Fall 15

HOCHSCHULE SRH HEIDELBERG

Personalized Nutritionist

Advanced Information Systems (M-2372-1854)

Prof. Dr. Barbara Sprick
Lec. Mr. Ajinkya Prabhune

Team:

Key-Value subgroup:

Sarah Al-Turki

Renu Thomas

Abdulla Alfadhel

Document Store subgroup:

Paras Dedhia

Tanmay Nath

Charan Ravikumar

Ye Sun

Graph subgroup:

Reuben Borrison

Nagashree Tadapatri

Ankush Charan

Manjunath Satyamurthy

Table of Contents

List of Figures

How to read this document

This document is a collaborative work that aims to present a problem statement and propose a resolution model that is implemented based on Big Data solutions.

The first part of this document is a walkthrough of the requirement, and highlights the necessity to develop a solution to address it.

The introduction will also cover the scope, user requirements, and user stories.

Translation of the user requirements to system requirements as well as addressing their functional and non-functional needs is covered by the second part, the architecture of the system.

The third part, Databases introduces the three NoSql solutions and their relevance to the project.

The fourth, Fifth, and Sixth parts are distinct documentation of the role of each NoSql Database. Each part will cover the role of the DB to deliver the solution in accordance with the problem statement and system analysis conducted at the first and second part. Each of the three parts will cover primarily the Data Model and implemented functionalities respectively. Integration of the three is briefly mentioned in each part and the big picture of bringing all together is demonstrated at the seventh part. The terms “system”, “application”, and “solution” are interchangeably used to refer to Personalized Nutritionist. Services, functions, and use-cases are requirements derived from the problem statement and user stories and are equivalent in meaning.

1. INTRODUCTION

"Tell me what you eat and I will tell you what you are" - Brillat-Savarin

Through intense research, powerful consultations and various negotiations, it has come to realization that one of the most wide struggles in the civilization of our 21st century is the struggle of self-educating oneself about the ideal diet and staying on track of a challenging healthy nutrition plan.

As characteristics, properties and personal choices differ for every individual, there is no one single diet plan that fit us all. Finding customized diet plans that fit our personal needs and our food choices are overpriced and time consuming.
On the other hand- and in the traditional sense- following a diet has been a challenging practice for countless number of people. The struggle starts from finding appointments with the experienced nutritionist, moving on to following harsh food choices, the battle of sticking to it and ending up with a breakdown in many cases. This battle has been obsessing numerous people with different age groups, different backgrounds, both genders equally.

People need advisors within reach to tell them what food serves their ultimate goals of living better, longer and healthier. They need a smart system that contains all the necessary information and knows what's best for them.

Thanks to technology, almost all daily activities are conducted with high reliance on expert systems that guide users to their ultimate destination. The destination (objective) surely differs from one system to another. Technology has acquired a huge trust from people in assisting and shaping smart decisions. Nowadays, uncountable number of people have been given access to great minds and expertise by a click of a button. The Personalized Nutritionist system is the most satisfying advisor of individual's likes, dislikes, allergies, and food restrictions, which shall allow in conveying the user to the best-fit, flexible, more efficient and effective diet. Personalized Nutritionist is about reading and relating to the user individuality and helping them towards their goal. This will be accomplished by having them on a healthy sustainable diet. The system shall evolve this process to a more convenient way saving time, effort, and money.

1.1Purpose

The Personalized Nutritionist System is not just about providing users with nutritional facts and delivering the number of calories in each food item. It is not just about suggesting healthy food for users and it's definitely not all about prescribing the most recent diets. It is an interactive system that understands each and every user and cooperates differently according to different cases. It essentially acts as the solid foundation of living a healthy, strong and determined lifestyle. Ultimately, Our goal of the system is to help each and every user towards his/her goal.

1.2 Scope

Dietitians and nutritionists are experts in food and nutrition. Their job is to guide people on what to eat, what not to eat, when to eat and when not to eat in order to lead them to a healthy lifestyle or help them achieve a specific health-related goal. Some dietitians and nutritionists provide customized diet plans for specific individuals. For example, a nutritionist might educate a patient with high blood pressure on how to reduce salt in a meal preparation; while, others develop particular nutritional plans that target segments where it's best applicable towards common goal. For instance, a dietitian tends to plan diets with less fat and sugar to help patients wanting to lose weight accomplish their goal.

The Personalized Nutritionist System acts exactly as a Dietitian or Nutritionist that understands different user needs. The system design, implements and integrates three different databases, which are, Key-Value store, Document store and Graph store. However, there will be neither an application layer nor a Graphical user interface considered part of this project's scope.

1.3 Requirements

Our aim is to enable personalized nutrition tactics for better handling of a healthy and determined lifestyle. This will be encouraged by providing the users with informed best fit food suggestions and a collection of diet programs as well as offering a distinctive service for tracking their daily meals and snacks.

Personalized Nutritionist is a repository of three main data stores:

1- User specific data containing static values such as height, weight and daily activities (food intake).
2- Nutrition facts, which contains a full pool of nutrition facts in a detailed format.
3- Diet plans- comprehensive, detailed, generic and particular diet plans.

The System shall collect and store the following information:

- User details
- User day-to-day activities- particularly food intakes.
- Primitive health descriptions of the user including allergies and restrictions.
- Customized food items.
- Customized diet plans.
- Suggestive-adaptive diet plans.
- Full prebuilt repository of nutritional content of food items.

With all above being collected, stored, and accessed; the application should be able to provide the following functions: -

- Aggregate and show total daily calorie intakes.
- Break down daily food intakes based on nutritional values.
- Show nutrition facts of common food items.
- Allow users to add new food items.
- Allow users to query their historical activities.
- Allow users to compare their nutrient (example: vitamin) intake with the suggested amount.
- Allow users to specify goal of either “weight loss”, “mass gain”, “maintain weight” or “be healthy”.
- Allow users to search for diets.
- Suggest best-fit diets for users based on matching parameters of diet to their parameters and goals as well as success rate of users of similar properties following these diets.
- Suggest food based on users likes and in accordance with their diet highlights to allow diet sustainability.

Personalized Nutritionist’s success is not in inventing diet plans for users but rather in having a platform that best match people to the right plan. Nevertheless, Personalized Nutritionist’s generic and non-food-restrictive diets are designed to be templates rather than static to allow following, convenient and healthy lifestyle choices.

1.4 User stories

The user stories in this document will explain the Personalized Nutritionist System features in the language of the user. They are written using short and simple description of what a user might want or need. Below are four examples of essential cases that the system could expect.

User Story 1

Sophie is 30-year-old, suffering from high blood pressure and diabetes. She struggles a lot finding flavorsome meals that contain neither sugar nor salt. She is looking for daily meals that fulfill her desires without out breaking her restrictions.

The personalized Nutritionist System offers users the ability to customize the kind of diet they want to follow. In the case of Sophie, she can find the chronic diseases she suffers from in the list of restrictions and by simply selecting them, the system will automatically start prescribing her healthy meals on a day to day bases. Those meals are specified for people with similar circumstances only.

User Story 2

Jan is a 25-year-old who weighs 95kg and 1.8m tall. His blood type is A+ and wants to lose10kg. He is willing to follow any diet plan as long as it's going to make him lose fat without losing any money.

In the personalized Nutritionist System, diets are not only sorted based on users goals of losing or gaining weight. They are not just sorted based on users likes and dislikes. The system pays more attention to the smallest details as well, and since Jan has blood type A+ and wants to lose weight, the diet that will be prescribed for him will include less meat because people with type A blood are known to digest meat in a very slow manner [1] causing it to be stored as fat in the body. The prescribed diet in this case will be specified for the people having same blood type and seek losing weight.

User Story 3

Ben is a healthy athletic who pays lots of attention to the food he eats. He must identify the food value and the number of calories in everything he eats. However sometimes he finds difficulties in finding the information he needs since not every food item is labeled with its nutritional value.

---

[1] Eat Right for Your Type: The Individualized Diet Solution to Staying Healthy, Living Longer & Achieving Your Ideal

Personalized Nutritionist offers users all the needed information about the number of calories and nutritional values such as the amount of protein, minerals, fat and vitamins in each food item. Such information should be available for more than 8000 food items. Ben can simply browse the list of all foods or just type the name of a particular food item to get all the information needed about it.

User Story 4

Elias has been following diets suggested from the personalized Nutritionist System for 2 years and he has witnessed lots of progress in his body shape and life style. However he is interested in viewing the history of his progress month by month since the day he started.

Personalized Nutritionist offers users the ability to view the history of all their logged in data. Therefore, if Elias had been logging in his weight information frequently, then a record of all his weight history will appear to him.

1.5 Use-case Diagram

Use case diagrams are used to represent the dynamic aspect of a system, which is the interaction of the user with the system. The diagram below identifies the external and internal factors influencing the personalized Nutritionist system as well as the relationships between the users and the system.

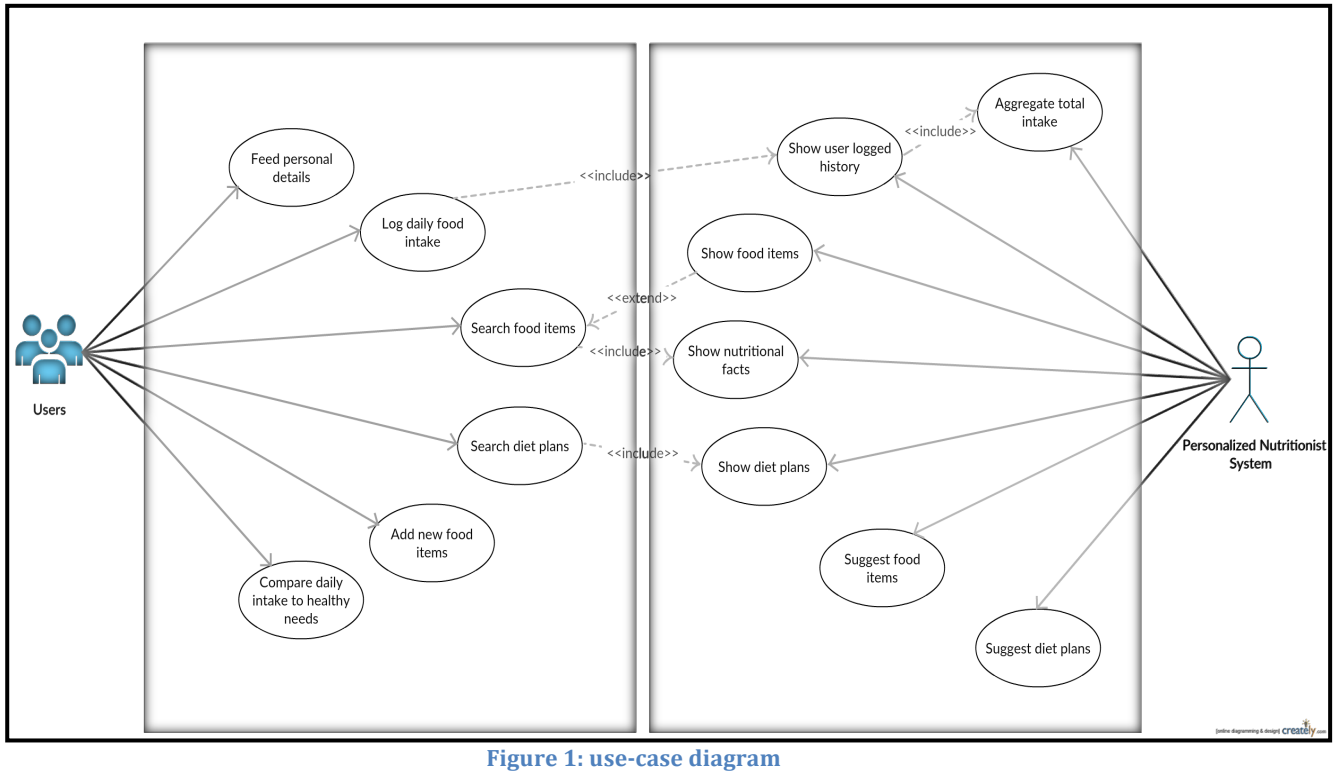


Figure 1: use-case diagram

2. Architecture

Having realized the problem statement illustrated in the first part, where this solution shall come in to fill the gap as a companion, within fingertip reach, personal nutritionist; Challenge lays itself for the application to meet functional and –as essential- non-functional requirements for it to be a true business enabler that successfully seizes to meet its vision. With the functional requirements being well illustrated in the first part of the document as services, use-cases, and user-stories; then comes the architectural part to highlight the non-functional requirements for the overall application and the services individually. Realizing which ensures the application comes out as intended by delivering the required services in the foresighted manner. In addition, it shall help draw the final architectural picture in which the best infrastructural decisions for the application are being met.

2.1 Semantic Breakdown

Looking at the use-cases and functions of the application as illustrated in the first part, services can be in link with one or more of three categories.

1. **User Data & Services**: A broad term that includes any services or data evolving around user identifying information storage as well as read and write functions. That extends to include physical health- and life style information in addition to requirements considered to be in relation of user's activities.
2. **Nutrition Fact Data & Services:** Functions and data in relation to food, food items, nutritional values, and food servings.
3. **Diet Plans Data & Services:** Functions that meant to draw relations between food and users as well as storage of data regarded to be diet plans in the traditional sense.

However, it is also essential to realize that the functional requirements are from the perspective of the user and therefore can be in link with more than one concept. In addition, these functions can also be classified as read, write, or analyze function; which shall help identify the non-functional requirements for them. Moreover, frequency and volume of the data being read, written, or analyzed shall be foresighted.

2.2 Metrics Descriptors

Each function, in addition to which semantic category it falls into, shall be classified in accordance to various important metrics that are important in naming non-functional requirements of use cases. Metrics shall classify functions based on their type, frequency, and data atomicity. A walkthrough of the named metrics and their measures is provided below.

- **Function Type**

  Functions are classified as to be read, write or analyze functions based on the main purpose of the function, although there could exist a slight overlapping.

  - Read: Read services are those, which regarded to fetch stored information either directly by the user or through another service. Functions could do both read and write. However, if the write is a result of aggregation, summation, or other calculation such as average; then, services of this kind are still regarded to be a read function as the write is for pure tuning purpose.
  - Write: Services, which are being fed with new and previously unknown data to the system, data that in no way can internally be calculated are regarded to be write functions.
  - Analyze: Services that do not fall in the aforementioned categories are most likely regarded to be of analyze type. Analyze services are functionalities in which artificial intelligence, instructional behavior, and processing is needed in addition to present data. Analyze services result in reasoning, machine learning, pattern and relationship recognition, and usually occur as a result of multi-dimensional inputs or accumulative multistep processing in which latter steps need the output of former ones.

- **Frequency**

  Frequency is a measure that takes into account how frequent a service shall be used, invoked, or triggered. Frequency doesn't take into account the volume of data being read or written but rather the function as single unit frequency. Frequencies are not always feasible to predict as a function of time, thus we apply measures in which how triggering occur is considered.

- On-Demand: On-Demand frequency is regarded for functions that are triggered in a non-systematic way that could involve for example looping. Functions that are regarded to be on-demand are not seen to be repeatable in a patterned time manner.
- Ad-hoc: Ad-hoc classified are the functions that are invoked in patterned manner, daily, weekly, hourly, or monthly. Most functions that are seen as batch functions in which bulk data are processed in a scheduled manner are viewed to be of an ad-hoc frequency.
- Real-Time: Functions and services that repeat in a systematic manner that for example involves looping and require fast response, processing, or acknowledgement are characterized to be real time functions. In addition, functions, which are predicted, to be invoked heavily due to high population beneficiaries are characterized to be Real-Time functions.

- **Data Volume**
  - Medium: Data are less likely to repeat or more seen to be unique and describe non-substantial objects are seen as with medium volume.
  - Large: Data is measured as large are when they seen to be repeated in patterned manner or are a result of services measured to be an ad-hoc frequent.
  - Substantial: Almost exclusively, any data that are result of real-time frequent services is seen to be substantial and enormous.

- **Data Atomicity**
  - Atomic: Data that describe one or more objects in a non-compound format and are meaningful of its own without linking or referencing other values are considered to be atomic data. Atomic data can still be referenced, described within themselves by descriptive information –metadata- which are in relation with the values themselves not the object being described.
  - Structured: On the contrary, values, which can only be meaningful descriptor of objects if coexisting or compound with other values, are referred to as structured data.

2.3 Metrics Matrix

Now that the metrics and their measures are well defined in the section above, application of measures to the four metrics is more agreeably non –subjective task.

Table 1 below classifies all functional requirements according to metric measures values.

Functional Requirement	Semantic Category[2]	Metrics[3]			
		Type	Frequency	Volume	Data Atomicity
Feed User Data and activity	User Data	Write	Real Time	Substantial	Atomic
Aggregate Daily Calorie Intake	User Data / Nutrition Facts	Read	Ad-hoc	Large	Atomic
Break Down daily intake to nutritional values	User Data / Nutrition Facts	Read	Ad-hoc	Large	Atomic
Show nutrition fact	Nutrition Facts	Read	On-Demand	Large	Structured
Add new Food Item nutrition fact	Nutrition Facts	Write	On-Demand	Medium	Structured
Show user history of activities	User Data	Read	On-Demand	Substantial	Atomic
Compare Food intake to healthy need	Nutrition Facts / User Data	Analyze	Ad-hoc	Substantial	Structured

[2] For more information see semantic breakdown, page 13.

[3] For more information see Metrics Descrpitors page 14.

Feed User Goal	User Data	Write	Real Time	Large	Atomic
Search Diets	Diet Plans	Analyze	On-Demand	Large	Structured
Match Diets	Diet Plans	Analyze	On-Demand	Large	Structured
Create Diet	Diet Plans	Write	On-Demand	Medium	Structured
Suggest Food intake	Diet Plans User Data Nutrition Facts	Analyze	Real-Time	Substantial	Structured

Table 1: Metrics Matrix

Table-1 above states each function in the first column, and semantic category it falls under. (See **2.1 Semantic Breakdown**, page 13 for more information).

At the last four columns on the most left appear all metrics where measures thereby are applied for each service individually. Full definitions of metrics and measure values are found in Metrics Descriptors, page 14.

2.4 Four-Dimensional metric measures and pattern recognition

As services are being classified according to the 4-dimensional metric measures (in Metrics Descriptors, page 14), and where each service belongs to a semantic category (User data, Food Data, Diet Data), patterns can be found among services belonging to the same semantic category to have similar metric measures.

In Table 2 below appear the 4-dimensional metric measures with yellow representing function type, dark blue representing frequency, green representing data volume, and pink representing data atomicity. The values filling the table in the figure are correspondent to Table 1: Metrics Matrix. Table 2 below shows that services in relation with User Data are five in six times atomic in the pink dimension, equally divided between being substantial and of medium/large size as well as rarely dealt with analytical services appears in the yellow dimension.

On the other hand, recognizing the patterns of the services lays the groundwork for identifying the division of the use-cases, services, and functions; it rationalizes the reasons behind the choices as coming in new sub sections of the architecture part.

		User Data	Nutrition Facts	Diet Plans	Write	Read	Analyze	Real Time	Ad-hoc	On-Demand	Medium/Large	Substantial	Atomic	Structured
	Functions Total	6	5	4	4	4	4	2	3	6	8	4	5	7
	Write	2	1	1	X	X	X	2	0	2	3	1	2	2
	Read	3	3	0	X	X	X	0	2	2	3	1	3	1
	Analyze	1	1	3	X	X	X	1	1	2	2	2	0	4
	Real Time	2	0	1	2	0	1	X	X	X	1	2	2	1
	Ad-hoc	4	3	0	0	2	1	X	X	X	2	1	2	1
	On-Demand	1	2	3	2	3	2	X	X	X	5	1	1	5
	Medium/Large	3	4	3	3	4	2	1	2	5	X	X	3	5
	Substantial	3	1	1	1	1	2	2	1	1	X	X	2	2
	Atomic	5	2	0	2	3	0	2	2	1	3	2	X	X
	Structured	1	3	4	2	2	4	1	1	5	5	2	X	X

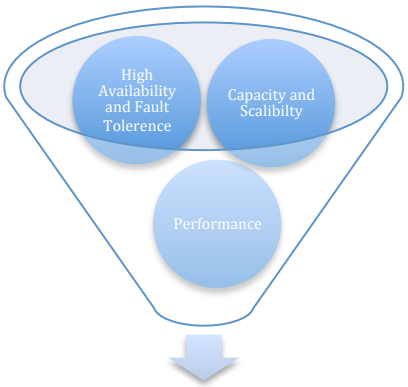
Table 2: Service Categories Vs. Metric Measures

2.5 Non-Functional requirement

Non-functional requirements for each service are directly derived from metric measures applied for each of them.

