Industrial Sciencetech Jurnal
ISSN [e], 2579-4264 | DOI, https,//doi.org/10.26760/jrh
Volume 1 | Number 1
Oktober 2024

Data Storage Database PostgreSQL and JSON File Format in Golang Using Gin-Gonic and Gin-Gonic with GORM

Dewi Rosmala[1], Irsan Rasyidin[2]
[1] Institut Teknologi Nasional Bandung , Bandung, Indonesia

Email, d_rosmala@itenas.ac.id[1], nasrirasyidin@gmail.com[2]

Received 18 Juli 2024 | Revised 7 september 2024 | Accepted 24 Oktober 2024

ABSTRACT

Proficient data management and storage are imperative. Among the various programming languages and frameworks available, Golang Gin-Gonic and Golang Gin-Gonic with GORM are widely favored by developers for their speed and concurrent processing efficiency. Golang Gin-Gonic is especially valued for creating high-performance applications, while GORM extends Golang's capabilities with a powerful GORM for database operations. Prior studies have emphasized Golang's effectiveness in data exchange and its superior response speed and resource efficiency compared to Java and Python. This study aims to analyze the performance of Golang Gin-Gonic and Golang Gin-Gonic with GORM concerning data storage, focusing on PostgreSQL databases and JSON file formats. PostgreSQL was selected for its robustness as an open-source RDBMS, while JSON was chosen for its lightweight and readable format. The research assesses the execution speed and concurrency performance of both frameworks during data storage tasks involving PostgreSQL and JSON. The study investigates the performance of Gin-Gonic and Gin-Gonic with GORM in handling concurrent data storage operations in PostgreSQL and JSON formats. Performance tests measured the duration and anomalies in data storage across three dummy data sizes, 1000, 2000, and 3000. The results indicated that, for basic Golang Gin-Gonic, JSON processing was generally faster than PostgreSQL, with a growing performance gap as data size increased. Specifically, JSON processing was 53.63% quicker for 1000 dummy data, 24.14% quicker for 2000 dummy data, and 43.76% quicker for 3000 dummy data. Conversely, Golang Gin-Gonic with GORM showed superior performance with PostgreSQL, being 21% faster for 1000 dummy data, 20.54% faster for 2000 dummy data, and 33.2% faster for 3000 dummy data compared to JSON processing. These findings imply that while JSON is more suitable for simpler configurations with basic Golang Gin-Gonic, PostgreSQL integrated with GORM offers enhanced performance for handling larger and more intricate data sets. This research provides valuable guidance for software developers in choosing the appropriate framework based on their specific requirements, considering factors such as speed, concurrency, and data storage type.

Keywords, *Golang, Gin-Gonic, GORM, Data Storage, Performance analysis*

1. INTRODUCTION

Data management and storage have become crucial aspects. Various programming languages and frameworks are used to handle these tasks, with Golang Gin-Gonic and Golang Gin-Gonic with GORM being popular choices among developers. Golang Gin-Gonic is known for its high execution speed and efficiency in concurrent performance, making it a primary choice for developing high-performance applications [9]. The GORM library further enhances Golang's capabilities by providing a robust Object-Relational Mapping (ORM) for database interactions, simplifying database operations and increasing maintainability [9].

Previous research has discussed comparisons of programming languages and frameworks. For instance, Wiji and Wiwin [8] demonstrated the successful implementation of Golang-based web services in addressing data synchronization issues at PT Sumber Alfaria Trijaya, Tbk. This research emphasized Golang's efficiency in data exchange with the JSON format [8]. Additionally, Dymora and Paszkiewicz [3] compared the performance of Golang with Java and Python, highlighting Golang's advantages in terms of response speed and resource efficiency. Their study showed that Golang outperformed Java and Python in terms of execution time and memory usage, which is critical in environments requiring fast and scalable solutions [3].

Andersson and Brenden [1] conducted a comparative study on parallelism in Go and Java, highlighting how Go's goroutines provide a significant advantage in handling concurrent tasks, making it a more efficient choice for parallel computing [1]. In contrast, Java relies on threads, which can introduce more overhead in large-scale applications.

Another study by Badalyan and Borisenko [2] explored execution control in Golang and Python for cloud orchestration. Their research concluded that Golang has significant performance benefits when used for automating cloud infrastructure processes, particularly in environments requiring high availability and fault tolerance [2].

