



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

School of Applied Computing

ACCA7002

Report on Database Implementation: ArtVista

By

2227183

Chisom Vanessa Nwankwo

MSc. Data Science and Analytics

Table of Contents

Abstract	iii
1.1. Overview of Business Goals and The Need for a Database	1
1.2. Relational Databases vs. NoSQL Databases.....	1
1.2.1. Document Stores (e.g., MongoDB).....	1
1.2.2. Graph Databases (e.g., Neo4j)	2
1.2.3. Key-Value Stores (e.g., Redis)	2
1.2.4. Wide Column Stores (e.g., Cassandra).....	3
1.3. Using NoSQL over Relational databases.....	3
1.4. Advantages of using NoSQL for ArtVista’s database	4
1.5. Database Recommendation.....	5
1.6. Deployment Strategies for the Hybrid Approach	7
1.7. Alignment of Solution with ArtVista’s Goals	9
1.8. Summary, Key Findings and Recommendations.....	10
References	12

Abstract

The report comprehensively analyses database solutions for ArtVista, a digital art marketplace. It compares Relational and NoSQL databases, highlighting their scalability, data models, and performance. Key database types like Document Stores, Graph Databases, Key-Value, and Wide Column Stores are evaluated for their use cases, data models, and scalability. The report recommends a hybrid approach for ArtVista, combining the flexibility and scalability of MongoDB for storing diverse, evolving artwork metadata and user-generated content with the solid data and relational integrity of PostgreSQL for transactional data. It also discusses deployment strategies, including cloud-based, on-premises, etc. Finally, it aligns the proposed solution with ArtVista's goals, emphasising the need for diversity in data handling, scalability, data integrity, complex queries, rapid development, user experience, cost-effectiveness, and security compliance. Key recommendations include implementing robust security measures, regular backups, continuous monitoring, and testing.

1.1. Overview of Business Goals and The Need for a Database

ArtVista is a digital art marketplace based in Swansea seeking expert consultation to explore and identify the most suitable database solution that aligns with its operational requirements, growth ambitions, and commitment to providing an exceptional online art marketplace. The ideal solution should offer scalability, reliability, security, and an architecture that supports a dynamic and engaging user experience.

1.2. Relational Databases vs. NoSQL Databases

Some of the factors to be considered when comparing both types of databases are:

- a. **Scalability:** Relational databases are usually scalable vertically, meaning you can increase their capacity by enhancing the server's CPU, RAM, or SSD. NoSQL databases, on the other hand, can be horizontally scaled, which allows the addition of more servers to handle more traffic [1].
- b. **Data Model:** Relational databases use a structured query language (SQL) and have a predefined schema. They are best for complex queries and ACID (Atomicity, Consistency, Isolation, Durability) compliance. NoSQL databases have dynamic schemas for unstructured data and are categorised into different types, each with its unique strengths and for different use cases [2].
- c. **Performance:** NoSQL databases provide better performance than legacy relational database systems for higher workloads, especially common in Big Data and Cloud Computing applications [3].

1.2.1. Document Stores (e.g., MongoDB)

Document stores manage data in JSON-like documents and are schema-free [4], unlike the strict, table-based structure in relational databases. They also offer high flexibility and are suitable for applications with evolving data models.

Features of Document Stores

- a. **Use Case:** Ideal for scenarios requiring high performance, flexibility, and scalability with data represented in JSON format.
- b. **Data Model:** Stores data in JSON-like documents, making it intuitive for developers.
- c. **Performance:** Generally, offers high performance for write and delete operations.
- d. **Scalability:** Supports horizontal scaling and is schema-free.

- e. ACID Properties: Partially supports ACID properties.
- f. Suitability: Best for applications needing to store complex, evolving data structures [5].

1.2.2. Graph Databases (e.g., Neo4j)

Graph databases are designed to store interconnected data and handle complex relationships efficiently, unlike relational databases, where relationships are defined through foreign keys and can become complicated to manage. They may offer higher performance for applications that heavily rely on the exploration of relationships (e.g., social networks) [2].

Features of Graph Databases

- a. Use Case: Suitable for highly interconnected data and evolving data models.
- b. Data Model: Stores data in nodes and edges, representing entities and relationships, respectively.
- c. Performance: REST performance is similar to document stores, but embedded operations might be slower.
- d. Scalability: Generally supports horizontal scaling and is schema-free.
- e. ACID Properties: Varies among implementations but often provides strong consistency.
- f. Suitability: Ideal for applications that require complex relationship traversals, like social networks or recommendation engines [6].

