Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control
Journal homepage: http://kinetik.umm.ac.id
ISSN: 2503-2267
Vol. 5, No. 2, May 2020, Pp. 161-168

Development and optimization of NoSQL database in food insecurity early warning system based on local community participation

Yani Nurhadryani*[1], Wiradani Ramadhan[2], Auzi Asfarian[3]
Institut Pertaniaan Bogor, Indonesia[1,2,3]

Article Info

Keywords:
Database Design, Database Optimization, NoSQL, Early Warning System, Information System

Article history:
Received 09 February 2020
Revised 12 February 2020
Accepted 30 April 2020
Available 06 May 2020

Cite:
Nurhadryani, Y., Ramadhan, W., & Asfarian, A. (2020). Development and Optimization of NoSQL Database in Food Insecurity Early Warning System Based on Local Community Participation. Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control, 5(2). doi:https://doi.org/10.22219/kinetik.v5i2.1048

*Corresponding author.
Yani Nurhadryani
E-mail address:
yani_nurhadryani@apps.ipb.ac.id

Abstract

As a part of the food insecurity early warning system based on local participation, a robust and scalable database service is required. This necessity caused by the large area of services which include 34 provinces, 416 districts, 7,215 sub-districts and 80,534 villages in Indonesia. The abundant number of the expected daily transaction might not be handled properly using the traditional model. In this research, we design, implement, and optimize the NoSQL database to create scalable, dynamic, and flexible database service for the early warning system. The cohesion of the model is then measured, resulting in 5 entities with high cohesion, 16 with moderate cohesion, and 3 with low cohesion. After refactoring, we reduced the number of the low-cohesion entity into one and increased the average cohesion from 0.62 to 0.67. An empirical experiment was conducted to compare the response time before and after the refactoring. As the results, the average response time is decreased from 11.0 ms to 7.99 ms or equal to 1.38 in speedup. The experiment results suggest there is an impact of the logical data model improvement, by increasing their cohesion, to the performance of the NoSQL database.

1. Introduction

In the previous works [1], we proposed the design of local community based early warning system for food insecurity in Indonesia. By using the system, the local community will send food security observation data on a real-time basis. The data then used by the system to create an early warning of food insecurity incident on the region. As the potential area of service is nationwide (covers 34 provinces, 416 districts, 7,215 sub-districts and 80,534 villages) and around 64.8% of Indonesia citizen has access to the Internet [2], a vast number of observation data will be harvested daily. To process all the data correctly and efficiently, a robust and scalable database service is required. This is a daunting task for the relational database as its performance degrades rapidly as the data volume increases [3]. The option for scaling up using distributed computing also comes with a significant drawback, i.e. maintaining the relationship in the distributed environment [4].

The recent and most popular solution for scalable database services is NoSQL (“not only SQL”) database. Although this technology does not guarantee ACID (atomicity, consistency, isolation, and durability) properties, it guarantees BASE (basically available, soft state, eventual consistency) properties and complies to CAP (consistency, availability, partition tolerance theorem) [3][5][6]. Fundamentally, the NoSQL was designed to give more priority to the availability and scalability of database services than the consistency of the stored data. A number of NoSQL solutions already available in the market can be divided into four distinct categories based on the data model [3]: key-value database (Redis and Flare), column-oriented database (Cassandra and Hypertable), document database (MongoDB and CouchDB), and graph-based database. The performance of read, write, and delete operations of these databases already measured with MongoDB and CouchDB in the top [7][8][9][10].

Nowadays, MongoDB [11] stands as the most widely used NoSQL database management system (DBMS) [12][13]. As a document-based database, MongoDB stored its data as a JavaScript Object Notation (JSON) document instead of a record table. Each document can be considered analogues to an object in object-oriented programming. MongoDB also has a high-performance load balancer using sharding technology [14]. This DBMS used by Google Compute Engine, Facebook, and Adobe. Previous research also implements social networks [15], textbook management system [16], disaster management information system [17], sensor-based internet of things networks [18], and big data analytics platform [19].

The aim of this study is to develop the NoSQL database design of the proposed food insecurity early warning system based on local community participation. We present the design using three-schema architecture (ANSI/SPARC) which includes the conceptual, logical, and physical data model [20][21]. We evaluate and improve the logical model based on the cohesion metric [22][23] to ensure high cohesiveness. Finally, we compare the improvement of the database system before and after the improvement based on cohesion metric.

2. Research Method

The database was designed using three-schema architecture which includes three most common types of data model: the conceptual, logical, and physical data model [20][21].

