Data in the time of COVID-19: A general methodology to select and secure a NoSQL DBMS for medical data

Kamal A. ElDahshan [1], AbdAllah A. AlHabshy[2], Gaber E. Abutaleb[3]

[1] Mathematics Department, Faculty of Science, Al-Azhar University, Cairo, Egypt.
[2] Mathematics Department, Faculty of Science, Al-Azhar University, Cairo, Egypt.
[3] Mathematics Department, Faculty of Science, Al-Azhar University, Cairo, Egypt.

Corresponding Author:
Gaber Abutaleb[3]
Tanta, 31512, Egypt
Email address: gaber_abutaleb@azhar.edu.eg

Abstract

Background: As the COVID-19 crisis endures and the virus continues to spread globally, the need for collecting epidemiological data and patient information also grows exponentially. The race against the clock to find a cure and a vaccine to the disease means researchers require storage of increasingly large and diverse types of information; for doctors following patients, recording symptoms and reactions to treatments, the need for storage flexibility is only surpassed by the necessity of storage security. The volume, variety, and variability of COVID-19 patient data requires storage in NoSQL database management systems (DBMSs). But with a multitude of existing NoSQL DBMSs, there is no straightforward way for institutions to select the most appropriate. And more importantly, they suffer from security flaws that would render them inappropriate for the storage of confidential patient data.

Motivation: This paper develops an innovative solution to remedy the aforementioned shortcomings. COVID-19 patients, as well as medical professionals, could be subjected to privacy-related risks, from abuse of their data to community bullying regarding their medical condition. Thus, in addition to being appropriately stored and analyzed, their data must imperatively be highly protected against misuse.

Methods: This paper begins by explaining the five most popular categories of NoSQL databases. It also introduces the most popular NoSQL DBMS types related to each one of them. Moreover, this paper presents a comparative study of the different types of NoSQL DBMS, according to their strengths and weaknesses. This paper then introduces an algorithm that would assist hospitals, and medical and scientific authorities to choose the most appropriate type for storing patients' information. This paper subsequently presents a set of functions, based on web services, offering a set of endpoints that include authentication, authorization, auditing, and encryption of information. These functions are powerful and effective, making them appropriate to store all the sensitive data related to patients.

40   **Results and contributions:** This paper presents an algorithm to select the most convenient
41   NoSQL DBMS for COVID-19 patients, medical staff, and organizations data. In addition, the
42   paper proposes innovative security solutions that eliminate the barriers to utilizing NoSQL
43   DBMSs to store patients' data. The proposed solutions resolve several security problems
44   including authentication, authorization, auditing, and encryption. After implementing these
45   security solutions, the use of NoSQL DBMSs will become a much more appropriate, safer, and
46   affordable solution to storing and analyzing patients' data, which would contribute greatly to the
47   medical and research effort against COVID-19. This solution can be implemented for all types of
48   NoSQL DBMSs; implementing it would result in highly securing patients' data, and protecting
49   them from any downsides related to data leakage.

50 1. Introduction

51   The exponential increase in the number of patients infected with COVID-19, and the complex
52   race towards finding treatment and prevention protocols, is a concern shared by all. Across the
53   globe, researchers are collecting every shred of information related to the disease, and all data
54   related to patients and their relations, which brings along the challenge of recording large swaths
55   of information. The volume of data is expected to be immense – matched only by its diversity
56   and variability. And as the number of patients increases, and alongside it, the breadth and
57   diversity of data, medical organizations, researchers, and authorities will require storage systems
58   offering greater flexibility, and which aren't beholden to a strict set of options. As it stands, the
59   list of COVID-19 symptoms remains ill-defined; this entails the need for a database management
60   system (DBMS) that can handle unstructured data, namely, a NoSQL DBMS. But until now,
61   NoSQL database systems still struggle with security issues; the lack of security of NoSQL
62   DBMSs may make it an obstacle to its adoption as the preferred standard to store COVID-19
63   patients' personal data.
64   This paper begins by explaining the five most popular categories of NoSQL databases,
65   introducing the most popular NoSQL DBMS types under each. Features of NoSQL DBMS are
66   briefly explained, with a strong emphasis on security. The paper addresses security problems
67   such as authentication, authorization auditing, and encryption. Not only does the paper mentions
68   security vulnerabilities in each type, but it also lays out a comparative study of the different types
69   of NoSQL DBMS, compares the strengths and weaknesses of each. This comparison is built on a
70   set of criteria, namely Data Model, Owner Company, Open source(Y/N), Implementation
71   Programming Language, Stable Release issue dates, Support Authentication, Support
72   Authorization, Support Auditing, Support Data Encryption, and Query Types. The paper then
73   moves to provide an algorithm that would help organizations choose the most appropriate type
74   for storing COVID-19 patients' information.
75   The purpose of this paper is to propose security solutions that resolve this weakness of NoSQL
76   DBMSs, this eliminating the hurdle preventing medical organizations and other organizations
77   storing patient information from using them. The proposed solution solves several security
78   problems, including authentication, authorization, auditing, and encryption. The implementation

79 of our security solutions will allow NoSQL DBMSs to become the more appropriate, safer, and
80 affordable solution to store and analyze COVID-19 patients' data.

81 ## 2. NoSQL SYSTEMS CATEGORIES AND THEIR SECURITY
82 ## ISSUES

83 The NoSQL acronym was introduced in 2009, during an event on distributed databases
84 (Lakshman & Malik, 2010). New technologies, such as Google's Bigtable (Chang & Dean,
85 2008) were presented as being capable of managing large amounts of data. Research on NoSQL
86 databases has blossomed since (Lourenço & Cabral, 2015). In addition to the five major
87 categories of NoSQL systems, some Multi-model NoSQL database systems, XML databases
88 (Elmasri & Navathe, 2017), as well as other types of systems that have been available even
89 before the term NoSQL came into use, grew in popularity and usage. This section thus discusses
90 the five main categories of NoSQL systems, as well as the Multi-model database ones, with an
91 emphasis on their security features, in the interest of patient data safety. This section will also
92 address the most commonly-used NoSQL DBMSs in terms of security. NoSQL databases suffer
93 from a variety of security issues (Sahafizadeh &Nematbakhsh, 2015). In NoSQL database
94 systems, security mechanisms such as Authentication and data encryption are either weak or
95 altogether inexistent.

96 First, we review possible attacks, and security mechanisms used to prevent them.

97 ### 2.1 Introduction to Security Issues in NoSQL DBMSs

98 **2.1.1 NoSQL Database Management Systems Classification.** NoSQL database
99 management systems are classified into five categories, namely Document-based Stores, key-
100 value stores, Column-based or wide column Stores, Graph-based Stores, and Object Stores.
101 Multi-model database categories may be added as shown in Fig. 1.

102 **2.1.2 NoSQL DBMSs comparison criteria.** Database management systems will be analyzed
103 according to their basic information, security mechanisms, and potential attacks.

104 - **Basic information includes:**
105   - The owner and year of first appearance
106   - The classification, specifying whether it is open source or proprietary
107   - The DBMS' implementation programming language
108   - Some characteristics of the database management system
109   - The query language used in each database management system.
110 - **Security mechanisms**
111   - **Authentication** is the process of verifying the entity identity (user or device)
112   permission to use the resources (data, application, etc.) (Zahid & Masood, 2014).
113   - **Verification** is usually achieved through accessing control mechanisms, such as
114   users' passwords and certificates (Grolinger & Higashino, 2013).
115   - **Authorization** is the mechanism that ensures that only an authorized user is allowed
116   to access system resources. Authorization is usually performed with the user's
117   permission (Grolinger & Higashino, 2013).
118   - **Auditing** is the process of keeping track and recording all actions performed by
119   database users (Elmasri & Navathe, 2017).

120 – **Data Encryption** refers to protecting and encrypting sensitive data throughout the
121   communications network so it cannot be read by unauthorized users or attackers
122   (Elmasri, 2017; Grolinger, 2013). There are three levels of encryption solution; Data
123   at rest, Client-to-server communication, and Server-to-server connections (Grolinger
124   & Higashino, 2013).
125 • **Attacks:** some attack types that DBMSs may endure include:
126   – **Script Injection**, wherein a security vulnerability is used by an adversary to inject
127   malicious code in pages or web forms to damage or retrieve data (Sullivan, 2011; Jim,
128   2007).
129   – **SQL Injection**, which may occur when an adversary injects a string input through the
130   application to damage, change the database, retrieve the sensitive data, and run
131   commands to deny service (Elmasri & Navathe, 2017).
132   – **Denial of service** (DOS), in which an adversary floods the server with requests, thus
133   preventing legitimate users from accessing the service, or the attacker may delete
134   some data (Elmasri & Navathe, 2017).
135   – **An inactive connection** may occur when an adversary opens the connection for a
136   long time.

