain aspects of the application to be concentrated on. With the testing only using Visual Studio Team Server it is a cost efficient method of testing and it evaluates the performance on the application from the client's perspective. The method of retrieving metrics from the Azure platform was not straightforward or intuitive. The knowledge gained in the process of retrieving and evaluating the Azure data can be used for any application running on the Azure Platform. The strategy for obtaining data metrics from SQL Azure was challenging and this research can be used as a solid base for any future work in this area.

The main contribution to this body of knowledge lies in the considerations for re-factoring an application running on the Azure Platform. Careful consideration must be given to the area in the application targeted for performance and efficiency gains. Tools and techniques that performed well on a Windows stand-alone server may not offer the same performance on a cloud environment. The processes followed in this research could be used to find the optimum performance for any of the layers in the re-factored enterprise application. The process of re-factoring the enterprise application did not yield results from the initial re-design, but it is now structured using a layered approach with excellent separation of concerns. Any layer in the application can now be re-factored without significantly impacting the design and operation of the application. From a perspective of minimising resource consumption over a period of time, re-factoring the application using enterprise architecture and design patterns is an excellent first step.

8.4. Experimentation, Evaluation and Limitation

To compare the efficiency of both applications resource consumption on the Azure Platform, data was recorded from the process as listed in section 8.2. The data gathered was the throughput from the Windows Azure instance and the throughput from SQL Azure. Following the Windows Azure price guide as shown in Appendix E, the throughput of data is one of the resource metrics charged for on a pay-per-use basis. The CPU usage and the benefits of adding additional instances to cater additional loads could not be investigated due to funding issues. The Azure trial currently only offers 750 hours of compute instance time, which is approximately one calendar month. Pushing the testing to include additional instances would have used a large number of compute instance hours. All data was taken from the Windows Azure Management Portal and the SQL Azure master database. All data was tabularised and graphed as shown in Appendix A.

The data recorded from SQL Azure showed the legacy application was nearly twice as efficient as the re-factored application for requesting and retrieving information from the database. The re-factored application generated additional traffic both inbound and outbound. Both enterprise applications had a repository layer. The legacy application used Linq to SQL as its Object Relational Mapper (ORM). The re-factored application used NHibernate as its ORM in conjunction with the Unit of Work pattern and the Query Object pattern. NHibernate was chosen as the ORM for the re-factored application for its recognition in industry as an efficient ORM and its in-built Identity Map functionality. The extra traffic inbound could be attributed to badly structured select statements from the Query Object patterns implementation or NHibernate's method of parsing a SQL query in its intermediary position between the application and the database. The results of the testing were not acted upon to try improve the performance of the re-factored applications repository layer. This was the findings of the research showing that re-factoring an application will not guarantee minimising resource consumption on the Azure Platform. It must be noted that the re-factored applications new architecture leaves it in a position to be easily updated or changed. If additional time was available it would have been possible to make changes to the repository layer to improve its resource consumption on the Azure Platform.

The throughput for the Windows Azure instance was lower for the re-factored application. This is attributed to the service and service cache layers that were introduced to the re-factored application. The service layer sits between the presentation layer and the domain layer providing an interface that will define the application's boundaries and the operations available to the client. For the product catalogue the controllers communicate first with the service cache layer. If a cached result exists this will be returned to the controller and if no result is available the service cache will then communicate with the service layer to retrieve the appropriate result before returning it to the controller. This caching mechanism showed a small improvement in throughput inbound for the re-factored application.

Throughput outbound for the Windows Azure instance was only slightly higher for the re-factored application. The result for this part was expected as both applications are serving the same content. The user interface, CSS, and images all remained the same. The only minor difference is the re-factored application used AJAX with JSON to sort the categories by brand. This added a little extra JavaScript to the re-factored application which can be observed in the graphs for throughput outbound for the Windows Azure instance.

The main limitations encountered would be restrictions on time, there were many different aspects that this project could have investigated. With the results from the initial testing it would have been desirable to test the application across the different layers recording metrics on how each layer was performing. A worker role could have been added to Azure to manage this process. It also would have been desirable to test the different ORM's available to find the most efficient and add it to the solution of the re-factored application. These are discussed in the future work section.

8.5. Future Work

Throughout the process of this research a number of additional research areas were identified. The following sections will explain the concepts of the new research areas and an overview of how this might be approached from a research perspective.

8.5.1 Automate Migration to Azure

To facilitate the research for this project two enterprise applications were migrated to the Windows Azure Platform. The complete process for the migration is listed in Appendix D. After following this process twice it was apparent that the steps involved were common to both applications. A future project could investigate the possibility of automating the migration process by making the appropriate changes to an existing enterprise application to allow it to run on the Windows Azure Platform. This would involve creating an application to manage the entire process as listed in Appendix D. This could be an application that runs locally in conjunction with the Azure SDK and Visual Studio. It could also be offered as Software as a Service (SaaS) running on Windows Azure. This could use a web role and a worker role to automise the migration process. The new application or tool would add a new cloud project to the existing solution. The existing Web UI would be identified and added as the new web role. Any DLL references would be found in the existing project and the migration will cater for them as needed. The new Azure application could be run locally or migrated directly to the Windows Azure Platform.

8.5.2 Azure Metrics Dashboard

After the completion of the research in this project it was apparent the data relating to Windows Azure resource consumption is not readily available or presented in a format for easy examination. As a solution to this problem an Azure application could be created to gather this information for a particular subscription. The solution would run on Windows Azure and use both a web role and a worker role to provide this service. It would be necessary to interface with Azure to get details particular to a customer subscription and perform analysis to present the data in a meaningful manner. This would also incorporate a test analysis to show how the application is performing and any potential issues that may be causing excessive resource consumption or identify poor performance of specific items.

8.5.3 Azure ORM's

From the results observed in this research project it was clearly evident the selection of the ORM is critical for application performance. For any application migration to the Windows Azure Platform it would be beneficial to know which ORM operates most efficiently. A research project could be undertaken to test the efficiencies of a variety of ORM tools to identify the most efficient. This would involve creating a constant repository layer which will implement the different ORM's. This would test the create, read, update, and delete (CRUD) functionality for all ORM's. A load test could be created for specific tests to measure performance. All data would be tabularised and graphed to identify the best ORM that may be used in enterprise applications running on the Azure cloud.

8.5.4 Azure Application Analysis

With the pay-per-use model on the Azure cloud it is vital to identify any possible inefficiencies in the application. This new research proposes the creation of an application to analyse the operation of an existing enterprise application running on the Windows Azure Platform. The existing application would be tested to find the appropriate instance size based on projected traffic volume, identify inefficiencies in the application and generate a recommendation report. The research would automate the above and would run on the Windows Azure Platform. It would be provided as an application to be hosted using the customers existing Azure subscription. It would use a web and worker role to provide the testing and analysis.

The above examples for future work were identified throughout the lifeline of this project. The potential for research ideas in the cloud environment is extensive. With cloud development moving at a fast pace the opportunity to find a novel solution to a real world problem is a distinct possibility.

8.6. Conclusion

The main objective of this research was to measure the effect re-factoring an application using design patterns has on cloud resource consumption. The changes made to the re-factored application offered minimal improvements on the throughput inbound for the Windows Azure instance and the results for throughput outbound performed as expected. The most interesting result came from the re-factoring of the repository layer. The changes implemented in the repository layer doubled the throughput consumed by the re-factored application. This extra traffic inbound could be attributed to badly structured select statements from the Query Object patterns implementation or NHibernate's method of parsing a SQL query in its intermediary position between the application and the database. Re-factoring an application will not guarantee the application will reduce resource consumption on the Windows Azure Platform. But this does not rule out re-factoring an application using enterprise design patterns and architecture. For a business or organisation with a long term commitment to reducing resource consumption on Azure the applications re-factor as performed in this research leaves it in position to perform further re-factors on a layered basis. Layering the application allows developers concentrate on a specific area of the application without having to re-code the majority of the application. While the data gathered was not the desired result it must be concluded that re-factoring an application using enterprise design patterns and architecture will eventually produce the desired result.

References

Alexander, C. et al., 1977. *A pattern language*, Oxford Univ. Pr.

Armbrust, M. et al., 2009. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28.*

AutoMapper, 2011. AutoMapper. Available at: http://automapper.codeplex.com/ [Accessed August 16, 2011].

Azure Platform Bills, 2011. Usage Charge Details for Windows Azure Platform Bills BPOS Standard. Available at: http://www.microsoft.com/online/help/en-us/helphowto/af25ac10-7c47-42dc-b139-dab954ed2eff.htm [Accessed August 19, 2011].

Azure Pricing, 2011. Windows Azure Platform Offer. Available at: http://www.microsoft.com/windowsazure/offers/popup/popup.aspx?lang=en&locale=en-us&offer=MS-AZR-0003P [Accessed August 5, 2011].

Azure SDK, 2011. Windows Azure SDK and Tools | Windows Azure Platform. Available at: http://www.microsoft.com/windowsazure/sdk/ [Accessed August 25, 2011].

Barr, J., 2010. Host Your Web Site In The Cloud: Amazon Web Services Made Easy Amazon EC2 Made Easy.

Bojanova, I. & Samba, A., 2011. Analysis of Cloud Computing Delivery Architecture Models. , p.453 - 458.

Codegod, 2006. NHibernate Tutorial (1) - and ASP.NET. Available at: http://www.codegod.biz/WebAppCodeGod/NHibernate-Tutorial-1---and-aeSP-NET-AID25.aspx [Accessed August 29, 2011].

Domain Language, Inc., 2011. What is Domain-Driven Design? | Domain-Driven Design Community. Available at: http://domaindrivendesign.org/resources/what_is_ddd [Accessed August 15, 2011].

Fowler, M., 2005. PATTRON: Layer Supertype. Available at: http://www.pattron.net/Pattern.aspx?Id=67 [Accessed August 15, 2011].

Fowler, M., 2003a. Patterns [software patterns]. *Software, IEEE*, 20(2), p.56-57.

Fowler, M., 2003b. *Patterns of enterprise application architecture*, Addison-Wesley Professional.

Gamma, E. et al., 1995. *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Professional.

Hay, C. & Prince, B., 2010. *Azure in action*, Manning Publications Co.

Kundra, V., 2011. Federal Cloud Computing Strategy.

MSDN, 2011. Windows Azure Platform. Available at: http://msdn.microsoft.com/en-us/library/dd163896.aspx#bk_Platform [Accessed August 25, 2011].

Mell, P. & Grance, T., 2009. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 53(6).

Microsoft ASP.NET, 2011. Home: The Official Microsoft ASP.NET Site. Available at: http://www.asp.net/ [Accessed September 5, 2011].

Microsoft MSDN, 2011a. Introducing SQL Azure Database. Available at: http://msdn.microsoft.com/en-us/library/ee336230.aspx [Accessed August 27, 2011].

Microsoft MSDN, 2011b. SQL Azure Overview. Available at: http://msdn.microsoft.com/en-us/library/ee336241.aspx [Accessed August 27, 2011].

Microsoft MSDN, 2011c. System Views (SQL Azure Database). Available at: http://msdn.microsoft.com/en-us/library/ee336238.aspx [Accessed August 31, 2011].

Microsoft Online Customer Portal, 2011. Microsoft Online Services Customer Portal. Available at: https://mocp.microsoftonline.com/site/default.aspx [Accessed August 31, 2011].

Millett, S., 2010. *Professional ASP NET Design Patterns*, Wrox.

NHibernate, 2011. NHibernate.com. Available at: http://www.nhibernate.com/ [Accessed August 16, 2011].

NIST, 2011. National Institute of Standards and Technology. Available at: http://www.nist.gov/index.html [Accessed June 27, 2011].

Sanderson, S., 2010. *Pro Asp. net MVC 2 Framework*, Springer.

StructureMap, 2011. StructureMap Home Page. Available at: http://structuremap.net/structuremap/ [Accessed August 29, 2011].

Troelsen, A., 2010. *Pro C 2010 and the. NET 4 Platform*, Springer.

Velte, T. et al., 2009. *Cloud computing: a practical approach*, McGraw-Hill Osborne Media.

Wang, L. et al., 2008. Scientific cloud computing: Early definition and experience. In High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on. Ieee, pp. 825-830.

Windows Azure Platform, 2011. Windows Azure Platform | Microsoft Cloud Services. Available at: http://www.microsoft.com/windowsazure/ [Accessed August 5, 2011].

Zhang, L.J. et al., 2010. Hot Topics in Cloud Computing. *IT Professional*, 12(5), p.17-19.

dofactory.com, 2011. .NET Design Patterns in C# and VB.NET - Gang of Four (GOF) - DoFactory. Available at: http://www.dofactory.com/Patterns/Patterns.aspx [Accessed August 12, 2011].

online-crm.com, 2011. Software as a Service Advantages and Disadvantages. Available at: http://www.online-crm.com/saas_advantages_disadvantages.htm [Accessed June 28, 2011].

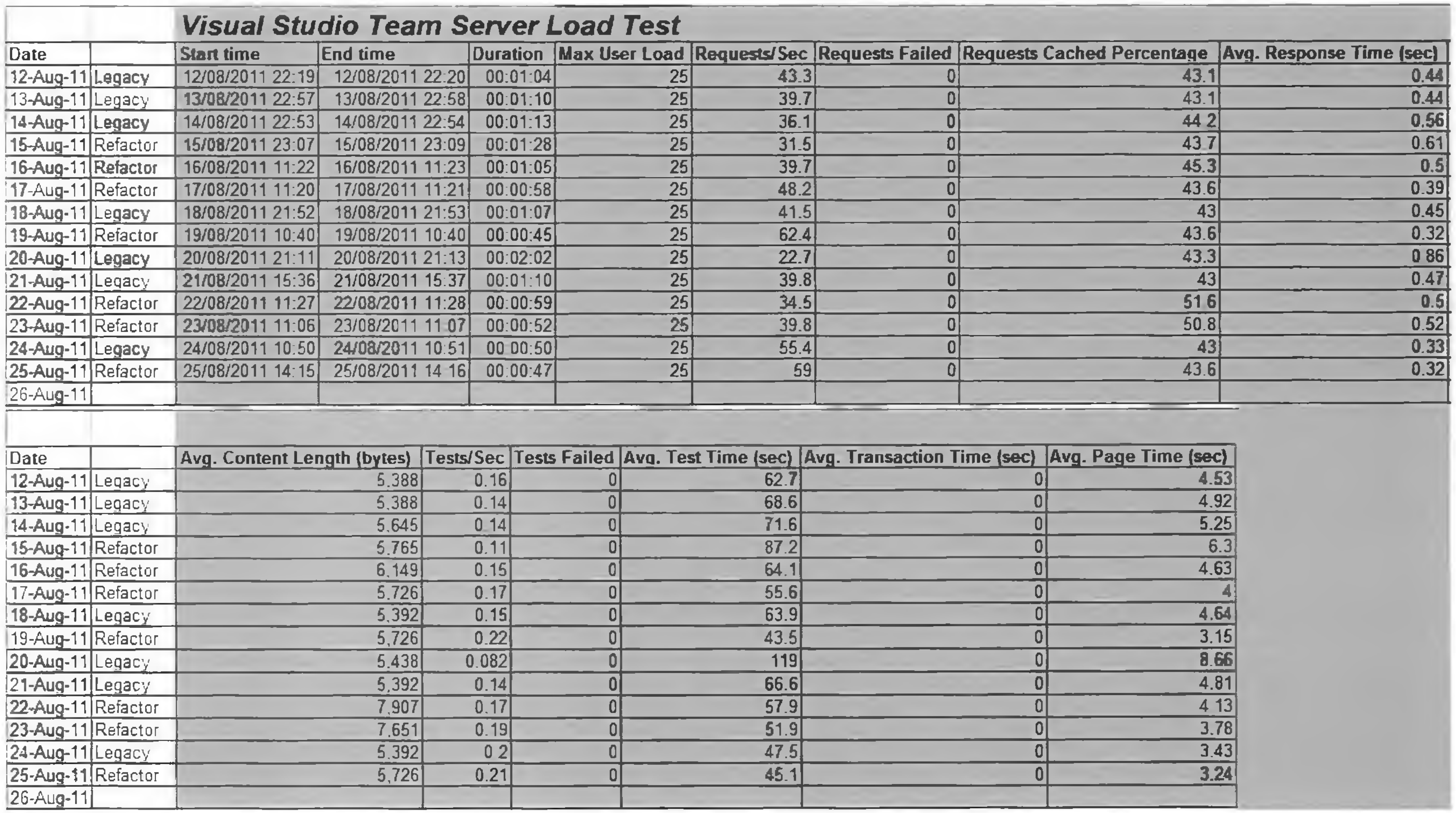
Appendix A. Azure Test Data

Client PC Data

		Three Broadband Data				
Date		Avg Upload Speed (kbps)	Avg Download Speed (kbps)	Vol Upload (MB)	Vol Download (MB)	Duration (m:s)
12-Aug-11	Legacy	76	933	1.7	20.7	3.11
13-Aug-11	Legacy	59	725	1.7	20.7	3.59
14-Aug-11	Legacy	91	1149	1.7	20.7	2.31
15-Aug-11	Refactor	103	1421	1.6	21.9	2.1
16-Aug-11	Refactor	101	1459	1.5	21.7	2.06
17-Aug-11	Refactor	113	1568	1.6	21.7	2.03
18-Aug-11	Legacy	77	948	1.7	20.5	16.01
19-Aug-11	Refactor	120	1698	1.6	21.9	1.49
20-Aug-11	Legacy	59	715	1.7	20.8	4.04
21-Aug-11	Legacy	110	1337	1.7	20.7	2.09
22-Aug-11	Refactor	83	1550	1.2	21.6	2.05
23-Aug-11	Refactor	84	1466	1.3	21.7	2.04
24-Aug-11	Legacy	134	1647	1.7	20.6	1.46
25-Aug-11	Refactor	115	1613	1.6	21.7	1.54
26-Aug-11	Refactor	62	1643	1.6	21.7	1.56

Azure Portal Data

	Database		Azure Metrics	
	SQL Azure In (kb)	SQL Azure Out (kb)	Data Transfer In (MB)	Data Transfer Out (MB)
Legacy	191	1307	1.929	21.161
Legacy	252	1611	1.82	21.279
Refactor	473	2124	1.629	21.436
Refactor	472	2119	1.728	21.617
Legacy	252	1578	1.84	20.242
Refactor	472	2118	1.713	21.566
Legacy	255	1655	1.896	20.729
Legacy	250	1568	1.846	20.519
Refactor	471	2113	1.367	21.373
Refactor	471	2119	1.396	21.387
Legacy	250	1567	1.823	20.346
Refactor	479	2209	1.769	21.648
Refactor	480	2212	No Data	No data
Refactor	470	2114	0.876	21.793
Refactor	471	2114	1.715	21.435
Refactor	471	2114	1.707	21.484
Legacy	53	222	1.93	22.531
Legacy	250	1574	1.907	20.819
Legacy	250	1568	1.391	20.805
Legacy	251	1572	1.882	20.773



Visual Studio Team Server Load Test

Date		Start time	End time	Duration	Max User Load	Requests/Sec	Requests Failed	Requests Cached Percentage	Avg. Response Time (sec)
12-Aug-11	Legacy	12/08/2011 22:19	12/08/2011 22:20	00:01:04	25	43.3	0	43.1	0.44
13-Aug-11	Legacy	13/08/2011 22:57	13/08/2011 22:58	00:01:10	25	39.7	0	43.1	0.44
14-Aug-11	Legacy	14/08/2011 22:53	14/08/2011 22:54	00:01:13	25	36.1	0	44.2	0.56
15-Aug-11	Refactor	15/08/2011 23:07	15/08/2011 23:09	00:01:28	25	31.5	0	43.7	0.61
16-Aug-11	Refactor	16/08/2011 11:22	16/08/2011 11:23	00:01:05	25	39.7	0	45.3	0.5
17-Aug-11	Refactor	17/08/2011 11:20	17/08/2011 11:21	00:00:58	25	48.2	0	43.6	0.39
18-Aug-11	Legacy	18/08/2011 21:52	18/08/2011 21:53	00:01:07	25	41.5	0	43	0.45
19-Aug-11	Refactor	19/08/2011 10:40	19/08/2011 10:40	00:00:45	25	62.4	0	43.6	0.32
20-Aug-11	Legacy	20/08/2011 21:11	20/08/2011 21:13	00:02:02	25	22.7	0	43.3	0.86
21-Aug-11	Legacy	21/08/2011 15:36	21/08/2011 15:37	00:01:10	25	39.8	0	43	0.47
22-Aug-11	Refactor	22/08/2011 11:27	22/08/2011 11:28	00:00:59	25	34.5	0	51.6	0.5
23-Aug-11	Refactor	23/08/2011 11:06	23/08/2011 11:07	00:00:52	25	39.8	0	50.8	0.52
24-Aug-11	Legacy	24/08/2011 10:50	24/08/2011 10:51	00:00:50	25	55.4	0	43	0.33
25-Aug-11	Refactor	25/08/2011 14:15	25/08/2011 14:16	00:00:47	25	59	0	43.6	0.32
26-Aug-11									

Date		Avg. Content Length (bytes)	Tests/Sec	Tests Failed	Avg. Test Time (sec)	Avg. Transaction Time (sec)	Avg. Page Time (sec)
12-Aug-11	Legacy	5,388	0.16	0	62.7	0	4.53
13-Aug-11	Legacy	5,388	0.14	0	68.6	0	4.92
14-Aug-11	Legacy	5,645	0.14	0	71.6	0	5.25
15-Aug-11	Refactor	5,765	0.11	0	87.2	0	6.3
16-Aug-11	Refactor	6,149	0.15	0	64.1	0	4.63
17-Aug-11	Refactor	5,726	0.17	0	55.6	0	4
18-Aug-11	Legacy	5,392	0.15	0	63.9	0	4.64
19-Aug-11	Refactor	5,726	0.22	0	43.5	0	3.15
20-Aug-11	Legacy	5,438	0.082	0	119	0	8.66
21-Aug-11	Legacy	5,392	0.14	0	66.6	0	4.81
22-Aug-11	Refactor	7,907	0.17	0	57.9	0	4.13
23-Aug-11	Refactor	7,651	0.19	0	51.9	0	3.78
24-Aug-11	Legacy	5,392	0.2	0	47.5	0	3.43
25-Aug-11	Refactor	5,726	0.21	0	45.1	0	3.24
26-Aug-11							

Appendix B. Create Web Test in Visual Studio

How to create a testing plan in Visual Studio Team Server 2008

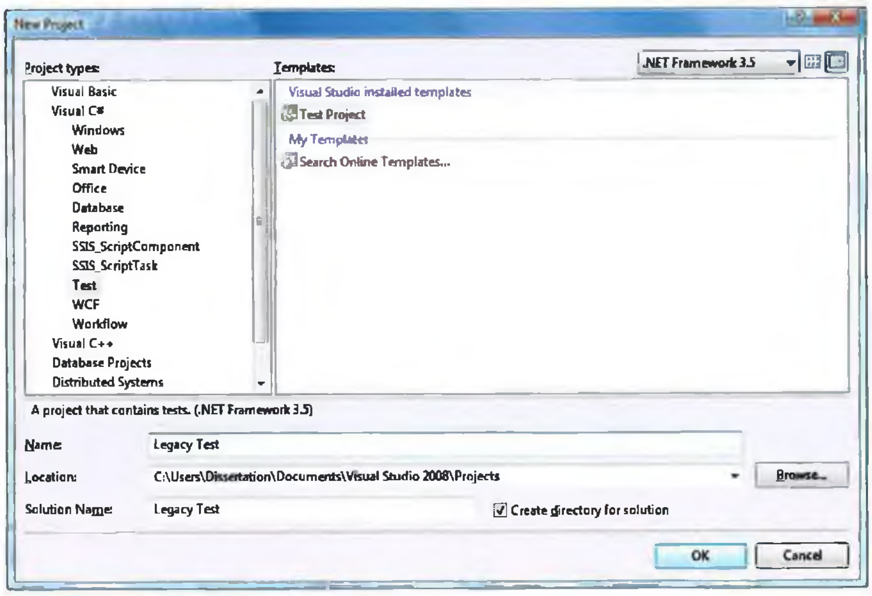
Open Visual Studio 2008

Create a new blank project

File – New Project

From the project types select Test

Then select Test Project from the installed templates on the right hand side.



