

# FINAL PROJECT REPORT

## HEALTHCARE INDUSTRY MANAGEMENT SYSTEM

**Pradyoth Singenahalli  
Prabhu**  
*Roll No: 42*  
*Student ID: 02071847*  
Data Science  
University Massachusetts  
Dartmouth, Dartmouth

**Bharath Anand**  
*Roll No: 02*  
*Student ID: 02044023*  
Data Science  
University Massachusetts  
Dartmouth, Dartmouth

**Varun W R**  
*Roll No: 49*  
*Student ID: 02070206*  
Data Science  
University Massachusetts  
Dartmouth, Dartmouth

## **Abstract:**

The healthcare industry is rapidly adopting new technologies to improve patient outcomes, increase efficiency, and reduce costs. One of the key challenges faced by healthcare organizations is managing vast amounts of data generated by patients, healthcare providers, and medical devices. Traditional relational databases have limitations in handling the volume and variety of data generated in healthcare. NoSQL databases have emerged as a solution to these challenges, offering scalability, high availability, and data security.

This project aimed to develop healthcare management system using NoSQL databases - DynamoDB, MongoDB, and CouchDB. The system is designed to efficiently manage patient data, medical records, and related information, providing a single platform for healthcare providers, patients, and other stakeholders to access and manage healthcare data. The system incorporates features such as data creation, loading, updating, and querying, and ensures data security and privacy.

The project involved the use of three different categories of NoSQL systems, each with a specific use case. DynamoDB was used to store data related to drugs, hospitals, insurance, patients, and pharmacy. MongoDB was used to store patient visit details, and CouchDB was used to store patient lab reports and diagnosis details. The system was designed to be scalable and highly available, ensuring that healthcare providers could access patient data in real-time, from any location.

The project contributes to the growing use of NoSQL databases in healthcare and demonstrates their potential to provide efficient management of large volumes of unstructured data. The use of NoSQL databases can help healthcare organizations overcome the limitations of traditional relational databases, enabling them to manage large volumes of data, improve patient outcomes, and reduce costs. Overall, this project demonstrates the potential of NoSQL databases to transform the healthcare industry and improve patient care.

## Introductions:

The healthcare industry generates vast amounts of data daily, including patient records, medical histories, prescribed medications, and demographic information. Efficient management of this data is critical to providing quality patient care, reducing costs, and ensuring compliance with regulatory requirements. Traditional relational databases have been the preferred choice for healthcare industry management systems. However, these databases have limitations in handling large volumes of unstructured data, leading to reduced efficiency and increased costs.

In recent years, NoSQL databases have emerged as a viable alternative to relational databases for managing large volumes of unstructured data. NoSQL databases offer several advantages, including scalability, flexibility, and the ability to handle complex data structures.

This project aims to enhance the effectiveness of healthcare industry management systems by incorporating NoSQL databases. The system will facilitate the efficient management of treatment records, prescribed medications, confidential patient demographics, and other relevant data. By utilizing NoSQL databases, the project aims to enhance scalability and flexibility in managing substantial amounts of unstructured data in a more economical and effective manner.

The system's primary goal is to efficiently organize treatment records, prescribed medications, and confidential patient demographics. Healthcare professionals will be able to access patient information quickly and accurately, leading to better patient care. This project will contribute to the growing use of NoSQL databases in healthcare and provide valuable insights into the benefits of incorporating NoSQL databases in healthcare industry management systems.

## Part 1: Incorporating Multiple Data Store Types in the Database-driven Application.

The planned database-driven application will make use of three different data store paradigms: DynamoDB, MongoDB, and CouchDB. Each data store will be used to manage different types of healthcare data, as follows:

DynamoDB will be used to manage structured data, including DRUGS, HOSPITAL, INSURANCE, PATIENTS, PHARMACY, and PATIENTS demographic data.

MongoDB will be used to manage semi-structured data, specifically patient visits to the hospital. MongoDB provides flexibility in managing complex data structures, making it ideal for storing and managing the medical records associated with patient visits.

CouchDB will be used to manage unstructured data, including lab reports that may include PDFs. CouchDB provides offline access and synchronization capabilities, making it ideal for managing

data that needs to be accessed in locations where internet connectivity may be limited or inconsistent.

Incorporating these three data store paradigms into the database-driven application will enhance its capabilities to manage various types of healthcare data more efficiently. The application will also be able to scale horizontally and vertically, thereby accommodating the increasing amount of data generated in the healthcare industry.

## Part - 2: Working Incorporation of 1st Data Store Paradigm – DynamoDB

To demonstrate the working incorporation of the first data store paradigm, DynamoDB, we have implemented the database-driven application with the following features:

- Creation, loading, updating, and querying of DRUGS, HOSPITAL, INSURANCE, PATIENTS, PHARMACY, and PATIENTS demographic data in DynamoDB.

The screenshots below demonstrate the working implementation of the DynamoDB database-driven application.