137 ## 2.2 Document-Based NoSQL Stores
138 Document-oriented, or document-based NoSQL systems store data as collections of similar
139 documents. These types of systems are also known as document stores (Moniruzzaman &
140 Hossain, 2013; Dindoliwala, Morena, 2017). Each document is similar to a row in relational
141 databases, but is more flexible because it has no schema (is schema-free) and is specified as self-
142 describing data (Elmasri & Navathe, 2017). The users or the processing applications have the
143 schematic responsibility in document databases. This may result in some disadvantages, such as
144 the lack of referential integrity and normalization. However, the lack of schema provides high
145 flexibility to store a wide range of data, which makes it suitable for stores of big data. In
146 document-based databases, a record, called a document, has its own internal structure, and can be
147 stored as a value for every key (document ID). Documents are separated from one another, but
148 are grouped as a collection of data, and the database ensures that each query receives the
149 document version with the largest number of changes. Since this cannot guarantee full
150 concurrency control, it is called eventual consistency. The consistency takes some time to
151 achieve, which significantly speeds up the data processing at the expense of transactional
152 security. In document-based databases, queries can be performed in parallel and, therefore, sped
153 up with the map-reduce procedure (Meier & Kaufmann, 2019). Document-based databases are
154 also fit for storing sparse data, or semi-structured data, but that would necessitate an extensive
155 use of Nulls for data completeness purposes (Moniruzzaman & Hossain, 2013). Documents can
156 be specified in various formats, such as XML (Extensible Markup Language), YAML (Yet
157 Another Markup Language) (Kaur & Rani, 2013), or BSON (Binary JSON (JavaScript Object
158 Notation) (Moniruzzaman & Hossain, 2013; Özsu & Valduriez, 2020). The popular language to
159 specify documents in NoSQL systems is JSON (Han & Haihong, 2011). Document-Based
160 NoSQL Systems are suitable for managing Big Data collections of literal documents, like XML
161 documents, email messages, text documents, online shopping, event logging, deep analytical
162 processing, and content management. Examples of Document-Based NoSQL DBMSs (database
163 management systems) include CouchDB (JSON), MongoDB (BSON) (Özsu & Valduriez, 2020),
164 Terrastore, ThruDB, OrientDB, RavenDB, Citrusleaf, SisoDB, CloudKit, Perservere and
165 Jackrabbit) (Tudorica & Bucur, 2011).

166 **Security Issues in Document-Based NoSQL DBMSs**
167 This subsection discusses the most popular types of document Stores NoSQL Database
168 Management Systems.
169 **2.2.1 MongoDB**
170   • Introduction:
171   MongoDB is an open-source document-oriented NoSQL DBMS. It is a schema-free,
172   highly available, fault-tolerant and scalable NoSQL DBMS. MongoDB supports sharding
173   by configuring shared clusters (Zahid & Masood, 2014). It was developed by MongoDB
174   Inc., first appearing in 2009; its stable release appeared in June 2018. MongoDB was
175   written in C++, C, and JavaScript (Nayak & Poriya, 2013, Özsu & Valduriez, 2020), and
176   it consists of one or more collections of documents. A collection is analogous to a table,
177   but has no pre-defined schema. The document, which is the data storing unit in
178   MongoDB, has an ID and is equivalent to a record in the relational DB. Insert, delete, and
179   update operations can be performed on a collection (Dindoliwala & Morena, 2017).
180   Mongo Query Language is the query language used By MongoDB DBMS to manipulate
181   certain documents from a database collection. Map-reduce and REST querying are
182   supported by MongoDB (Chodorow, 2013).
183   • Security Analysis:
184   – Authentication: granting to users through the database itself or integration with an
185   external mechanism like LDAP (Lightweight Directory Access Protocol). MongoDB
186   grant Authentication to entities by using the SCRAM-SHA-1 IETF standard (Zugaj &
187   Beichler, 2019). An authentication command is used to authenticate the users, whereas
188   database-nodes are authenticated to the MongoDB cluster via key files. The default
189   Authentication mechanism in MongoDB is SCRAM-SHA-1, which verifies the supplied
190   user credentials with the name of the user, user's password, and Authentication database.
191   With LDAP, MongoDB can authenticate and authorize users directly via corporate LDAP
192   infrastructure to enforce centralized access policies (Dindoliwala & Morena, 2017).
193   MongoDB doesn't support authentication, however, if running in shared mode
194   (Sahafizadeh, 2015 & Noiumkar, 2014). When the authentication in MongoDB is based
195   on the key or the password, it's called a pre-shared secret. This password has been hashed
196   with the MD5 algorithm before being stored in the key file. Hackers can crack the pre-
197   shared secret's hashed value by cracking the MD5 (Dindoliwala & Morena, 2017),
198   because the MD5 algorithm is not secured (Sahafizadeh & Nematbakhsh, 2015).
199   MongoDB uses SSL with X.509 certificates to secure communication between the user
200   and the MongoDB cluster and intra-cluster authentication (Zahid & Masood, 2014).
201   – Authorization: Administrators in MongoDB can determine the permissions for users or
202   applications. They can also decide what data can be seen by users or applications when
203   performing a database query. The authorization mechanism in MongoDB contains LDAP
204   authorization, roles, and Field-Level Security with Read-Only Views. In MongoDB
205   privileges are assigned to roles, then roles are assigned to users. MongoDB involves
206   built-in roles that are used by administrators to control access to the MongoDB system. A
207   privilege consists of a specified resource and actions permitted on this resource. A
208   resource is a collection, set of collections, databases, or a cluster. MongoDB supports a
209   simple role-based authentication system that allows administrators to decide who has
210   access to the database and which level of access they have. Furthermore, MongoDB
211   supports authorization through LDAP (Dindoliwala & Morena, 2017).

212   – Auditing: Auditing framework stores actions such as operations that have been performed
213   by the users in the database, and authentication and authorization activities along with
214   write and read operations on the database. Administrators can extract and filter audit
215   trails for any operation with MongoDB. For example, it is possible to audit and log the
216   identities of users who accessed specific documents and the changes they made to the
217   database during their session. Administrators can configure MongoDB to log all the
218   actions or specific actions. MongoDB Enterprise Advanced (Dindoliwala & Morena,
219   2017) also supports the role-based auditing. With MongoDB, it is possible to report the
220   DB activities as well.
221   – Data Encryption: in MongoDB, data can be encrypted over the network, in backups, and
222   the rest in permanent storage. MongoDB Enterprise Advanced supports FIPS 140-2
223   encryption when it run in FIPS Mode (Dindoliwala & Morena, 2017). In MongoDB, the
224   data files are not encrypted automatically, so all data is stored as plaintext. This means
225   that any malicious user with access to the file system can extract information from the
226   files. MongoDB uses RESTful to manage its server via the HTTP protocol. However,
227   there is no data encryption for these ports, which means that hackers can display, capture
228   and get access to the data since no data encryption was performed while the data was
229   being sent back and forth between the client and server (Noiumkar, 2014).
230   – Script Injection: Using JavaScript, hackers can attack MongoDB. Examples about the
231   attacks that occurred on MongoDB can be found in this reference (Jim, 2007). NoSQL
232   databases are exposed to injection attacks like SQL databases. Son presented four
233   examples of GitHub site for injection vulnerabilities in PHP MongoDB-based
234   applications (Son, 2013).

235   **2.2.2 CouchDB**
236   • Introduction:
237   CouchDB is an open-source NoSQL DBMS. It was developed using Erlang (George,
238   2013). It became an Apache project in 2008. It was released in 2005 and the stable
239   release appeared in November 2017. CouchDB runs on Hadoop Distributed File Systems
240   (HDFS). CouchDB is document-based NoSQL DBMS, fault-tolerant, highly scalable,
241   available, and flexible (Sahafizadeh & Nematbakhsh, 2015). It supports document
242   redistribution across nodes by cluster configuration for large performance improvements.
243   A cluster uses the phenomenon of incremental replication by periodically copying any
244   changes in a document on a single node to other nodes. The incremental replication
245   results in inconsistency and data redundancy in the cluster. CouchDB uses the JavaScript,
246   JSON, and SQL++ as its query language (Özsu & Valduriez, 2020). Map-reduce and
247   REST querying are supported by CouchDB as well (Noiumkar, 2014).
248   • Security Analysis:
249   – Authentication: CouchDB uses a process called CRUD to perform authentication
250   (Noiumkar, 2014). The authentication in CouchDB is based on both cookies and
251   passwords. It uses the PBKDF2 hash algorithm to encrypt the password and send it over
252   the network using SSL protocol (Sahafizadeh, 2015 & Zahid, 2014).
253   – Authorization: There are three levels of users within the CouchDB; database admin,
254   server admin, and database member (Grolinger & Higashino, 2013). Authorization is
255   implemented only at the database level where only a single role in access control is
256   supported (Zahid & Masood 2014).

257   –   Auditing: The level of auditing in CouchDB is moderate (logging of all changes to user-
258   profiles and the level of database). Auditing is provided to log events and views in log
259   files. However, CouchDB does not support automatic backups of database logs, replicas,
260   and automatic logging; the configuration of logs is thus the responsibility of the database
261   administrators (Zahid & Masood 2014).
262   –   Data Encryption: CouchDB does not have automatic data encryption, so the data files are
263   at risk of being accessed and read directly (Sahafizadeh, 2015 & Noiumkar, 2014). Data
264   Encryption is supported in Client/Server via SSL. Also, data encryption could be
265   supported using HTTPS connections in Server/Server (Grolinger & Higashino 2013).
266   –   Denial of service attack: Websites such as securityfocus.com display vulnerabilities that
267   exist only on Apache CouchDB 1.5.0 (SecurityFocus, 2020). These vulnerabilities are
268   used by hackers to crash it. So CouchDB is potentially vulnerable to Denial of service
269   attacks (Sahafizadeh & Nematbakhsh, 2015).
270   –   Script Injection: CouchDB uses JSON to manipulate the data, making it vulnerable to
271   Script Injection. In other words, hackers can launch JSON Injection to attack CouchDB
272   (Sahafizadeh, 2015 & Noiumkar, 2014).

