

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

10/27/2019

CAB432

Assignment 2

Scaling and Persistence

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Kunal Chand & Yu Gen Yeap
N1002345 & N 9470379

1 Introduction

1.1 Purpose

This application allows users to check if a number is prime (as well as providing batch processing for this capability), the factors of a number, and to retrieve certain prime numbers (e.g. the 5th or 100th prime number).

While other sites/applications allow for users retrieve a list of prime numbers in bulk (that is it will return all prime numbers in certain ranges), this application will allow users much more functionality offering users to select certain prime numbers instead of those in bulk.

1.2 Services Used

This application does not use any external APIs, instead opting for custom functionality. This was approved by a tutor and the professor as shown in Appendix 1. Below, the endpoints of the program are stated, along with their description, and an example input when hitting the endpoint (directly using the URL).

1.2.1 Endpoint 1: isPrime

This end point allows the user to query whether a certain number is prime or not.

Original Endpoint	<host>:3000/isPrime?number=
Example	localhost:3000/isPrime?number=56

1.2.2 Endpoint 2: batchPrime

This end point allows the user to query which numbers in a list are prime or not.

Original Endpoint	<host>:3000/batchPrime?list=
Example	localhost:3000/batchPrime?list=34,67,89,2,3,4,7,5,3,2,1,3,90

1.2.3 Endpoint 3: factorise

This end point allows the user to query what factors a number has.

Original Endpoint	<host>:3000/factorise?number=
Example	localhost:3000/factorise?number=360

1.2.4 Endpoint 4: nextPrime

This end point allows the user to query what the next prime number is after a certain number.

Original Endpoint	<host>:3000/nextPrime?number=
Example	localhost:3000/nextPrime?number=126

1.2.5 Endpoint 5: orderPrime

This end point allows the user to query what the nth prime number is, where n is the user's input.

Original Endpoint	<host>:3000/orderPrime?number=
Example	localhost:3000/orderPrime?number=567

2 Use Cases

2.1 Use Case 1

As a	Mathematician
I want	To retrieve significant amounts of prime numbers
So that	I can observe if there is a pattern or not (thereby attempt to write an equation for the prime numbers)

