Service-Centric & Cloud Computing Report

Maks Miketa 2020 (N0945100)

2024

Contents

[A & B Self Evaluation 2](#_Toc156380867)

[Summary 2](#_Toc156380868)

[Logging System 2](#_Toc156380869)

[View Your Trips 2](#_Toc156380870)

[Add New Trip 3](#_Toc156380871)

[View All Trips 3](#_Toc156380872)

[C. System analysis and design for cloud migration with QoS considerations 4](#_Toc156380873)

[Introduction 4](#_Toc156380874)

[TravelBuddyFinder Cloud System 4](#_Toc156380875)

[QoS Considerations 5](#_Toc156380876)

[Performing benchmark tests using Apache JMeter 5](#_Toc156380877)

[D. Analysis of Big Data scenarios and ways of mitigating them through cloud computing 8](#_Toc156380878)

[Introduction 8](#_Toc156380879)

[Volume 8](#_Toc156380880)

[Velocity 8](#_Toc156380881)

[Variety 8](#_Toc156380882)

[Veracity 9](#_Toc156380883)

[Conclusion 9](#_Toc156380884)

[References 9](#_Toc156380885)

# A & B Self Evaluation

## Summary

Both the frontend (client) and the backend of the RESTful service (orchestrator) of the TravelBuddyFinder application were created using Java JDK 15. The orchestrator was deployed on Apache Tomcat 9 server that was hosted on Microsoft Azure Cloud. MySQL was database was also hosted on Azure Cloud and used to store the orchestrator’s data. All the requests and responses sent within the project were in JSON format and parsed on both the client and orchestrator. All code is error handled to ensure the application does not fail and is functional at all times.

### Logging System

A client must login to access the application’s features. A POST request is sent to the login endpoint to validate the user to access the features. The orchestrater receives the request and checks if the email and password match within the database and sends a response back to the client.

The user can register with a valid email account that does not already exist and a password that is hashed before getting sent as a POST request to registration endpoint. The password is also hashed in the login request to improve security. The orchestrator receives the request and checks the provided email within the database to see if it already exists. If the email does not exist, the client sends a POST request to random.org API to provide a random unique userID which is stored to the database alongside the email and password that was already hashed by the client and sends back the response. The user is greeted with the menu screen after logging in and their user ID is displayed. The user can view their own trips, add trips, and view all the other user’s trips.

A screenshot of a computer code

Description automatically generated

Figure 1 - random.org API

### View Your Trips

The user is shown their trips within a table that refreshes every 10 seconds from a GET request. The user can click on a trip and view their details, upon clicking on their trip, the current weather details are provided for the trip from a GET request when clicking on the selected trip which is sent to the weather endpoint. The orchestrator sends a GET request to the weather API to provide current weather details.

The user can press the delete trip button that sends a DELETE request to delete interest endpoint. The orchestrator checks the database and deletes the trip and sends back the response.

The View Interests button sends a GET request to view interests’ endpoint. The orchestrator checks the database provide users that expressed interest in the selected trip.

### Add New Trip

The location field must not be empty, include special characters or provide an invalid location else, a pop up will appear explaining the issue for the user to correct. The first character is capitalised in case the user did not capitalise the first character. Past dates within the start and end date calendar cannot be chosen to prevent adding trips in the past. Upon sending a POST request to the endpoint, if successful, the user is provided with a success response with the trip details alongside a unique generated Trip ID and the current weather details for the location. The orchestrator receives the request and sends a GET request with the location details to the google geocoding API and receives the longitude and latitude coordinates. These coordinates are parsed to a GET request sent to weatherapi.com API to provide the weather details. A POST request is sent to random.org to provide a random unique trip ID. All trip details are stored in the database and the response is sent to the client with the trip details and the weather information.

A screenshot of a computer program

Description automatically generated

Figure 2 - wetherapi.com and google geocoding API

### View All Trips

A GET request is sent to endpoint and provides a list of all users’ trips including all the details of the trip within a table that refreshes every 10 seconds. Only present trips are shown. The user can select a trip and the details are shown as well as the current weather details the same as when viewing the user’s own trips. The user can press the express interest button that sends a POST request to interest endpoint. The orchestrator checks in the database if the interest is already expressed or not and sends back the response.

# C. System analysis and design for cloud migration with QoS considerations

## Introduction

The TravelBuddyFinder application was designed to be deployed to the cloud however, it was first developed and tested on a local host to ensure it was functional before planning to migrate it to the cloud. Migration to the cloud would provide numerous benefits that would improve many aspects of the application including the quality of service (QoS) and address many challenges without this transition.