This study aims to evaluate the performance of Golang Gin-Gonic and Golang Gin-Gonic with GORM in the context of data storage, focusing on PostgreSQL databases and JSON file formats. PostgreSQL is chosen for its reliability as an open-source relational database management system (RDBMS), offering robustness and the ability to handle complex queries efficiently [4]. JSON, on the other hand, is selected for its popularity as a lightweight and easy-to-read data exchange format, which is crucial in high-performance web applications [10].

Suwarno and Yulandi [5] conducted a similar study comparing Golang with Node.js in backend frameworks, concluding that Golang demonstrated superior concurrency handling, making it a better choice for high-performance web applications [5]. Likewise, research by Galih Wiseso, Imrona, and Alamsyah [4] examined the performance of PostgreSQL, comparing it with NoSQL alternatives like Neo4j and MongoDB, further solidifying PostgreSQL's reliability for handling large datasets [4].

In terms of data serialization, Vanura and Kriz [6] compared the performance of JSON, XML, and binary formats across different programming languages, noting that Golang's serialization libraries for JSON performed particularly well in terms of speed and efficiency [6]. Whitney [7] further analyzed Golang's concurrency model, focusing on Communicating Sequential Processes (CSP), which enables Go to handle distributed computing environments efficiently [7]. This concurrency model is a key reason why Golang Gin-Gonic is favored for building scalable, high-throughput web applications.

The results of this study showed that JSON processing is faster with basic Golang Gin-Gonic, particularly for smaller datasets. Specifically, JSON processing was 53.63% faster than PostgreSQL for 1000 entries [4]. Conversely, Golang Gin-Gonic with GORM exhibited better performance with PostgreSQL, being 21% faster than JSON for the same dataset size [4].

By understanding the performance differences between Golang Gin-Gonic and Golang Gin-Gonic with GORM in this context, this study is expected to provide guidance for software developers in choosing the most suitable framework for their specific needs. Factors such as execution speed, concurrency, and type of data storage are important considerations in selecting a framework for optimal application development [9], [10].

2. METODOLOGY

2.1 Collection of Dummy Data

The data collection process for this research involves creating dummy data in minimal, moderate, and maximal quantities, with the primary goal of analyzing the performance of the Golang Gin-Gonic and Golang Gin-Gonic with GORM. The determination of the division of data quantities aims to ensure that performance observations can be conducted more comprehensively and in detail.

By using three different categories of data quantities—minimal, moderate, and maximal—this study can provide a more comprehensive picture of how the performance of these two with increasing data. The selection of variations in data quantity also allows researchers to detect patterns or trends that may emerge in their performance when faced with different workloads.

The importance of data quantity variation is key to producing reliable and relevant results. By comparing the minimal, moderate, and maximal scenarios, this study not only provides information about the absolute performance of each programming but also allows for the identification of potential problems or specific advantages on different scales.

2.2 General Design

The general design explains the working process of the PostgreSQL database storage system and JSON file format in Golang Gin-Gonic and Golang Gin-Gonic with GORM through flowcharts and case study.

2.2.1 Main Flowchart

Figure 1 shows the flowchart for both Golang Gin-Gonic and Golang Gin-Gonic, including several processes as follows,

a. **Application Configuration**, Configure the application using Golang Gin-Gonic and Golang Gin-Gonic. This can include database connection configuration.
b. **Error Checking**, The application checks for possible errors. If there are errors, the application sends an error message, displays the error message on the console, and terminates. If there are no errors, the application continues its process.
c. **Dashboard Display**, Display a dashboard page to select dummy data in the form of a CSV file and upload it.
d. **Data Upload**, Read the uploaded dummy data and store it in a temporary variable to be inserted into JSON format and the PostgreSQL database.
e. **Insert Dummy Data**, Take the temporary variable containing the dummy data and insert it first into a JSON file, then into the PostgreSQL database.
f. **Error Checking**, The application checks for possible errors. If there are errors, the application sends an error message, displays the error message on the console, and terminates. If there are no errors, the application continues its process.
g. **Select JSON Table Page**, Navigate to the JSON data table page for the next process.
h. **Get JSON Data**, Retrieve all data from the JSON file and display it on the JSON data table page. Search data in the JSON file based on Id, Name, Email, and Gender. Display the retrieved data on the JSON data table page.