### Create Table:

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The 'Table details' step is active. The 'Table name' field contains 'DRUGS'. The 'Partition key' section shows 'name' as the key type. The 'Sort key - optional' section is present but empty. At the bottom, there is a 'Table settings' section which is currently collapsed. The top navigation bar includes 'CloudShell', 'Feedback', 'Language', and links for 'Privacy', 'Terms', and 'Cookie preferences'.

## Loading item:

The screenshot shows the AWS DynamoDB console with the 'Attributes' tab selected. The item details are displayed in JSON format:

```
1 [ {  
2   "drugs_id": {  
3     "N": "10"  
4   },  
5   "name": {  
6     "S": "Basaglar"  
7   },  
8   "DRUGS_DISEASE_TREATED": {  
9     "S": "Diabetes"  
10  },  
11  "DRUG_ACTIVE_INGREDIENT": {  
12    "S": "Insulin Glargine"  
13  },  
14  "DRUG_TYPE": {  
15    "S": "Branded"  
16  },  
17  "METHOD_OF_ADMIN": {  
18    "S": "Injection"  
19  }  
20 } ]
```

Below the JSON pane, status information is shown: JSON, Ln 1, Col 1, Errors: 0, Warnings: 0. At the bottom right are 'Cancel' and 'Save changes' buttons. The footer includes links for CloudShell, Feedback, Language, and cookie preferences.

The screenshot shows the AWS DynamoDB console with the 'Items returned' table view. The table displays 10 items with the following data:

	drugs_id	name	DRUG_ACTIVE_INGREDI...	DRUG_T...	DRUGS_DISEASE_TREA...
	8	Amoxicillin	Amoxicillin Trihydrate	Generic	Bacterial Infections
	10	Basaglar	Insulin Glargine	Branded	Diabetes
	3	Proair	Albuterol Sulfate	Branded	Asthma
	2	Lisinopril	Prinivil	Generic	High Blood Pressure
	9	Gabapentin	Gabapentin	Generic	Seizures
	4	Azithromycin	Azithromycin Dihydrate	Generic	Bacterial Infections
	6	Losartan Potassium	Hydrochlorothiazide	Generic	High Blood Pressure
	1	Ventolin	Salbutamol	Branded	Asthma
	5	Omeprazole	Omeprazole Magnesium	Generic	Acid Reflux

## Update Item:

The screenshot shows the AWS DynamoDB console with the URL [voclabs/user2486983=psingenahalliprabhu@umassd.edu @ 4169-296...](#). The navigation bar includes 'Services', 'Search', '[Option+S]', and 'N. Virginia'. The main area shows 'DynamoDB > Items: HOSPITAL > Edit item'. The title 'Edit item' is displayed, with a note: 'You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep.' Below is a table titled 'Attributes' with columns 'Attribute name', 'Value', and 'Type'. The table contains four items:

Attribute name	Value	Type
hospital_id - Partition key	3	Number
name - Sort key	Hospital C	String
address	Fair Haven	String
phone	3122870875	Number

Buttons at the bottom include 'Cancel' and 'Save changes'.

## Query Item:

### Query 1:

```
SELECT * FROM DRUGS WHERE name='Sertraline'
```

The screenshot shows the 'Items returned (1)' section of the DynamoDB console. A search bar 'Find items' is present. The table has columns: drugs\_id, DRUG\_T..., METHOD\_OF\_A..., DRUGS\_DISEASE\_TREA..., name, and DRUG\_ACTIVE\_INGREDIENT. One item is listed:

drugs_id	DRUG_T...	METHOD_OF_A...	DRUGS_DISEASE_TREA...	name	DRUG_ACTIVE_INGREDIENT
7	Generic	Orally ingested	Depression	Sertraline	Sertraline Hydrochloride

Buttons include 'Download results to CSV', '< 1 >', and a refresh icon.

### Query 2:

```
SELECT * FROM HOSPITAL WHERE name='Hospital A'
```

The screenshot shows the 'Items returned (1)' section of the DynamoDB console. A search bar 'Find items' is present. The table has columns: hospital\_id, address, phone, and name. One item is listed:

hospital_id	address	phone	name
1	New Bedford	2125977488	Hospital A

Buttons include 'Download results to CSV', '< 1 >', and a refresh icon.