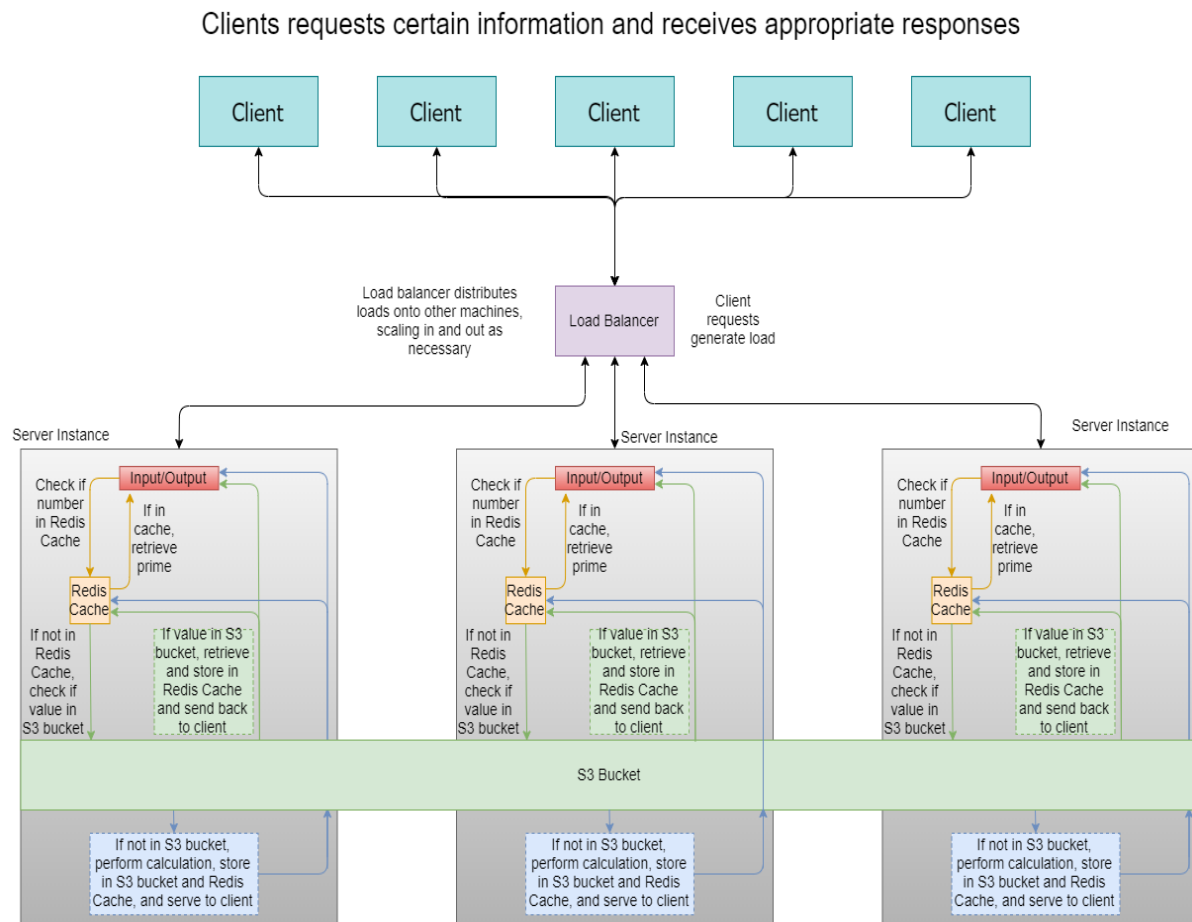
2.2 Use Case 2

As a	Cryptanalyst
I want	To get the factors of a certain number (either by factorising (factorise endpoint) or brute force (next prime endpoint))
So that	I can decrypt communications between two parties who are using encrypted communications, where the cipher is the ASCII number of each character multiplied by a certain number (key).

3 Technical Breakdown

3.1 Architecture (overall)

The figure below describes the overall system architecture, displaying the data flow from the client's request to the server process.



At the top, many clients can hit the application, which sits behind a load balancer. This load balancer ensures that the application remains responsive to the client's needs, adapting with the load requirements. The more load there is, the more server instances are activated, which decreases the load on other servers and ensures that the application is responsive.

Within a server instance, the client's request is processed. The request is first checked against the Redis cache (which is local to each instance). If the request is in the cache, then the server can retrieve it from the cache and serve it to the client (via the load balancer). If it is not in the Redis cache, then proceed to check in the S3 bucket.

If the client's query is in the S3 bucket, the server will retrieve it and store it in the Redis cache as well as respond to the client. If the query is not in S3, the server will then proceed to perform the calculation required to fulfil the client's request.

Once the server has performed the calculation, it will store the query and value in both the Redis cache and the S3 bucket.

Note that the Redis cache is a local memory instance, which means that if the client were to first send a request to the load balancer, it would be stored in a machine's Redis cache. If the client were to send the request again, but this time the load balancer changed the instance on which the client's request is processed, it may not be in the Redis cache of the machine.

The S3 bucket, however, is available to all the instances. This means that if a client's request has been processed before, the request will be stored in the S3 bucket, making it available to all other instances. If the client was to perform the same request again but the load balancer changes the instance on which the client's request was performed on, then this different instance is still able to retrieve the request from the S3 bucket as it has been processed and stored before.

Finally, note that since the application does not interface with any other APIs, the load balancer is client facing, which means that the only significant loads are from either memory (size of request or calculation), CPU (request processing), or network requests (how many clients are using it simultaneously).

On the client-side, a form is presented to the user, as shown below. This form will allow the user to interact with the application by hitting different application endpoints for different functions that are requested by the user.

Welcome to the Prime Number Calculator!

Select from a list of calculations to perform on a number:

Enter a number to see if it is prime:

Is this Prime?

Enter a number to see what ordered prime number it is:

Which Order Prime?

