

Article

Not peer-reviewed version

Implementing Distributed System using Auto Promote and Web Services

[Wisnu Uriawa](#)*, [Silvia Nurrobiani](#), [Teuku Muhammad Saif](#), [Raden Ibnu Huygenz Widodo](#),
[Yuda Ristian Asgari](#)

Posted Date: 12 July 2024

doi: 10.20944/preprints202406.1995.v1

Keywords: database replication; multi-master method; real-time data availability; PostgreSQL; scalability and reliability



Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Article

Implementing Distributed System Using Auto Promote and Web Services

Wisnu Uriawan *, Silvia Nurrobiani, Teuku Muhammad Saif, Raden Ibnu Huygenz Widodo and Yuda Ristian Asgari

Informatics Department, UIN Sunan Gunung Djati Bandung, Jawa Barat, Indonesia; silvianurrobiani@gmail.com; saifsyafii22@gmail.com; ibnuwdodo@gmail.com; yudaramasenju@gmail.com

* Correspondence: wisnu.uriawan@uinsgd.ac.id

Abstract: While advances in information and communication technology have increased the efficiency of information distribution, it has also brought several challenges to the database industry. One of the main challenges is ensuring the real-time availability of data, which is critical. In addition, downtime in centralized database systems can cause significant failures. Therefore this research applies database replication, especially the multi-master method, is used to increase availability and reduce the possibility of failure. For distributed transaction distribution, this research uses the auto-promote master method and web services for PostgreSQL database replication. The focus of the research is on real-world college site selection transactions, where speed, scalability, and high data availability are critical. The results show that combining these methods can improve system performance and protect data availability. With fast response time, high throughput, and low error rate, the API performs well with various loads. As a testament to scalability and reliability, the system can handle large volumes of traffic without experiencing significant crashes. For containerization, Docker supports efficient and stable performance and ensures consistency and manageability. Future recommendations include improving the data security mechanism and further efforts to optimize the distribution system architecture.

Keywords: database replication; multi-master method; real-time data availability; PostgreSQL; scalability and reliability

1. Introduction

The rapid evolution of information and communication technology has profoundly impacted various aspects of life, making information dissemination more efficient. However, this advancement also introduces new challenges, particularly in the realm of databases. Databases have become indispensable in highly valuable assets. Ensuring real-time data availability can be complex in certain situations, yet it is crucial for both profit-driven institutions and service-oriented organizations [1].

Facing the many requests for information needs in today's global environment, it is necessary to analyze the needs and high availability that are owned to handle the many requests. Companies cannot afford to lose their customers when the services offered become limited mainly due to server downtime which will almost certainly occur and result in high cost losses.[2]

A centralized database system has its own advantages, but when a problem occurs, it will affect all aspects. Therefore, a database system that can be distributed to several place is needed to minimize the problem.

Since the last few decades, database replication has become an important part of information system architecture. In the beginning, database replication was used to duplicate data from one main server to another, improving system availability and performance. However, database replication technologies are constantly evolving, ranging from simple ones like master-slave to more complex ones like multi-master.

The earliest method for replication was master-slave, where one primary server (master) sends data changes to one or more secondary servers (slaves). However, this method has a drawback in terms of availability, because if the master fails, the system will experience downtime until the master is restored or replaced. The multi-master method, on the other hand, allows multiple servers to function

as masters and replicate data to each other, thereby increasing availability and reducing the possibility of single-point failures.

One of the main issues with database replication is ensuring that data is consistent across all nodes of the distribution system. This is especially difficult when data conflicts occur due to changes made on multiple nodes simultaneously. In addition, sending data between servers in a distributed system can lead to increased latency, which can compromise overall application performance. In order for replication to run efficiently without sacrificing performance, proper architectural design is required when managing a distributed system that has multiple nodes. Since data is spread across many places, strong security mechanisms such as encryption and strict access control are required.

The basic unit of replication is the node, which represents one database server. A node can act as a master or as a slave server. There are no standard names for "master" and "slave" nodes, and each company can have different labels for them, for example, for masters this includes primary, publisher, and leader, while for slaves this includes standby, subscriber, and follower. These names can have relationships with different types of replication logic, but often they express the same behavior of the replication process.[3]

There are several reasons for building a distributed database, such as sharing, reliability, availability and query processing speed. The main advantage that we can get from a distributed database is the ability to use and access data simultaneously reliably and efficiently. Replication is useful in terms of improving system performance and protecting the availability of accurate data.[4]

There are many reasons why PostgreSQL was chosen as the database management system (DBMS) for this research. PostgreSQL is considered to be one of the most powerful and flexible DBMS available, with many extensibility capabilities and support for various data types. Moreover, PostgreSQL's built-in replication features, such as streaming and logical replication, enable the use of data replication in various scenarios. In addition, PostgreSQL has extensive documentation and an active community of developers and users, which makes it easy for users to get support and find out the best way to implement database solutions. With features such as ACID transactions and advanced query optimization capabilities, PostgreSQL has reliable capabilities for critical business applications.

One of the main benefits of a distributed database system is its ability to better maintain data availability. Because when one database have a problem, they have a another database can backup that by distributing data across multiple sites or servers. That thing make instances or agencies who use that architecture can avoid the risk of data loss due to failure at central point.

However, implementing a distributed database system comes with its own set of challenges. Careful architectural design is needed so that data can be distributed efficiently without sacrificing performance. In addition, consistent data management and synchronization between locations are important factors in maintaining the reliability of a distributed database system.

The previous research used as a reference in this paper uses the auto promote master method in implementing its database distribution. in another research is uses a web service to send results of transactions performed in a distributed manner to the webmaster in order to obtain data consistency across all nodes. We combine the two methods to produce better data availability with a web service as a sender of distributed transaction data and to handle problems at a node using the auto promote method.

In addition, the implementation of distributed database replication system using web services and auto promote on PostgreSQL is also driven by the development of information technology paradigms that demand higher speed, scalability, and data availability. Modern companies and organizations not only rely on data as a strategic asset, but also expect instant and real-time access to data to support fast and accurate decision making.

This research combines the auto promote master method in PostgreSQL database replication with the use of web services for distributed transaction distribution. The focus is on ensuring better data availability, especially in the context of real work study site selection transactions. The development of

information technology that demands speed, scalability, and high data availability is the main drive behind this implementation.

2. Related Work

In 2020, T. Pohanka and colleagues evaluated database replication mechanisms in PostgreSQL and MySQL for spatial data, comparing the performance of PostgreSQL with the Slony extension and MySQL's native streaming replication. This study tested a real national vector spatial dataset to identify replication process bottlenecks, analyzing factors such as total processing time, CPU workload, network workload, and their impact on computer resources. The results provided insights into the suitability of each replication mechanism for different scenarios, emphasizing the importance of considering server performance, connection bandwidth, and replication techniques when setting up a distributed environment for spatial data [3].

In 2023, D. Muliawan discussed the implementation of synchronous replication in an online KRS data application at Universitas Jabal Ghafur using PostgreSQL to improve data availability and security. The methodology used in this research involved various data collection techniques such as field research, interviews, and observations, resulting in an Entity Relationship Diagram, Data Flow Diagram, and interface designs for administrators, lecturers, and students. The replication implementation aimed to ensure data backup and system stability in case of network issues[5].