A project that contains tests. (.NET Framework 3.5)

Name: Legacy Test

Location: C:\Users\Dissertation\Documents\Visual Studio 2008\Projects

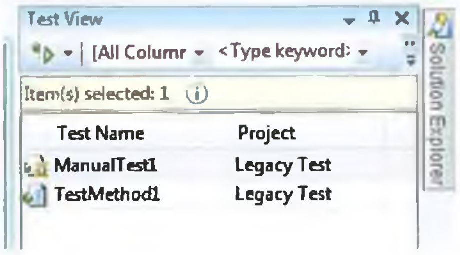
Solution Name: Legacy Test

[✓] Create directory for solution

Click OK

This will load the new project.

There are to manual tests created automatically in the Test View Pane on the right hand site, remove these from the project.

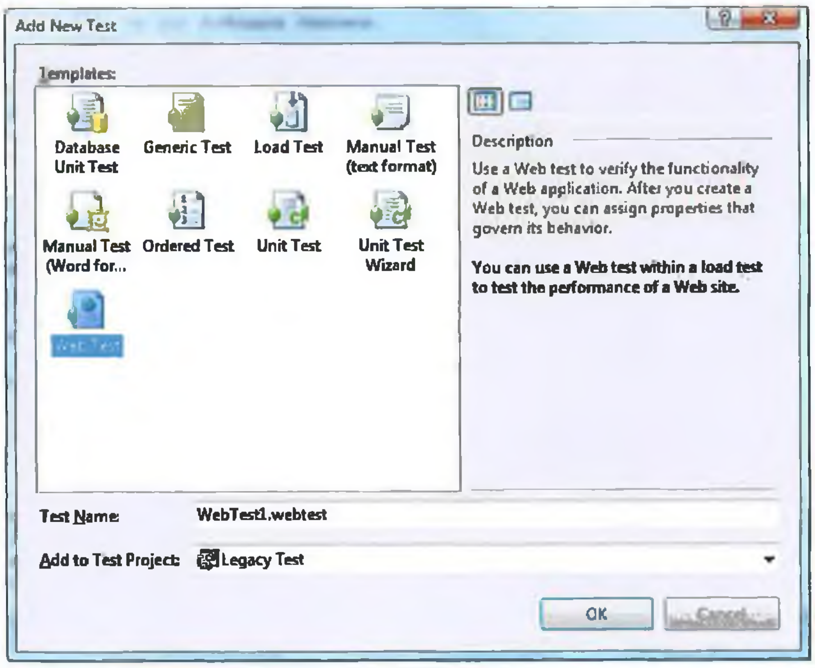


Item(s) selected: 1

Test Name	Project
ManualTest1	Legacy Test
TestMethod1	Legacy Test

The first step is to create the web test containing all the pages to be visited in the web test.

On the toolbar click the New Test button, the following window will appear:



Description

Use a Web test to verify the functionality of a Web application. After you create a Web test, you can assign properties that govern its behavior.

You can use a Web test within a load test to test the performance of a Web site.

Test Name: WebTest1.webtest

Add to Test Project: Legacy Test

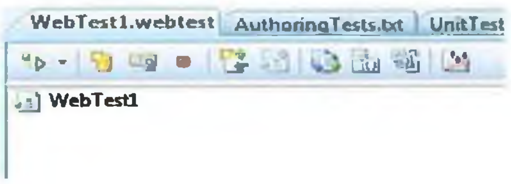
Select Web Test and click OK.

A new browser window will be opened using Internet Explorer.

Close this window as the URL's will be entered manually.

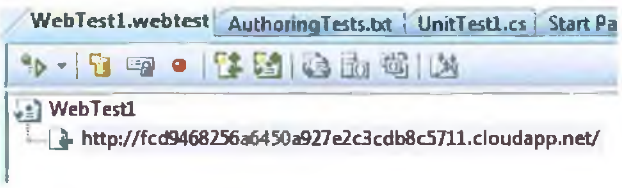
In Visual Studio click stop recording of it is still showing on the top of the window.

The new web test will now be displayed in Visual Studio



The next step is to populate the Web Test Plan with the URL's simulating a customers browsing session. Right click WebTest1 then Add Request

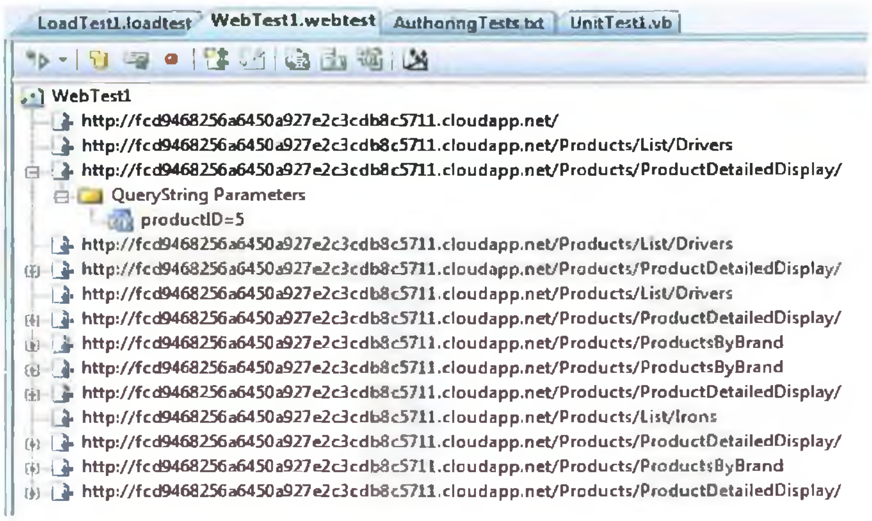
A new request url will appear, this is set to http://localhost/ as default. Click on the new request and in the properties window enter the url specific to the application test.



WebTest1.webtest | AuthoringTests.txt | UnitTest1.cs | Start Pa

```
WebTest1
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/
```

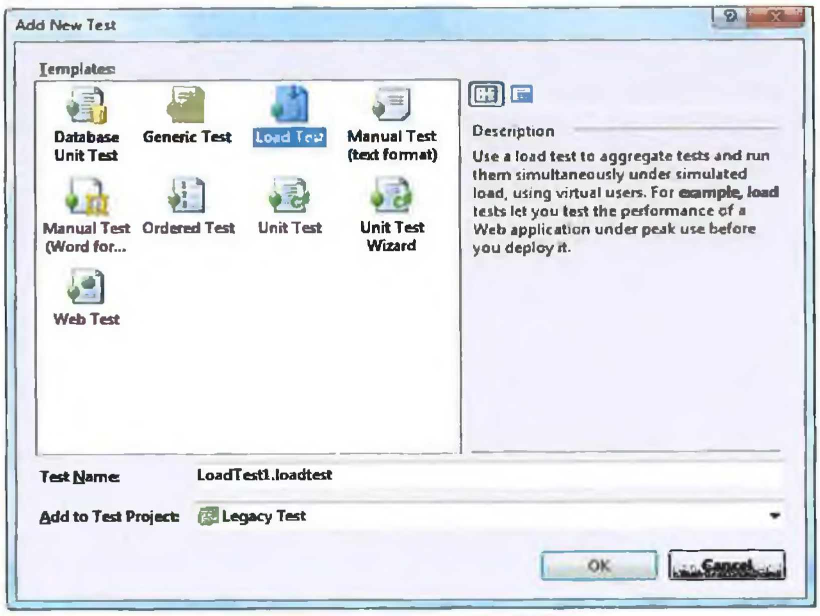
The first url for the web test is now in place, repeat this for all the urls needed to simulate the customers browsing experience.



LoadTest1.loadtest | WebTest1.webtest | AuthoringTests.txt | UnitTest1.vb

```
WebTest1
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/List/Drivers
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/
    QueryString Parameters
      productID=5
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/List/Drivers
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/List/Drivers
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductsByBrand
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductsByBrand
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/List/Irons
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductsByBrand
  http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/
```

When the web test is completed, it is necessary to create a load test. The load test will be used to simulate multiple concurrent customers using the site. This can be run for a specific duration of time or for a set number of iterations. To add a load test click the new test button.



Description

Use a load test to aggregate tests and run them simultaneously under simulated load, using virtual users. For example, load tests let you test the performance of a Web application under peak use before you deploy it.

Test Name: LoadTest1.loadtest

Add to Test Project: Legacy Test

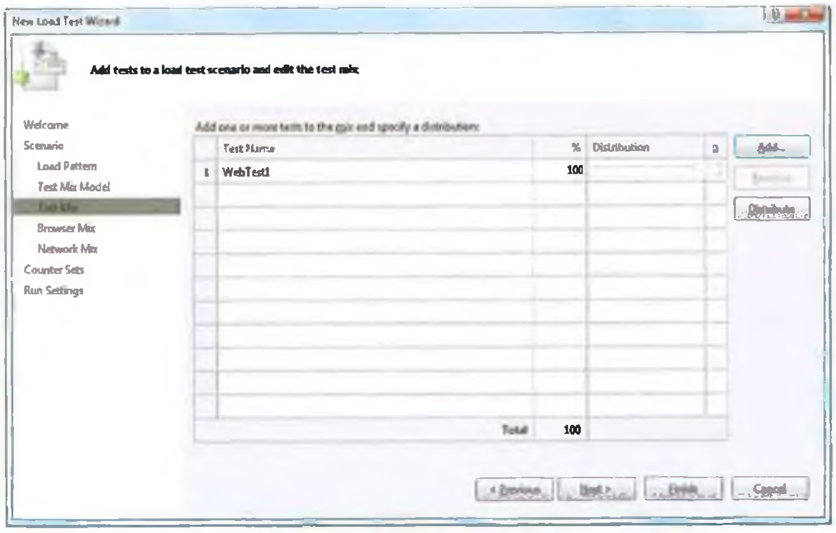
Select the Load Test and click OK.

The New Load Test Wizard will appear.

Click the Test Mix from the menu on the right hand side.

This screen allows the web test created earlier to be added to the load test.

Click the Add button on the right hand side of the screen, and add WebTest! To the load test.



Add tests to a load test scenario and edit the test mix

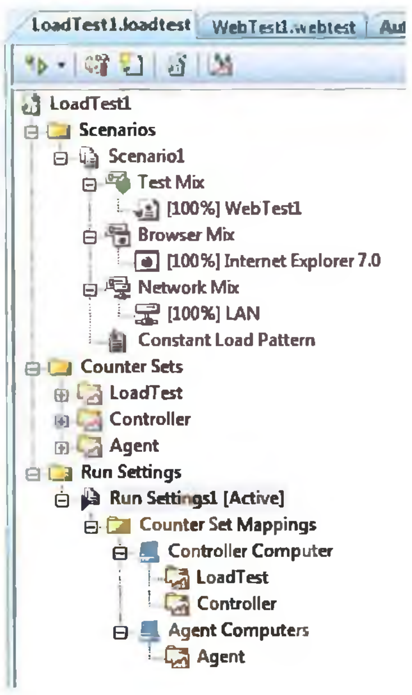
Add one or more tests to the mix and specify a distribution:

	Test Name	%	Distribution
1	WebTest1	100	
	Total	100	

The Wizard should look like figure x.

Click Finish.

The new load test will be shown as follows.



```
LoadTest1
⊟ Scenarios
  ⊟ Scenario1
    ⊟ Test Mix
        [100%] WebTest1
    ⊟ Browser Mix
        [100%] Internet Explorer 7.0
    ⊟ Network Mix
        [100%] LAN
      Constant Load Pattern
⊟ Counter Sets
  ⊞ LoadTest
  ⊞ Controller
  ⊞ Agent
⊟ Run Settings
  ⊟ Run Settings1 [Active]
    ⊟ Counter Set Mappings
      ⊟ Controller Computer
          LoadTest
          Controller
      ⊟ Agent Computers
          Agent
```

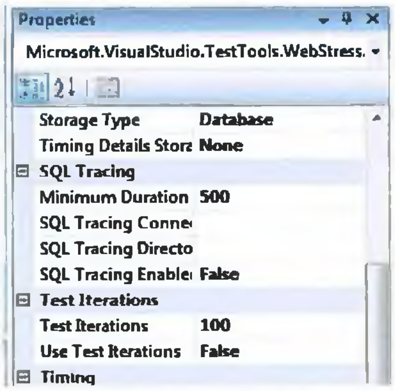
The final step is to configure the run settings for the load test.

Click Run Setting1 [Active]

The run settings will now be shown in the Properties box.

The default setting is to run the test for a specific duration, this needs to be changed to run the test for 10 complete iterations.

In the Test Iterations section, Change Use Test Iterations to True and set the Test Iterations to 10.



Properties

Microsoft.VisualStudio.TestTools.WebStress.

Storage Type	Database
Timing Details Stora	None
⊟ SQL Tracing	
Minimum Duration	500
SQL Tracing Conne	
SQL Tracing Directo	
SQL Tracing Enable	False
⊟ Test Iterations	
Test Iterations	100
Use Test Iterations	False
⊟ Timing	

The run properties for the load test should look as above.

The setup of the application test is now complete. To run the test, click the Run all Tests in Solution button on the main menu. When the test is complete all the results will be displayed on screen.

Appendix C. Test Plan URLs

Legacy App Test Plan URLs

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/List/Drivers

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/?productID=5

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/List/Drivers

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/?productID=7

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/List/Drivers

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/?productID=8

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductsByBrand?brand=Titliest

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductsByBrand?brand=Nike

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/?productID=17

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/List/Irons

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/?productID=10

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductsByBrand?brand=Titliest

http://fcd9468256a6450a927e2c3cdb8c5711.cloudapp.net/Products/ProductDetailedDisplay/?productID=12

Re-Factor App Test Plan URLs

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/GetProductsByCategory?categoryId=1

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/Detail/7

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/GetProductsByCategory?categoryId=1

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/Detail/4

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/GetProductsByCategory?categoryId=1

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/Detail/5

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/GetProductsByCategory?categoryId=3

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/GetProductsByCategory?categoryId=1

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/Detail/17

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/GetProductsByCategory?categoryId=3

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/Detail/12

http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/GetProductsByCategory?categoryId=3 http://65785833381d43d5bfe3ef5c7c238121.cloudapp.net/Product/Detail/8

Appendix D. Migrate an Existing ASP.NET Application to run on Azure

Step 1: Open the existing project in Visual Studio Professional 2010

Step 2: Add a Windows Azure Cloud Service to the solution

Right click solution

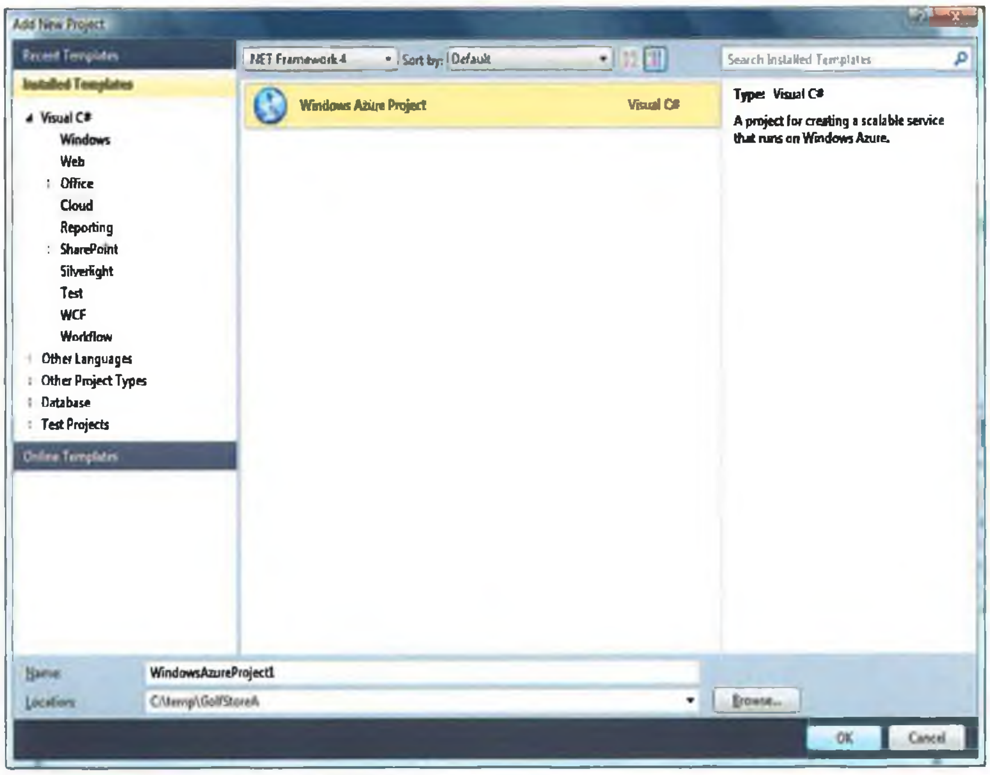
▶ Add

▶ New Project

Under Visual C#

Select "Cloud"

Then select "Windows Azure Project", click OK.



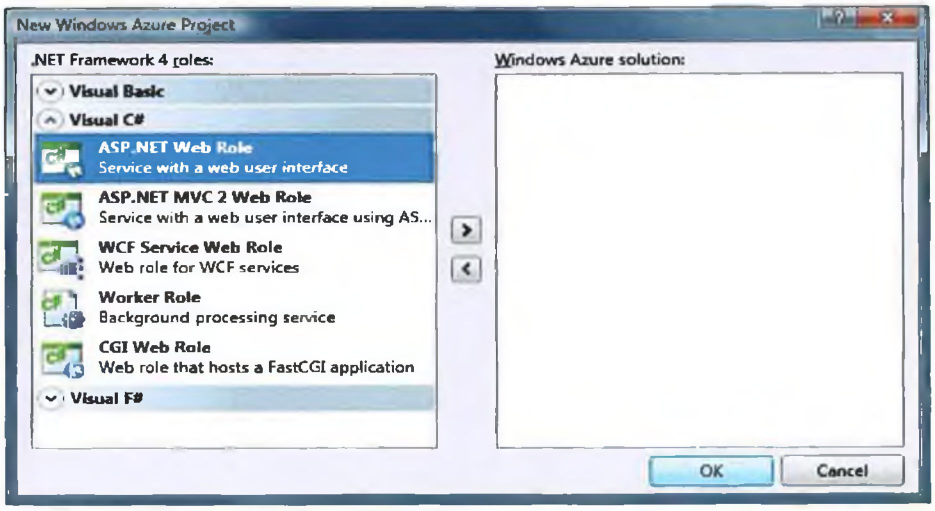
Type: Visual C#

A project for creating a scalable service that runs on Windows Azure.

Name: WindowsAzureProject1

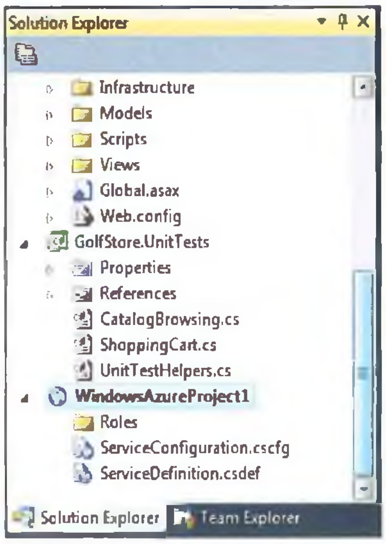
Location: C:\temp\GolfStoreA

This will now show the "New Windows Azure Project"


New Windows Azure Project
.NET Framework 4 roles:
Visual Basic
Visual C#
ASP.NET Web Role
Service with a web user interface
ASP.NET MVC 2 Web Role
Service with a web user interface using AS...
WCF Service Web Role
Web role for WCF services
Worker Role
Background processing service
CGI Web Role
Web role that hosts a FastCGI application
Visual F#
Windows Azure solution:
OK
Cancel

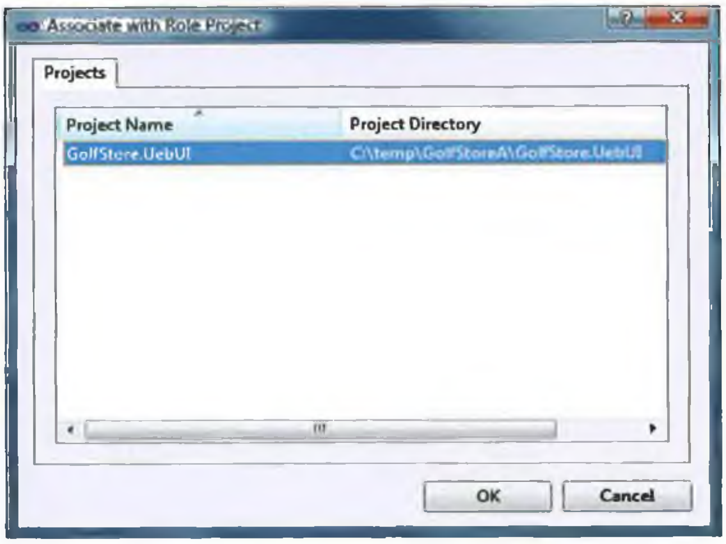
Click OK again, don't add any roles, the existing web project will be used as the Web Role for this solution.

Step 3: In the solution explorer, the new cloud service project is listed



In the solution explorer, right click the Roles folder in the cloud service project.

