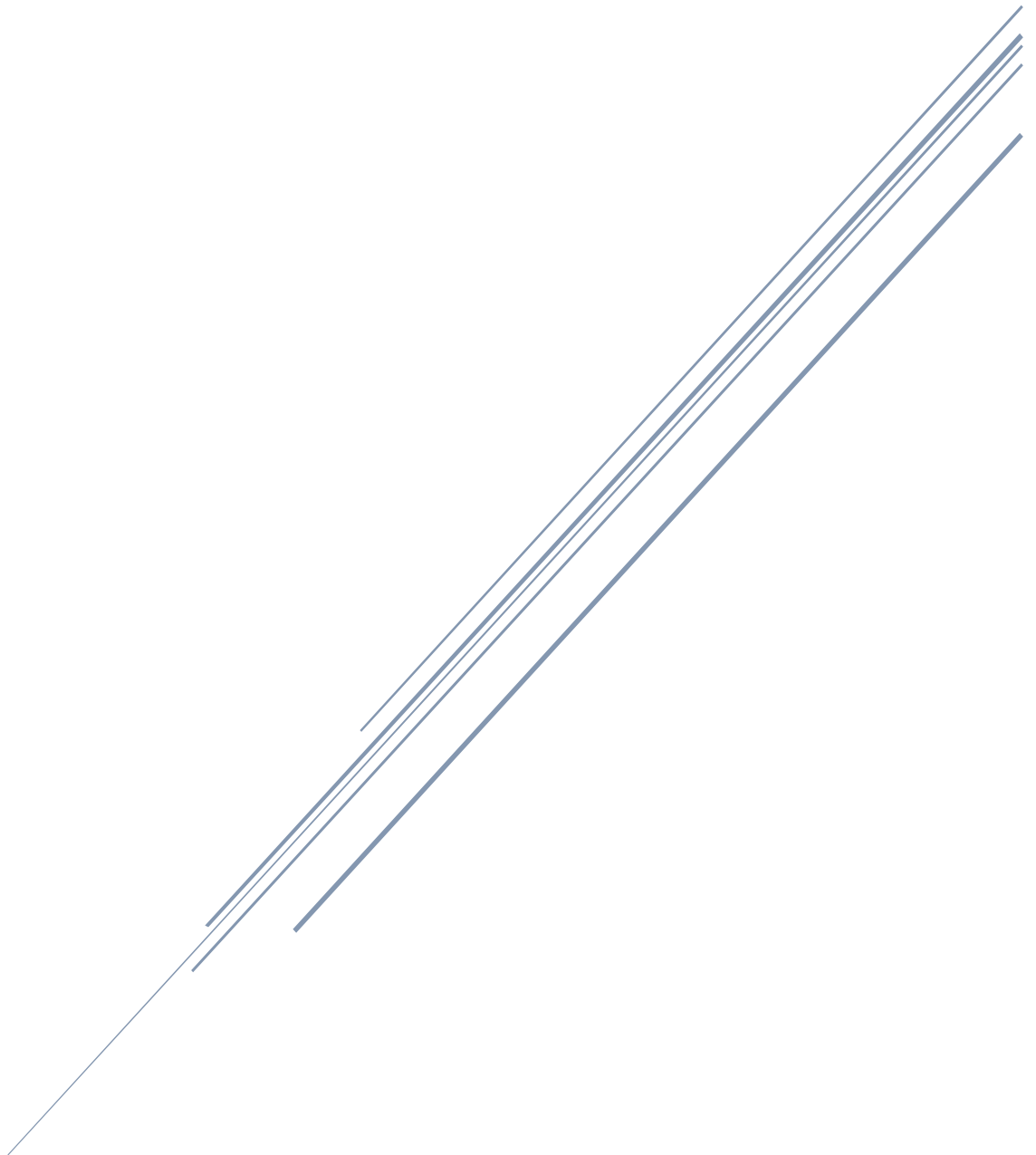


SOFTWARE APPLICATIONS INTEGRATION USING WEB SERVICES

COMP30231 - Service-Centric Cloud Computing



Ruairidh Taylor
N0629719

Implementation evaluation

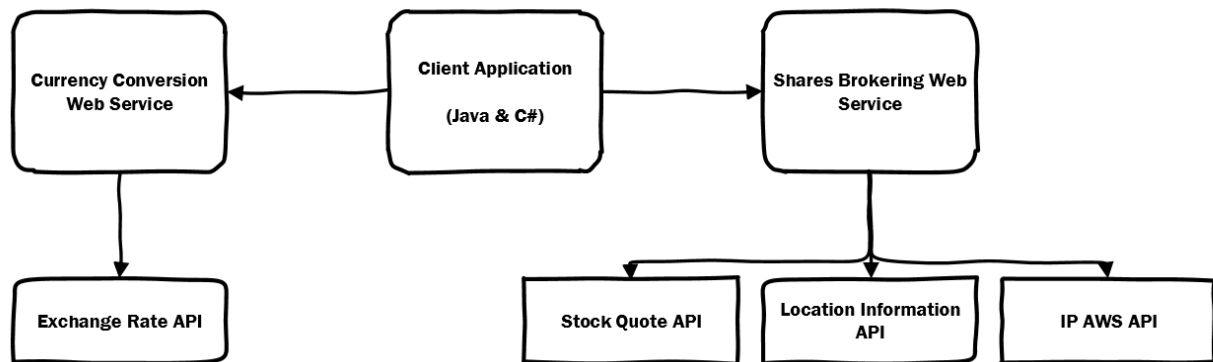


Figure 1 - Top Level Design Diagram

Self-Evaluation –

From Figure 1 you can see that my implementation has achieved all the core features of the assignment as well as some additional features such as additional APIs, and a C# client in addition to the Java client.

