# MongoDB + Node.js (MongoClient) – Beginner Explanation Guide

This document explains **everything from scratch**:

- How to install MongoDB
- What MongoClient is
- How Node.js connects to MongoDB
- How to insert, fetch, and update data

This is written for **absolute beginners**.

---

## 1. What is MongoDB?

MongoDB is a **NoSQL database**. Instead of tables and rows, it stores data as **documents** (JSON-like objects).

Example document:

```
{
  name: "John",
  age: 25
}
```

---

## 2. Install MongoDB (Database Server)

### Step 1: Download MongoDB

Download MongoDB Community Server from: https://www.mongodb.com/try/download/community

### Step 2: Install

Follow the installer steps for your OS (Windows / Mac / Linux).

### Step 3: Start MongoDB

After installation, MongoDB runs on this address:

```
mongodb://127.0.0.1:27017
```

Check if MongoDB is running:

```
mongosh
```

If it opens successfully, MongoDB is running ✅

---

## 3. Install MongoDB Driver for Node.js

Inside your Node.js project:

```
npm init -y
npm install mongodb
```

This installs the **MongoDB Node.js Driver**.

---

## 4. What is MongoClient?

`MongoClient` is a **class provided by MongoDB** that:

- Connects Node.js to MongoDB
- Manages the database connection
- Lets you access databases and collections

Think of it as:

> 💹MongoClient = bridge between Node.js and MongoDB

---

## 5. MongoDB Connection URL

```
const uri = "mongodb://127.0.0.1:27017";
```

- `127.0.0.1` → your local computer
- `27017` → default MongoDB port

---

## 6. Complete Code With Clear Comments

```
// Import MongoClient from mongodb package
const { MongoClient } = require("mongodb");

// MongoDB server address
const uri = "mongodb://127.0.0.1:27017";

// Create MongoClient instance (does NOT connect yet)
```

```
const client = new MongoClient(uri);

// Async function because MongoDB operations take time
async function connection() {
    try {
        // 1️⃣ Connect to MongoDB SERVER
        await client.connect();
        console.log("MongoDB connected");

        // 2️⃣ Connect to DATABASE
        // If it does not exist, MongoDB creates it automatically
        const db = client.db("sample");
        console.log("db connected----", db.databaseName);

        // 3️⃣ Connect to COLLECTION
        // Collection is created when data is inserted
        const users = db.collection("users");
        console.log("users---------", users.collectionName);

        // 4️⃣ INSERT ONE DOCUMENT
        const insertResult = await users.insertOne({
            name: "John",
            age: 25
        });
        console.log("Inserted ID:", insertResult.insertedId);

        // 5️⃣ FETCH ALL DOCUMENTS
        const allUsers = await users.find().toArray();
        console.log("All Users:", allUsers);

        // 6️⃣ UPDATE ONE DOCUMENT
        const updateResult = await users.updateOne(
            { name: "John" },       // filter: find John
            { $set: { age: 26 } }   // update: change age
        );
        console.log("Updated count:", updateResult.modifiedCount);

        // 7️⃣ FETCH UPDATED DOCUMENT
        const updatedUser = await users.findOne({ name: "John" });
        console.log("Updated User:", updatedUser);

    } catch (error) {
        // Error handling
        console.log("mongo error---", error);

    } finally {
        // 8️⃣ Close MongoDB connection
        await client.close();
        console.log("MongoDB connection closed");
    }
}
```

```
// Call the function
connection();
```

---

## 7. MongoDB Flow (How It Works)

```
Node.js
   ↓
MongoClient
   ↓
MongoDB Server
   ↓
Database (sample)
   ↓
Collection (users)
   ↓
Documents
```

---

## 8. CRUD Operations Summary

| Operation | Method |
|---|---|
| Create | `insertOne()` |
| Read | `find()`, `findOne()` |
| Update | `updateOne()` |
| Delete | `deleteOne()` |

---

## 9. Important Beginner Notes

- MongoDB creates databases & collections automatically
- Always use `await` with MongoDB methods
- Updates require operators like `$set`
- Keep one MongoClient connection

---

You now understand **MongoDB + MongoClient from scratch**.

---

# Express REST API + MongoDB Connection (Beginner Guide)

This section explains **how to connect MongoDB to an Express REST API** in the **correct and beginner-friendly way**.