2.1 Conceptual Data Model

We gather data from previous research [1] to identify the relevant entities involved in the system and their relationship. The system is divided into two parts: an e-participation application for food insecurity reporting and an e-initiative application for crowdfunding campaigns against food insecurity. From the data, we create a high-level map of entities and their relationship.

2.2 Logical Data Model

Logical data modelling is a process of capturing the details of a business process. The process of modelling logical data describes processes that are more detailed than the process of conceptual data [21]. We choose to represent the logical data model using the unified modelling language (UML), i.e. the class diagram [24], as the model also used for the other applications, i.e. the backend service, the web application, and the mobile application.

In this phase, we check the quality of the logical data model by measuring its cohesion [22][23]. Class cohesion refers to the relatedness of the class members and serves as one of the important aspects of the class design quality [25]. The high cohesion value indicates the module already well-divided into smaller components with a strong internal relationship between attributes, methods, and classes. The cohesion value range is between 0 to 1. According to [26], the value between 0.8-1.0 shows high cohesiveness, 0.5-0.8 shows normal cohesiveness, and the rest shows weak cohesiveness. Equation 1 is used to measure the connectedness value from each method in particular class X [27].

$$\mathrm{R(M)}=\frac{\mathrm{MO}}{\mathrm{TG}} \tag{1}$$

R(M)	: connectedness value from method M.
MO	: the number of groups in which methods M appears.
TG	: total number of groups in class X.

To calculate the overall cohesion of each class X, Equation 2 is used to get the average connectedness value from all method in class X.

$$\mathrm{Cohesion(X)}=\frac{\sum \mathrm{R(M)}}{\mathrm{TM}} \tag{2}$$

Cohesion(X)	: class X cohesion value.
$\sum R(M)$	: the total sum of all connectedness values of all methods in class X.
TM	: total number of methods in class X.

Depends on the results of the cohesion measurement, a refactoring of the logical data model will be conducted to achieve high cohesiveness. The comparison of cohesion value before and after refactoring will be compared. In this research, we also do an empirical experiment to measure the impact of the refactoring to the database performance.

2.3 Physical Data Model

The physical data model is more detailed and less generic data model [21]. We implement the physical data model in MongoDB as documents and collections.

3. Results and Discussion

3.1 Conceptual Data Model

The conceptual data model of the database is depicted in Figure 1. The model consists of 22 entities involved in the use case description [1]. Several key entities from previous research are described in Table 1.

