3. ARCHITECTURE DESIGN

3.1. Research Methodology

The methodology employed for this research work is Object-Oriented Analysis and Design (OOAD) Methodology. Object-oriented programming (OOP) is a programming paradigm that uses "objects" and their interactions to design applications and computer programs. It has been touted as the great advance in software engineering. It promises to reduce development time, reduce the time and resources required to maintain existing applications, increase code reuse, and provide a competitive advantage to organizations that use it.

3.2. Analysis of Existing System

Graph query processing is essential for graph analytics, but can be very time-consuming as it entails. Eeda (2017) proposed a GraphQL-based UI Architecture which was integrated into the existing Bluemix system to boost an application's performance. The key performance improvement of the real-time dashboards was explored. The performance of the existing web applications was boosted by fetching only the client-specific data faster. To avoid fetching more or less of the needed data, the client strictly dictates to the server the kind and amount of data to fetch. The architecture was created by developing a GraphQL layer and placing it above already existing data sources at the backend the system was using. These data sources could be a relational database, NOSQL database, REST endpoint, SOAP endpoint, message bus or a combination of any or all of these data sources. The architecture incorporates GraphQl to brings in all the advantages of GraphQL which includes returning the result of a query through a single endpoint, making fast and simple queries. Making a simple query improves the stability of a query process. When a client issues request to a GraphQL server, the request is splitted into multiple forms on reaching the GraphQL layer as proposed in the architecture. They designed and stored the states of all the components that were to be rendered on the UI using Redux and the different parts of React were binded with its value as shown in figure 3. A caching mechanism and subscription channel were also incorporated in their design.

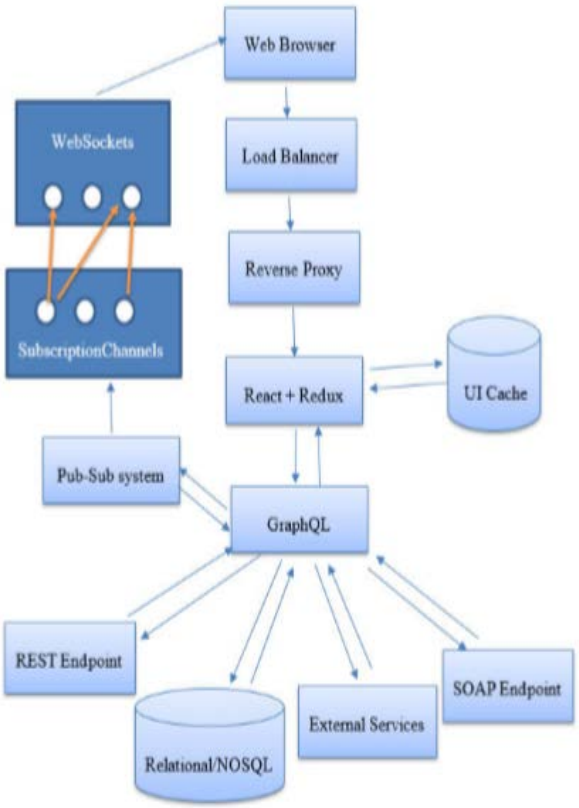

Figure 3: Existing Architecture of Rendering real–time dashboards using a GraphQL–based UI
(Source: Eeda, Naresh, 2017)

Contrary to the old method of loading all the data at once, data was gradually loaded onto the user interface without waiting for overall completion of the execution of other queries. They binded parts of the User Interface to the exact queries and loaded them immediately response was gotten. This allowed them to asynchronously load the data onto the UI. The fact that early component skeleton was kept in the redux, helped them in loading the user interface elements without the real data to be rendered as soon as the request was received. This improved the user experience by displaying the web page almost instantaneously.

3.2.1. Disadvantages of the Existing System

The following disadvantages of the Existing System are:
ii) Inadequate modeling of highly interconnected data: The efficient modeling of data was lacking in the Existing System.
ii) Complexities involved in writing graph queries: This is also a major issue associated with the existing system which arises from how efficiently we can find an unknown graph using distance or shortest path queries between its vertices

3.3 Analysis of the Proposed System

The GrapQL Model is deployed in a political party database and a graph database is built for a party administration system. In the proposed system, records of party members are collected and stored in a database as shown in figure 4. The various local governments regularly generate data of membership and various financial and non-financial activities which are stored in local databases in their offices. The states also have party offices which equally keep record of party officials and party activities at the state level. The parties also

have a National Secretariat which equally has databases of members, officials, contestants and party officials. There are equally financial records at each level of the data store at the local government and states. The databases of the party members are kept separate from each other in many states. This leads to data redundancy as the same data are repeated in the relations. Accessing the databases requires generation of multiple queries. In other states where a relational databases are used, challenges are encountered as the records get larger.