In 2023, Shrestha, R., and Tandel, T. presented a research study evaluating methods to compare highly available modern cluster-based database solutions, specifically focusing on Percona XtraDB Cluster and MySQL NDB Cluster. The evaluation method included qualitative attributes such as architecture and replication type, as well as quantitative metrics like throughput and response time. The study highlighted that each solution has advantages and disadvantages, emphasizing the importance of choosing the right solution based on the specific needs of the application or service. The goal of this paper was to provide insights for making informed decisions when selecting high-availability database solutions [6].

In 2021, Mazurova, O. and colleagues explored the implementation of ACID transactions in distributed databases using replication technology, focusing on comparing the performance of MongoDB and VoltDB. This research involved database design, transaction development, and experiments to measure performance metrics. The results indicated that MongoDB is more efficient for read operations, insert operations, and some write requests, while VoltDB is recommended for update and delete operations due to its better performance in various transaction scenarios [7].

In 2021, Triyono, J. and colleagues discussed the implementation of distributed systems using database replication and web services to improve data availability and efficiency in information management at a national seminar. The methodology included system architecture design, system requirements identification, and system implementation. The research results showed that the right combination for implementing distributed applications can be achieved safely. The implementation of web service server and client systems was conducted for practical value transaction operations. Testing was carried out by observing the status of the master and slave, as well as database design. The testing results indicated that the replication process ran smoothly. The conclusion was that the replication process combined with web services could function well. Recommendations for further development included setting up data communication processes, transaction data formats, and scheduling synchronization from master to slave [8].

In 2019, E. Asriyar and T. Sutendi conducted research on the implementation of a master-slave PostgreSQL database replication system using tools such as Repmgr with an auto-promote master feature. The process involved steps such as editing configuration files, creating replica users, setting up firewalls, configuring slave servers, installing REPMGR and PgBouncer, and testing replication and failover scenarios. The goal was to ensure the continuity of database operations if the master server fails by promoting the slave server to become the master [1].

In 2022, N. Azizah and colleagues explored the comparison between single-master and multi-master replication strategies in distributed database systems. The aim was to understand the benefits of distributed databases and their potential as a replacement for centralized databases. Prototype testing of the distributed database showed that single-master replication outperformed multi-master replication in terms of memory usage, CPU usage, and replication speed. This research provided insights into the effectiveness of replication strategies in distributed databases and highlighted the challenges in building distributed database systems [9].

In 2019, A. Heryanto and A. Albert in this paper some of the major issues faced by distributed database systems are discussed. These include real-time availability of good data, efficient system performance, accurate data availability, high data availability, and high transaction performance. System performance, the number of people who can view the data, heavy data network traffic, and data protection from repair, damage, and natural disasters are some of these issues.

To solve this problem, the author offers a multi-master database replication method. This mechanism can form a database cluster with a replication time of less than 0.2 seconds, ensuring high data availability. Applications can read and write from any node through the Galera Replication cluster, which is a MySQL plug-in that enables synchronous master-master configuration across all nodes. By using Linux Virtual Server, the software can also direct network traffic to the appropriate server, which supports shared use and improves system performance.

The results show that this method is successful and yields many great advantages. First, high data availability occurs because data can be accessed synchronously and is not lost despite node failures. Second, high transaction performance is achieved because transactions can be performed synchronously without high latency. Third, the distributed database system can work efficiently and meet the needs of users. Fourth, data is effectively protected against various intrusions and threats, which ensures the availability of accurate data. Finally, data can be available in real-time, which is an important requirement for many applications. Therefore, the multi-master database replication method and Galera Replication successfully overcome the problems in the distributed database system.[4]

In 2019, Arifin and colleagues find the existence of failures on the main server that can cause data to be unavailable when there is a crash or damage to the server. To solve this problem, the author uses the database replication method using a master server and a slave server. The master server is used to perform transactions, and the slave server is used as a backup to anticipate the failure of the master server.

The solution offered is to use the synchronous replication method, the data entered on the master server is also replicated in real-time to the slave server. Thus, slave servers that have the same data can be used to handle transactions and ensure data availability when the master server experiences problems.

The results show that with 100% data accuracy and no time delay, the synchronous replication method can improve data availability. In the test, the slave server can accept client requests with the number of 50, 100, and 250 without any time delay, which shows that the synchronous replication method can be used to increase data availability and anticipate the failure of the main server.[2]

3. Methodology

Replication Manager (REPMGR) with auto promote masterdb is one of the features in replication management for PostgreSQL databases. It is an approach to managing replication that allows a slave database to be automatically promoted to master if the current master fails[1]. Then it uses web services so that the transaction process can be done in a distributed manner in applications that are close to the slave while the transaction results will be sent to the web master.

The methodology used in this study involves several important stages. First, an initial configuration of PostgreSQL replication using REPMGR is carried out. This configuration includes installing REPMGR on all nodes (master and slave), setting replication parameters, and determining which

nodes will be the master and slaves. Additionally, automatic failover scripts are prepared to enable REPMGR to automatically promote a slave to master in the event of a failure.

In computer internetworking terminology, failover is defined as the ability of a system to switch manually or automatically to a backup system when the primary system fails. Therefore, the main objective of failover is to ensure operational continuity by providing a backup system that can replace the failed server.[1]