i. **Error Checking**, The application checks for possible errors. If there are errors, the application sends an error message, displays the error message on the console, and terminates. If there are no errors, the application continues its process.
j. **Select JSON Update Page**, Navigate to the JSON data update page for the next process.
k. **Update JSON Data**, Retrieve the data based on Id from the JSON file, display it on the JSON data update page, and update the data.
l. **Error Checking**, The application checks for possible errors. If there are errors, the application sends an error message, displays the error message on the console, and terminates. If there are no errors, the application continues its process.
m. **Select JSON Delete Page**, Navigate to the JSON data deletion page for the next process.

Delete JSON Data, Retrieve the data based on Id from the JSON file and delete it or delete all data in the JSON file.

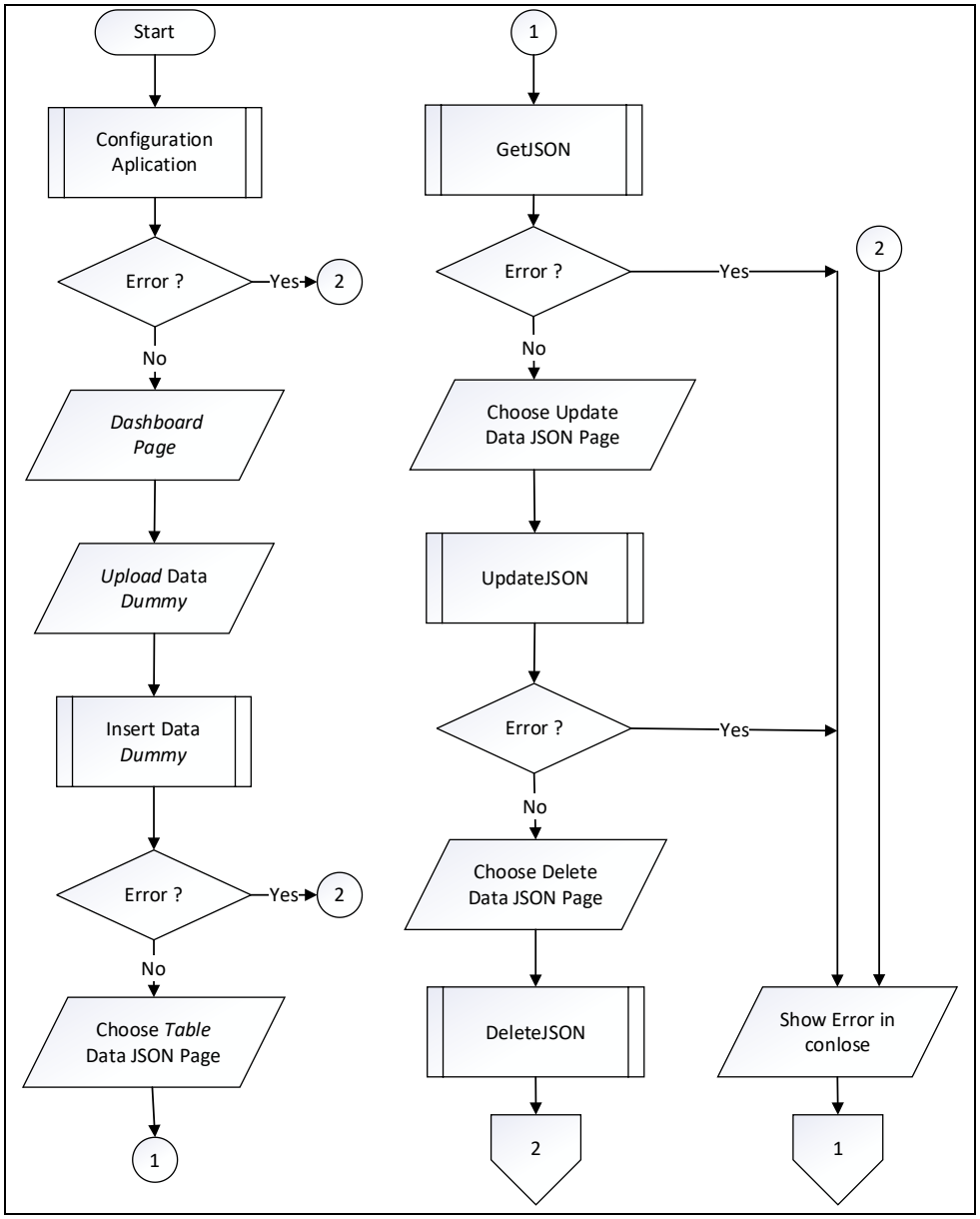

Figure 1. Main Flowchart

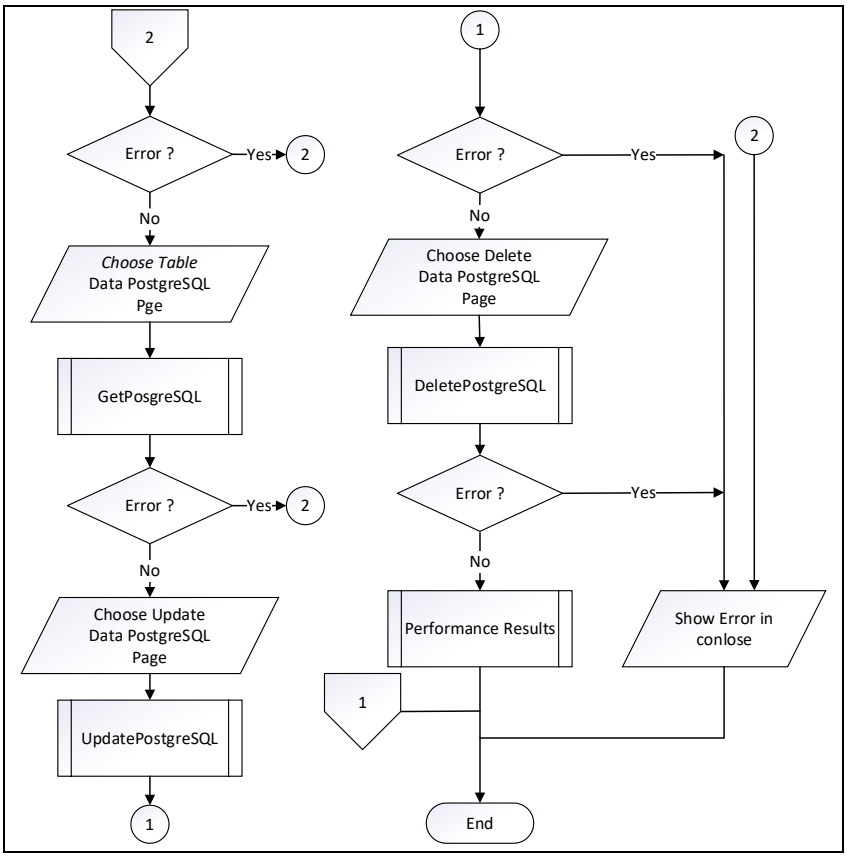

Figure 2. Main Flowchart of Golang(Continued)

Figure 2 show continuation of the flowchart for both Golang Gin-Gonic and Golang Gin-Gonic with GORM applications includes several processes as follows,

a. **Error Checking**, The application checks for possible errors. If there are errors, the application sends an error message, displays the error message on the console, and terminates. If there are no errors, the application continues its process.
b. **Select PostgreSQL Table Page**, Navigate to the PostgreSQL data table page for the next process.