273 **2.2.3 Couchbase Server**
274   •   Introduction:
275   Couchbase Server is an open-source, document-oriented NoSQL DBMS. Couchbase
276   Server was developed by Couchbase Inc., initially released in August 2010 with the
277   stable release launched in February 2018. Couchbase Server was developed using Erlang,
278   C, and C++ (George, 2013). Various data centers contain all the cluster servers. The
279   documents are stored in vbuckets, which are special data containers uniformly distributed
280   across the cluster. More nodes can be added and removed because Couchbase cluster
281   scales are completely horizontal (Zahid & Masood, 2014). Map-reduce and REST
282   querying is supported by the Couchbase server (Grolinger & Higashino, 2013).
283   Couchbase also supports SQL++, an SQL like language which has been designed to
284   become a unified query language for NoSQL system.
285   •   Security Analysis:
286   –   Authentication: SASL (Simple Authentication and Security Layer) is supported by name
287   and password on Couchbase Server HTTP. Basic authentication is used in Couchbase
288   management REST (Representational State Transfer) API (Application Programming
289   Interface) (Grolinger & Higashino, 2013). External Authentication is also supported
290   (Zahid & Masood 2014).
291   –   Authorization: Couchbase Server does not support authorization (Grolinger & Higashino,
292   2013).
293   –   Auditing: Couchbase Server does not support auditing (Grolinger & Higashino, 2013).
294   –   Data Encryption: Couchbase Server does not support data Encryption in Data at rest,
295   Client/Server, and Server/Server (Grolinger & Higashino, 2013).

296 **2.3 Key-Value NoSQL Stores**
297 These are the simplest NoSQL databases. Key-value stores focus on high availability,
298 performance, and scalability, by storing data in a distributed storage system (Abed, 2020). The
299 data model used in key-value stores is simple, and in most such systems, there is no query
300 language, but rather a set of operations that can be used by the application programmers. The key
301 is a unique identifier associated with a value, and is used to rapidly locate it (Kaur & Rani,

302 2013). The value may have different formats for different key-value storage systems. In some
303 cases, the value is just an array or a string of bytes. Key-value stores are suitable for efficient
304 reading and writing of extensive amounts of data. Some disadvantages of this model include the
305 lack of some traditional capabilities such as atomicity and consistency, and as the data volume
306 increases, retaining more values as keys may become more difficult (Dindoliwala & Morena,
307 2017). The application using the key-value store has to interpret the structure of the data value.
308 Different key-value stores can thus store structured, unstructured, or semi-structured data items
309 (Elmasri & Navathe, 2017). NoSQL Key-Value stores are needed for application tasks such as
310 managing user profiles, sessions, or retrieving product names (Moniruzzaman & Hossain, 2013).
311 Examples of this type of NoSQL DBMSs include Redis, Voldemort (LinkedIn), Riak, and
312 BerkeleyDB (Özsu & Valduriez, 2020).

313 **Security Issues in Key-Value Stores NoSQL DBMSs**
314   This subsection discusses the most popular types of Key-Value Stores NoSQL Database
315   Management Systems.

316 **2.3.1 Redis**

317 • Introduction:
318   Redis is an Open source NoSQL DBMS (Grolinger & Higashino, 2013), developed using
319   C and C++ (Meier & Kaufmann., 2019) by Salvatore Sanfilippo. Redis was initially
320   released in May 2009 and the stable release appeared in June 2018. It is a Key-value
321   memory database, So, Redis data will be loaded into memory (Dayan, 2018) when it runs
322   and all operations are run in memory. Redis periodically saves data asynchronously to the
323   hard disk. It achieves high performance thanks to its use of pure memory. Redis can
324   handle more than 100,000 write or read operations per second. The maximum limit of
325   value is 1GB. However, Redis cannot be used as big data storage, and scalability is poor
326   because the capacity of the Database is limited by the physical memory. As such, Redis is
327   suitable for providing high-performance computing for a small amount of data (Han &
328   Haihong, 2011). SQL-like and Map-reduce querying is not supported in Redis but REST
329   querying is supported by third-party APIs (Grolinger & Higashino, 2013).

330 • Security Analysis:
331   – Authentication: Redis provides a password-based authentication to its clients.
332   Passwords are set by system administrators and stored in plaintext format. Redis
333   does not provide default authentication and listens on all IP addresses on port
334   6739(Han & Haihong, 2011).
335   – Authorization: Redis does not support any kind of access control (Han &
336   Haihong, 2011) i.e. it does not support authorization (Paterson, 2006).
337   – Auditing: Redis does not provide support for auditing (Tudorica, 2011; Paterson,
338   2006).
339   – Data Encryption: Redis does not provide support for Data encryption. It stores
340   data in plaintext (Paterson, 2006). As such, hackers - or anyone who can access
341   the Redis server - will be able to read all the data in the database. No data
342   encryption is performed in the communication between client and server, and
343   between other servers, neither at the same cluster nor different clusters (Nayak &
344   Poriya, 2013). Redis does not have data-at-rest encryption (Zugaj & Beichler,
345   2019).

346   –   Denial of service attack: Denial of service attacks on Redis was not reported
347   (Nayak & Poriya, 2013).
348   –   Script Injection: Redis does not have a concept of string escaping, so injection
349   becomes impossible (Nayak, 2013, Paterson, 2006).

350   **2.3.2 Voldemort**

351   •   Introduction:
352   Voldemort is an Open source key-value NoSQL DBMS, developed by LinkedIn (Dayan,
353   2018). Its initial release was in 2009 and the stable release appeared in July 2017
354   (Grolinger & Higashino, 2013). Voldemort was developed using Java (Deka, 2013).
355   Three simple operations are included in Voldemort; reading, writing, and deletion. All of
356   them are executed using a key (Chen, 2014). There is no SQL-like querying language in
357   Voldemort but Map-reduce querying is supported (Grolinger & Higashino, 2013).

358   •   Security Analysis:
359   –   Authentication: There is no authentication mechanism in Voldemort.
360   –   Authorization: There is no authorization mechanism in Voldemort.
361   –   Auditing: There is no auditing in Voldemort.
362   –   Data Encryption: No encryption, neither between the client and the server nor among
363   the servers (Grolinger & Higashino, 2013).

364   **2.3.3 Oracle Berkeley DB (BDB)**

365   •   Introduction:
366   Oracle Berkeley DB is a closed source key-value NoSQL DBMS. Oracle Berkeley DB
367   was developed by Sleepycat Software. Oracle Berkeley DB initial release appeared in
368   1994 and the stable release was in June 2018. It is a high-performance data management
369   services to applications. It uses a simple function for data access and management
370   (Sullivan, 2011). SQLite and REST querying is supported by Oracle Berkeley DB, but
371   Map-reduce querying is not (Grolinger & Higashino, 2013). In terms of implementation
372   of Oracle Berkeley DB, was implemented in C, Oracle Berkeley XML was implemented
373   in C++ and Oracle Berkeley JE was implemented in Java.

374   •   Security Analysis:
375   –   Authentication: There is no authentication mechanism in Berkeley DB.
376   –   Authorization: There is no authorization mechanism in Berkeley DB.
377   –   Auditing: There is no auditing in Berkeley DB.
378   –   Data Encryption: Berkeley DB supports data at rest encryption. No encryption
379   between Servers (Grolinger, Higashino 2013).

380 **2.4 NoSQL Graph Stores**

381 A graph database is a database that uses graph structures for semantic queries with nodes, edges,
382 and properties to represent and store data. A key concept of this system is the graph, which
383 relates data items in the store. Both edges and nodes can be labeled to indicate the types of
384 vertices and associations they represent, and it is generally possible to store data associated with
385 both individual nodes and individual edges (Elmasri & Navathe, 2017). In the Graph database
386 model, the database is represented as a network structure containing edges between nodes to
387 illustrate the relationships among nodes. Nodes may also contain properties that describe the real
388 data contained within each object. Edges may also have properties. An edge (or a relationship)
389 connects two nodes and it may be directed. When directions are added, relationships between

390   nodes are identified by the names of the nodes, and can be traversed in both directions (Kaur &
391   Rani, 2013). The graph uses the Index-Free adjacency technique, in which each node contains a
392   pointer pointing to the adjacent node. For every node in the Graph database, the system can
393   access its direct neighbor, without the need to consider all edges (Abed, 2020). This is known as
394   the index-free adjacency property; and it is the key feature of the graph databases (Meier &
395   Kaufmann, 2019). Graph databases are efficient schema-less DBs, which makes them suitable
396   for semi-structured data storage. The queries in Graph databases are expressed as traversals;
397   which make them much faster than relational databases (Nayak & Poriya, 2013). In general,
398   graph databases are appropriate when the main interest is the relationships between data (e.g.,
399   generating recommendations, social networks, conducting forensic investigations, network
400   search, and fraud detection (Moniruzzaman & Hossain, 2013). Examples of Graph Databases
401   include Neo4j, Sones GraphDB, InfiniteGraph, InfoGrid, and AllegroGraph (Khasawneh, AL-
402   Sahlee, 2020).

403 **Security Issues in Graph NoSQL DBMSs**
404 This subsection discusses the most popular types of graph Stores NoSQL Database Management
405 Systems.