## Part - 3 (Working Incorporation of 2nd Data Store Paradigm) – MongoDB

For the second data store paradigm, we have used MongoDB to store patients' visits to the hospital. The structure of the data stored in MongoDB is as follows:

- `_id`: The unique ID of the document.
- `id`: A unique ID for each visit.
- `visit_date`: Date of the visit.
- `visit_time`: Time of the visit.
- `reason`: The reason for the visit.
- `doctor`: The doctor who attended to the patient during the visit.
- `location`: Information about the hospital where the visit took place.
  - `hospital_id`: The unique ID of the hospital.
  - `name`: The name of the hospital.
  - `address`: The address of the hospital.
  - `city`: The city where the hospital is located.
  - `state`: The state where the hospital is located.
  - `zip`: The ZIP code of the hospital.
- `patient`: Information about the patient.
  - `patient_id`: The unique ID of the patient.
  - `first_name`: The first name of the patient.
  - `last_name`: The last name of the patient.
  - `address`: The address of the patient.
  - `city`: The city where the patient lives.
  - `state`: The state where the patient lives.
  - `zip`: The ZIP code of the patient.
  - `phone`: The phone number of the patient.
- `insurance`: Information about the insurance policy held by the patient.
  - `insurance_id`: The ID of the insurance policy.
- `diagnosis`: Diagnosis details related to the patient.
  - `diagnosis_id`: The unique ID of the diagnosis.
  - `name`: The name of the diagnosis.
  - `description`: A description of the diagnosis.
- `medication`: Medication details related to the patient.
  - `medication_id`: The unique ID of the medication.
  - `name`: The name of the medication.
  - `description`: A description of the medication.
- `procedure`: Procedure details related to the patient.
  - `procedure_id`: The unique ID of the procedure.
  - `name`: The name of the procedure.
  - `description`: A description of the procedure.
- `notes`: Any notes about the patient's condition or treatment.

Screenshots of the working database-driven application with MongoDB integration have been provided below. These screenshots demonstrate that the application is able to create, load, update, and query data in accordance with our requirements.

## Create Collection:

The screenshot shows the MongoDB Compass interface. A modal window titled "Create Database" is open. In the "Database Name" field, "medical" is entered. In the "Collection Name" field, "visits" is entered. There are two checkboxes: "Time-Series" (unchecked) and "Capped Collection" (unchecked). Below the checkboxes is a section titled "Additional preferences (e.g. Custom collation, Capped, Clustered collections)". At the bottom right of the modal are "Cancel" and "Create Database" buttons. The background shows a list of databases and collections, with "medical.visits" selected. The status bar at the bottom indicates "10 DOCUMENTS" and "1 INDEXES".

## Load Document:

The screenshot shows the MongoDB Compass interface. A modal window titled "Insert Document" is open, indicating it is for the "medical.visits" collection. The document data is displayed in a code editor-like area:

```
1 {  
2   "id": 1,  
3   "visit_date": "2014-02-01",  
4   "visit_time": "10:00",  
5   "reason": "Headache",  
6   "doctor": "Dr. Narendra Modi",  
7   "location": {  
8     "hospital_id": "1",  
9     "name": "HospitalA",  
10    "address": "123 Main St.",  
11    "city": "Anytown",  
12    "state": "NY",  
13    "zip": "12345"  
14  },  
15  "patient": {  
16    "patient_id": "1",  
17    "first_name": "John",  
18  }  
}
```

At the bottom right of the modal are "Cancel" and "Insert" buttons. The background shows a list of documents in the "medical.visits" collection, with one document partially visible. The status bar at the bottom indicates "10 DOCUMENTS" and "1 INDEXES".

The screenshot shows the MongoDB Compass interface connected to the database 'medical' and collection 'visits'. The left sidebar lists databases like admin, config, local, med-db, and medical, with 'visits' selected. The main area displays two documents:

```
_id: ObjectId('6434ad39da5a7ea7cb6696a7')
id: 1
visit_date: "2014-02-01"
visit_time: "10:00"
reason: "Headache"
doctor: "Dr. Narendra Modi"
location: Object
patient: Object
insurance: Object
diagnosis: Array
medication: Array
procedure: Array
notes: "Patient is allergic to penicillin."
```

```
_id: ObjectId('6434ad39da5a7ea7cb6696a8')
visit_date: "2023-04-10"
visit_time: "14:30"
reason: "Annual Physical Exam"
doctor: "Dr. Lee"
location: Object
patient: Object
insurance: Object
diagnosis: Array
medication: Array
procedure: Array
```

## Update Document:

The screenshot shows the MongoDB Compass interface with the same database and collection setup. The first document is selected for modification. The document details are shown in the main pane, and a yellow banner at the bottom indicates 'Document modified.' A modal window is open at the bottom, showing the updated document with its fields and types:

