MongoDB—document database (noSQL) - key value pair- JSON- opposite of relation db

**Express**—a Node.js framework for building APIs

Angular—front-end application framework

**Node.js**—server-side JavaScript runtime environment

---------------

In a MEAN stack, the server-side is typically managed using **Node.js** with **Express.js.**

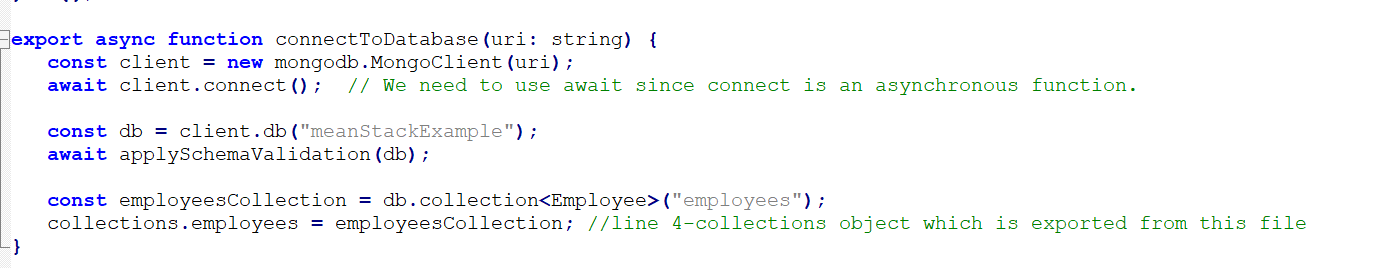
**For mean-stack-example folder to run server:**

**npx ts-node src/server.ts**

**http://localhost:3000/employees**

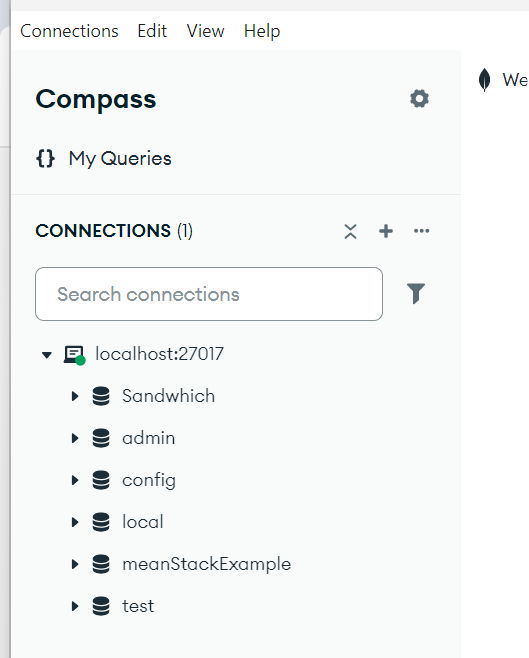
**This will call the /employees endpoint and should return the list of employees if the database is populated.**

* **Ensure that MongoDB is running locally or is properly connected to your cloud database.**

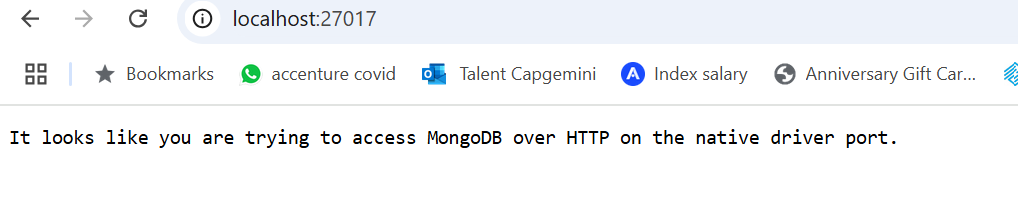
****

**local MongoDB instance**

**mongodb://localhost:27017**

****

[**http://localhost:27017/**](http://localhost:27017/)

****

**How They Work Together**

* **The Express server acts as a middleman between the client and the database:**
  1. **A client sends an HTTP request to http://localhost:5200 (e.g., GET /employees).**
  2. **The Express server handles the request.**
  3. **If the request involves database operations, the server connects to MongoDB (port 27017) to fetch or store data.**
  4. **The server sends the response back to the client.**