## TravelBuddyFinder Cloud System

A diagram of a software company

Description automatically generated

Figure 3 – TravelBuddyFinder Cloud System

The system was coded in Java and contains multiple endpoints of which some use three external APIs and the system’s data is stored on MySQL database. The Lift and Shift method (IBM, 2024), was used to deploy the application on to Azure cloud after the service met the functional requirements on the local host. The Lift and Shift method migrates the entire application architecture with minimal alterations to the application’s code thus, enabling a fast and cost-effective migration to the cloud compared to other methods. The database was also deployed to the cloud (Azure MySQL) as it was originally running on the local host therefore, the only modification to the system’s code was the connection to the database.

Transitioning the system to the cloud can be costly, particularly when performing major changes within an organisation especially if the current infrastructure is on-premises and involves implementing robust QoS measures. Migration to the cloud was not an issue for TravelBuddyFinder as it does not belong to an organisation and has no stakeholders to affect this decision. Transitioning from an on-premise infrastructure to cloud-based infrastructure provides benefits such as scalability, flexibility, fault-tolerance, accessibility and so on (Sether, 2016). The QoS considerations focused for TravelBuddyFinder will be scalability and fault-tolerance within this report.

## QoS Considerations

Scalability and fault tolerance play an important role in the TravelBuddyFinder application, as the service may have to handle many users at a given time. An on-premise setup requires powerful hardware to handle a large flow of traffic. Scaling up can become expensive from improving hardware to address the rise in user traffic and capacity. If the hardware is not improved from the increase in traffic, the service may suffer from reduced performance and service interruptions resulting in compromised fault tolerance and QoS. Additionally, variations in user traffic may lead to increased and unnecessary energy consumption on physical hardware as it may be overpowered and unoptimized for when the service has a small amount of traffic.

Services such as Microsoft Azure Scale Sets (ASS) can be used to address these challenges and increase QoS while maintaining appropriate costs. ASS provides load balancing with instances of virtual machines that dynamically scale in response to load changes thus, greatly improving scalability. ASS are cost-effective, ensuring fault tolerance without service interruptions during maintenance or sudden traffic spikes. In contrast to on-premise setups, instances of VMs in ASS seamlessly detect and distribute workload across availability zones which avoids slowdowns or interruptions to the service in case of failures or required maintenance.

## Performing benchmark tests using Apache JMeter

Apache JMeter is an open-source tool designed for performance and load testing on web applications. The TravelBuddyFinder app was tested to see how it performs after it was migrated to the cloud. The benchmark performed on the application within this report was a HTTP GET request where the server gets all trips from the database and sends back the response.

A screenshot of a computer

Description automatically generated

Figure 4 - Graph results after 100 requests

The first test simulated 100 users sending the request simultaneously. From the graph results in figure 2, the latency increased the more requests were sent. The deviation and average increased after every request sent too.

A graph on a computer screen

Description automatically generated

Figure 5 - Graph results after 1000 requests

The same trend continues within figure 3, this time with a greater deviation.

A screenshot of a computer

Description automatically generated

Figure 6 - Graph results after 10000 requests

Lastly, 10000 requests were sent in figure 4 to test the services fault tolerance. The trend remained the same as from the previous graphs until the service failed at 7000-8000 requests due to hardware failure. There was also a greater increase in latency resulting in users waiting longer for responses.

The benchmark results signify the importance of testing the application for fault tolerance as its capabilities and weaknesses are identified. The application manages a large amount of data, and it is heavily dependent on accessing it to the database, so any failures would make the service inaccessible. To make the application more scalable and fault tolerant, an ASS would be set up with a dynamic load balancer that would distribute the VM instances in accordance with the load capacity. The cost would also be reduced compared to on premise servers as the balancer would descale the resources to only use the necessary hardware required while maintaining optimal QoS and performance.

Another QoS enhancement would be to implement a message queuing system such as RabbitMQ which decouples communication between the web server and the database through asynchronous messages. RabbitMQ handles incoming requests independently to ensure continuous operation even within an event of failure on the service which improves fault tolerance.

# D. Analysis of Big Data scenarios and ways of mitigating them through cloud computing

## Introduction

Big data is a term referred to the challenges posed by large, diverse, and complex datasets that traditional processing methods struggle to manage, analyse, and store. Big data is generated from various sources and stored in databases for analysis. The analysed data is useful to companies and organisations to help them gain richer and deeper understanding and an advantage over their competition or within their sector. To achieve this, the data must be analysed and executed as accurately as possible (Sagiroglu & Sinanc, 2013).

Big Data is characterised of 8 V’s but, 4 V’s will be discussed regarding the TravelBuddyFinder application.

## Volume

Storing the application’s datasets such as user information, messages, trip details, interests etc. on traditional physical storage drives can become problematic from the large amount of stored data which can lead to possible hardware failures and traditional data protection methods incapable to handle such large volume of data. The traditional storage system also suffers with scalability issues as it is difficult and expensive to constantly expand the storage for the growing datasets.