Although I think the assignment went well, I know that the quality of code isn't great as I haven't done much programming, however now that I know how web services work there are a lot of things I would do differently. If I was to start over, I would make sure I plan out the classes using class diagrams to make sure the program was completely object-oriented, I would implement user accounts to allow users to buy, track and sell shares and I would make the server multithreaded to make sure the stock prices are updated regularly.

Functional Description –

My implementation consists of a currency conversion web service and a share brokering web service, the currency conversion web service is linked to an external API which gets the most up to date conversion rates for over 150 currencies, so it covers all but four of the currencies from the enum. When the API is run the up to date rates are stored in a text file so if the API does go down it will have them stored. When the conversion is called it will get the currencies from the text file and add them to a list it will then check the currencies inputted against the currencies in the list and if it finds it, it will return the conversion rate or -1 if it's not there. Once both the currencies have been returned it will

check if it is equal to -1, if it is it will get the rate from the enum and work out conversion rate, this is then returned to the client and the list locally is converted to the target currency.

The broker web service has the option to return the full list of companies, update the share price via an API, buy shares and get the server location information by an API. To get the full list the web service will unmarshal the XML, put it into a list and return the full list. To update the price of the shares is very similar to the currency conversion API as it gets the prices, puts them into a text file, the file is then read into a list where if the name of the company is on the list it will update the price and marshal it back to XML, unfortunately I wasn't able to get the correct date put in place because the API is very sketchy.

Design Decisions –

A client connecting to 2 different web services – One of the major design decisions I made was to connect the client to 2 different web services rather than connecting to 1 web service and that web service connecting to the other. I decided to go down this route because it would mean less stress on the server as any conversion rate requests would go directly to the currency conversion WS and not have to hop through broker WS.

- API – For all my APIs when the client application is opened it will run all the APIs, once the API is run it stores the XML into a text file, this is so that if the API goes down, I will still have currencies that are as up to date as the last time the API was ran.
- Currency conversion WS – My currency conversion web service works by getting the up to date prices from the text file and putting it into a list, the 2 currencies you pass into the WS will first be checked against my list of conversion rates to get the most up to date ones and if it's not there it will then be checked against the enum.
- Stock quote API – Similar to the currency conversion WS it will get the company name and share price and put it into a list, I then have a loop which goes through the currencies in my list and checks against the APIs, if it finds the name in the API list it will update it to that price.
- Location Information API – I decided to implement the location information to get the servers IP address to retrieve its location I did this

to let the client know what server they are connecting to as if there was more than one server, they could see what server they are connected to and possibly switch between servers to get a better connection.

Analysis of Quality of services 'QoS'

Service Oriented Architecture (SOA) is a platform-independent service which can be accessed remotely and acted upon and updated independently. There are 2 main issues that affect my implementation of the broker web service, these being the number of concurrent connections and the XML payload due to parsing.

I tested the number of concurrent client's vs the connection time (ms), I started by testing on my laptop which you can see in Table 1 and then again on my desktop which you can see in Table 2, the rest of the testing can be found in the appendix.

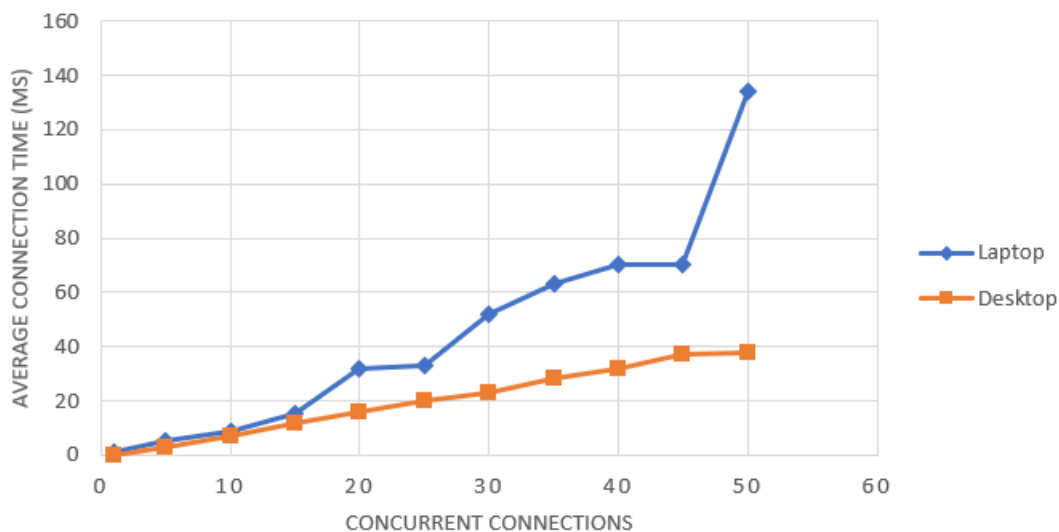
Concurrent Connections	Sample Size	Average Connection Time (ms)
1	14260	1
5	17018	5
10	22511	9
15	20458	15
20	12422	32
25	15449	33
30	11639	52
35	11313	63
40	11572	70
45	12954	70
50	7828	134

Table 1 - Laptops concurrent connections

Concurrent Connections	Sample Size	Average Connection Time (ms)
1	21160	0
5	31168	3
10	25252	7
15	23656	12
20	24890	16
25	24598	20
30	26388	23
35	25097	28
40	24930	32
45	24271	37
50	25819	38

Table 2 - Desktops concurrent connections

From both tests, you can see that as the number of concurrent clients increases so does the average connection time. I decided to run the 2 tests to show that if the amount of computer resources increases, the average connection time decreases as shown in Figure 2, the desktop being the more powerful of the two.