406   **2.4.1 Neo4j**
407   •   Introduction:
408   Neo4j is an open-source graph NoSQL DBMS (Paterson, 2006), developed by Neo
409   Technology. It was initially released in 2007 and the stable release appeared in April
410   2018. It was developed using Java. The advantages of Neo4j include providing object-
411   oriented, flexible network structure, reliability, highly available, and scalability. Neo4j
412   uses the graph data model which consists of nodes and edges and the relationships
413   between them. Neo4j uses CYPHER query language. REST querying is supported by
414   Neo4j but Map-reduce is not supported (Grolinger & Higashino, 2013). Neo4j is used
415   extensively in software that has complex relationships such as social networking and
416   recommendation engines (Moniruzzaman & Hossain, 2013).
417   •   Security Analysis:
418   –   Authentication: Neo4j does not support Authentication (Grolinger & Higashino,
419   2013).
420   –   Authorization: Neo4j does not support authorization (Grolinger, 2013; Paterson,
421   2006)
422   –   Auditing: Neo4j does not support auditing (Grolinger, 2013; Paterson, 2006).
423   –   Data Encryption: Neo4j does not support data at rest encryption, nor does it support
424   data encryption between servers (Grolinger, Higashino, 2013); however it uses SSL-
425   based communication protocol between the client and the server (Paterson, 2006).

426   **2.4.2 HyperGraphDB**
427   •   Introduction:
428   HyperGraphDB is an Open source NoSQL DBMS that implements powerful knowledge
429   management formalism known as directed hypergraphs (Grolinger, Higashino, 2013).
430   HyperGraphDB is developed by Kobrix Software, Inc. The stable release of
431   HyperGraphDB appeared in May 2017. Hyper-GraphDB was developed using Java.
432   Higher-order Relations are naturally represented by the HyperGraphDB model. The
433   HyperGraphDB model is very useful in areas related to data modeling such as artificial

434	intelligence and knowledge of representation and bioinformatics. This data model is
435	designed for AI, Knowledge representation and the semantic Web (Kobrix, 2020; Angles,
436	2012). HyperGraphDB provides SQL- like query language. REST querying is supported
437	by HyperGraphDB but Map-reduce is not (Grolinger & Higashino, 2013).
438	• Security Analysis:
439	– Authentication: HyperGraphDB does not support authentication.
440	– Authorization: HyperGraphDB does not support authorization.
441	– Auditing: HyperGraphDB does not support auditing.
442	– Data Encryption: HyperGraphDB does not support data encryption (Grolinger &
443	Higashino, 2013).
444	**2.4.3 AllegroGraph**
445	• Introduction:
446	AllegroGraph is a closed-source graph NoSQL DBMS, developed by Franz Inc. The
447	stable release appeared in October 2017 (Khasawneh & AL-Sahlee 2020; Grolinger &
448	Higashino 2013). It was developed using Common Lisp (Franz, 2020). It offers high-
449	performance and is mainly used for developing semantic web applications. It can store
450	data and meta-data as RDF (Resource Description Framework) triples. AllegroGraph
451	maintains high performance when accessing billions of quads due to efficient memory
452	utilization along with disk-based storage (Arora, 2013; Franz, 2020). SPARQL and
453	Prolog are the query languages used by AllegroGraph. REST querying is supported by
454	AllegroGraph but Map-reduce is not (Grolinger & Higashino, 2013).
455	• Security Analysis:
456	– Authentication: Authentication is supported by AllegroGraph.
457	– Authorization: Authorization is supported by AllegroGraph such as read, write, and
458	delete.
459	– Auditing: AllegroGraph can be used to record specific changes in the audit log.
460	– Data Encryption: AllegroGraph does not support data at rest encryption. But it uses an
461	HTTPS-based communication protocol between the client and the server (Grolinger &
462	Higashino 2013).

463 2.5 Column-Based or Wide Column NoSQL Stores

464	By providing an additional structure, Column-based databases utilize the key-value concept. In
465	certain uses, it has been proven that to optimize the read operations, it is much better to store the
466	data into the relational datasets per column, not per row. This is due to the occasional need for all
467	columns in a single row at once; however, there are groups of columns that are often read
468	together (Meier & Kaufmann, 2019). Column-based data stores are designed to address the huge
469	number of columns and frequent changes in the schema (Moniruzzaman & Hossain, 2013). Wide
470	column stores have tables that contain columns. A column is specified by a combination of a
471	Column Family and a Column Qualifier; each column family has a name that should be declared
472	when the table is created, and cannot be changed subsequently. After loading data into a table,
473	each column family is associated with column qualifiers. Column qualifiers are not declared
474	when the table is created, but can be dynamically created and inserted into the table. Read and
475	write are done using columns rather than rows. The reason for structuring the data in groups of
476	columns—column families—as storage units is that it is a sound way to optimize the data access.
477	Column-based databases adopt this model; they store the data not in relational datasets but rather

478   in enhanced and structured multi-dimensional keyspaces (Meier & Kaufmann, 2019). Wide-
479   column stores have great features such as high performance and high scalability (Elmasri &
480   Navathe, 2017). These types of DBMSs are suitable for data processing, distributed data storage,
481   content management, event logging, and categorizing for analytics (Moniruzzaman & Hossain,
482   2013). The most common open-source column-oriented databases are HBase, Hypertable. HBase
483   are derivatives of Bigtable (Kaur & Rani, 2013).

484   **Security Issues in Column-based NoSQL DBMSs**
485   This subsection discusses the most popular types of column-based Stores NoSQL Database
486   Management Systems.

487   **2.5.1 Hypertable**
488   • Introduction:
489   Hypertable is a column-oriented NoSQL DBMS. It was developed by Zvents before 2008
490   and the stable release appeared in March 2016. Hypertable was developed using C++
491   (Deka, 2013). It is a high-performance open-source DBMS, and can run on Hadoop
492   Distributed File Systems (HDFS). It is modeled after Google's Bigtable DBMS. Like
493   Bigtable, Hypertable stores data using the column-oriented methodology (Khasawneh,
494   AL-Sahlee, 2020). There is no SQL-like query language in Hypertable.
495   • Security Analysis:
496   – Authentication: Hypertable does not support Authentication (Nayak, 2013; Paterson,
497   2006). Moreover, it does not support Authentication in communication between the client
498   and the server or among its servers (Nayak, Poriya 2013).
499   – Data Encryption: Hypertable does not support encryption for its data files (Nayak, 2013,
500   Paterson, 2006). Hypertable does not support encryption in communication between the
501   client and server or between its servers (Nayak &Poriya 2013).
502   – Denial of service attack: There is no information reported about denial of service attacks
503   on Hypertable.
504   – Script Injection: Though Hypertable has an HQL (Hypertable Query Language), which is
505   similar to the SQL, it is not vulnerable to script injection (Nayak & Poriya 2013).

506   **2.5.2 HBase**
507   • Introduction:
508   HBase is an open-source column-oriented NoSQL DBMS. HBase was developed by the
509   Apache Software Foundation. Its initial release was in February 2015 and the stable
510   release was in April 2018. It was developed using Java (Meier & Kaufmann, 2019).
511   Features include automatic distributability and scalability. HBase is based on the concept
512   of Google's Bigtable and implemented in Java (Tudorica, 2011; Paterson, 2006). There is
513   no SQL-like query language in HBase, but Map reduce and REST querying are supported
514   (Grolinger, Higashino, 2013).
515   • Security Analysis:
516   – Authentication: HBase supports token-based Authentication, for MapReduce tasks and
517   user Authentication is done by using SASL (Simple Authentication and Security Layer
518   with Kerberos (Tudorica, 2011; Paterson, 2006).
519   – Authorization: HBase supports authorization by access control list. Permissions include
520   create, read, write and admin (Grolinger, 2013; Paterson, 2006).
521   – Auditing: Auditing is supported by HBase (Han & Haihong, 2011).

522   –   Data Encryption: HBase does not support the encryption of data at rest (Tudorica, 2011;
523   Paterson, 2006), but HBase supports the encryption in communication between client and
524   server (Grolinger & Higashino 2013).
525   –   Denial of service attack: No report for denial of service attack.
526   –   Script Injection: No report for script injection (Paterson, 2006).

527   2.6 Object Store NoSQL Stores

528 An object-oriented database stores the information or data as objects according to the object-
529 oriented paradigm (Nayak & Poriya, 2013). Object-oriented stores can be thought of as a
530 combination of object-oriented programming (OOP) methodology and database principles. Data
531 encapsulation, polymorphism, inheritance, and all other features of OOP are offered by other
532 object-based databases. The classes, attributes, and objects in the Object-oriented stores are
533 comparable to tables, columns, and tuples in a table in RDBMS respectively. Each object has an
534 identifier that uniquely represents it. Because the object can easily be retrieved using pointers,
535 data access becomes much faster in object-oriented databases. Harnessing the Object-based
536 databases can ease the agility of modern software development processes (Nayak & Priya,
537 2013). These types of data stores are not relational databases and are not queried using SQL
538 (Dindoliwala & Morena, 2017). Object Stores are the most convenient category for storing and
539 retrieving binary large objects such as files, images, videos, and audio files. They are ideal for
540 applications that require complex relationships between objects and changing object structures,
541 or if the application defines members that are grouped in collections. Common uses for the
542 object-oriented databases are applications of scientific research, telecommunication, computer-
543 aided drafting, etc. The disadvantage of object-oriented databases is the limited programming
544 languages they are bounded to (Nayak & Poriya 2013). Examples of Object Store NoSQL
545 DBMS' include Gemstone, db4o, and objective/DB (Dindoliwala &Morena 2017).