1.2.3. Key-Value Stores (e.g., Redis)

Key-value stores are simple, fast, and suitable for scenarios requiring quick access to data via a key. They don't offer the complex querying capabilities of relational databases. They are ideal for sessions, user preferences, and caching [2], where the data access pattern is simple.

Features of Key-Value Stores

- a. Use Case: Efficient for scenarios where quick access to data via a key is needed.
- b. Data Model: A simple model where each item is stored as a key paired with its value.
- c. Performance: Fast write and delete operations.
- d. Scalability: Supports horizontal scaling and is schema-free.
- e. ACID Properties: Varies among implementations.
- f. Suitability: Best for caching, sessions, and scenarios where data is frequently read and written [5].

1.2.4. Wide Column Stores (e.g., Cassandra)

Wide column stores allow for storing large volumes of data with columns that can vary from row to row, unlike the fixed schema of relational tables. They are known for high scalability and performance, especially for analytical applications [3].

Features of Wide Column Stores

- a. Use Case: Suitable for handling large volumes of data and accommodating various data types.
- b. Data Model: Stores data in tables with rows and columns but allows for variability in columns across rows.
- c. Performance: Fast write operations but comparatively slower read operations.
- d. Scalability: Designed for horizontal scaling and high availability.
- e. ACID Properties: Partially supports ACID properties.
- f. Suitability: Ideal for analytical applications and any scenario requiring high write throughput and scalability [5].

1.3. Using NoSQL over Relational databases

Several scenarios where NoSQL databases are preferable over relational databases are as follows:

1. Handling Large Volumes of Unstructured or Semi-structured Data

NoSQL is preferable when dealing with massive amounts of unstructured or semi-structured data, such as social media content, sensor data, or logs. NoSQL databases like MongoDB and CouchDB [7] are designed to handle a wide variety of data types and structures without a predefined schema, making them more suitable for unstructured data than relational databases' rigid schema requirements.

2. IoT and Real-time Analytics

Applications requiring real-time processing and analytics of data from IoT devices or other real-time data sources also necessitate using NoSQL databases. NoSQL databases are often used in distributed, cloud, and IoT systems due to their ability to handle rapidly expanding datasets and their emphasis on availability, scalability, and fault tolerance [7].

3. Storing and Processing Time Series Data

In applications that need to store and manage time-related data, such as measurements from sensors in cars, smartwatches, or IoT devices, often characterised by high frequency and volume, using NoSQL databases, particularly time series databases like InfluxDB [8], can outperform relational and other database systems in write-throughput and resource utilisation for time series data.

3. Scalability and Cost-Effectiveness in Distributed Environments

Systems requiring scalability across multiple clusters in cloud and on-premises environments, especially when cost-effectiveness is a consideration, can use NoSQL databases like MongoDB and Cassandra, which offer better scalability options compared to traditional relational databases like PostgreSQL [9]. They are optimised for specific use cases and can be more cost-effective depending on the scenario.

5. Rapid Development and Iterations

Projects with agile development methodologies that require frequent changes to the data model or quick prototyping are better with NoSQL. The flexible schema of NoSQL databases allows developers to iterate quickly without the need to perform costly database migrations or re-designs that you would typically encounter with the rigid structure of relational databases.

6. Complex Relationships and Data Agility

Applications that require the ability to navigate and understand complex relationships between data entities, like social networks, recommendation engines, or fraud detection systems, can leverage Graph databases like Neo4j or Amazon Neptune, which are optimised for storing and querying complex networks of relationships using nodes, edges, and properties.

1.4. Advantages of using NoSQL for ArtVista's database

1. Handling Diverse Digital Art Metadata:

Since ArtVista deals with a wide variety of digital artworks, Document-oriented NoSQL databases (e.g., MongoDB) can efficiently store this diverse and complex metadata. They allow for a flexible schema that can accommodate the different attributes of each artwork without the need for a rigid structure.

2. Scalability for Growing Inventory and User Base:

As ArtVista expands its online presence across the UK and worldwide, it expects a significant increase in the number of artworks, artists, and users on the platform. NoSQL databases can handle large volumes of data and high user loads with ease, growing horizontally by adding more servers.