Enter a number to see which number after it, is prime:

Next Prime?

Enter a number to see it's factors:

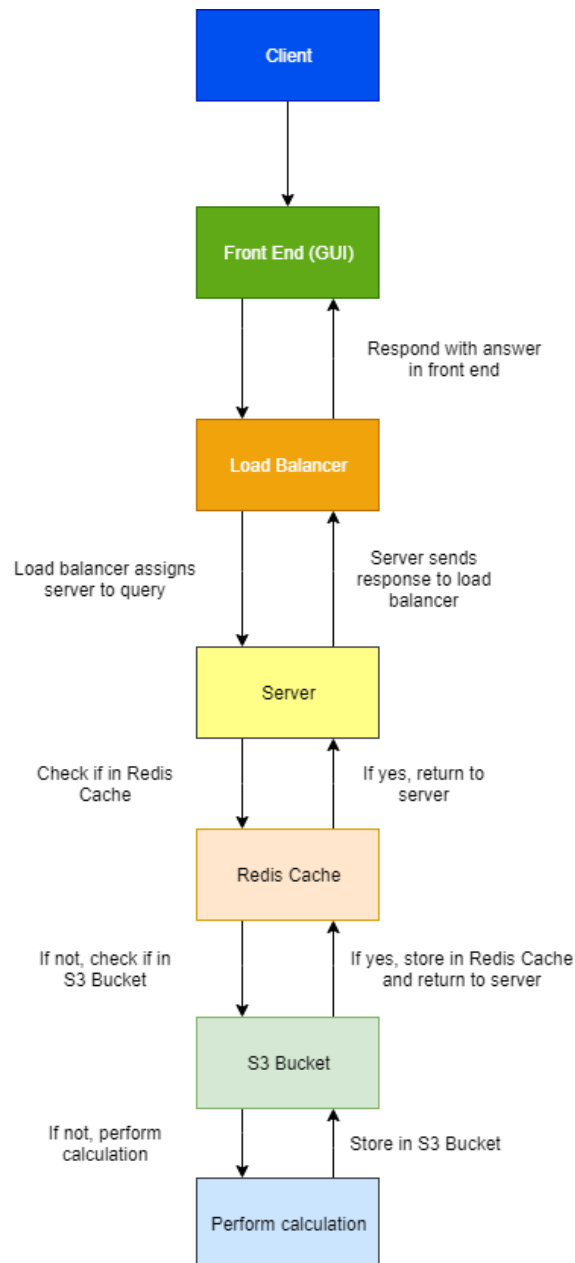
Factorise

Enter a list of numbers separated by a comma to see if they are prime:

Batch Primes

3.2 Architecture (endpoints)

While there are 5 different endpoints in the application, the dataflow through each endpoint is the same, and takes on the general form shown below.



For every endpoint, the data would start when the user inputs values into the form available. After submitting the request, the client-side would then send this information to the load balancer. The load balancer would then see which server instances were available or had little load upon them, and would assign the request to be handled by that server. In the event that all instances were under heavy load, the load balancer would create another server instance to reduce the load on the existing instances.

After having assigned the request to a server instance, the server instance would then check if the request has been performed before. It does this by first checking the Redis cache (a local form of persistent storage). If it is in the Redis cache of the server, then the server will simply retrieve the value and send it back to the load balancer, which then sends it back to the original client.

If the value is not in the Redis cache, then the server will check the S3 Bucket (a form of persistence available to all instances running this application). If the value is in the S3 bucket, it will retrieve the

value, store it in the Redis cache, and send it back to the load balancer, which then sends it back to the client.

If the value is completely new (not in Redis cache or S3 bucket), then the server will process the value. Once calculated, the answer will be stored in both the Redis cache and the S3 bucket. It will then be returned to the load balancer, which will then return it to the client.

3.3 Client/Server demarcation of responsibilities

The client-side is defined as the graphical user interface (GUI). It is responsible for allowing the user to interact with the application by entering values and selecting which function to execute, displaying the response from the server, informing the user if they have entered invalid data, and to display an error page if the application has crashed or encountered an error.