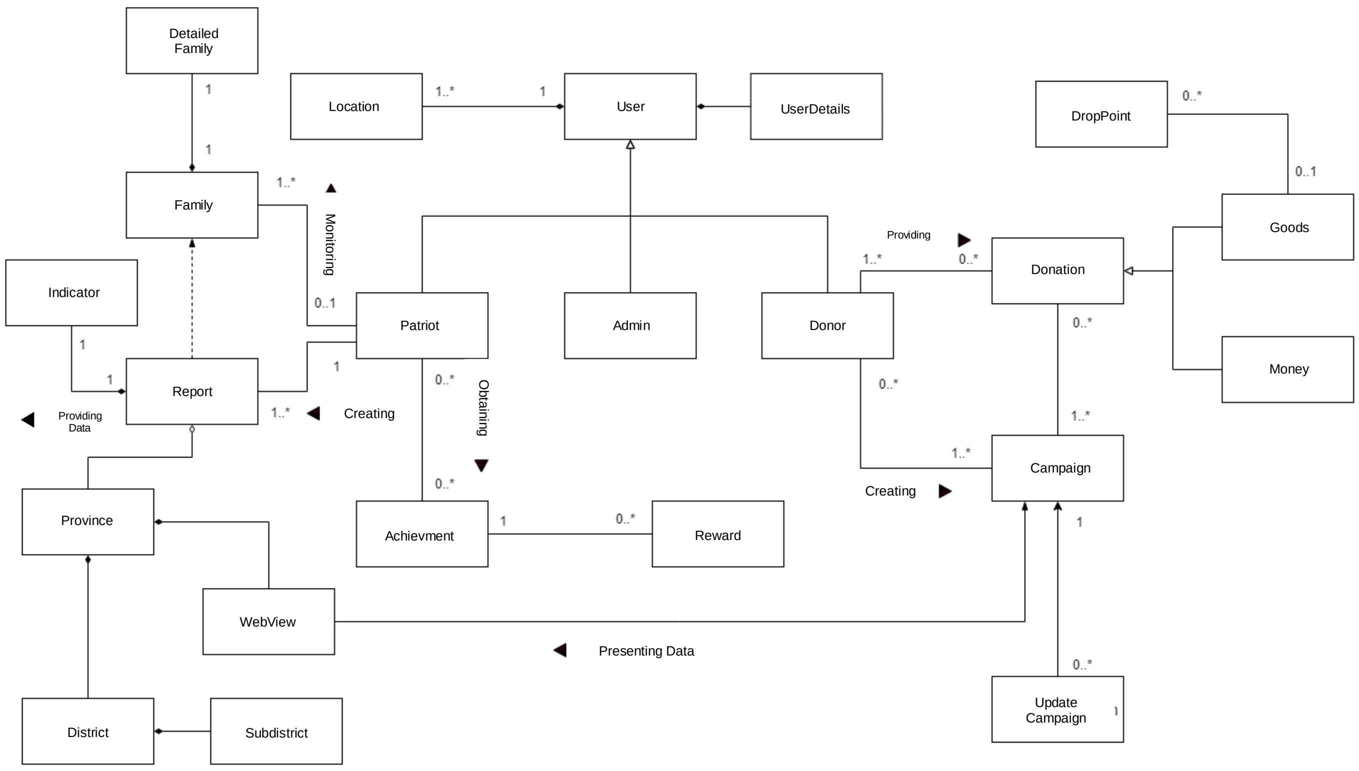


Figure 1. The Conceptual Model

Table 1. The Description of a Key Entities in the Early Warning System

Entity	Description
User	The super class of three main users in the early warning system: patriot, donor, and admin.
Patriot	A local recruited to regularly observe at most ten poorest family in their region. Their identity needs to be verified and they can have a reward for their contribution.
Donor	A donor can donate for a food insecurity campaign.
Admin	An actor whom responsible to verified the patriot and donors. An admin also responsible to input the observed family into the database.
Campaign	A campaign to gather a fund or goods to eradicate observed food insecurity which previously reported by the patriot. A campaign should have a regularly updated information.
Family	A high-risk family that prone to food insecurity.
Report	The regular report from the patriot based on their observation. This report consists of eleven yes-no indicators that can be processed to predict food insecurity incident.
Reward	A virtual reward in form of badge given to an active patriot.
Donation	Money or good donation from donors to a campaign.

3.2 Logical Data Model

In the logical data model, we design a more detailed diagram of the data model. We use a class diagram to depict the entity along with their relationship and attributes Figure 2. Afterwards, the cohesion value of each class in the logical data model will be calculated.

3.2.1 Evaluation of Logical Data Model by Cohesion Measurement

In this phase, we evaluate the logical data model using cohesion metrics [27]. For illustration, we illustrate this process using the User class in our logical data model. Table 2 illustrates the four methods and five attributes mapping in the User class. From this mapping, each attribute then grouped based on their related methods Table 2. Using Equation 1, the connectedness value of each method can be calculated Equation 3, Equation 4, Equation 5, and Equation 6.