*Figure 2 - Concurrent connections vs Average connection time (ms)*

From my testing, I can conclude that increasing the number of resources given to the web service reduces the average connection time to the web service. Therefore if I was to scale up the number of concurrent users, I would also need to scale up the amount of computer resources.

The cloud computing model is an on-demand computing infrastructure, this would be able to facilitate my computing needs as the demand grows and falls throughout the day as the markets open and close. This means that as the number of users grows so will my demand for computing resources and fall again when the number of concurrent users decreases, cloud computing will be able to scale as needed.

Another challenge is increased payload due to using XML which causes a longer response time, this is because XML files need to be read, parsed and written to and only my program can do that. Whereas if I was to use SQL instead it would

reduce overhead as they're specifically used for storing and processing. SQL is also more robust which means it will scale better.

There are many factors that will affect the quality of a system or application. Flexibility, the ability of the software to manage the functionality without destroying the system. Maintainability and readability, maintainability is a little similar with flexibility, but it focusses on modifications about error correction. (Hashem H.Ramadan et al, 2017)

However, no matter how great it is, it has its issues when migrating from your current infrastructure to the cloud as a lot of things can go wrong and a lot of work needs to go into it –

- Cloud-amenable applications – When migrating to the cloud the application itself needs to be able to be migrated such as the transitioning from XML from text files into a SQL database. Some legacy software may not be compatible and either changed or to keep it on the current infrastructure.
- Security - This entails both the communication and data aspects, but also the physical/digital aspects. With everything being on the cloud and not physically on site if there was to be an attack on the company hosting the data and they closed off connections this would affect how you and the clients get the data.
- Cost – Migrating to the cloud is very expensive, not just because of the cost of the hosting itself (CPUs and storage) but because of staff training, transitioning between XML to SQL, staff training and the possibility of paying contractors to do the migration work itself.
- Downtime – When migrating to the cloud there is going to be some downtime when switching from the current infrastructure to the cloud infrastructure, in the example of broker web service there wouldn't be much backlash due to downtime as the switchover could take place outside of trading hours.

Application of Semantic Web and Linked Data technologies

Introduction

"The semantic web provides a common framework that allows data to be shared and reused across application, enterprise and community boundaries" (W3C, 2013). For the semantic web to work the information needs to be standardised so that it's understood by both humans and computers, this is called an ontology – "An ontology encompasses a representation, formal naming, and definition of the categories, properties, and relations between the concepts, data, and entities that substantiate one, many, or all domains" (J.S Hare et al, 2006). By using an ontology language, it allows the user to write a clear, detailed conceptualisation of domain models.

Currently, my search works by pattern recognition using Regex filter and can only search for information that is in the table. The semantic web and linked data can be used to improve my search implementation by accessing data from the web and linking it back to companies in my list.

Domain Analysis

Domain analysis of other refers to the subject of interest in my example I am going to be looking at supermarkets as an area of interest however there are a load more, I could investigate such as technology or utilities. However, to keep it short I'm only going to investigate the supermarket domain.

Semantic technologies could be implemented into my version of the web brokering service as it would all users to be able to search for something related to the company and it would return the company or companies that have a relation to the data. For example, the user could type in 'Supermarket' and in my case, it would return Tesco, Morrisons and Sainsburys. I could even go a step further and the user could type in 'German Supermarkets' and hypothetically it would return Lidl.

Ontology engineering

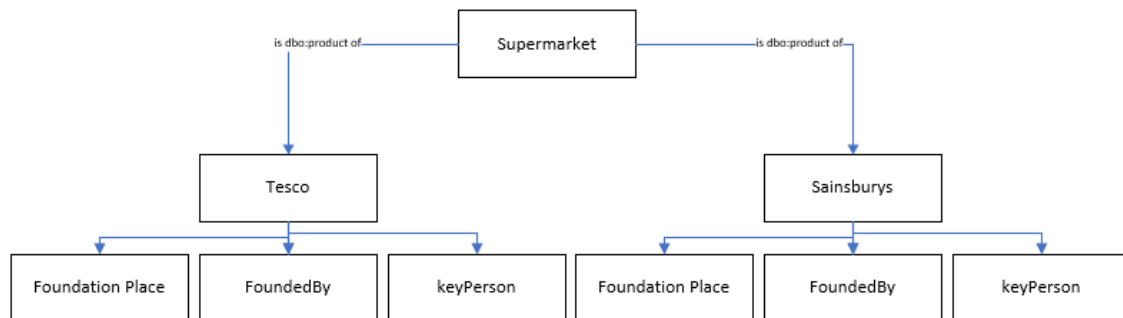


Figure 3 - Ontology Tree for finding supermarkets

An ontology tree shows the relationship between classes and subclasses - "Classes, subclasses and relations among entities are a very powerful tool for Web use. We can express a large number of relations among entities by assigning properties to classes and allowing subclasses to inherit such properties." (Berners-Lee, Hendler & Lassila, 2001).

The tags that are applied to each company will differ based on what has been added to the semantic web and what type of company it is, in Figure 3 I have used very generic subclasses for my tree but there are more specific tags that only supermarkets will have, such as the product of Tesco has the value 'supermarket'. These subclasses could be broken down even further allowing more companies to be linked to the search request.

For example, if the user added some more search criteria such as 'Supermarket, London' and would still return those two shops as they're based in London but let's say I added Lidl to the list it would be able to remove that because it can see in the tree that Lidl was not founded in London. However, by adding a multi-search filter would require me to completely rework how the normal search functionality worked unless I could work out where to split the words to find the correct subclasses to be returned.

Semantic Tagging and reasoning

Semantic tagging is powered by a knowledge graph that combines public and private data it helps in facilitating communication and finding information, by adding semantic tags to the information you are putting online you are providing

more information about your post. "A knowledge graph is a special kind of database which stores knowledge in a machine-readable form and provides a means for information to be collected, organised, shared, searched and utilised" (DBpedia, 2019). How does semantic tagging work?