The server-side is defined as the load balancer and every server instance that is created by the load balancer, as well as the S3 bucket available to all server instances. The server-side is responsible for accepting the client-side input, checking if it has already been processed and stored in the Redis cache or the S3 bucket, or if it is not, then perform the calculation and store it in the Redis cache and S3 bucket. It is also responsible for sending the answer (response) back to the client.

Since the load balancer is part of the server-side, the server-side is also responsible for ensuring that the application remains responsive, even under heavy load. This is achieved by creating many server instances (scaling out) to serve the client's needs and remain responsive during peak usage. After the peak usage, the application may then begin to scale in (terminate server instances) to save resources and money during off-peak usage.

4 Scaling and Performance

Since the application was mainly performing operations on numbers (scalar), scaling according to CPU usage seemed to be the best option, as more clients would generate load on the CPU aspect of the application far more than the network requests (which was another valid metric for scaling out). Also of note is that in most of the calculations, the general algorithm is based on a search algorithm (find numbers with certain properties within a certain range).

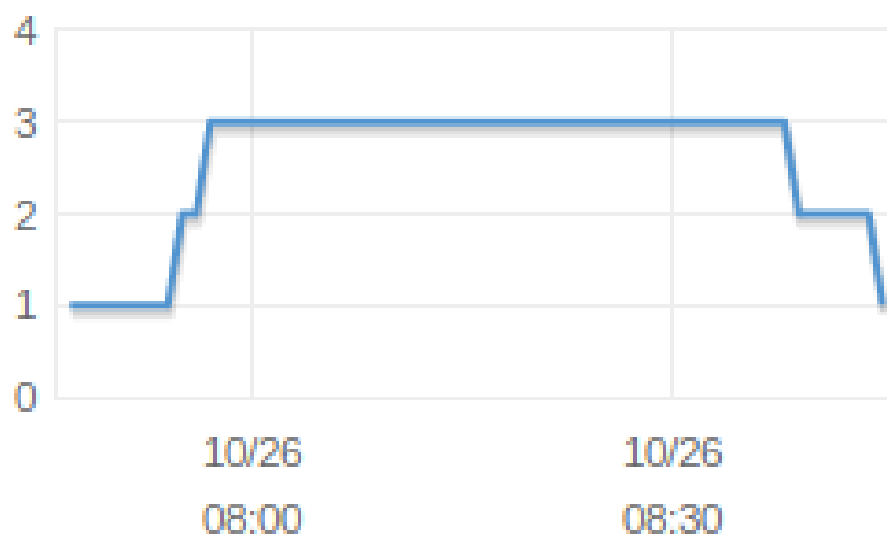
The setting for scaling out was that if the overall CPU usage should have an average of 40%. If the average was above this threshold, then the application would scale out, which means that it would create more server instances running the application and that were capable of handling client requests. It would check the average CPU usage every minute.

The setting for scaling in (terminating server instances due to low demand to save cost and to be more efficient) was that when the average CPU usage was below 40%, the application would terminate a server instance. It would perform this check once every minute.

While the application performs well when there are many clients, it does not perform as great when there is a single client that has placed really large numbers into the application. This is because the load balancer would only see how many requests are coming into the application, not how large they are. Based on this information, it will decide to either scale out or in. So when the client enters a single number that is really large, the application will remain at one server instance, but this server instance will be required to perform the search-based algorithm on a very large search space. This is the reason the application may slow down.

Below is a diagram of the application scaling out and in according to demand. Postman was used to input multiple requests over a prolonged period (5 minutes) where the client entered 9 digit numbers into the application and queried the primality of these numbers, after which the application would be required to scale out and then back in (after the calculations were complete).

In Service Instances (Count)



It can be seen that while the application rapidly scales out to cope with the demand, scaling in takes some time. This is because the duration of the test was approximately 45 minutes, and most of that

time the CPU usage was at 100%. This meant that for the application to scale back in, the CPU usage had to be below 40% for a prolonged period of time before the load application would meet the scaling-in setting. Added to this was that the client requested relatively large numbers (9 digit numbers), which on average would take the application approximately 147 seconds to process, and since the application scales out and not up, this calculation time was the same across all server instances.