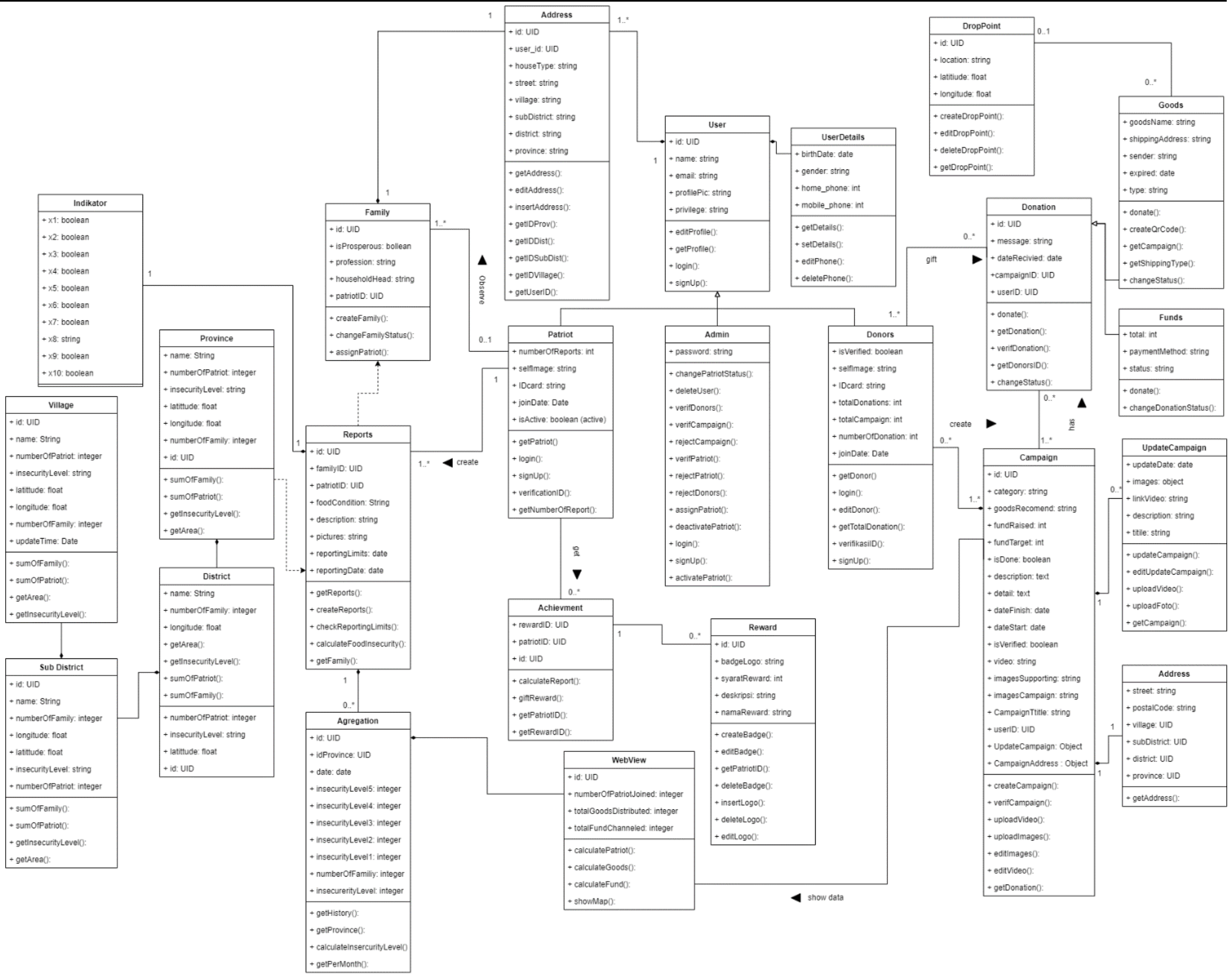


Figure 2. The Logical Model Presented Using a Class Diagram

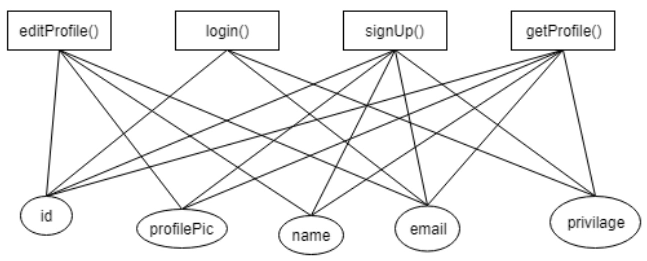


Figure 3. Methods and Attributes Mapping Example of the User Class

Table 2. Usage Mapping Results of Attributes of the User Class

Class	Atribute	*method*	Group
User	id	editProfil(),getProfil(),login(),signUp()	1
	email	editProfil(),getProfil(),login(),signUp()	1
	name	editProfil(),getProfil(),signUp()	2
	profilePic	editProfil(),getProfil(),SignUp()	2
	privilege	getProfil(),login(),signUp()	3