This service analyses the text extracts concepts, identify topics, keywords, and important relationships, and disambiguates similar entities. The resulting semantic fingerprint of the document comprises metadata, aligned to a knowledge graph that serves as the foundation of all content management solutions - (ontotext, 2019).

Semantic tagging will be useful for my implementation as it allows information to be found faster. Semantic reasoning engine allows the user to more effectively search as a semantic searcher needs a reasoning engine to work.

Linked Data

Linked data is core to the semantic web as the semantic web is about making links between datasets that have an ontology. For the semantic web to become more powerful more events, people, locations and information need to be added. However, for information to be linked, it must follow the Resource Description Framework (RDF), and it must use a Uniform Resource Identifier (URI) to give unique names to anything. Therefore as the number of links between datasets increases, this would allow for more efficient and precise searching.

"The DBpedia community project extracts structured, multilingual knowledge from Wikipedia and makes it freely available on the Web using Semantic Web and Linked Data technologies" (Lehmann et al, 2015)

As DBpedia contains over 400 million facts that describe over 3.7 million things it would be a great place to get linked data about the companies in my broker web service. I could use an API to extract up to date information from DBpedia. From the data I've extracted, I could use SPARQL which is an RDF query language able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. So, in the case of getting the supermarkets, I would use SPARQL to find the 'supermarket' tag under the product ontology, this should return the names of the supermarkets which I can use to display in the company table.

References

BERNERS-LEE, T., HENDLER, J. and LASSILA, O. (2001). *The Semantic Web*. [ebook] Scientific American, pp.1-4. Available at: https://www-sop.inria.fr/acacia/cours/essi2006/Scientific%20American_%20Feature%20Article_%20The%20Semantic%20Web_%20May%202001.pdf [Accessed 30 Jan. 2019].

Computerphile (2013). *Cloud Computing (Cloudy with a Chance of Pizza) - Computerphile*. [video] Available at: <https://www.youtube.com/watch?v=7DgxjQ6Qd54> [Accessed 30 Jan. 2019].

Dbpedia.org. (2019). *About: http://dbpedia.org/ontology/*. [online] Available at: <http://dbpedia.org/ontology/> [Accessed 30 Jan. 2019].

H. Ramadan, H. and Kashyap, D. (2017). *Quality of Service (QoS) in Cloud Computing*. International Journal of Computer Science and Information Technologies. [online] pp.318, 320. Available at: <https://ijcsit.com/docs/Volume%208/vol8issue3/ijcsit2017080301.pdf> [Accessed 30 Jan. 2019].

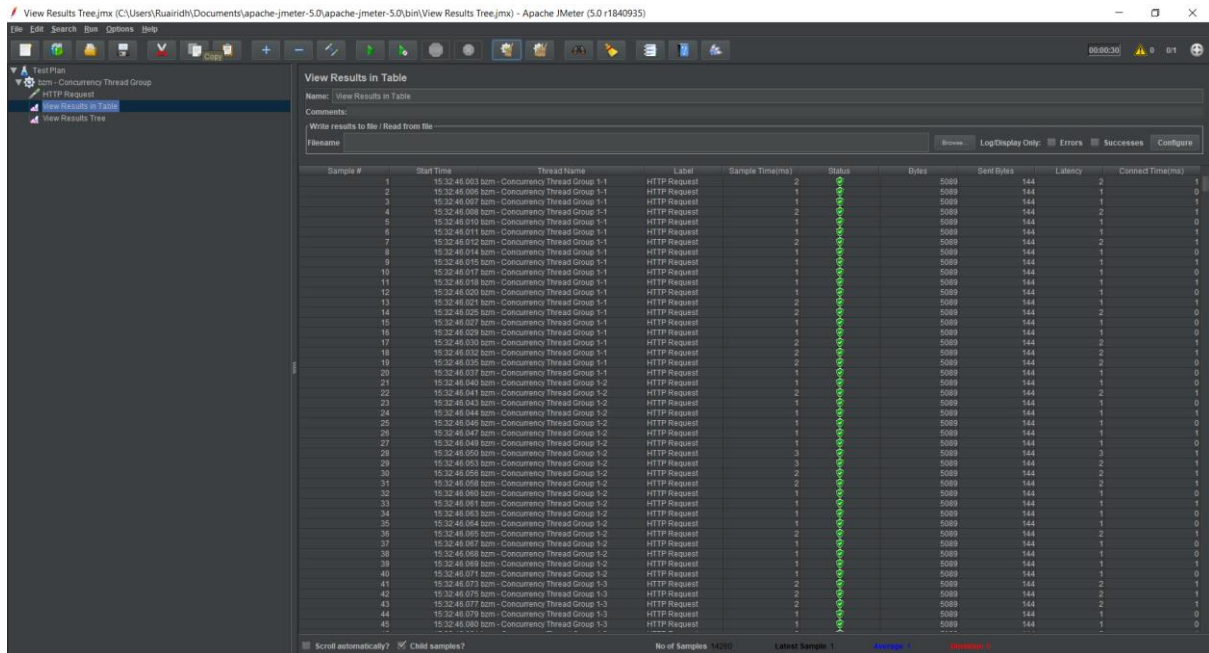
Lehmann, J., Isele, R., Jakob, M. and Jentzsch, A. (2015). DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, [online] 6(2), pp.167-195. Available at: <https://content.iospress.com/articles/semantic-web/sw134> [Accessed 30 Jan. 2019].

Ontotext. (2019). *Semantic Tagging | Ontotext*. [online] Available at: <https://www.ontotext.com/technology-solutions/semantic-tagging/> [Accessed 30 Jan. 2019].