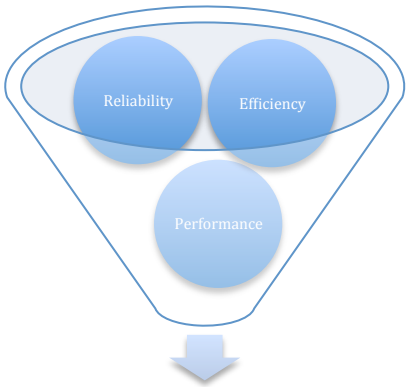
This section shall present important non-functional requirements such as Reliability, Scalability, Stability, Capacity, Performance, Efficiency and Fault Tolerance as well as their relationship with the different metrics measures the services are classified with.

First, *High availability* and *high performance* platforms are clearly essential characteristics of the container of all services regarded to be real-time frequency; because real time frequency services are heavily used and require fast response time. Moreover, Real Time frequency services are closely related to substantial volume metrics measures as appear in Table 1: Metrics Matrix, page 17, and recognized in Table 2, which on its part requires technological platforms to be within capacity to host significant amount of data and scalable to the rapidly growing size.



Real Time Frequency
Substantial volume

On the other hand, services and functions that of the analytical nature (metric measure type: analyze) are more regarded to need analytical handling and high processing in reasoning, pattern recognition, and multi-step heavy duty calculations which on its part inferred to demand *Reliability*, *High Performance* and Efficiency.



Analyze Type Services

3. Databases

A NoSql database is a platform that facilitates a mechanism for storage and retrieval of data, which is modeled in different means than the famous relational model. Relational Model has dominated the lion's share since the early rise of relational SQL Database Management Systems in late 1970s; as the most famous adopted model, which had seen a big success in traditional sense usage. As the early years of the 21st century have seen the commencing of the world creating large scale Web Applications to a scale that cater millions of users and growing to accommodate even billions with the introduction of the Internet of things. Such moves have resulted in complete new requirements that have been pushing Relational DBMS products beyond their limits. NoSql technologies have emerged to address those requirements that go beyond the capabilities of traditional RDBMSs.

This section of the document shall walk you through the databases of choice, their strength, and the reasoning behind being best fit for the use case.

> "Focus on the best architectural choice. Select the NoSQL database that fulfills your project's use cases. You may want a key-value DBMS for rapid storage and retrieval of binary data, a document DBMS for semistructured data and rapid development, a table-style DBMS for event log or clickstream data, or a graph DBMS for relationship analysis."
>
> Source: Gartner - Framework for Assessing NoSql Databases

3.1 Key-Value Database

Key-Value databases, whose records take the general form of atomic data pairs who's key stand as a unique identifier. Key-value DBMSs store both the key and value as binary objects. Dating back to Indexed Sequential Access Method (ISAM) in the 1970s, key-value is actually the oldest NoSql model. Key-value DBMSs evolved to support rapid scaling for simple data collections by automating the process of distributing data across several nodes. Key-value DBMSs support data access patterns that are driven by key lookups and require consistent access times. They are typically used in scenarios requiring the rapid storage and retrieval of binary data that is identified by a unique key.[4]

[4] Gartner - Evaluation Criteria for Big Data and NoSql DBMSs (2014).

Key-Value scalability explained

Key-Value DBs are demanded to bridge large-scale data management gap by bringing a highly available, vertically and horizontally scalable [5], and ultra fast DBMS. Thus, it's pushed by design to come up with an innovative solution to have the desired easy growth without affecting other aspects.

An interesting illustration brought by analyst Lyn Robison in Gartner published Research in 2013 " NoSql Databases: The Right Tool for Certain Jobs "explains it as follow: -

In a key-value database, Data is partitioned using a hash map to distribute it across multiple nodes. The data is split into shards (or "buckets") based on a hash of the key for each record. These shards are stored on nodes (physical machines). Each node can hold multiple shards, and physical machines can be added to reduce the number of shards per node and thus "scale out" the database. The database can also replicate shards to additional nodes as needed for redundancy. When an external application writes data to the database, the database finds the appropriate shard based on the key for the record provided by the application and writes the new value that the application gives for that record. If that shard has been replicated, the database can pick the most available or readily accessible shard in which to write the data. Thus, key-value databases can be partitioned to scale out quite readily. However, key-value databases have no intrinsic support for transactions that ensure data consistency between records or between shards. They also have no intrinsic support for joins that bring data together from different records or indexes that enable non sequential access of records based on their values. Therefore, key-value databases can be optimized for OLTP applications to write data to a highly available database. However,

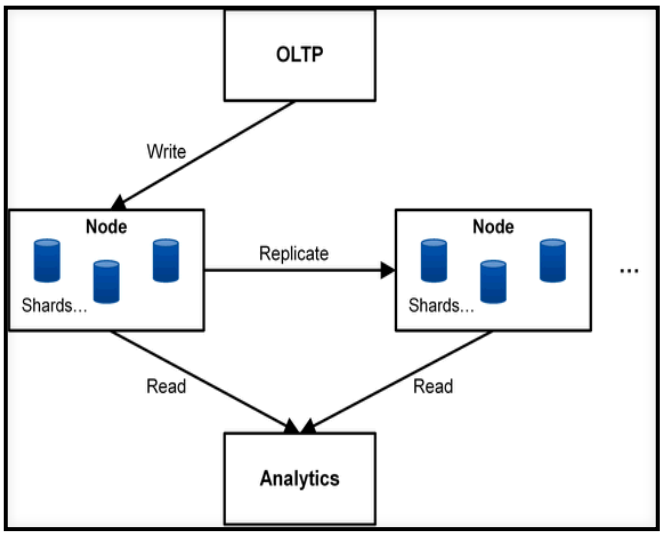


Figure 2: Key-Value DB clustering Architecture

Source: Gartner (May 2013)

[5] Vertically scalable: adding more nodes to clusters whilst horizontally scalable: increasing CPUs and RAMs capacity individual of nodes.

transactions, data consistency, joins and indexes all become application-level considerations.

3.2 Document Store Database

MongoDB is a cross-platform document style NoSQL database storage system that uses BSON to store data. It uses dynamic schemas which allows a user to create records without first defining the structure, such as the fields or the types of their values. It allows the user to change the structure of records (which are called documents in the MongoDB) simply by adding new fields or deleting existing ones. This flexibility allows the development systems to evolve the data model rapidly as and when the requirements change.

MongoDB has the ability to carry out in-depth queries. It converts JSON into BSON which provides more efficiency in computations and creates less amount of data. It is efficient when it comes to performing dynamic queries on documents using the document-based query language. Also, MongoDB creates indexes on a document using a single function call instead of using a more complex map reduced operation as used by CouchBase. It employs a custom binary protocol providing significant faster reads than CouchBase.

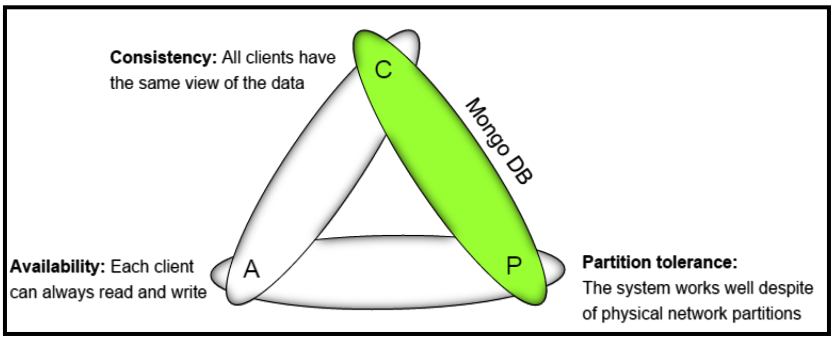


Figure 3: MongoDB stand on CAP theorem

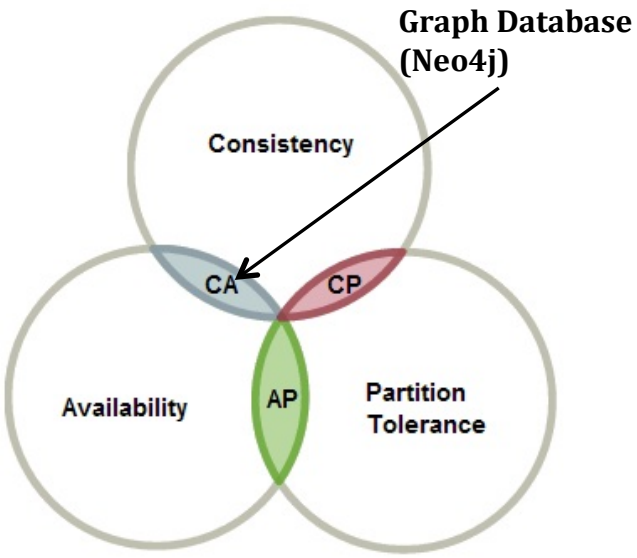
The task of our document store is to provide data regarding the different food items to the user (Key/Value database) and the diet plans (Graph database). Each food item can differ in its structure depending on the amount of information gathered. Hence a schema less system was key to successfully storing our data. As our data will always expand, defy a strict structure and would not require any internal relationship, it could be handled most efficiently by a document store.

MongoDB follows a Replication model to provide consistency. A primary node is used for all the write operations and by default the reads. The data is then replicated over to a set of secondary nodes. Querying the data is faster and can be used to create simple queries

without the need to perform a map-reduce. These features of MongoDB made it a very suitable option for our database.

3.3 Graph Database

Neo4j is a widely used the NOSQL graph database. As CAP theorem states, the three basic requirements for designing a system using NOSQL databases are Consistency, Availability and Partition tolerance. Theoretically, it is not possible to accomplish all the three aforementioned requirements. Therefore, the existing NOSQL databases fulfill different combination of consistency, availability and partition tolerance of CAP theorem.



Neo4j supports high availability and guarantees consistency by supporting ACID properties for transactions. Availability is achieved with the help of clustering through master slave architecture with one master for write operations and many slaves for read operations.

As per CAP theorem, Neo4j supports *Consistency* and *Availability* combination.

Significance of Graph Database

Graph databases are powerful systems that provide an efficient way to analyze and relate two entities. In the chosen user case, Personalized Nutritionist, it is vital to create a best diet plan according to the user's profile (height, weight, age, lifestyle etc.) and reuse this plan for already existing users with the same requirements.

Traditional relational databases require a connection between entities using special properties such as foreign keys. A graph database enables us to build an advanced model our specific problem domain by congregating the nodes and relationships into connected structure. In the chosen use case the data is captured from two different NOSQL databases and hence requires structuring and building relationships between the three

databases. Therefore, implementing the use case using a relational database increases join operation costs and hence is not fully fit for this use case.

By using a graph database each node (entity or attribute) holds a list of relationship information that represent its relationships to other nodes. Such relationship information is framed by type and direction and may contain additional attributes. Whenever a similar JOIN operation is run, the database utilizes this list and directly accesses related nodes, eradicating the need for a costly match or search operation. This capacity of pre-calculated relationships in the database structure permits Neo4j to provide faster performance in case of heavy join queries; an advantage that is truly rewarding as complexity increases.

4. Key-Value Database

4.1 Adoption of Key-Value in the project

A NoSql database and a relational databases both offer mechanisms for storing and retrieving data. However the data is modeled differently. NoSql databases have been successful in resolving the issue of managing big data. Therefore, it is the ideal and most optimal alternative for several of Personalized Nutritionist functional requirements.

As the architecture part of this document have unleashed through the four-dimensional metric measures analysis[6] the non-functional requirements for the application services which on its part calls for high responsiveness, high availability, and scalability.

Based on the findings of the requirements analysis, Key-Value DB has been landed as a platform choice for several services that demands the non-functional requirements aforementioned and where key-value structure is best applicable. The arguments supporting the proposal to name Redis out of other popular key-value databases are as follow: - (see Figure 4)

1- Redis is an open source and a non-commercial database, a requirement that crosses Oracle and IBM products off list.

2- Redis has been named Leader [7] in Garner's 2015 Magic Quadrant[8].

3- Redis implements key-value database to its definition without any extra features such as buckets as in the case of Riak DB; which -in our opinion- limits the powerful simplicity of Key-Value database.

	Type of Database	Form Factor
Oracle NoSQL Database Enterprise Edition	Key-value	Commercial software
IBM InfoSphere BigInsights	Key-value	Commercial software
Basho Technologies Riak	Key-value	Open-source project
Redis	Key-value	Open-source project

Figure 4: Key-Value DBs with highest market share. source: Gartner (May 2013)

Redis NoSQL database will be used to facilitate the storage of User fed Data that is atomic and with high frequency CRUD operation.. The strength of key-value databases will be benefited of in logging such sensor-like data and meeting the functional and non-functional requirements highlighted in the architecture analysis.

---

[6] For more details about the metric measures definitions, see Metrics Descriptors)page 14).

[7] "Execute well against their current vision and are well positioned for tomorrow." - Gartner Magic Quadrant

[8] "The Gartner Magic Quadrant is the brand name for a series of market research reports published by Gartner Inc., a US-based research and advisory firm. According to Gartner, the Magic Quadrant aims to provide a qualitative analysis into a market and its direction, maturity and participants" - Magic Quadrant ™.

Key-Value data store shall collect user fed data as listed below: -

- User details such as name, age, birthdate, height and …etc.
- User day-to-day activities- particularly food intakes and weight.
- Primitive health descriptions of the user including allergies and restrictions.
- User diet goals such as losing weight, gaining weight and so on.

4.2 Data Model and Organization

4.2.1 Key definition

The main role of the Key-Value store of Personal Nutritionist provides an efficient- highly available- and scalable mean of storage. However, Key-Value store shall be frequently referred to, queried, and aggregated. Moreover, accessing Key-Value data shall not be limited to bulk value list fetching and then processing at the application layer, but rather a more efficient way. As a result, we aim in designing the data model of the key-value store in a way that facilitates a strong mechanism of filtering the data without affecting the simplicity of Key-Value, the spirit of the model above all. Therefore, we have omitted the usage of accessory features such as built in lists, buckets, or other product specific features in order to stick the powerful simplicity of key-value. On the other hand, we have adopted a strong semantic glossary in the key that shall include three main elements.

The elements of the key are: -

1- The user whom the data describes, belongs to, or fed by, representing in the design, data model, or integration as ($Id) and refers to the user id in the Application Personal Nutritionist. A unique numerical value corresponds to a single user.

2- The data group the Key-Value corresponds to. Key-Value pair shall always belongs to a particular data group, which are specified as follows: -

   a. UserDetails. User static information, including name, birthdate, and so on.
   b. foodIntake. Food items/ elements of meals as added eaten or fed by user.
   c. Custom entries such as User Goals, Weight, etc.

3- Time representation. Time representation is an important element of the key in order to facilitate time filtering relying on the engine of the database rather than the application layer, not to mention additional added values such as archiving and faster sorting.

Below appear an example a particular property (user Birthdate), at it would fall under user details sub category.

Figure 5: Key breakdown example

Having the key element built in this format shall facilitate not only readability but also querying particular data or fetching a broad category of user details using key pattern. Querying Key-Value DB with Key pattern is a method that is basically reliant on regular expression[9]. Fetching the desired keys using key pattern is a powerful tool that inherits its strength from well-defined key elements.

4.2.2 Value Definition

The value contains primarily atomic information of interest to the object that is being indicated in the key. The data being described in the value is regarded as atomic from the perspective of Personalized Nutritionist Key-Value design is when the value that describes the key is meaningful of its own without linking or referencing other values. Atomic values can still be referenced, described within themselves by descriptive information –metadata- which are in relation with the values themselves not the object being described (the key).

However, the value can be yet compound of several entries or non-compound. For example, body weight as a value shall not be just a single value of the kilograms but rather weight, fat percentage, water percentage, and etc.

[9] Framework for searching text using pattern, highly used in validation of user inputs for example dates and phone numbers. e.g. regular expression query A[0-9] would search the text where a single digit occurs after the letter A. matches would include A5, A6, and so on.

Metadata

In addition, the value shall contain standard relevant metadata as needed. Meaning, whenever a value is being fed, the value itself should also describe who created it, or modified it and when as well as through which channel. More information of the values metadata is found in Table 3.

Metadata property	Description
Created By	Reference to an entity [10] by which the value was created.
Last Modified By	Reference an entity by which the value was last modified.
Creation TS	An Epoch Time [11] value indicating date and time in which the value was created.
Last Modified TS	A value indicating date and time in which the value was last modified.
Channel	A reference to s channel[12], through which the value was created.

Table 3: KV - Values Metadata Reference

Defining such metadata for the values that is closely maintained with each KV pair is believed to be a good design that stresses not only security but also encourages the solution to stay well-aware of its building blocks which on its part shall have high analytical valuable return.

Adoption of metadata closely to what can be seen, as particles of the KV DB will not, on the contrary, contradicts to the atomic value definition stated as it is in relevance with only the value rather than the key, or the object being described.

Value Data Type

Values provided by Redis can be of most major data types such as Strings, Jsons, Hash maps or others.

However, for the sake of embracing powerful simplicity of KV as well as for easier integration and interoperability, Values shall be always of Comma Separated Strings. Nonetheless, avoidance to use JSON or other formats provided by various Key-Value DB vendors is a decision inherited from the fact that the design of Personalized Nutritionist strive to meet portability, when needed, and avoid unnecessary redundancy of the data as sub-value labels would be included in the case of JSON in values quite redundantly.

---

[10] An entity may refer to a user, a system process or component, or a device such as a scale.

[11] Also known as UNIX time, a system for describing instants in time, defined as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970.
Source: IEEE Std 1003.1, 2013.

[12] System channel can be an app, Website, or exposed APIs for example.

4.2.3 Key-Value Property Specification

Table below clarifies all the key-value properties and stand as a blueprint for the values along with their metadata when integrating with other system DBs or components. Every property listed will be supported with an example. Each property has a key definition that its structure is explained in 4.2.1 Key definition, (page 26) and a value reference following a standard format explained in the previous subsection. .