$$R(getProfil)=\frac{3}{3}=1.00 \tag{3}$$

$$R(signUp)=\frac{3}{3}=1.00 \tag{4}$$

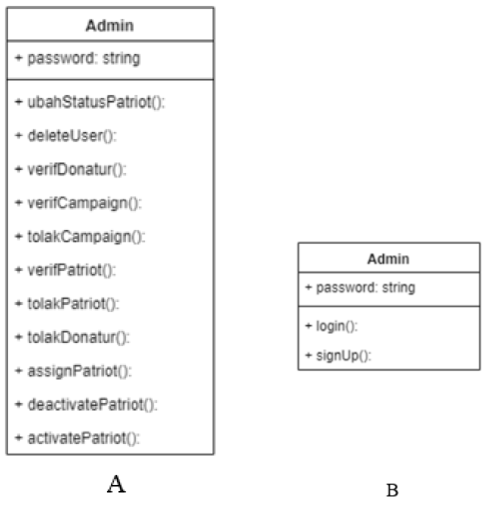
$$R(login)=\frac{2}{3}=0.67 \tag{5}$$

$$R(editProfil)=\frac{2}{3}=0.67 \tag{6}$$

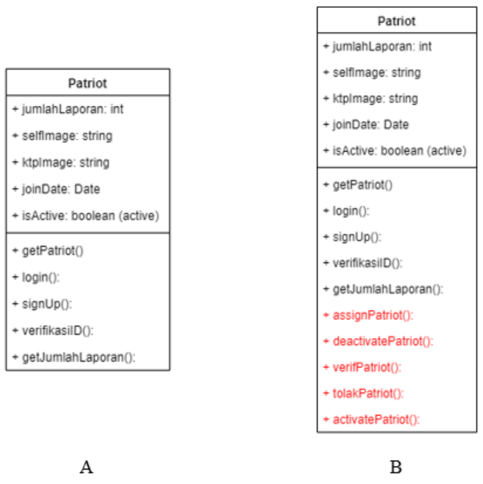
These values are then used in Equation 2 and the cohesion of the User class can be calculated and equal to 0.83 Equation 7. The cohesion value of the rest of the class is presented in Table 3. From the 22 classes measured, there are 5 classes with high cohesiveness, 16 with normal cohesiveness, and 3 with low cohesiveness. The low-cohesive classes are Donor, Patriot, and Admin. They are the classes which represent the user in the data model. To improve this, the refactoring of the logical data model is conducted.

$$Cohesion(User)=\frac{\sum R(M)}{TM}=\frac{(1.00+1.00+0.67+0.67)}{4}=\frac{3.33}{4}=0.83 \tag{7}$$

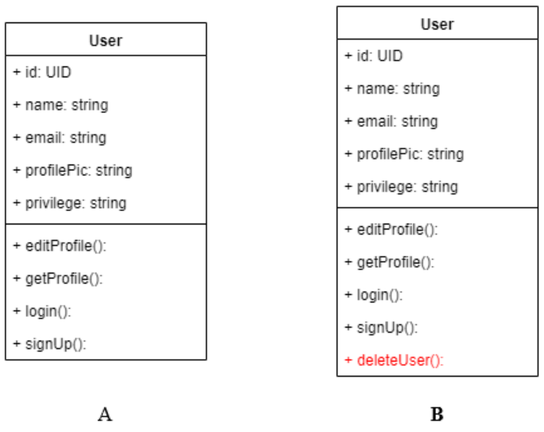
3.2.2 Refactoring the Logical Data Model



(a) Refactoring of Admin class



(b) Refactoring of Patriot class



(c) Refactoring of User class

Figure 4. The Refactoring Process of the Low-Cohesive Classes

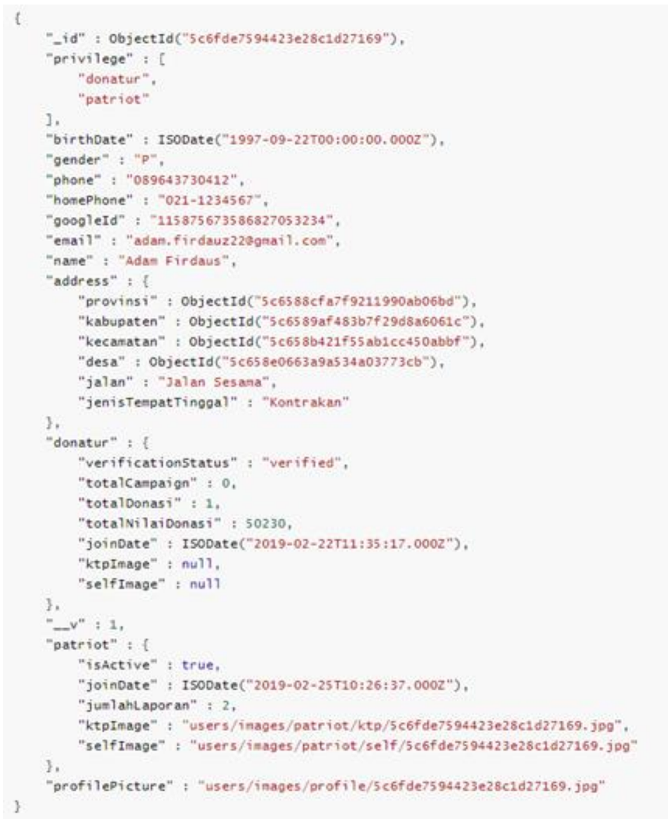
The refactor is done by moving the method from the class with the lowest cohesion value to another class with more relevance to the methods. In this case, we move several methods from the Admin class to Patriot, Donor, User, and Campaign class Figure 4. After the refactoring finished, we recalculate the cohesion of all updated class and the results show a significant increase in cohesion in several classes Table 3. Currently, there are 5 classes with high cohesiveness, 18 with normal cohesiveness, and 1 with low cohesiveness. The average value of cohesion also increases from 0.62 to 0.67 after the refactoring.