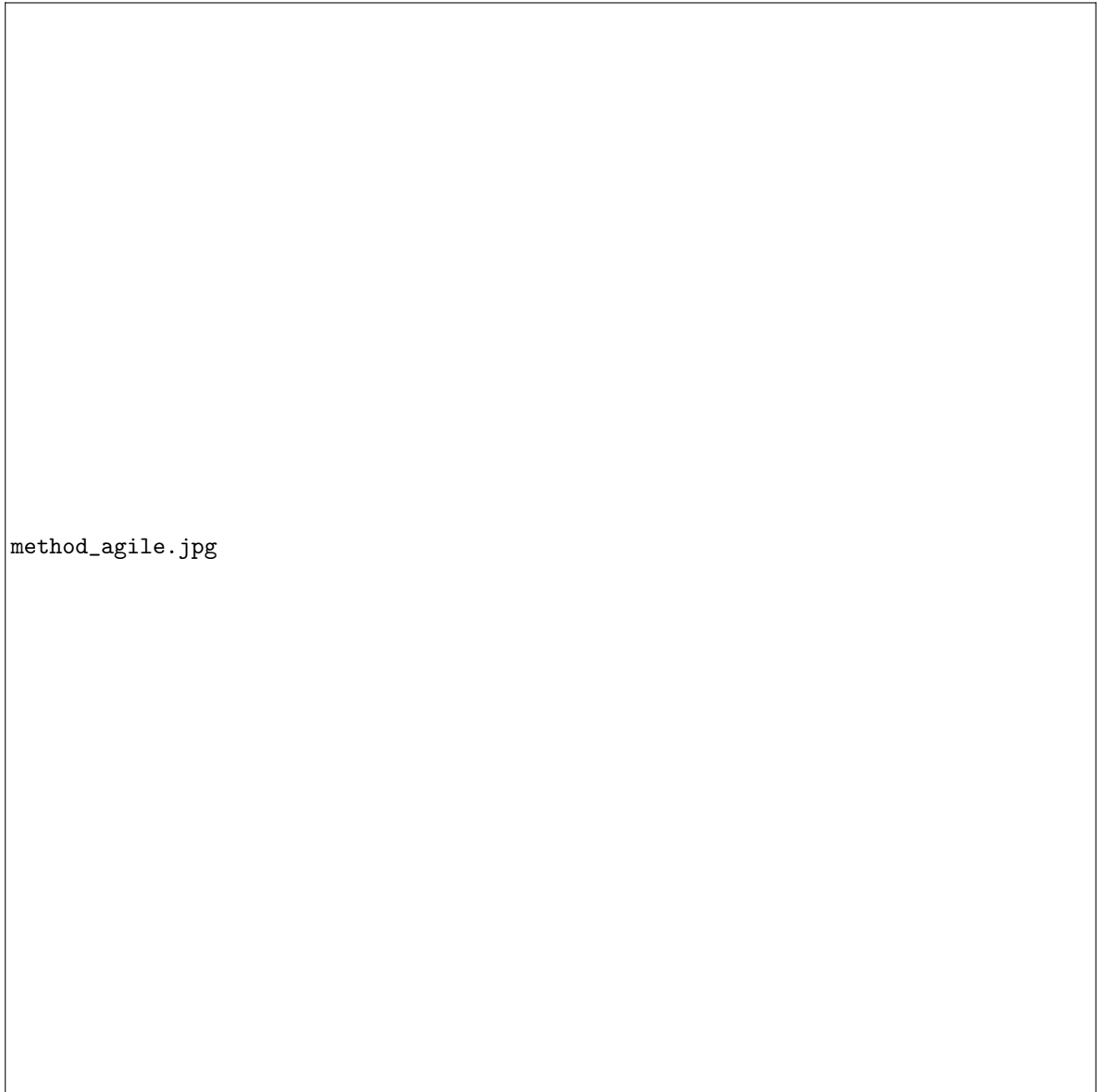
Replication database in PostgreSQL is facilitated by REPMGR, involves copying and maintaining database object across multiple instances to ensure consistency and availability. REPMGR is a robust tool designed to simplify the setup, management, and monitoring of a PostgreSQL replication cluster. It allows administrators to easily configure replication by registering master and standby nodes, initializing replication, and ensuring that changes in the master database are replicated to standby database seamlessly[10].

Database auto-promotion refers to the automatic evaluation of a database that is not under high load to the role of master in case of failure or planned maintenance of the current master. This process on PostgreSQL databases can be managed using software such as REPMGR, ensuring high availability and minimizing downtime by detecting master node failures and promoting appropriate standby nodes to take over the master role without manual intervention. This ensures the database continues to serve and data integration is still available for as long as possible[1].

After the replication configuration is complete, the next step is setting up the web service for transaction distribution. This web service is designed to direct transactions to applications that are close to the slave. This is done to enhance the efficiency and speed of transaction processes by minimizing network latency. Each transaction result processed by the slave is then sent to the web master for synchronization and data consolidation.

The software development life cycle used to create this web service is the agile model. The agile model is used to develop a web service that will handle the transaction distribution process. Agile is an approach that focuses on change readiness at the development stage.[11]

This method has advantages in continuous quality improvement, an approach that prioritizes active collaboration with customers, adaptability to change, so as to achieve high success with maximum results, making this method popular among software developers.[12]



method_agile.jpg

Figure 1. Agile Software Development Life Cycle

Based on the following Figure 1, the development stages use an agile approach:

1. Planning

The planning stage in Agile is akin to the initial configuration of PostgreSQL replication using REPMGR. This phase begins with data collection and analysis of both functional and non-functional requirements. In the context of a software engineering project, these requirements are gathered from various stakeholders, such as students, advisors, and administrators[11].

Understanding the needs of each user group is crucial in this stage. For example, students might need a system that is intuitive and easy to use, while advisors may require robust data analysis tools. Administrators, on the other hand, would likely prioritize security and reliability.

By thoroughly analyzing these requirements, the team can develop a comprehensive plan that outlines the scope, objectives, and deliverables of the project. This plan serves as a roadmap for the subsequent stages of development, ensuring that all stakeholder needs are met effectively.

2. Design

The design stage in Agile involves preparing an automatic failover script for REPMGR. This script is essential for designing a system that can automatically promote a slave to master when a failure occurs. This aspect of design ensures that the system remains robust and resilient against potential failures.

In addition to the failover script, the design phase also includes creating solution designs such as Use Case Diagrams, User Interfaces, Activity Diagrams, and Entity Relationship Diagrams. These diagrams provide a visual representation of the system's functionality and architecture, aiding in the understanding and communication of the system's design.

By meticulously planning these elements, the design phase sets a strong foundation for the development phase. It ensures that all team members have a clear understanding of the system's structure and behavior, which is crucial for the successful implementation of the project.

3. Development

The development phase in Agile is similar to the configuration and development of a PostgreSQL replication setup using REPMGR. During this phase, the development team works iteratively to build the required features and functionality, ensuring that each iteration adds value and brings the project closer to completion.

This iterative process allows for continuous feedback and improvement, ensuring that any issues or changes in requirements can be addressed promptly. The team continuously integrates and tests new features, ensuring that the system remains functional and meets the users' needs.

By focusing on incremental development and regular testing, the development phase ensures that the project progresses smoothly and efficiently. This approach minimizes risks and ensures that the final product is of high quality and meets all specified requirements.

4. Testing

Testing in Agile and the use of REPMGR focus on ensuring the reliability of the replication system. This phase involves validating the automated failover scripts to ensure that they function correctly and that the system can handle failovers without data loss or downtime.

Various types of testing are conducted, including unit tests, integration tests, and system tests. Each type of test serves a specific purpose: unit tests validate individual components, integration tests ensure that components work together correctly, and system tests verify the overall functionality of the system.

Through rigorous testing, any issues or bugs can be identified and resolved before deployment. This process is crucial for maintaining the system's reliability and performance, ensuring that it can meet the demands of its users effectively.

5. Deployment

The deployment stage in Agile involves the final configuration of PostgreSQL replication and the preparation of web services for transaction distribution. This phase ensures that the system is ready for use and that all necessary components are properly configured and integrated.

Deployment includes setting up the production environment, transferring data, and configuring network settings. It also involves ensuring that the system can handle real-world usage and that all failover mechanisms are operational.

By carefully planning and executing the deployment phase, the team can ensure a smooth transition from development to production. This minimizes potential disruptions and ensures that users can begin using the system efficiently and effectively.

6. Review

Reviews in Agile evaluate the development process and the system's performance. This phase involves monitoring the replication and failover processes to ensure they are working correctly and making adjustments as needed.

The review phase includes collecting feedback from users and stakeholders to identify any issues or areas for improvement. This feedback is crucial for continuous improvement and for ensuring that the system meets the evolving needs of its users.

By regularly reviewing and refining the system, the team can maintain high standards of quality and performance. This iterative process of evaluation and improvement is a key aspect of Agile development, ensuring that the project remains aligned with its goals and objectives.

This method makes the replication system more robust and resistant to failure. In addition, deploying transactions through web services close to the slaves reduces the master’s tasks and speeds up application response time. The results show that using REPMGR along with auto promote and web service-based transaction distribution can significantly improve system availability and performance. Figure 2 shows how the rempmgr flow will be created and integrated with agile methodology.

