|  |  |
| --- | --- |
|  | Grade (by the assessor): |
| Student ID: N1020843 |
| Word count (Sections II.3 and II.4 only): (750+839) = 1589 |

# Self-evaluation of the implementation for Sections II.1 & II.2 (Must not exceed 1 page)

**II.1 Core Web Service**

Orchestrator and REST Services:

* The orchestrator was implemented as a RESTful web service using Java and deployed on a Tomcat server. It coordinates the functionality of different services (rooms, weather, distance, and hidden which is the geocoding service).
* Room REST services were created to handle user operations such as viewing rooms, applying for rooms, and managing applications. For simplicity, data is stored in JSON files.

Client Integration:

* A text-based client was implemented to interact with the REST services. It allows users to: View all rooms, apply for rooms and manage applications, and check proximity and weather information based on postcodes and rooms. An admin profile option was added to be able to accept pending room offers

Data Handling:

* Data related to rooms, users, and applications is handled through classes like Room, User, and RoomApplication. Serialization and deserialization ensure data integrity across the system.

**II.2 Distance Calculation, Weather, and Geocoding Services**

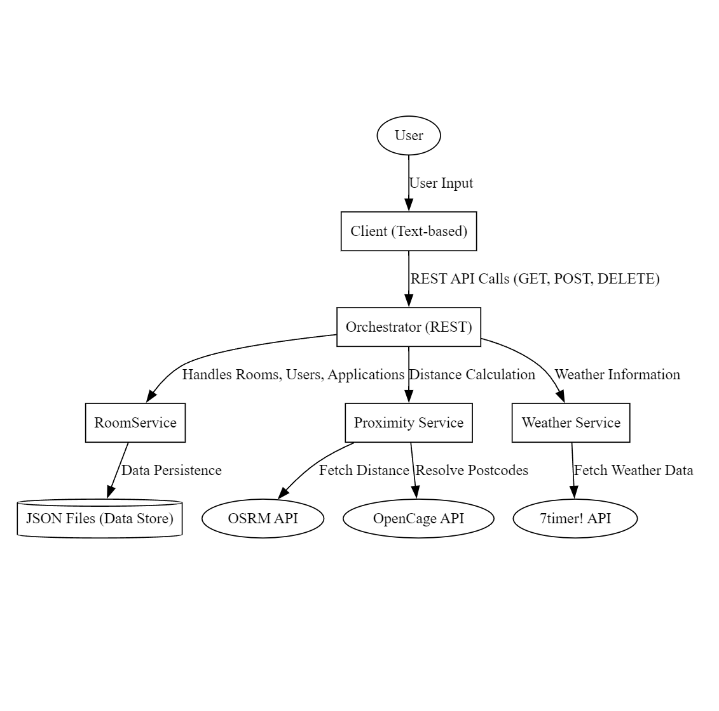
Distance Calculation Service:

* Uses the OSRM API to calculate distances between user and room locations.
* Geocoding using the OpenCage API converts postcodes into latitude/longitude for integration with OSRM.

Weather Service:

* Uses the 7timer! API to fetch weather information for room locations or user-specified postcodes.
* Responses are simplified to display key metrics (temperature and cloud cover).

Geocoding Service:

* Resolves postcodes into latitude and longitude using the OpenCage API.
* Ensures both the distance calculation and weather services work with standardised coordinates.