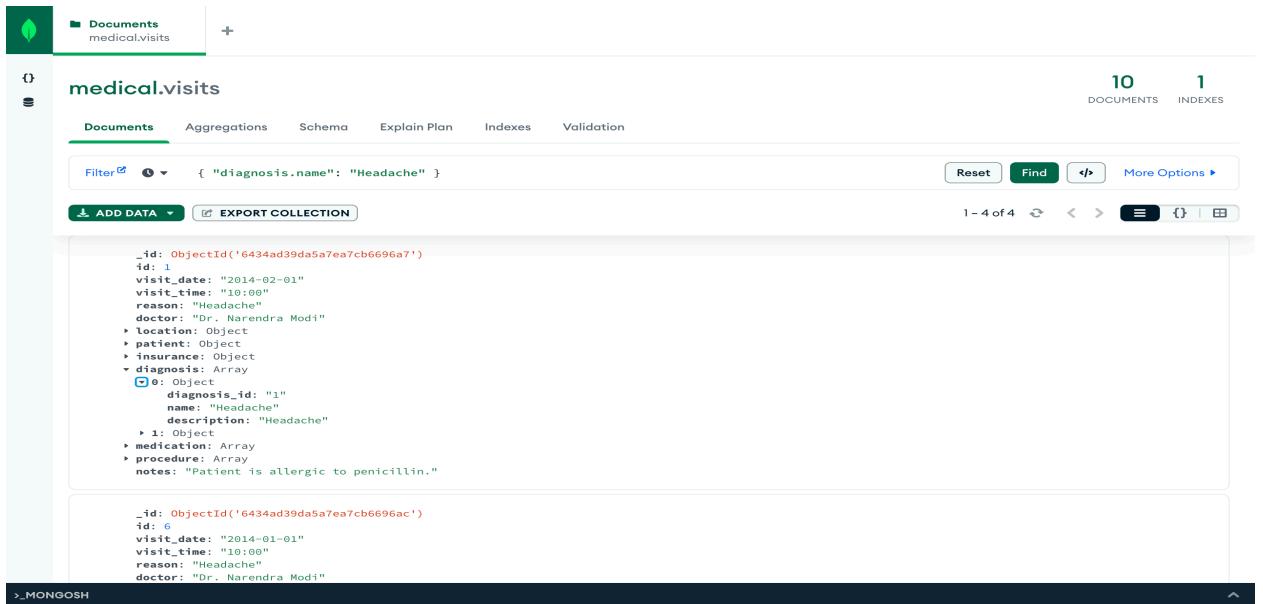
1	_id: ObjectId('6434ad39da5a7ea7cb6696a7')	ObjectId
2	id: 1	Int32
3	visit_date: "2014-02-01"	String
4	visit_time: "10:00"	String
5	reason: "Headache"	String
6	doctor: "Dr. Narendra Modi"	String
7	location: Object	Object
8	patient: Object	Object
9	insurance: Object	Object
10	diagnosis: Array	Array
11	medication: Array	Array
12	procedure: Array	Array
13	notes: "Patient is allergic to penicillin."	String

Buttons for 'CANCEL' and 'UPDATE' are visible at the bottom right of the modal.

## Query Document:

1. Find all patient visits where a particular diagnosis (Headache) was made:

Query: { "diagnosis.name": "Headache" }



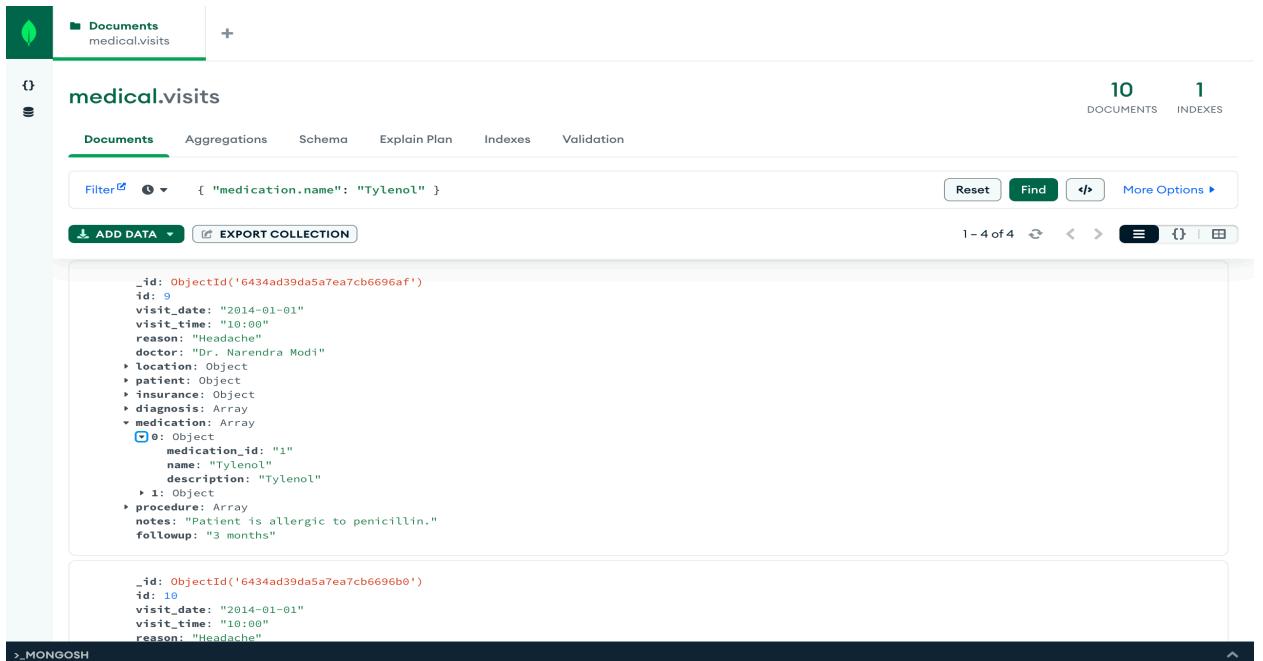
The screenshot shows the MongoDB Compass interface with the 'medical.visits' collection selected. A filter bar at the top contains the query: { "diagnosis.name": "Headache" }. Below the filter, there are two documents listed:

```
_id: 1
id: 1
visit_date: "2014-02-01"
visit_time: "10:00"
reason: "Headache"
doctor: "Dr. Narendra Modi"
location: Object
patient: Object
insurance: Object
diagnosis: Array
  0: Object
    diagnosis_id: "1"
    name: "Headache"
    description: "Headache"
  1: Object
medication: Array
procedure: Array
notes: "Patient is allergic to penicillin."
```

```
_id: ObjectId('6434ad39da5a7ea7cb6696ac')
id: 6
visit_date: "2014-01-01"
visit_time: "10:00"
reason: "Headache"
doctor: "Dr. Narendra Modi"
```