c. **Get PostgreSQL Data**, Retrieve all data from the PostgreSQL database, display it on the PostgreSQL data table page, and search for data in the PostgreSQL database based on Id, Name, Email, and Gender. Display the retrieved data on the PostgreSQL data table page.
d. **Error Checking**, The application checks for possible errors. If there are errors, the application sends an error message, displays the error message on the console, and terminates. If there are no errors, the application continues its process.
e. **Select PostgreSQL Update Page**, Navigate to the PostgreSQL data update page for the next process.
f. **Update PostgreSQL Data**, Retrieve the data based on Id from the PostgreSQL database, display it on the PostgreSQL data update page, and update the data.
g. **Error Checking**, The application checks for possible errors. If there are errors, the application sends an error message, displays the error message on the console, and terminates. If there are no errors, the application continues its process.
h. **Select PostgreSQL Delete Page**, Navigate to the PostgreSQL data deletion page for the next process.
i. **Delete PostgreSQL Data**, Retrieve the data based on Id from the PostgreSQL database and delete it or delete all data in the PostgreSQL database.

j. **Error Checking**, The application checks for possible errors. If there are errors, the application sends an error message, displays the error message on the console, and terminates. If there are no errors, the application continues its process.
k. **Display Performance Results**, Display the execution duration data in bar charts to compare the average execution speed of JSON and PostgreSQL.