546 Security Issues in Object Store NoSQL DBMSs

547 This subsection discusses the most popular types of object Stores NoSQL Database Management
548 Systems.

549   2.6.1 Gemstone

550   •   Introduction:
551   Gemstone is an open-source an Object NoSQL DBMS. It was developed by GemTalk
552   Systems. Its first appearance was in 1986 and the stable release is 64 Bit 3.3.7. Gemstone
553   is Proprietary commercial software. Multiuser environments are supported by Gemstone;
554   each user can have one or more sessions, and multiple user sessions can be active at the
555   same time. Gemstone provides security at several levels from login authorization to
556   object access (Dindoliwala & Morena, 2017). There is no general query language like
557   SQL in Gemstone.
558   •   Security Analysis:
559   –   Authentication: In Gemstone, each user is identified by a unique user ID and a password.
560   The user is represented by an instance of the class User Profile. The User Profile contains
561   a user ID, password, default authorization information, and the user's group. Only users
562   who have a User Profile can logon to the system.
563   –   Authorization: Authorization exists within Gemstone and controls individual object
564   access.

565   – Auditing: The DB administrator can also configure the Gemstone system to monitor
566   failures to login.
567   – Data Encryption: Gemstone does not support Data Encryption (Dindoliwala & Morena,
568   2017).
569   **2.6.2 db4o**
570   • Introduction:
571   db4o is an open-source object NoSQL DBMS. It was developed by Action and the stable
572   release 8.0. db4o was developed using Java and C# (Noiumkar, 2014). db4o-SQL is an
573   interface to allow SQL queries into a db4o database.
574   • Security Analysis:
575   – Authentication: db4o provides only an internal authentication mechanism.
576   – Authorization: db4o offers file-level authorization.
577   – Auditing: db4o lacks auditing capabilities.
578   – Data Encryption: db4o uses the eXtended Tiny Encryption Algorithm (Dindoliwala &
579   Morena, 2017) for data encryption.
580   **2.6.3 Objectivity/DB**
581   • Introduction:
582   Objectivity/DB is an object NoSQL DBMS. Objectivity/DB is proprietary to Objectivity
583   Inc. and first commercialized in 1990. Objectivity/DB was developed using Java, C#,
584   C++, and Python. Since the Objectivity/DB schema is created from internal class
585   definitions from the programming language, it is easier to maintain changes to the
586   schema (Dindoliwala & Morena, 2017). Objectivity/DB provides OQL (Object Query
587   Language which is like SQL querying language
588   • Security Analysis:
589   – Authentication: For Authentication Objectivity/DB has two mechanisms; the first (by
590   default) is Kerberos; the second is by using Advanced Multithreaded Server
591   (AMSprotocol).
592   – Authorization: Relies on the operating system and file systems for access control.
593   – Auditing: No auditing capabilities have been mentioned in Objectivity/DB.
594   – Data Encryption: Objectivity/DB doesn't provide any inbuilt encryption (Dindoliwala
595   & Morena, 2017).
596

597   ## 2.7 Multi-model NoSQL database Stores
598   This term refers to at least two types of NoSQL categories integrated into one system. In some of
599   the NoSQL systems, a combination of more than one NoSQL system is used. This integration
600   ensures that all the features in each category are combined. For example, Cassandra is a NoSQL
601   DBMS used on Facebook. It combines column stores and key-value stores (Nayak & Poriya,
602   2013). DynamoDB is a NoSQL database used in Amazon. It combines document and key-value
603   data models (Sahafizadeh & Nematbakhsh, 2015).
604   **Security Issues in Multi-model NoSQL DBMSs**
605   This subsection discusses the most popular types of Multi-model database Stores NoSQL
606   Database Management Systems.
607   **2.7.1 Cassandra**
608   • Introduction:

609	Cassandra is an open-source Multi-model database NoSQL DBMS. It was developed by
610	Apache Software Foundation and its stable release appeared in February 2018. Cassandra
611	was developed using Java, and is used by Facebook. Cassandra is a Multi-model database
612	of both column stores and key-value stores (Moniruzzaman & Hossain, 2013). Its key
613	characteristics include: a free schema, flexibility, convenience of adding or deleting
614	fields, support for range queries, partition tolerance, and high scalability (Han &
615	Haihong, 2011). Cassandra has three basic components; data, centers, and clusters nodes.
616	The data in a cluster is organized into key spaces (databases), which contain tables.
617	Tables contain rows, and rows have columns. The query language used is Cassandra
618	Query Language (CQL) also Cassandra supports Map-reduce querying and REST
619	querying by Third-party APIs (Dindoliwala & Morena, 2017).
620	• Security Analysis:
621	– Authentication: authentication in Cassandra is particularly weak since passwords are
622	encrypted using the MD5 hash. The entire authentication in Cassandra is provided
623	between the client and the Cassandra cluster. In other words, the inter-node message
624	exchange does not support Authentication by default. Hence, a malicious user with access
625	to the network used by the Cassandra cluster can bypass the client-side Authentication
626	and extract data or damage it. However, the transmission security at the data center, rack,
627	and a cluster is provided by Cassandra via enabling SSL/TLS in its configurations (Han
628	& Haihong, 2011). Cassandra supports only internal Authentication mechanisms.
629	Cassandra 3.0 Authentication is role-based and stored internally in Cassandra system
630	tables (Dindoliwala & Morena, 2017).
631	– Authorization: The entire authorization mechanism in Cassandra is provided between
632	the client and the Cassandra cluster (Han & Haihong, 2011). As such, a malicious user
633	with access to the network used by the Cassandra cluster can bypass the client-side
634	authorization and extract data or damage it. In 2015, it had been proven that Cassandra
635	did not support authorization (Sahafizadeh & Nematbakhsh 2015; Paterson, 2006).
636	Cassandra uses the GRANT/REVOKE mechanism to manage permissions as part of the
637	authorization mechanism. In Cassandra 2.2 and later versions, Role-based access control
638	is available and permissions may be applied on resources such as keyspace, table,
639	function (Dindoliwala & Morena, 2017).
640	– Auditing: Cassandra does not support auditing (Tudorica, 2011; Paterson, 2006).
641	Cassandra auditing is available in Enterprise Cassandra as a log4jbased integration and a
642	per-node basis. Filters are available for logging using a combination of the following
643	categories – ADMIN, AUTH, ALL, DCL, DML, and QUERY (Dindoliwala & Morena,
644	2017).
645	– Data Encryption: Data files in Cassandra are stored without encryption and the database
646	does not have automatic data encryption, so malicious users can access the data and read
647	it (Nayak, 2013; Tudorica, 2011). Since version 3.2, Cassandra supports at-rest data
648	encryption through Transparent Data Encryption (TDE) (Dindoliwala & Morena, 2017).
649	– SQL Injection: Cassandra Query Language (CQL) has a similar syntax to SQL's, so it
650	is believed that it can be attacked in the same way SQL is vulnerable to attacks by SQL
651	injection (Nayak, 2013; Paterson, 2006).
652	– Denial of service attack: Cassandra performs one thread per client. As such, it is
653	vulnerable to denial of service attacks. A malicious user creating enough fake
654	connections can drain Cassandra's resources. If hackers knew the IP addresses of all

655   Cassandra servers in the cluster and created enough fake connections, they could drive it
656   out of service (Nayak & Poriya 2013).
657   – Inactive connections: Cassandra does not set the timeout value for inactive connections;
658   consequently, it has a problem in managing inactive connections (Nayak, 2013; Paterson,
659   2006).

660   **2.7.2 DynamoDB**

661   • Introduction:
662   DynamoDB is a Closed source NoSQL DBMS used by Amazon. Its initial release
663   appeared in January 2012 (Grolinger & Higashino, 2013). DynamoDB was developed
664   using Java (Meier & Kaufmann, 2019). It is fast and flexible and supports both the
665   document and key-value data models (Paterson, 2006). The query language is proprietary
666   in DynamoDB but MapReduce and REST query are supported (Grolinger & Higashino,
667   2013).

668   • Security Analysis:
669   – Authentication: Authentication is supported by DynamoDB (Paterson, 2006). It is
670   ensured by the integration of Identity and Access Management (IAM services (Grolinger,
671   2013; Paterson, 2006).
672   – Authorization: Authorization is supported by DynamoDB (Paterson, 2006). It allows
673   users to create policies and operations on domains (Grolinger & Higashino 2013).
674   – Auditing: DynamoDB integrates with Amazon Cloud Watch Service. Access
675   information about latencies for operations, requests throughput, and the amount of data
676   stored (Grolinger & Higashino 2013) are subject to auditing at the ACWS.
677   – Data Encryption: DynamoDB does not support the encryption of data at rest (Grolinger
678   & Higashino, 2013). Communication between the client and the server uses the https
679   protocol (Paterson, 2006).

680 # 3. QUERY LANGUAGE

681 A query language is a set of user-entered statements that define, manipulate, and retrieve data
682 from database and information systems by sending queries. NoSQL DBMSs don't use a standard
683 query language, and most of their providers have developed their query languages. This brings
684 some stumbling blocks when users want to switch from one NoSQL DBMS to another; for
685 instance, Cassandra supports Cassandra Query Language (CQL), while MongoDB uses Mongo
686 query language, and so on. To resolve this, a common query language that can support a variety
687 of NoSQL database users was needed. Unstructured Query Language, or UnQL (pronounced
688 'uncle'), is a joint effort of several NoSQL DBMS providers that brings a commonplace and
689 standardized data definition, manipulation, and retrieval language to the NoSQL platform. UnQL
690 is being developed by the creators of Couch and SQLlite. It is considered as the superset of SQL;
691 providing a SQL-like syntax, and familiarity to the database developers and users. The model
692 and syntax of the UnQl is appropriate for the unstructured, self-describing data formats. UnQl
693 provides features to fetch and process complex document structures. It also provides the
694 elasticity of the NoSQL schema-less design and the structured table format of the relational
695 database. Data stored in JSON format as well as in document and non-relational stores can be
696 queried using UnQl. UnQl is open for developers and academic communities for further
697 enhancements and development (Nayak & Poriya, 2013).