1	**Key Model**	**$id.userDetails.name**							
	Value Model	First name	Middle name	Last Name	Creation TS	Created By	Last Modified TS	Last Modified By	channel
	Example	"Renu" ,"", "Thomas" , " 1454255672","1","","",""							
2	Key Model	$id.userDetails.birthdate							
	Value Model	Date	Month	Year	Creation TS	Created By	Last Modified TS	Last Modified By	channel
		Date is given by : dd.mm.yyyy							
	Example	"30.04.1992", "1454255672","1","","",""							
3	Key Model	$id.userDetails.height							
	Value Model	Height	Creation TS	Created By		Last Modified TS		Last Modified By	Channel
		Height: is described in centimeter							
	Example	"160","1454255672","","","",""							
4	Key Model	$id.userDetails.bloodtype							
	Value Model	Blood type	Creation TS	Created By		Last Modified TS		Last Modified By	Channel
		Blood type :it Can be A+,A-,B+,B-,O+,O-,AB+,AB-							
	Example	"B+", "1454255672","1","","",""							
5	Key Model	$id.userDetails.Allergies							
	Value Model	allergy,allergy,…	Creation TS	Created By		Last Modified TS		Last Modified By	Channel
		Allergies: Refers to the food items, which some user may be allergic to.							
	Example	"Eggs,Kiwi", "1454255672","1","","",""							
6	Key Model	$id.userDetails.foodPreference							
	Value Model	Preference, Preference,..	Creation TS	Created By		Last Modified TS		Last Modified By	Channel
		Personal preference: Refers to food items/elements that the user avoid due to some personal preference and restrictions							
	Example	"Beef", "1454255672","1","","","channel"							

7	**Key Model**	**$id.userDetails.conditions**						
	Value Model	condition, condition,…	Creation TS	Creation By	lastModified TS	LastModified By	Channel	
		condition: Refers to illnesses, diseases, or chronic conditions, which user may suffer.						
	Example	“Hypertension”, “1454255672”,”1”,””,””,””						
8	Key Model	$id.userGoal						
	Value Model	User Goal	Current Weight	Targeted Weight	Creation TS	Created By	Last Modified TS	Last Modified By	channel
		User Goals: weightLoss, massGain, maintainWeight							
	Example	“weightLoss”, “65”, “50”, “1454255672”, “1”, “”, “”							
9	Key Model	$id.userWeight							
	Value Model	weight	Fat%	Water %	Creation TS	Created By	Last Modified TS	Last Modified By	channel
	Example	“70”,””,”” “1454255672” , “1”, “”, “”,””							
10	Key Model	$id.foodIntake.ddmmyyyy.EatenFor.ItemId							
	Value Model	id	itemId	item Name	serving	quantity	Eaten For	date	time	Metadata[13]
		ItemId: refers to the food item id as stored in Nutrution Fact Store. Item Name: refers to name as fetched from Nutrition facts store (Document store). Serving: refers to serving type (tablesppn/cup/gram, etc) as fetched from document store. Quantity: quantity of serving as provided by user. EatenFor: breakfast/lunch/dinner /snack.								
	Example	“1”,"18012","Biscuits","1.0 biscuit","2","snack","29012016","16:35h”,” 1454255672”,”1”,””,””,””								
11	Key Model	$id.userDetails.gender							
	Value Model	gender	Creation TS	Created By	Last Modified TS	Last Modified By	channel		
		GenderCode------→ M: Male. F:Female. O:other/Not Specified.							
	Example	“F”, “1454255672”,”1”,””,””,”channel”							
12	Key Model	$id.userDetails.FoodPractice							
	Value Model	FoodPractice	Creation TS	Created By	Last Modified TS	Last Modified By	channel		
		Veg: Vegetarian, general definition indicating withholding oneself from the consumption of the flesh of any animal. OvoVeg: Ovo vegetarianism, allows for the consumption of eggs but not dairy products. LactoVeg: Lacto vegetarianism, a vegetarian diet that includes dairy products but excludes eggs.								
	Example	“Veg”, “1454255672”,”1”,””,””,”channel”								

Table 3: Key-Value Property Specification

[13] Merged for spacing purpose, for more information see Metadata, page 27.

4.2.4 Where Key-Value DB stand

Key-Value store in Personal Nutritionist is a multi-node and a highly scalable elastic persistent cache, which serves fast data dumping by users through various channels such as phones and tablets. On the other hands, Key-Value store is a repository for frequently needed data by other modules of Personal Nutritionist. Diagram below illustrates role of Key-Value database as intermediate with users through channels and other modules through the application layer. It's worth mentioning that Key-Value DB attempts in its Data Model to remain well aware of both the channels and service beneficiaries by maintaining a low-level metadata that stays on track of both. More on that thoroughly explained in 4.2.2 value Definition (page 27).

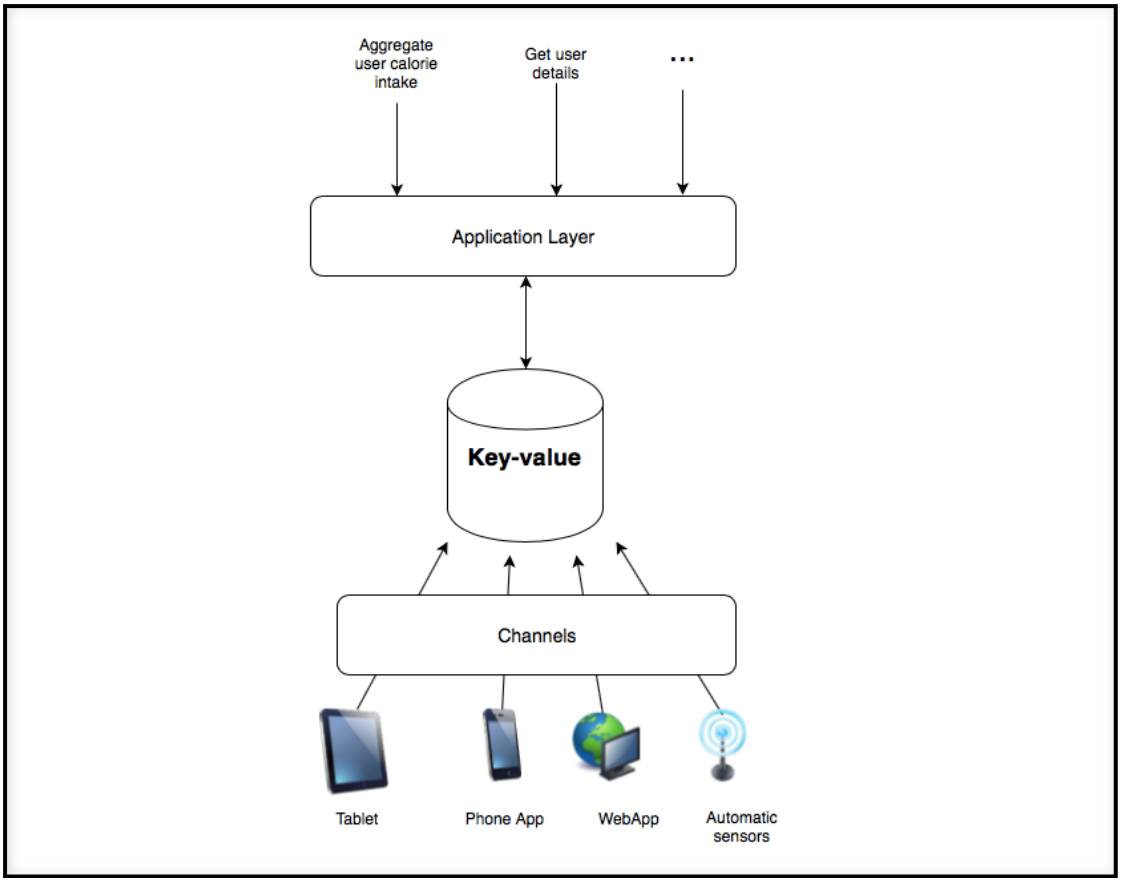


Figure 6: Key-Value DB , architectural view diagram

4.3 CRUD Operations

The following pages will be concentrating on the CRUD operations of the Personalized Nutritionist System. It will guide through different user operations that can be executed on the system along with examples of some conducted queries.

Different users of the Personalized Nutritionist System may have different CRUD cycles based upon their requirements. Database access is done using various channels such Phone-App, Website or automatic sensors through different platforms. Each channel enables the user to conduct CRUD operations on their data. They are free to implement database access with numerous choices of supported client types such as Java, PHP, Web Services, REST APIs and so. Nevertheless, Redis offers a GUI (Graphical User Interface) desktop application in which the admin can access, read the data and directly execute CRUD operations on them.

In the Personalized Nutritionist System, users can have the ability to create their details such as their names, birthdates, their food preference and etc. Users can then access (retrieve) their stored details whenever needed; they can also modify (update) existing data such as their weight and can as well delete any of their details if necessary.

4.3.1 User operations

In this personalized Nutritionist system, all users can Create, Update, Retrieve and Delete the following information:

- Name
- Birthdates
- Gender
- Height
- Weight
- Blood type
- Location
- Food preference
- Allergies
- Food practice
- Food intake

Create Operation

Create through Admin console

Selecting the “add key” button shows a window where the user can provide the key. By typing the respective value, key-Value pairs can be created.

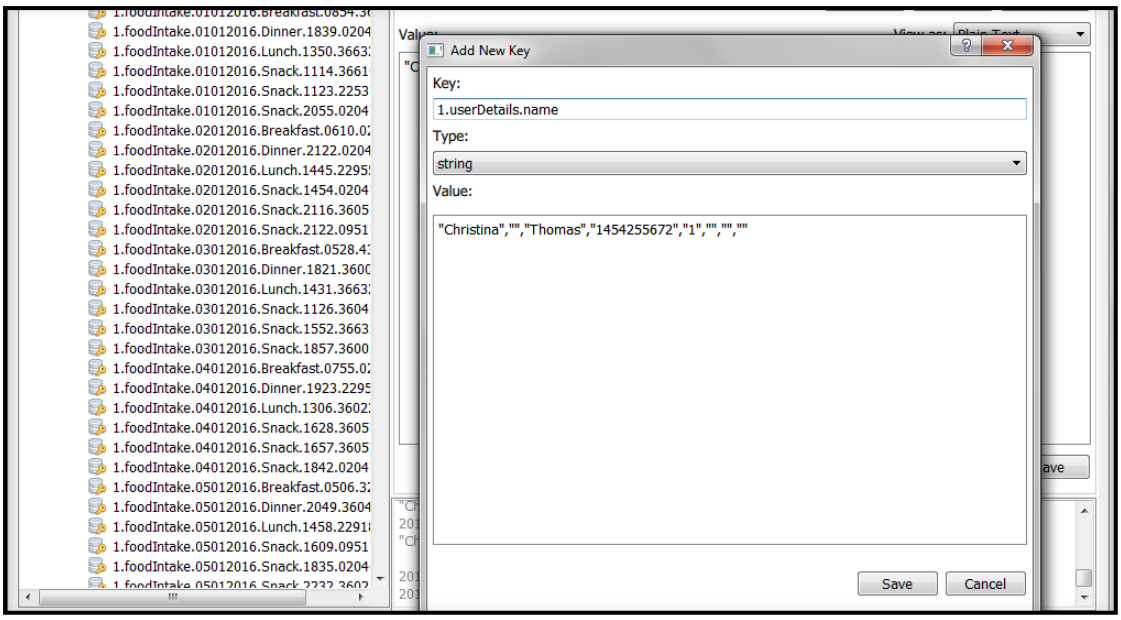


Figure 7: Key-Value creation through Admin console.

Create, Programmatically

```
for(KeyValuePair kv:keyValPairs)
{
    jedis.set(kv.key, kv.value);
    //System.out.println(kv.key+"="+kv.value);
}
```

Figure 8: Pushing KV pair conducted with one line of code

Update Operation

UPDATE through Admin console

Locating the desired key and editing it directly on the console can simply update key-Value pairs.

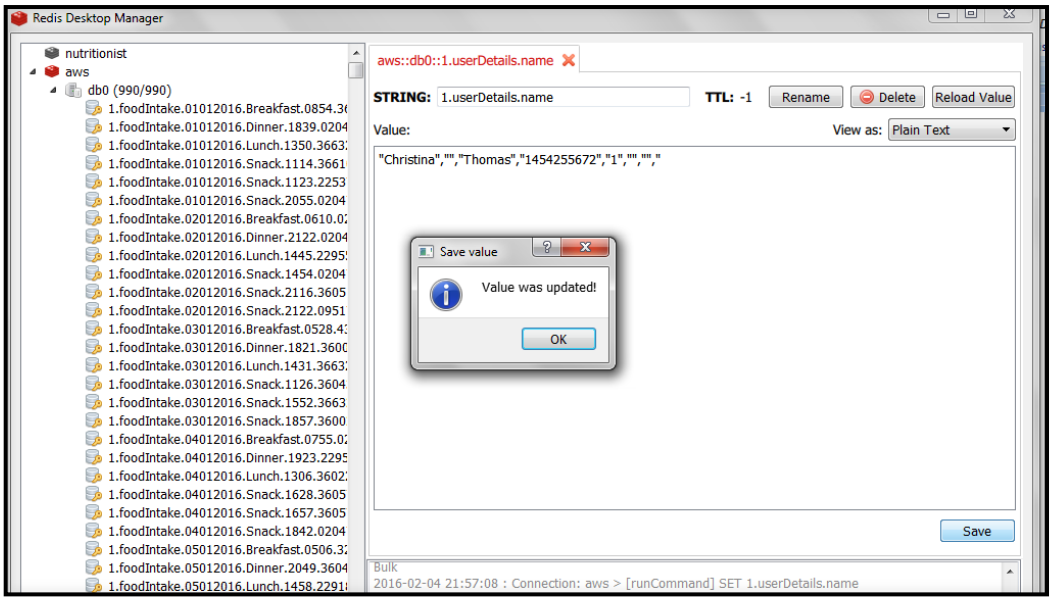


Figure 9: Value update is an easy operation through admin console

UPDATE, Programmatically!

Updating Key-Value pairs programmatically is rather an easy operation that updates the values of keys matching key pattern operation.

```
run:
Connection to server sucessfully
1.userDetails.allergies Updated!
Connection and update operation elapsed: 82 Milliseconds
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
long startTime = System.currentTimeMillis();
//Connecting to Redis server on localhost

Jedis jedis = new Jedis(serverAddress);
System.out.println("Connection to server sucessfully");


//jedis.set("Hello World", "test");

jedis.update("1.userDetails.allergies", "\"kiwi\",\"1454255672\",\"1\",\"\",\"\",\"\"");

 System.out.println("Connection and update operation elapsed: "+ (System.currentTimeMillis()-startTime)+" Milliseconds");


if(1==1)
    return;
```

Figure 10: Updating Key-Value pairs programmatically

Retrieve Operation

The user can retrieve all their stored information whenever needed by using the filter key button, which shows the data of the corresponding key.

Retrieve relies on Regular expression strong capabilities to filter out data.

Retrieve through admin console

Appears to the left of the window, the regular expression filter that shall match key that contains user food preference.

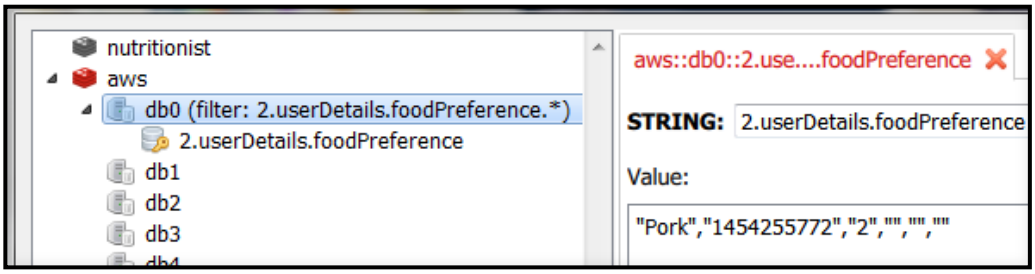


Figure 11: Retrieving food intake through admin console

Retrieve Programmatically

Shown below is how to retrieve programmatically food intake logged for breakfast by all users.

Key pattern:

```
*.foodIntake.21012016.Breakfast.*
```

Result:

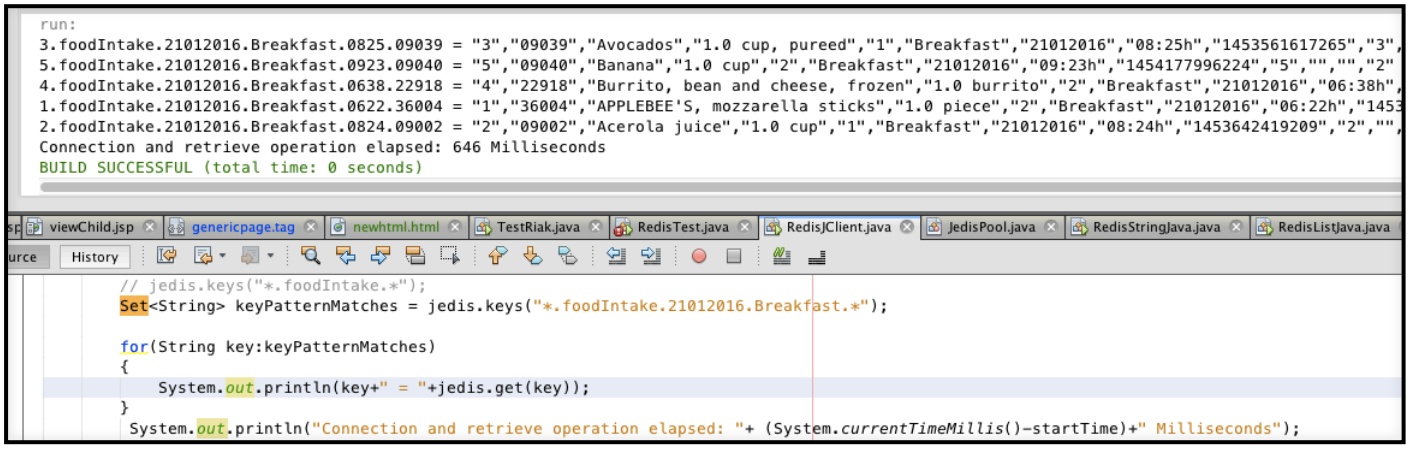


Figure 12: Retrieving food intake programmatically

Delete Operation

Delete through admin console

Deletion through admin console can be done with a single command that will instruct to remove the key-Value pair.

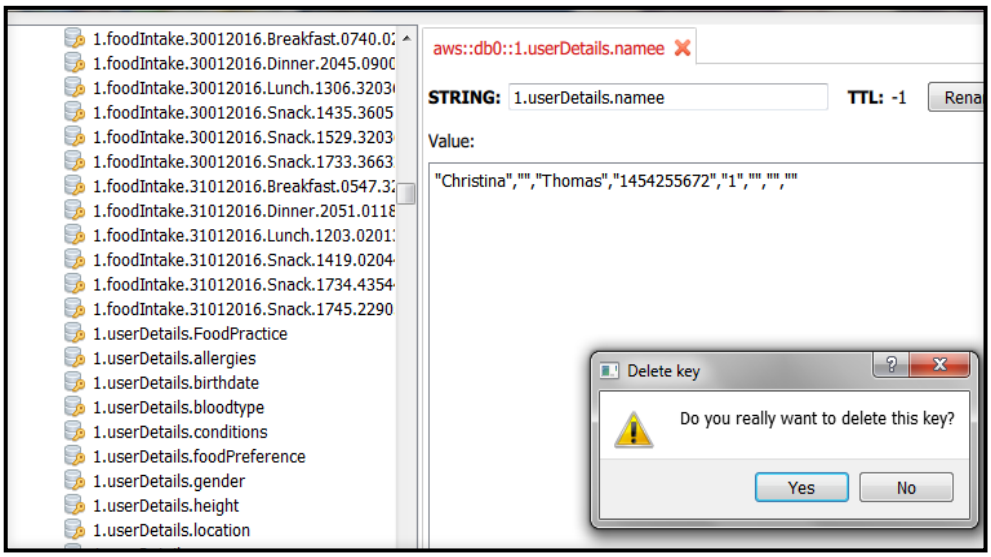


Figure 13: Deletion through admin console

4.3.2 Bulk operations

As Key-Value database supports regular expression Key-Value pairs retrieval, a highly rewarding feature to employ and exploit when cleansing, archiving, or deleting the data. An example below shows deleting 620 records in a single operation.

In the following example, our proof-of-concept will aim to delete all records of food intake for all users and all meals at all times.

Key Pattern to remove all Food intake records:

.foodIntake.

Result:-

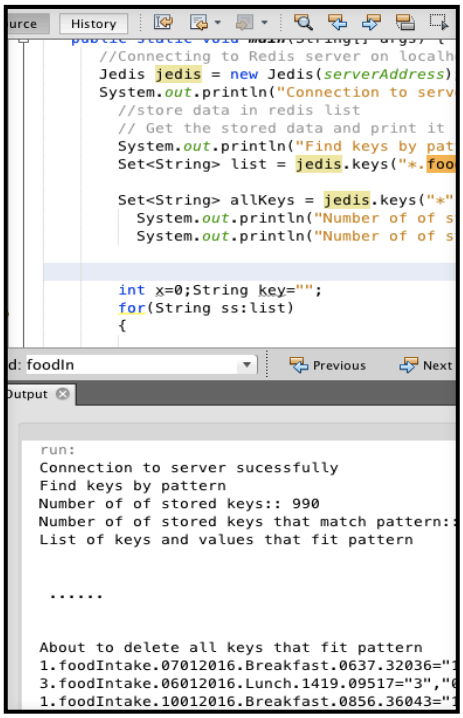


Figure 14: Bulk Operation Example Result

4.4 Benchmarking of Key-Value database

It is important to conduct personalized evaluation of performance metrics of the database to test whether it meets what it claims to be; as well as, whether it can accommodate the assumed CRUD operations at its highest and thus meet the non-functional requirements derived from the four-dimensional requirement analysis[14].