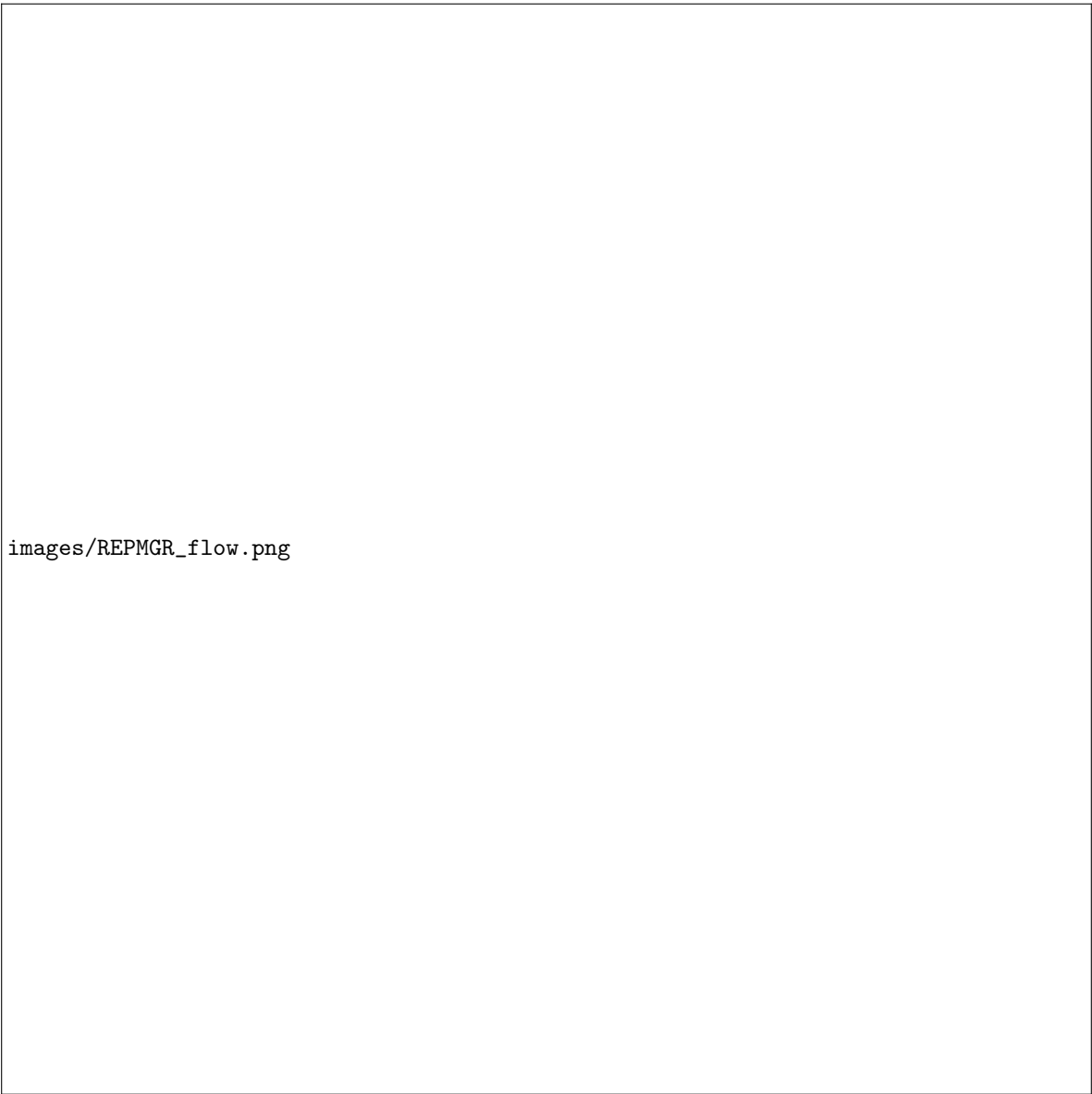


Figure 2. REPMGR Flow

4. Result and Discussion

4.1. Result

In this section, the results of the system development using Agile methodology to implement the REPMGR feature in PostgreSQL database replication management will be discussed in detail. The methodology used includes several key stages, including planning, design, development, testing, deployment, and continuous evaluation. Each of these stages is designed to ensure that the implemented replication system is not only reliable but also able to adapt to changes and meet user needs efficiently.

4.2. Planning

System will have functionality to replicate database automatically and using can capture the transaction of database operation then send it using web service to another database. The database that choose for save transactions data is should the reliable one or the lowest traffic, so when inserting data to that database can save it properly.

Specification of the system will Application Programming Interface (API) based, because on this study the focus is on how database replication with auto promote can handle data transaction with lot of users and can the API send transaction data to idle one database.

Functional Requirement:

1. Create, Read, Update, Delete (CRUD) operation must be can handle by the system
2. The database can replicate data from other nodes using tools like REPMGR for PostgreSQL
3. Web service can perform transaction record insert operations into the database based on reliability and low traffic criteria
4. Direct monitoring and management of the replication status between database nodes. This includes monitoring node health, ensuring consistent data synchronization, and automatically managing failover processes.
5. The system should be able to optimize data replication performance by considering factors such as throughput, latency, and network resource usage. This is important to ensure that the replication process does not interfere with the performance of the main application.
6. Provide adequate logging and auditing features to record all transaction operations that occur. This includes data change logging, replication event logs, and user activity traces for monitoring and auditing purposes.

Non-Functional Requirement:

1. The database chosen to store transaction data must be reliable, so that the entered data can be stored correctly without data loss or duplication.
2. The system must be able to handle a large number of users with efficiency, especially in managing transaction operations using APIs and replicating data between database nodes.
3. The system must be able to be easily expanded according to the growth in the number of users and the volume of incoming transaction data.
4. Sensitive data protection and proper access rights settings must be implemented to ensure the security of CRUD operations and data replication.
5. Web services should be designed to minimize network latency when saving transaction operations to other selected databases.
6. The system must remain available around the clock to ensure continuity of service, especially in the context of automatic failover management using REPMGR.

4.3. Design

System Design gonna be the next step after requiring and analyzing what software needs and feature. For more comprehensive understanding about the system that will develop, diagram have good advantage to handle that, especially Unified Modeling Language (UML).

The Design will cover the use case, activity, entity relational and physical data model. All of that diagram is enough to develop software using agile methodology. Architecture of system will be create using flowchart.

In addition, in this system design phase, the key technologies that would be used to develop the software solution were determined. PostgreSQL was chosen as the main database management system, with the help of Replication Manager (REPMGR) to manage database replication and ensure high data availability.

The ExpressJS framework will be used to develop the backend or API of the application, utilizing its speed and flexibility in building robust and efficient web services. The selection of this technology is not only based on its technical capabilities, but also its compatibility with Agile methodologies, which allow teams to iteratively build, test, and refine solutions based on feedback from users and stakeholders. With this approach, it is expected that the developed system can provide optimal added value in accordance with the needs that have been analyzed in depth beforehand.

1. Architecture of the system

System architecture is the basic structure that defines the major components of a software system, how they interact, and how they are organized to achieve the overall system goals.

The Figure 3 shows the system architecture for an application that uses database replication with auto-promote capabilities using PostgreSQL and REPMGR. The system consists of several key components that are interconnected to ensure data reliability and availability. MainDB1 and MainDB2 are two instances of PostgreSQL managed by REPMGR.