2. Find all patient visits where a particular medication (Tylenol) was prescribed:

Query: { "medication.name": "Tylenol" }



The screenshot shows the MongoDB Compass interface with the 'medical.visits' collection selected. A filter bar at the top contains the query: { "medication.name": "Tylenol" }. Below the filter, there are two documents listed:

```
_id: ObjectId('6434ad39da5a7ea7cb6696af')
id: 9
visit_date: "2014-01-01"
visit_time: "10:00"
reason: "Headache"
doctor: "Dr. Narendra Modi"
location: Object
patient: Object
insurance: Object
diagnosis: Array
medication: Array
  0: Object
    medication_id: "1"
    name: "Tylenol"
    description: "Tylenol"
  1: Object
procedure: Array
notes: "Patient is allergic to penicillin."
followup: "3 months"
```

```
_id: ObjectId('6434ad39da5a7ea7cb6696b0')
id: 10
visit_date: "2014-01-01"
visit_time: "10:00"
reason: "Headache"
```

## Part - 4 (Working Incorporation of 3rd Data Store Paradigm) – CouchDB

For the third data store paradigm, we have used CouchDB to store patients' lab reports including PDFs.

Here are the details of the fields for a report in the CouchDB database:

- `_id`: The unique identifier for the document.
- `_rev`: The revision number of the document.
- `visit_id`: The unique identifier of the visit associated with the report.
- `procedure`: An array of procedures performed during the visit, with details about each procedure.
- `procedure_date`: The date when the procedure was performed.
- `procedure_time`: The time when the procedure was performed.
- `procedure_notes`: Any notes related to the procedure.
- `procedure_status`: The status of the procedure, such as "completed" or "pending".
- `procedure_result`: The result of the procedure, such as "normal" or "abnormal".
- `procedure_result_notes`: Any notes related to the result of the procedure.
- `procedure_result_date`: The date when the result of the procedure was determined.
- `procedure_result_time`: The time when the result of the procedure was determined.
- `lab`: Details about the laboratory where the procedure was performed, including its name, address, and contact information.
- `lab_pathologist`: Details about the pathologist who interpreted the results of the procedure, including their name and contact information.

Create Database:

The screenshot shows the Fauxton interface for managing CouchDB databases. On the left is a sidebar with various icons for database management. The main area is titled 'Databases' and lists existing databases: '\_replicator', '\_users', and 'visits-report'. Each entry includes columns for Name, Size, # of Docs, and Partitioned status. A 'Create Database' button is visible at the top right of the list. To the right of the list, a modal window is open for creating a new database. The modal has a title 'Create Database' and a 'Database name' input field containing 'visits-report'. Below it is a 'Partitioning' section with two radio buttons: 'Non-partitioned - recommended for most workloads' (unchecked) and 'Partitioned' (checked). A note below explains partitioning: 'This is an advanced feature. If you are unsure whether you need a partitioned database, you probably do not. A partitioned database requires a partition key for every document, where the document \_id format is <partition\_key>:<doc\_key>'. At the bottom of the modal are 'Cancel' and 'Create' buttons.

Name	Size	# of Docs	Partitioned
_replicator	0 bytes	0	No
_users	2.3 KB	1	No
visits-report	2.4 MB	8	No

Create Database

Database name: visits-report

Partitioning

Non-partitioned - recommended for most workloads

Partitioned

Which should I choose?

This is an advanced feature. If you are unsure whether you need a partitioned database, you probably do not. A partitioned database requires a partition key for every document, where the document \_id format is <partition\_key>:<doc\_key>. A partition is a logical grouping of documents. Partition queries are often faster than global ones.