Another contributing factor to the prolonged scale in period was that there were limitations when scaling out. Since the application was deployed using an AWS Educate account, the maximum server instances allowed was 4. Another project had already consumed one machine, which meant that the application could only scale out to 3 to cope with the load. Since the load was prolonged, this inevitably lead to requests being held in queue for approximately 2 minutes before they were processed. This is a general disadvantage of scaling out rather than up.

The deployment guide is shown in appendix 2.

5 Test Plan



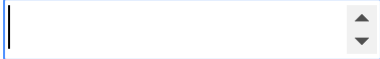




5.1 Positive

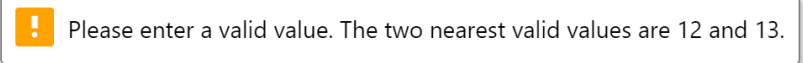

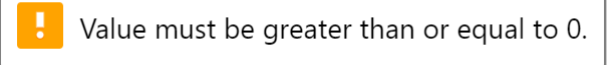
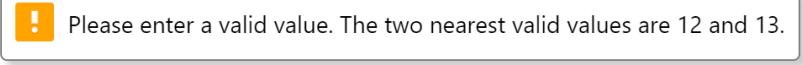

<u>Test</u>	<u>Description</u>	<u>Input</u>	<u>Output Description</u>	<u>Output</u>	<u>Pass/Fail</u>
Application home page	Can the client hit the application's homepage when querying the domain?	Load balancer's IP address	Form showing various inputs under various functions	<p>Welcome to the Prime Number Calculator!</p> <p>Select from a list of calculations to perform on a number:</p> <p>Enter a number to see if it is prime:</p> <input type="text"/> <input type="button" value="Is this Prime?"/> <p>Enter a number to see what ordered prime number it is:</p> <input type="text"/> <input type="button" value="Which Order Prime?"/> <p>Enter a number to see which number after it, is prime:</p> <input type="text"/> <input type="button" value="Next Prime?"/> <p>Enter a number to see it's factors:</p> <input type="text"/> <input type="button" value="Factorise"/> <p>Enter a list of numbers separated by a comma to see if they are prime:</p> <input type="text"/> <input type="button" value="Batch Primes"/>	PASS
isPrime valid input	Returns true if the number is a prime number	29	True/false statement	<p>Is 29 prime?</p> <p>true</p> <p>Result retrieved from server calculation</p> <input type="button" value="Back to Homepage"/>	PASS

Factorise valid input	Factors of number should be returned	360	Factors of the number	<p>The factors of 360 are:</p> <p>2 and 180</p> <p>3 and 120</p> <p>4 and 90</p> <p>5 and 72</p> <p>6 and 60</p> <p>8 and 45</p> <p>9 and 40</p> <p>10 and 36</p> <p>12 and 30</p> <p>15 and 24</p> <p>18 and 20</p> <p>Result retrieved from server calculation</p> <p>Back to Homepage</p>	PASS
nextPrime() valid input	Next prime number of inputted value should be returned	200	Prime number marginally greater than input number should be returned	<p>The next prime number bigger than 200 is</p> <p>211</p> <p>Result retrieved from server calculation</p> <p>Back to Homepage</p>	PASS

orderPrime() valid input	This would return the inputted instance of the prime number (e.g. 5 would return the 5 th prime number)	50	Prime number should be returned that corresponds to the order the number appears in the ascending list of primes	<p>The 50th prime number is</p> <p>229</p> <p>Result retrieved from server calculation</p> <p>Back to Homepage</p>	PASS
batchPrime() valid input (single)	Returns true if the single number entered is prime	20	Single True/False statement	<p>The following numbers primality are listed below:</p> <p>20 : false</p> <p>Result retrieved from server calculation</p> <p>Back to Homepage</p>	PASS
batchPrime() valid input (multiple)	Returns either true or false for every number entered	20, 30, 47	True/False statements describing each number's primality	<p>The following numbers primality are listed below:</p> <p>20 : false</p> <p>30 : false</p> <p>47 : true</p> <p>Result retrieved from server calculation</p> <p>Back to Homepage</p>	PASS