# System analysis and design for cloud migration with QoS considerations

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Write your discussion here (max 800 words).**  Deploying the GlobalDorm application on a distributed cloud infrastructure involves restructuring the system to leverage cloud-native services for scalability, availability, and performance. Microsoft Azure serves as an example platform for this discussion.  The frontend can be hosted using Azure App Service, which simplifies deployment and provides high availability. This allows the client application to scale dynamically to accommodate increased user traffic. The backend services, including the orchestrator, RoomService, WeatherService, and ProximityService, would be deployed on Azure Kubernetes Service (AKS). AKS supports container orchestration, fault tolerance, and horizontal pod autoscaling, ensuring that backend resources adjust to varying workloads. For data storage, application data such as rooms, users, and applications can be migrated to Azure SQL Database, which offers relational storage and the ability to scale using Hyperscale. This ensures seamless management of growing datasets.  To manage API integrations with third-party services, such as weather and geocoding APIs, Azure API Management can be utilized. This adds caching, rate limiting, and monitoring capabilities, reducing the dependency on external APIs while improving reliability. This setup ensures the application can handle a geographically distributed user base and manage traffic spikes effectively.  **Challenges and Solutions for Cloud Migration**  **Latency in API Calls** Integrating external APIs for weather and proximity services can introduce latency, especially during high user activity. Caching frequently requested data using Azure Cache for Redis can mitigate this issue. By storing recently fetched weather and proximity results, the system reduces the need for repetitive external API calls, improving response times.  **Autoscaling** As user traffic grows, maintaining consistent performance requires dynamic resource allocation. Configuring AKS to autoscale based on CPU and memory usage ensures that additional container instances are deployed as demand increases. Horizontal pod autoscaling (HPA) can be implemented to automatically scale pods up or down.  **Database Scalability** Handling growing datasets and ensuring fast query responses require scalable database solutions. Azure SQL Database with Hyperscale provides dynamic storage capabilities for large datasets. Enabling geo-replication ensures high availability and fault tolerance by replicating data across multiple regions.  **Fault Tolerance** To achieve fault tolerance, deploying backend services across multiple Azure regions is critical. Azure Traffic Manager can route requests to the nearest available region, ensuring uninterrupted service even during regional outages. This approach enhances reliability and minimizes downtime.  **Quality of Service (QoS) Analysis**  **QoS Metrics**:   * **Latency**: Measures the response time for REST services. The target is to keep latency below 200 milliseconds to ensure seamless user interactions. * **Throughput**: Refers to the number of requests handled per second. The goal is to optimize the backend to support a high number of concurrent users without performance degradation. * **Availability**: Tracks the system uptime, with a target of 99.9% availability to ensure minimal disruptions.   Two areas for improving QoS include scalability and payload optimization. Scalability can be enhanced by configuring autoscaling policies for both backend services and the database. This ensures that resources are dynamically adjusted to meet demand. Payload optimization focuses on reducing the size of data transferred between the client and backend. Enabling Gzip compression for API responses reduces network overhead, while implementing pagination for large datasets, such as room listings, ensures manageable response sizes.  **Testing Setup** Quality of Service testing was conducted using Apache JMeter to evaluate the system's scalability under different conditions. The tests simulated multiple users accessing REST endpoints, varying payload sizes. Metrics such as response time, throughput, and error rate were analysed to identify bottlenecks.  **Testing Scenarios and Results**   1. **Concurrent Users** Simulations included 100, 1000, 10000 concurrent users accessing the /rooms/weather endpoint. At 100 users, the system maintained a stable response time of 350 milliseconds. At 1000 users the system remained stable. At 10000 users, response times increased exceeding 500 milliseconds, with a noticeable increase in deviation. Autoscaling policies were proposed to address this limitation, ensuring backend resources scale dynamically during traffic spikes. 2. **Payload Size** The /rooms endpoint was tested with payloads containing 100, 500, and 1000 room entries. For smaller payloads (100 entries), response times remained below 200 milliseconds. Larger payloads (1000 entries) caused response times to increase to 700 milliseconds, highlighting the need for optimization. Implementing pagination and enabling Gzip compression reduced the payload size, leading to improved performance.   Recommendations for QoS Improvements  To enhance QoS, caching frequently accessed data and optimizing payload sizes should be prioritized. Azure Cache for Redis can significantly reduce latency for repetitive API calls. Additionally, implementing pagination and compression for large payloads ensures faster data transfer and processing. Future testing should focus on integrating these optimizations and evaluating their impact on performance under real-world conditions. | | | | | | |
| **Criteria/ Grade** | **First 1st** **\*Exceptional** | High | Mid| Low | **Upper Second 2:1** High| Mid| Low | **Lower Second 2:2** High |Mid |Low | **Third 3rd** High |Mid |Low | **Fail** Marg |Mid |Low | **ZERO** |
| **Report II.3**  **10%** | Well-elaborated  approach for cloud deployment with advanced technical solutions and understanding of QoS.  **\*Comprehensive** Physical platform testing and analysis of results when increasing service demand both at the connection and payload levels and accurate identification of hardware constraints. Realistic solutions are proposed and analysed.  **\*The discussion is fully supported by physical platform testing. Draws insightful conclusions.** | Good analysis of a system design for cloud deployment with insight on SOA QoS challenges linked to the implemented solution and backed by some physical tests although analysis may lack some depth or focusses on one level. Proposal lacks methods to reduce the incurred overheads. Evidence of appropriate selection and critical evaluation of relevant research. | Adequate explanation of cloud deployment with satisfactory solutions and limited understanding of QoS. Limited testing. Some hardware constraints are identified.  Most of the discussion is focussed on the application but the text relies on generic content with few traces of critical thinking. | Basic approach with minimal understanding of distributed deployment, basic solutions, and incomplete or no physical testing. Limited or no hardware constraints identification.  Although the discussion is focussed to some extent on the application it lacks depth and most of the text is generic. | Some descriptive information may be present about the system design without references to a cloud-based deployment, and about the QoS in SOA. Presented information is widely found on the internet and not focussed on the application. | Work of no merit or work not submitted. |
| **Grade:**  **Feedback** |  | | | | | |