Querying a large dataset is tedious and time consuming. The fact is that it involves multiple joins of the table leading to complex query formation. This has negatively affected the administration of political party system as query processing seems to be a very difficult task. Expanding a relational model is difficult as a result of strict schema maintained by the relational model. One of the relational model's design motives was to achieve a fast row-by-row access (Codd, 1970).Although relational model manages related records, many joins operations involve complex queries as many attributes of different tables are considered. Resolving relationships in relational model involves creating multiple joins with the tables. In working with relational models, foreign key constraints should be considered when retrieving relationships and this constitutes to the challenges faced by relational model. The cost and time of the query operation is on the increase too.
In the proposed system, the different records from the wards, local governments and states are collected and relationships among them are created. These records are stored in the graph database for easy access and query operation. Graph database is preferred because of its ability to handle big volume of data.

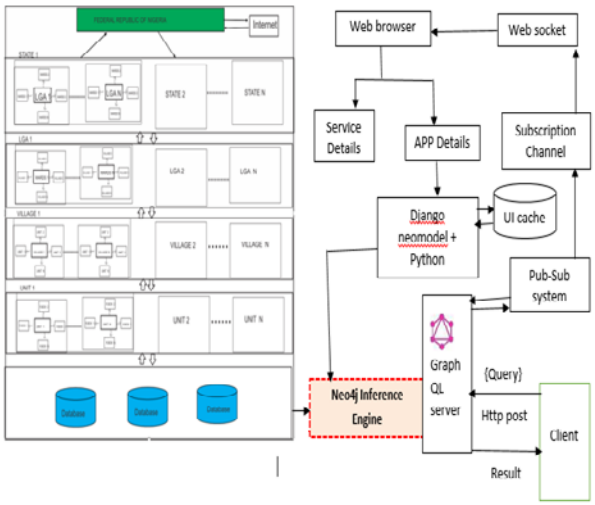

Figure 4: Proposed System Architecture of an improved Graph Query Language Model

Graph database overcomes some of the limitations of related databases. When it comes to a dataset that associates, graph database is faster. Graph database connects straight to the form of object-oriented programs. Normally, they are well suited to datasets that are very big as they do not not make use of computations involving joins. Join operations are very expensive. As they depend more on flexible schema, they are promoted as convenient to handle adhoc and developing schema.
A database is an important aspect of a software application. Apart from the fact that it stores information, it positively affect the general accomplishment of a software. In spite of the numerous benefits of relational database, however, they don't perform well with continuous increase in the volume of related data. So, selecting a database suitable for political party system is crucial. Using a graph database for a political party administration will make way for easy and fast query of records of party members.

The proposed architecture uses GraphQL to query the database of a political party system. The database has to be structured in form of graph for fast query. Graph databases perform well when the databases gets larger compared to relational databases. As the complexity in data and value in relationships increases, the ability of relational databases to address the data requirements decreases and use of graph database increases, which leads to the adoption of improved GraphQL system for a graph database in the proposed system. This will enable an easy and most private way to communicate with unit members in the whole state and also most promising channel of communication within party executives.

3.3.2 Advantages of the Proposed System

The following advantages of the proposed system are:

i) **Easy Query:** GraphQl query is easy to manage because of its flexibility. API does not require too much maintenance or modification.
ii) **The ability to make fast query:** The client can make only one request to retrieve only information they need. A backend server only needs to fetch and prepare what is being asked for, so the entire response can be delivered in a single network delivery.
iii) **Flexible Graph-Query Platform in the Neo4j Environment:** Neo4j uses property graphs to extract added value of data of any company with great performance and in an agile, flexible and scalable way. In terms of performance Graph databases such as Neo4j perform better than relational (SQL) and non-relational (NoSQL) databases.

4. IMPLEMENTATION AND SAMPLE RESULTS

To implement the improved GraphQl model for political party distributed database administration, detailed analysis were extensively done on the existing system to identify the major drawbacks faced in the storage method of political party information and to compare and contrast it with the new system. A graph database (Neo4j) is used in the new system instead of relational database system. Graph database helped us to model, query and expand data in a faster and more intuitive way when compared with a traditional SQL approach. For easy accessibility, the Neo4j database was hosted on the cloud. As the party member registers as shown in Figure 5, the information is store in the cloud.