Table 3. Comparison of Cohesion Value Before and After the Refactoring Process

No	Class	Cohesion		No	Class	Cohesion	
		Original	Refactored			Original	Refactored
1	Achivement	1.00	1.00	12	Agregation	0.56	0.56
2	Drop Point	1.00	1.00	13	Village	0.55	0.55
3	Fund	1.00	1.00	14	SubDistrict	0.55	0.55
4	User*	0.83	0.87	15	District	0.55	0.55
5	UpdateCampaign	0.80	0.80	16	Province	0.55	0.55
6	Family	0.67	0.67	17	Goods	0.52	0.52
7	UserDetails	0.67	0.67	18	Report	0.52	0.52
8	Reward	0.64	0.64	19	Campaign	0.50	0.56
9	Address	0.63	0.63	20	Donor*	0.46	0.54
10	Donations	0.60	0.60	21	Patriot*	0.25	0.40
11	WebView	0.58	0.58	22	Admin*	0.15	1.00

3.3 Physical Data Model

We implement the physical data model in the MongoDB 4.0 using the assistance of RoboMongo3T as a graphical user interface. As MongoDB uses documents and collections to represent each entity, we create the representation of the logical data model as documents and collection that were written in JSON. This documents then formatted into Binary JSON (BSON) format and stored in the MongoDB database. The JSON format stores data in the form of an array of strings and allows an attribute to store an object in the form of an array of objects. Figure 5 depicts the JSON of the MongoDB collection to represent the user in the database.



```
{
    "_id" : ObjectId("5c6fde7594423e28c1d27169"),
    "privilege" : [
        "donatur",
        "patriot"
    ],
    "birthDate" : ISODate("1997-09-22T00:00:00.000Z"),
    "gender" : "P",
    "phone" : "089643730412",
    "homePhone" : "021-1234567",
    "googleId" : "115875673586827053234",
    "email" : "adam.firdauz22@gmail.com",
    "name" : "Adam Firdaus",
    "address" : {
        "provinsi" : ObjectId("5c6588cfa7f9211990ab06bd"),
        "kabupaten" : ObjectId("5c6589af483b7f29d8a6061c"),
        "kecamatan" : ObjectId("5c658b421f55ab1cc450abbf"),
        "desa" : ObjectId("5c658e0663a9a534a03773cb"),
        "jalan" : "Jalan Sesama",
        "jenisTempatTinggal" : "Kontrakan"
    },
    "donatur" : {
        "verificationStatus" : "verified",
        "totalCampaign" : 0,
        "totalDonasi" : 1,
        "totalNilaiDonasi" : 50230,
        "joinDate" : ISODate("2019-02-22T11:35:17.000Z"),
        "ktpImage" : null,
        "selfImage" : null
    },
    "__v" : 1,
    "patriot" : {
        "isActive" : true,
        "joinDate" : ISODate("2019-02-25T10:26:37.000Z"),
        "jumlahLaporan" : 2,
        "ktpImage" : "users/images/patriot/ktp/5c6fde7594423e28c1d27169.jpg",
        "selfImage" : "users/images/patriot/self/5c6fde7594423e28c1d27169.jpg"
    },
    "profilePicture" : "users/images/profile/5c6fde7594423e28c1d27169.jpg"
}
```

Figure 5. Example of the JSON Collection Describing a User in MongoDB Implementation

3.4 Analysis of Performance Improvement

Lastly, we test the performance of the database using the original and refactored logical data model to see whether the refactoring does improve the response time. We implement a simple backend system using Node.js to access the database. We install the backend system in the following environment: Ubuntu Server 18.0 as an operating system, Intel Core i7 3.4 GHz, RAM 8 GB, and HDD 1 terra byte. To do the API call, we used Postman 7.2.0. The trial was repeated 10 times for each method to get the response time of each method. The response time is measured using Postman 7.2.0 in a millisecond format. For the measurement purposes, we load the database with dummy data (> 1000 users) that represent the actual condition of the data.

For the results, all methods tested show a decrease in response time Table 4. The average response time is decreased from 11.0 ms to 7.99 ms. This is equal to 1.38 in speedup. In conclusion, the increase in class cohesion value was accompanied by a decrease in response time. However, in further research, we recommend testing the performance in a distributed environment similar to the condition in which the proposed early warning system will be installed.