5.2 Negative

Test	Description	Input	Output Description	Program Output	Pass/Fail
isPrime() invalid input (- negative number)	Prevent user from entering invalid input (negative number)	-7	Should show warning to user that value is invalid	 Value must be greater than or equal to 0.	PASS
isPrime() invalid input (decimal number)	Prevent user from entering invalid input (decimal number)	12.3	Should show warning to user that value is invalid	 Please enter a valid value. The two nearest valid values are 12 and 13.	PASS
isPrime() invalid input (character)	Prevent user from entering invalid input (non-numeric)	hello	Input remains empty		PASS
factorise() invalid input (negative number)	Prevent user from entering invalid input (negative number)	-7	Should show warning to user that value is invalid	 Value must be greater than or equal to 0.	PASS
factorise() invalid input (decimal number)	Prevent user from entering invalid input (decimal number)	12.3	Should show warning to user that value is invalid	 Please enter a valid value. The two nearest valid values are 12 and 13.	PASS
factorise() invalid input (character)	Prevent user from entering invalid input (non-numeric)	hello	Input remains empty		PASS
nextPrime() invalid input	Prevent user from entering invalid input	-7	Should show warning to user that value is invalid	 Value must be greater than or equal to 0.	PASS

(negative number)	(negative number)				
nextPrime() invalid input (decimal number)	Prevent user from entering invalid input	12.3	Should show warning to user that value is invalid		PASS
nextPrime() invalid input (character)	Prevent user from entering invalid input (non-numeric)	hello	Input remains empty		PASS
orderPrime() invalid input (negative number)	Prevent user from entering invalid input (negative number)	-7	Should show warning to user that value is invalid		PASS
orderPrime() invalid input (decimal number)	Prevent user from entering invalid input	12.3	Should show warning to user that value is invalid		PASS
orderPrime() invalid input (character)	Prevent user from entering invalid input (non-numeric)	hello	Input remains empty		PASS

5.3 Edge Cases

There were no edge cases to be tested in this program.

5.4 Non-Functional cases

<u>Test</u>	<u>Description</u>	<u>Input</u>	<u>Output Description</u>	<u>Output</u>	<u>Pass/Fail</u>
Error page	Show error page	Go to /error endpoint (created endpoint for invalid value redirection). Otherwise unable to generate error page.	Show error message and provide way to get back to form	<p>Sorry, there was an error loading the page.</p> <p>Back to Homepage</p>	PASS

6 Extensions

If this project were to be developed further, then there are many improvements that could be made. The first improvement would be to implement scaling according to the number size rather than the frequency of numbers. This would allow the application to scale out and in more gracefully, thereby keeping the application responsive.

The second improvement would be to remove the 4 instance limit (do not use the AWS educate account type). This would allow the application to serve more clients, even under load.

The final improvement would be to use better server instances. Currently, the application uses server instances that have 1 virtual CPU. Using a more powerful machine would decrease the calculation time for the server, and would make the entire program more responsive to larger requests.

Extensions on the application are also possible. The main extension that should be applied is to have batch processing functionality for all endpoints. Currently, the application only has batch processing to determine whether or not a number is prime (`batchPrime()`). Batch processing could also be applied to the `factorise()`, `nextPrime()`, and `orderPrime()` endpoints. This would allow the user to perform the same operations on multiple numbers at once.

7 User Guide

The user needs to first gain the IP address from the original developers (listed at the bottom of the cover page).

Once the IP address has been gained, the user may then proceed to hit the end point at port 3000.

After hitting the port, the user should see a simple form with multiple inputs under some headings. The user may then proceed to query the primality properties of their choice of numbers (as long as it is a valid input). Enter a number in any of the query boxes, and then press enter or click the button.