Select Add | Web Role Project in solution



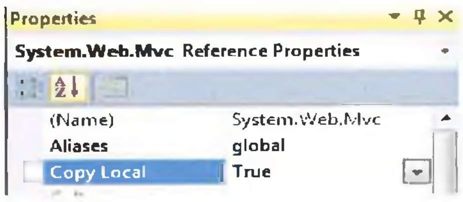
Project Name	Project Directory
GolfStore.WebUI	C:\temp\GolfStoreA\GolfStore.WebUI

All web application projects in the solution will be listed here.

Click on the desired web project.

Step 4: In the case of an MVC web application, the assembly reference to

System.Web.MVC may not have the Copy Local property set to “True”.



(Name)	System.Web.Mvc
Aliases	global
Copy Local	True

This needs to be set to ensure that the System.Web.MVC assembly is available in the cloud. To check this setting highlight System.Web.MVC and check properties for “Copy Local”.

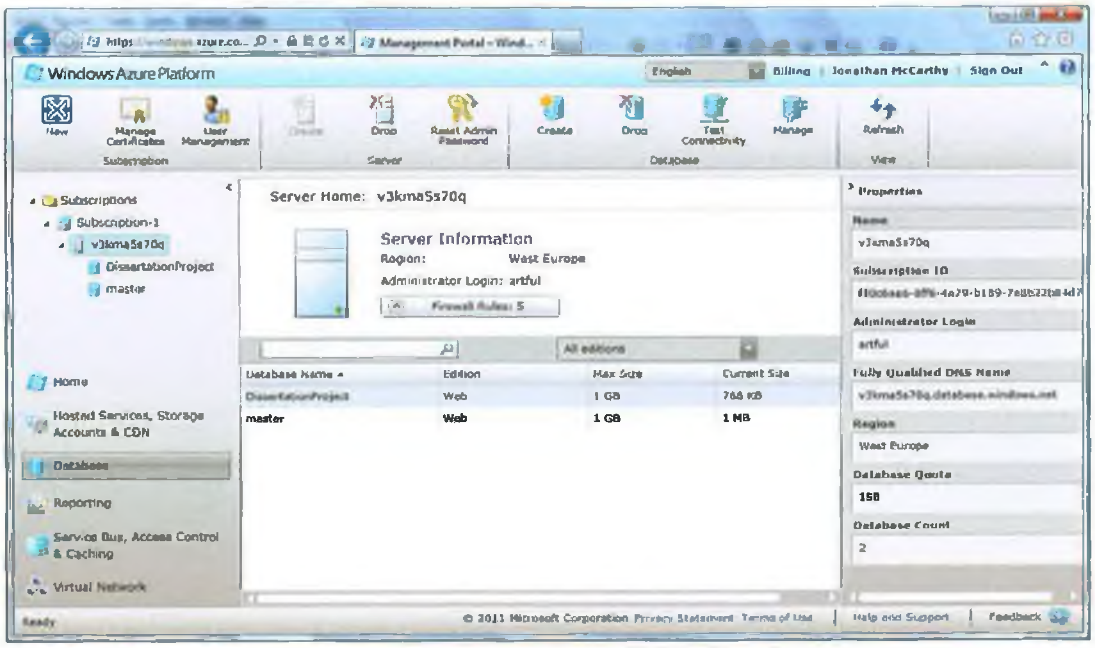
Step 5: Run the Project

The next step is to add any config files to the service package.

Step 6: Any dll files referenced by any projects in the solution need to be referenced in the "New Cloud Service Project"

Step 7: With the application running locally, the next step is to prepare the database to run on the Azure Platform. The database for this application was created using SQL Server Management Studio in the form of a SQL script. This will allow the database to be recreated on SQL Azure.

Login to the Windows Azure Management Portal - https://windows.azure.com/



Server Home: v3kma5s70q

Server Information
Region: West Europe
Administrator Login: artful
Firewall Rules: 5

All editions

Database Name	Edition	Max Size	Current Size
DissertationProject	Web	1 GB	768 KB
master	Web	1 GB	1 MB

Properties
Name: v3kma5s70q
Subscription ID: [illegible]-4a79-b189-7e8b22[illegible]
Administrator Login: artful
Fully Qualified DNS Name: v3kma5s70q.database.windows.net
Region: West Europe
Database Quota: 150
Database Count: 2

Click the Database button in the left menu.

The subscriptions folder will appear, drill down to the database

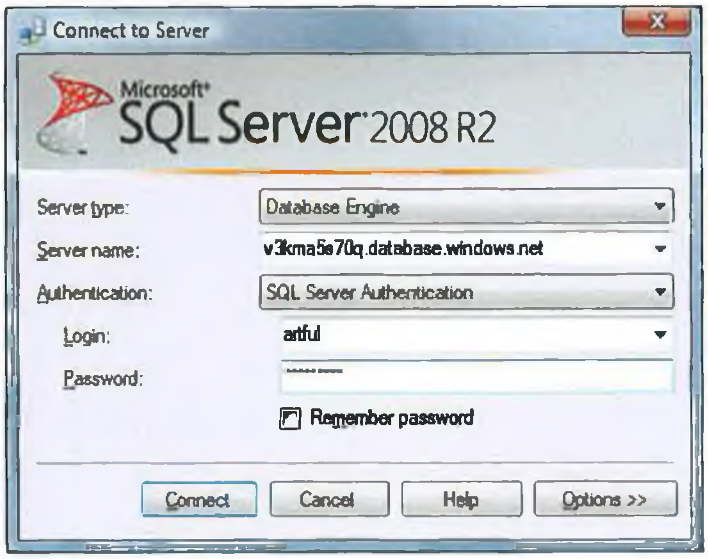
Create a new database named DissertationProject (or any other meaningful name)

Run through the procedure to create a new admin username and password

With the database infrastructure in place the next step is to create the tables and populate them with content.

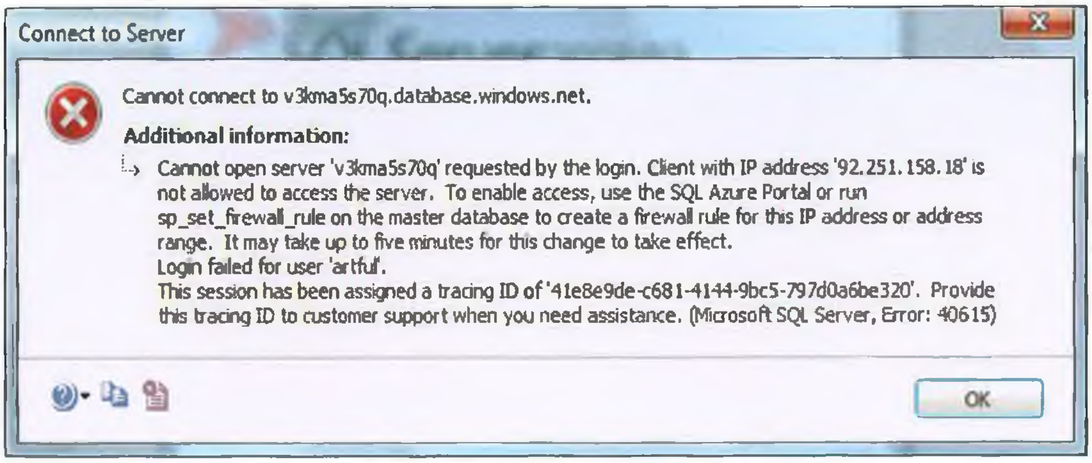
Step 8: Populate the new database with content

Open SQL Server Management Studio



Server type: Database Engine
Server name: v3kma5s70q.database.windows.net
Authentication: SQL Server Authentication
Login: artful
Password: ********
Remember password

Login to SQL Server Management Studio using the credintials for the new SQL Azure database, these are available in the Azure Management Portal as shown in the figure above. The first connection attempt will give an error:

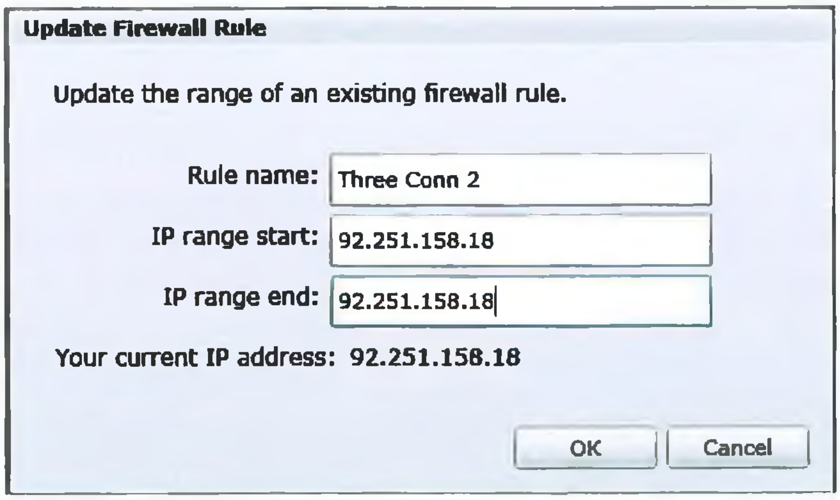


Cannot connect to v3kma5s70q.database.windows.net.

Additional information:

Cannot open server 'v3kma5s70q' requested by the login. Client with IP address '92.251.158.18' is not allowed to access the server. To enable access, use the SQL Azure Portal or run sp_set_firewall_rule on the master database to create a firewall rule for this IP address or address range. It may take up to five minutes for this change to take effect.
Login failed for user 'artful'.
This session has been assigned a tracing ID of '41e8e9de-c681-4144-9bc5-797d0a6be320'. Provide this tracing ID to customer support when you need assistance. (Microsoft SQL Server, Error: 40615)

The SQL Azure database is sitting behind a firewall for security, a firewall exception will need to be added to allow access. In the Management Portal in the database section there is an option to view firewall exceptions, as shown in the figure above.


Update Firewall Rule

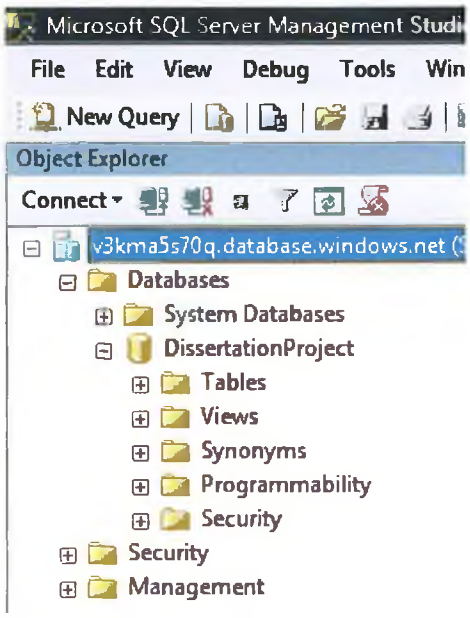
Update the range of an existing firewall rule.

Rule name:	Three Conn 2
IP range start:	92.251.158.18
IP range end:	92.251.158.18

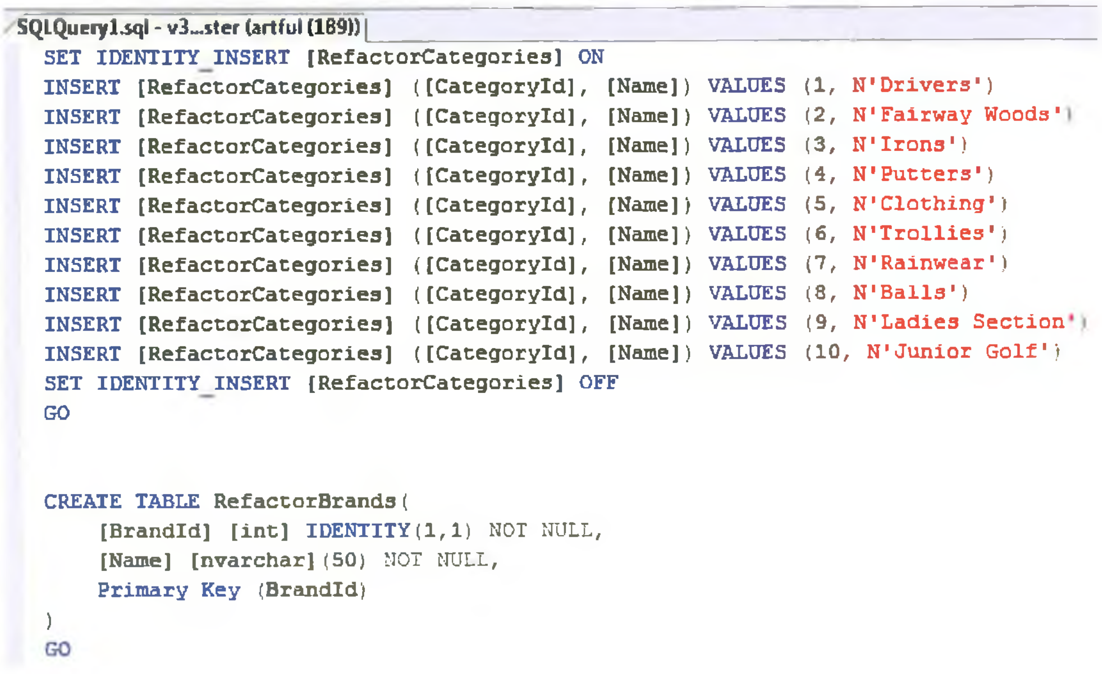
Your current IP address: 92.251.158.18

Add a new firewall exception, the portal automatically displays the current IP address of the PC.

Login again to SQL Server Management Studio using the credentials for the new SQL Azure database. The following will appear.



Open the database script for the application


SQLQuery1.sql - v3...ster (artful (189))

```
SET IDENTITY_INSERT [RefactorCategories] ON
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (1, N'Drivers')
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (2, N'Fairway Woods')
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (3, N'Irons')
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (4, N'Putters')
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (5, N'Clothing')
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (6, N'Trollies')
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (7, N'Rainwear')
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (8, N'Balls')
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (9, N'Ladies Section')
INSERT [RefactorCategories] ([CategoryId], [Name]) VALUES (10, N'Junior Golf')
SET IDENTITY_INSERT [RefactorCategories] OFF
GO


CREATE TABLE RefactorBrands(
    [BrandId] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
    Primary Key (BrandId)
)
GO
```

Run the script, this should pass without any errors.

The SQL Azure database for the application is now in place.

Step 9: Test the application using the new database.

The Management Portal will give the format of the new connection string.

Update the connection string of the application to:

```
<property name="connection.connection_string">
  Server=tcp:v3kma5s70q.database.windows.net,1433;Database=DissertationProject;
  User ID=artful@v3kma5s70q;Password=$$$$$$$;Trusted_Connection=False;Encrypt=True;
</property>
```

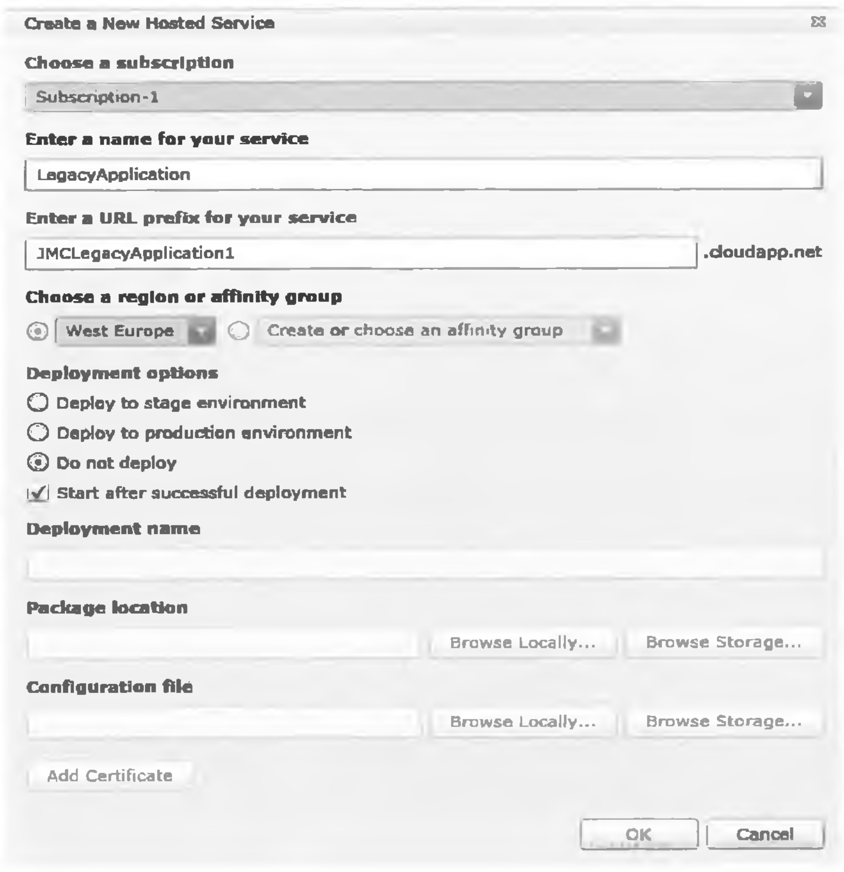
Where $$$$$$$ is the password for the database account!

Test the application locally, if it is functioning correctly the final step in the application migration can be completed.

Step 10: Migrate the Application

Login to the Windows Azure Management Portal

Under Hosted Services, Storage Accounts and CDN, click new Hosted Service.



Create a New Hosted Service

Choose a subscription: Subscription-1

Enter a name for your service: LegacyApplication

Enter a URL prefix for your service: JMCLegacyApplication1 .cloudapp.net

Choose a region or affinity group: [✓ West Europe] Create or choose an affinity group

Deployment options:
Deploy to stage environment
Deploy to production environment
[✓ Do not deploy]
[✓ Start after successful deployment]

Deployment name

Package location

Configuration file

Enter a meaningful name for the application.

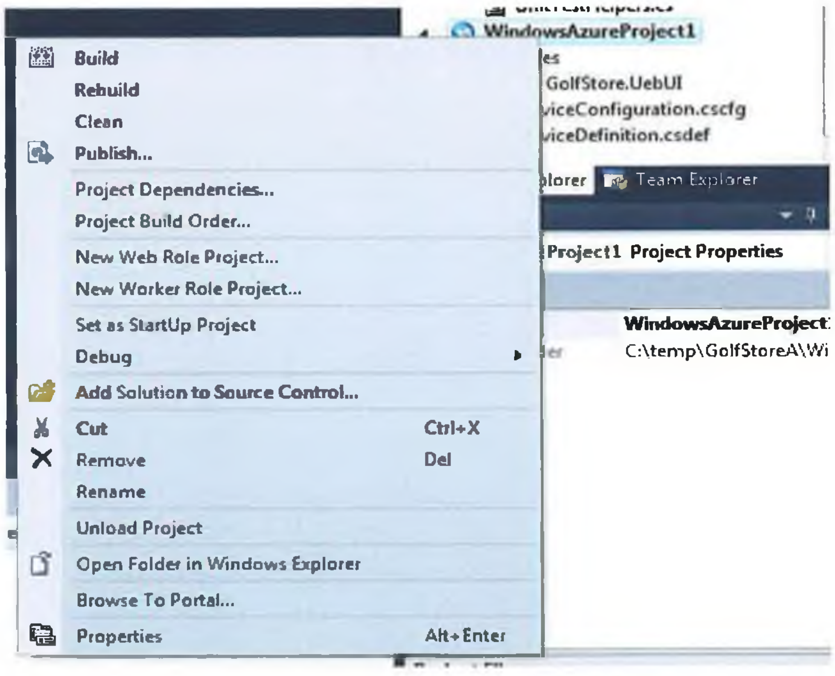
A url prefix is also required, this must be unique.

Select the region, this is important as this will determine your hosting location. There are different tariffs for transfers between regions. Select West Europe.

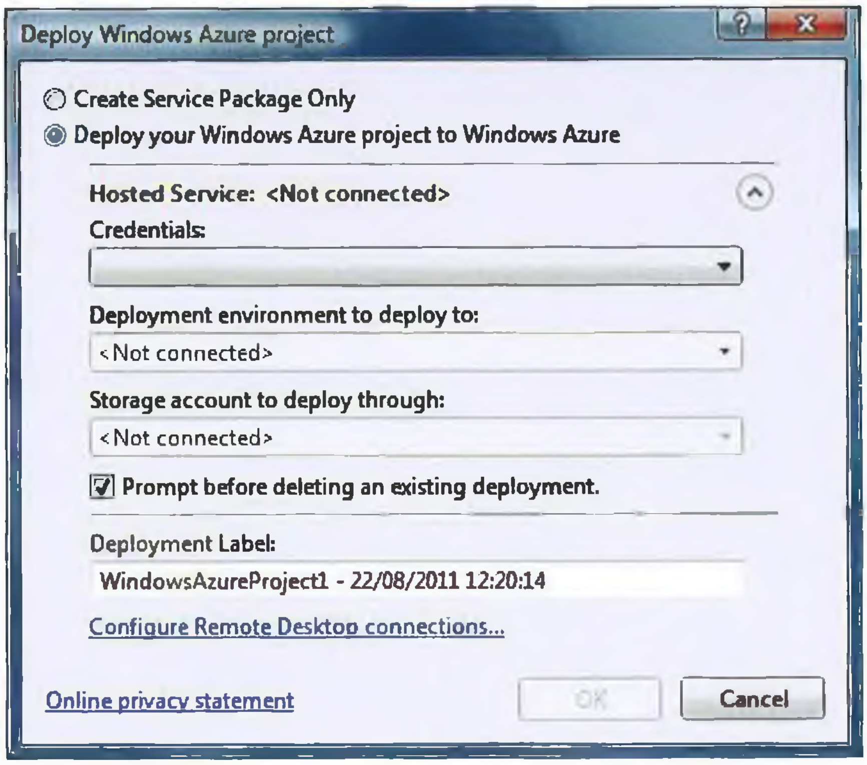
The migration of the application will be managed by Visual Studio, this will be the final step in the migration. So at this point select “Do not deploy”.

Click OK, the new Subscription will be created, this can take a few minutes.