**The client never directly connects to MongoDB—it only interacts with the Express server.**

**Important Note**

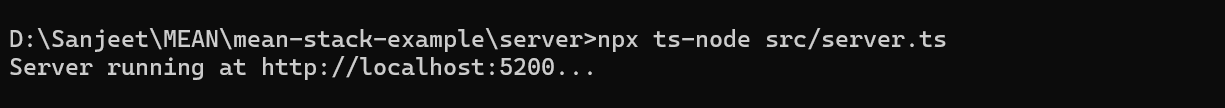
* **The port 5200 is for HTTP requests (frontend ↔ backend).**
* **The port 27017 is for database communication (backend ↔ MongoDB).**
* **Both ports are unrelated, so there’s no conflict.**

**Step-by-step**

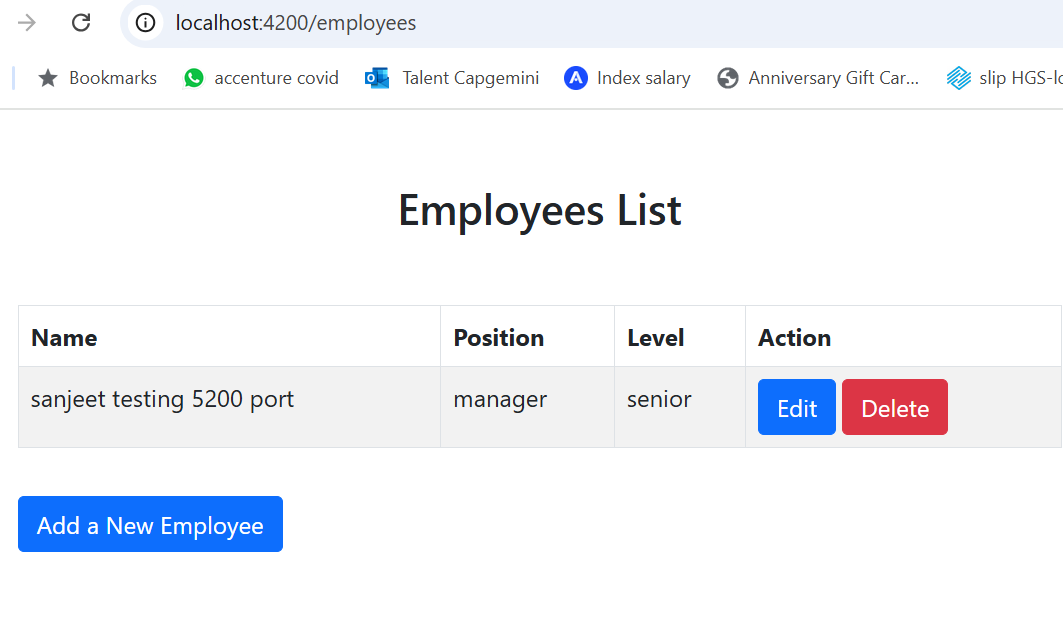
**Configured uri to 27017 localmongodb**