2.3 Case Study

Building web applications with the Gin framework in Golang, there are notable differences between using Gin alone and integrating it with GORM for database interactions. Gin alone does not provide built-in support for database operations, requiring developers to use raw SQL or another database library, which increases the code complexity due to the additional boilerplate needed for handling database connections and queries. In contrast, integrating Gin with GORM significantly simplifies database interactions by leveraging GORM's ORM (Object-Relational Mapping) capabilities. This integration reduces the amount of code required and enhances maintainability by allowing developers to use structured commands like db.Create(&user) to insert records into the database effortlessly.
For example, to import, configure the database, and create entries in the database,

Gin-Gonic

```
import (
   "database/sql"
   _ "github.com/lib/pq"
)
```

```
func dbConfig(){
  connStr ,= "host=localhost user=youruser password=yourpassword dbname=yourdb
  sslmode=disable"
  var err error
  db, err = sql.Open("postgres", connStr)
  if err != nil {
    log.Fatal(err)
  }
  defer db.Close()
  if err = db.Ping(); err != nil {
    log.Fatal(err)
  }
  r ,= gin.Default()
}
```

```
func createUser(user User) {
  sqlStatement ,= `INSERT INTO users (username, email) VALUES ($1, $2) RETURNING id`
  err ,= db.QueryRow(sqlStatement, user.Username, user.Email).Scan(&user.ID)
  if err != nil {
    log.Fatalf("Unable to execute the query. %v", err)
  }
  log.Printf("Inserted user with ID %d\n", user.ID)
}
```

Gin-Gonic With GORM

```
import (
   "gorm.io/driver/postgres"
   "gorm.io/gorm"
)
```

```
func dbConfig(){
  dsn ,= "host=localhost user=youruser password=yourpassword dbname=yourdb
  sslmode=disable"
  var err error
  db, err = gorm.Open(postgres.Open(dsn), &gorm.Config{})
  if err != nil {
    log.Fatal(err)
  }
  r ,= gin.Default()
}
```

```
func createUser(user *User) {
  if err ,= db.Create(user).Error; err != nil {
    log.Fatalf("Unable to execute the query. %v", err)
  }
  log.Printf("Inserted user with ID %d\n", user.ID)
}
```

3. RESULTS AND DISCUSSION

3.1 Usage of Dummy Data

Dummy datasets are used for testing and development during the final project development. The number of dummy data entries is 1000, 2000, and 3000, and these are used as representative samples to assess application performance. The dummy data is converted into JSON format and stored in a PostgreSQL database.

The application is tested and evaluated in various scenarios, from small-scale to larger-scale setups, using different types of data. The objective is to identify and address potential performance issues and ensure that the application functions well under various conditions and workloads.

Table 1. Usage for Dummy Data

Label	JSON	PostgreSQL
Insert	30	30
Get ALL	30	30
Get By ID	30	30
Get By Name	30	30
Get By Email	30	30
Get By Gender	30	30
Update By ID	30	30
Delete By ID	30	30
Delete ALL	30	30
Total	270	270

Based on Table 1, testing was conducted for each dummy data set across each type of storage, totaling 270 tests. Overall, the performance results amounted to 3240.

3.2 Evaluation Results for Golang Gin-Gonic

Based on the test results, the Golang using Gin-Gonic produced an average time for JSON processing of 30.435 ms for 1000 dummy data, 59.154 ms for 2000 dummy data, and 85.672 ms for 3000 dummy data. For PostgreSQL, the average times were 46.757 ms for 1000 dummy data, 73.25 ms for 2000 dummy data, and 123.164 ms for 3000 dummy data. In comparison, JSON processing at 1000 dummy data is 53.63% or 16.322 ms faster than PostgreSQL. For 2000 dummy data, JSON processing is 24.14% or 14.275 ms faster than PostgreSQL, and for 3000 dummy data, JSON processing is 43.76% or 37.492 ms faster than PostgreSQL.