The SDK for Azure allows Visual Studio Manage the applications migration. Right click the new Azure project and click Publish



The following will appear:


Deploy Windows Azure project

○ Create Service Package Only
● Deploy your Windows Azure project to Windows Azure

Hosted Service: <Not connected>

Credentials:

Deployment environment to deploy to: <Not connected>

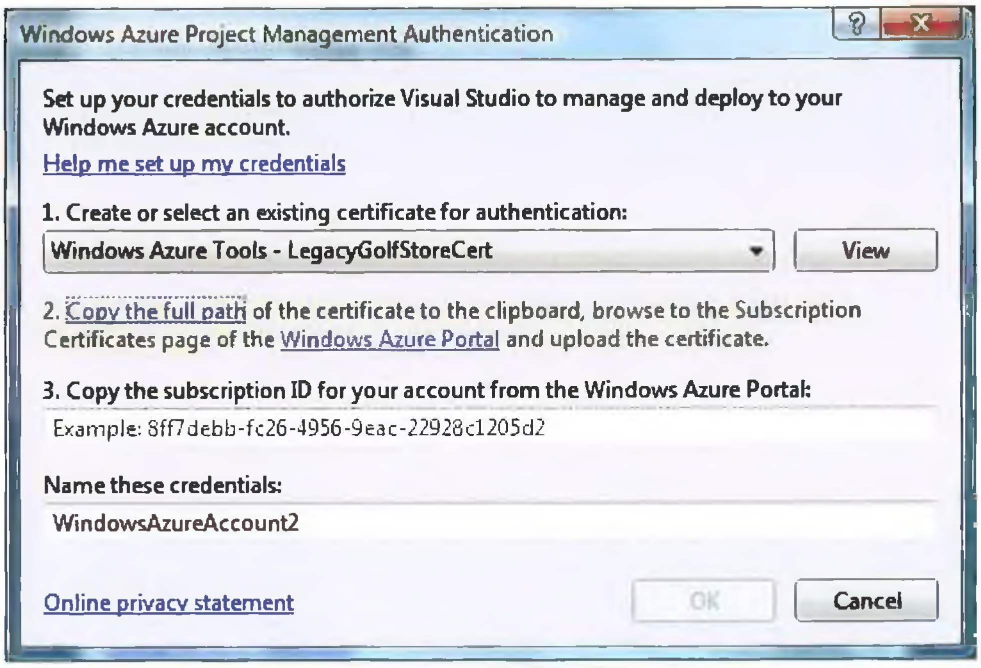
Storage account to deploy through: <Not connected>

☑ Prompt before deleting an existing deployment.

Deployment Label: WindowsAzureProject1 - 22/08/2011 12:20:14

Configure Remote Desktop connections...

In the Credentials section Click Add


Windows Azure Project Management Authentication

Set up your credentials to authorize Visual Studio to manage and deploy to your Windows Azure account.

Help me set up my credentials

1. Create or select an existing certificate for authentication: Windows Azure Tools - LegacyGolfStoreCert

2. Copy the full path of the certificate to the clipboard, browse to the Subscription Certificates page of the Windows Azure Portal and upload the certificate.

3. Copy the subscription ID for your account from the Windows Azure Portal: Example: 8ff7debb-fc26-4956-9eac-22928c1205d2

Name these credentials: WindowsAzureAccount2

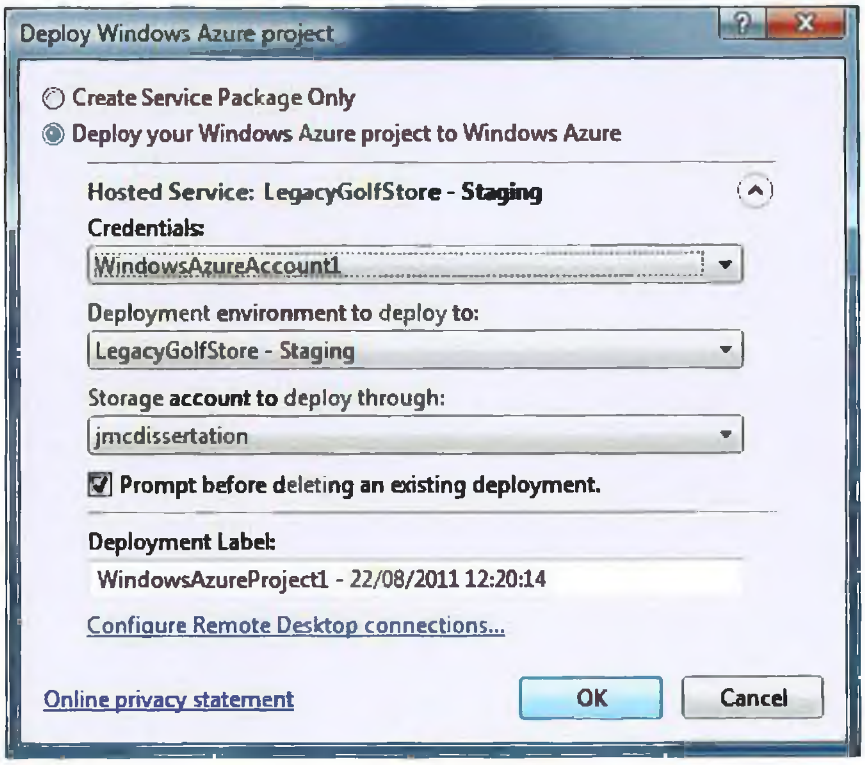
A new certificate needs to be created to associate with the Azure Account

Create a new certificate and copy it to the clipboard

Browse to he Subscription Certificates page of the Windows Azure Portal and upload the certificate

In the Management Portal the Subscription ID will be displayed, copy it and paste it into the Subscription field.

Click OK


Deploy Windows Azure project

○ Create Service Package Only
● Deploy your Windows Azure project to Windows Azure

Hosted Service: LegacyGolfStore - **Staging**

Credentials: WindowsAzureAccount1

Deployment environment to deploy to: LegacyGolfStore - Staging

Storage account to deploy through: jmcdissertation

☑ Prompt before deleting an existing deployment.

Deployment Label: WindowsAzureProject1 - 22/08/2011 12:20:14

Select the Deployment environment, this is the Hosted Service that was created in the Management Portal. Be careful to select the correct one, as it will delete any existing application and replace it with the new one.

Click OK.

The application will begin the transfer over to the Azure Platform.

The progress can be monitored in the Azure Management Portal.

The migration transfer takes approx. 20 mins.

When complete the Azure Management Portal will give the URL of the application.

Open the url in a browser, for the first connection an error will show as a firewall rule is needed for the instance to access the database.

Update the databases firewall rules table as documented in Step x.

The application should not be running fully functional on the Azure Platform.

Appendix E. Azure Pricing Plan

Windows Azure

- Compute [1]
    - Extra small instance: $0.05 per hour [2]
    - Small instance (default): $0.12 per hour
    - Medium instance: $0.24 per hour
    - Large instance: $0.48 per hour
    - Extra large instance: $0.96 per hour
- Virtual Network [3]
    - Windows Azure Connect - No charge during CTP
- Storage
    - $0.15 per GB stored per month
    - $0.01 per 10,000 storage transactions
- Content Delivery Network (CDN)
    - $0.15 per GB for data transfers from European and North American locations
    - $0.20 per GB for data transfers from other locations
    - $0.01 per 10,000 transactions

SQL Azure

- Web Edition
  - $9.99 per database up to 1 GB per month
  - $49.95 per database up to 5 GB per month
- Business Edition
  - $99.99 per database up to 10 GB per month
  - $199.98 per database up to 20 GB per month
  - $299.97 per database up to 30 GB per month
  - $399.96 per database up to 40 GB per month
  - $499.95 per database up to 50 GB per month

AppFabric

- Access Control [4]
  - $1.99 per 100,000 transactions
- Service Bus
  - $3.99 per connection on a 'pay-as-you-go' basis
  - Pack of 5 connections $9.95
  - Pack of 25 connections $49.75
  - Pack of 100 connections $199.00
  - Pack of 500 connections $995.00
- Caching [5]
  - 128 MB cache for $45.00
  - 256 MB cache for $55.00
  - 512 MB cache for $75.00
  - 1 GB cache for $110.00
  - 2 GB cache for $180.00
  - 4 GB cache for $325.00

Data Transfers

- North America and Europe regions
  - $0.15 per GB out
- Asia Pacific region
  - $0.20 per GB out
- All inbound data transfers are at no charge.

Where:

[1] Compute hours are calculated based on the number of hours that your application is deployed. Please refer to the Compute Instances section of this offer for further details.

[2] Extra small compute instances are available in beta and are billed separately from other compute instance sizes.

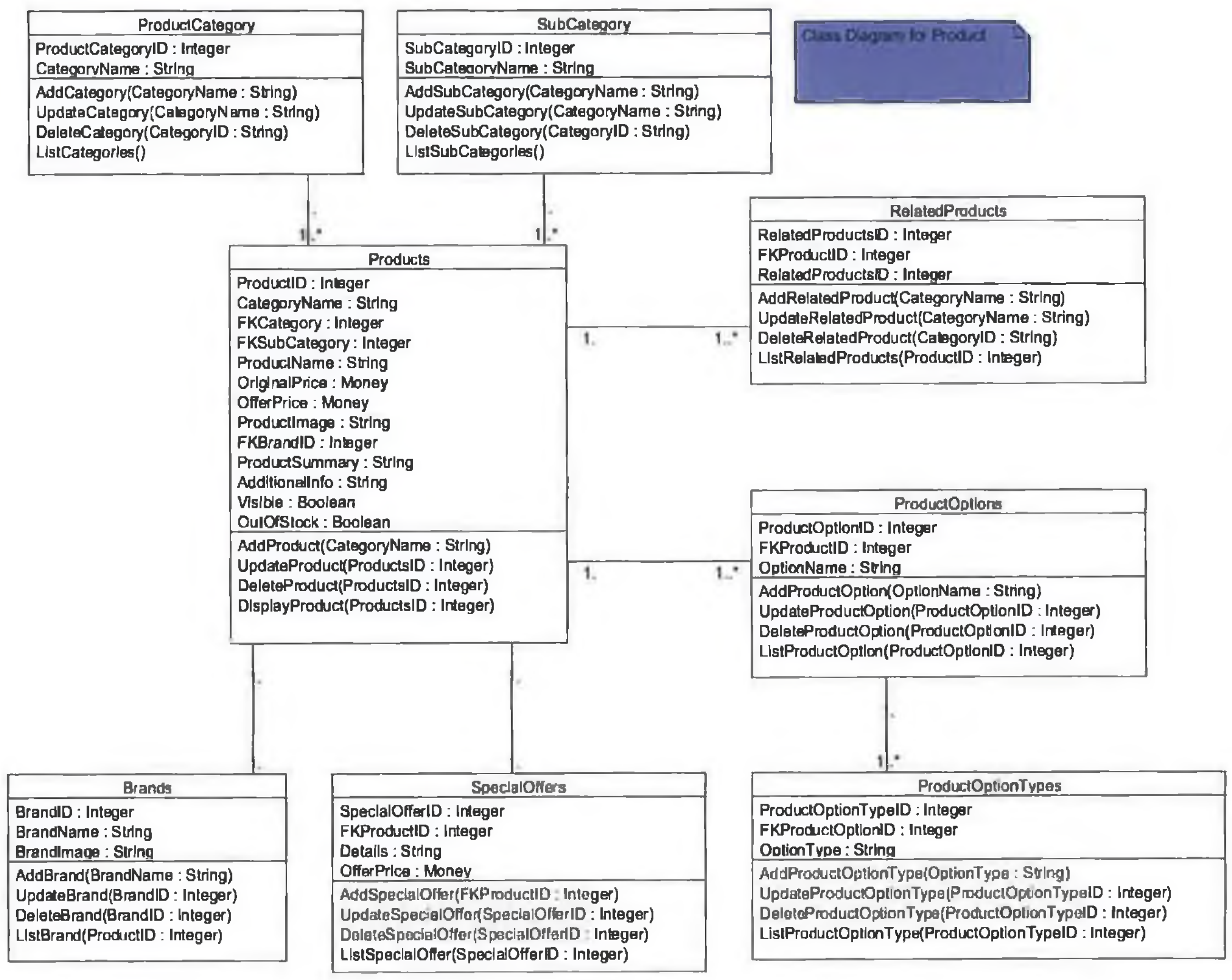
[3] The Windows Azure Connect service is available in Community Technology Preview (CTP).

[4] No charge for billing periods before January 1, 2012.

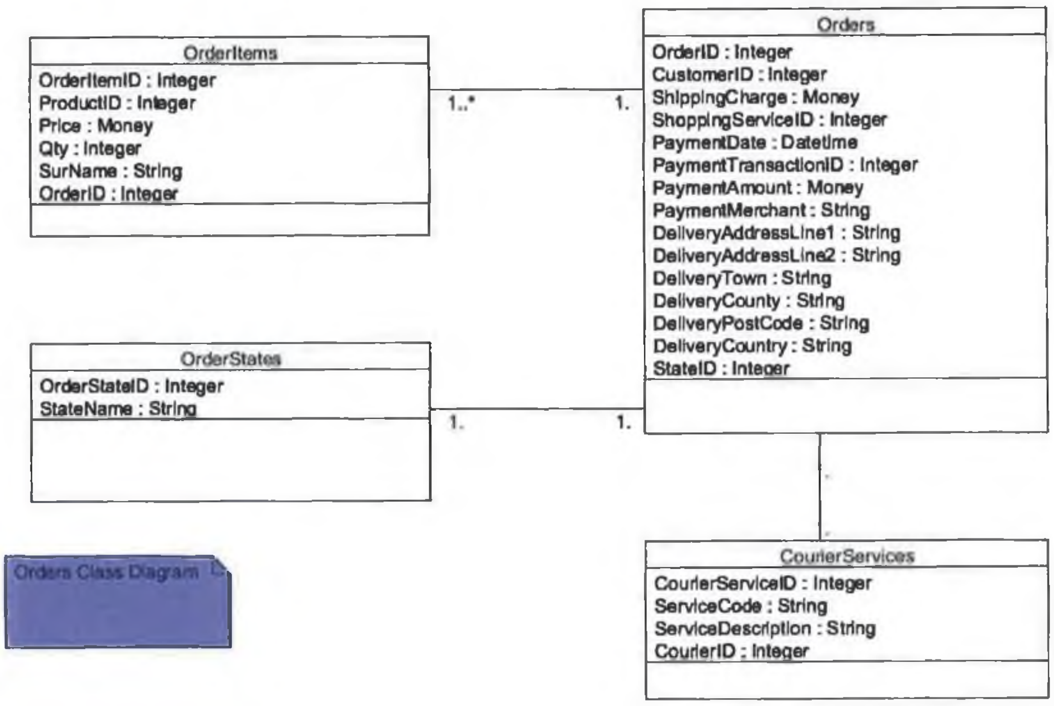
[5] The Windows Azure AppFabric Caching service is provided at no charge for billing periods prior to August 1, 2011.

Appendix F. UML Diagrams

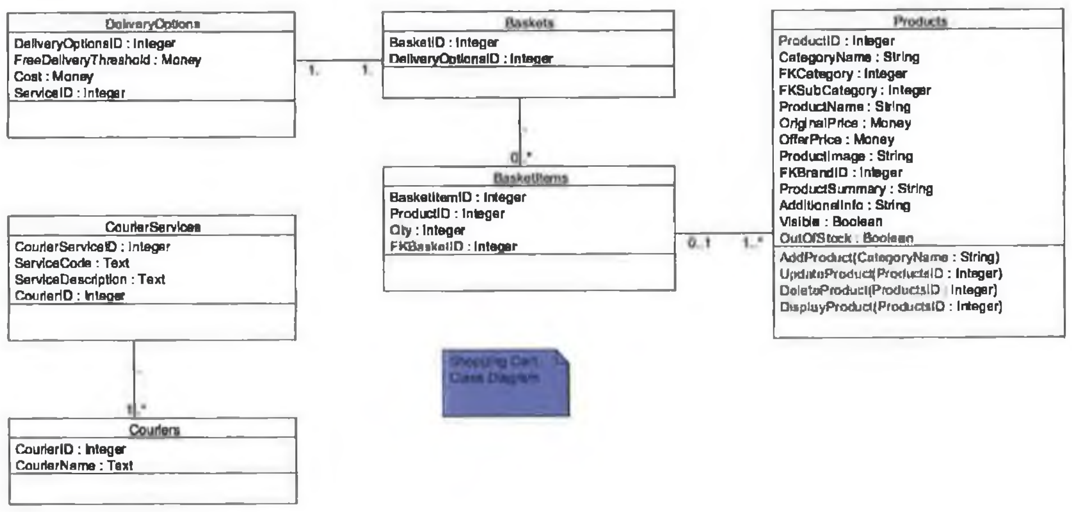
Product Class Diagram


Class Diagram for Product
ProductCategory
ProductCategoryID : Integer
CategoryName : String
AddCategory(CategoryName : String)
UpdateCategory(CategoryName : String)
DeleteCategory(CategoryID : String)
ListCategories()
SubCategory
SubCategoryID : Integer
SubCategoryName : String
AddSubCategory(CategoryName : String)
UpdateSubCategory(CategoryName : String)
DeleteSubCategory(CategoryID : String)
ListSubCategories()
RelatedProducts
RelatedProductsID : Integer
FKProductID : Integer
RelatedProductsID : Integer
AddRelatedProduct(CategoryName : String)
UpdateRelatedProduct(CategoryName : String)
DeleteRelatedProduct(CategoryID : String)
ListRelatedProducts(ProductID : Integer)
Products
ProductID : Integer
CategoryName : String
FKCategory : Integer
FKSubCategory : Integer
ProductName : String
OriginalPrice : Money
OfferPrice : Money
ProductImage : String
FKBrandID : Integer
ProductSummary : String
AdditionalInfo : String
Visible : Boolean
OutOfStock : Boolean
AddProduct(CategoryName : String)
UpdateProduct(ProductsID : Integer)
DeleteProduct(ProductsID : Integer)
DisplayProduct(ProductsID : Integer)
ProductOptions
ProductOptionID : Integer
FKProductID : Integer
OptionName : String
AddProductOption(OptionName : String)
UpdateProductOption(ProductOptionID : Integer)
DeleteProductOption(ProductOptionID : Integer)
ListProductOption(ProductOptionID : Integer)
Brands
BrandID : Integer
BrandName : String
BrandImage : String
AddBrand(BrandName : String)
UpdateBrand(BrandID : Integer)
DeleteBrand(BrandID : Integer)
ListBrand(ProductID : Integer)
SpecialOffers
SpecialOfferID : Integer
FKProductID : Integer
Details : String
OfferPrice : Money
AddSpecialOffer(FKProductID : Integer)
UpdateSpecialOffer(SpecialOfferID : Integer)
DeleteSpecialOffer(SpecialOfferID : Integer)
ListSpecialOffer(SpecialOfferID : Integer)
ProductOptionTypes
ProductOptionTypeID : Integer
FKProductOptionID : Integer
OptionType : String
AddProductOptionType(OptionType : String)
UpdateProductOptionType(ProductOptionTypeID : Integer)
DeleteProductOptionType(ProductOptionTypeID : Integer)
ListProductOptionType(ProductOptionTypeID : Integer)
1..*
1..*
1.
1..*
1.
1..*
1..*

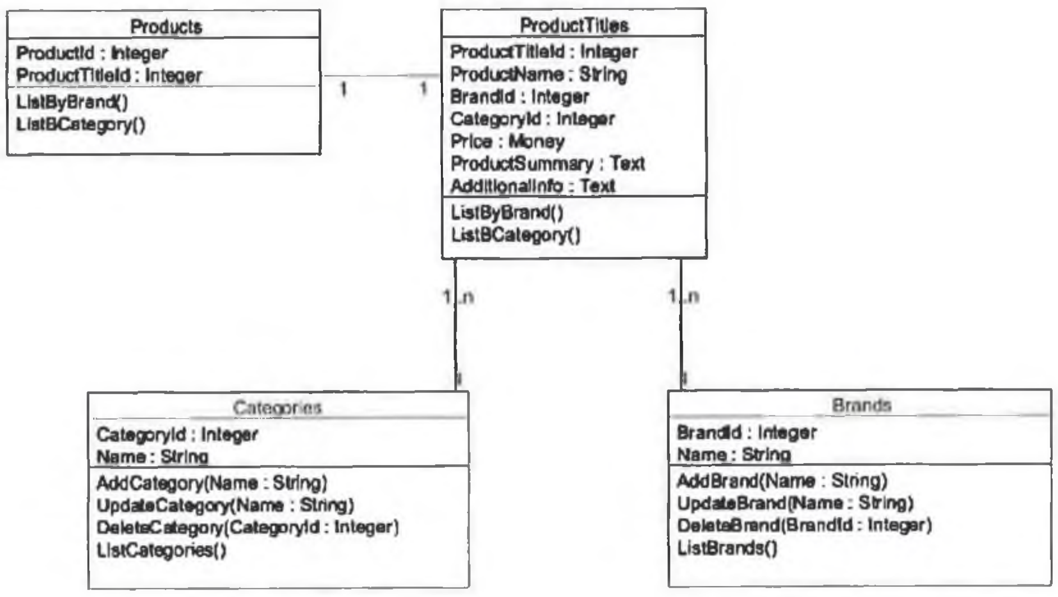
Orders Class Diagram



Shopping Cart Class Diagram

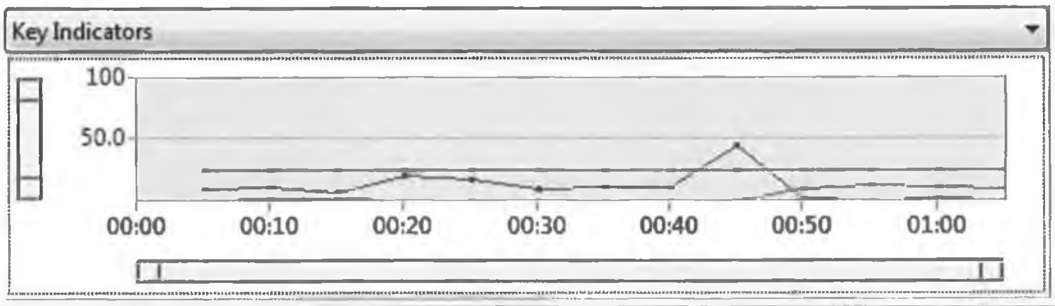


New Product Catalogue Class Diagram



Appendix G. Test Data

12th August 2011 - Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
Key Indicators								
✓ User Load	_Total	LoadTest:Scenario	POSSERVER	—	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	—	1,000	6.20	125	43.3
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	—	10	0.12	4.62	0.44
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	—	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	—	0	0	0	0