Nonetheless, our benchmarking of the database shall come out to answer the following several important questions: -

1- is the database fault tolerant for high demanding access in a short period?

2- how responsive and fast is the database in handling CRUD operations?

3- Can the DB handle simultaneous access and without responsiveness and speed affected?

Benchmarking case

Two parallel processes run against Redis Key-Value Database in aim to flush a pool of 930 keys. The benchmarking test case is ranked successful when the database satisfies test criteria's.

First the database handles all requests and remains responsive.

Second, CRUD operation shall remain within average of less than 40 milliseconds.

Third, insertion rate is not affected with simultaneous access, that both threads will meet similar standard throughout the whole experiment.

Finally, as Redis installation and implementation is hosted on a remote server that reside more than 15 network hubs away and 100 km in distance from where the experiment code is run, we assume an average on request traveling elapses 7 milliseconds.

[14] See more on the analysis description and services classifications on the Architecture part. (Page 14)

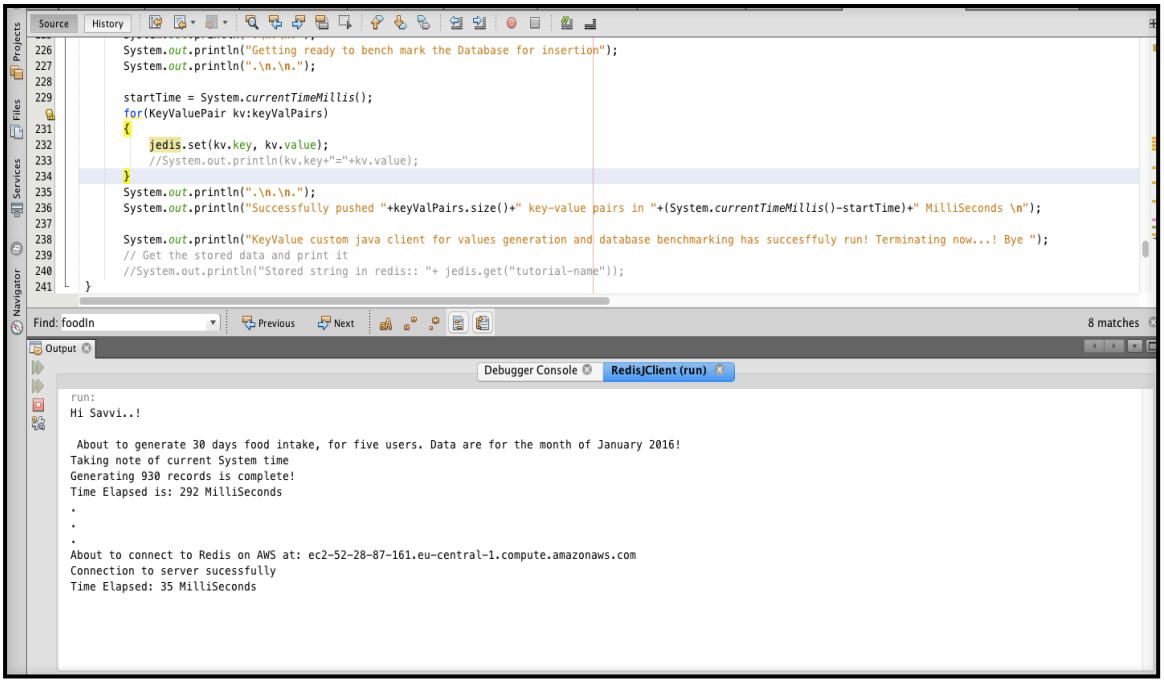


Figure 15 Custom written java client about to benchmark the DB.

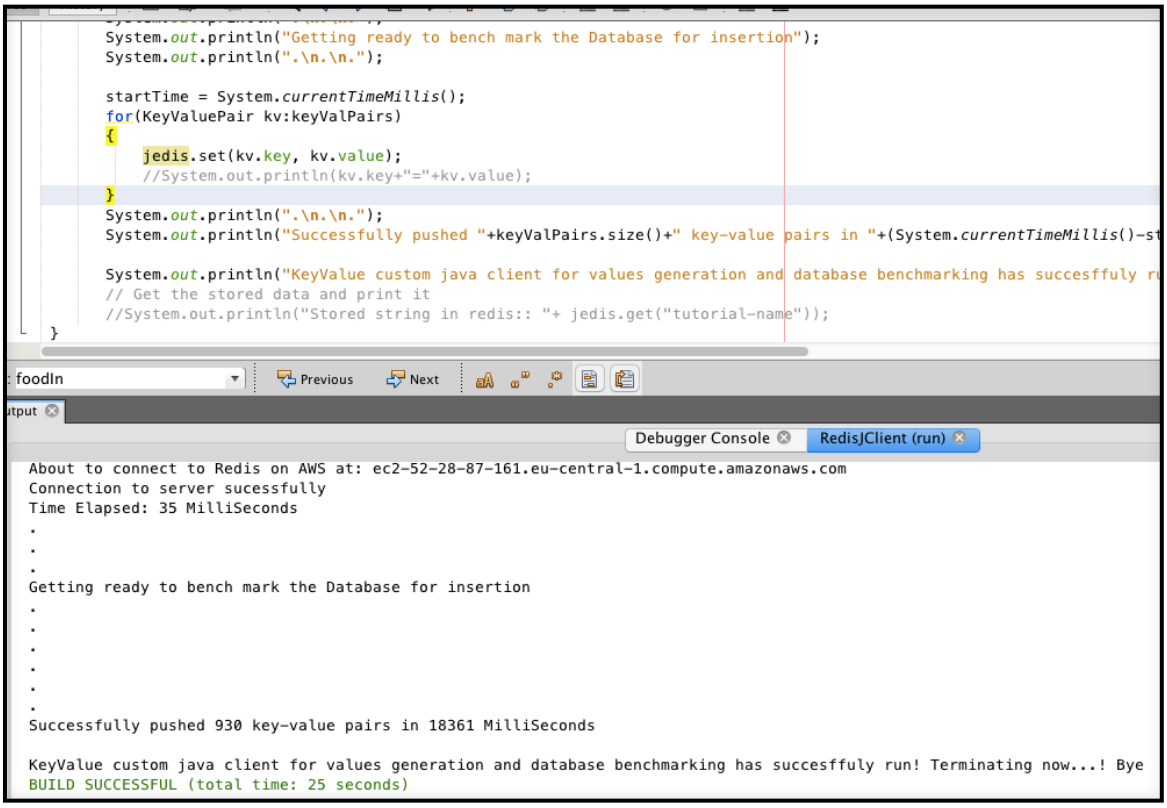


Figure 16: Result of benchmarking

In Figure-15 above, multithreaded connection to Redis DB has successfully reached its goal with 930 keys generated on the fly and pushed over the network as Redis is hosted

on a remote site. 930 keys were simultaneously pushed in 16,166 seconds, with average 17 Milliseconds per record.

Total Records	**930**	Records
Thread 1 Average	17.29	Milliseconds
Thread 2 Average	17.25	Milliseconds
Max Elapsed	24	Milliseconds
Min Elapsed	12	Milliseconds
Average Deviation	2.08	Milliseconds
Standard Deviation	2.92	Milliseconds
Variance	8.55	Milliseconds

Test result above ensures that the DB is fault tolerant against high demand.

The result of average insertion rate being less than 20 Milliseconds is outstanding and shows high performance responsiveness and responsiveness that even exceeds requirements. Finally, Running a multithreaded access to the DB with insertion and retrieval proves high efficiency of parallel read/write operations as the insertion rate kept within appropriate deviation from the average. Moreover, the two threads have successfully managed to divide share of the benchmarking data due to the high responsive rate of the database with no affect of large variations; as seen below. parallel access responsiveness

Figure 17 visualizes the detailed simultaneous insertion rate of every record for both threads where blue is Thread-1.

We can notice that both threads at the beginning started at a very low rate seemingly averaging 15 Milliseconds whilst both rose to around 24 Milliseconds. This is an indication that the slight affect is caused by environmental affects- mainly network. On the other hands, and as clearly seen in the two parallel accesses – Redis has maintained an efficient and fast capacity is parallel access responsiveness

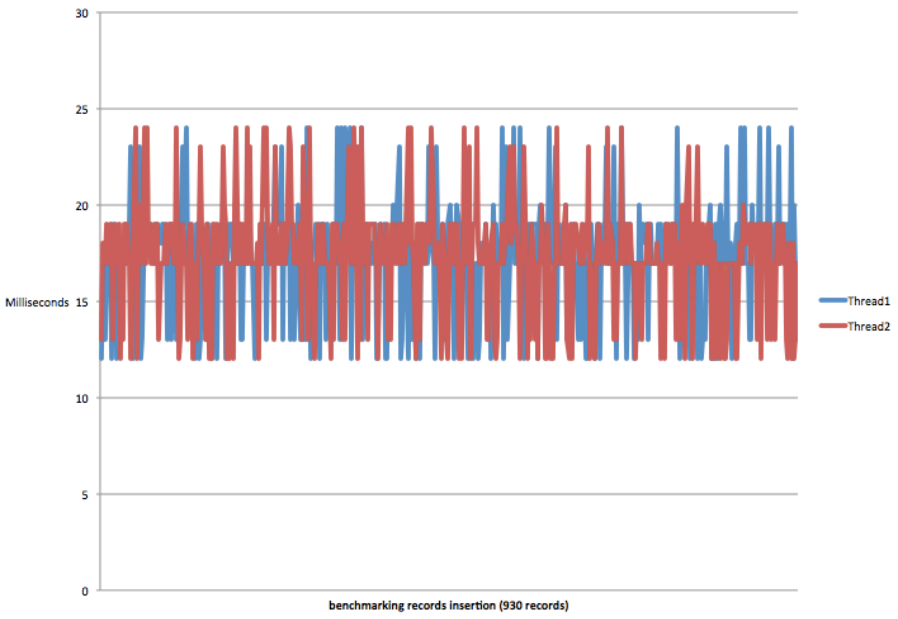


Figure 17: Full visualization of simultaneous CRUD benchmarking test case.

4.5 Meaningful Queries (by key pattern)

1- Get food intake per day for a user with Id=1.

Key Model for Food intake (for more details about key elements and data model see page 26):

UserId . foodIntake . date(ddmmyyyy) . Meal . hhmm(time). FoodItemId

Key Pattern for query:

```
1.foodIntake.01012016.*
```

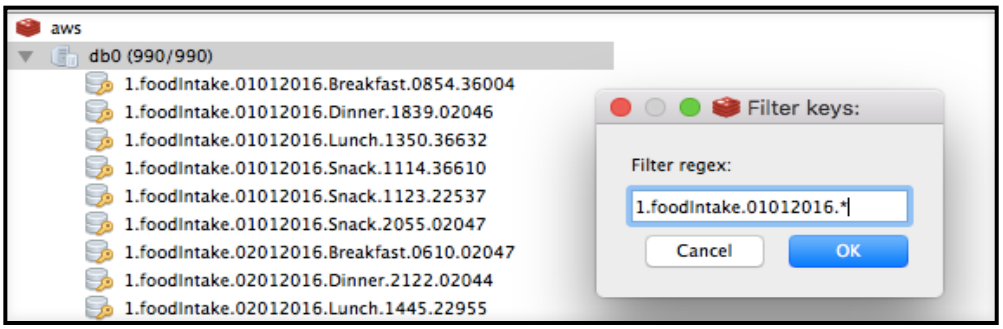


Figure 18: key pattern queries using regular expression (Search Dialoge)

Figure 19: Key pattern queries using regular expressions (Result Set)

2- get food intake per user per meal

Key Pattern for query:

```
1.foodIntake.01012016.Dinner.*
```

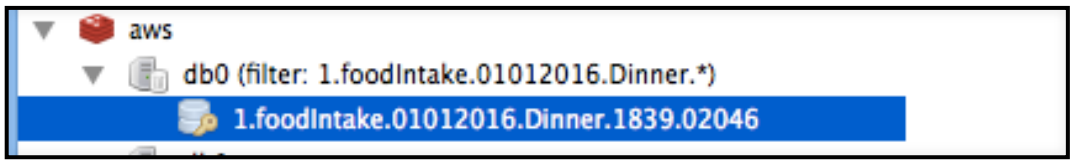


Figure 20: Fetching dinner food intake record

3- Get all users snacks

```
1.foodIntake.*.Snack.*
```

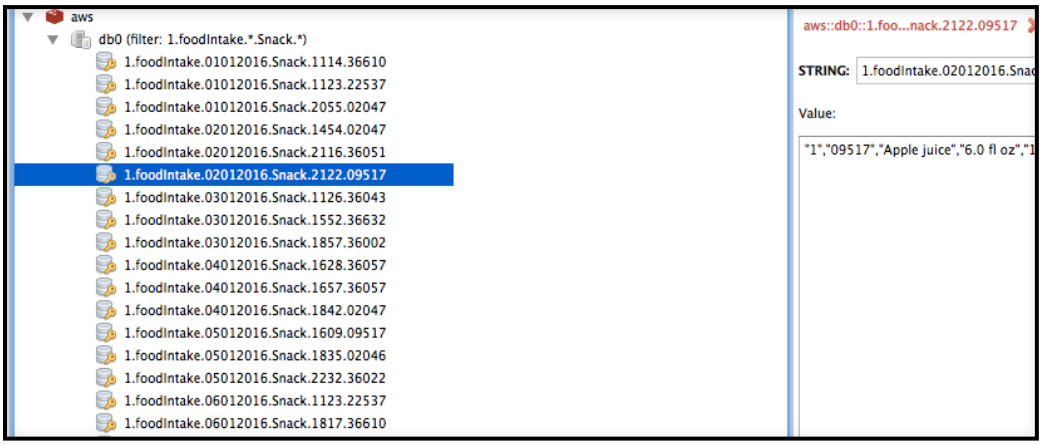


Figure 21: Fetching snack food intake record (Result Set Partial)

4- Get all snacks eaten on a particular hour (22:00h -22:59h) on a given day (02.01.2016)

Key Pattern Query:

```
1.foodIntake.01012016.Snack.22*.*
```

```
 * @author abdul
 */
public class RedisJClient {

    /**
     * @param args the command line arguments
     */
    static String serverAddress="ec2-52-28-87-161.eu-central-1.compute.amazonaws.com";
   public static void main(String[] args) {
      //Connecting to Redis server on localhost
      Jedis jedis = new Jedis(serverAddress);
      System.out.println("Connection to server sucessfully");
        //store data in redis list
        // Get the stored data and print it
        System.out.println("Find keys by pattern");
        Set<String> list = jedis.keys("*.foodIntake.02012016.Snack.22*.*");
        //Set<String> allKeys = jedis.keys("*");
        //System.out.println("Number of of stored keys:: "+allKeys.size());
        System.out.println("Number of of stored keys that match pattern:: "+list.size());
        for(String ss:list)
        {
            System.out.println(ss+"="+jedis.get(ss));
        }
```

foodIn | Previous | Next

tput

Debugger Console | RedisJClient (run)

```
run:
Connection to server sucessfully
Find keys by pattern
Number of of stored keys that match pattern:: 1
2.foodIntake.02012016.Snack.2258.36610="2","36610","DENNY'S, french fries","1.0 serving","2","Snack","02012016","22:58h"
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 22: Retrieving Key-Value pair programmatically

5- Get all Vegetarian users' names

Getting all vegetarian users by fetching Food Practices of all users through key pattern and processing them programmatically.

```
public class RedisJClient {

    /**
     * @param args the command line arguments
     */
    static String serverAddress="ec2-52-28-87-161.eu-central-1.compute.amazonaws.com";
    public static void main(String[] args) {
      //Connecting to Redis server on localhost
      Jedis jedis = new Jedis(serverAddress);
      System.out.println("Connection to server sucessfully");
        //store data in redis list
        // Get the stored data and print it
        System.out.println("Find keys by pattern");
        Set<String> list = jedis.keys("*.userDetails.FoodPractice");
        //Set<String> allKeys = jedis.keys("*");
        //System.out.println("Number of of stored keys:: "+allKeys.size());
        System.out.println("Number of of stored keys that match pattern:: "+list.size());
        for(String ss:list)
        {
           // System.out.println(ss+"="+jedis.get(ss));
            if(jedis.get(ss).contains("Veg"))
                System.out.println(getNameFromNameValue(jedis.get(ss.substring(0, ss.indexOf("."))+".userDetails.name"))
                +" is "+jedis.get(ss).split(",")[0]);
        }
```

```
oodIn    Previous    Next

ut

Debugger Console    RedisJClient (run)

run:
Connection to server sucessfully
Find keys by pattern
Number of of stored keys that match pattern:: 5
Sallie Santos is "LactoVeg"
Luanna Marsh is "OvoVeg"
Christina Thomas is "Veg"
Vicky Salmeron is "OvoVeg"
Elsa Scaria is "LactoVeg"
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 23: Programmatically retrieving names of all Vegetarian users.

11- Get all food practices of all users

Key Pattern Query:

```
*.userDetails.FoodPractice
```

Figure 24: Retrieving food practices records of all users using key pattern.

5. Document Store Database

5.1 Role of a NoSql database in the project:

NoSQL databases and relational databases both offer mechanisms for data storage and retrieval. However, relational databases were not built to cope the ever-growing scalability and agile challenges faced by modern applications, nor they were built to take advantage of the processing power available today. NoSql databases on the other hand enclose a wide variety of the various database technologies that were developed in response to the ever-growing demands present in modern day applications. Therefore, a NoSql database is ideal and the most optimal fit for the Personalized Nutritionist system.

MongoDB database will be used as the document store of the Personalized Nutritionist system. The documents can contain one or more fields, including arrays, binary data and sub-documents, which will be used mostly in case of our project. The strength of document store databases is benefited from, by storing a repository of the nutrient facts of the commonly available food items.

The document store shall store the nutrition facts of every common food item available in a detailed description. The food items comprise of various categories such as those of Baked Products, Beef/Pork/Poultry Products, Fruits, Vegetables, Restaurant meals and many more. Nutrition facts capture calories, proteins, carbohydrates fats, various vitamins, minerals and many more.

5.2 Data Model

5.2.1 Structure of the document store:

The food items are individually stored as a single document and each document consists of the following fields:

- Name
- ID
- Weight
- Measure (Serving size)

- Nutrients [{Nutrient_id, Nutrient, Unit, Value, Gm}]

  The array nutrients is specified for the nutrients : Protein, Fats, Carbohydrates, Calories, Fiber, Cholesterol, Sugar, Water, Vitamin A, Vitamin B, Vitamin C, Vitamin D, Calcium, Magnesium, Iron, Potassium, Copper and Zinc.

Below is the schematic representation of the document.