Cancel Create

## Insert Document:

The screenshot shows the Fauxton interface for editing a document. The top navigation bar displays the database name "visits-report" and the document ID "bc41859104e2c7eb13f9723d5a00fb1b". Below the navigation is a toolbar with icons for Save Changes, Cancel, View Attachments, Upload Attachment, Clone Document, and Delete. The main area contains the JSON document content:

```

1 - {
2   "_id": "bc41859104e2c7eb13f9723d5a00fb1b",
3   "_rev": "7-71f995a03c778c61fba092d5bc51614",
4   "visit_id": 1,
5   "procedure": [
6     {
7       "procedure_id": "1",
8       "name": "X-Ray",
9       "description": "left leg X-Ray"
10    },
11    {
12      "procedure_id": "2",
13      "name": "MRI",
14      "description": "MRI"
15    }
16  ],
17  "procedure_id": "1",
18  "procedure_date": "2018-01-01",
19  "procedure_time": "10:00:00",
20  "procedure_notes": "X-Ray of the left knee",
21  "procedure_status": "Completed",
22  "procedure_result": "Normal",
23  "procedure_result_notes": "No abnormalities found",
24  "procedure_result_date": "2018-01-01",
25  "procedure_result_time": "10:00:00",
26  "lab": {
27    "lab_id": "1",

```

The bottom left corner shows the Fauxton version "v.3.3.1" and links for "What's New?" and "Log Out".

The screenshot shows the Fauxton interface for managing documents. The top navigation bar displays the database name "visits-report". Below the navigation is a toolbar with icons for All Documents, Run A Query with Mango, Permissions, Changes, Design Documents, and a Create Document button. The main area shows a table of documents with columns: \_id, lab, lab\_pathologist, procedure, and procedure\_date. The table lists 15 documents, showing the first 8:

	_id	lab	lab_pathologist	procedure	procedure_date
<input type="checkbox"/>	bc41859104e2c7...	{ "lab_id": "1", "na...	{ "pathologist_id":...	[ { "procedure_id":...	2018-01-01
<input type="checkbox"/>	bc41859104e2c7...	{ "lab_id": "3", "na...	{ "pathologist_id":...	[ { "procedure_id":...	2018-01-01
<input type="checkbox"/>	bc41859104e2c7...	{ "lab_id": "4", "na...	{ "pathologist_id":...	[ { "procedure_id":...	2018-01-01
<input type="checkbox"/>	bc41859104e2c7...	{ "lab_id": "5", "na...	{ "pathologist_id":...	[ { "procedure_id":...	2018-01-01
<input type="checkbox"/>	bc41859104e2c7...	{ "lab_id": "6", "na...	{ "pathologist_id":...	[ { "procedure_id":...	2018-01-01
<input type="checkbox"/>	bc41859104e2c7...	{ "lab_id": "2", "na...	{ "pathologist_id":...	[ { "procedure_id":...	2018-01-01
<input type="checkbox"/>	bc41859104e2c7...	{ "lab_id": "7", "na...	{ "pathologist_id":...	[ { "procedure_id":...	2018-01-01
<input type="checkbox"/>	bc41859104e2c7...	{ "lab_id": "7", "na...	{ "pathologist_id":...	[ { "procedure_id":...	2018-01-01

At the bottom, there are links for "Showing 5 of 15 columns.", "Show all columns.", "Showing document 1 - 8.", "Documents per page: 20", and navigation arrows.

Using HTTP Rest API to get all documents:

Using Postman application:

The screenshot shows the Postman interface with a project named "DB-Project / all\_doc". A GET request is selected with the URL {{BASE\_URL}}/visits-report/\_all\_docs. The "Body" tab is active, showing "none" selected. The response body is displayed in "Pretty" format, showing a JSON array of documents with fields like id, key, value, and rev.

```
2   "total_rows": 8,
3   "offset": 0,
4   "rows": [
5     {
6       "id": "bc41859104e2c7eb13f9723d5a00fb1b",
7       "key": "bc41859104e2c7eb13f9723d5a00fb1b",
8       "value": {
9         "rev": "7-71f995a03c778c61fbaa92d5hc51614"
10      }
11    },
12    {
13      "id": "bc41859104e2c7eb13f9723d5a0153df",
14      "key": "bc41859104e2c7eb13f9723d5a0153df",
15      "value": {
16        "rev": "2-55562364a216080739a9939f97ee759b"
17      }
18    },
19    {
20      "id": "bc41859104e2c7eb13f9723d5a016462",
21      "key": "bc41859104e2c7eb13f9723d5a016462",
22      "value": {
23        "rev": "1-34acf979d44e71b1d262e948a588d735"
24      }
25    }
26  ]
27 }
```

Using curl :