Table 2. Evaluation Results for Golang Gin-Gonic

Data	Function Name	Average (ms)	Standard Deviation (ms)	Outlier	Fastest (ms)	Slowest (ms)
1000	**InsertJSON**	1.605	0.728	0	0.707	3.732
2000		2.256	0.603	0	1.244	3.445
3000		2.84	0.477	0	1.763	3.95
1000	**InsertPostgreSQL**	28.021	7.839	0	19.304	43.959
2000		56.415	3.090	0	51.01	63.755
3000		100.011	6.563	0	87.939	112.979
1000	**GetAllJSON**	3.573	0.674	0	2.549	5.771
2000		7.417	0.981	0	5.994	9.761
3000		10.371	0.989	0	8.412	5.768
1000	**GetAllPostgreSQL**	2.386	1.095	0	1.243	5.88
2000		2.636	0.707	0	1.763	4.498
3000		3.981	0.926	0	2.773	12.626
1000	**GetIdJSON**	2.997	0.437	0	2.185	3.847
2000		6.907	0.959	0	5.496	8.74
3000		9.103	0.934	0	7.819	11.156
1000	**GetIdPostgreSQL**	1.067	1.35	0	0.505	7.449
2000		0.666	0.445	0	0.504	2.708
3000		0.592	0.202	0	0.318	1.203
1000	**GetNameJSON**	3.285	0.427	0	2.204	3.974
2000		6.573	0.776	0	5.217	8.921
3000		9.447	0.807	0	7.803	11.832
1000	**GetNamePostgreSQL**	3.95	1.188	0	1.978	8.081
2000		3.884	0.761	0	3.155	7.41
3000		5.717	0.503	0	4.895	7.307
1000	**GetEmailJSON**	3.185	0.408	0	2.152	4.015
2000		6.636	0.635	0	5.479	7.87
3000		10.327	0.858	0	8.391	12.227
1000	**GetEmailPostgreSQL**	4.998	0.82	0	3.853	7.049
2000		4.827	0.661	0	3.547	6.409
3000		6.694	0.352	0	6.04	7.6
1000	**GetGenderJSON**	3.220	0.372	0	2.46	3.939
2000		6.177	0.566	0	5.241	7.844
3000		9.582	0.711	0	8.298	11.357

1000	**GetGenderPostgreSQL**	2.721	1.292	1	0.856	5.249
2000		1.662	0.324	0	1.231	2.31
3000		2.26	0.528	0	1.47	3.386
1000	**UpdateJSON**	7.646	0.824	0	6.518	10.157
2000		14.217	0.829	0	12.662	16.589
3000		21.214	1.377	0	19.219	24.778
1000	**UpdatePostgreSQL**	1.534	0.779	0	0.507	4.308
2000		1.028	0.611	0	0.505	3.248
3000		1.128	0.397	0	0.598	2.219
1000	**DeleteIdJSON**	4.318	0.375	0	3.508	5.019
2000		8.328	0.795	0	6.512	1.094
3000		12.205	1.264	0	10.561	15.524
1000	**DeleteIdPostgreSQL**	0.907	0.314	0	0.505	1.522
2000		0.56	0.138	0	0.503	9.793
3000		0.586	0.218	0	0.504	1.622
1000	**DeleteAllJSON**	0.653	0.343	0	0.503	2.205
2000		0.633	0.239	0	0.096	1.505
3000		0.583	0.179	0	0.504	1.443
1000	**DeleteAllPostgreSQL**	1.173	0.853	0	0.503	3.568
2000		1.736	0.584	0	0.509	3.142
3000		2.195	0.693	0	1.509	4.088

3.2 Evaluation Results for Golang Gin-Gonic with GORM

Based on the test results, the Golang using Gin-Gonic with GORM produced an average time for JSON processing of 33.314ms for 1000 dummy data, 59.164ms for 2000 dummy data, and 89.042ms for 3000 dummy data. For PostgreSQL, the average times were 27.533ms for 1000 dummy data, 49.084ms for 2000 dummy data, and 66.849ms for 3000 dummy data. In comparison, PostgreSQL processing at 1000 dummy data is 21% or 5.781ms faster than JSON. For 2000 dummy data, PostgreSQL processing is 20.54% or 10.08ms faster than JSON, and for 3000 dummy data, PostgreSQL processing is 33.2% or 22.193ms faster than JSON.