MainDB1 serves as the main database, while MainDB2 is the replica. REPMGR is responsible for managing replication and auto-promote, so that if MainDB1 fails, MainDB2 can immediately become the main database without significant disruption.

The Load Database Handler Server acts as an intermediary that manages data traffic between MainDB1, MainDB2, and other system components, helping to optimize load and ensure data requests are processed efficiently.

The Main Server is responsible for handling requests from users and directing those requests to the appropriate components in the system, ensuring that users can access the data and services they need in real-time.

The Transaction Server handles all transaction operations performed by users, and these transactions are then stored in the TransactionRecordDB to ensure data integrity and auditability. Users interact with the system through the Main Server, sending requests and receiving responses from this server, which then accesses the required data from the relevant database or transaction server. This architecture is designed to ensure reliable data replication, efficient transaction handling, and high data availability, thus providing an optimal experience for users.



Figure 3. Architecture System

2. Use Case Diagram

Use Case Diagram is a graphical description or framework in the UML modeling language that serves to determine the functionality of a system by showing the relationship between actors, use cases, and relationships within it.[\[13\]](#)



Figure 4. Use Case Diagram

3. Activity Diagram

This activity diagram illustrates the detailed steps in the process of registering for the Community Service Program (KKN) by students. The process begins with students initiating by logging into the system. Students must enter their credentials (username and password) to access the system. Once successfully logged in, they are directed to select the KKN registration menu available on the system interface.

The next step is filling out the KKN registration form. At this stage, students are required to provide various necessary information such as personal details, academic information, or their preferred KKN location. After ensuring that all the data has been accurately filled out, students then submit the form through the system.

The system then proceeds to verify the information provided in the registration form. This verification aims to ensure that all the data entered by the students are valid and comply with the KKN registration requirements. There are two possible outcomes from this verification process: if the data is valid, the system will save the information and confirm that the registration has been successful. Conversely, if the data is invalid or incomplete, the system will display an error message. Students will be asked to correct the invalid data and resubmit the registration form.

Finally, if all the submitted data is successfully validated, the system will display a message confirming that the KKN registration has been successful, and this process reaches its endpoint (End). If not, students need to return to the form-filling step to correct the incorrect or missing information, and then resubmit for further verification.

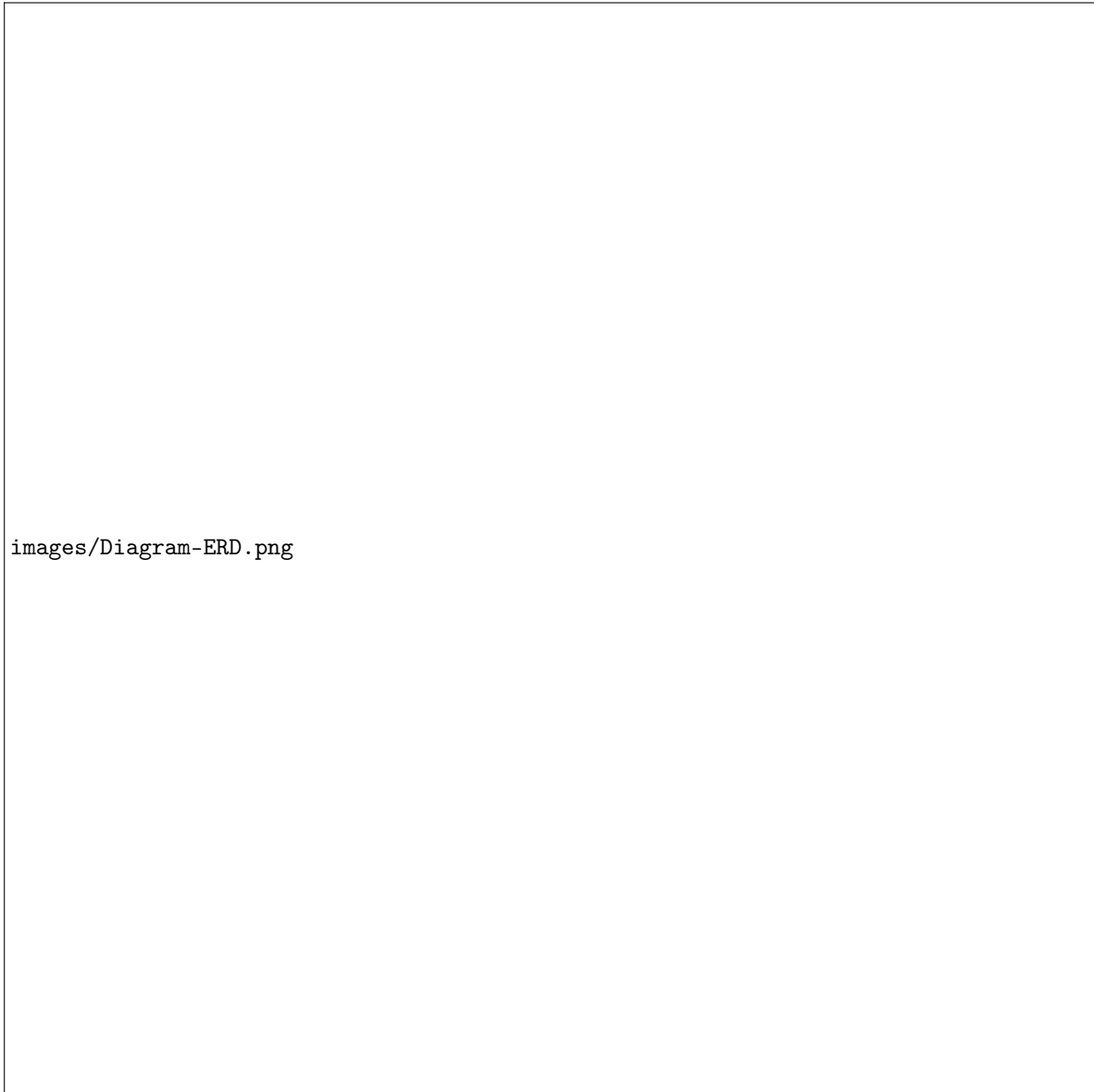
This activity diagram is crucial for mapping the entire workflow of the KKN registration process, from logging in to confirming registration or handling errors. It aids in understanding and ensuring that each step in the registration process is correctly and efficiently implemented within the system. Additionally, it provides a clear guide for the development and improvement of the KKN registration system in the future.



Figure 5. Activity Diagram

4. Entity Relational

The ERD (Entity Relational Diagram) diagram describes the relationship between entities in the KKN system database design and shows how the relationship between objects or entities and their attributes in detail. This diagram consists of five main entities including students, registration, mentors, KKN locations, and admin. The following is a detailed explanation of the attributes and relationships of each entity.



images/Diagram-ERD.png

Figure 6. Entity Relational Diagram

Entity: Mahasiswa

(a) Attributes:

- i. Id Mahasiswa (PK): id of each student.
- ii. nama: name of each student.
- iii. NIM: nim of each student.
- iv. email: email of each student.
- v. password: The password of each student.
- vi. status: participant status of each student.

(b) Relation:

- i. Mahasiswa have a one-to-many (1:n) relationship with Pendaftaran entity, which means that each student can only make one registration.

Entity: Pendaftaran

(a) Attributes:

- i. Id Pendaftaran (PK): Registration Id of each student.
- ii. Id Mahasiswa (FK): Id Mahasiswa of each student.