Cloud storage services such as Amazon S3 provide unlimited storage and robust fault tolerance that address these issues. (Yang, et al., 2017). Hadoop, a distributed computing framework commonly used for large-scale data processing, can be integrated with Amazon S3 to leverage its scalable storage capabilities. This integration allows organisations to benefit from the durability and scalability of Amazon S3 while using Hadoop's distributed processing capabilities to analyse and process large volumes of data efficiently while increasing performance and avoiding network congestions (Sandhu, 2022). A drawback to this approach is the cost to transfer a substantial volume of existing data to the cloud. To avoid this issue, the application would have to be set up to store the data on a cloud storage service such as the mentioned service before deployment, to avoid migrating the data to the cloud later at a greater cost.

## Velocity

Velocity, refers to the speed at which data is generated, processed, and analysed. In the context of stream processing, challenges such as data latency, system latency, and resource latency can hinder real-time data processing. The application can encounter these challenges if many users send requests simultaneously. To enhance efficiency and reduce latency, the application would implement an optimised stream processing model leveraging Apache Beam with Google Cloud Dataflow. This involves a systematic flow design, integrating a pipeline and watermark strategy. The pipeline enables parallel processing of incoming data streams, maintaining a continuous flow, while watermarks assess input completeness that mitigate delays and improve overall speedup (Kumar, et al., 2021). In situations where concurrent user access is high, this optimised approach ensures that the application delivers prompt and accurate results thus, safeguarding the applications conditions of high data velocity.

## Variety

The application currently includes and is expected to have a wider variety of data types with the implementation of additional features in the future. As an example, when users engage with the "view trips" feature, the application needs to manage diverse data formats. This includes structured data like trip details and itineraries, and unstructured data in the form of images and videos showing the trip's location which will expand more in the future. Traditional storage methods can encounter difficulties in adapting to the increased variety of data types resulting in degraded data processing and issues with scalability. To overcome these limitations, the application could benefit from integrating Hadoop Distributed File System (HDFS) which is an integral component of the Hadoop framework. HDFS is purpose-built for handling large-scale data across distributed nodes and is designed to allow efficient storage and retrieval of both structured and unstructured data thus, providing a scalable and adaptable solution for expanded data formats while ensuring optimal data processing performance (Kaur & Goraya, 2019).

## Veracity

Data must be accurate, reliable, and consistent which is a challenge when producing and processing an immense volume of data. As the volume of the data increases, the quality of the data may degrade which if not solved can lead to many consequences in the long term. To address this issue, cloud computing services such as Azure Data Factory can be used to monitor and perform quality checks on the data. The service offers a diverse range of features to filter relevant and required data that would provide a huge benefit to the application (Eswararaj, 2023).

## Conclusion

After analysing the impact of big data on the TravelBuddyFinder application, it is evident that addressing the outlined scenarios is essential to avoid these challenges. The effective approach to use cloud services not only mitigates these challenges, but also provides numerous benefits to the application. It is crucial for the application to be scalable and adapt to the rapid growth of stored data to avoid issues with performance and high maintenance cost.

Not only storing the data is an issue as managing the stored data is equally as important to analyse it and use it for continuous improvement to further optimise the application and enhance the user experience.

The outlined cloud services within this report including many other services available provide possible solutions to mitigate the identified issues and make sure the application operates flawlessly, is optimised and future proof while keeping up with the latest changes in technology and avoiding expensive costs.

# References

Eswararaj, D., 2023. *Developing a Data Quality Framework on Azure Cloud: Ensuring Accuracy, Completeness, and Consistency,* s.l.: International Journal of Computer Trends and Technology.

IBM, 2024. [Online]   
Available at: https://www.ibm.com/topics/lift-and-shift  
[Accessed 10 01 2024].

Kaur, H. & Goraya, M. S., 2019. *IJERT-Role of Big Data in Cloud Computing: A Review,* s.l.: International Journal of Engineering Research and Technology (IJERT).

Kumar, V., Mishra, V. & Sharma, D., 2021. *Improved Optimization and Speed up in Big Stream Data Processing,* s.l.: Turkish Journal of Computer and Mathematics Education (TURCOMAT).

Sagiroglu, S. & Sinanc, D., 2013. *Big data: A review,* San Diego: 2013 International Conference on Collaboration Technologies and Systems (CTS).

Sandhu, A. K., 2022. *Big data with cloud computing: Discussions and challenges,* s.l.: TUP.

Sether, A., 2016. *Cloud Computing Benefits,* s.l.: SSRN 2781593.

Yang, C. et al., 2017. *Big Data and cloud computing: innovation opportunities and challenges,* s.l.: International Journal of Digital Earth.