3. Personalised User Experience:

To engage users effectively, ArtVista might want to offer personalised recommendations and maintain user preferences. Therefore, NoSQL databases, particularly graph databases, can efficiently manage complex user relationships and preferences to deliver personalised content. They can also handle the semi-structured data generated by user interactions.

4. Rapid Development and Iteration:

ArtVista must continuously evolve and update its platform to introduce new features and improve user experience in the competitive digital marketplace. NoSQL databases provide a flexible schema that allows quick iterations and modifications without extensive database re-design or downtime, facilitating agile development and rapid deployment of new features.

5. Global Distribution:

Also, as ArtVista aims to expand worldwide, it needs to ensure that users across the globe can access the platform quickly and reliably. This is another scenario where NoSQL databases are a good choice, as they offer built-in support for data distribution across multiple regions. ArtVista can store data closer to users' location, reduce latency, and improve user experience globally.

1.5. Database Recommendation

1. Data Organization:

- a. Relational (PostgreSQL): Data is structured in tables with predefined schemas. It is beneficial for ArtVista if the data relationships are complex and need to be strictly maintained, such as the relationships between artists, artworks, users, and orders.
- b. NoSQL (MongoDB): Data is stored in a more flexible format, typically as JSON-like documents. This is useful if ArtVista requires a flexible schema to accommodate various artwork metadata or rapidly evolving data structures.

2. Data Integrity:

- a. Relational (PostgreSQL): Offers strong data integrity features like ACID compliance, foreign keys, and transaction management. This ensures that all transactions in ArtVista are processed reliably and the data remains consistent.
- b. NoSQL (MongoDB): While it provides some level of transaction support and document validation, it's not as inherently strict as PostgreSQL. This might be suitable if ArtVista can tolerate some flexibility in data consistency for other benefits.

3. Data Volume:

- a. Relational (PostgreSQL): Can handle large volumes of data but might require more maintenance and scaling strategies (like sharding or replication) as data grows.
- b. NoSQL (MongoDB): It is better suited for handling huge volumes of data and unstructured data. Its horizontal scaling is more straightforward, making it a potentially better choice for ArtVista if it expects to handle a massive volume of artworks and user data.

4. Query Complexity:

- a. Relational (PostgreSQL): SQL queries can be very powerful and complex, allowing for intricate queries and reports, which might be necessary for ArtVista's administrative and analytical needs.
- b. NoSQL (MongoDB): Querying is typically more straightforward but might not support the same depth of query complexity without additional effort or processing.

5. Scalability:

- a. Relational (PostgreSQL): This can scale but is typically more complex and costly as it often involves vertical scaling and more sophisticated replication strategies.
- b. NoSQL (MongoDB): It is designed with horizontal scalability, making it easier to scale out across multiple servers and handle large amounts of read/write operations, which could benefit a growing platform like ArtVista.

6. Ease of Development:

- a. Relational (PostgreSQL): SQL is a well-known query language, and PostgreSQL has a strong ecosystem and community. However, schema changes can be more rigid and require more maintenance.
- b. NoSQL (MongoDB): Offers more flexibility for developers with its dynamic schema. This can lead to faster development and iteration, especially in the early stages of ArtVista or when adding new features.

In summary, for ArtVista, a hybrid approach might be the most suitable given the varied needs.

Use **MongoDB** for storing artwork and user data where flexibility, scalability, and high-volume data handling are crucial. Its schema-less nature allows for easily accommodating a wide variety of artwork metadata and rapidly evolving user-generated content.

Use **PostgreSQL** for transactional data where relationships and data integrity are paramount, such as orders, payments, and artist contracts. The robust ACID compliance and relational integrity will ensure that all transactions are processed reliably.

This hybrid approach allows ArtVista to leverage the strengths of both databases, ensuring flexibility and scalability while maintaining strong data integrity where it matters most.

1.6. Deployment Strategies for the Hybrid Approach

1. Cloud-Based Deployment:

- a. Public Cloud: Utilise public cloud providers like AWS, Azure, or Google Cloud to host both PostgreSQL and MongoDB. These platforms offer managed services (e.g., Amazon RDS for PostgreSQL and MongoDB Atlas) that handle maintenance, backups, and scaling.
- b. Private Cloud: For additional control and security, deploy the databases in a private cloud within a virtual private cloud (VPC) environment. This approach is beneficial if handling sensitive data or needing to comply with specific regulations.