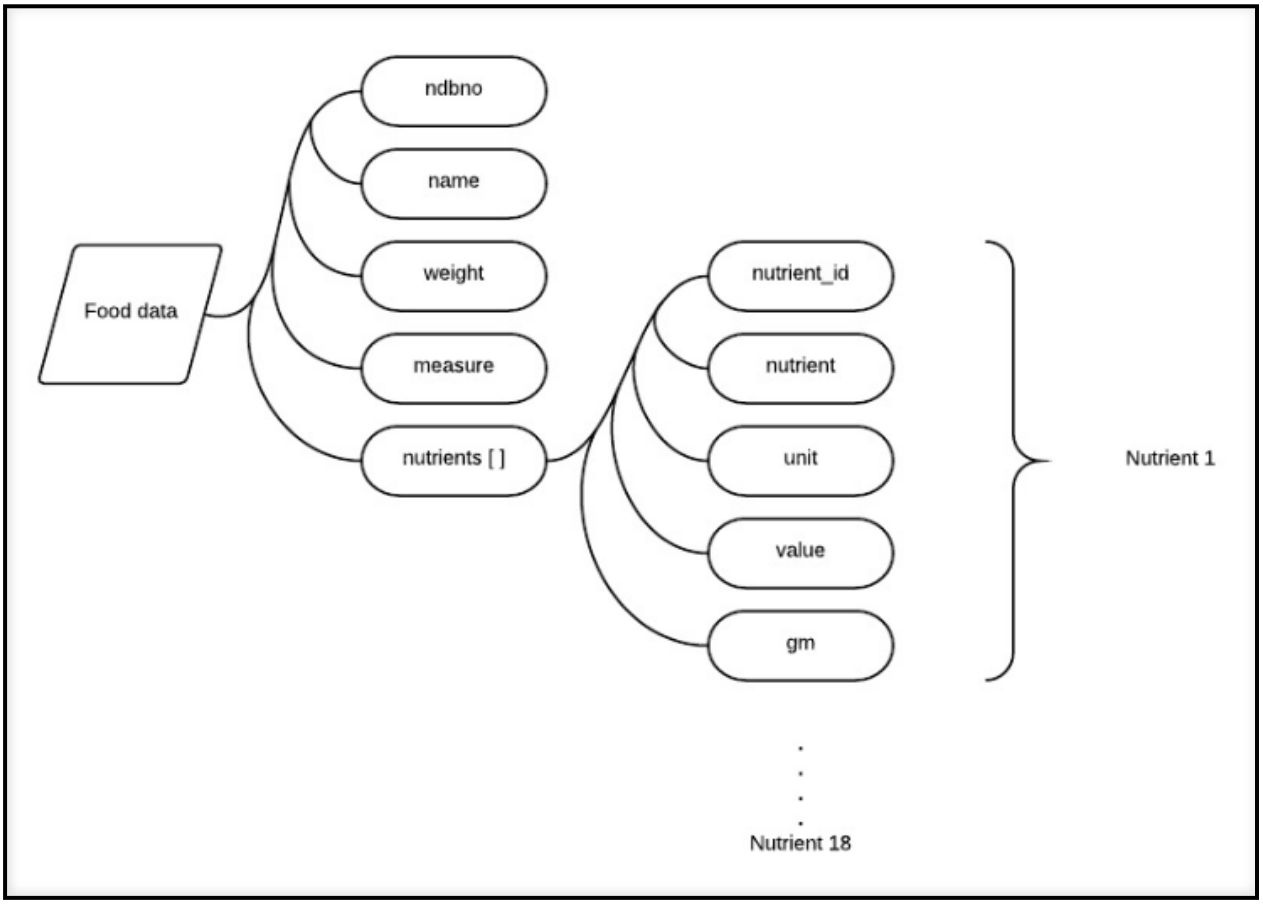


Figure 25: Data model of the document store

Note: The basic schema of the document shall remain the same but the number of nutrients may vary depending on data availability.

5.2.2 Systematic collaboration of the three databases:

The objective of the document store in Personalized Nutritionist as an application is to store the nutrition facts of every common food item available in a detailed description.

These details will be feed to the graph database for constructing diet plans. Also, this data will be utilized by the user to keep a track on his/her daily nutritional intake.

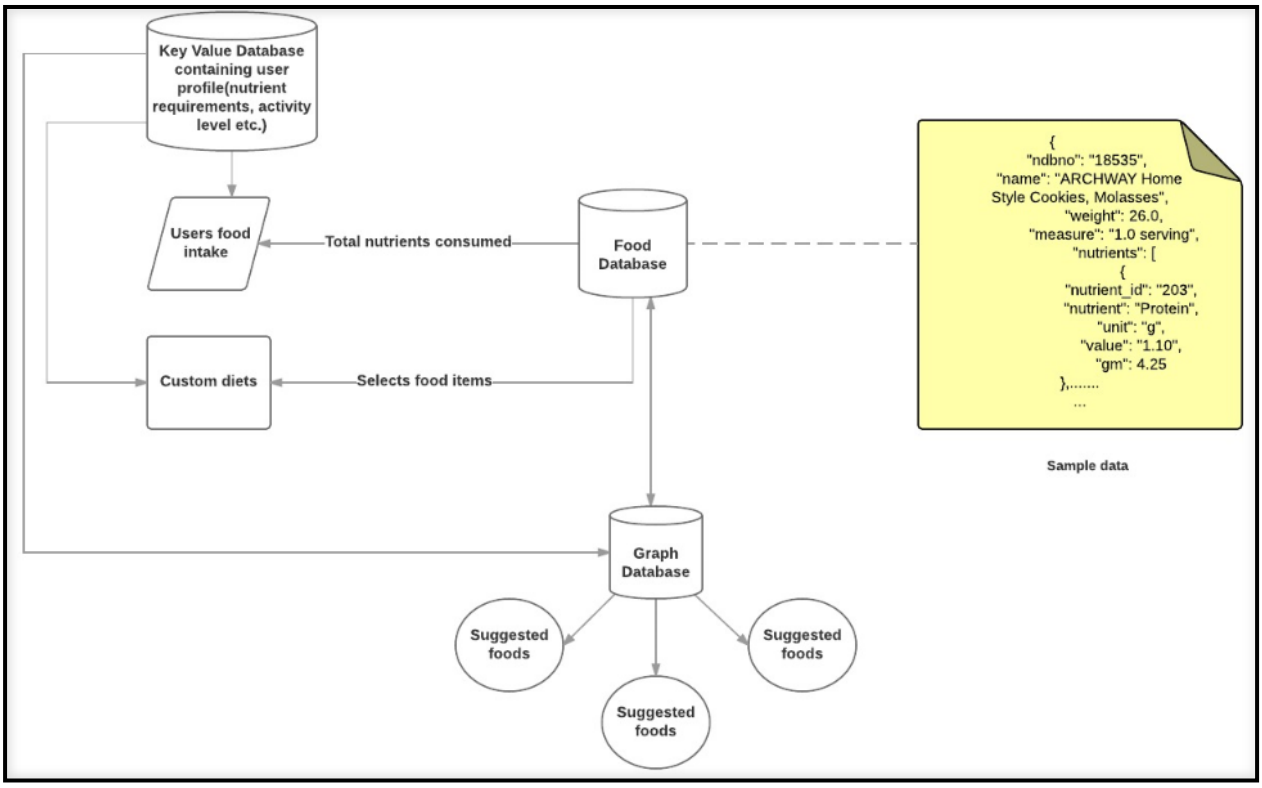


Figure 26: Information flow of all three databases

5.3 Features provided by the document store:

1) Aggregate the total nutrient intake of the user.
   Based on the food selected by the user, the program will calculate his total calories consumed along with the aggregation of the various other nutrients.

2) Show nutrition facts of common food items.
   The document store will contain all the food items and the nutritional values they carry. This data will be returned to the user when he/she selects a particular food item.

3) Allow addition of new food items.
   There are different food items available from different parts of the world, so there is always scope of expansion. Some of this data might not be readily available in the

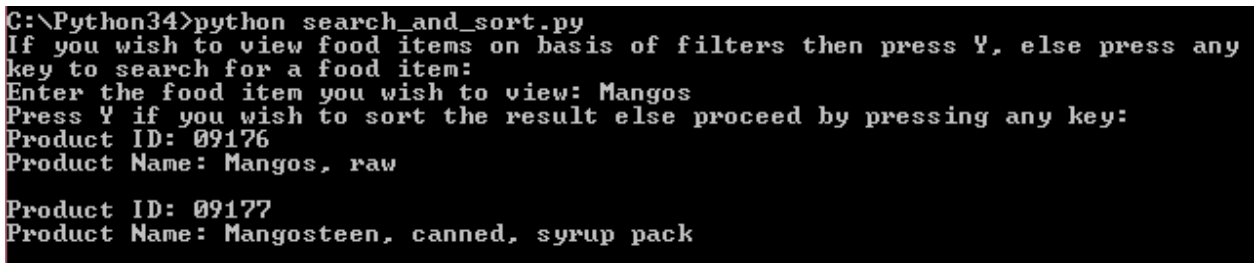
database. Hence, an option is provided to the user to add his own food (e.g. Homemade food delicacies such as pulao, halwa, currywurst).

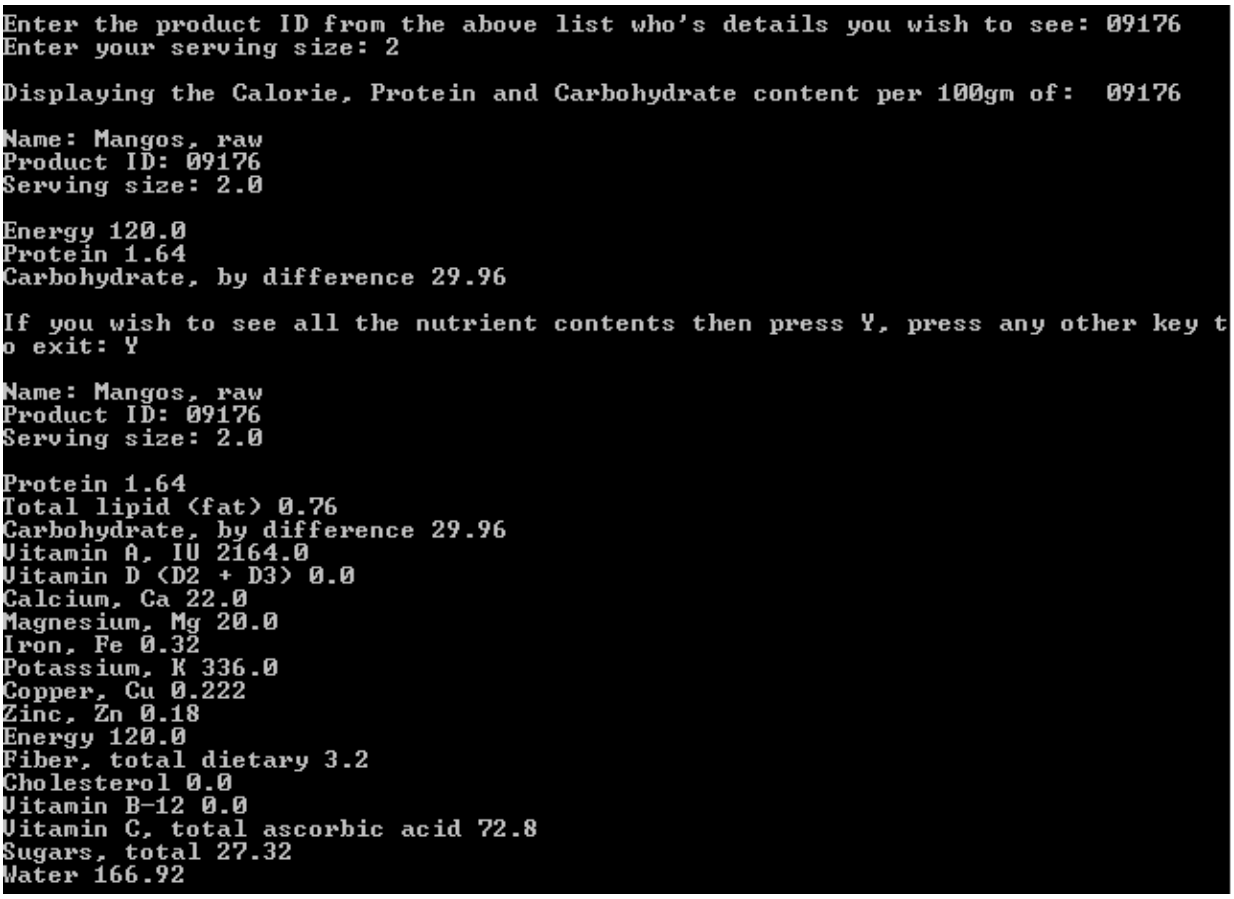
4) Modify existing data of any food items.
   If any discrepancy is found in our database regarding the nutritional content, we can update the data after performing a thorough check about the food.

5.3 Implementation of features using CRUD operations:

We have used Python programming language to carry out the CRUD operations for the feature implementations. Pymongo was used as the operation driver in MongoDB to execute CRUD operations.

1) A user wants to check the daily nutrient intake.
   The user can select n food items and provide the serving size, at the end of the day a program will calculate the total of each nutrient and present to the user so that he can check if he is keeping up with his goal.





User enters the details of a food item navigating the filter options as well. He is then asked if he wishes to sum up his food intake.

```
Press Y if you wish to add your food intake as well, press any key to exit: y
Enter the number of food items you wish to add for checking your intake: 1

Enter Product ID: 09177
Enter serving size: 1
Name: Mangosteen, canned, syrup pack
Product ID: 09177
Serving size: 1.0
Protein 0.41
Total lipid (fat) 0.58
Carbohydrate, by difference 17.91
Vitamin A, IU 35.0
Vitamin D (D2 + D3) 0.0
Calcium, Ca 12.0
Magnesium, Mg 13.0
Iron, Fe 0.3
Potassium, K 48.0
Copper, Cu 0.069
Zinc, Zn 0.21
Energy 73.0
Fiber, total dietary 1.8
Cholesterol 0.0
Vitamin B-12 0.0
Vitamin C, total ascorbic acid 2.9
Sugars, total 0.0
Water 80.94

Total Content

Protein 3.28
Total lipid (fat) 1.52
Carbohydrate, by difference 59.92
Vitamin A, IU 4328.0
Vitamin D (D2 + D3) 0.0
Calcium, Ca 44.0
Magnesium, Mg 40.0
Iron, Fe 0.64
Potassium, K 672.0
Copper, Cu 0.444
Zinc, Zn 0.36
Energy 240.0
Fiber, total dietary 6.4
Cholesterol 0.0
Vitamin B-12 0.0
Vitamin C, total ascorbic acid 145.6
Sugars, total 54.64
Water 161.88
```

Figure 27: Summing up user food intake function

On confirming the option to sum up his food intake, the user later enters the details for another food item. The program computes the sum of each individual nutrient and displays it to the user.

2) A user wishes to see the nutritional facts of a food item

The user can search for any food item by providing the exact name or a part of the name. The program will fetch all the food items that match the input provided by the user. User can also choose filters that can be applied while searching for the food item, or can be used to sort the entire data based on a nutrient. Below are the filters provided:

- High in calories
- Low in calories
- High in proteins
- Low in fats
- High in fiber

5.4 Suitable examples

Case 1: If a user wants to sort the data based on high to low calorie content, then this can be done using the following query:

result=collection.aggregate([{'$unwind':"$nutrients"},{'$match':{"nutrients.nutrient": "Energy"}},{'$sort':{"nutrients.gm":pymongo.DESCENDING}},{'$limit':50}])

The nutrient "Energy" is matched and sorted in the descending order. The number of records are also restricted by specifying the $limit parameter

```
C:\Python34>python search_and_sort.py
If you wish to view food items on basis of filters then press Y, else press any
key to search for a food item: Y
Press a for High calorie content
Press b for Low calorie content
Press c for High protein content
Press d for High fibre content
Press e for Low fat content a
Name: Fat, mutton tallow
Product ID: 04520
Serving size: 1.0 tbsp
Energy 902.0 gms

Name: Fish oil, sardine
Product ID: 04594
Serving size: 1.0 tbsp
Energy 902.0 gms

Name: Lard
Product ID: 04002
Serving size: 1.0 tbsp
Energy 902.0 gms

Name: Fish oil, salmon
Product ID: 04593
Serving size: 1.0 tbsp
Energy 902.0 gms

Name: Fish oil, menhaden
Product ID: 04591
Serving size: 1.0 tbsp
Energy 902.0 gms

Name: Fat, beef tallow
Product ID: 04001
Serving size: 1.0 tbsp
Energy 902.0 gms

Name: Fish oil, herring
Product ID: 04590
Serving size: 1.0 tbsp
Energy 902.0 gms

Name: Fish oil, menhaden, fully hydrogenated
Product ID: 04592
Serving size: 1.0 tbsp
Energy 902.0 gms

Name: Fish oil, cod liver
Product ID: 04589
Serving size: 1.0 tsp
Energy 902.0 gms

Name: Fat, chicken
Product ID: 04542
Serving size: 1.0 tbsp
```

Figure 28: Filtering out food by high calorie content

The user selects the high calorie content filter on all the food items.

Case 2: If a user is searching for a food and he wants to sort the result based on low to high fat content, this is done using the following query:

```
result=collection.aggregate([{'$match':{"name":{'$regex':food_name,'$options':'i'}}},
{'$unwind':"$nutrients"},{'$match':{"nutrients.nutrient_id":"204"}},{'$sort':{"nutrient
s.gm":pymongo.ASCENDING}}])
```

The food name specified by the user is stored in the variable 'food_name' and is searched across the database using the regular expression parameter $regex .The nutrient "Fat" is matched by its nutrient ID and sorted in the ascending order

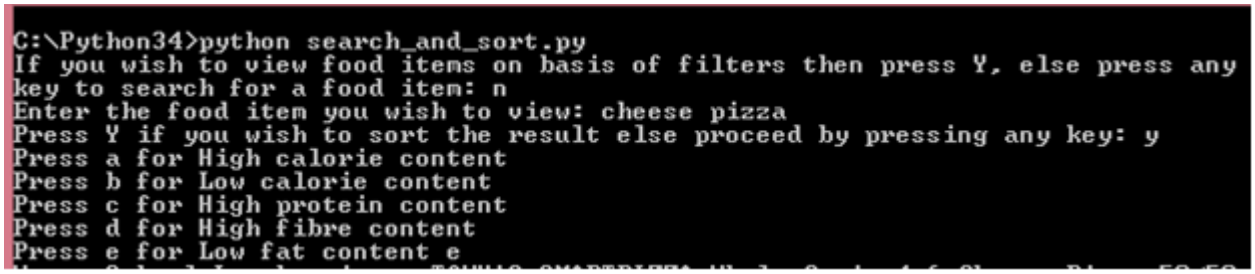


Figure 29: Searching Document Store by keywords

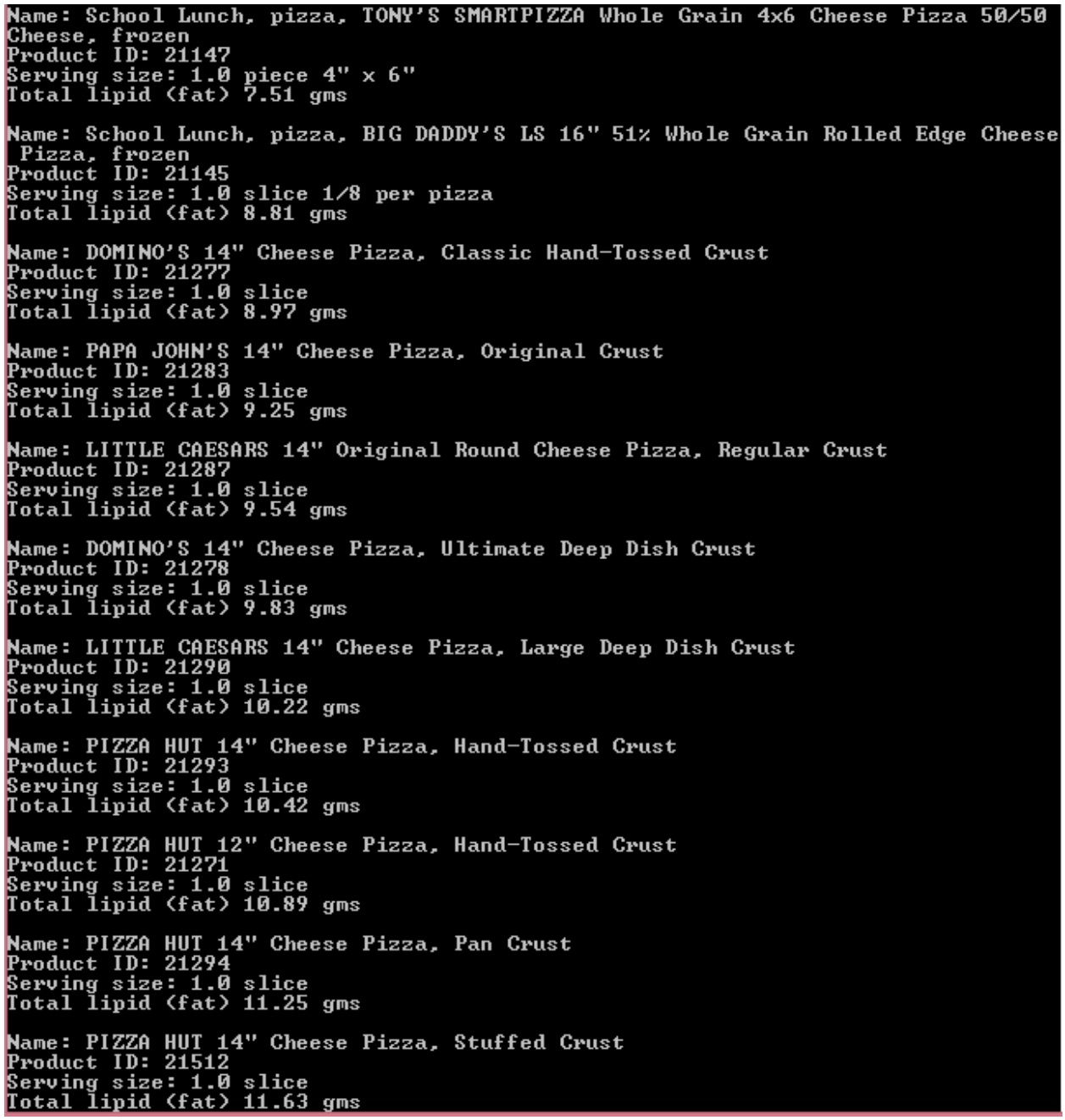


Figure 30: Result set of searching Document Store by keyword

```
Name: PIZZA HUT 12" Cheese Pizza, Pan Crust
Product ID: 21272
Serving size: 1.0 slice
Total lipid (fat) 12.56 gms

Name: PIZZA HUT 14" Cheese Pizza, THIN 'N CRISPY Crust
Product ID: 21295
Serving size: 1.0 slice
Total lipid (fat) 12.8 gms

Name: PIZZA HUT 12" Cheese Pizza, THIN 'N CRISPY Crust
Product ID: 21273
Serving size: 1.0 slice
Total lipid (fat) 14.1 gms

Name: DOMINO'S 14" Cheese Pizza, Crunchy Thin Crust
Product ID: 21279
Serving size: 1.0 slice
Total lipid (fat) 15.1 gms

Name: PAPA JOHN'S 14" Cheese Pizza, Thin Crust
Product ID: 21286
Serving size: 1.0 slice
Total lipid (fat) 15.66 gms

Name: LITTLE CAESARS 14" Cheese Pizza, Thin Crust
Product ID: 21292
Serving size: 1.0 slice
Total lipid (fat) 16.99 gms
```

