# Activity: Guide for MongoDB Installation and First Database Setup

Step-by-step guide to help you with zero prior experience set up MongoDB, create your first database, and build a basic NoSQL-supported back-end with an API.

## Part 1: Installing MongoDB

### Step 1: Download MongoDB

- **Go to MongoDB's official website**: MongoDB Community Download.
- **Select your operating system** (Windows, macOS, Linux).
- **Download the MongoDB Installer**.

### Step 2: Install MongoDB

- **Run the downloaded installer**.
- **Follow the installation wizard**:
    1. Choose **"Complete"** setup option.
    2. Ensure **"Install MongoDB as a Service"** is checked.
    3. Click **Next** and then **Install**.

### Step 3: Verify the Installation

- Open the **Command Prompt** or **Terminal**.
- Type `mongos --version` and `mongod --version`.
- If MongoDB is correctly installed, the version details will appear.
- If it's not working, then you can add the (mongos bin) path in in environment variable

## Part 2: Setting Up MongoDB and Creating the First Database

### Step 1: Start the MongoDB Server

- Open the **Command Prompt** (Windows) or **Terminal** (macOS/Linux).
- Type `mongod` and hit Enter.
- You should see a message indicating that MongoDB has started successfully.

### Step 2: Open the MongoDB Shell

- Open another **Command Prompt/Terminal** window.
- Type `mongo` and hit Enter.
- This will connect you to the MongoDB server.

### Step 3: Create Your First Database

- In the MongoDB shell, type the following command:

```
use myFirstDatabase
```

- This command creates a new database named `myFirstDatabase` (if it doesn't exist) and switches to it.

### Step 4: Create a Collection and Insert Data

Open the MongoDB Shell or Connect via MongoDB Compass

- **Option 1: MongoDB Shell**

  - **Launch the MongoDB shell** by typing `mongo` in your command line. This command will start the MongoDB shell and connect to your local database instance

  - You'll see a command prompt like this:

```
MongoDB shell version v4.x.x
>
```

- **Option 2: MongoDB Compass (GUI)**

  - Open **MongoDB Compass**.
  - **Connect to your local MongoDB instance** (e.g., `mongodb://localhost:27017`).
  - Select the **Database** tab to begin interacting with your data.

  - Collections in MongoDB are similar to tables in relational databases.
  - Create a collection named `students` and insert a sample document (record):

```
db.students.insertOne({ name: "John Doe", age: 21, major: "Computer Science" })
```

  - This command inserts a single document into the `students` collection.

### Step 5: View the Inserted Data

- To see the data you just inserted, use the command:

```
db.students.find()
```

- This will display all documents in the `students` collection.

## Part 3: Setting Up NoSQL Database-Supported Back-End Using Node.js

### Step 1: Install Node.js and npm if not installed

- Go to Node.js official website and download the latest version.
- Run the installer and follow the instructions.

### Step 2: Set Up a Basic Node.js Project

- Create a new project folder and navigate to it:

```
mkdir mongodb-backend
cd mongodb-backend
```

- Initialize a new Node.js project:

```
npm init -y
```

- This will create a `package.json` file.

### Step 3: Install Required Packages

- Install the `express` and `mongodb` packages:

```
npm install express mongodb
```

### Step 4: Create the Back-End Server

- Create a new file named `server.js` in the `mongodb-backend` folder.
- Open it in your code editor (e.g., Visual Studio Code) and add the following code:

```
const express = require('express');
const { MongoClient } = require('mongodb');

const app = express();
const PORT = 3000;
const url = "mongodb://localhost:27017";
const dbName = "myFirstDatabase";

let db;

// Connect to MongoDB
MongoClient.connect(url, { useUnifiedTopology: true }, (err, client) =>
{
  if (err) return console.error(err);
  console.log("Connected to MongoDB!");
  db = client.db(dbName);
});

// API to get all students
app.get('/students', (req, res) => {
  db.collection('students').find().toArray((err, results) => {
    if (err) return res.status(500).send(err);
```

```
      res.json(results);
    });
  });

  // Start the server
  app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
  });
```

- **Explanation**:
  - This script connects to your MongoDB database and creates a simple Express.js server.
  - It provides an API endpoint (`/students`) that retrieves all documents from the `students` collection.

### Step 5: Start the Node.js Server

- In the Command Prompt or Terminal, navigate to the `mongodb-backend` folder.
- Run the following command:

```
node server.js
```

- You should see the message: `Server is running on http://localhost:3000`.

### Step 6: Test the API

- Open your browser and go to `http://localhost:3000/students`.
- You should see a JSON array of student records.

## Part 4: Using and Testing the API

### Step 1: Insert More Data Using the API

- Use tools like **Postman** or **curl** to test the API by adding more data.
- Example of adding a new student record using Postman:
  - Use the `POST` method on the URL `http://localhost:3000/students`.
  - Set the request body to:

    ```
    { "name": "Jane Smith", "age": 23, "major": "Data Science" }
    ```

### Step 2: Expand the API Functionality

- Add new endpoints to `server.js` for:
  - **Creating** new students.
  - **Updating** existing student records.

    - **Deleting** a student by ID.

## Summary

- **MongoDB** is a powerful, flexible, and scalable NoSQL database, ideal for handling semi-structured data.
- By following these steps, you will understand how to set up MongoDB, create a basic database, and build a simple back-end application.
- The next step would be exploring more advanced MongoDB concepts such as aggregation, indexing, and performance optimization.