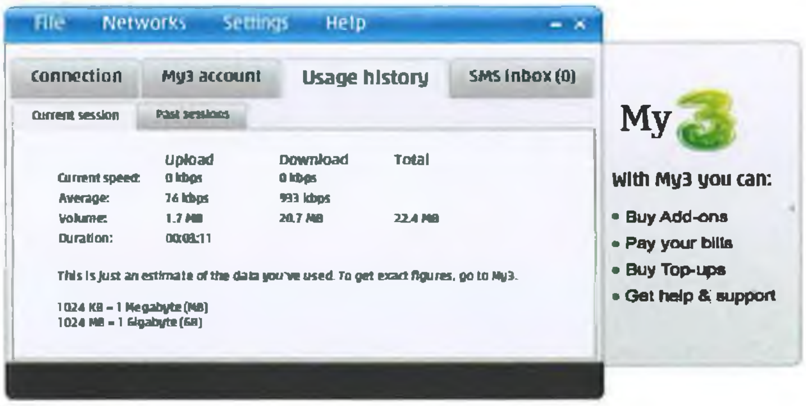
Test Run Information

Load test name	LoadTest1
Description	
Start time	12/08/2011 22:19:01
End time	12/08/2011 22:20:05
Duration	00:01:04
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	43.3
Requests Failed	0
Requests Cached Percentage	43.1
Avg. Response Time (sec)	0.44
Avg. Content Length (bytes)	5,388
Tests/Sec	0.16
Avg. Test Time (sec)	62.7
Avg. Transaction Time (sec)	0
Avg. Page Time (sec)	4.53

Section 2 – Client Test Machine

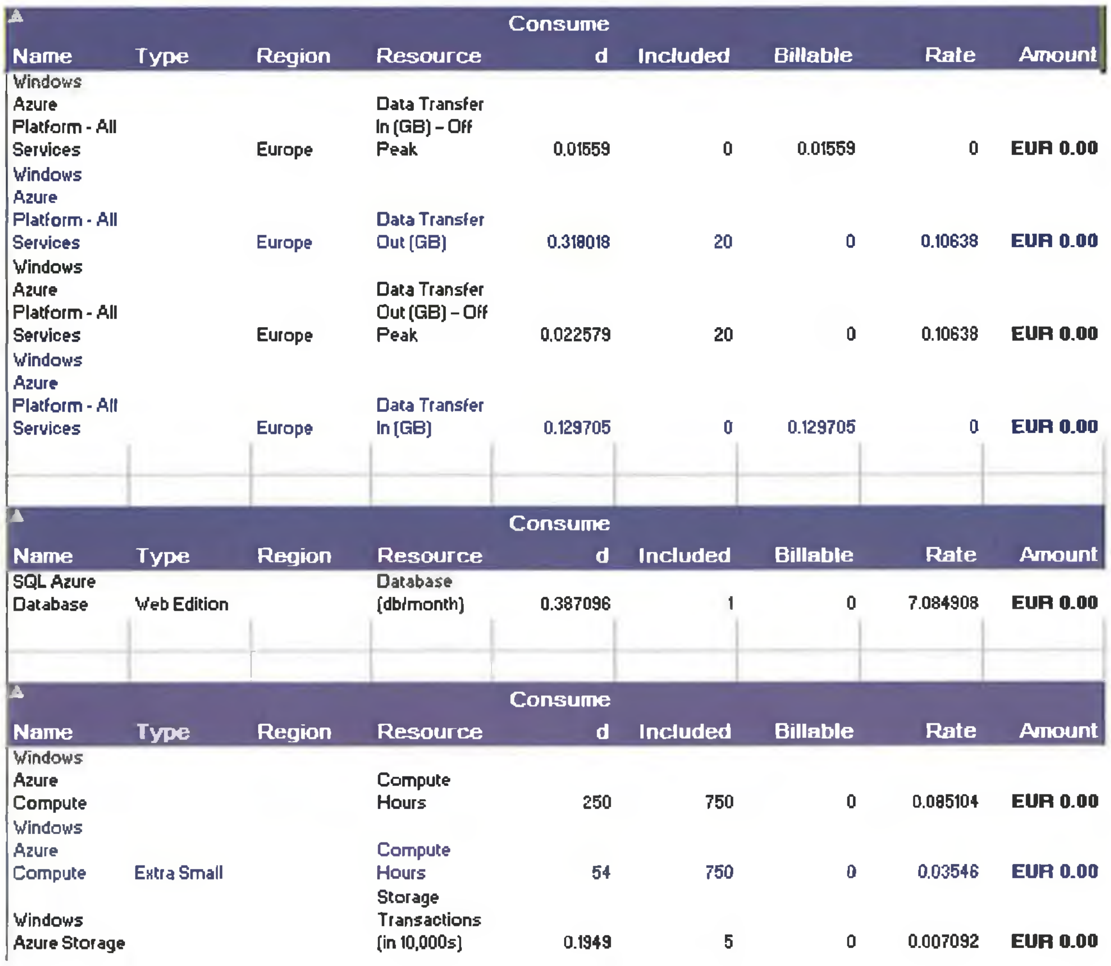


```
               Upload     Download    Total
Current speed: 0 kbps     0 kbps
Average:       76 kbps    933 kbps
Volume:        1.7 MB     20.7 MB     22.4 MB
Duration:      00:03:11
```

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data



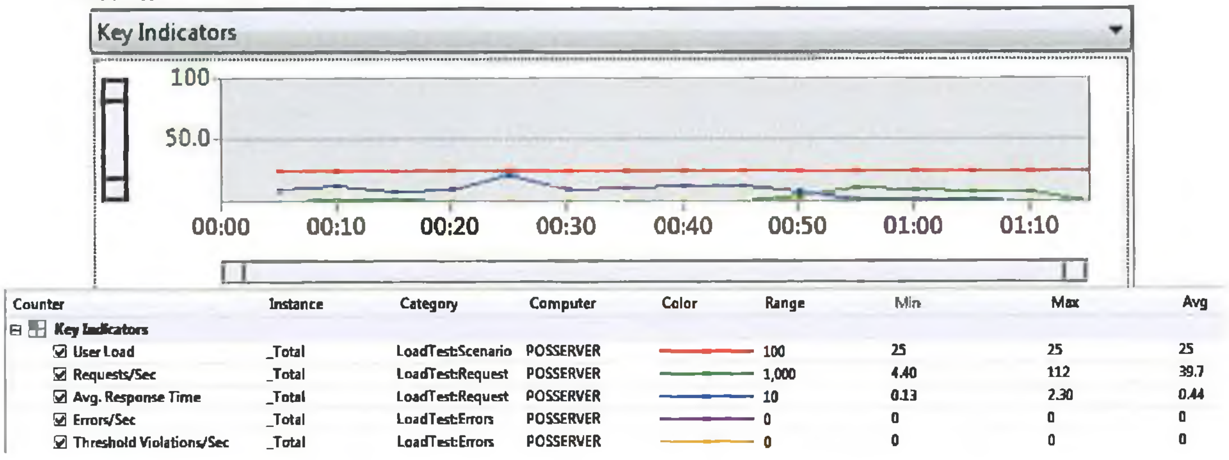
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.01559	0	0.01559	0	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.318018	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.022579	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.129705	0	0.129705	0	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.387096	1	0	7.084908	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	EUR 0.00
Windows Azure Compute	Extra Small		Compute Hours	54	750	0	0.03546	EUR 0.00
Windows Azure Storage			Storage Transactions (in 10,000s)	0.1949	5	0	0.007092	EUR 0.00

13th August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
⊟ Key Indicators								
☑ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	4.40	112	39.7
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.13	2.30	0.44
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

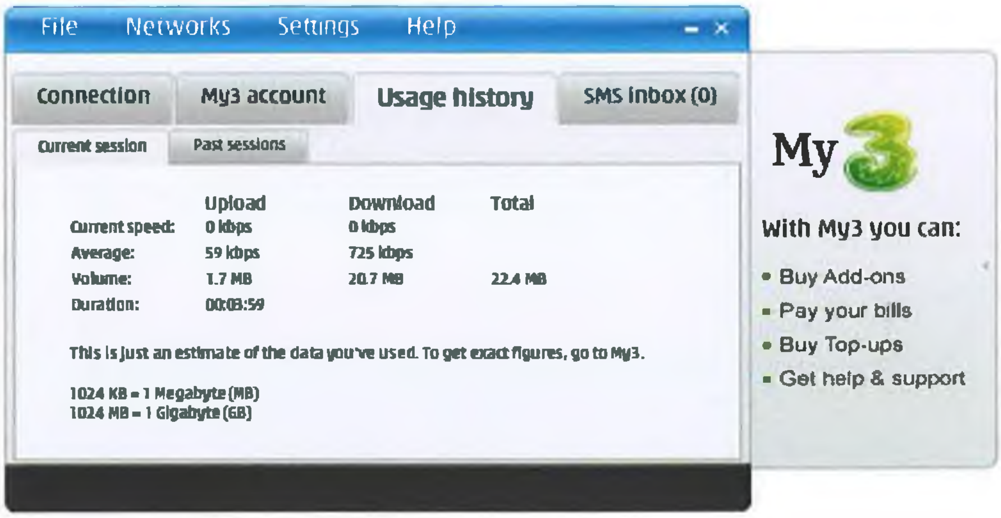
Test Run Information

Load test name	LoadTest1
Description	
Start time	13/08/2011 22:57:26
End time	13/08/2011 22:58:36
Warm-up duration	00:00:00
Duration	00:01:10
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	39.7
Requests Failed	0
Requests Cached Percentage	43.1
Avg. Response Time (sec)	0.44
Avg. Content Length (bytes)	5,388
Tests/Sec	0.14
Tests Failed	0
Avg. Test Time (sec)	68.6
Avg. Page Time (sec)	4.92

Section 2 – Client Test Machine

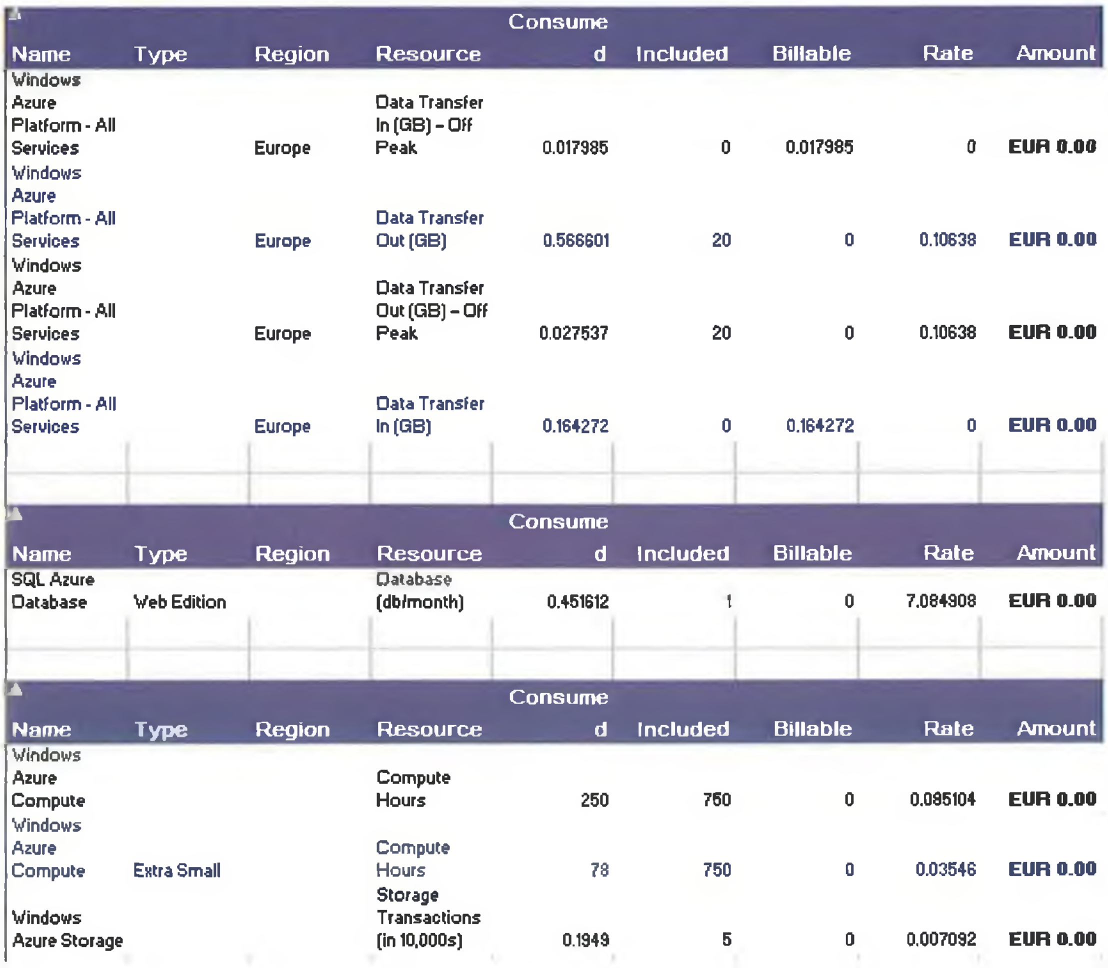


	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	59 kbps	725 kbps	
Volume:	1.7 MB	20.7 MB	22.4 MB
Duration:	00:03:59		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data



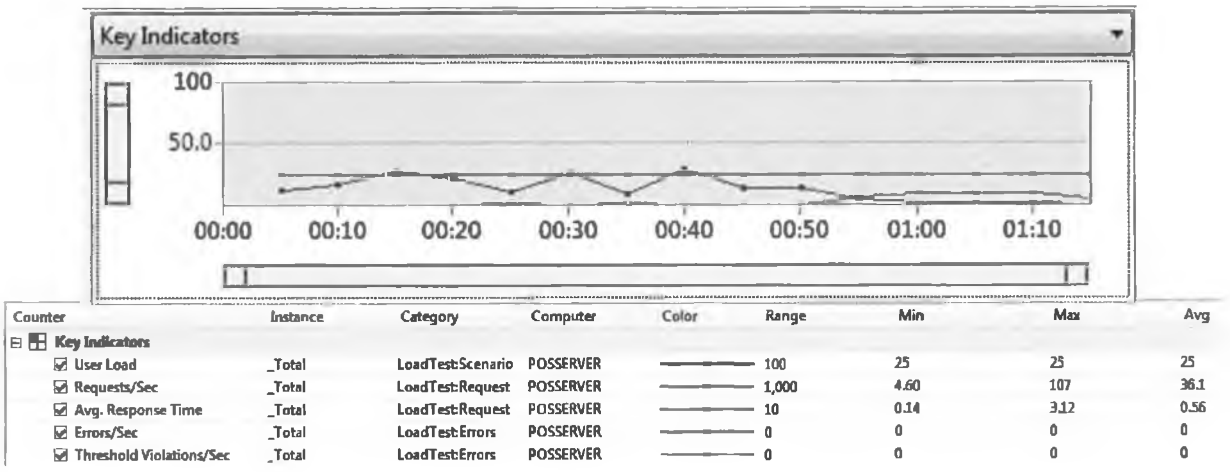
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.017985	0	0.017985	0	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.566601	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.027537	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.164272	0	0.164272	0	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.451612	1	0	7.084908	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	**EUR 0.00**
Windows Azure Compute	Extra Small		Compute Hours	78	750	0	0.03546	**EUR 0.00**
Windows Azure Storage			Storage Transactions (in 10,000s)	0.1949	5	0	0.007092	**EUR 0.00**

14th August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
Key Indicators								
✓ User Load	_Total	LoadTest:Scenario	POSSERVER	—	100	25	25	25
✓ Requests/Sec	_Total	LoadTest:Request	POSSERVER	—	1,000	4.60	107	36.1
✓ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	—	10	0.14	3.12	0.56
✓ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	—	0	0	0	0
✓ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	—	0	0	0	0

Load Test Summary

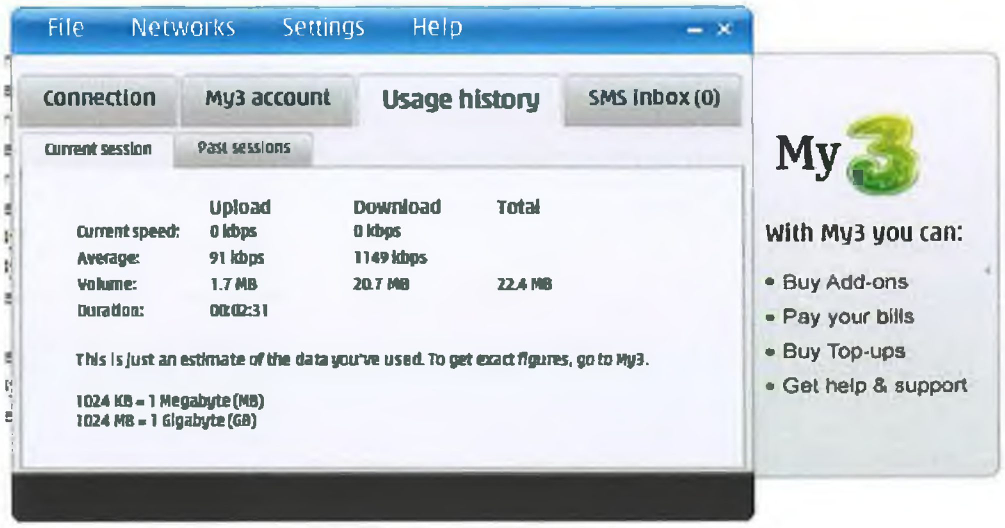
Test Run Information

Load test name	LoadTest1
Start time	14/08/2011 22:53:25
End time	14/08/2011 22:54:39
Warm-up duration	00:00:00
Duration	00:01:13
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	36.1
Requests Failed	0
Requests Cached Percentage	44.2
Avg. Response Time (sec)	0.56
Avg. Content Length (bytes)	5,645
Tests/Sec	0.14
Tests Failed	0
Avg. Test Time (sec)	71.6
Avg. Page Time (sec)	5.25

Section 2 – Client Test Machine

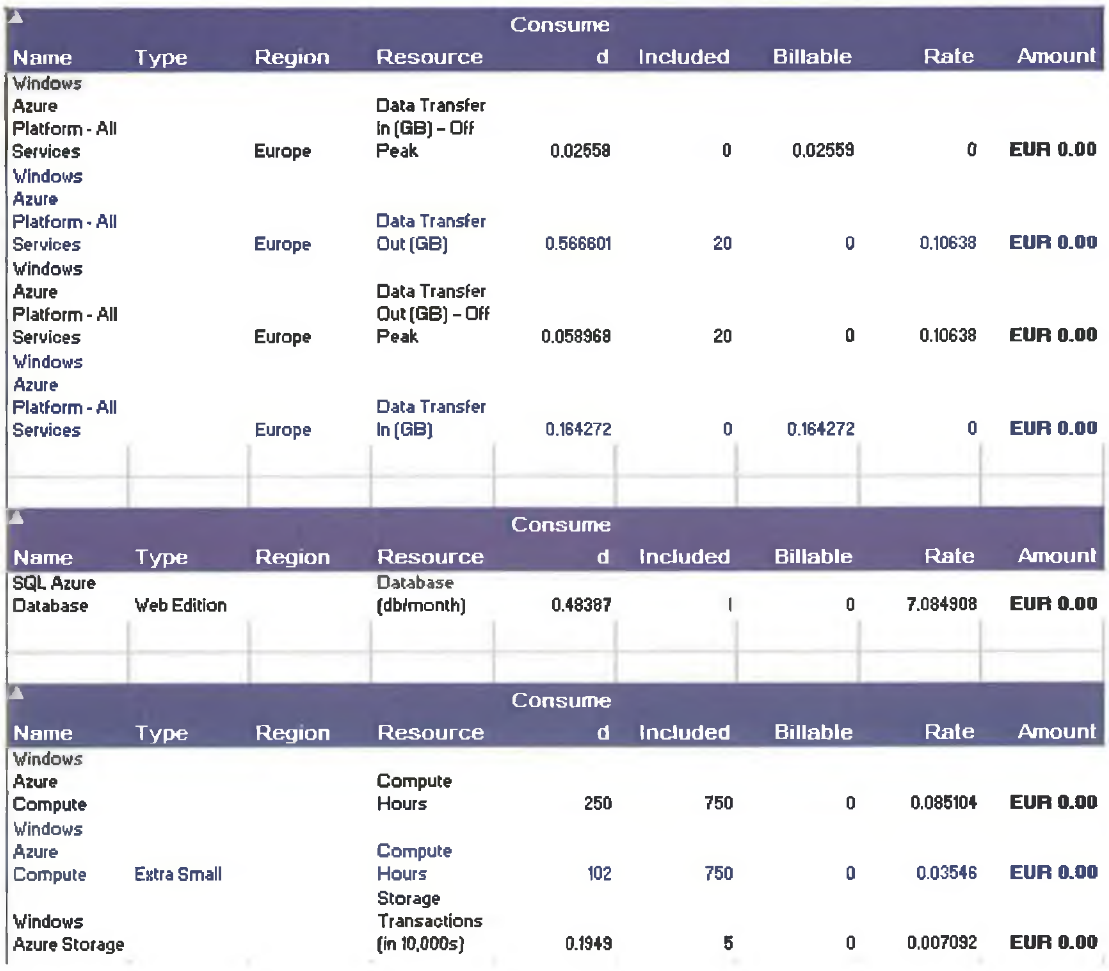


	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	91 kbps	1149 kbps	
Volume:	1.7 MB	20.7 MB	22.4 MB
Duration:	00:02:31		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data



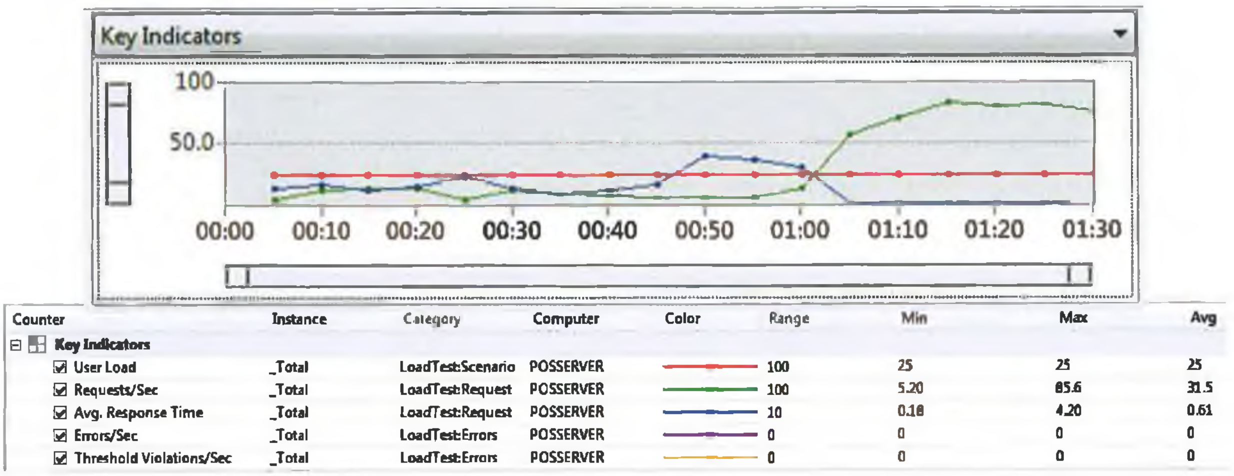
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.02558	0	0.02559	0	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.566601	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.058968	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.164272	0	0.164272	0	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.48387	1	0	7.084908	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	EUR 0.00
Windows Azure Compute	Extra Small		Compute Hours	102	750	0	0.03546	EUR 0.00
Windows Azure Storage			Storage Transactions (in 10,000s)	0.1949	5	0	0.007092	EUR 0.00