S. Hare, J., H. Lewis, P., G. B. Enser, P. and J. Sandom, C. (2006). *Mind the Gap: Another look at the problem of the semantic gap in image retrieval*. [online] Southampton: University of Southampton. Available at: <https://eprints.soton.ac.uk/261887/1/article.pdf> [Accessed 30 Jan. 2019].

W3.org. (2019). *W3C Semantic Web Activity Homepage*. [online] Available at: <https://www.w3.org/2001/sw/> [Accessed 30 Jan. 2019].

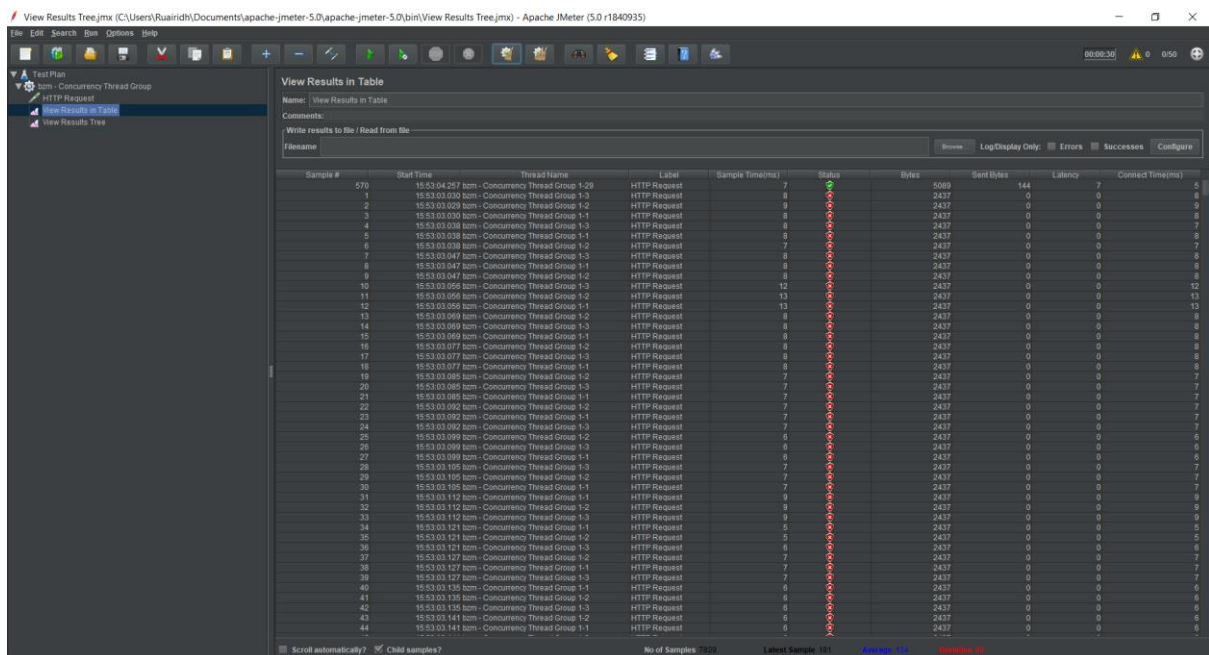
Appendix



The screenshot shows the Apache JMeter 5.0.1 View Results Tree. The test plan is named 'Test Plan' and contains a 'Concurrent Thread Group' with 1 thread. The results table shows 45 samples, all of which are 'HTTP Request' and have a status of 'Success'. The table columns are: Sample #, Start Time, Thread Name, Label, Sample Time(s), Status, Bytes, Sent Bytes, Latency, and Connect Time(s). The status column shows green checkmarks for all samples, indicating successful requests.

Sample #	Start Time	Thread Name	Label	Sample Time(s)	Status	Bytes	Sent Bytes	Latency	Connect Time(s)
1	15:32:46.063	Concurrent Thread Group 1-1	HTTP Request	2	Success	5089	144	2	1
2	15:32:46.066	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
3	15:32:46.067	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
4	15:32:46.068	Concurrent Thread Group 1-1	HTTP Request	2	Success	5089	144	2	1
5	15:32:46.070	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
6	15:32:46.071	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
7	15:32:46.072	Concurrent Thread Group 1-1	HTTP Request	2	Success	5089	144	2	1
8	15:32:46.074	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
9	15:32:46.075	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
10	15:32:46.076	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
11	15:32:46.078	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
12	15:32:46.080	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
13	15:32:46.081	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
14	15:32:46.082	Concurrent Thread Group 1-1	HTTP Request	2	Success	5089	144	2	0
15	15:32:46.087	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
16	15:32:46.089	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
17	15:32:46.090	Concurrent Thread Group 1-1	HTTP Request	2	Success	5089	144	2	1
18	15:32:46.092	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
19	15:32:46.093	Concurrent Thread Group 1-1	HTTP Request	2	Success	5089	144	2	1
20	15:32:46.097	Concurrent Thread Group 1-1	HTTP Request	1	Success	5089	144	1	0
21	15:32:46.098	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
22	15:32:46.099	Concurrent Thread Group 1-2	HTTP Request	2	Success	5089	144	2	1
23	15:32:46.103	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
24	15:32:46.104	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
25	15:32:46.106	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
26	15:32:46.107	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	1
27	15:32:46.108	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
28	15:32:46.109	Concurrent Thread Group 1-2	HTTP Request	3	Success	5089	144	3	1
29	15:32:46.110	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
30	15:32:46.112	Concurrent Thread Group 1-2	HTTP Request	2	Success	5089	144	2	1
31	15:32:46.116	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	1
32	15:32:46.118	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
33	15:32:46.119	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	1
34	15:32:46.123	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
35	15:32:46.124	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
36	15:32:46.125	Concurrent Thread Group 1-2	HTTP Request	2	Success	5089	144	2	1
37	15:32:46.127	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
38	15:32:46.128	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
39	15:32:46.129	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	1
40	15:32:46.131	Concurrent Thread Group 1-2	HTTP Request	1	Success	5089	144	1	0
41	15:32:46.132	Concurrent Thread Group 1-3	HTTP Request	2	Success	5089	144	2	1
42	15:32:46.135	Concurrent Thread Group 1-3	HTTP Request	2	Success	5089	144	2	1
43	15:32:46.137	Concurrent Thread Group 1-3	HTTP Request	1	Success	5089	144	1	0
44	15:32:46.139	Concurrent Thread Group 1-3	HTTP Request	1	Success	5089	144	1	0
45	15:32:46.140	Concurrent Thread Group 1-3	HTTP Request	1	Success	5089	144	1	0