4. COMPARISON OF NOSQL DATABASES CATEGORIES

A generic introduction to NoSQL databases through the categorization of different NoSQL databases is presented in Table 1 (Strauch & Sites, 2011). If we add the Object Store NoSQL systems to the table, it becomes as follows:

5. SUMMARIZED TABLE

The aforementioned exhaustive study is summarized in Table 2, which presents the result of the comparison between the mentioned DBMSs in terms of the selected determining criteria.

6. NoSQL DBMS SELECTION ALGORITHM

In this section, we propose a smart agent algorithm which selects NoSQL DBMSs according to the user's needs – in our case, that would be selecting the most convenient DBMS for large medical records storage. The algorithm gives a weight to each criterion, measured as (user weight/total weights). The user sets the appropriate weights based on their needs. The algorithm arranges weights in an ascending order, and returns an ordered table according to the arranged weights.

Result: an ordered table (output rows)

Input : 10 variables;
- Data Model
- Company
- Open-source
- Implementation Programming Language
- Stable Release issue date
- Support Authentication
- Support Authorization
- Support Auditing
- Support Data Encryption
- Query Types

Inputs Weight:
- Data Model weight
- Company weight
- Open source weight
- Implementation Programming Language weight
- Stable Release issue date weight
- Support Authentication weight
- Support Authorization weight
- Support Auditing weight
- Support Data Encryption weight
- Query Types weight

Lists:
- arranged weight list
- selected List
- output rows

output: an ordered table

740   **if** all input! = null then
741   sort inputs incrementally according to their weights.
742   **end**
743   arranged weight list = arrange (Inputs Weight).
744   /* Extract all tuples from the table that satisfy the user's constraints
745   and store data in the selected list. */
746   selected List= Select * from table where column name equal inputs value for each given
747   input value.
748   output rows =sort selected List function (the maximum value from arrange weight list,
749   selected list)
750   **Algorithm: NoSQL DBMS selection algorithm**

751 # 7. Results and Discussion

752 LDAP is a directory service and a searchable database repository that allows authorized users and
753 services to find information related to people, computers, network devices, and applications (Lasisi
754 & Ajagbe, 2012). LDAP directories are used mostly for reads, and its servers are simple to install
755 and maintain. LDAP is an Open Standard Protocol and is Lightweight. Moreover, it offers common
756 access for multiple database management systems such that MongoDB, MySQL, PostgreSQL,
757 Oracle 9i, IBM DB2, etc.
758 **Apache Directory Studio** is an integrated directory tool platform intended for use with any
759 LDAP server, but is specifically designed to be used with ApacheDS. Creating and launching a
760 new LDAP server now takes less than 10 seconds (Apache Software Foundation, 2020).
761 **JSON Web Token (JWT)** is an open standard that characterizes a minimized and independent
762 approach to safely move data between parties as a JSON object. (IETF JWT, 2020).
763 **Spring Boot** gives a good platform to Java designers to build up an independent and production-
764 grade spring application that you can simply run (Pivotal Team, 2020).
765 **Logback** is expected as a replacement to the famous log4j venture, picking up where log4j leaves
766 off.
767 This paper presents a set of functions, based on web services. The latter offers a set of endpoints
768 that include authentication, authorization, auditing, and encryption of information. It will be secure
769 to use the client created via these web services in all types of NoSQL DBMS.
770 This section will explain the web services identified.
771 **First**, web services related to the authentication process depend on authentication through the
772 Apache directory server. In this process, the users and units they belong to are defined within the
773 Apache server, so that the username and password are sent via secure channel. If the user
774 credentials are valid, JSON Web Token (JWT) returns all the user's information including their
775 role within the organization and the units they belong to. With the returned information, NoSQL
776 DB allows the user to log into the database after ensuring that the user is authorized. If the
777 validation process fails, access is denied. This web service related to the authentication process
778 not only allows for user verification, but also provides all the processes related to the user
779 management, such as creating a new user, updating, or deleting an existing user within the
780 organization. The returned information might be useful for databases in one way or another. It

781 also facilitates the interaction with the authentication server. The authentication process is done
782 by the authentication controller, which contains two main functions: the first is the generate JWT
783 token function, which is responsible for completing the user authentication process and returning
784 the user's token. The second is responsible for validating the token, and ensuring that it is valid
785 for use and its content has not been changed or that its validity has not expired.
786 The other user-related processes in the authentication server are performed by the user CRUD
787 Controller. This controller contains five main functions, which can be described as follows:

788 • **Get all users:** which gets all information to of all users on the authentication server and
789 returns JWT token with the essential information, such as display name, user name,
790 organizational unit, … etc.

791 • **Get user by UID:** which returns specific user's information by the UID.

792 • **Delete user by UID:** which deletes any user by the UID.

793 • **Create a new user:** which creates a new user within the organization.

794 • **Update User by UID:** which updates the information of any user by UID inside the
795 authentication server.

796 All processes related to the authentication processes are shown in Fig. 2. and Fig. 3.
797 **Second**, the web services related to the authorization process. Most of the processes related to
798 authorization are built within the databases. in this case, authentication is done from outside, and
799 the user's predefined privileges are used within the databases. But sometimes the authorized
800 operations may be related to the roles of people within the organization, so that the databases
801 enable users to perform the operations they are allowed to. We hereby provide a solution to
802 authorize the usage of the role within the organization, so that they are allowed/rejected to
803 perform their operations through their organizational units. Databases can thus allow/reject
804 users' operations based on their roles in the organization. The authorization process is done using
805 an authorization controller that performs five basic functions. They can be described as follows:

806 • **Get all groups (organization unit):** By using this function within the NoSQL DB, the
807 whole organization unit inside the server is returned. The returned values contain important
808 information such as organization unit (OU) name and description.

809 • **Get a group by OU name:** By using this function inside the NoSQL DB, OU information
810 is returned by the OU name.

811 • **Delete the group by OU name.**

812 • **Create a new OU:** This function enables the NoSQL DB to create a new group within the
813 organization.

814 • **Update group by OU name:** This function enables updating any OU information within
815 the apache directory server.

816 All processes related to the authorization processes are shown in Fig. 4.
817 **Third**, web services related to the auditing process. the auditing process is crucial for any type of
818 database. Auditing documents all the operations that occur in the databases. We offer a set of
819 APIs related to the auditing process, based on the Logback. The proposed solution contains the

820 auditing controller, which contains a basic function to define the auditing process. Here is either
821 the INFO, ERROR or WARN and the content to be stored in the auditing file. The databases will
822 thus be able to easily record all the processes that have been performed along with their specific
823 timestamp. Fig. 5. shows one such recorded audit operations.
824 **Fourth**, web services related to the encryption process. Since securing such sensitive data as
825 medical records is absolutely critical on DB, encryption is a basic requirement within NoSQL
826 DB. Encryption helps to store the information safely and confidentially so that it is not being
827 accessed by unauthorized users. This paper offers a set of web services to encrypt sensitive data,
828 such as RSA, Triple DES, AES, and BlowFish. The web services contain four types of
829 controllers; namely, AESController, BlowfishSsecretKey, TDESController, and RSAController.
830 Each controller includes two functions. One is used for encryption, the other for decryption; with
831 the possibility of storing the used keys easily and changing them if necessary. Consequently,
832 NoSQL DB can use one of the four strongest encryption cryptosystems to encrypt sensitive data.
833 All processes related to the encryption processes are shown in Fig. 6.
834 The APIs described in the above paragraphs are illustrated in Fig. 7.
835 This figure shows the integration between the proposed system and NoSQL DBMS.
836 • **Login request:** In this request, the user logs in to the NoSQL DBMS. NoSQL DBMS
837 makes a login request to the Apache directory server with the password and username
838 sent. The Apache directory server checks the username and password and returns a JWT
839 token in case of success; access is denied in case of failure. The JWT token is sent with
840 all requests made to the proposed system.
841 • **Authorization request:** The user makes a query that requires verification of its
842 privileges. The NoSQL DBMS checks the user's permissions, either from the stored JWT
843 token, or the Apache directory server. NoSQL DBMS completes or rejects the query
844 based on the person's authorization.
845 • **Auditing request:** The NoSQL DBMS uses the audit API to record all operations
846 performed on it. The DB admin obtains the audit report to review all operations that
847 occurred on the NoSQL DBMS.
848 • **Encryption/Decryption request:** The NoSQL DBMS uses the API to encrypt and
849 decrypt all sensitive information using four different types of encryption algorithms.
850 It is important to mention that in the testing phase, all communications between NoSQL DB and
851 web services will be secured with HTTPS instead of HTTP, for the data to be transmitted in a
852 protected channel using a self-signed certificate. In the live phase, using an authentic certificate
853 from a certificate authority is recommended.

8. Contribution:

855 This paper presented:
856 1. A taxonomy of NoSQL systems categories and their DBMSs.
857 2. A comparative study among NoSQL DBMS, along with several criteria including
858 security issues.