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Write your discussion here (max 900 words)**  Today’s social applications generate massive data volumes at variable rates, requiring robust architectures to process and store data efficiently. In the context of the 8 V’s of Big Data, this section examines three key dimensions—Volume, Velocity, and Variety—as they relate to the GlobalDorm application. These are analysed alongside cloud-enabled solutions to address the challenges posed by these dimensions.  Volume refers to the scale of data generated and stored by an application. For GlobalDorm, data volume arises from user profiles, room listings, application records, and dynamic API responses.  **Estimated Data Volume from sources**:   * **User Profiles**: While specific data on average user profile sizes is limited, it's reasonable to estimate that each user profile, including personal details and activity logs, requires approximately 500 KB of storage. With 10,000 active users, this would result in approximately 5 GB of storage. * **Room Listings**: Detailed data on the average size of room listings is not readily available. However, considering typical information such as room descriptions, photos, and amenities, it's estimated that each room listing requires around 5 KB of storage. For 1,000 rooms, this would amount to approximately 5 MB of storage. * **Application History**: Information on the average size of application records is scarce. Assuming each application record, including metadata like timestamps and user details, averages around 10 KB, processing 50,000 applications annually would generate approximately 500 MB of data. * **Dynamic Data (API Responses)**: While specific data on the size of API responses is limited, it's reasonable to estimate that external API responses, such as weather and proximity data, average 2 KB per request. With 10,000 requests per day, this would result in approximately 7.3 GB of data annually.   These data volumes have a significant impact on storage and query performance as the application scales.  **Cloud Solution t**o manage these data volumes effectively, consider the following Azure services:   * **Azure SQL Database**: A fully managed relational database service that offers scalability and high availability, suitable for structured data like user profiles and application histories. * **Azure Blob Storage**: An object storage solution optimised for storing large amounts of unstructured data, ideal for handling dynamic API responses and room listing data.   In Section II.3, the proposed use of Azure SQL Database and Blob Storage aligns with the need to handle varying data volumes efficiently, ensuring scalability and performance as the application grows.  **Velocity** measures the speed at which data is generated, processed, and analysed. GlobalDorm experiences high-velocity challenges due to frequent API calls and real-time user interactions.  **Estimated Request Rate**:   * **API Calls**: E-commerce platforms often implement API rate limits to manage the number of requests within a specific timeframe. For instance, Shopify's REST Admin API allows for a burst of up to 40 requests per app per store, with a recovery rate of 2 requests per second. This means that during peak times, an application can handle a significant number of requests, but sustained high-frequency requests are regulated to prevent overloading the system. * **User Activity**: As of 2024, the average daily social media usage worldwide is approximately 143 minutes. Assuming that during this time, a user performs various interactions such as likes, comments, shares, and posts, it's reasonable to estimate that an active user might generate several dozen interactions per day. For GlobalDorm, with 10,000 daily active users, this could translate to hundreds of thousands of backend interactions daily (Dixon, 2024).   These high request rates necessitate a backend capable of processing large volumes of concurrent traffic while maintaining low latency.  **Cloud Solution t**o address velocity-related challenges:   * **Azure Event Grid**: Enables event-driven architecture for asynchronous processing of high-velocity user interactions, reducing backend bottlenecks. * **Azure Functions**: Provides serverless, on-demand processing for lightweight tasks such as API requests, ensuring low-latency responses.   