Figure 31:Searching Document store by multiple filters

```
Enter the product ID from the above list who's details you wish to see: 21512
Enter your serving size: 1

Displaying the Calorie, Protein and Carbohydrate content per 100gm of:  21512

Name: PIZZA HUT 14" Cheese Pizza, Stuffed Crust
Product ID: 21512
Serving size: 1.0

Energy 274.0
Protein 12.23
Carbohydrate, by difference 30.0

If you wish to see all the nutrient contents then press Y, press any other key t
o exit: y

Name: PIZZA HUT 14" Cheese Pizza, Stuffed Crust
Product ID: 21512
Serving size: 1.0

Protein 12.23
Total lipid (fat) 11.63
Carbohydrate, by difference 30.0
Vitamin A, IU 511.0
Vitamin D (D2 + D3) 0.0
Calcium, Ca 238.0
Magnesium, Mg 22.0
Iron, Fe 2.04
Potassium, K 229.0
Copper, Cu 0.081
Zinc, Zn 1.49
Energy 274.0
Fiber, total dietary 1.7
Cholesterol 0.0
Vitamin B-12 0.4
Vitamin C, total ascorbic acid 3.1
Sugars, total 2.9
Water 43.37
```

Figure 32: Result of searching Document Store by filter

By default, calories, proteins and carbohydrates for the selected food item are displayed to the user. The user then can opt whether he wishes to view all the other nutrients as well.

3) A user wants to add a food item not present in the current database.

The user will be asked to provide the details of the food item, which includes:

- The name of the food
- The serving size of the food
- The number of nutrients and their nutrient content

These values will be first validated for any irregularities by us and then inserted into the database.

The insert is then performed by using the following function:

result=collection.insert_one(food)

Here, Food is the user-defined dictionary, which will be created programmatically by asking him to input the food item as well as the nutrient details

Figure 33: Adding a new food item by user

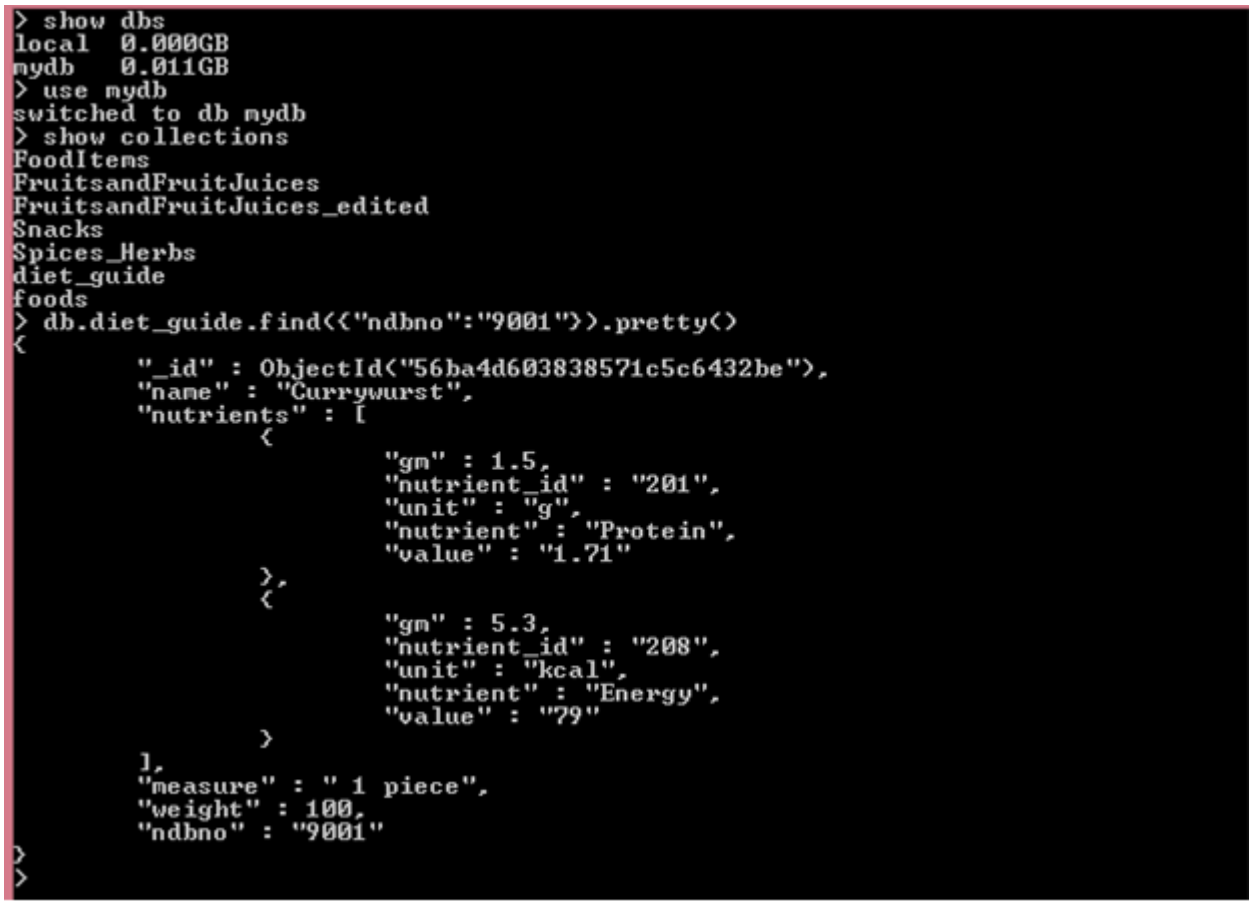


Figure 34: Validation of adding new food items

4) A user wants to update specific nutrient values for any food item

The user will be asked to provide the details of the food item, which he wishes to modify. This request will be first validated for any irregularities by us and then updated into the database. The update is then performed by using the following query:

result=collection.update({"ndbno":product_id,"nutrients.nutrient_id":nutrient_id},{'$set':{"nutrients.$.gm":nutrient_value}})

The food ID and the nutrient ID provided by the user will be saved under the variables product_id, nutrient_id respectively. $set operator is used to update the specific field and the new value provided by the user will be saved under the variable nutrient_value.

```
C:\Python34>python update.py
Enter the product_id: 11980
Name: Peppers, chili, green, canned
Product ID: 11980
Serving size: 1.0 cup

ID: 203 , Protein : 1.0
ID: 204 , Total lipid (fat) : 0.27
ID: 205 , Carbohydrate, by difference : 4.6
ID: 318 , Vitamin A, IU : 126.0
ID: 328 , Vitamin D (D2 + D3) : 0.0
ID: 301 , Calcium, Ca : 36.0
ID: 304 , Magnesium, Mg : 4.0
ID: 303 , Iron, Fe : 1.33
ID: 306 , Potassium, K : 113.0
ID: 312 , Copper, Cu : --
ID: 309 , Zinc, Zn : 0.09
ID: 208 , Energy : 21.0
ID: 291 , Fiber, total dietary : 1.7
ID: 601 , Cholesterol : --
ID: 418 , Vitamin B-12 : --
ID: 401 , Vitamin C, total ascorbic acid : 34.2
ID: 269 , Sugars, total : --
ID: 255 , Water : 93.25
```

Figure 35: Modifying Food Items

The user is asked to follow the above menu driven program in order to enter the details pertaining to the food item needed to be modified.

```
Which nutrient_id you wish to change: 203
Enter the new value: 0.72
Record updated

Name: Peppers, chili, green, canned
Product ID: 11980
Serving size: 1.0 cup

Protein 0.72
Total lipid (fat) 0.27
Carbohydrate, by difference 4.6
Vitamin A, IU 126.0
Vitamin D (D2 + D3) 0.0
Calcium, Ca 36.0
Magnesium, Mg 4.0
Iron, Fe 1.33
Potassium, K 113.0
Copper, Cu --
Zinc, Zn 0.09
Energy 21.0
Fiber, total dietary 1.7
Cholesterol --
Vitamin B-12 --
Vitamin C, total ascorbic acid 34.2
Sugars, total --
Water 93.25
```

Figure 36: Result of food item modification

The highlighted field shows the value provided by the user is updated successfully.

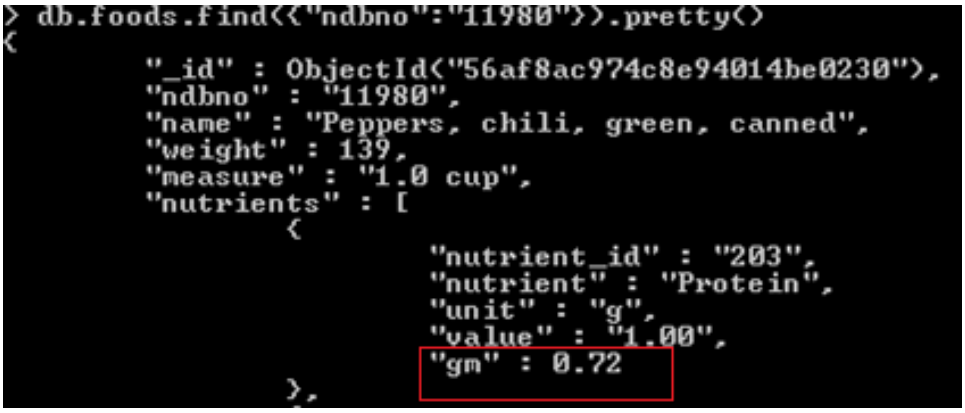


Figure 37: Result of food item after modification being displayed in search results

The updated record is reflected in the database as well.

6. Graph Database

6.1 Introduction

Graphs are the most efficient and natural way of working with data. The graph database leverages one of the most important aspects of NOSQL databases: agility. It provides an ideal platform to integrate data coming from disparate sources, and make relationships among those data to arrive at a best-fit solution.

Graph databases store and represent information with the help of nodes, relationships and properties. The properties are in the form of key value pairs and can be stored in nodes a well as relationships. As nodes can be related to each other with the help of a relationship there is no need of foreign keys or out of band processing such as Map Reduce to infer data connections.

The NOSQL graph database Neo4j, viz. *Personalized Diet Planner*, will be used in Personalized Nutritionist application to integrate data from Key-Value and Document data stores, create meaningful relationships among the datasets and arrive at a solution to meet users' goals.

6.2 Role Of Personalized Diet Planner

The objective of graph database is to create diet plans for users of Personalized Nutritionist application. This is achieved by integrating users' data residing in Key-Value database, and food and nutrition data residing in Document database. Based on the user's body type and goal, a diet plan is created that comprises of food items with adequate calorie and macronutrient content.

Having data coming from two disparate NOSQL data stores means that the structure of data is evidently discrete. Linking these data and generating insight from it demands that these data be highly interrelated. Though it can be achieved by traditional Relational Database Management systems (RDBMS) but it will be highly complex and query processing will be quiet slow. Therefore, using a graph database provides us an efficient way to relate and query on the stored data.

6.3 System Architecture

The user personal data, such as height, weight, age, gender etc., and food intake details are retrieved from Redis(Key value Database) using *Redis-py* Python library.It is then processed and then stored in the form of nodes and relationships.

Foods and their nutritional values are fetched from Mongo (Document Database) using *Pymongo* Python library, and based on the categories of food made in Neo4j,a subset of food items is stored and related appropriately.

The graph database (Neo4j) simultaneously creates relationships among user and food item nodes and gives appropriate analysis to the application layer, which then provides detailed information of their diet plans to the users.

The below figure shows the system architecture for Personalized Diet Planner subsystem.

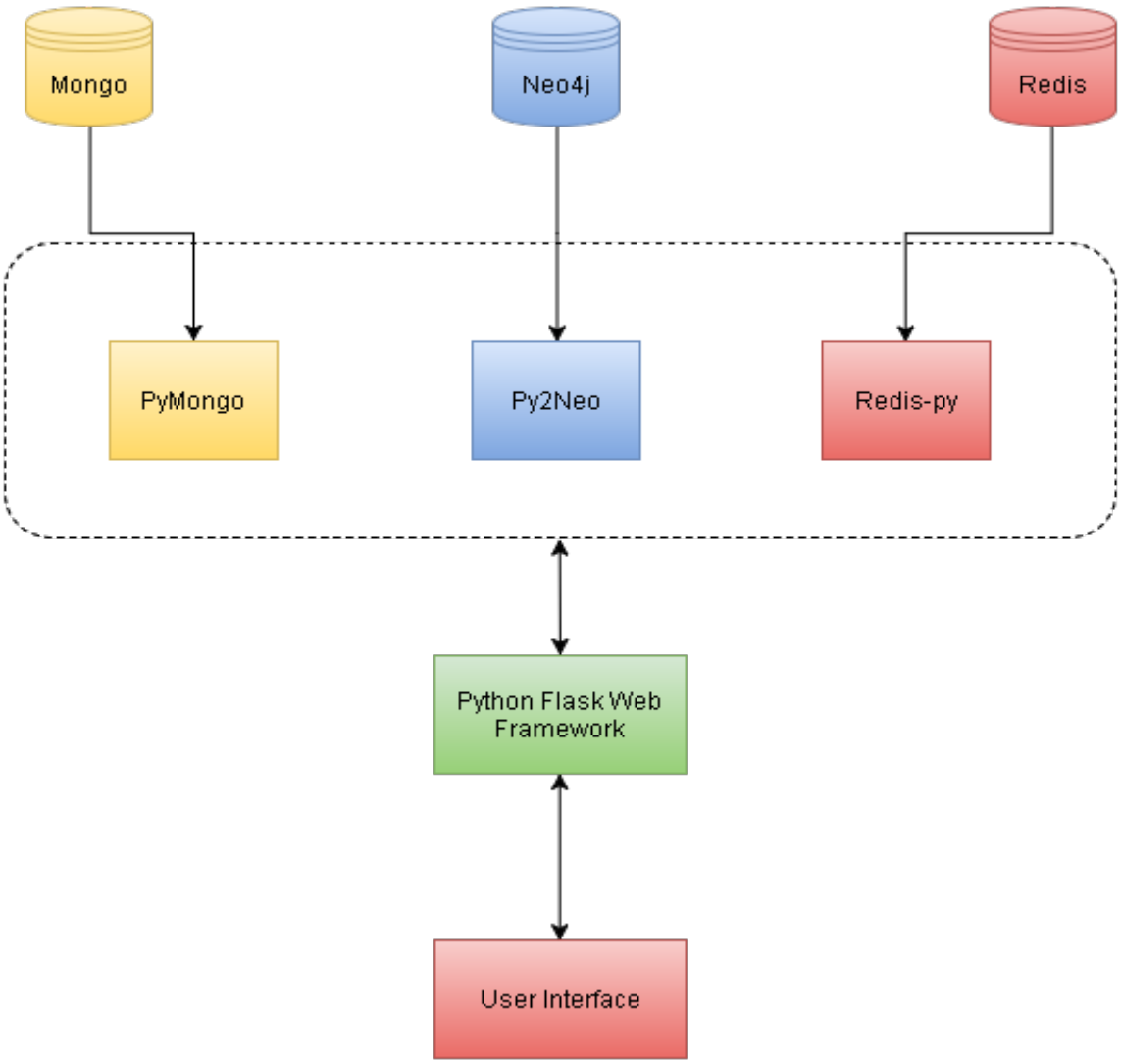


Figure 38 System Architecture

The below screen capture shows code containing details of connection to Redis, MongoDB and FLASK application.

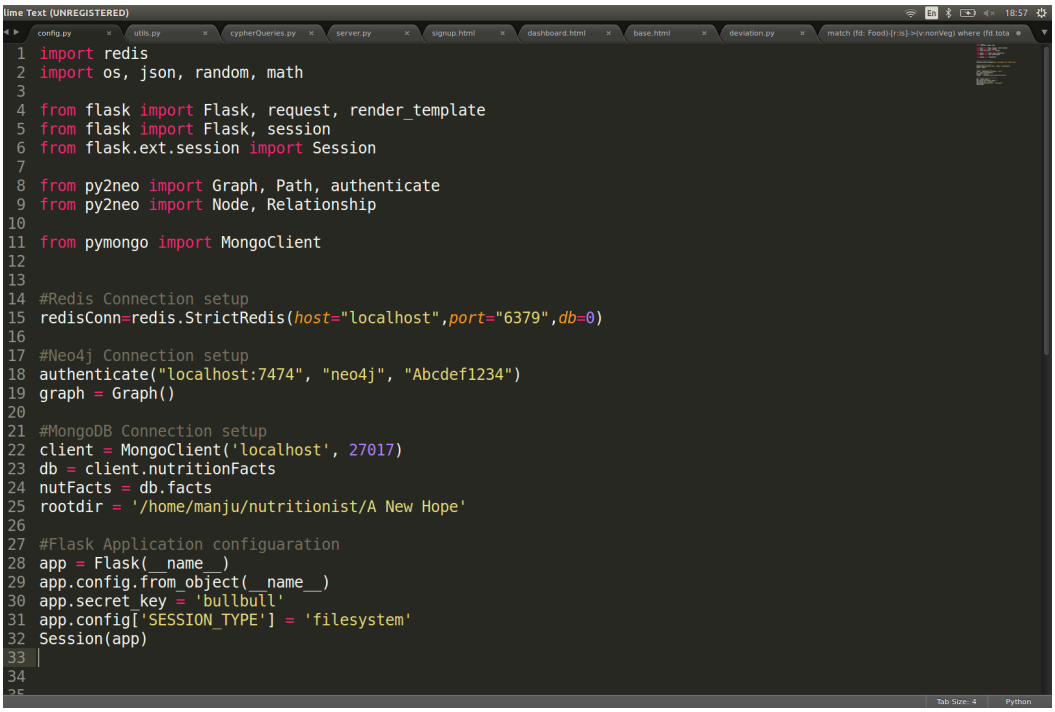


Figure 39Code snippet showing connection details

6.4 Data Model

The below diagram shows the data model for Personalized Diet Planner.