- iii. Id Pembimbing (FK): Id Pembimbing of each lecturer.
- iv. Id Lokasi (FK): Id Lokasi of each KKN location.
- v. created at: date and time when the user was registration.

(b) **Relation:**

- i. Pendaftaran is connected to Mahasiswa entity through the Id Mahasiswa, indicating that each registration data will be associated with one specific student.
- ii. Pendaftaran is connected to the Pembimbing entity through the Id Pembimbing, indicating that each registration data will be associated with one specific supervisor.
- iii. Pendaftaran is connected to the KKN Location entity through the Id Lokasi, indicating that each registration data will be associated with one specific KKN location.

Entity: Pembimbing

(a) **Attributes:**

- i. Id Pembimbing (PK): id of each supervisor.
- ii. nama: name of each supervisor.
- iii. email: email of each supervisor.
- iv. password: The password of each supervisor.

(b) **Relation:**

- i. Pembimbing have a one to many (1:n) relationship with the Pendaftaran entity which means that each supervisor can only make one registration.

Entity: Lokasi KKN

(a) **Attributes:**

- i. Id Lokasi (PK): id of each KKN Location.
- ii. Nama Lokasi: the name of each KKN location name.
- iii. Alamat: address of each KKN location address.
- iv. kuota: quota of each available quota.

(b) **Relation:**

- i. Lokasi KKN have a one to many (1:n) relationship with the Pendaftaran entity, which means that each KKN location is only associated with one registration.

Entity: Admin

(a) **Attributes:**

- i. Id Admin (PK): id of each admin.
- ii. nama: name of each admin.
- iii. email: email of each admin.
- iv. password: The password of each admin.

(b) **Relation:**

- i. Admin have no direct relationships with other entities in the context of registration. The admin acts as a data manager for the Student, Supervisor, and KKN Site entities, but no explicit ERD relationships are required.

5. Physical Data Model The Physical Data Model (PDM) is a detailed representation of the database structure, including tables, columns, data types, constraints, and relationships between tables. Here is the physical data model for the KKN system database design based on the provided Entity Relationship Diagram (ERD):

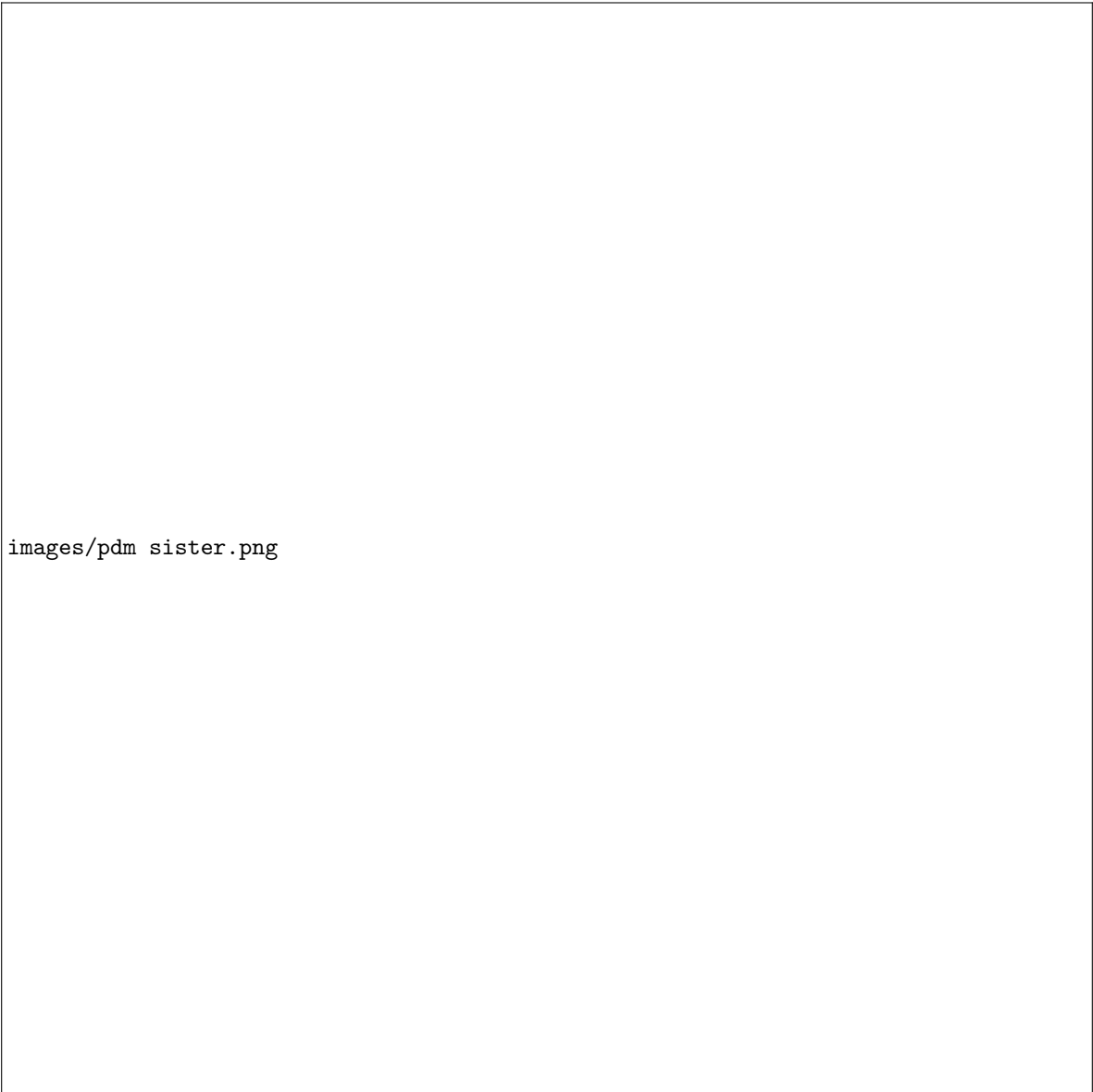


Figure 7. Physical Data Model (PDM)

The physical data model presented is designed for managing KKN (Kuliah Kerja Nyata) activities for university students. It consists of several tables, each representing different entities in the system, including 'Mahasiswa' (Students), 'Pembimbing' (Advisors), 'Pendaftaran' (Registrations), 'Lokasi KKN' (KKN Locations), and 'Admin'. Each table is defined with a primary key (PK) and relevant attributes to ensure data integrity and relationships between entities.

The 'Mahasiswa' table holds student data, identified by 'Id_Mahasiswa' as the primary key. This table includes attributes such as 'Nama' (Name), 'NIM' (Student Identification Number), 'Email', 'Password', and 'Status'. These fields store essential information about students participating in KKN programs, allowing for the identification and authentication of users within the system.

The 'Pembimbing' table represents the advisors, with 'Id_Pembimbing' as the primary key. It contains attributes such as 'Nama', 'Email', and 'Password'. These attributes ensure that advisors can be identified, contacted, and authenticated, playing a crucial role in overseeing and guiding the students during their KKN activities. The relationship between advisors and students is managed through the 'Pendaftaran' table.