Table 3. Evaluation Results for Golang Gin-Gonic with GORM

Data	Function Name	Average (ms)	Standard Deviation (ms)	Outlier	Fastest (ms)	Slowest (ms)
1000	**InsertJSON**	2.828	7.62	0	0.629	43.087
2000		2.821	3.424	0	1.108	20.389
3000		2.855	0.588	0	2.073	4.369
1000	**InsertPostgreSQL**	13.909	3.663	0	10.955	28.82
2000		27.661	8.595	1	20.371	58.812
3000		36.595	5.468	0	30.922	51.315
1000	**GetAllJSON**	3.599	0.918	0	2.117	5.975
2000		7.547	1.809	0	5.631	12.617
3000		10.105	1.252	0	8.607	15.405
1000	**GetAllPostgreSQL**	3.216	0.76	0	2.394	5.233
2000		6.309	0.9	0	4.569	8.743
3000		8.65	1.027	0	6.843	10.794

1000	**GetIdJSON**	3.468	0.718	0	2.152	4.912
2000		6.221	0.601	0	5.055	7.617
3000		9.588	0.765	0	8.169	10.624
1000	**GetIdPostgreSQL**	0.451	0.227	0	0.504	1.035
2000		0.513	0.136	0	0.177	0.84
3000		0.527	0.13	0	0.055	0.995
1000	**GetNameJSON**	3.584	0.55	0	2.364	4.827
2000		6.337	0.863	0	5.155	8.759
3000		9.382	0.752	0	8.471	10.967
1000	**GetNamePostgreSQL**	1.992	0.402	0	1.071	3.25
2000		3.461	0.488	0	2.774	5.382
3000		5.349	0.455	0	4.535	6.664
1000	**GetEmailJSON**	3.62	1.047	0	2.367	6.535
2000		6.437	0.683	0	5.503	8.296
3000		11.732	1.503	0	8.909	15.28
1000	**GetEmailPostgreSQL**	2.773	0.329	0	2.274	3.69
2000		4.869	0.922	0	4.017	9.047
3000		7.901	2.027	0	6.014	14.429
1000	**GetGenderJSON**	3.408	0.788	0	2.185	5.268
2000		6.457	0.493	0	5.648	7.878
3000		12.218	2.513	0	8.54	20.187
1000	**GetGenderPostgreSQL**	1.842	0.479	0	1.171	3.192
2000		2.47	0.572	0	1.704	3.846
3000		3.616	0.715	0	2.582	5.184
1000	**UpdateJSON**	7.336	0.741	0	6.188	9.723
2000		14.345	2.256	0	12.597	25.092
3000		20.862	1.301	0	19.211	25.131
1000	**UpdatePostgreSQL**	1.265	0.771	0	0.506	4.694
2000		1.161	0.629	0	0.506	4.232
3000		1.183	0.516	0	0.562	3.602
1000	**DeleteIdJSON**	4.987	1.498	0	3.299	11.914
2000		8.366	2.427	0	6.872	20.724
3000		11.764	1.092	0	10.391	15.777
1000	**DeleteIdPostgreSQL**	0.779	0.207	0	0.545	1.266
2000		0.914	0.231	0	0.505	1.256
3000		0.726	0.204	0	0.505	1.512
1000	**DeleteAllJSON**	0.574	0.2	0	0.504	1.014
2000		0.633	0.292	0	0.505	1.566
3000		0.536	0.222	0	0.069	1.511
1000	**DeleteAllPostgreSQL**	1.306	0.666	0	0.503	3.325
2000		1.726	0.507	0	1.029	3.081
3000		2.302	0.761	0	1.512	4.381

4. CONCLUSION

Performance testing was conducted to measure the duration and outliers in data storage using PostgreSQL and JSON file format with Golang Gin-Gonic and Golang Gin-Gonic with GORM. The testing was performed in two stages, alpha testing for Golang Gin-Gonic and alpha testing for Golang Gin-Gonic with GORM, using three sizes of dummy data, 1000, 2000, and 3000.