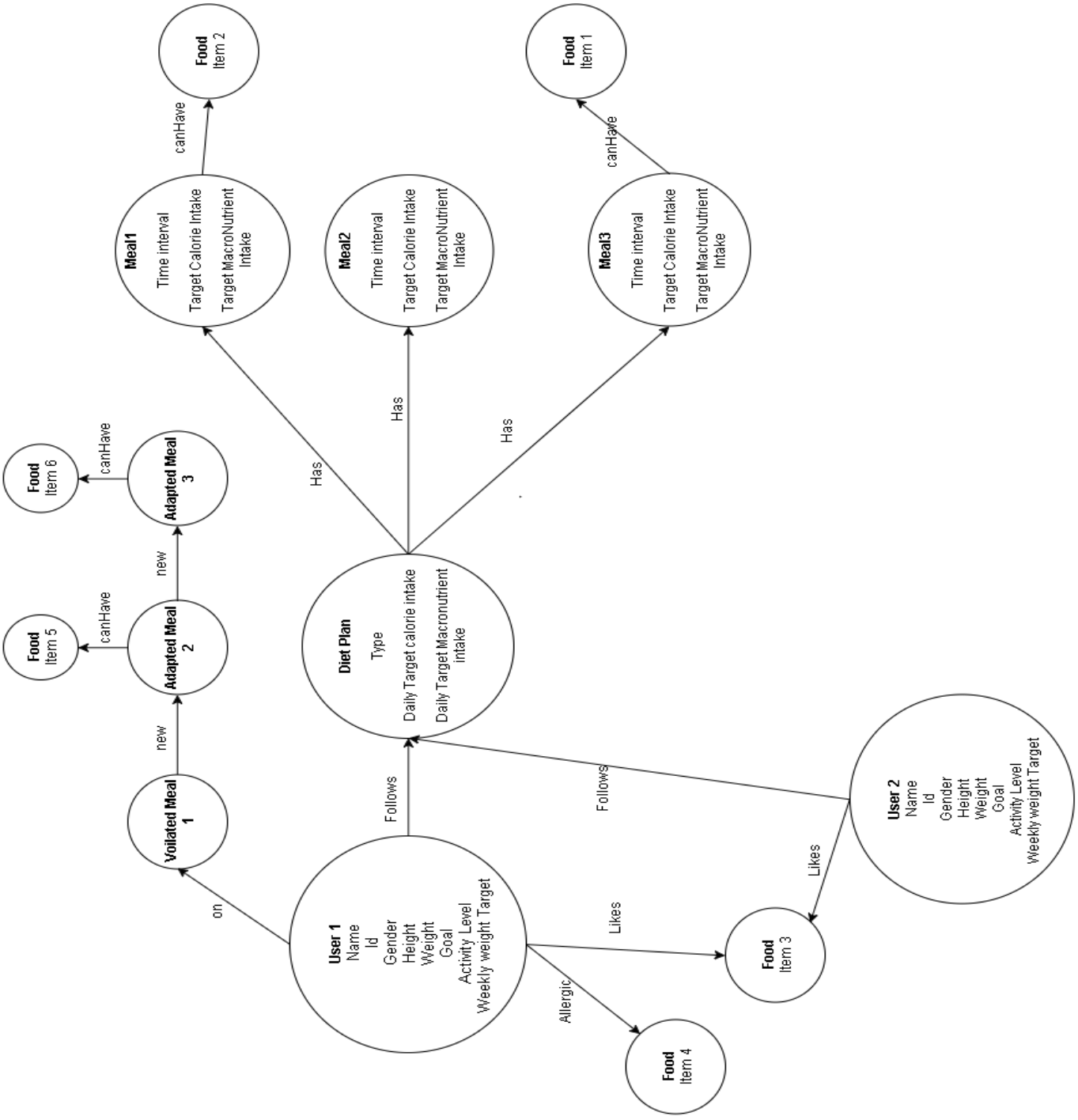


Figure 40Graph data model

Nodes

User node:

- Name
- Id
- Age
- Gender
- Height
- Weight
- Goal: lose weight, gain weight or maintain weight
- Activity Level
- Weekly Weight Target (Only applicable for weight loss and weight gain plans)

Diet Plan node:

- Type
- Daily Target calorie intake
- Daily Target Fat Intake
- Daily Target Maximum Carbohydrate Intake
- Daily Target Minimum Carbohydrate Intake
- Daily Target Maximum Protein Intake
- Daily Target Minimum Protein Intake

Meal nodes:

- Suggested time interval
- Target calorie intake
- Target Range of Carbohydrate intake
- Target Range of Protein intake
- Target Fat intake

Food Item nodes:

- Food item ID
- Food item name
- Calorie content
- Carbohydrate content
- Protein content
- Fat content

Relationships

- USER-[*FOLLOWS*]->DIET PLAN
- USER-[*ON*]->VOILATED MEALS
- USER-[*LIKES*]->FOOD ITEM
- USER-[*ALLERGIC*]->FOOD ITEM
- DIET PLAN-[*HAS*]->MEAL TIME INTERVALS
- MEAL TIME INTERVALS-[*OPTION*]->FOOD ITEM

6.5 Functionality Of Personalized Diet Planner

The following are the operations performed in graph database based on the inferences drawn from the graph data model in order to meet the use case requirements.

COMPUTATION OF DAILY CALORIE INTAKE: It is calculated using the user information and standard formulas [1].

COMPUTATION OF MACRONUTRIENT INTAKE: The adequate amount of macronutrients i.e., carbohydrates, proteins and fat required for the user to reach his goal is computed.

CREATION OF DIET PLAN: The application queries neo4j to match any existing diet plan that suits the user's requirements and goal. If not found any match, then a new diet plan is created for the user, which involves the following steps, which are automated in the application.

1. Create a node called dietPlan with the total calorie intake. Relate the user to this newly created node with the relationship "follows".
2. Create 3 nodes related to dietPlan called Meal1, Meal2 and Meal3 with the relationship "has". And the nutritional values for each meal in the node's respective properties.
3. After this the application queries the mongoDB to find all the suitable foods which satisfy the nutritional requirements for each of the meals and import these foods to neo4j as nodes and create a relationship "canHave" to Meal1, Meal2, Meal3 nodes respectively.

SUGGESTION OF FOODS: The user is required to interact with the application to update likes on some particular foods. This helps the application to better suggest foods for the user based on the user's likes and common likes of other users.

Also foods can be suggested based on popular foods among people following a particular type of diet plan.

ADAPTIVE DIET PLANS: The user is expected to update the food intake regularly so as to keep check on the user's diet plan and to check for any deviation that the user might have made from the diet plan. In case the user did make a deviation from the diet plan the

application creates a new set of nodes directly related to the person node which has a temporary diet plan just to accommodate the deviation.

CALCULATION OF USER PUNCTUALITY: User can see how successfully he/she is following the diet plan.

6.6 CRUD Operations

Most of the CRUD operations were executed using the python Application Programming Interface **Py2Neo**and the results below are shown using the **Neo4J Web Admin Interface**. CRUD operations that were implemented for the project to fulfill the user stories as per the use case are given below:

User Story #1

As a user, I want to follow a diet plan, not created by an advisor, so that I can opt for an energetic life style.

The user details such as Height, Weight, Gender, Age and Activity level are used to find out a suitable diet plan so that user can keep up with his goal.

Goal could be different for different users and can be broadly divided into three cases (Weight Loss/Mass Gain/Maintain Weight). Each case is further divided into two parts that is **searching for a suitable diet plan** and if no matching diet plan is found a **diet plan is created** for him using a set of standard formulas [1].

To reduce redundancy in the document only one case is described as it is same for the rest.

Searching for a diet plan - Weight loss

System automatically tries to relate the user with different users with the same parameters and goal requirement. If a suitable diet plan is found, then he is linked with the best-fit diet plans according to the average success rate of the diet plan.

Query:

```
match(p:person)-[:follows]->(d:dietPlan) whered.type CONTAINS ‘weightLoss’
andp.height=167.0andp.weight=90.0andp.age=22andp.gender=’F’
Return d,p
```

Creation of a diet plan

If no matching user with the same parameter is found then a new diet plan, with all the meal nodes attached, is created.

Query for creation of diet plan, part of the web application code base.

```
    x ="match (p:person) where p.id='"+_id+
"' create (d:dietPlan {name : 'DietPlan', type:'"+goal+
"' ,calorie:"+str(calReq)+",ProtienMax:"+
str(maxProtien)+",ProtienMin:"+str(minProtien)+
",CarbMax:"+str(maxCarb)+",CarbMin:"+str(minCarb)+
",FatReq:"+str(reqFat)+"}),(m1:Meal1 {name : 'Breakfast',
type:'breakfast',cal:"+str(calReq/3)+
",ProtienMax:"+str(maxProtien/3)+",ProtienMin:"+str(minProtien/3)+
",CarbMax:"+str(maxCarb/3)+",CarbMin:"+str(minCarb/3)+
",FatReq:"+str(reqFat/3)+"}),(m2:Meal2 {name : 'Lunch',
type:'lunch',cal:"+str(calReq/3)+
",ProtienMax:"+str(maxProtien/3)+",ProtienMin:"+str(minProtien/3)+
",CarbMax:"+str(maxCarb/3)+",CarbMin:"+str(minCarb/3)+
",FatReq:"+str(reqFat/3)+"}),(m3:Meal3 {name : 'Dinner',
type:'dinner',cal:"+str(calReq/3)+
",ProtienMax:"+str(maxProtien/3)+",ProtienMin:"+str(minProtien/3)+
",CarbMax:"+str(maxCarb/3)+",CarbMin:"+str(minCarb/3)+
",FatReq:"+str(reqFat/3)+"}),(p)-[:follows]->(d),(d)-[:has]->(m1),(d)-
[:has]->(m2),(d)-[:has]->(m3) return d"

return graph.cypher.execute(x)
```

Diet Plan

Username	manju
Calorie	2055.0
Min Carbs	231.1875
Max Carbs	308.25
Min Protiens	56.0
Max Protiens	70.0
Fats	68.5

Figure 41: Final Diet plan being shown in the web application to the user (Note: the above image is not of the query executed above as they were taken at different times)

User Story #2

As a user, I want suggestions of food items based on my likes and allergies so that I can enjoy my food with convenience.

The food suggestions will be based on food likings of people liking a particular food. For example, consider user A and B. Both users like Dumplings. But user A also likes eggs so eggs will be shown as a suggestion to user B.

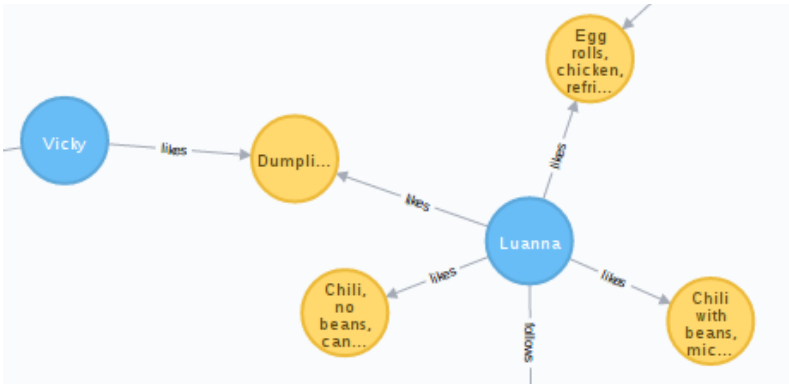


Figure 42: Visualization for the example given above

And If the user is allergic to particular food than those foods won't be shown to him.

Query:

```
match(p:person{id:2})-[:likes*2..3]-(f:food) where not (f.name CONTAINS
'chicken'orf.name CONTAINS 'Chicken') return f
```

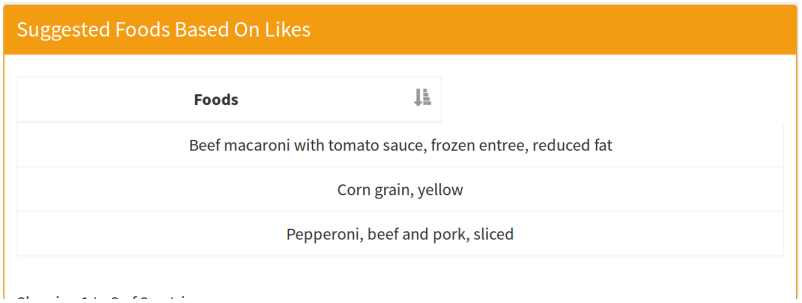


Figure 43: Food suggestions based on likes and allergies of the user in the web application

User Story #3

As a user, I want the food according to my food habit but which might differ from the suggestion from the system so that I can opt for it without any trouble.

The diet plan for a particular day can be **adaptive** to what user eats. This adaptation can be achieved taking the meal from the user and adjusting the subsequent meals.

The user is asked to update his food intake on daily basis on the web interface.

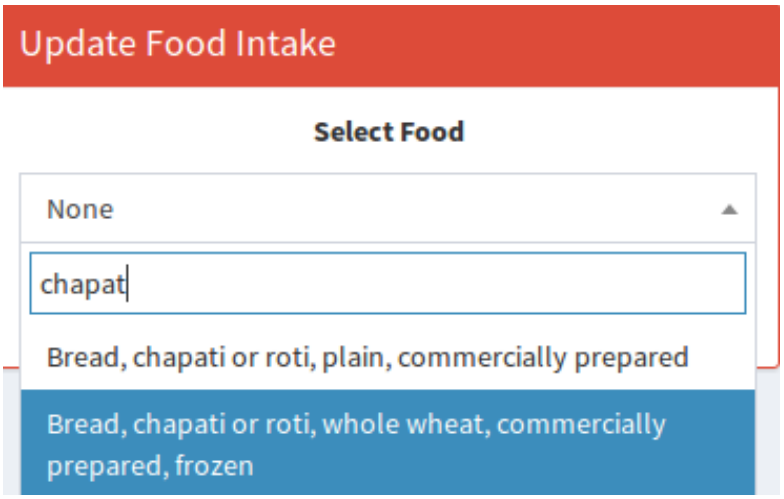


Figure 44: user entering his food intake

Based on the food intakes he enters mongo is queried with the amount of nutrients are there in that particular food and according to the time of day an adaptive diet plan is created and attached to the user node.

Query:

```
match(p:person{id:1}) create(Meal1:Meal1{time:'411',cal:1200.0,
```

```
ProtienMax:23.0,ProtienMin:10.0,CarbMax:23.0,CarbMin:10.0,FatReq:20.0})
create(Meal2:Meal2{time:'11-17',cal:1258.00625,ProtienMax:12.5,
ProtienMin:25.0,CarbMax:197.525703125,CarbMin:273.7009375,FatReq:34.6223611
11}) create(Meal3:Meal3{time:'17-4',cal:1258.00625,ProtienMax:12.5,
ProtienMin:25.0,CarbMax:197.525703125,CarbMin:273.7009375,FatReq:34.6223611
11}) create(p)-[on:on{date:'8/2/2016'}]->(Meal1) create(Meal1)-[new:new]-
>(Meal2) create(Meal2)-[:new]->(Meal3)
```

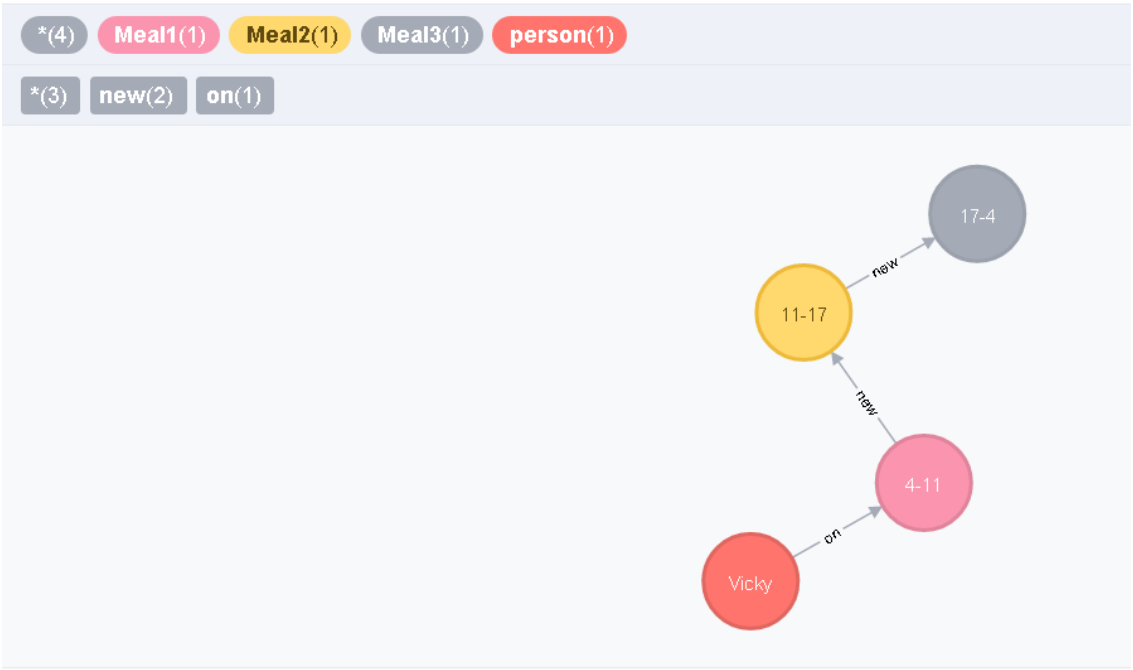


Figure 45:Visual representation of adaptive diet plan on Neo4j web console

User Story #4

As a user, I want to know am I following the diet Plan properly.

Based on the number of times the user has deviated from the diet plan his punctuality can be calculated based on the formula given below

$$\text{Punctuality/Success rate} = \left(1 - \left(\frac{numberofdeviations}{totalnumberofdays}\right)\right) \times 100$$

Number of deviations: Total no. of days the user didn't follow the diet plan

Total number of days: number of days that have passed since the user started using the diet plan(data needs to be fetched from Redis).

Query:

```
match(p:person{id:1})-[date:on]->(neighbors)return
(1-count(neighbors)/10.0)*100.0
```

```
enter your id: 1
match(p:person{id:1})-[:on]->(n) return (1.0-(count(n)/10.0))*100.0, p
name: Christina
Punctuality: 50.0 %
```

Figure 46 Program displaying the punctuality of a user in following a diet plan

User Story #5

As a user I want to see what foods are popular among other users of same goal as mine so that I can select foods effectively.

Foods that are more liked by the users having same diet plan goal can be counted and shown in the descending order.

Query:

```
match(f:food)<-[r:likes]-(p:person)-[:follows]-(d:dietPlan)
whered.type='weightLoss' return count(r),f.name,d.typeORDER BYcount(r) DESC
```

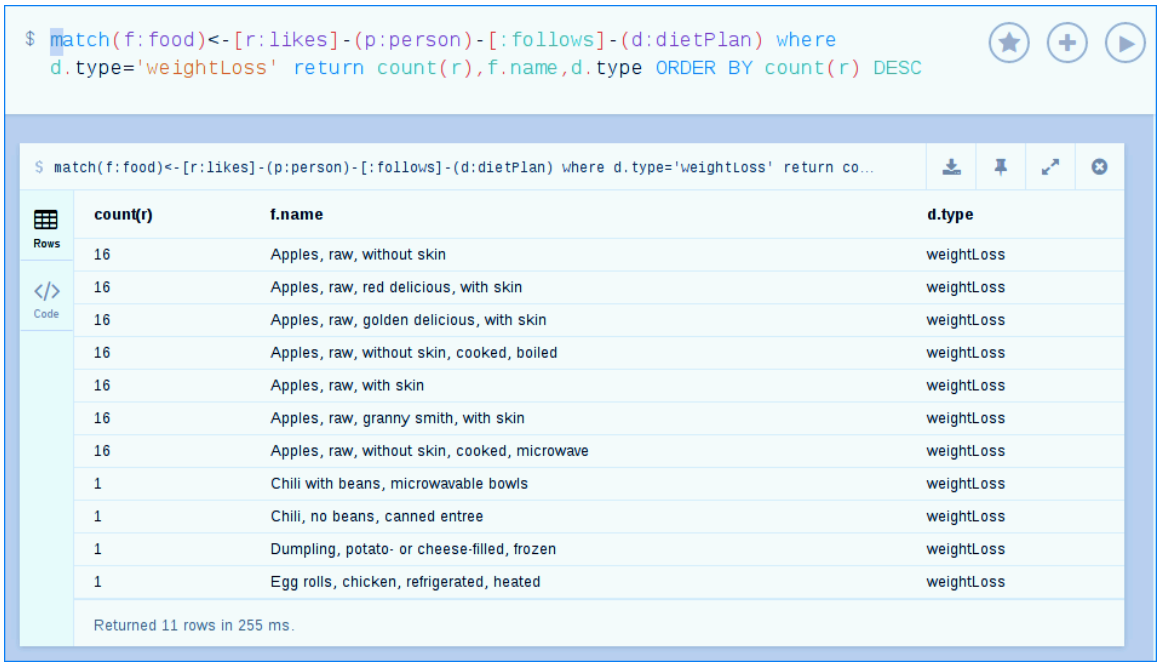


Figure 47: Displaying popular food count in descending order in the Neo4j web console

7. Integration

To implement our project in full extent required integration of the 3 databases into one complete system. Each database holds different aspects of the project which when combined gives us the complete picture of the various use cases and their implementation. A simple analogy to explain the relationship among the three databases: The document store, graph and key/value databases are the kitchen, mother and child in a regular household. The kitchen (document store) holds the food items. The mother (graph) uses the kitchen to make recipes and diets for the child(user for key/value). The more consistent these 3 systems are with each other, more efficient and better results will be yielded. Also, since the child (user) can also use the kitchen(document store) directly for food, this need has also been catered to.