The 'Pendaftaran' table serves as a junction table that captures the registration details of students for the KKN program. It includes 'Id_Pendaftaran' as the primary key, and foreign keys (FK) like 'Id_Mahasiswa', 'Id_Pembimbing', and 'Id_Lokasi' to link to the respective 'Mahasiswa', 'Pembimbing', and 'Lokasi KKN' tables. This table also records the timestamp of the registration ('created_at'), ensuring that all registration activities are tracked and can be audited if necessary. Lastly, the 'Lokasi KKN' table stores information about the various KKN locations, identified by 'Id_Lokasi' as the primary key. It includes 'Nama_lokasi' (Location Name), 'Alamat' (Address), and 'Kuota' (Quota) to manage the capacity of each location. Additionally, the 'Admin' table holds data about system administrators, with 'Id_Admin' as the primary key, and attributes like 'Nama', 'Email', and 'Password' for managing access to the system. This structured design ensures that all entities are interconnected, facilitating efficient management of the KKN program.

4.4. Development

The Agile methodology's iterative approach was central to the development of the system, guiding the team through a series of sprints focused on implementing planned features like REPMGR integration and web services for transaction distribution. This iterative process ensured that development progress was steady and aligned closely with stakeholder priorities. Each sprint, typically lasting two to four weeks, allowed the team to deliver tangible improvements and functionalities incrementally. This approach also facilitated ongoing feedback loops, enabling the team to adapt swiftly to changing requirements and user needs as they emerged during development.

The architectural implementation of the system involved configuring three distinct servers, each with defined roles and responsibilities. This architecture was meticulously designed to distribute workloads evenly across the servers, thereby optimizing resource utilization and bolstering overall system reliability. By segregating tasks among these servers—such as database management, application logic processing, and web service distribution—the system could efficiently handle varying demands without compromising performance or scalability.

One of the key advantages of this architectural setup was its robust load balancing capabilities. As users interacted with the system, requests were intelligently routed among the servers based on current loads and specialized functions. This dynamic allocation of resources ensured that no single server was overwhelmed, thereby maintaining consistent response times and minimizing the risk of service disruptions even during peak usage periods.

Moreover, the division of responsibilities among the servers enhanced the system's resilience to failures and faults. In the event of a server malfunction or maintenance requirement, the remaining servers could seamlessly continue operations, mitigating downtime and preserving service continuity. This redundancy was critical for ensuring high availability and reliability, crucial factors for systems operating in mission-critical environments.

Additionally, the architecture's scalability played a pivotal role in accommodating future growth and expansion. By adding additional servers as needed, the system could effortlessly scale its capacity to meet increasing user demands and evolving business requirements. This scalability not only future-proofed the system but also ensured that it could adapt to changing market conditions and operational needs over time.

The successful integration of Agile practices with a well-architected infrastructure enabled the development team to deliver a system that is both adaptable and robust. The iterative nature of Agile development ensured that the system evolved iteratively, with each sprint delivering incremental value and improvements. This iterative refinement process allowed the team to refine features, optimize performance, and address emerging issues in a structured manner.

As the result when combining Agile methodology and a scalable, well-structured architecture empowered the development team to create a system that not only met current requirements but also positioned itself for future growth and innovation. By embracing Agile principles and leveraging a resilient architecture, the team achieved a balance between responsiveness to immediate needs

and long-term scalability, ultimately delivering a high-performance solution capable of meeting the dynamic demands of modern software applications.

The performance of the system, as extensively tested, demonstrates robust capabilities in handling various workloads and maintaining responsive user experiences under different conditions. Through rigorous testing of response times, throughput, error rates, and scalability, the system has shown consistent performance across simulated user loads, from moderate to high intensity. These tests confirm that the implemented architecture, including load balancing and server redundancy, effectively supports the system's ability to manage peak traffic without compromising speed or reliability. Overall, the testing results underscore the system's performance reliability, ensuring it can meet operational demands while providing a seamless user experience.

4.5. Testing

System tests were conducted thoroughly to ensure that the replication features used were operating properly and according to specifications. The tests also tested the reliability of the automated failover scripts and validated the system's performance in managing various workloads and finding and handling failures effectively.

Parameters tested in this study include response time, throughput, error rate, and the number of users the system can handle. The software used for load testing includes Apache JMeter and Postman.

Test cases were defined for each parameter with specific criteria. The first parameter tested is the response time, and the test case design is detailed as follows:

4.5.1. Response Time

The test case design for response time is divided into three parts, with user loads incrementing in each test from 1000, 3000, and 5000 users. The scenario of the test begins with using 1000 virtual users/threads to simulate a medium load on the servers. The second test uses 3000 virtual users to handle a mid to high load. The final test uses 5000 virtual users to test the average maximum user access at the same time. Each test scenario is run 5 times, and the results are recorded.

4.5.2. Throughput

The throughput test aims to measure the system's ability to handle varying loads over time. The throughput is measured in terms of the number of requests processed per second. The test begins with 1000 users, gradually increasing to 3000 users, and finally to 5000 users. Each test scenario runs for a duration of 10 minutes, and the number of successful requests per second is recorded. The results include the average throughput, peak throughput, and any significant drops in throughput during the test period.

4.5.3. Error Rate

The error rate test evaluates the system's robustness by measuring the number of failed requests against the total number of requests. The test is conducted with 1000, 3000, and 5000 users. The duration of each test scenario is 10 minutes, and the error rate is calculated as the percentage of failed requests out of the total requests made. The results include the total number of requests, the number of failed requests, and the calculated error rate for each user load.

4.5.4. User Load

The user load test determines the maximum number of users the system can handle without significant performance degradation. The test starts with 1000 users and increases incrementally by 1000 users until performance degrades significantly. Each load level is maintained for 5 minutes to stabilize and observe the system's behavior. Performance metrics such as response time, throughput, and error rate are recorded. The results include the maximum user load handled, response time, throughput, and error rate at each load level.

These tests provide a comprehensive analysis of the system's performance, reliability, and scalability, ensuring that it meets the required specifications under varying conditions.

4.6. Deployment

The deployment phase involves placing the final configuration of PostgreSQL replication into the production environment using REPMGR. This phase includes the installation and configuration on each master and slave node, as well as setting up web services to support effective transaction distribution. Ensuring that the solution can provide immediate additional value to end users necessitates making adjustments between the development and implementation phases.

The system is encapsulated in containers using Docker as the container manager. This approach standardizes all dependencies and facilitates easy deployment to the server. Docker's use ensures consistency across different environments, reducing the potential for configuration discrepancies and making it easier to manage and scale the system.

In this study, the server specifications used are as follows: 2 Virtual CPUs, 3 GB RAM, 20GB solid state storage, and Ubuntu Server 18.04. These specifications provide a balance between performance and resource utilization, ensuring that the system can handle the expected load while maintaining responsiveness.

During the deployment process, extensive testing is conducted to ensure that all components are functioning correctly. This includes verifying the replication setup, testing failover mechanisms, and ensuring that the web services are correctly distributing transactions. Any issues identified during this phase are promptly addressed to ensure a smooth transition to production.

4.7. Review

Continuous evaluation is conducted to monitor the real-time performance of the system and identify areas that need improvement. This ongoing assessment is critical for maintaining the system's reliability and efficiency over time. By consistently observing the system's behavior under different conditions, we can proactively address any potential issues before they escalate.

With predefined metrics, replication system surveillance plays a vital role in detecting potential issues early on. These metrics include response time, throughput, error rate, and system load, among others. By closely monitoring these parameters, we can quickly identify deviations from expected performance and take corrective actions as needed.