For the evaluation of Golang using Gin-Gonic, the results indicated that the average time for JSON processing was 30.435 ms for 1000 dummy data, 59.154 ms for 2000 dummy data, and 85.672 ms for 3000 dummy data. Conversely, the average times for PostgreSQL were 46.757 ms for 1000 dummy data, 73.25 ms for 2000 dummy data, and 123.164 ms for 3000 dummy data. Comparing the two, JSON processing was 53.63% or 16.322 ms faster than PostgreSQL for 1000 dummy data. For 2000 dummy data, JSON processing was 24.14% or 14.275 ms faster, and for 3000 dummy data, it was 43.76% or 37.492 ms faster than PostgreSQL.

In the assessment of Golang Gin-Gonic with GORM, the findings showed that JSON processing averaged 33.314 ms for 1000 dummy data, 59.164 ms for 2000 dummy data, and 89.042 ms for 3000 dummy data. For PostgreSQL, the average times were 27.533 ms for 1000 dummy data, 49.084 ms for 2000 dummy data, and 66.849 ms for 3000 dummy data. This demonstrated that PostgreSQL processing was 21% or 5.781 ms faster than JSON for 1000 dummy data, 20.54% or 10.08 ms faster for 2000 dummy data, and 33.2% or 22.193 ms faster for 3000 dummy data.

Overall, the performance tests revealed that for basic Golang Gin-Gonic, JSON processing is generally faster than PostgreSQL, particularly as the size of the dummy data increases. However, when using Golang Gin-Gonic with GORM, PostgreSQL exhibits better performance compared to JSON processing. This suggests that while JSON may be more efficient for simpler setups, PostgreSQL with GORM integration provides superior performance for larger and potentially more complex data sets.

When evaluating PostgreSQL performance for both Golang Gin-Gonic and Golang Gin-Gonic with GORM, it was observed that the basic Golang Gin-Gonic setup had slower average times for PostgreSQL across all data sizes compared to its JSON processing counterpart. Specifically, PostgreSQL took 46.757 ms for 1000 dummy data, 73.25 ms for 2000 dummy data, and 123.164 ms for 3000 dummy data, indicating increased processing times as data size grew.

On the other hand, PostgreSQL performance improved significantly when used with Golang Gin-Gonic with GORM. The average times for PostgreSQL were notably faster, 27.533 ms for 1000 dummy data, 49.084 ms for 2000 dummy data, and 66.849 ms for 3000 dummy data. This enhancement in performance highlights the efficiency of integrating GORM with PostgreSQL, making it a more suitable choice for handling larger volumes of data with better processing times.

REFERENCES

[1] T. Andersson and C. Brenden, "Parallelism in Go and Java," Digitala Vetenskapliga Arkivet, 2018.

[2] D. Badalyan and O. Borisenko, "Ansible execution control in Python and Golang for cloud orchestration," SoftwareX, 2022.

[3] P. Dymora and A. Paszkiewicz, "Performance analysis of selected programming languages in the context of supporting decision-making processes for Industry 4.0," MDPI, pp. 1-17, 2020.

[4] L. Galih Wiseso, M. Imrona, and A. Alamsyah, "Analisis performa Neo4j, MongoDB, dan PostgreSQL sebagai database manajemen Big," e-Proceeding of Engineering, p. 9690, 2020.

[5] Suwarno and A. P. Yulandi, "Analisis performa backend framework, Studi komparasi framework Golang dan Node.js," Jurnal Riset Sistem Informasi Dan Teknik Informatika (JURASIK), pp. 155-168, 2023.

[6] J. Vanura and P. Kriz, "Performance evaluation of Java, JavaScript and PHP serialization libraries for XML, JSON and binary formats," Lecture Notes in Computer Science (including

subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 166-175, 2018.
[7] J. G. Whitney, "Distributed execution of communicating sequential process-style concurrency, Golang case study," 2019.
[8] S. Wiji and S. Wiwin, "Implementasi web service dengan metode REST berbasis Golang pada layanan Google Cloud Platform di PT Sumber Alfaria Trijaya, Tbk," Universitas Kristen Satya Wacana Salatiga, 2020.
[9] R. Santoso and H. Firmansyah, "Efisiensi kinerja framework Gin-Gonic dalam pengembangan aplikasi web," 2018.
[10] Y. A. Susetyo, P. O. Saian, and R. Somya, "Pembangunan sistem informasi zona potensi sumber daya kelautan Kabupaten Gunungkidul berbasis HMVC menggunakan Google Maps API dan JSON," 2018.