7.1 Document Store

The user can search through our database for foods that he has consumed in a day. This required an exchange of information between the key/value and document store where in the document store will provide a user with the nutritional facts about various foods he searches for and the key/value will in return provide us with the list of foods a user has selected. Based on this information, we will add the nutritional contents of the foods and present to the user the total amount of each nutrient he has consumed in a day. This information will help the user to keep track of his goals and progress.

The crux of our project lies in the diet plans and nutrition charts prepared by the graph database. The diets will be specific to a user and to successfully implement a diet, the graph database will analyze the data in the document store based on the nutritional content of a food and user goals. Keeping these two things in mind, the document store provides a flexible search option where in the food values can be sorted based on a nutrient. Another key feature of the searching is that it can provide the food which will help the user reach his goals based on how much he has already consumed and what part of his nutrient intake remains. These two functions will help the graph in drawing out an efficient and more user specific diet and food suggestions.

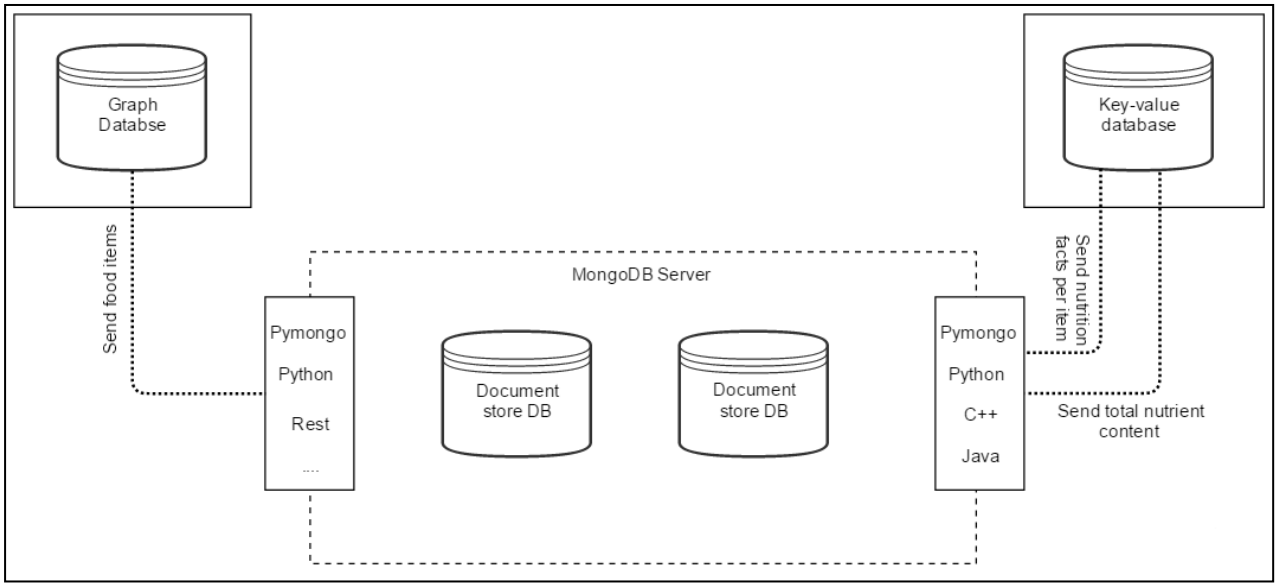


Figure 48: Document Store Integration Diagram

Function	Provider/Beneficiary	Details
Send food items	Document store/ Graph and Key value database	Share a list of all food items requested
Share nutrition facts	Document store/ Graph and Key value database	Share a list of all nutritional facts requested
Aggregate food intake	Document store/ Key value	Aggregate the user daily intake of his food items. The calculated value will be sent to the Key Value DB in aggregated and broken down format.
Add new food	Document store	Add a new food item in the database
Update existing food	Document store	Update an existing food item detail if it is found to be incorrect/old

Table: Document Store implemented functions

7.2 Key-Value Database

Key-Value database plays a core role in the rack of NoSql technologies, as it is the main repository for the user data, which the application revolves around.

From the perspective of Key-Value, Integration is an essential factor in bringing the solution pieces together.

Graph database relies on Key-Value data to fetch users' properties, like, dislikes, goals, allergies, restrictions, and food habit as it needs to land users on their best fit diet plan- an essential functionality of Personal Nutritionist.

On the other side, Document store database- the central repository for Nutrition Facts- is essentially integrated with Key-Value DB for several functionalities, comes on top of which aggregating and breaking down user's daily food intake in term of caloric sum, and nutritional parts such as protein, vitamins, and etc.

Figure below shows how Key-Value cluster serves both read and write services to what's seen to it in this case as OLTP applications, Document Store and Graph modules.

As can be seen in the Figure, Key-Value DB is communicated with relying on several supported clients.

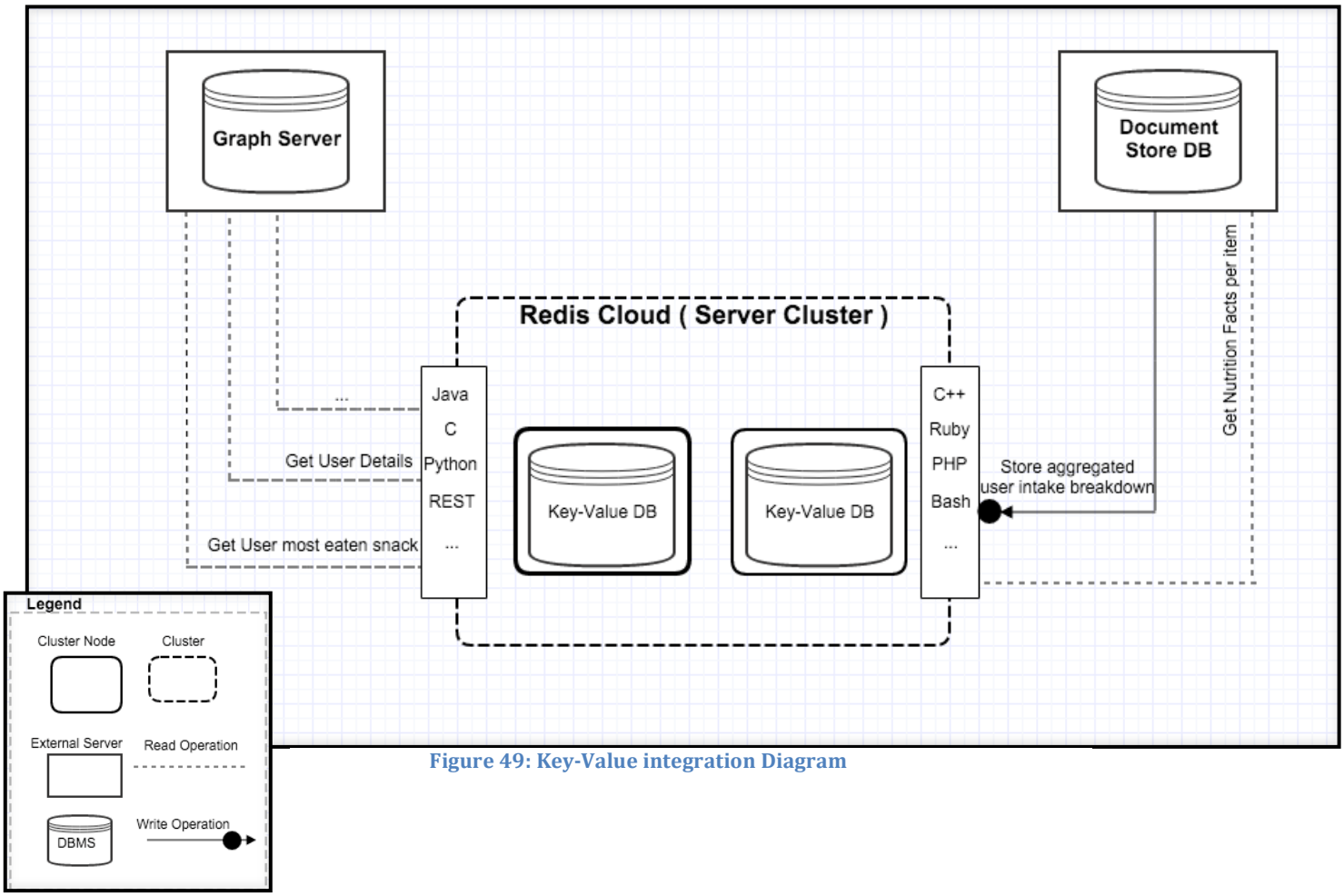


Figure 49: Key-Value integration Diagram

Exposed services of Key-Value module

Service	Argument	Type /Direction	Provider/Beneficiary	Details
Get User Details	User ID	Read/In-bound	Document Store | Graph	Fetch user basic details such as name, age, height, allergies, and etc.
Get Weight	User ID	Read/In-bound	Graph	Fetch list of a particular users recorded weight
Get Food intake	User ID, time period	Read/In-bound	Document Store | Graph	Fetch list of all food intake in a given period for a given user
Get User Meal habit	User ID, Meal, Time Period	Read/In-bound	Graph	Fetch list of most eaten food items by a user for a certain meal, such as most eaten snack.
Aggregate Food Intake	User ID, Food item List [item id, quantity, date]	Write/Out-bound	Document Store	Aggregated User daily intake shall be calculated at the document store side and saved at the Key Value DB in aggregated and broken down format

Table 4: Exposed services of Key-Value module

7.3 Graph Database

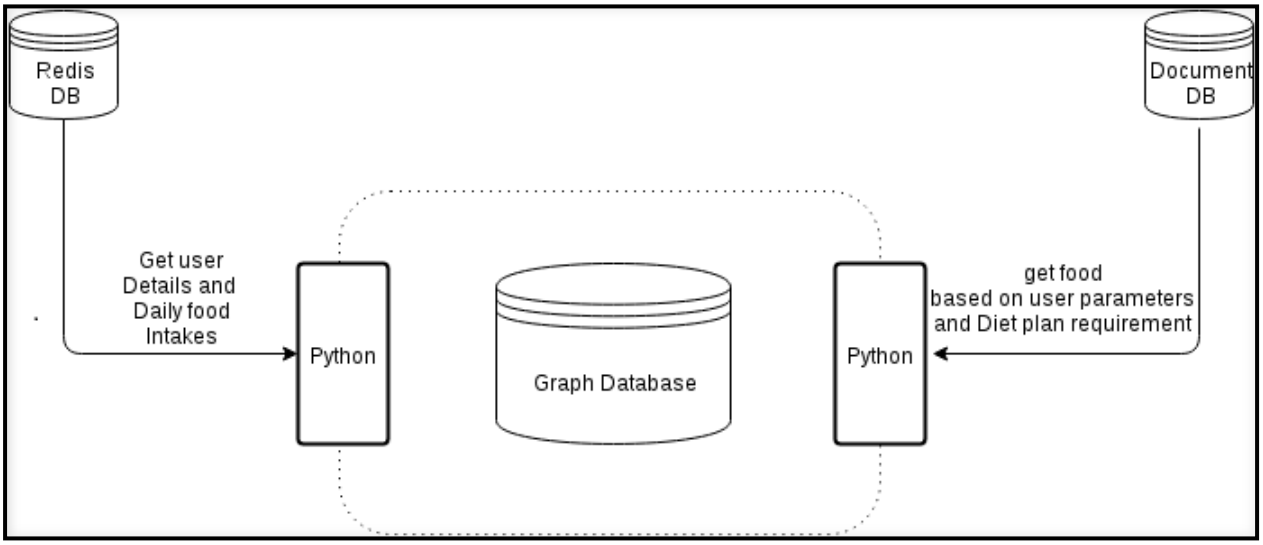


Figure 50: Graph Data Base

The Graph database(neo4j) interacts with the Redis database in order to get user specific data and food intakes of user. It then creates the respective nodes and relationships according to the data received .

Further it interacts with Mongo database to retrieve the food items according to the nutrition facts of the user's diet plan. Neo4j stores the selected food item in the form of nodes and relationships.

As python Application Programming Interface, hereafter referred as API, Py2neo is used to interact with Neo4J. Therefore, Graph database connects with other databases using following python APIs :

Mongo database with Pymongo

Redis database with Redis-py

8. Evaluation

Considering the amount of time and money people spend in keeping their health in check, our aim was to provide them with accurate and effective solutions on the fly with our personal nutritionist application. Implementation of this project was a challenging task since there were a lot of parameters to take into account before we could present a proper solution to the user. Also, we had a time constraint on our head due to which we had to scale down the concept to some extent.

NoSQL was a new concept for most of us. Grasping it in time and then implementing it was the most rewarding outcome of this project. Each group was provided with the database which was tailor made for their requirement. Key-value was the optimal choice for creating user profiles, while the document store was used as a large schema less data store and finally the graph database was adept in analyzing and drawing out relations between the user, his goals and the food data.

The application has come a far way from its realization and can now be considered as a personal nutritionist who takes your current health facts, level of activity, fitness and goals to make a diet charts with suggestive food items. The user is also provided with an optimized search to display results according to his goals and the graph can make relationships based on user's preferences (from key-value) and also among users with similar profiles and goals. Personal likes and dislikes will also help the graph in scrutinizing the diets and food suggestions.

8.1 GANTT Chart

A Gantt chart, frequently used in managing projects, is one of the most useful ways of presenting the tasks or events of a project exposed in contradiction of time. It provides a graphical vision of a timetable or agenda that helps in planning, directing, organizing and tracking specific tasks in a project.

Creating a Gantt chart was our first step in this project. It has helped us a lot along the whole period of our task, not just in terms of management; but in terms of evaluation as

well. It has been used in testing the level of our progress and weather we have been following the time bound for each task or not.

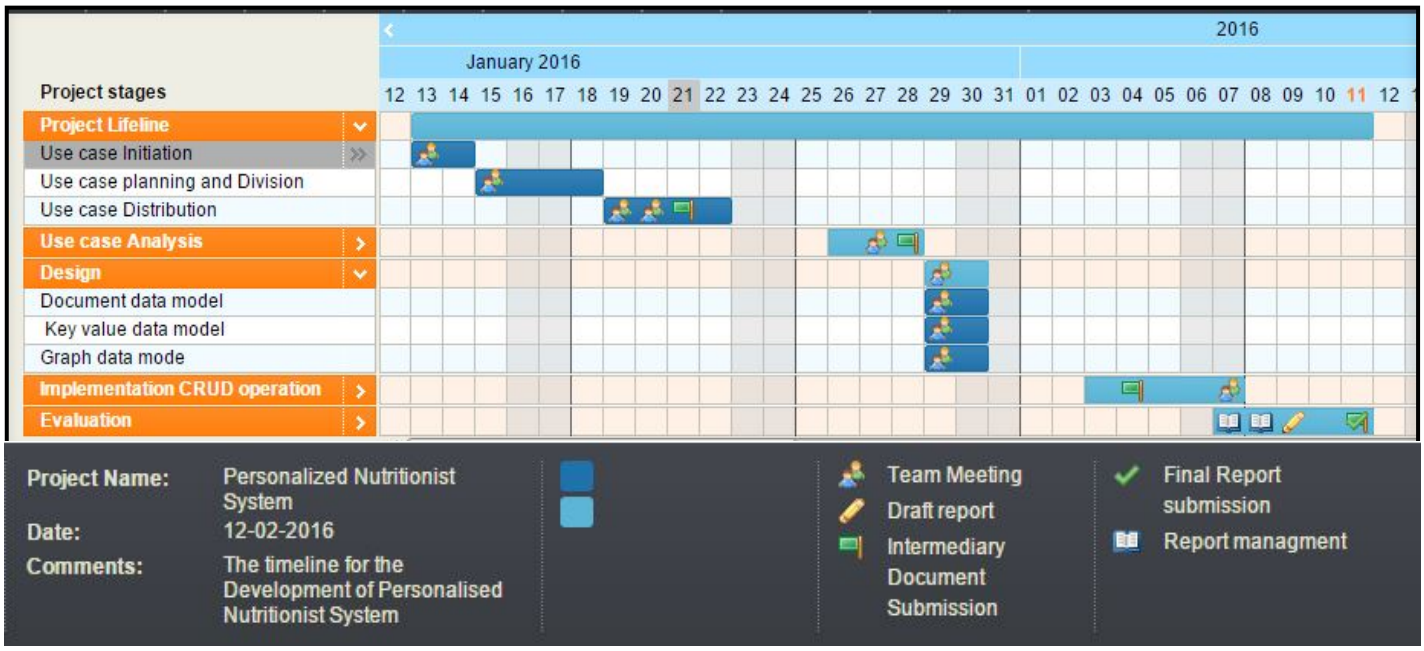


Figure 51: Gantt chart of the project

As seen in the figure below, on the left of the chart is the project stages containing the list of the activities in our project and along the top is the planned time scale for each activity.

8.2 Future Scope

Though much has been achieved from where we started, there is always scope for expansion in an application like this. Currently, our food data is limited mostly to the western foods. The expansion of this data will be done through further research of foods and feedbacks from user wherein we receive a request to add a new food item previously missing. Another aspect that can be added is processing the system to make it able to hand out a complete diet based on the user profile.

9. Conclusion

Nutrition plays a very important role in all individual's lives. It is the building block that must be started from the beginning to strive to be the best that can be done. It could help extend or reduce the life span as well as characterize the degree of livelihood as one progresses into old age. With the potential to live longer, comes the vital need to keep a track and adjust the amount, the type of food and supplements that are consumed. Realizing the benefits of a healthy lifestyle, one must continue to practice self-discipline and rigorous monitoring and dedication to consistently have proper nutrition for maintaining optimum health and wellness. Having a personalized nutritionist will lead to a natural habit of a healthy lifestyle with convenience.

Personalized Nutritionist helps better handling of a healthy and a determined lifestyle. This will be achieved by providing the users with informed best fit nutrition suggestions and a collection of diet programs as well as offering a distinctive service for tracking their food intake. The initial plan was to provide the user an end-to-end solution in order to keep him fit and healthy taking into account his needs. Due to the time constraints, Requirements had to be of different priorities to ensure meeting project key-services is met within timely manner. The following points are some of the features that were assessed important and were implemented:

- Logging intricate details of the user by taking into account his personal choices.
- Searching any food item by relevant name, user goals and popularity.
- Suggesting diet plans that matches the user requirements.
- Showing nutritional facts of commonly available food items
- Allowing the users to add new food items that are not present in the current pool of food items.
- Allowing the users to check their historical activities, compare nutritional intake against the adequate suggested amount.

Good nutritional practices and a balanced diet are not developed in one day, nor are they ruined in one unbalanced meal. A healthy lifestyle can be achieved by making smart choices. What is learned about user's habit in the initial stages will help us establish good dietary patterns. In the future, Personalized Nutritionist hopes to employ this kind of intelligence in matching people to their best fit diets based on true success stories of people with similar characteristics and properties, likes and dislikes, and lifestyle.

References

1. http://www.bmi-calculator.net/bmr-calculator/bmr-formula.php
2. http://neo4j.com/why-graph-databases/