

| Title: Implementation of MongoDB, Node.js and Express js. |
| --- |

**AIM:** Implementation of MongoDB, Node.js and Express js.

**Problem Definition:**

**Resources used:**

1. **Node.js Official Documentation**
2. **MongoDB Official Documentation**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 4:** **Test the concepts and components of various front-end, back-end web app**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Shelly Powers Learning Node O’ Reilly 2 nd Edition, 2016.

**Pre Lab/ Prior Concepts:**

**Write details about the following content**

* Mongo DB
* Connection using node js Express js And MongoDB

**MongoDB**

MongoDB is a NoSQL database that stores data in a flexible, JSON-like format (BSON). It allows for dynamic schemas, meaning you can easily change the structure of your data without needing to alter existing data or the schema definitions. This flexibility makes MongoDB particularly suitable for applications with evolving data models.

#### Key Features of MongoDB:

1. Document-Oriented: Data is stored as documents in collections, allowing for rich data representation.
2. Scalability: Designed for horizontal scalability, making it easy to distribute data across multiple servers.
3. High Performance: Optimized for read and write operations, making it suitable for large volumes of data.
4. Flexible Schema: No predefined schema is required, allowing different documents within a collection to have different structures.

**Node**

Node.js is a JavaScript runtime built on Chrome's V8 engine, allowing developers to execute JavaScript on the server side. It uses an event-driven, non-blocking I/O model, which makes it efficient and suitable for building scalable network applications.

#### Key Features of Node.js:

1. Asynchronous and Event-Driven: Enables handling multiple connections simultaneously without blocking.
2. Single Language for Full Stack Development: Allows developers to use JavaScript on both the front end and back end.
3. Rich Ecosystem: A vast range of libraries and frameworks are available via npm (Node Package Manager).

**Express**

Express.js is a web application framework for Node.js that simplifies building web applications and APIs. It provides robust features for routing, middleware, and handling requests and responses.

#### Key Features of Express.js:

1. Middleware Support: Allows you to define middleware functions for processing requests, which can handle tasks such as logging, authentication, and error handling.
2. Routing: Provides a powerful routing system to manage different endpoints in your application.
3. Simplicity: Minimalistic design that allows for easy setup and quick development of applications.

### Connecting Node.js and Express.js to MongoDB

1. Database Driver: To connect Node.js with MongoDB, developers typically use an Object Data Modeling (ODM) library called Mongoose. Mongoose provides a straightforward way to interact with MongoDB, including features for schema definition, validation, and query building.
2. Establishing Connection:
   * Use Mongoose to connect to a MongoDB instance. This involves specifying the connection URI, which includes the database location (either local or remote).
   * Handle connection events (e.g., successful connection, errors) to manage application behavior accordingly.
3. Defining Schemas:
   * With Mongoose, define schemas that represent the structure of documents in a collection. This schema can enforce data types, validation rules, and default values.
   * Create models based on these schemas, allowing for CRUD (Create, Read, Update, Delete) operations on the documents.
4. Building APIs:
   * Use Express.js to create RESTful APIs that interact with the MongoDB database. Define routes for different operations (e.g., retrieving all documents, creating a new document).
   * Implement middleware for handling JSON requests and responses.
5. Error Handling:
   * Implement error handling mechanisms to catch and respond to errors related to database operations, ensuring a robust application.

**Methodology:**

* Set up the development environment with Node.js and MongoDB.
* Create a new Node.js project and install necessary dependencies (Express and Mongoose).
* Establish a connection between Node.js and MongoDB using Mongoose.
* Create a basic Express.js server that handles CRUD operations (Create, Read, Update, Delete).
* Implement routes to interact with the database.
* Test the application to ensure data is correctly stored and retrieved.

**Implementation Details:**

Item.js:

const mongoose = require('mongoose');

const itemSchema = new mongoose.Schema({

name: { type: String, required: true },

quantity: { type: Number, required: true }

});

module.exports = mongoose.model('Item', itemSchema);

server.js:

const express = require('express');

const mongoose = require('mongoose');

const Item = require('./models/Item');

const app = express();

const PORT = process.env.PORT || 3000;

*// Middleware*

app.use(express.json());

*// Connect to MongoDB*

mongoose.connect('mongodb://localhost:27017/mydatabase')

.then(() => console.log('MongoDB connected'))

.catch(*err* => console.log(err));

*// Create Item*

app.post('/items', async (*req*, *res*) => {

const item = new Item(*req*.body);

try {

await item.save();

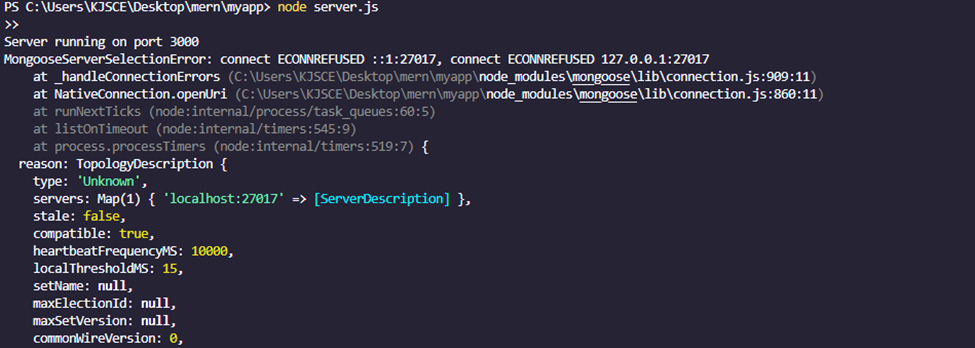
*res*.status(201).send(item);

} catch (error) {

*res*.status(400).send(error);

}

});



**Conclusion:**

**Learnt mongo db , express and node.**