15th August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
Key Indicators								
☑ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	100	5.20	85.6	31.5
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.18	4.20	0.61
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

Load Test Summary

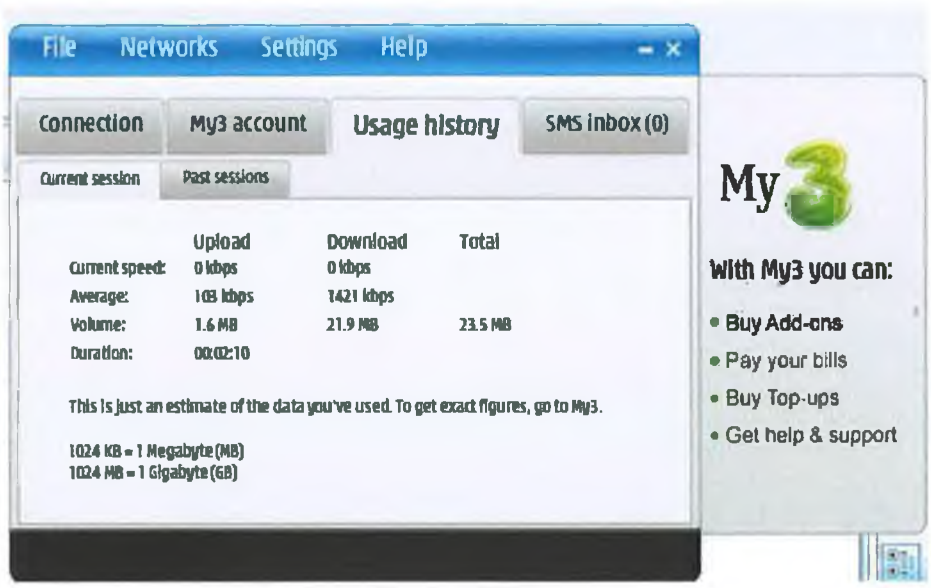
Test Run Information

Load test name	LoadTest1
Start time	15/08/2011 23:07:34
End time	15/08/2011 23:09:02
Warm-up duration	00:00:00
Duration	00:01:28
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	31.5
Requests Failed	0
Requests Cached Percentage	43.7
Avg. Response Time (sec)	0.61
Avg. Content Length (bytes)	5,765
Tests/Sec	0.11
Tests Failed	0
Avg. Test Time (sec)	87.2
Avg. Page Time (sec)	6.30

Section 2 – Client Test Machine

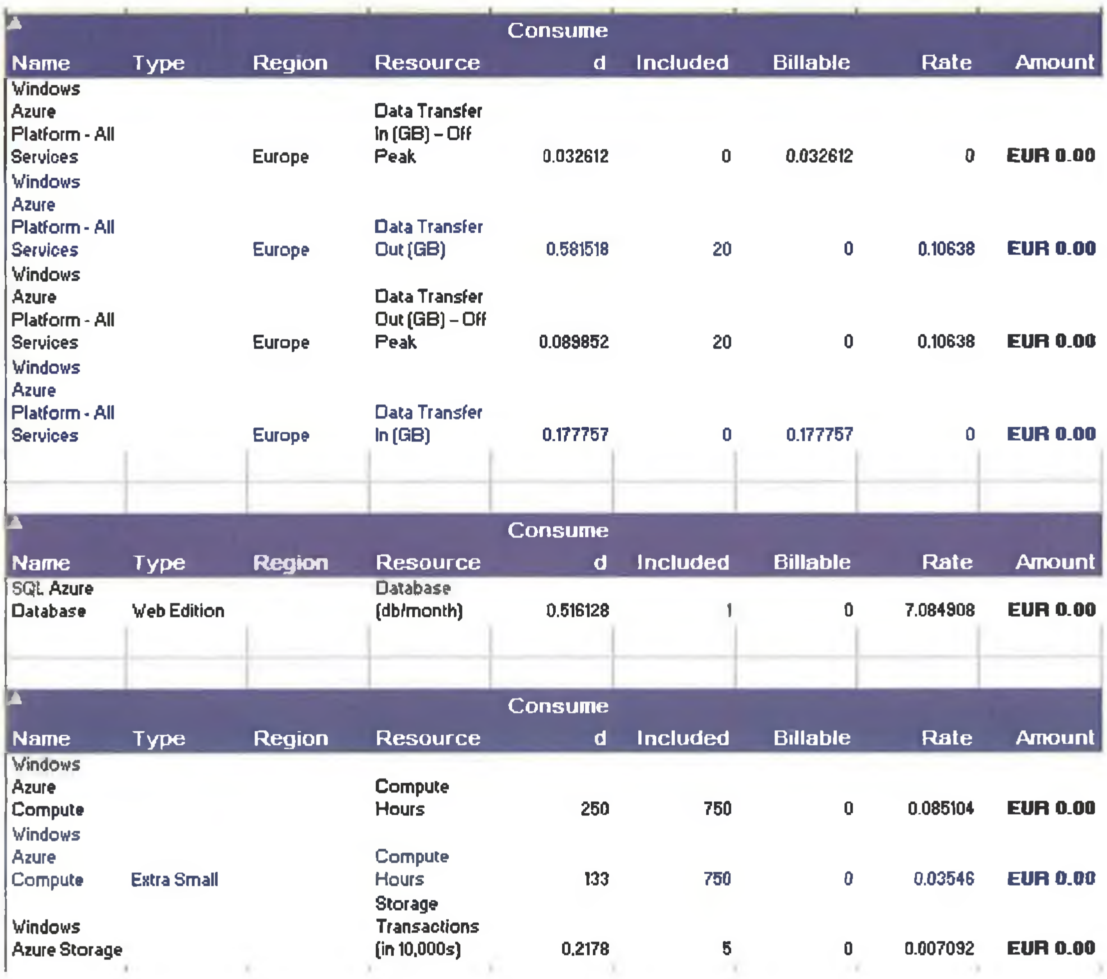


	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	103 kbps	1421 kbps	
Volume:	1.6 MB	21.9 MB	23.5 MB
Duration:	00:02:10		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data



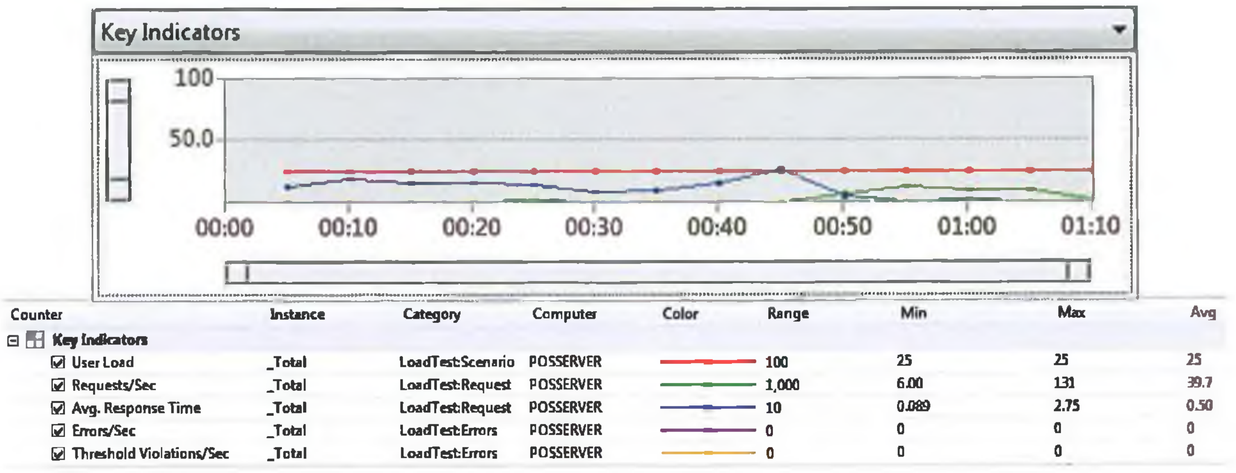
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.032612	0	0.032612	0	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.581518	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.089852	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.177757	0	0.177757	0	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.516128	1	0	7.084908	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	**EUR 0.00**
Windows Azure Compute	Extra Small		Compute Hours	133	750	0	0.03546	**EUR 0.00**
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	**EUR 0.00**

16th August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
⊟ **Key Indicators**								
☑ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	6.00	131	39.7
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.089	2.75	0.50
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

Load Test Summary

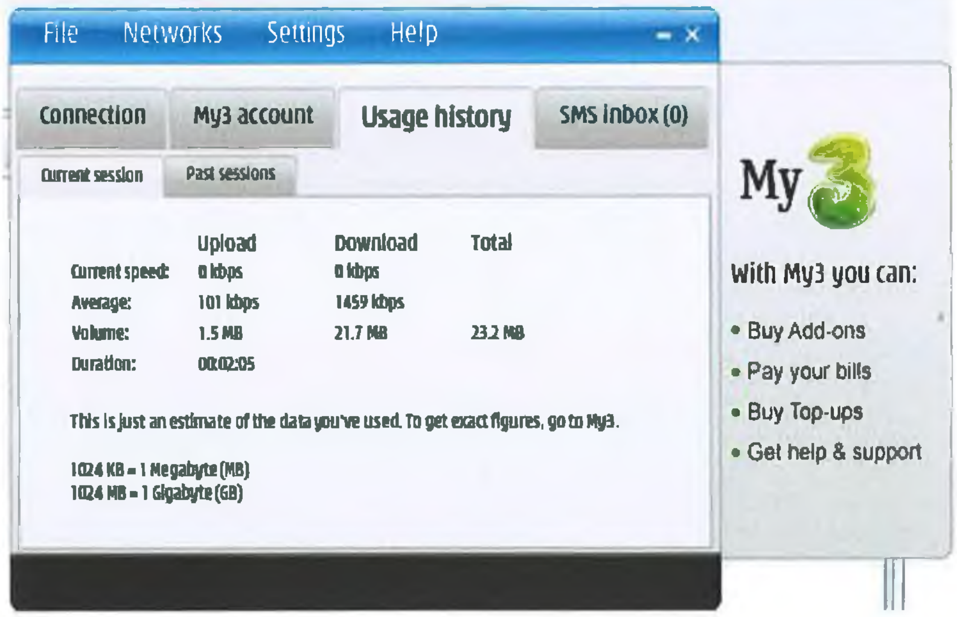
Test Run Information

Load test name	LoadTest1
Start time	16/08/2011 11:22:39
End time	16/08/2011 11:23:44
Warm-up duration	00:00:00
Duration	00:01:05
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	39.7
Requests Failed	0
Requests Cached Percentage	45.3
Avg. Response Time (sec)	0.50
Avg. Content Length (bytes)	6,149
Tests/Sec	0.15
Tests Failed	0
Avg. Test Time (sec)	64.1
Avg. Page Time (sec)	4.63

Section 2 – Client Test Machine



	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	101 kbps	1459 kbps	
Volume:	1.5 MB	21.7 MB	23.2 MB
Duration:	00:02:05		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

My3

With My3 you can:

- Buy Add-ons
- Pay your bills
- Buy Top-ups
- Get help & support

Section 3 – Azure Data

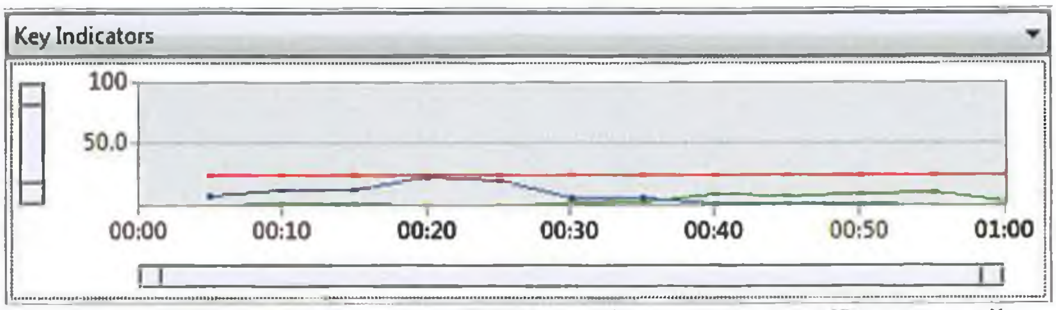
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.03612	0	0.03612	0	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.605041	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.134213	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.179476	0	0.179476	0	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.548386	1	0	7.084908	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	EUR 0.00
Windows Azure Compute	Extra Small		Compute Hours	181	750	0	0.03546	EUR 0.00
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	EUR 0.00

17th August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
⊟ **Key Indicators**								
☑ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	4.80	120	48.2
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.12	2.39	0.39
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

Load Test Summary

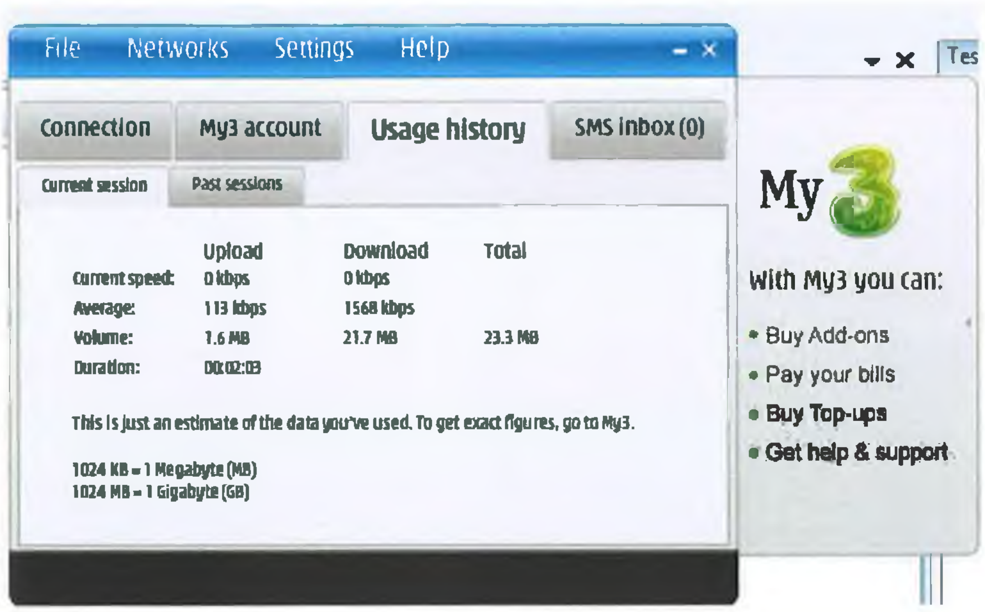
Test Run Information

Load test name	LoadTest1
Start time	17/08/2011 11:20:05
End time	17/08/2011 11:21:03
Warm-up duration	00:00:00
Duration	00:00:58
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	48.2
Requests Failed	0
Requests Cached Percentage	43.6
Avg. Response Time (sec)	0.39
Avg. Content Length (bytes)	5,726
Tests/Sec	0.17
Tests Failed	0
Avg. Test Time (sec)	55.6
Avg. Page Time (sec)	4.00

Section 2 – Client Test Machine

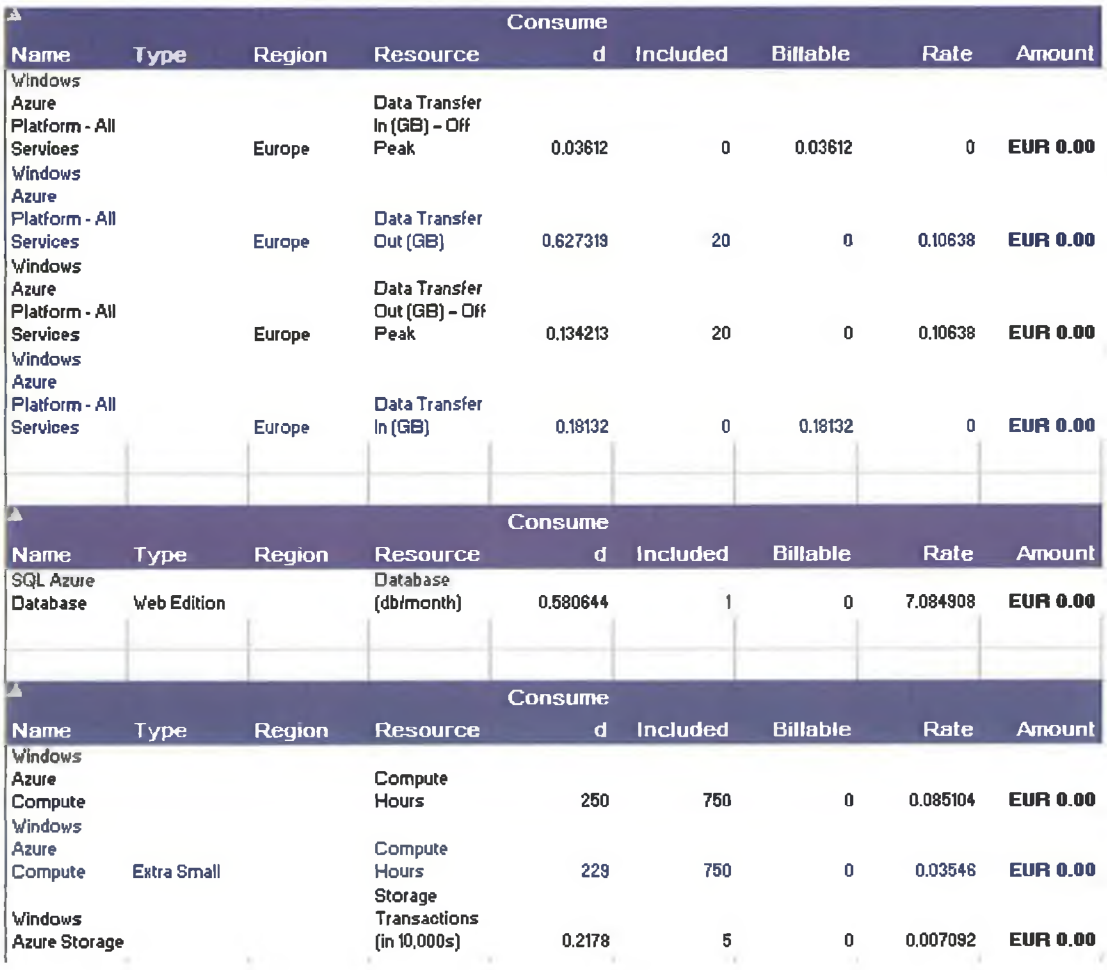


	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	113 kbps	1568 kbps	
Volume:	1.6 MB	21.7 MB	23.3 MB
Duration:	00:02:03		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data



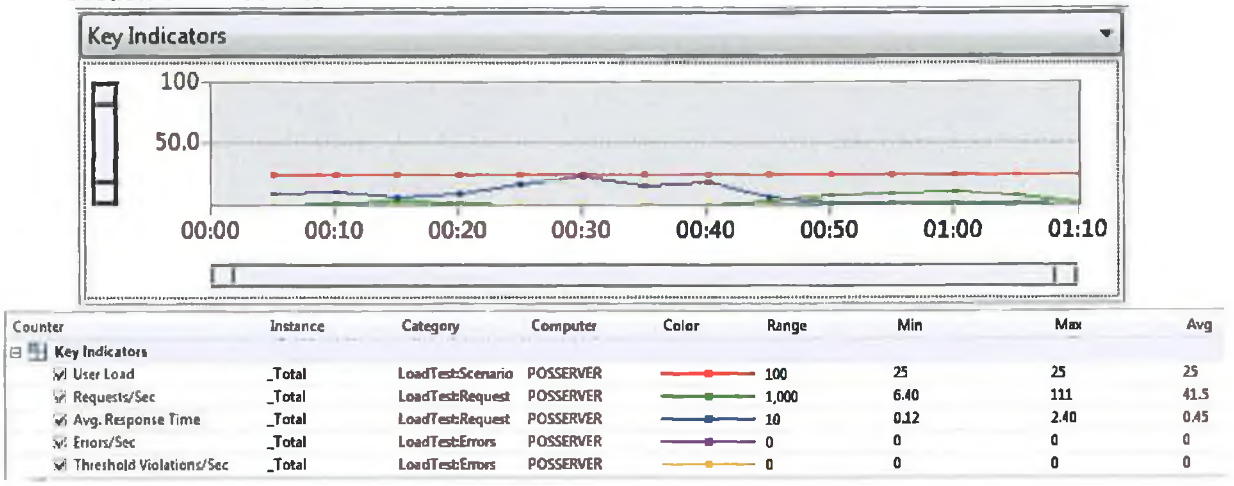
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) - Off Peak	0.03612	0	0.03612	0	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.627319	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) - Off Peak	0.134213	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.18132	0	0.18132	0	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.580644	1	0	7.084908	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	EUR 0.00
Windows Azure Compute	Extra Small		Compute Hours	229	750	0	0.03546	EUR 0.00
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	EUR 0.00

18th August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
⊟ Key Indicators								
✓ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
✓ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	6.40	111	41.5
✓ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.12	2.40	0.45
✓ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
✓ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

Load Test Summary

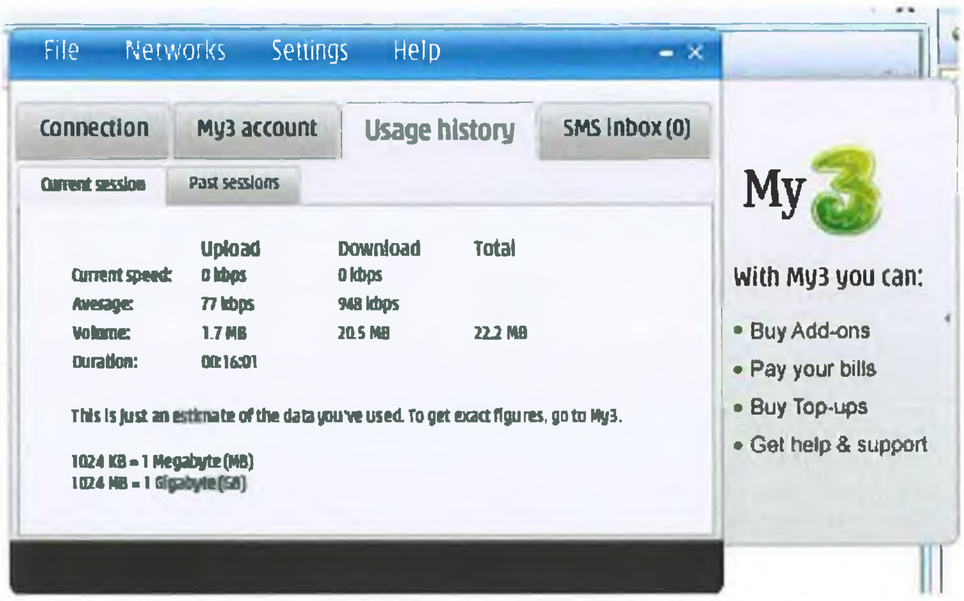
Test Run Information

Load test name	LoadTest1
Start time	18/08/2011 21:52:50
End time	18/08/2011 21:53:57
Warm-up duration	00:00:00
Duration	00:01:07
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	41.5
Requests Failed	0
Requests Cached Percentage	43.0
Avg. Response Time (sec)	0.45
Avg. Content Length (bytes)	5,392
Tests/Sec	0.15
Tests Failed	0
Avg. Test Time (sec)	63.9
Avg. Page Time (sec)	4.64