The application will then send the request to the server-side, which will process the request. This is invisible to the user.

After the request has been processed, the server should respond with either true/false or a numeric answer. This will be displayed on the screen (GUI)


Congratulations, you have completed the introductory tutorial to using this application!

8 References


No external references were made in this paper

9 Appendix

9.1 Appendix 1: Proposal Acceptance Email from Professor and Tutor



Jim Hogan <j.hogan@qut.edu.au>
17/10/2019 7:38 AM



To: Michael Esteban; Yu Gen Yeap; Kunal Chand

This one is fine.

From: Michael Esteban
Sent: Wednesday, 16 October 2019 3:21 PM
To: Yu Gen Yeap <yugen.yeap@connect.qut.edu.au>; Jim Hogan <j.hogan@qut.edu.au>; Kunal Chand <kunal.chand@connect.qut.edu.au>
Subject: Re: CAB432 Assignment 2 Proposal

Hi Yu Gen,

Azure Blob sounds good. Please start developing.

Best,
Michael

From: Yu Gen Yeap <yugen.yeap@connect.qut.edu.au>
Sent: 16 October 2019 15:11
To: Michael Esteban <m.esteban@qut.edu.au>; Jim Hogan <j.hogan@qut.edu.au>; Kunal Chand <kunal.chand@connect.qut.edu.au>
Subject: Re: CAB432 Assignment 2 Proposal

Hi Michael,

Thanks for pointing that out. If that is the case with the storage we will use Azure Blob storage, the equivalent of AWS s3 buckets in Azure and will follow a similar caching and storage approach done in the persistence practical.

We have sent an informal brief explanation to Jim about our project earlier this week and he said it was okay.

Please let us know if there are any other concerns with our project as we would like to get started as soon as possible.

Cheers
Yu Gen

From: Michael Esteban <m.esteban@qut.edu.au>
Sent: Monday, 14 October 2019 3:43 PM
To: Yu Gen Yeap <yugen.yeap@connect.qut.edu.au>
Cc: Jim Hogan <j.hogan@qut.edu.au>; Kunal Chand <kunal.chand@connect.qut.edu.au>
Subject: Re: CAB432 Assignment 2 Proposal

Hi Yu Gen & Kunal,

Thanks for your email. As you are proposing your own project, Jim will need to approve it.

From my perspective: the architecture seems OK. My main concern is that the project only seems to be using one form of persistence (Redis). The assignment requires two. Are you planning on using S3 or a database as well?

9.2 Appendix 2: Deployment Guide on AWS

An Ubuntu virtual machine instance was created with security groups allowing connections for SSH (22) and a custom TCP connection for node (3000). The instance was connected via SSH. The working application was cloned from GitHub into the new instance and the following dependencies required for the application were installed:

Node and npm were installed by entering the following command in the terminal

```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash - && sudo apt-get install -y nodejs
```

The following npm packages were installed by running the "npm install" command which retrieved the appropriate packages listed in the package.json file.

http-errors

aws-sdk

redis

response-time (optional for checking how long the server takes to respond)

The Redis server was installed onto the instance using the following command

```
sudo apt install redis-server
```

Pm2 was installed using the following command

```
sudo npm install -g pm2
```

Finally the credentials had to be placed in the .aws/credentials directory to allow us access to the S3 buckets for 3 hours. As we are using an AWS Educate account, this is one of the major limitations of this application

The following commands were used to set up the pm2 server to keep it automatically running after the machine has restarted

```
pm2 startup
```

```
sudo env PATH=$PATH:/usr/bin:/usr/lib/node_modules/pm2/bin/pm2 startup system -u ubuntu -hp /home/ubuntu
```

```
pm2 start npm -- start
```

```
pm2 save
```

This allowed the server to automatically start-up once the instance is booted up. Removing the need to manually start the server with the npm start command.

An image of the Ubuntu instance was created along with two security groups for the load balancer and auto-scaling for multiple instances. The load balancer allowed connections from HTTP 80 and the auto scaling group allowed connections from TCP 3000 and SSH (22).