Responding to environmental changes quickly is another key aspect of the evaluation process. The dynamic nature of production environments means that unforeseen issues can arise at any time. Having a robust monitoring and evaluation framework in place allows us to adapt to these changes efficiently, ensuring that the system continues to operate smoothly.

This evaluation process also involves collecting feedback from end users to ensure that the overall system meets expectations. User feedback is invaluable for understanding how the system performs in real-world scenarios and identifying areas for improvement. By incorporating user insights into our evaluation process, we can make targeted enhancements that improve the user experience and system performance.

4.8. Discussion

Based on the testing results presented in the result section, the API demonstrates robust performance under varying loads. The tests conducted for response time, throughput, error rate, and user load indicate that the system can handle the expected traffic efficiently, with minimal errors and acceptable response times even under peak loads. These results are encouraging, as they validate the effectiveness of the replication and failover mechanisms implemented in the system.

The response time tests show that the system maintains quick response times across different user loads. At the initial load of 1000 users, the response times are very fast, demonstrating that the system can handle moderate traffic with ease. Even as the load increases to 3000 and 5000 users, the response

times remain within acceptable limits. This performance indicates that the system's architecture is well-designed to manage high traffic volumes without significant degradation.

Throughput results indicate that the system can process a high number of requests per second, which is critical for maintaining performance during periods of high demand. The tests reveal that the system can handle the incremental increase in user load efficiently, maintaining a steady throughput rate. This consistency in throughput is essential for applications that require real-time processing of large volumes of data, ensuring that the system can meet the needs of its users without delays.

The error rate tests provide insights into the system's robustness and reliability. The low error rates observed during the tests indicate that the system can handle high volumes of requests with minimal failures. This reliability is crucial for maintaining a smooth user experience and ensuring that users can trust the system to perform as expected. The ability to minimize errors even under high load conditions is a testament to the system's stability and the effectiveness of its error handling mechanisms.

The user load tests demonstrate the system's scalability. By incrementally increasing the load and monitoring the system's performance, we can confirm that the system can scale to meet increased demand without significant performance issues. This scalability ensures that the system can grow with user needs, providing a reliable and efficient service over time. The ability to scale effectively is particularly important for applications that anticipate growth in user base or usage intensity, as it ensures that the system can continue to perform well as demand increases.

Additionally, the combination of Apache JMeter and Postman for load testing proved to be effective in simulating real-world usage scenarios. These tools allowed for comprehensive testing of various performance parameters, providing a clear picture of the system's capabilities and areas for potential improvement. The detailed results obtained from these tests offer valuable insights for further optimization and enhancement of the system.

The deployment process, which utilized Docker for containerization, also contributed to the system's robust performance. Docker ensures consistency across different environments, reducing the potential for configuration discrepancies and making it easier to manage and scale the system. This consistency is reflected in the stable performance metrics observed during testing, highlighting the benefits of using containerization for deployment.

In summary, the testing results indicate that the system is well-equipped to handle high traffic volumes with minimal errors and acceptable response times. The robust performance, scalability, and reliability demonstrated in the tests validate the design and implementation of the system. Continuous monitoring and user feedback will further ensure that the system meets user expectations and performs reliably in production environments. These findings provide a strong foundation for future development and optimization efforts, ensuring that the system can continue to deliver high-quality service to its users.

5. Conclusion

This research is expected to provide various significant benefits. The implementation of data replication with regular synchronization on the backup server results in a more even distribution of workload among multiple servers. This not only reduces the risk of overloading the primary server but also improves the system's ability to handle traffic spikes without sacrificing performance. Fast access to data is guaranteed even in the event of a single server failure, significantly improving system availability.

Additionally, the automatic mechanism to promote the backup server to the primary server when a failure occurs optimizes system recovery time, enhances user experience, and effectively maintains service continuity. This ensures that service disruptions are minimized, and users experience consistent and reliable access to services. Overall, the deployment of such a robust replication strategy provides a resilient, high-performance system capable of meeting the demands of modern applications and ensuring uninterrupted service delivery.

References

1. Asriyar, E.; Sutendi, T. Implementasi Sistem Replikasi Database PostgreSQL Master-Slave REPMGR dengan Auto Promote MasterDB. *Jl-Tech* **2019**, *15*, 8–29.
2. Arifin, Z.; Triyono, J.; Rachmawati, R.Y. MEMBANGUN SERVER DAN ANALISIS BACKUP DATABASE POSTGRESQL MENGGUNAKAN TEKNIK REPLICATION MASTER/SLAVE. *Jurnal SCRIPT* **2019**, *7*, 107–114.
3. Pohanka, T.; Pechanec, V. Evaluation of replication mechanisms on selected database systems. *ISPRS International Journal of Geo-Information* **2020**, *9*, 249.
4. Heryanto, A.; Albert, A. Implementasi Sistem Database Terdistribusi Dengan Metode Multi-Master Database Replication. *Jurnal Media Informatika Budidarma* **2019**, *3*, 30–36.
5. Muliawan, D. PENERAPAN REPLIKASI SYNCHRONOUS PADA APLIKASI DATA KRS ONLINE PADA UNIVERSITAS JABAL GHAFUR MENGGUNAKAN POSTGRESQL. *Future Academia: The Journal of Multi-disciplinary Research on Scientific and Advanced* **2023**, *1*, 31–45.
6. Shrestha, R.; Tandel, T. An Evaluation Method and Comparison of Modern Cluster-Based Highly Available Database Solutions. *CLOSER*, 2023, pp. 131–138.
7. Mazurova, O.; Naboka, A.; Shirokopetleva, M. Research of ACID transaction implementation methods for distributed databases using replication technology. **2021**, pp. 19–31.
8. Triyono, J.; Nadira, P.; Subkan, C.A. Implementasi Sistem Terdistribusi menggunakan Replikasi Database dan Web Service. *Prosiding Seminar Nasional Multidisiplin Ilmu*, 2021, Vol. 3, pp. 84–91.
9. Azizah, N.; Hartajaya, V.; Riady, S. Comparison of replication strategies on distributed database systems. *International Journal of Cyber and IT Service Management* **2022**, *2*, 20–29.
10. Nahak, Y.J.S.; Purnomo, H.D. Perancangan Sistem Replikasi Dan Sistem Backup Database Postgresql Menggunakan Repmgr Dan Barman. *Progresif: Jurnal Ilmiah Komputer* **2023**, *19*, 867–877.
11. Faruq, M.N.; Maryam, M. IMPLEMENTASI METODE AGILE PADA PENGEMBANGAN APLIKASI MANAJEMEN PENGELOLAAN LAYANAN WIFI. *JATI (Jurnal Mahasiswa Teknik Informatika)* **2023**, *7*, 3472–3478.
12. Rasheed, A.; Zafar, B.; Shehryar, T.; Aslam, N.A.; Sajid, M.; Ali, N.; Dar, S.H.; Khalid, S. Requirement engineering challenges in agile software development. *Mathematical Problems in Engineering* **2021**, 2021, 6696695.
13. Arifin, M.N.; Siahaan, D. Structural and Semantic Similarity Measurement of UML Use Case Diagram. *Lontar Komputer: Jurnal Ilmiah Teknologi Informasi* **2020**, *11*, 88.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.