859   3.   An algorithm to select the most convenient NoSQL DBMS for a given application. For
860   our current purposes, this algorithm can be used to select the appropriate solution for the
861   COVID-19 patients', medical staff, and organizations data.
862   An API based security system. It is DBMS independent, and may be used to embed
863   security in any NoSQL DBMS. This achieves the goal of securing COVID-19 related
864   data.

9. Conclusions

866 This research paves the way for the use of NoSQL databases to store and protect COVID-19
867 patients' information, removing existing hurdles to their adoption. It does so by understanding
868 the five main categories of NoSQL databases, offering a comparative study among these
869 categories based upon a set of comparison criteria, namely Performance, Scalability, Flexibility,
870 Complexity, Functionality, and Security issues. This comparative study reveals the strengths and
871 weaknesses of each category. The most commonly used database management systems in each
872 category were discussed. DBMSs were discussed from two points of view: the pertained to the
873 information, the second the security analysis. Building on that discussion, a table was built to
874 compare the strengths and weaknesses of each DBMS, according to their comparison criteria are
875 Data Model, Company, Open source, Implementation Programming Language, Stable Release
876 issue dates, Supporting Authentication, Support Authorization, Support Auditing, Support Data
877 Encryption, and Query Types. The table provides individuals users and organizations a clear
878 understanding of the various NoSQL DBMSs. The choice of a NoSQL DBMS will depend on its
879 convenience to store and secure the information of COVID-19 patients. For this purpose, a new
880 decision-support algorithm was developed, to assist in the selection process of the most secure
881 and convenient NoSQL DBMS to store and maintain the security of the information collected on
882 each individual patient.
883 The paper developed and presented a complete solution to the essential security problems
884 common to all types of NoSQL DBMSs, thus successfully resolving their weaknesses, which
885 previously kept them from being the favored storage solution for sensitive data. Those solutions
886 relate to the processes of authentication, authorization, auditing, and encryption of sensitive
887 information. Moreover, the proposed solution is sufficiently general to allow the patients'
888 database designer to freely choose any NoSQL DBMS to implement their design, while
889 successfully resolving those security pitfalls.

References

891 **Abed Amira Hassan. (2020).** "Recovery and Concurrency Challenging in Big Data and NoSQL
892 Database Systems." 11(04): 4321-4329.
893 **Angles Renzo. 2012.** A comparison of current graph database models. In Data Engineering
894 Workshops (ICDEW), 2012 IEEE 28th International Conference on. IEEE, 171–177.
895 **Apache Software Foundation (2020).** Available at https://directory.apache.org/studio/
896 (accessed 24 April 2020).

897 **Arora Rupali and Rinkle Rani Aggarwal. 2013**. An Algorithm for Transformation of Data
898 from MySQL to NoSQL (MongoDB). International Journal of Advanced Studies in Computer
899 Science and Engineering 2, 1 (2013), 6–12.
900 **Chen, Min, Shiwen Mao, and Yunhao Liu**."Big data: A survey." Mobile networks and
901 applications (2014) 19(2): 171-209.
902 **Chodorow, Kristina.** MongoDB: the definitive guide: powerful and scalable data storage, "
903 O'Reilly Media, Inc."(2013).1–432 pages.
904 **Chang, Fay, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra,**
905 **A. Fikes and R. E. Gruber (2008).** "Bigtable: A distributed storage system for structured data."
906 ACM Transactions on Computer Systems (TOCS) 26(2): 4.
907 **Dayan Niv, Manos Athanassoulis, and Stratos Idreos. 2018**. Optimal Bloom Filters and
908 Adaptive Merging for LSM-Trees. ACM Trans. Database Syst. 43, 4, Article 16 (Dec. 2018), 48
909 pages. https://doi.org/10.1145/3276980.
910 **Deka Ganesh Chandra**. A survey of cloud database systems. IT Professional 16(2) (2014), 50–
911 57.
912 **Dindoliwala, Vaishali. J. and R. D. Morena (2017).** "Survey on Security Mechanisms In
913 NoSQL Databases." International Journal of Advanced Research in Computer Science 8(5).
914 **Elmasri, Ramez. and S. Navathe (2017).** Fundamentals of database systems, Pearson.
915 **Franz Inc.(2020).** AllegroGraph. http://www.franz.com/agraph/allegrograph.2020 (accessed
916 April 23, 2020)
917 **George, S. (2013)**. "NoSQL–NOT ONLY SQL." International Journal of Enterprise Computing
918 and Business Systems 2: 11.
919 **Grolinger, Katarina, Wilson A Higashino, Abhinav Tiwari, Miriam AM Capretz, (2013)**.
920 "Data management in cloud environments: NoSQL and NewSQL data stores." 2(1): 22.
921 **Han, J., E. Haihong, G. Le and J. Du (2011).** Survey on NoSQL database. 2011 6th
922 international conference on pervasive computing and applications, IEEE.
923 **Jim Trevor, Nikhil Swamy, and Michael Hicks. 2007**. Defeating script injection attacks with
924 browser-enforced embedded policies. In Proceedings of the 16th international conference on
925 World Wide Web. ACM, 601–610.

926 **IETF JWT 2020**. Available at https://jwt.io/introduction/ (accessed 24 April 2020).
927 **Kaur, K. and R. Rani (2013).** Modeling and querying data in NoSQL databases. 2013 IEEE
928 International Conference on Big Data, IEEE.
929 **Khasawneh Tariq, Mahmoud AL-Sahlee, Ali Safia (2020).** SQL, NewSQL, and NOSQL
930 Databases: A Comparative Survey (2020**).** 11th International Conference on Information and
931 Communication Systems (ICICS), IEEE.
932 **Kobrix Software Inc. (2020)**. The HyperGraphDB. www.hypergraphdb.org.2020(accessed
933 April 23, 2020).
934 **Lakshman, A. and P. Malik (2010).** "Cassandra: a decentralized structured storage system."
935 ACM SIGOPS Operating Systems Review 44(2): 35-40.
936 **Lasisi, Ayodele Nojeem, Musibau Akintunde Ajagbe. Communications (2012).** "Samba
937 OPENLDAP: An Evolution And Insight." 2(3): 354-362
938 **Lourenço, J. R., B. Cabral, P. Carreiro, M. Vieira and J. Bernardino (2015).** "Choosing the
939 right NoSQL database for the job: a quality attribute evaluation." Journal of Big Data 2(1): 18.
940 **Meier, A. and M. Kaufmann (2019).** SQL & NoSQL Databases, Springer.

941 **Moniruzzaman, A. B. M. and S. A. Hossain (2013)**. "NoSQL Database: New Era of Databases
942 for Big data Analytics - Classification, Characteristics and Comparison." CoRR abs/1307.0191.
943 **Nayak, A., A. Poriya and D. Poojary (2013)**. "Type of NOSQL databases and its comparison
944 with relational databases." International Journal of Applied Information Systems 5(4): 16-19.
945 **Noiumkar Preecha and Tawatchai Chomsiri. 2014.** A comparison the level of security on top
946 5 open source NoSQL databases. In The 9th International Conference on Information
947 Technology and Applications (ICITA2014). 6.
948 **Özsu, M.T. and Valduriez, P., (2020).** NoSQL, NewSQL, and Polystores. In Principles of
949 Distributed Database Systems (pp. 519-557). Springer, Cham.
950 **Pivotal Team 2020.** Available at https://spring.io/projects/spring-boot (accessed 24 April 2020).
951 **Paterson Jim**, **Stefan Edlich, Henrik Hrning, and Reidar Hrning. 2006**. The Definitive
952 Guide to db4o. Springer. 485 pages.
953 **Sahafizadeh, E. and M. A. Nematbakhsh (2015)**. "A survey on security issues in Big Data and
954 NoSQL." Advances in Computer Science: an International Journal 4(4): 68-72.
955 **SecurityFocus (2020).** Available at https://www.securityfocus.com/bid/66474/discuss
956 (accessed 27 April 2020).
957 **Son Sooel, Kathryn S McKinley, and Vitaly Shmatikov. 2013**. Diglossia: detecting code
958 injection attacks with precision and efficiency. In Proceedings of the 2013 ACM SIGSAC
959 conference on Computer & communications security. ACM, 1181–1192.
960 **Strauch, C., U.-L. S. Sites and W. Kriha (2011).** "NoSQL databases." Lecture Notes, Stuttgart
961 Media University 20.
962 **Sullivan Bryan, and Vincent Liu. 2011.** Web application security, a beginner's guide (1 ed.).
963 McGraw-Hill Education Group. 1–352 pages.
964 **Tudorica, B. G. and C. Bucur (2011).** A comparison between several NoSQL databases with
965 comments and notes. 2011 RoEduNet international conference 10th edition: Networking in
966 education and research, IEEE.
967 **Zahid, A., R. Masood and M. A. Shibli (2014)**. Security of sharded NoSQL databases: A
968 comparative analysis. 2014 Conference on Information Assurance and Cyber Security (CIACS),
969 IEEE.
970 **Zaki Asadulla Khan. 2014.** NoSQL databases: new millennium database for big data, big users,
971 cloud computing and its security challenges. International Journal of Research in Engineering
972 and Technology (IJRET) 3, 15 (2014), 403–409.
973 **Zugaj, Wilhelm, and Anita Stefanie Beichler. (2019).** "Analysis of Standard Security Features
974 for Selected NoSQL Systems." 3(2): 41-49.
975