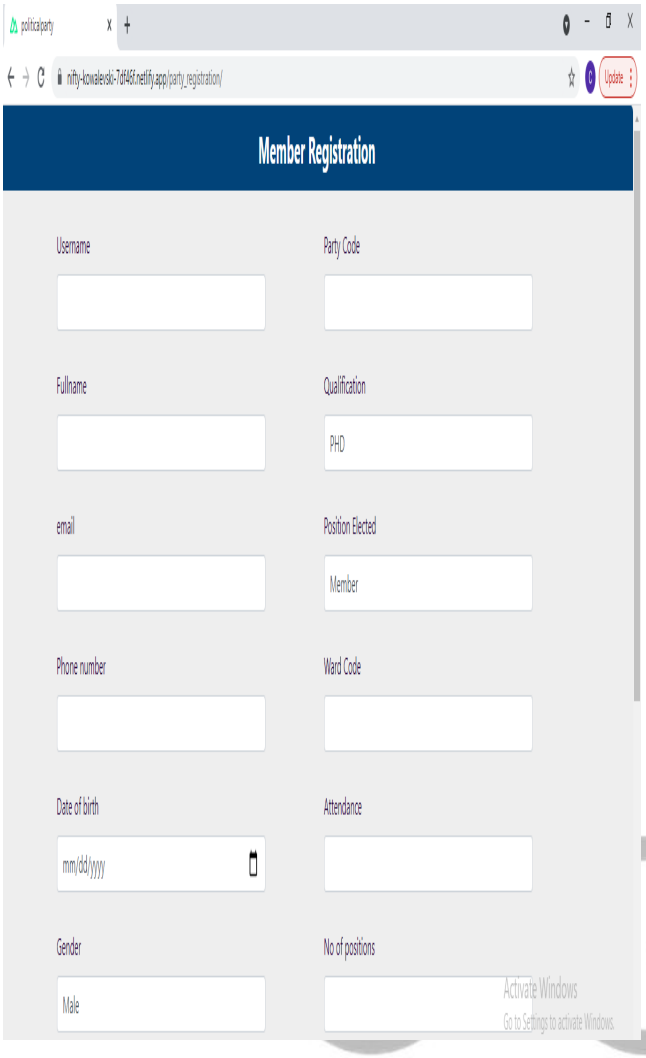

Figure 5: Registration Form of the Party members

Neo4j is a non- relational graph database that is optimized for managing relationships. Neo4J database can help in building high performance and scalable applications that use large volumes of connected data. To add neo4j-based functionality to the proposed system, Django-neomodel plugin was used. Django neomodel is an Object Graph Mapper (OGM) for the neo4j graph database built on the awesome neo4j_driver. This module allowed us to use the neo4j graph database with Django using neomodel. With this, data displayed will be obtained from an API.

To handle the relationship between the database, the python models and other aspects of web development, an object mapper, Django was incorporated. A Django module allowed us to use the Neo4j database with Django using neomodel. Django made it possible to create the entire site and database backend, along with the plugins used. Django is based on the idea of models. A model is a class of objects directly linked to objects in a database. To connect to the Neo4j database, Bolt protocol URL was used. Bolt connected only to the server with the IP specified. It will not route anywhere else. All queries over this protocol would go only to this machine, whether they're read or write queries. GraphQL was used as an abstraction layer to hide the database internals. Request sent by Client to GraphQL server would always be validated and executed by collecting the information from the Neo4j database. The data should be requested from the API. To create some level of abstraction, the results are displayed in tabular form. Figure 6 shows sample results of the query.