Figure 4 - Laptop testing (1 concurrent connection)



The screenshot shows the Apache JMeter 5.0.1 View Results Tree. The test plan is named 'Test Plan' and contains a 'Concurrent Thread Group' with 50 threads. The results table shows 44 samples, all of which are 'HTTP Request' and have a status of 'Success'. The table columns are: Sample #, Start Time, Thread Name, Label, Sample Time(s), Status, Bytes, Sent Bytes, Latency, and Connect Time(s). The status column shows green checkmarks for all samples, indicating successful requests.

Sample #	Start Time	Thread Name	Label	Sample Time(s)	Status	Bytes	Sent Bytes	Latency	Connect Time(s)
570	15:53:04.257	Concurrent Thread Group 1-26	HTTP Request	7	Success	5089	144	7	8
1	15:53:03.030	Concurrent Thread Group 1-3	HTTP Request	8	Success	2437	0	8	8
2	15:53:03.029	Concurrent Thread Group 1-2	HTTP Request	9	Success	2437	0	9	9
3	15:53:03.030	Concurrent Thread Group 1-1	HTTP Request	8	Success	2437	0	8	8
4	15:53:03.038	Concurrent Thread Group 1-3	HTTP Request	8	Success	2437	0	8	8
5	15:53:03.038	Concurrent Thread Group 1-1	HTTP Request	8	Success	2437	0	8	8
6	15:53:03.039	Concurrent Thread Group 1-2	HTTP Request	7	Success	2437	0	7	7
7	15:53:03.047	Concurrent Thread Group 1-3	HTTP Request	8	Success	2437	0	8	8
8	15:53:03.047	Concurrent Thread Group 1-1	HTTP Request	8	Success	2437	0	8	8
9	15:53:03.047	Concurrent Thread Group 1-2	HTTP Request	8	Success	2437	0	8	8
10	15:53:03.056	Concurrent Thread Group 1-3	HTTP Request	12	Success	2437	0	12	12
11	15:53:03.056	Concurrent Thread Group 1-2	HTTP Request	13	Success	2437	0	13	13
12	15:53:03.056	Concurrent Thread Group 1-1	HTTP Request	13	Success	2437	0	13	13
13	15:53:03.069	Concurrent Thread Group 1-2	HTTP Request	8	Success	2437	0	8	8
14	15:53:03.069	Concurrent Thread Group 1-3	HTTP Request	8	Success	2437	0	8	8
15	15:53:03.069	Concurrent Thread Group 1-1	HTTP Request	8	Success	2437	0	8	8
16	15:53:03.077	Concurrent Thread Group 1-2	HTTP Request	8	Success	2437	0	8	8
17	15:53:03.077	Concurrent Thread Group 1-3	HTTP Request	8	Success	2437	0	8	8
18	15:53:03.077	Concurrent Thread Group 1-1	HTTP Request	8	Success	2437	0	8	8
19	15:53:03.085	Concurrent Thread Group 1-2	HTTP Request	7	Success	2437	0	7	7
20	15:53:03.085	Concurrent Thread Group 1-3	HTTP Request	7	Success	2437	0	7	7
21	15:53:03.086	Concurrent Thread Group 1-1	HTTP Request	7	Success	2437	0	7	7
22	15:53:03.092	Concurrent Thread Group 1-2	HTTP Request	7	Success	2437	0	7	7
23	15:53:03.092	Concurrent Thread Group 1-1	HTTP Request	7	Success	2437	0	7	7
24	15:53:03.092	Concurrent Thread Group 1-3	HTTP Request	7	Success	2437	0	7	7
25	15:53:03.099	Concurrent Thread Group 1-2	HTTP Request	6	Success	2437	0	6	6
26	15:53:03.099	Concurrent Thread Group 1-3	HTTP Request	6	Success	2437	0	6	6
27	15:53:03.099	Concurrent Thread Group 1-1	HTTP Request	6	Success	2437	0	6	6
28	15:53:03.105	Concurrent Thread Group 1-3	HTTP Request	7	Success	2437	0	7	7
29	15:53:03.105	Concurrent Thread Group 1-2	HTTP Request	7	Success	2437	0	7	7
30	15:53:03.105	Concurrent Thread Group 1-1	HTTP Request	7	Success	2437	0	7	7
31	15:53:03.112	Concurrent Thread Group 1-1	HTTP Request	9	Success	2437	0	9	9
32	15:53:03.112	Concurrent Thread Group 1-2	HTTP Request	9	Success	2437	0	9	9
33	15:53:03.112	Concurrent Thread Group 1-3	HTTP Request	9	Success	2437	0	9	9
34	15:53:03.121	Concurrent Thread Group 1-1	HTTP Request	5	Success	2437	0	5	5
35	15:53:03.121	Concurrent Thread Group 1-2	HTTP Request	5	Success	2437	0	5	5
36	15:53:03.121	Concurrent Thread Group 1-3	HTTP Request	6	Success	2437	0	6	6
37	15:53:03.127	Concurrent Thread Group 1-2	HTTP Request	7	Success	2437	0	7	7
38	15:53:03.127	Concurrent Thread Group 1-1	HTTP Request	7	Success	2437	0	7	7
39	15:53:03.127	Concurrent Thread Group 1-3	HTTP Request	7	Success	2437	0	7	7
40	15:53:03.135	Concurrent Thread Group 1-1	HTTP Request	6	Success	2437	0	6	6
41	15:53:03.135	Concurrent Thread Group 1-2	HTTP Request	6	Success	2437	0	6	6
42	15:53:03.135	Concurrent Thread Group 1-3	HTTP Request	6	Success	2437	0	6	6
43	15:53:03.141	Concurrent Thread Group 1-2	HTTP Request	6	Success	2437	0	6	6
44	15:53:03.141	Concurrent Thread Group 1-1	HTTP Request	6	Success	2437	0	6	6