Table 4. Comparison of Empirical Performance of the Database, Before and After Refactoring Process

No	Method	A: Original (ms)	B: Refactored (ms)	Speed-up (A/B)
1	assignPatriot()	12.5	8.3	1.51
2	deactivatePatriot()	10.2	8.0	1.28
3	verifPatriot()	10.3	7.6	1.36
4	rejectPatriot()	10.2	7.9	1.29
5	activatePatriot()	8.8	7.6	1.16
6	rejectDonor()	10.4	7.0	1.49
7	verifDonor()	11.5	7.5	1.53
8	rejectCampaign()	12.8	10.7	1.20
9	deleteUser()	12.3	7.3	1.68
	AVERAGE	11.0	7.99	1.38

3.5 Impact of Cohesion Improvement to NoSQL Database Performance

From the refactoring process, we observe that by increasing cohesion of the logical data model, the performance of the database increased by the factor of 1.38. Cohesion, along with coupling, is fundamental metrics to measure the quality of the object-oriented system by quantifying the modularity of the system [28]. This is in line with the nature of NoSQL database design which prefers de-normalized form contrary to the conventional relational database [29]. Although previous research already suggested UML for NoSQL database design [30], there is a lack of confirmation regarding the impact of cohesion and coupling to the performance of NoSQL database. We suggest further research to investigate this phenomenon using more robust testing to measure the correlation between cohesion of the logical data model with the performance of the database.

4. Conclusion

In conclusion, this research has developed and optimized the NoSQL database for food insecurity early warning system based on local community participation. We optimize the logical data model by using the cohesion measurement. After the refactoring, we improve the average cohesion from 0,62 to 0,67. The refactoring has a significant impact on the performance of the database. All methods tested show a decrease in response time. The average response time is decreased from 11.0 ms to 7.99 ms. This is equal to 1.38 in speedup. The increase in class cohesion value was accompanied by a decrease in response time. We suggest a further research to investigate this phenomena using more robust testing.

References

[1] A. P. Panatagama, “Analysis and Design of Patriot Pangan : Towards Electronic Participation and Initiative Platform to Help Reduce Food Insecurity in Indonesia,” in *Proceedings of IEEE Region 10 Humanitarian Technology Conference (R10-HTC) 2019*, 2019. https://doi.org/10.1109/R10-HTC47129.2019.9042477
[2] APJII, “Penetrasi & Profil Perilaku Pengguna Internet Indonesia Tahun 2018,” *Apjii*, Pp. 51, 2019.
[3] N. Ameya, P. Anil, and P. Dikshay, “Type of NOSQL databases and its comparison with relational databases.,” Int. J. Appl. Inf. Syst., Vol. 5, No. 4, Pp. 16–19, 2013.
[4] N. Leavitt, “Will NoSQL databases live up to their promise?,” *Computer (Long. Beach. Calif).*, Vol. 43, No. 2, Pp. 12–14, 2010. https://doi.org/10.1109/MC.2010.58
[5] R. Cattell, “Scalable SQL and NoSQL data stores,” *SIGMOD Rec.*, Vol. 39, No. 4, Pp. 12–27, 2010, https://doi.org/10.1145/1978915.1978919.
[6] J. Han, E. Haihong, G. Le, and J. Du, “Survey on NoSQL database,” *Proc. - 2011 6th Int. Conf. Pervasive Comput. Appl. ICPCA 2011*, Pp. 363–366, 2011, https://doi.org/10.1109/ICPCA.2011.6106531.
[7] Y. Li and S. Manoharan, “A performance comparison of SQL and NoSQL databases,” *IEEE Pacific RIM Conf. Commun. Comput. Signal Process. - Proc.*, No. August 2013, Pp. 15–19, 2013, https://doi.org/10.1109/PACRIM.2013.6625441.