2. **On-Premises Deployment:** Deploy PostgreSQL and MongoDB on dedicated servers in an owned data centre. This approach gives complete control over the hardware and database configurations but requires significant maintenance and upfront investment.
3. **Hybrid Cloud Deployment:** Deploy the databases across different cloud providers to avoid vendor lock-in and increase resilience. For instance, you might use AWS for PostgreSQL and Google Cloud for MongoDB, or vice versa.
4. **Hybrid On-Premises and Cloud:** Keep sensitive data on-premises for security or compliance reasons and use the cloud for less sensitive operations. This approach allows you to balance control with the scalability and features of cloud services.
Alternatively, cloud bursting can be employed to maintain the primary database on-premises and use cloud resources to handle peak loads or as a backup and disaster recovery solution.
5. **Microservices and Containerisation:**
 - a. **Docker:** Use Docker containers to package each database instance with its dependencies, ensuring consistency across different environments and simplifying deployment and scaling.
 - b. **Kubernetes:** Manage these containers using an orchestration system like Kubernetes, which can automate deployment, scaling, and operations of application containers across clusters of hosts.

Other database deployment considerations include:

6. **Disaster Recovery and High Availability:**
 - a. **Multi-Region Deployment:** Deploy the databases in multiple regions to ensure high availability and disaster recovery. This approach is crucial for maintaining service continuity in case of regional outages.
 - b. **Backup and Restore Strategies:** Implement regular backup procedures and test restore processes to ensure data integrity and availability.
7. **Security and Compliance:**
 - a. **Encryption:** Ensure that data is encrypted at rest and in transit, utilising built-in features of PostgreSQL and MongoDB or third-party tools.

- b. Access Control: Implement strict access controls and auditing to monitor who accesses the data and when.

8. Monitoring and Maintenance:

- a. Performance Monitoring: Use tools to monitor the performance of both databases, ensuring they are operating optimally and identifying and troubleshooting issues.
- b. Regular Updates: Keep the database systems updated with the latest patches and versions to ensure security and performance.

1.7. Alignment of Solution with ArtVista's Goals

1. Diverse Data Types and Structures:

MongoDB's flexible, schema-less design is ideal for handling the diverse and evolving data types associated with artworks and user-generated content. Meanwhile, PostgreSQL's structured schema is perfect for transactional data like sales, where consistency and integrity are paramount.

2. Scalability and Performance:

MongoDB excels in horizontal scalability, making handling large volumes of unstructured data and high write loads easier. PostgreSQL provides robust performance for complex queries and transactions, ensuring the marketplace operates efficiently.

3. Data Integrity and Reliability:

PostgreSQL's ACID compliance and strong transactional integrity ensure that all marketplace transactions are processed reliably. This maintains trust and legal compliance in the platform's operations.

4. Complex Queries and Analytics:

PostgreSQL's powerful SQL querying capabilities allow for complex analytics and reports. It aids ArtVista to gain deep insights needed to understand the market and optimise operations.

5. Flexibility and Rapid Development:

MongoDB's schema-less nature allows for rapid development and iteration, enabling ArtVista to quickly introduce new features and adapt to changing market demands without worrying about schema migrations.

6. User Experience and Performance:

The hybrid approach allows for optimising each aspect of the platform's operations. MongoDB can serve content quickly and handle large, unstructured data sets efficiently, while PostgreSQL ensures that transactions are processed rapidly and reliably.

7. Cost-Effectiveness and Resource Optimisation:

Using a hybrid approach, ArtVista can optimise costs by choosing the right tool for each job, avoiding over-provisioning, and leveraging cloud-based solutions for scalability and maintenance.

8. Security and Compliance:

Both PostgreSQL and MongoDB offer robust security features. PostgreSQL's strong access controls and transaction logs are vital for sensitive transactional data, while MongoDB provides flexible security models for diverse datasets.

1.8. Summary, Key Findings and Recommendations

ArtVista handles a wide variety of data. This diversity requires a flexible approach to data management. Maintaining data integrity and consistency is crucial for transactional data, such as purchases and artist contracts. As an online marketplace, it is expected to grow in terms of the number of users and the volume of data; scalability, rapid development, data backup and security are important processes to accommodate this growth without performance degradation. General Recommendations for the database to be implemented include:

1. **Implement Robust Security Measures:** This includes encryption, secure access controls, regular security audits, and staying updated with the latest security protocols and patches. Data security is vital in the database as it protects sensitive information, maintains user trust, ensures regulatory compliance, and prevents Financial Fraud.