Figure 5 - Laptop testing (50 concurrent connections)

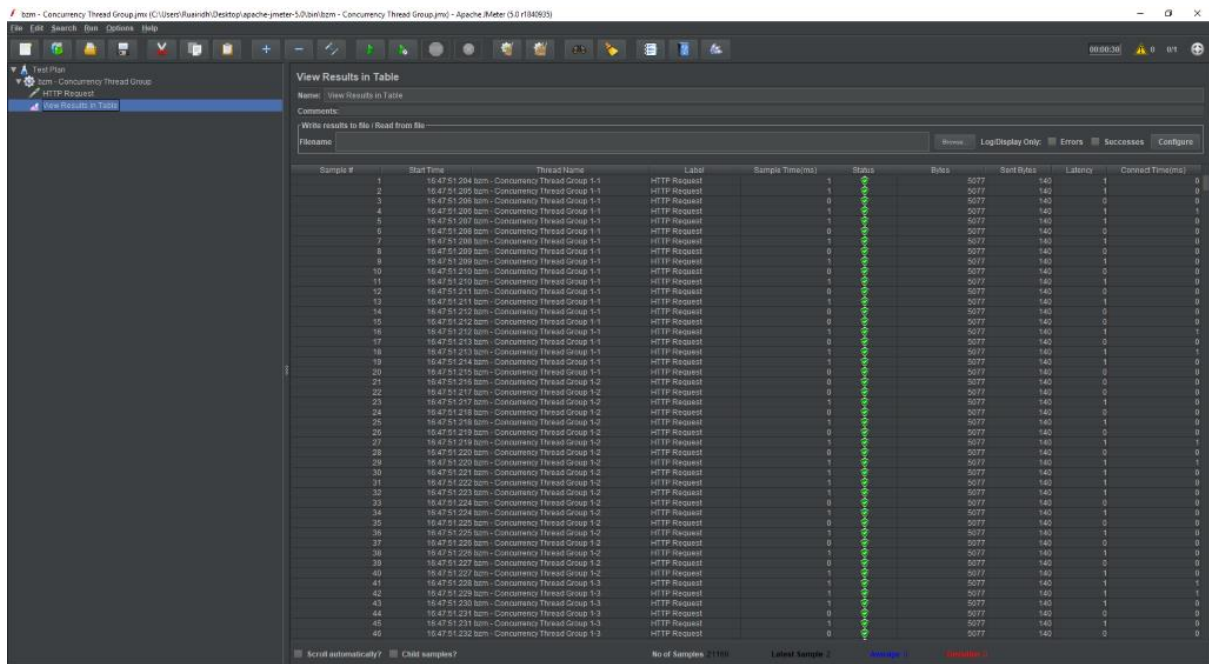


Figure 6 shows the Apache JMeter Desktop interface for a single concurrent connection test. The 'View Results in Table' window displays a list of 45 samples, all with a status of 'Success' and a latency of 140ms. The test is named 'HTTP Request' and is part of a 'Concurrency Thread Group'.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Send Bytes	Latency	Connect Time(ms)
1	16.47.51.284	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
2	16.47.51.285	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
3	16.47.51.286	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
4	16.47.51.288	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
5	16.47.51.287	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
6	16.47.51.288	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
7	16.47.51.288	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
8	16.47.51.289	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
9	16.47.51.289	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
10	16.47.51.210	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
11	16.47.51.210	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
12	16.47.51.211	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
13	16.47.51.211	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
14	16.47.51.212	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
15	16.47.51.212	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
16	16.47.51.212	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
17	16.47.51.213	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
18	16.47.51.213	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	1
19	16.47.51.214	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
20	16.47.51.215	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
21	16.47.51.216	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
22	16.47.51.217	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
23	16.47.51.217	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
24	16.47.51.218	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
25	16.47.51.218	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
26	16.47.51.219	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
27	16.47.51.219	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
28	16.47.51.220	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
29	16.47.51.220	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	1
30	16.47.51.221	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
31	16.47.51.222	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
32	16.47.51.223	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
33	16.47.51.224	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
34	16.47.51.224	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
35	16.47.51.225	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
36	16.47.51.225	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
37	16.47.51.226	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
38	16.47.51.226	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
39	16.47.51.227	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
40	16.47.51.227	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
41	16.47.51.228	Concurrency Thread Group 1-3	HTTP Request	1	Success	5077	140	1	0
42	16.47.51.228	Concurrency Thread Group 1-3	HTTP Request	1	Success	5077	140	1	1
43	16.47.51.229	Concurrency Thread Group 1-3	HTTP Request	1	Success	5077	140	1	0
44	16.47.51.231	Concurrency Thread Group 1-3	HTTP Request	1	Success	5077	140	1	0
45	16.47.51.231	Concurrency Thread Group 1-3	HTTP Request	1	Success	5077	140	1	0
46	16.47.51.232	Concurrency Thread Group 1-3	HTTP Request	0	Success	5077	140	0	0