In Section II.3, the proposed autoscaling for Azure Kubernetes Service (AKS) is complemented by Event Grid and Functions, ensuring efficient handling of event-based traffic spikes.  **Variety** refers to the diversity of data types handled by the application. GlobalDorm processes structured, semi-structured, and unstructured data.  **Data Types and Sources**   * **Structured Data**: User profiles, room details, and application records stored in SQL. * **Semi-structured Data**: API logs and metadata in JSON format. * **Unstructured Data**: Weather and proximity API responses.   Managing diverse data formats requires flexible systems capable of transforming and integrating data from multiple sources which impacts the infrastructure.  **Cloud Solution** to address variety-related challenges:   * **Azure Data Factory**: Automates ETL workflows, converting external API responses into structured datasets for analysis. * **Azure Blob Storage**: Stores unstructured data, such as archived API responses, in a cost-effective manner.   Section II.3 highlighted Azure API Management for handling external integrations. Data Factory and Blob Storage extend this capability by processing and storing diverse datasets.  **Cloud Deployment Model**  The **Platform-as-a-Service (PaaS)** model is the most suitable choice for GlobalDorm. PaaS provides scalability, ease of deployment, and cost efficiency by abstracting infrastructure management. Key advantages include:   1. Seamless Scalability: Autoscaling in Azure App Service and AKS adjusts resources dynamically to traffic spikes. 2. Simplified Deployment: Managed services like Azure SQL Database and Cosmos DB reduces operational overhead. 3. Cost Efficiency: PaaS optimizes resource usage with pay-as-you-go pricing.   References:   * (No date) *Rest admin api rate limits*. Available at: https://shopify.dev/docs/api/admin-rest/usage/rate-limits (Accessed: 10 January 2025). * Dixon, S.J. (2024) Global daily social media usage 2024, Statista. Available at: https://www.statista.com/statistics/433871/daily-social-media-usage-worldwide/?utm\_source=chatgpt.com (Accessed: 10 January 2025). | | | | | | |
| **Criteria/ Grade** | **First 1st** **\*Exceptional**| High| Mid| Low | **Upper Second 2:1** High| Mid| Low | **Lower Second 2:2** High |Mid |Low | **Third 3rd** High |Mid |Low | **Fail** Marg |Mid |Low | **ZERO** |
| **Report**  **II.4 20%** | Demonstrates excellent understanding of the selected 3 V’s with specific examples and estimates relevant to the application. Provides an insightful discussion of infrastructure and software implications, including a justification for the chosen cloud deployment model. Shows how cloud solutions effectively manage Big Data challenges.  **\*Evidenced by references to recent research to support the arguments put forward for proposing the current solutions. The analysis goes beyond what has been taught.** | Demonstrates good understanding of big data issues and the selected 3 V’s. Includes realistic numbers and estimates relevant to the application. Cloud-enabled solutions for the selected V’s are critically analysed but may be lacking in depth. The selected cloud computing deployment models is justified and other models are referenced but analysis is incomplete. Uses some academic sources. | Provides a basic description of the 3 V’s. Discussion of infrastructure and software implications is general, with a somewhat justified choice of cloud computing model but lacking details. Cloud-enabled solutions for the selected V’s are included but may not be fully adequate or critically analysed. Uses some academic sources but mostly relies on informal sources. | The 3 V’s have been identified and an appropriate cloud solution for at least 1 V has been presented and discussed. Minimal discussion and basic cloud solutions for the other V’s, if any. Little justification of the cloud deployment model. Purely descriptive. | Only 1 or 2 V’s are identified. The cloud deployment model may not have been identified and cloud-enabled solutions may have been mentioned but not discussed. Presented information is widely found on the Internet and not focussed on the application. | Work of no merit or work not submitted. |
| **Grade:**  **Feedback** |  | | | | | |