Section 2 – Client Test Machine

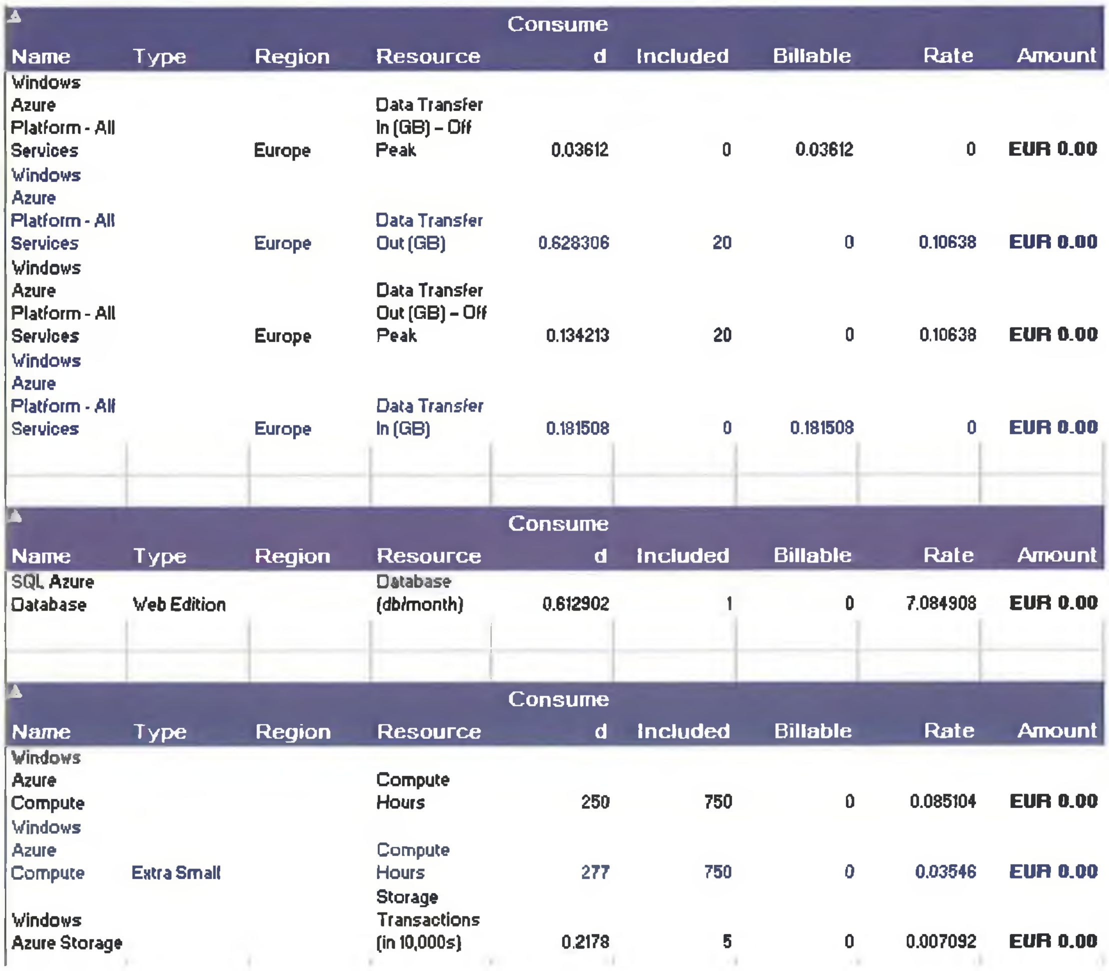


	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	77 kbps	948 kbps	
Volume:	1.7 MB	20.5 MB	22.2 MB
Duration:	00:16:01		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data



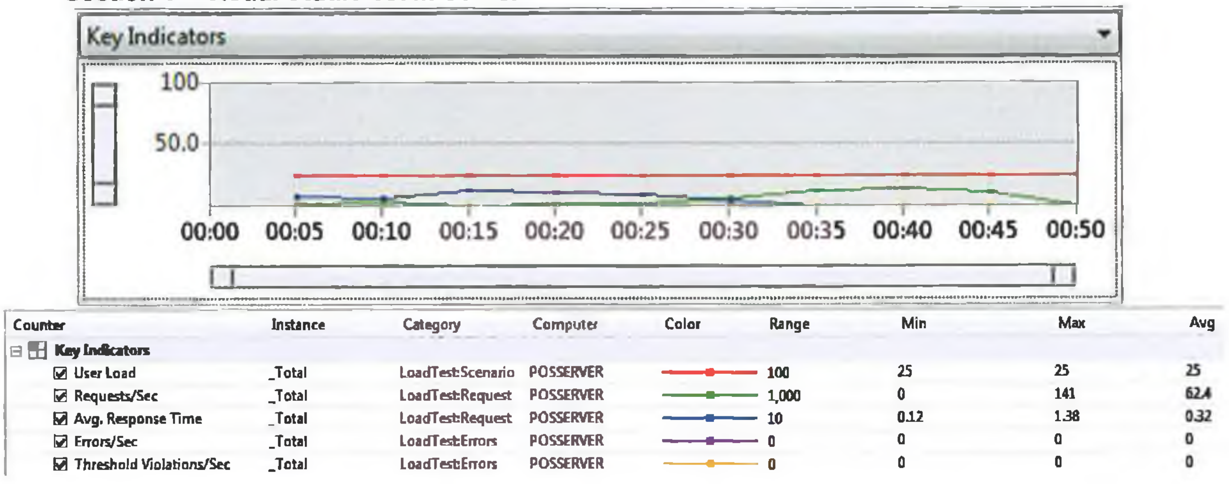
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.03612	0	0.03612	0	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.628306	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.134213	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.181508	0	0.181508	0	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.612902	1	0	7.084908	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	EUR 0.00
Windows Azure Compute	Extra Small		Compute Hours	277	750	0	0.03546	EUR 0.00
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	EUR 0.00

19th August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
Key Indicators								
☑ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	0	141	62.4
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.12	1.38	0.32
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

Load Test Summary

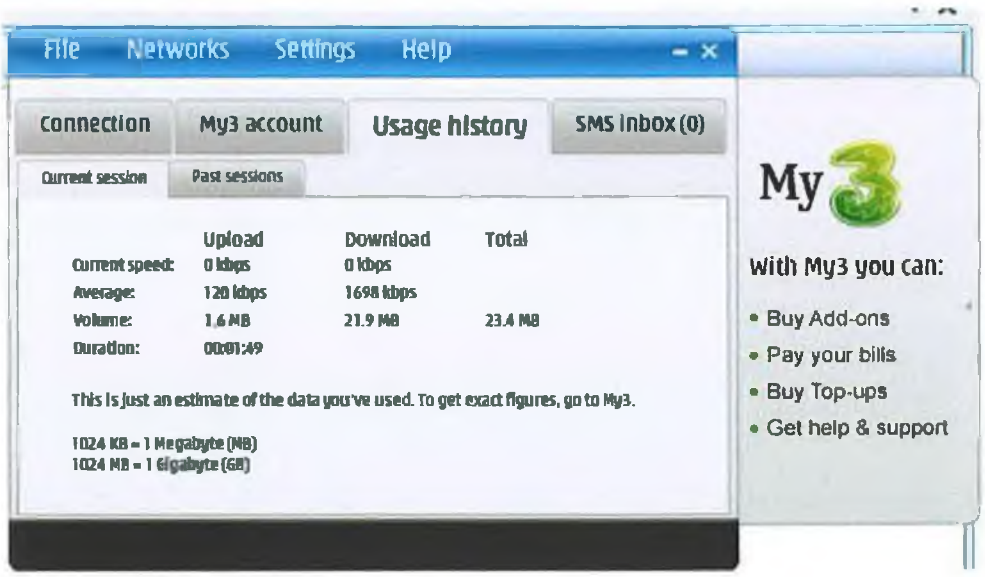
Test Run Information

Load test name	LoadTest1
Start time	19/08/2011 10:40:03
End time	19/08/2011 10:40:48
Warm-up duration	00:00:00
Duration	00:00:45
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	62.4
Requests Failed	0
Requests Cached Percentage	43.6
Avg. Response Time (sec)	0.32
Avg. Content Length (bytes)	5,726
Tests/Sec	0.22
Tests Failed	0
Avg. Test Time (sec)	43.5
Avg. Page Time (sec)	3.15

Section 2 – Client Test Machine



	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	120 kbps	1698 kbps	
Volume:	1.6 MB	21.9 MB	23.4 MB
Duration:	00:01:49		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data

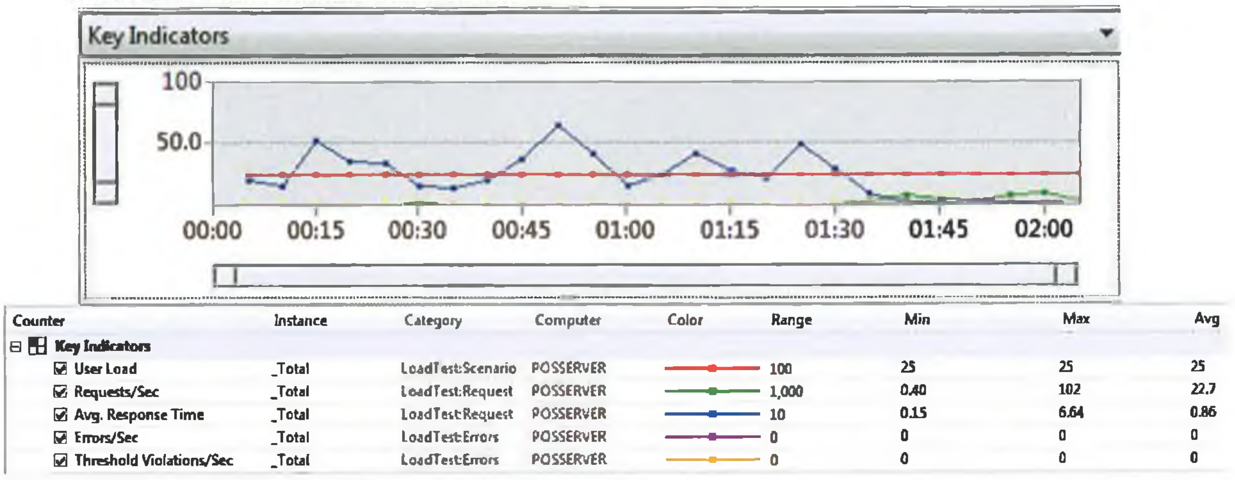
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) - Off Peak	0.036141	0	0.036141	0	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.670915	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) - Off Peak	0.134362	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.185201	0	0.185201	0	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.64516	1	0	7.084908	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	**EUR 0.00**
Windows Azure Compute	Extra Small		Compute Hours	327	750	0	0.03546	**EUR 0.00**
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	**EUR 0.00**

20th August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
⊟ Key Indicators								
☑ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	0.40	102	22.7
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.15	6.64	0.86
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

Load Test Summary

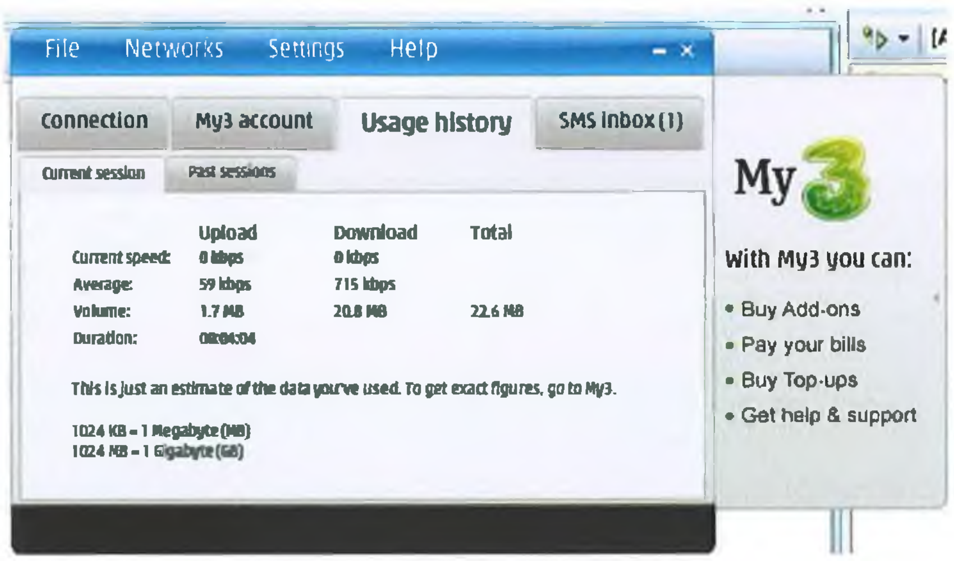
Test Run Information

Load test name	LoadTest1
Start time	20/08/2011 21:11:49
End time	20/08/2011 21:13:51
Warm-up duration	00:00:00
Duration	00:02:02
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	22.7
Requests Failed	0
Requests Cached Percentage	43.3
Avg. Response Time (sec)	0.86
Avg. Content Length (bytes)	5,438
Tests/Sec	0.082
Tests Failed	0
Avg. Test Time (sec)	119
Avg. Page Time (sec)	8.66

Section 2 – Client Test Machine



	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	59 kbps	715 kbps	
Volume:	1.7 MB	20.8 MB	22.6 MB
Duration:	00:04:04		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data

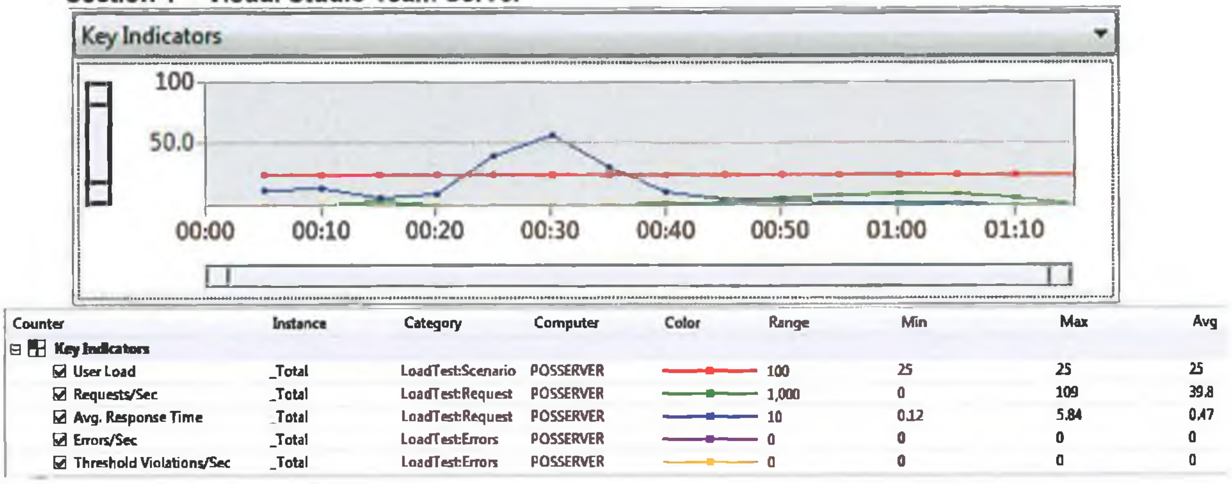
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.036163	0	0.036163	0	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.67195	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.134536	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.185381	0	0.185381	0	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.677418	1	0	7.084908	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	**EUR 0.00**
Windows Azure Compute	Extra Small		Compute Hours	373	750	0	0.03546	**EUR 0.00**
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	**EUR 0.00**

21st August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
⊟ Key Indicators								
☑ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	0	109	39.8
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.12	5.84	0.47
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

Load Test Summary

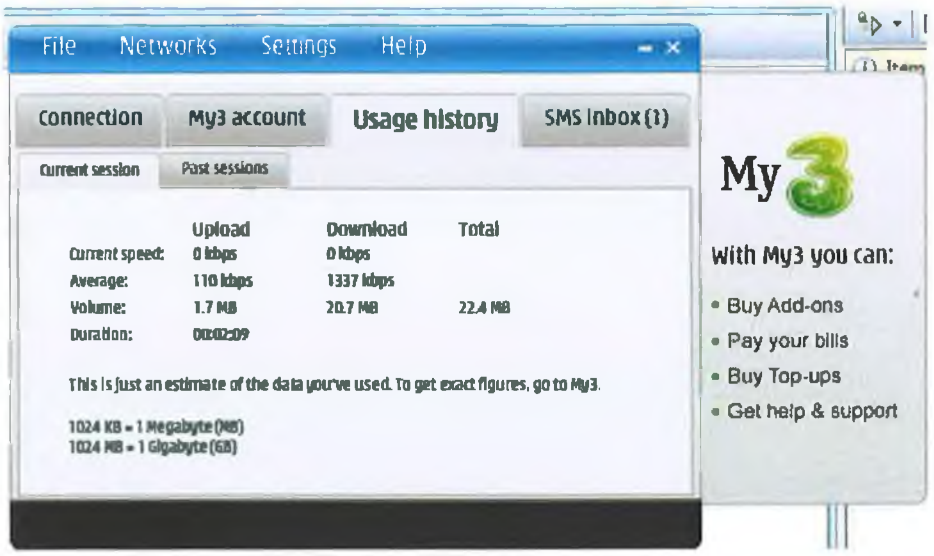
Test Run Information

Load test name	LoadTest1
Start time	21/08/2011 15:36:04
End time	21/08/2011 15:37:14
Warm-up duration	00:00:00
Duration	00:01:10
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	39.8
Requests Failed	0
Requests Cached Percentage	43.0
Avg. Response Time (sec)	0.47
Avg. Content Length (bytes)	5,392
Tests/Sec	0.14
Tests Failed	0
Avg. Test Time (sec)	66.6
Avg. Page Time (sec)	4.81

Section 2 – Client Test Machine

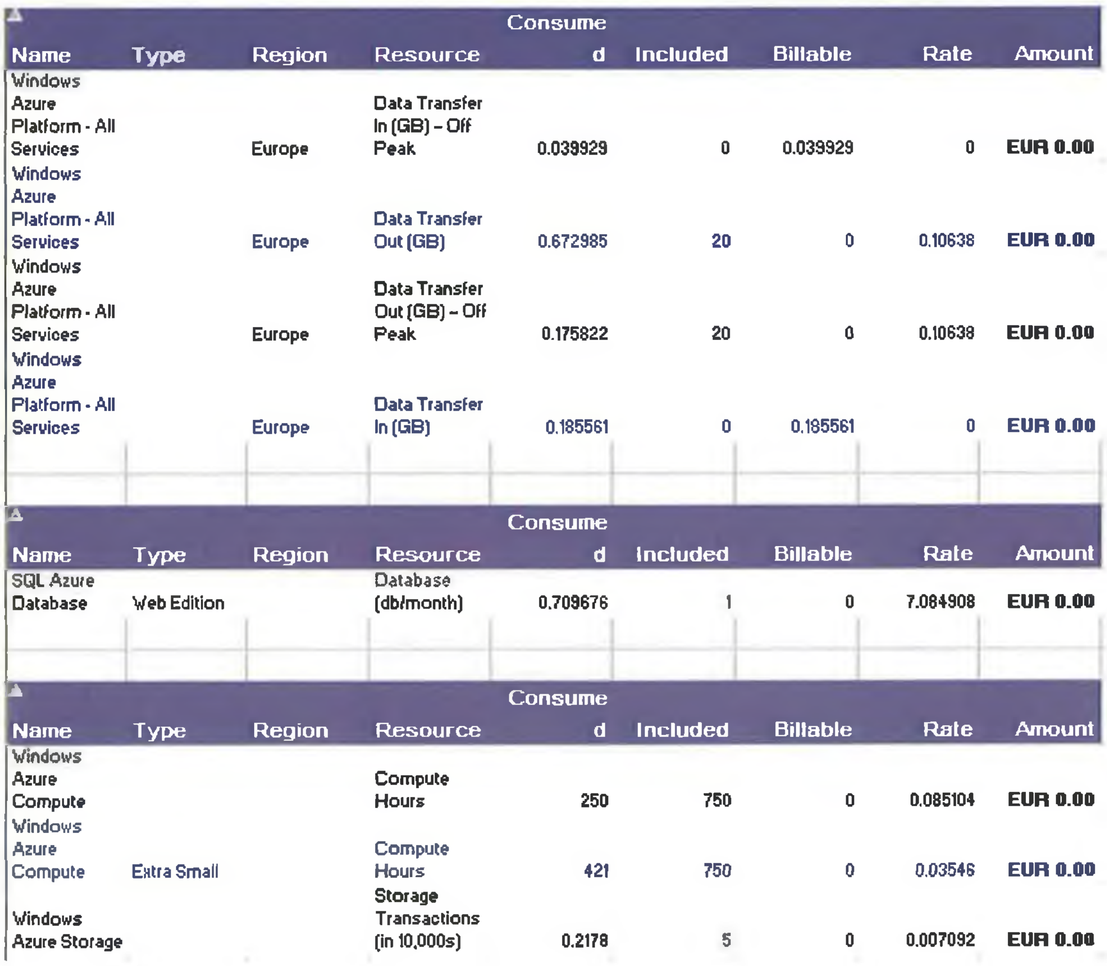


	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	110 kbps	1337 kbps	
Volume:	1.7 MB	20.7 MB	22.4 MB
Duration:	00:02:09		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data