Comparison of NoSQL databases categories

2 *Table 1 comparison of NoSQL databases categories*

NoSQL DB Categories	Performance	Scalability	Flexibility	Complexity	Functionality
Document stores	High	Variable (High)	High	Low	Variable (Low)
Key-Value Stores	High	High	High	None	Variable (None)
Graph DB	Variable	Variable	High	High	Graph Theory
Column stores	High	High	Moderate	Low	Minimal
Object Store	High	Variable (High)	High	Low	Object Oriented Programming

Summarized Table

1   *Table 2 Summarized Table*

DBMS	Model	Company	Open-source	Implementation Programing Language	Stable Release issue date	Support Authentication?	Support Authorization?	Support Auditing?	Support Data Encryption?	Query Types
MongoDB	Document	MongoDB Inc.	Yes	C++, C, JavaScrip	June 2018	Yes	Yes	weak	Yes, but not support data at rest encryption	Mongo Query Language, REST, MapReduce
CouchDB	Document	apache	Yes	Erlang	November 2017	Yes	Yes	weak	No, only Client/Server SSL-based	Map reduce, REST
Couchbase Server	Document	Couchbase Inc.	Yes	Erlang, C, C++	February, 2018	Yes	No	No	No	SQL++, Map reduce, REST
Redis	Key-Value	Salvatore Sanfilippo	Yes	C, C++	June ,2018	weak	No	No	No	REST by Third-party APIs
Voldemort	Key-Value	LinkedIn	Yes	Java	July ,2017	No	No	No	No	Map reduce
Oracle BerkeleyDB	Key-Value	Oracle	No	C, C++, Java	June ,2018	No	No	No	No, only data at rest	SQLite, REST
Neo4j	Graph	Neo Technology	Yes	Java	April ,2018	No	No	No	No, only Client/Server SSL-based	Cypher, Gremlin and SparQL, REST
HyperGraphDB	Graph	Kobrix Software, Inc.	Yes	Java	May, 2017	No	No	No	No	SQL like querying, REST
AllegroGraph	Graph	Franz Inc.	No	Common Lisp	October, 2017	Yes	Yes	weak	No, only Client/Server HTTPS	SparQL and Prolog, REST
HyperTable	Column-based	Zvents before	Yes	C++	March ,2016	No	No	No	No	HQL
HBase	Column-based	apache	Yes	Java	April,2018	Yes	Yes	Yes	No, only Client/Server	Map reduce, REST
GemStone	Object Store	GemTalk	Yes	Smalltalk	June, 2018	Yes	Yes	weak	No	No querying like SQL
db4o	Object Store	Actian	Yes	Java, C#	Sep. ,2011	weak	Yes	No	weak	db4o-sql
Objectivity/DB	Object Store	Objectivity Inc	No	Java, C#, C++ and Paython	June 2017	Yes	weak	No	No	OQL
Cassandra	Multi-model	apache	Yes	Java	February, 2018	Yes	Yes	weak	Yes	CQL, Map reduce, REST by Third-party APIs
DynamoDB	Multi-model	Amazon	No	Java	2012	No	No	weak	No	Map reduce REST

Figure 1

NoSQL Database Management Systems Classification
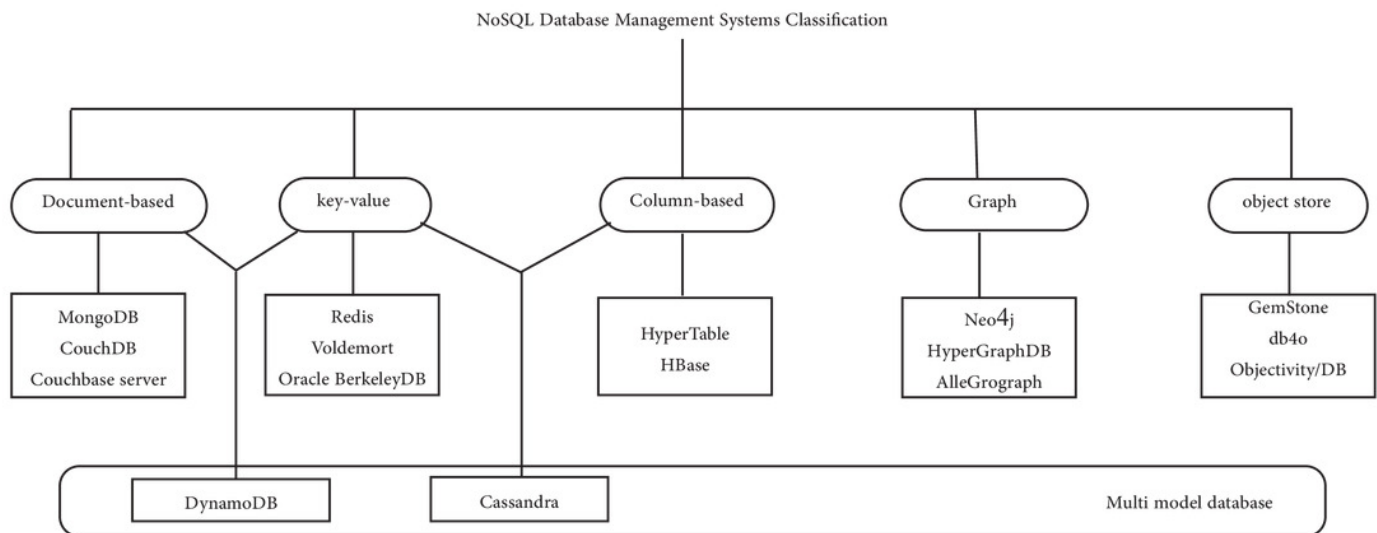
Figure 2

Authentication processes (a)

Figure 3

Authentication processes (b)

Figure 4

Authorization processes

Figure 5

Auditing process

Figure 6

Encryption processes
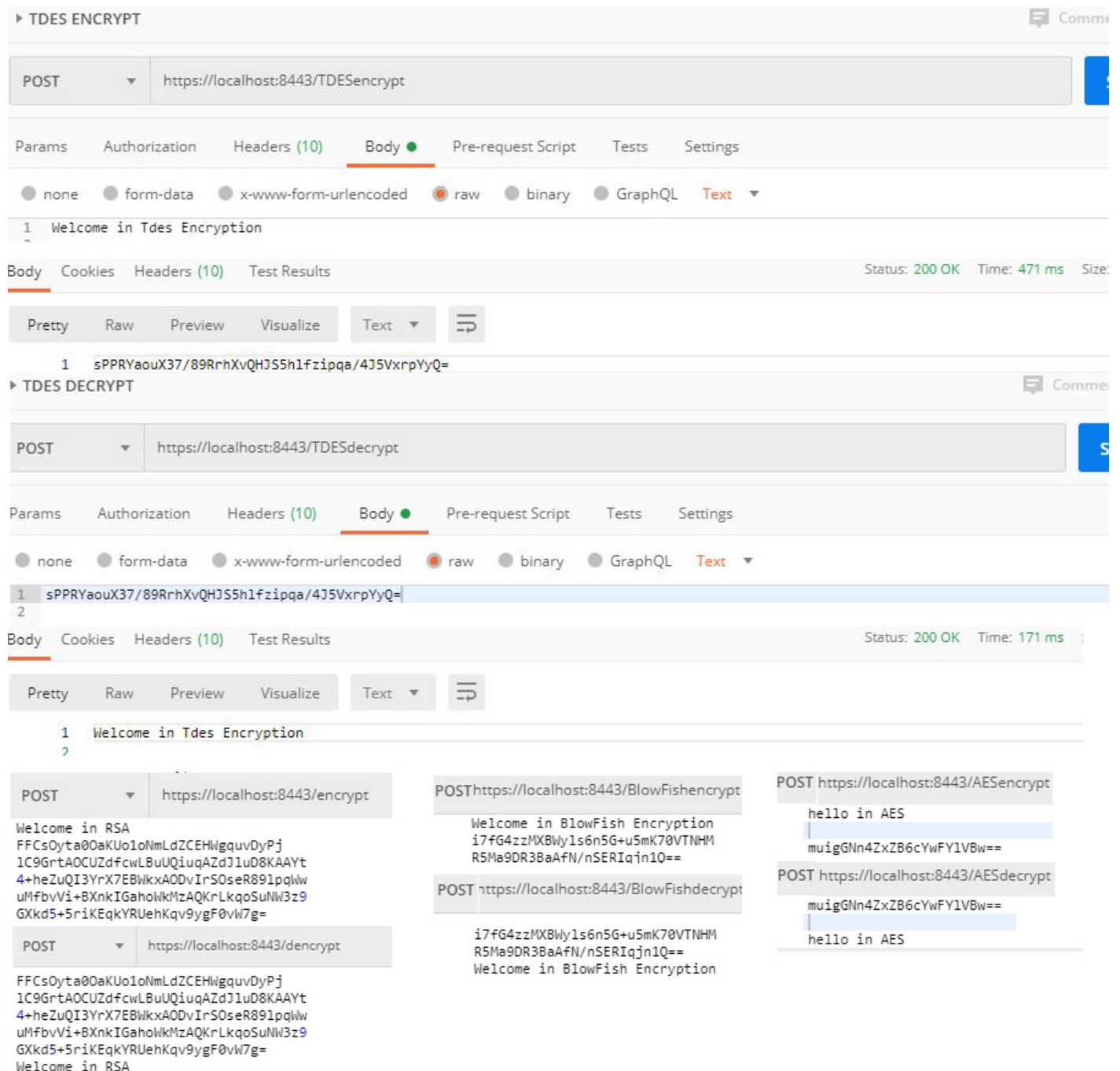
Figure 7

An API based security system