---

## 10. What is Express?

**Express.js** is a **Node.js web framework** used to:

- Create APIs
- Handle HTTP requests (GET, POST, PUT, DELETE)
- Send responses to clients (browser, Postman, frontend)

Think of it as:

> 🚳Express = door between client and database

---

## 11. Install Express

Inside your project folder:

```
npm install express mongodb
```

---

## 12. Recommended Project Structure

```
project/
|── server.js
|── db.js
|── package.json
```

---

## 13. Create MongoDB Connection File (IMPORTANT)

`db.js`

This file connects to MongoDB **once** and reuses the connection.

```
const { MongoClient } = require("mongodb");
const uri = "mongodb://127.0.0.1:27017";

// Create MongoClient
const client = new MongoClient(uri);

let db; // store database connection

async function connectDB() {
  if (db) return db; // reuse existing connection

  await client.connect();
  console.log("MongoDB connected");

  db = client.db("sample");
  return db;
}

module.exports = connectDB;
```

---

## 14. Create Express Server

`server.js`

```
const express = require("express");
const connectDB = require("./db");

const app = express();
app.use(express.json()); // parse JSON body

// Test route
app.get("/", (req, res) => {
  res.send("API running");
});

// GET all users
app.get("/users", async (req, res) => {
  const db = await connectDB();
  const users = await db.collection("users").find().toArray();
  res.json(users);
});

// POST create user
app.post("/users", async (req, res) => {
  const db = await connectDB();
  const result = await db.collection("users").insertOne(req.body);
  res.json(result);
});
```

```
// PUT update user
app.put("/users/:name", async (req, res) => {
  const db = await connectDB();
  const result = await db.collection("users").updateOne(
    { name: req.params.name },
    { $set: req.body }
  );
  res.json(result);
});

// DELETE user
app.delete("/users/:name", async (req, res) => {
  const db = await connectDB();
  const result = await db.collection("users").deleteOne({ name:
req.params.name });
  res.json(result);
});

app.listen(3000, () => {
  console.log("Server running on port 3000");
});
```

---

## 15. How Express + MongoDB Works Together

```
Client (Postman / Browser)
   ↓ HTTP Request
Express API (server.js)
   ↓
MongoDB Connection (db.js)
   ↓
MongoDB Database
```

---

## 16. Test API Using Postman or Browser

### GET all users

```
GET http://localhost:3000/users
```

### POST new user

```
POST http://localhost:3000/users
Body (JSON):
{
  "name": "John",
  "age": 25
}
```

**UPDATE user**

```
PUT http://localhost:3000/users/John
Body (JSON):
{
  "age": 30
}
```

**DELETE user**

```
DELETE http://localhost:3000/users/John
```

---

## 17. Why This Connection Method is Best

- MongoDB connects **only once**
- Express reuses the connection
- Faster performance
- Production-ready pattern

---

You now know how to build a **full Express REST API with MongoDB connection**.

---

# Environment Variables (.env) – Beginner Explanation

This section explains **what environment variables are**, **why we use ********``**, and **how to use them in Node.js + Express + MongoDB**.

---

## 18. What are Environment Variables?

**Environment variables** are values that:

- Are stored **outside your code**
- Hold **sensitive or changeable information**
- Can change between development and production

Examples:

- Database URL

- Port number
- Passwords
- API keys

⛎Instead of writing these directly in code, we store them in `.env`.

---

## 19. Why NOT Hardcode Values?

❌Bad practice:

```
const uri = "mongodb://127.0.0.1:27017";
```

Problems:

- Security risk
- Difficult to change later
- Different values needed for production

✅Good practice:

```
process.env.MONGO_URI
```

---

## 20. What is a `.env` File?

`.env` is a simple text file that stores **key = value** pairs.

Example:

```
PORT=3000
MONGO_URI=mongodb://127.0.0.1:27017
DB_NAME=sample
```

⚠️ `.env` file should **never be pushed to GitHub**.

---

## 21. Install dotenv Package

```
npm install dotenv
```

`dotenv` loads `.env` values into `process.env`.

---

## 22. Create `.env` File

In your project root:

```
project/
|── server.js
|── db.js
|── .env
```

`.env`

```
PORT=3000
MONGO_URI=mongodb://127.0.0.1:27017
DB_NAME=sample
```

---

## 23. Load `.env` in Application

**server.js (TOP of file)**

```
require("dotenv").config();
```

This makes `.env` variables available everywhere.

---

## 24. Use Environment Variables in db.js

`db.js`

```
require("dotenv").config();
const { MongoClient } = require("mongodb");

const client = new MongoClient(process.env.MONGO_URI);

let db;

async function connectDB() {
  if (db) return db;

  await client.connect();
  console.log("MongoDB connected");

  db = client.db(process.env.DB_NAME);
  return db;
}
```

```
module.exports = connectDB;
```

---

## 25. Use Environment Variables in server.js

`server.js`

```
require("dotenv").config();
const express = require("express");
const connectDB = require("./db");

const app = express();
app.use(express.json());

app.listen(process.env.PORT, () => {
  console.log(`Server running on port ${process.env.PORT}`);
});
```

---

## 26. How process.env Works

```
process.env.PORT      // 3000
process.env.MONGO_URI // mongodb://127.0.0.1:27017
process.env.DB_NAME   // sample
```

`process.env` is a global Node.js object.

---

## 27. Add `.env` to .gitignore (IMPORTANT)

`.gitignore`

```
.env
node_modules
```

This prevents secrets from being uploaded.

---

## 28. Environment Variables Flow

```
.env file
   ↓
dotenv package
```

```
    ↓
process.env
    ↓
Used in code
```

## Benefits of Using .env

- Secure credentials
- Easy configuration changes
- Clean code
- Production ready