URL: [http://root:password@localhost:5984/visits-report/\\_all\\_docs](http://root:password@localhost:5984/visits-report/_all_docs)

```
curl -X GET http://root:password@localhost:5984/visits-report/_all_docs

{"total_rows":8,"offset":0,"rows":[
{"id": "bc41859104e2c7eb13f9723d5a00fb1b", "key": "bc41859104e2c7eb13f9723d5a00fb1b", "value": {"rev": "7-71f995a03c778c61fbaa92d5hc51614"}, "id": "bc41859104e2c7eb13f9723d5a0153df", "key": "bc41859104e2c7eb13f9723d5a0153df", "value": {"rev": "2-55562364a216080739a9939f97ee759b"}, "id": "bc41859104e2c7eb13f9723d5a016462", "key": "bc41859104e2c7eb13f9723d5a016462", "value": {"rev": "2-5cfb91ae142b3e9e81d0fbfae3fc51"}, "id": "bc41859104e2c7eb13f9723d5a017adff", "key": "bc41859104e2c7eb13f9723d5a017adff", "value": {"rev": "1-b30bcd2307bd705d83bc0008ca690e20"}, "id": "bc41859104e2c7eb13f9723d5a018ca2", "key": "bc41859104e2c7eb13f9723d5a018ca2", "value": {"rev": "1-34acf979d44e71b1d262e948a588d735"}, "id": "bc41859104e2c7eb13f9723d5a020b07", "key": "bc41859104e2c7eb13f9723d5a020b07", "value": {"rev": "1-5f410911faf0187ae6cf190c1fadec31c"}, "id": "bc41859104e2c7eb13f9723d5a03117d", "key": "bc41859104e2c7eb13f9723d5a03117d", "value": {"rev": "3-89daab576d30e74d69fd8a8f166f6411"}, "id": "bc41859104e2c7eb13f9723d5a035774", "key": "bc41859104e2c7eb13f9723d5a035774", "value": {"rev": "2-f92e421d923fa185a0fedcd6d027a3a35"}]
```

Using HTTP Rest API to get single document using \_id :

Using Postman application:

The screenshot shows the Postman interface with a project named "DB-Project". A collection named "get\_doc" contains a single GET request. The URL is set to `{{BASE_URL}}/visits-report/bc41859104e2c7eb13f9723d5a00fb1b`. The "Params" tab is selected, showing a "Query Params" table with one entry: "visit\_id": 1. The "Body" tab is selected, showing the response body in "Pretty" format. The response is a JSON object representing a medical visit record with two procedures: X-Ray and MRI. The "Send" button is highlighted in blue.

```
1
2   "_id": "bc41859104e2c7eb13f9723d5a00fb1b",
3   "_rev": "7-71f995a403c778c61fbaa92d5bc51614",
4   "visit_id": 1,
5   "procedure": [
6     {
7       "procedure_id": "1",
8       "name": "X-Ray",
9       "description": "left leg X-Ray"
10      },
11      {
12        "procedure_id": "2",
13        "name": "MRI",
14        "description": "MRI"
15      }
16    ],
17    "procedure_id": "1",
18    "procedure_date": "2018-01-01",
19    "procedure_time": "10:00:00",
20    "procedure_notes": "X-Ray of the left knee",
21    "procedure_status": "Completed",
22    "procedure_result": "Normal",
```

Using CURL:

URL: curl -X GET <http://root:password@localhost:5984/visits-report/bc41859104e2c7eb13f9723d5a00fb1b>

```
curl -X GET http://root:password@localhost:5984/visits-report/bc41859104e2c7eb13f9723d5a00fb1b
{
  "_id": "bc41859104e2c7eb13f9723d5a00fb1b",
  "_rev": "7-71f995a403c778c61fbaa92d5bc51614",
  "visit_id": 1,
  "procedure": [
    {
      "procedure_id": "1",
      "name": "X-Ray",
      "description": "left leg X-Ray"
    },
    {
      "procedure_id": "2",
      "name": "MRI",
      "description": "MRI"
    }
  ],
  "procedure_id": "1",
  "procedure_date": "2018-01-01",
  "procedure_time": "10:00:00",
  "procedure_notes": "X-Ray of the left knee",
  "procedure_status": "Completed",
  "procedure_result": "Normal",
  "lab_id": "1",
  "name": "Lab 1",
  "address": "123 Main St",
  "city": "New York",
  "state": "NY",
  "zip": "10001",
  "phone": "212-555-1212",
  "fax": "212-555-1213",
  "attachments": [
    "Sample-chest-radiography-report-that-includes-a-group-e-mail-account-and-an-office-phone.png"
  ],
  "content_type": "image/png",
  "revpos": 6,
  "digest": "md5-a27FYTxY2INK0r8kVgclZw==",
  "length": 51057,
  "stub": true
}, {
  "content_type": "image/png",
  "revpos": 5,
  "digest": "md5-zb0aWBzmF72k5+nS+Ixw==",
  "length": 132139,
  "stub": true
}, {
  "geo-data.json": {
    "content_type": "application/json",
    "revpos": 3,
    "digest": "md5-YbozJ4jb5Jckw8nLshjIA==",
    "length": 148429,
    "stub": true
  },
  "data.json": {
    "content_type": "application/json",
    "revpos": 2,
    "digest": "md5-YdAy1f4Kexxbdu1u4frfQ==",
    "length": 157451
  },
  "stub": true
}
```