[8] C. Győrödi, R. Gyorodi, G. Pecherle, and A. Olah, “A Comparative Study: MongoDB vs. MySQL,” No. June, 2015, https://doi.org/10.13140/RG.2.1.1226.7685.
[9] A. Boicea, F. Radulescu, and L. I. Agapin, “MongoDB vs Oracle - Database comparison,” *Proc. - 3rd Int. Conf. Emerg. Intell. Data Web Technol. EIDWT 2012*, No. September 2012, Pp. 330–335, 2012, https://doi.org/10.1109/EIDWT.2012.32.
[10] Z. Parker, S. Poe, and S. V Vrbsky, “Comparing NoSQL MongoDB to an SQL DB,” 2013. https://doi.org/10.1145/2498328.2500047
[11] mongodb.com. Last access: 02 Feb 2020.
[12] db-engines.com. Last access: 02 Feb 2020.
[13] K. Chodorow, *MongoDB: The Definitive Guide.*, Third Edit. Sebastopol (CA): O’Reilly, 2019.
[14] M. M. Patil, A. Hanni, C. H. Tejeshwar, and P. Patil, “A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retrieval operations using a web/android application to explore load balancing-Sharding in MongoDB and its advantages,” in *Proceedings of the International Conference on IoT in Social, Mobile, Analytics and Cloud, I-SMAC 2017*, Pp. 325–330, 2017, https://doi.org/10.1109/I-SMAC.2017.8058365.
[15] S. Kanoje, V. Powar, and D. Mukhopadhyay, “Using MongoDB for social networking website deciphering the pros and cons,” in *ICIIECS 2015 - 2015 IEEE International Conference on Innovations in Information, Embedded and Communication Systems*, 2015, https://doi.org/10.1109/ICIIECS.2015.7192924.
[16] Z. Wei-Ping, L. Ming-Xin, and C. Huan, “Using MongoDB to implement textbook management system instead of MySQL,” in *2011 IEEE 3rd International Conference on Communication Software and Networks, ICCSN 2011*, Pp. 303–305, 2011. https://doi.org/10.1109/ICCSN.2011.6013720.
[17] Y. Widyani, H. Laksmiwati, and E. D. Bangun, “Mapping spatio-temporal disaster data into MongoDB,” in *Proceedings of 2016 International Conference on Data and Software Engineering, ICoDSE 2016*, 2017, https://doi.org/10.1109/ICODSE.2016.7936157.
[18] N. Yilmaz, O. Alatli, B. Ciloglugil, and R. C. Erdur, “Evaluation of storage and query performance of sensor based Internet of Things data with MongoDB,” in *2018 International Conference on Artificial Intelligence and Data Processing, IDAP 2018*, 2019, https://doi.org/10.1109/IDAP.2018.8620837.
[19] M. G. Jung, S. A. Youn, J. Bae, and Y. L. Choi, “A study on data input and output performance comparison of MongoDB and PostgreSQL in the big data environment,” in *Proceedings - 8th International Conference on Database Theory and Application, DTA 2015*, Pp. 14–17, 2016, https://doi.org/10.1109/DTA.2015.14.
[20] S. Hoberman, *Data Modeling for MongoDB: Building Well-Designed and Supportable MongoDB Database*, First Edit. New Jersey (US): Technics Publication, 2014.
[21] Wilson da Rocha França, *MongoDB Data Modeling: Focus on Data Usage and Better Design Schemas with the Help of MongoDB*, First Edit. Birmingham (UK): Packt Publishing, 2015.
[22] M. Paixao, M. Harman, Y. Zhang, and Y. Yu, “An Empirical Study of Cohesion and Coupling : Balancing Optimisation and Disruption,” No. C, Pp. 1–21, 2017. https://doi.org/10.1109/TEVC.2017.2691281.
[23] J. Al Dallal and L. C. Briand, “An object-oriented high-level design-based class cohesion metric,” *Inf. Softw. Technol.*, Vol. 52, No. 1, Pp. 1346–1361, 2010, https://doi.org/10.1016/j.infsof.2010.08.006.
[24] A. Silberschatz, H. F. Korth, and S. S, *Database System Concepts*, Sixth Edit. New York (US): McGraw-Hill Education, 2010.
[25] J. Al Dallal and L. C. Briand, “A precise method-method interaction-based cohesion metric for object-oriented classes,” *ACM Transactions on Software Engineering and Methodology*, Vol. 21, No. 2. Pp. 1–34, 01-Mar-2012, https://doi.org/10.1145/2089116.2089118.
[26] A. Kumar and S. Kaur Khalsa, “Determine Cohesion And Coupling For Class Diagram Through Slicing Techniques,” IJACE, Vol. 4, No. 1, Pp. 19–24, 2012.
[27] I. Baig, “Measuring Cohesion and Coupling of Object-Oriented Systems-Derivation and Mutual Study of Cohesion and Coupling,” Blekinge Institute of Technology, Karlskrona, 2004.
[28] S. Tiwari and S. Singh Rathore, “Coupling and Cohesion Metrics for Object-Oriented Software: A Systematic Mapping Study,” 2018, https://doi.org/10.1145/3172871.3172878.
[29] E. M. Kuszera, L. M. Peres, M. Didonet, and D. Fabro, “Toward RDB to NoSQL: Transforming Data with Metamorfose Framework,” 2019, https://doi.org/ 10.1145/3297280.3299734.
[30] K. Shin, C. Hwang, H. Jung, and H. Jung, “NoSQL Database Design Using UML Conceptual Data Model Based on Peter Chen’s Framework,” 2017.