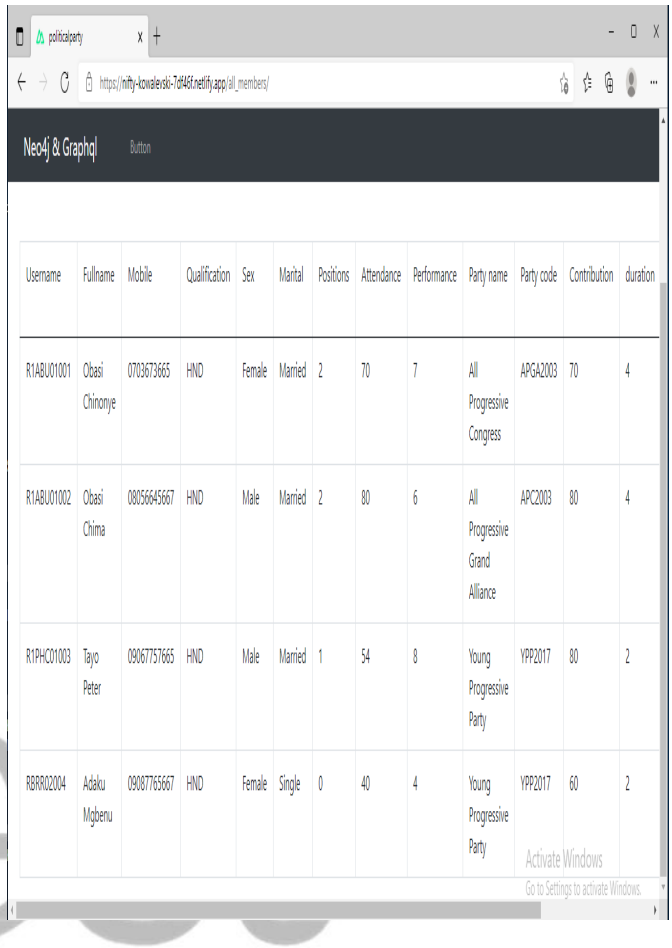

Figure 6: Sample of GraphQL Query Results of all Party Members

5. CONCLUSION

Relational Database is a mature way of storing records, the building block is table. Graph database is also another storage mechanism. The building block is graph. There is gradual increase in dataset as the day goes by. Hence the need for a database system that can accommodate the massive increase in data. As the amount of related data increases, the ability of a relational database to handle such data drops. Relational database is not efficient in handling highly interconnected data. Graph database on the hand displays great performance in handling highly interconnected data. GraphQL, which provides some level of abstraction is very efficient in querying graph database. Although it is not a graph database query language but an Application Program Language. It delivers query result within the shortest possible time. Querying Graph database with GraphQL helps in generating faster and easy query. The improved GraphQL model for a political party distributed database administration uses of a Neo4j database as a graph database . Based on the findings of the research work, the new system generates better results. Querying the database is faster and easy as GraphQL returns the result of a query through a single endpoint unlike its alternative REST that returns the result of a query through multiple endpoints. There is no under fetching and over fetching of data in graphql query.

6. ACKNOWLEDGMENTS

Our thanks go to the Almighty God for His grace and sustenance. We are also grateful to the lecturers in Computer Science Department, University of Port Harcourt, Choba, Rivers State, Nigeria for their inputs and support during the course of this research work.

7. REFERENCES

[1] Bhavani T. and Ammiel K. (2010). Secure Query Processing in Distributed Database Management

[2] Buna, S. (2017). REST APIs are REST-in-Peace APIs. Long Live GraphQL URL: https://medium.freecodecamp .org/rest-apis-are-rest-in-peace-apis-long-live-graphql

[3] Codd E.F. (1970). A relational model of data for large shared data banks. Communications of the ACM. 13(6). 377-387

[4] Eeda, Naresh (2017) "Rendering real-time dashboards using a GraphQL-based UI Architecture".Electronic Thesis and Dissertation Repository. 5136.

[5] Idowu, S.A, Maitanmi S.O. (2014) "Transactions-Distributed Database Systems: Issues and Challenges" International Journal of Advances in Computer Science and Communication Engineering (IJACSCE) 2(I), ISSN 2347-6788,24-26, Ilisan Remo, Ogun State, Nigeria.

[6] Jindal A, Madden S (2014) .GRAPHiQL: A graph intuitive query language for relational databases. In:2014 IEEE international conference on big data, big data, Washington, DC, USA, October 27–30,441–450

[7] Jing W., N. Nikos, T. Peter (2017), GraphCache: A Caching System for Graph Queries, Open Proceedings, 10.5441/002/edbt.2017.03, 13 – 24

[8] Njoku Donatus O., Nwokorie Chioma E., Madu Fortunatus U(2016). An Enhanced Query Processing Algorithmfor Distributed Database Systems,International Journal of Scientific & Engineering Research, 7(10)

[9] Smita A. and A. Patel (2016), A Study on Graph Storage Database of NoSQL, *International* Journal of Soft Computing, AI and Applications (IJSCAI), 5(1), 33 – 39

[10] Srinath S. (2016), Query Processing Issues in Data Ware Houses,Research gate Publications, https://www.researchgate.net/publication/2618638,1–13

[11] Stubailo, S. 2017. GraphQL vs. REST. URL: https://dev-blog.apollodata.com/graphql-vs□rest-5d425123e34b Accessed 22 November 2017

[12] Sturgeon, P. 2017. GraphQL vs REST: Overview. URL: https://philstur□geon.uk/api/2017/01/24/graphql-vs-rest-overview/ Accessed 20 November 2017

[13] Thuraisingham M. (2010). Secure Query Processing in Intelligent Database Management Systems.The MITRE Corporation, Burlington Road, Bedford.