Figure 6 - Desktop testing (1 concurrent connection)

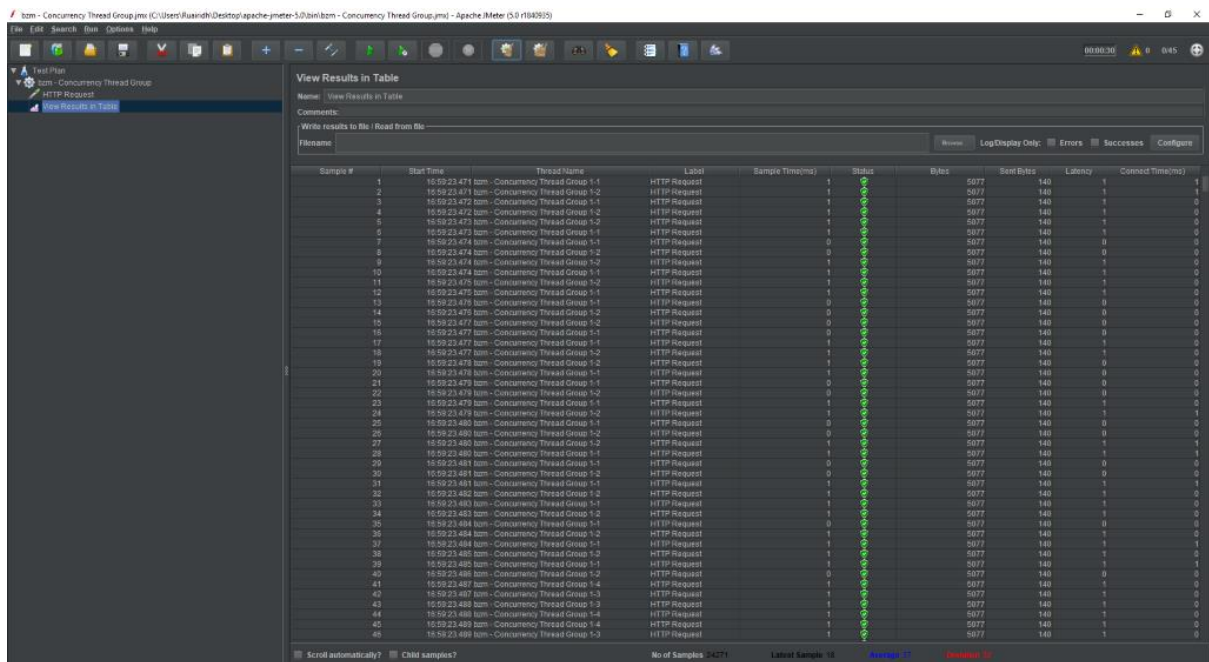


Figure 7 shows the Apache JMeter Desktop interface for a 50 concurrent connections test. The 'View Results in Table' window displays a list of 45 samples, all with a status of 'Success' and a latency of 140ms. The test is named 'HTTP Request' and is part of a 'Concurrency Thread Group'.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Send Bytes	Latency	Connect Time(ms)
1	16.59.23.471	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	1
2	16.59.23.471	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	1
3	16.59.23.472	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
4	16.59.23.472	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
5	16.59.23.473	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
6	16.59.23.473	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
7	16.59.23.474	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
8	16.59.23.474	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
9	16.59.23.474	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
10	16.59.23.474	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
11	16.59.23.475	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
12	16.59.23.475	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
13	16.59.23.476	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
14	16.59.23.476	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
15	16.59.23.477	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
16	16.59.23.477	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
17	16.59.23.477	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
18	16.59.23.477	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
19	16.59.23.478	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	0	0
20	16.59.23.478	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	0	0
21	16.59.23.479	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
22	16.59.23.479	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
23	16.59.23.479	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	1
24	16.59.23.479	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	1
25	16.59.23.480	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
26	16.59.23.480	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	1
27	16.59.23.480	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	1
28	16.59.23.480	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
29	16.59.23.481	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
30	16.59.23.481	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
31	16.59.23.481	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
32	16.59.23.480	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
33	16.59.23.483	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
34	16.59.23.483	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
35	16.59.23.484	Concurrency Thread Group 1-1	HTTP Request	0	Success	5077	140	0	0
36	16.59.23.484	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
37	16.59.23.486	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	1
38	16.59.23.487	Concurrency Thread Group 1-2	HTTP Request	1	Success	5077	140	1	0
39	16.59.23.480	Concurrency Thread Group 1-1	HTTP Request	1	Success	5077	140	1	0
40	16.59.23.486	Concurrency Thread Group 1-2	HTTP Request	0	Success	5077	140	0	0
41	16.59.23.487	Concurrency Thread Group 1-4	HTTP Request	1	Success	5077	140	1	0
42	16.59.23.487	Concurrency Thread Group 1-3	HTTP Request	1	Success	5077	140	1	0
43	16.59.23.488	Concurrency Thread Group 1-3	HTTP Request	1	Success	5077	140	1	0
44	16.59.23.488	Concurrency Thread Group 1-4	HTTP Request	1	Success	5077	140	1	0
45	16.59.23.489	Concurrency Thread Group 1-4	HTTP Request	1	Success	5077	140	1	0
46	16.59.23.489	Concurrency Thread Group 1-3	HTTP Request	1	Success	5077	140	1	0

Figure 7 - Desktop testing (50 concurrent connections)

There are tests in-between but it's too much printing.