2. Regular and Redundant Backups: To ensure redundancy, perform regular backups and store them in multiple locations, including off-site or cloud storage. Frequent data backup provides disaster Recovery, maintains data integrity, and aids business continuity and historical data preservation.
3. Continuous Monitoring and Testing: Regularly monitor security systems for breaches or vulnerabilities and periodically test backup processes to ensure data can be effectively restored.

References

- [1] T. Capris, P. Melo, N. M. Garcia, I. M. Pires, and E. Zdravevski, "Comparison of SQL and NoSQL databases with different workloads: MongoDB vs MySQL evaluation," *IEEE Xplore*, Oct. 01, 2022. <https://ieeexplore.ieee.org/abstract/document/10041513>
- [2] G. Mihai, "Comparison between Relational and NoSQL Databases," *Annals of Dunarea de Jos University of Galati. Fascicle I. Economics and Applied Informatics*, vol. 26, no. 3, pp. 38–42, Dec. 2020, doi: <https://doi.org/10.35219/eai15840409134>.
- [3] N. B. Seghier and O. Kazar, "Performance Benchmarking and Comparison of NoSQL Databases: Redis vs MongoDB vs Cassandra Using YCSB Tool," *2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI)*, Sep. 2021, doi: <https://doi.org/10.1109/icrami52622.2021.9585956>.
- [4] M. A. Kausar, M. Nasar, and A. Soosaimanickam, "A Study of Performance and Comparison of NoSQL Databases: MongoDB, Cassandra, and Redis Using YCSB," *Indian Journal Of Science And Technology*, vol. 15, no. 31, pp. 1532–1540, Aug. 2022, doi: <https://doi.org/10.17485/ijst/v15i31.1352>.
- [5] R. Parate, "NoSQL Databases: Facebook Case study and Analysis," *International Journal of Creative Research Thoughts*, vol. 11, no. 7, pp. 2320–2882, 2023, Available: <https://www.ijcrt.org/papers/IJCRT2307237.pdf>
- [6] S. Sicari, A. Rizzardi, and A. Coen-Porisini, "Security&privacy issues and challenges in NoSQL databases," *Computer Networks*, vol. 206, no. 1389–1286, p. 108828, Feb. 2022, doi: <https://doi.org/10.1016/j.comnet.2022.108828>.
- [7] J.-K. Chen and W.-Z. Lee, "An Introduction of NoSQL Databases Based on Their Categories and Application Industries," *Algorithms*, vol. 12, no. 5, p. 106, May 2019, doi: <https://doi.org/10.3390/a12050106>.
- [8] J. Monteiro, Filipe Sá, and J. Bernardino, "Experimental Evaluation of Graph Databases: JanusGraph, Nebula Graph, Neo4j, and TigerGraph," *Applied sciences*, vol. 13, no. 9, pp. 5770–5770, May 2023, doi: <https://doi.org/10.3390/app13095770>.
- [9] S. B, "Examining and juxtaposing the two NoSQL databases MongoDB and CouchDB," *International Journal of Research Publication and Reviews*, vol. 3, no. 9, pp. 884–888, Sep. 2022, doi: <https://doi.org/10.55248/gengpi.2022.3.9.28>.
- [10] Christoph Praschl, S. Pritz, O. Krauß, and M. Harrer, "A Comparison Of Relational, NoSQL and NewSQL Database Management Systems For The Persistence Of Time Series Data," 2022

International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Nov. 2022, doi: <https://doi.org/10.1109/iceccme55909.2022.9988333>.

- [11] Didar Yedilkhan, Assel Mukasheva, Dariya Bissengaliyeva, and Yerulan Suynullayev, “Performance Analysis of Scaling NoSQL vs SQL: A Comparative Study of MongoDB, Cassandra, and PostgreSQL,” *2023 IEEE International Conference on Smart Information Systems and Technologies (SIST)*, pp. 479–483, May 2023, doi: <https://doi.org/10.1109/sist58284.2023.10223568>.
- [12] S. Halsne, “Competitive Strategies in the European High North,” *Scandinavian Journal of Military Studies*, vol. 5, no. 1, pp. 31–44, Jan. 2022, doi: <https://doi.org/10.31374/sjms.93>.
- [13] Drazena Gaspar and Ivica Coric, *Bridging relational and NoSQL databases*. Hershey, Pennsylvania: Igi Global, 2018.