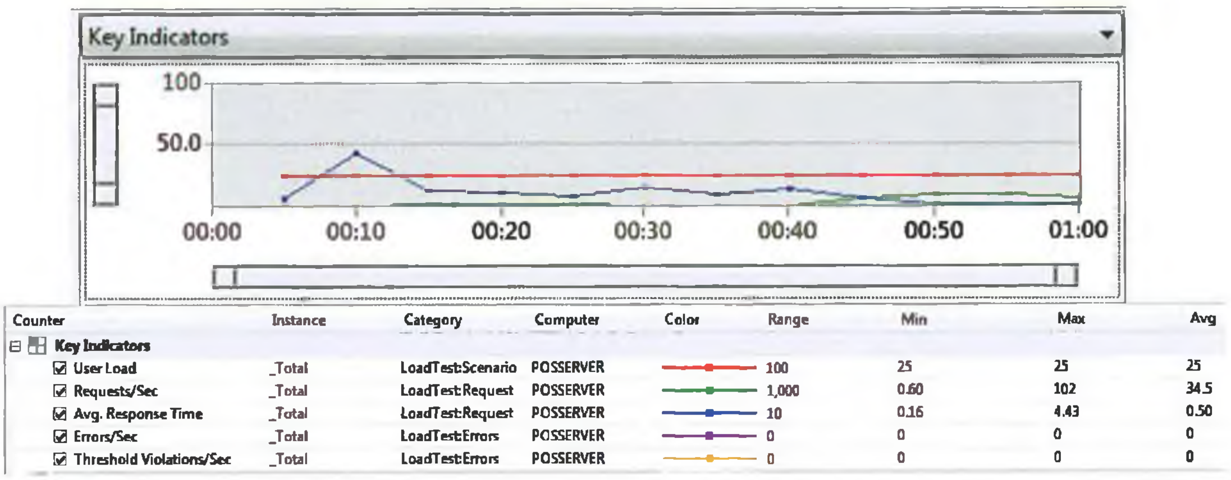
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.039929	0	0.039929	0	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.672985	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.175822	20	0	0.10638	EUR 0.00
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.185561	0	0.185561	0	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.709676	1	0	7.084908	EUR 0.00

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	EUR 0.00
Windows Azure Compute	Extra Small		Compute Hours	421	750	0	0.03546	EUR 0.00
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	EUR 0.00

22nd August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
⊟ Key Indicators								
☑ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	0.60	102	34.5
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.16	4.43	0.50
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

Load Test Summary

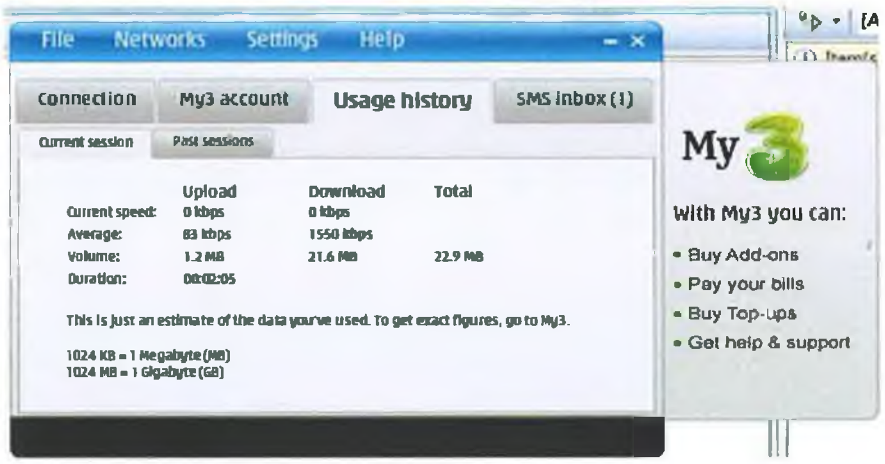
Test Run Information

Load test name	LoadTest1
Start time	22/08/2011 11:27:36
End time	22/08/2011 11:28:35
Warm-up duration	00:00:00
Duration	00:00:59
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	34.5
Requests Failed	0
Requests Cached Percentage	51.6
Avg. Response Time (sec)	0.50
Avg. Content Length (bytes)	7,907
Tests/Sec	0.17
Tests Failed	0
Avg. Test Time (sec)	57.9
Avg. Page Time (sec)	4.13

Section 2 – Client Test Machine



```
                Upload       Download     Total
Current speed:  0 kbps       0 kbps
Average:        83 kbps      1550 kbps
Volume:         1.2 MB       21.6 MB      22.9 MB
Duration:       00:02:05

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)
```

Section 3 – Azure Data

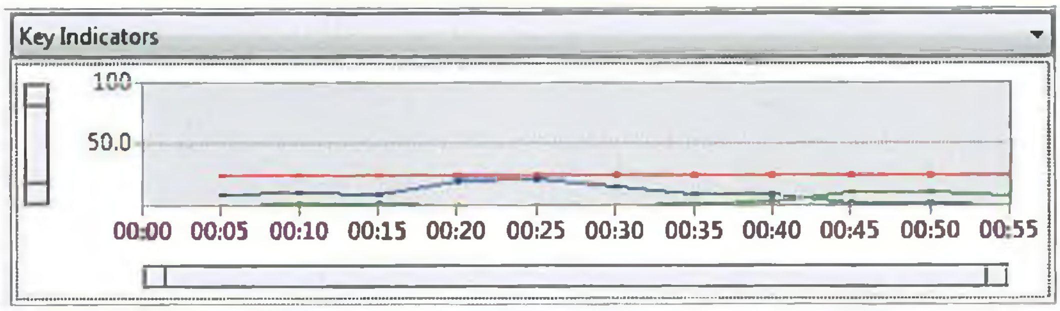
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.039929	0	0.039929	0	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.672985	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.175822	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.18702	0	0.18702	0	**EUR 0.00**
Windows Azure Platform - All Services		Zone 1	Data Transfer In (GB)	0.000195	0	0.000195	0.007092	**EUR 0.00**
Windows Azure Platform - All Services		Zone 1	Data Transfer Out (GB)	0.000987	0	0.000987	0.007092	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.741934	1	0	7.094908	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	**EUR 0.00**
Windows Azure Compute	Extra Small		Compute Hours	469	750	0	0.03546	**EUR 0.00**
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	**EUR 0.00**

23rd August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
Key Indicators								
✓ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
✓ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	6.80	119	39.8
✓ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.16	2.28	0.52
✓ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
✓ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟧	0	0	0	0

Load Test Summary

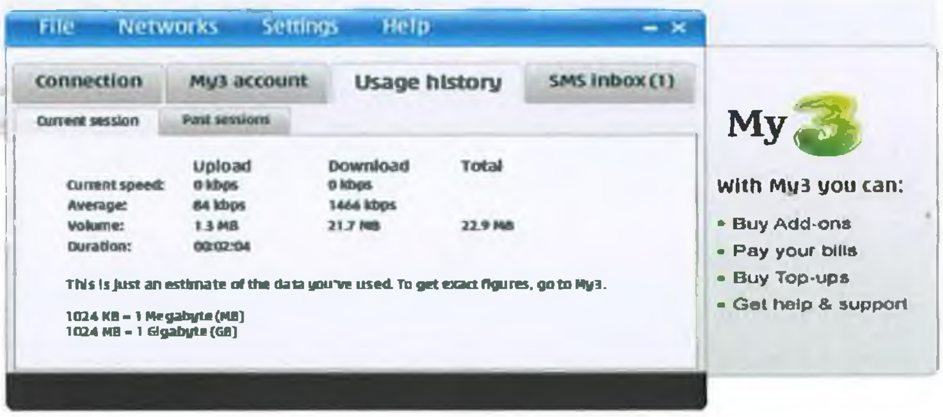
Test Run Information

Load test name	LoadTest1
Start time	23/08/2011 11:06:46
End time	23/08/2011 11:07:39
Warm-up duration	00:00:00
Duration	00:00:52
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	39.8
Requests Failed	0
Requests Cached Percentage	50.8
Avg. Response Time (sec)	0.52
Avg. Content Length (bytes)	7,651
Tests/Sec	0.19
Tests Failed	0
Avg. Test Time (sec)	51.9
Avg. Page Time (sec)	3.78

Section 2 – Client Test Machine



```
                Upload      Download     Total
Current speed:  0 kbps      0 kbps
Average:        84 kbps     1464 kbps
Volume:         1.3 MB      21.7 MB      22.9 MB
Duration:       00:02:04

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)
```

Section 3 – Azure Data

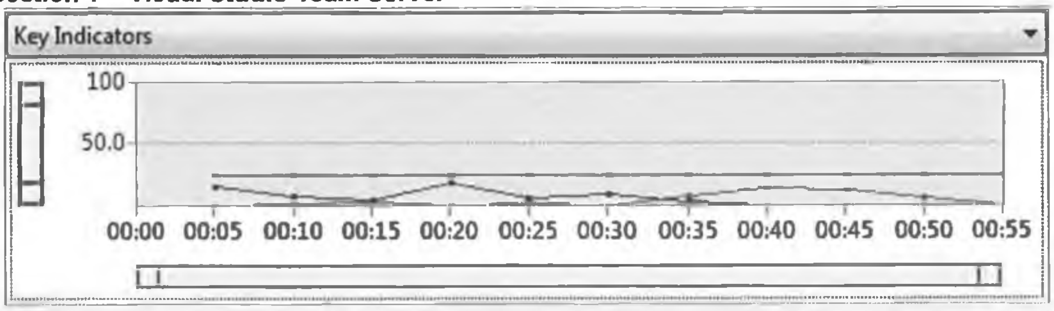
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.039929	0	0.039929	0	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.716271	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.175822	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.188419	0	0.188419	0	**EUR 0.00**
Windows Azure Platform - All Services		Zone 1	Data Transfer in (GB)	0.000845	0	0.000845	0.007092	**EUR 0.00**
Windows Azure Platform - All Services		Zone 1	Data Transfer Out (GB)	0.004277	0	0.004277	0.007092	**EUR 0.00**

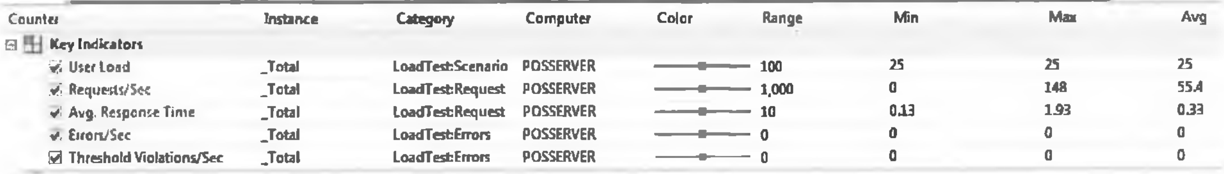
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.774192	1	0	7.084908	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	**EUR 0.00**
Windows Azure Compute	Extra Small		Compute Hours	517	750	0	0.03546	**EUR 0.00**
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	**EUR 0.00**

24th August 2011

Section 1 - Visual Studio Team Server





Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
⊟ Key Indicators								
✓ User Load	_Total	LoadTest:Scenario	POSSERVER	●	100	25	25	25
✓ Requests/Sec	_Total	LoadTest:Request	POSSERVER	●	1,000	0	148	55.4
✓ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	●	10	0.13	1.93	0.33
✓ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	●	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	●	0	0	0	0

Load Test Summary

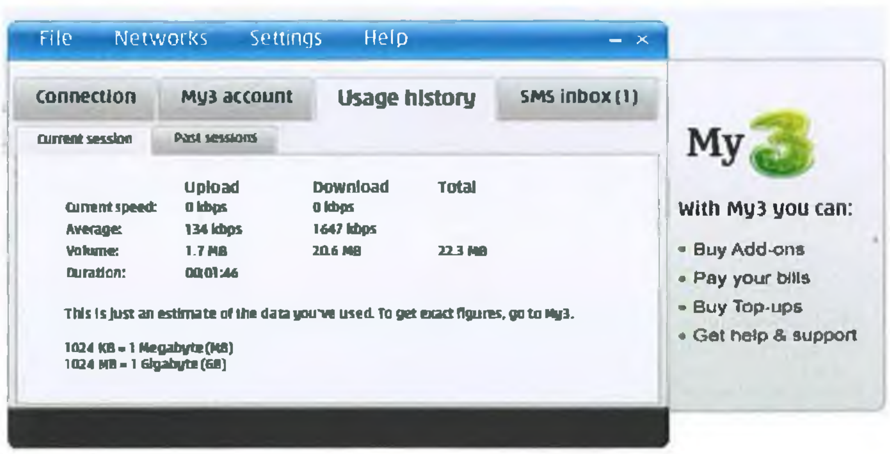
Test Run Information

Load test name	LoadTest1
Start time	24/08/2011 10:50:54
End time	24/08/2011 10:51:44
Warm-up duration	00:00:00
Duration	00:00:50
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	55.4
Requests Failed	0
Requests Cached Percentage	43.0
Avg. Response Time (sec)	0.33
Avg. Content Length (bytes)	5,392
Tests/Sec	0.20
Tests Failed	0
Avg. Test Time (sec)	47.5
Avg. Page Time (sec)	3.43

Section 2 – Client Test Machine



	Upload	Download	Total
Current speed:	0 kbps	0 kbps	
Average:	134 kbps	1647 kbps	
Volume:	1.7 MB	20.6 MB	22.3 MB
Duration:	00:01:46		

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data

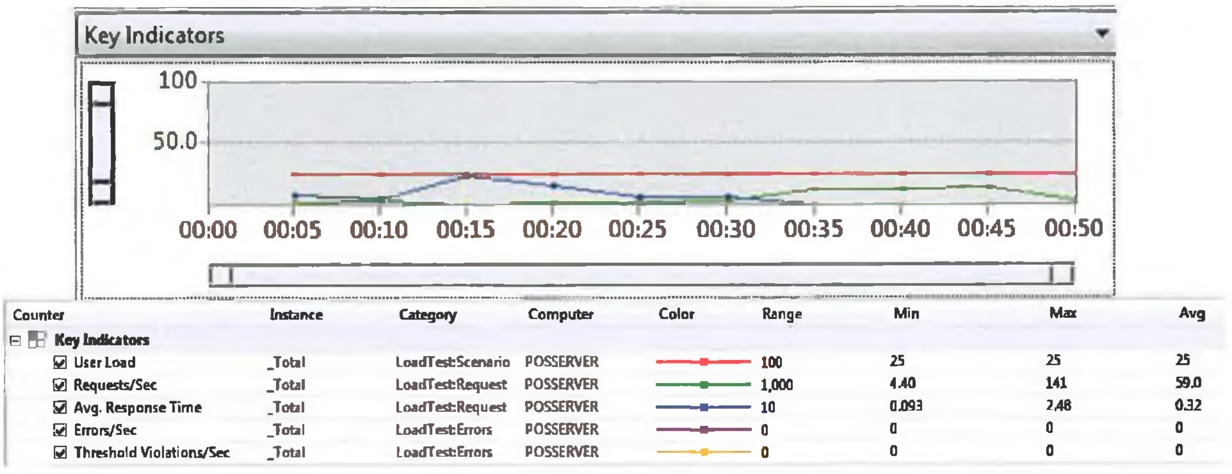
Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) – Off Peak	0.039929	0	0.039929	0	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.736617	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) – Off Peak	0.175822	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Zone 1	Data Transfer In (GB)	0.00156	0	0.00156	0.007092	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.190242	0	0.190242	0	**EUR 0.00**
Windows Azure Platform - All Services		Zone 1	Data Transfer Out (GB)	0.007886	0	0.007896	0.007092	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.80645	1	0	7.084908	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	**EUR 0.00**
Windows Azure Compute	Extra Small		Compute Hours	565	750	0	0.03546	**EUR 0.00**
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	**EUR 0.00**

25th August 2011

Section 1 - Visual Studio Team Server



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg
⊟ **Key Indicators**								
☑ User Load	_Total	LoadTest:Scenario	POSSERVER	🟥	100	25	25	25
☑ Requests/Sec	_Total	LoadTest:Request	POSSERVER	🟩	1,000	4.40	141	59.0
☑ Avg. Response Time	_Total	LoadTest:Request	POSSERVER	🟦	10	0.093	2.48	0.32
☑ Errors/Sec	_Total	LoadTest:Errors	POSSERVER	🟪	0	0	0	0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	POSSERVER	🟨	0	0	0	0

Load Test Summary

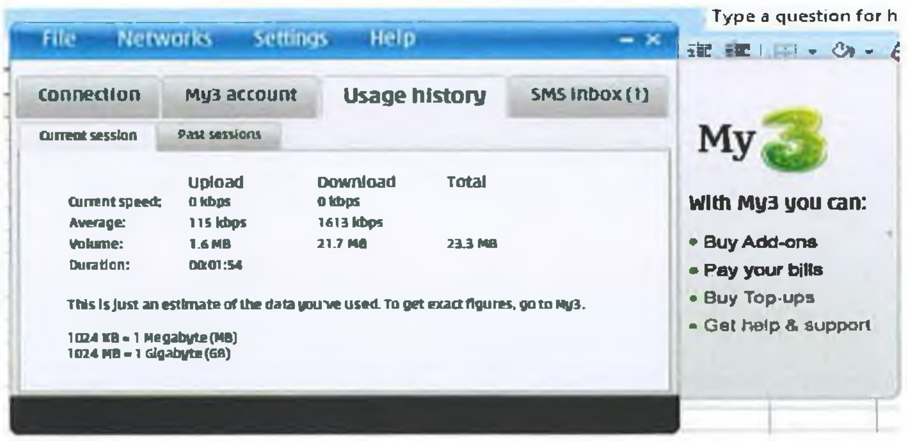
Test Run Information

Load test name	LoadTest1
Start time	25/08/2011 14:15:52
End time	25/08/2011 14:16:39
Warm-up duration	00:00:00
Duration	00:00:47
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	25
Requests/Sec	59.0
Requests Failed	0
Requests Cached Percentage	43.6
Avg. Response Time (sec)	0.32
Avg. Content Length (bytes)	5,726
Tests/Sec	0.21
Tests Failed	0
Avg. Test Time (sec)	45.1
Avg. Page Time (sec)	3.24

Section 2 – Client Test Machine

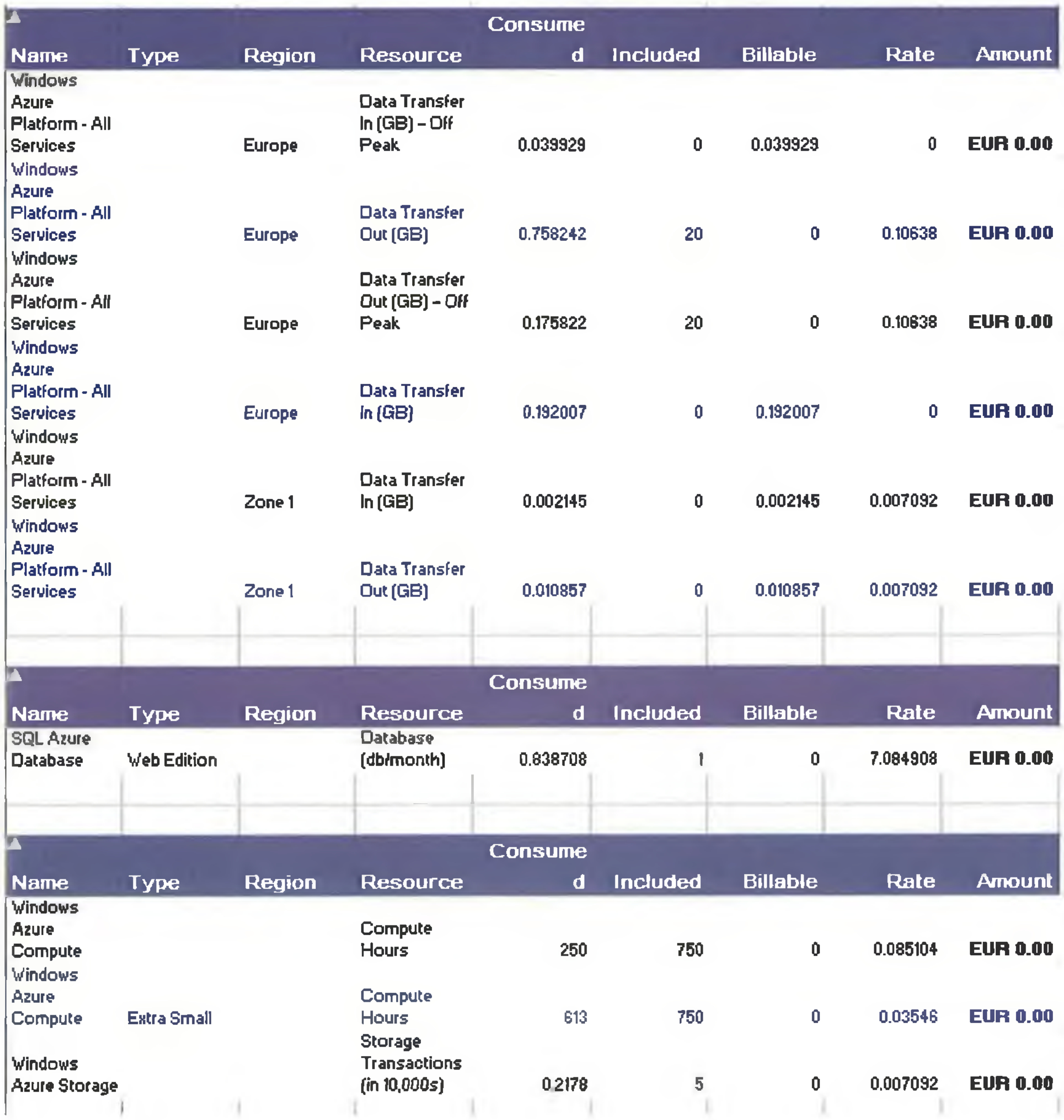


```
                Upload      Download     Total
Current speed:  0 kbps      0 kbps
Average:        115 kbps    1613 kbps
Volume:         1.6 MB      21.7 MB      23.3 MB
Duration:       00:01:54
```

This is just an estimate of the data you've used. To get exact figures, go to My3.

1024 KB = 1 Megabyte (MB)
1024 MB = 1 Gigabyte (GB)

Section 3 – Azure Data



Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Platform - All Services		Europe	Data Transfer In (GB) - Off Peak	0.039929	0	0.039929	0	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB)	0.758242	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer Out (GB) - Off Peak	0.175822	20	0	0.10638	**EUR 0.00**
Windows Azure Platform - All Services		Europe	Data Transfer In (GB)	0.192007	0	0.192007	0	**EUR 0.00**
Windows Azure Platform - All Services		Zone 1	Data Transfer In (GB)	0.002145	0	0.002145	0.007092	**EUR 0.00**
Windows Azure Platform - All Services		Zone 1	Data Transfer Out (GB)	0.010857	0	0.010857	0.007092	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
SQL Azure Database	Web Edition		Database (db/month)	0.838708	1	0	7.084908	**EUR 0.00**

Name	Type	Region	Resource	Consumed	Included	Billable	Rate	Amount
Windows Azure Compute			Compute Hours	250	750	0	0.085104	**EUR 0.00**
Windows Azure Compute	Extra Small		Compute Hours	613	750	0	0.03546	**EUR 0.00**
Windows Azure Storage			Storage Transactions (in 10,000s)	0.2178	5	0	0.007092	**EUR 0.00**