## Update Document:

```
visits-report > bc41859104e2c7eb13f9723d5a017adf
Save Changes Cancel
14     "description": "Urine Test"
15   }
16 ],
17 "procedure_id": "5",
18 "procedure_date": "2018-01-01",
19 "procedure_time": "11:00:00",
20 "procedure_notes": "Blood test and urine test",
21 "procedure_status": "Completed",
22 "procedure_result": "Normal",
23 "procedure_result_notes": "No abnormalities found",
24 "procedure_result_date": "2018-01-01",
25 "procedure_result_time": "10:00:00",
26 "lab": {
27   "lab_id": "5",
28   "name": "Lab 1",
29   "address": "123 Main St",
30   "city": "New York",
31   "state": "NY",
32   "zip": "10001",
33   "phone": "212-555-1212",
34   "fax": "212-555-1213"
35 },
36 "lab_pathologist": {
37   "pathologist_id": "5",
38   "name": "Dr. John Smith",
39   "phone": "212-555-1212",
40   "fax": "212-555-1213"
41 }
```

Query:

### 1. To find lab reports assigned to a specific pathologist (Dr. John Smith):

The provided query is searching for lab reports where the name of the pathologist is "Dr. John Smith". It is using the *selector* syntax of CouchDB, which is a way of specifying search criteria for documents in the database. The query specifies the key *lab\_pathologist.name* to search for the name of the pathologist. The value for this key is set to "Dr. John Smith". Therefore, the query will return all the lab reports that have a pathologist with the name "Dr. John Smith".

Using Postman:

Query:

```
{
  "selector": {
    "lab_pathologist.name": "Dr. John Smith"
  }
}
```

The screenshot shows the Postman application interface. In the top navigation bar, the tabs include Home, Workspaces, API Network, and Explore. The main workspace is titled "My Workspace". A collection named "DB-Project" is selected, containing several requests like "GET all\_doc", "POST create\_doc", etc. The current request is "POST query - pathologist - name" with the URL `((BASE_URL))/visits-report/_find`. The "Body" tab is active, showing the following JSON:

```

1 {
2   "selector": {
3     "lab_pathologist.name": "Dr. John Smith"
4   }
5 }

```

Below the body, the response status is 200 OK with a response time of 20 ms and a size of 8.04 KB. The response body is displayed in Pretty format:

```

1 {
2   "docs": [
3     {
4       "_id": "bc41859104e2c7eb13f9723d5a0fb1b",
5       "_rev": "7-71f995a403c778c61fbaa92d5bc51614",
6       "visit_id": 1,
7       "procedure": [
8         {
9           "procedure_id": "1",
10          "name": "X-Ray",
11          "description": "left leg X-Ray"
12        },
13        {
14          "procedure_id": "2",
15          "name": "MRI",
16          "description": "MRI"
17        }
      ]
    }
  ]
}

```

## Using CURL:

```
curl -X POST http://root:password@localhost:5984/visits-report/_find \
-H 'Content-Type: application/json' \
-d '{"selector": {"lab_pathologist.name": "Dr. John Smith"}}'
```

The terminal window shows the execution of the CURL command and its output. The command is:

```
curl -X POST http://root:password@localhost:5984/visits-report/_find \
-H 'Content-Type: application/json' \
-d '{"selector": {"lab_pathologist.name": "Dr. John Smith"}}'
```

The output is a JSON response with the following structure:

```
{
  "docs": [
    {
      "_id": "bc41859104e2c7eb13f9723d5a0fb1b",
      "_rev": "7-71f995a403c778c61fbaa92d5bc51614",
      "visit_id": 1,
      "procedure": [
        {
          "procedure_id": "1",
          "name": "X-Ray",
          "description": "left leg X-Ray"
        },
        {
          "procedure_id": "2",
          "name": "MRI",
          "description": "MRI"
        }
      ]
    }
  ]
}
```

## Conclusion:

In conclusion, this project involved the development of a healthcare application that allows doctors to manage patient information, diagnoses, medications, and procedures. The application was designed with scalability and reliability in mind, utilizing three different NoSQL databases: DynamoDB, MongoDB, and CouchDB.

DynamoDB was used to store patient and doctor information, as well as diagnosis and medication details. MongoDB was used to store visit and procedure information, providing fast and reliable access to data through its highly scalable and fully managed NoSQL database service. Finally, CouchDB was used to store patient lab reports, including unstructured data like PDFs and images.

The use of NoSQL databases allowed for flexible and dynamic data modeling, enabling quick and efficient retrieval of data. The implementation of multiple data stores ensures high availability, fault tolerance, and disaster recovery capabilities.

Overall, this project demonstrates the power and flexibility of NoSQL databases in developing highly scalable and reliable applications for the healthcare industry.