****

**Started express.js server**

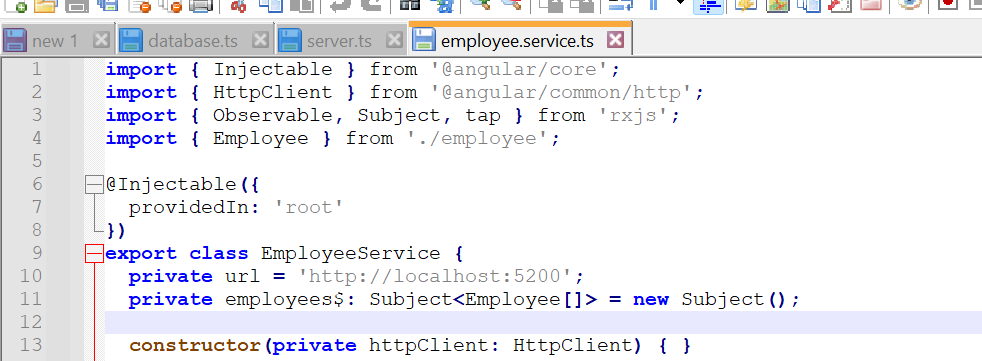
****

**Send data from angular to 5200 port**

****

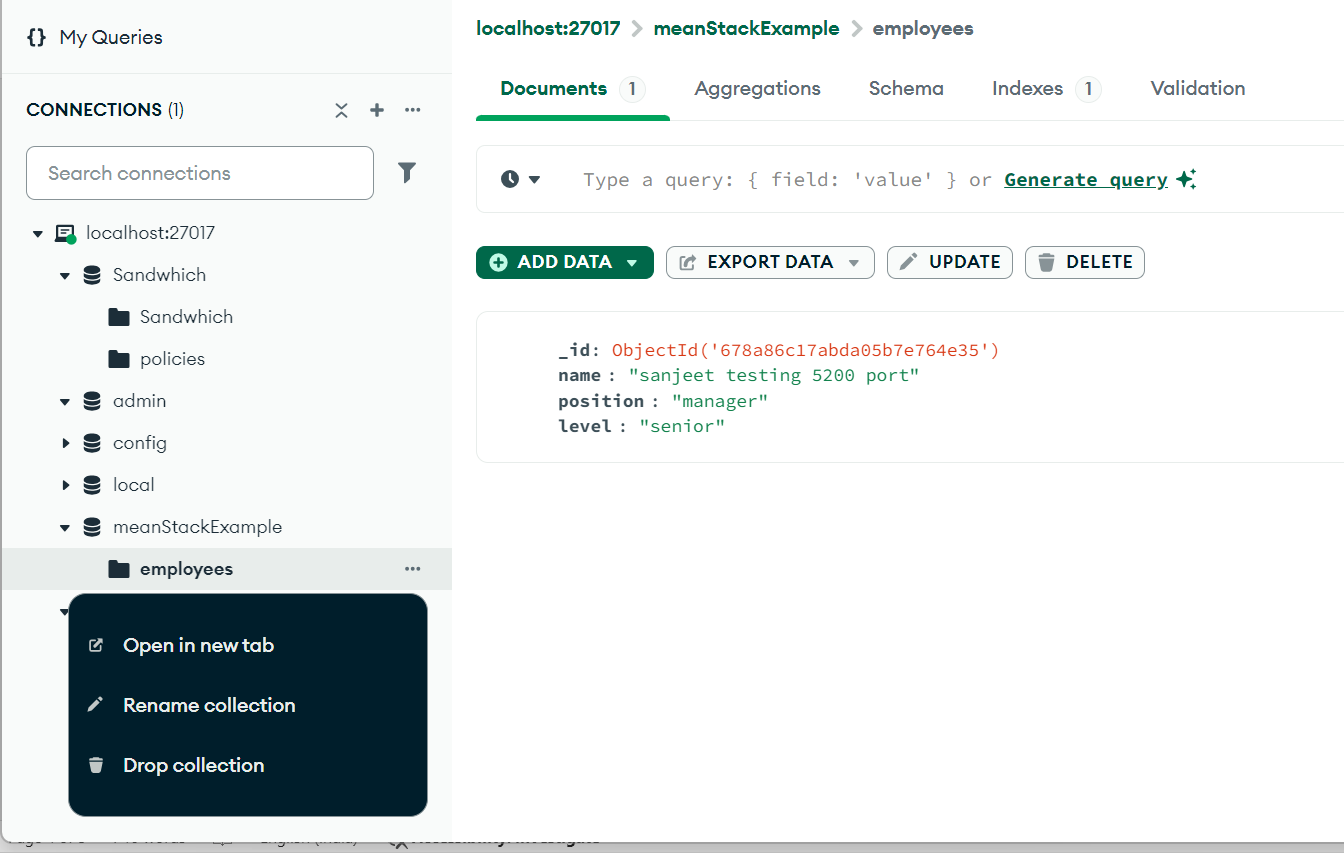
**5200 port as as middleman**

**This is mentioned in employee service file**

****

**Once the employee is added, it reflects in mongodb compass**

**Use open in new tab to see data added**

****

**1. Refresh Data in MongoDB**

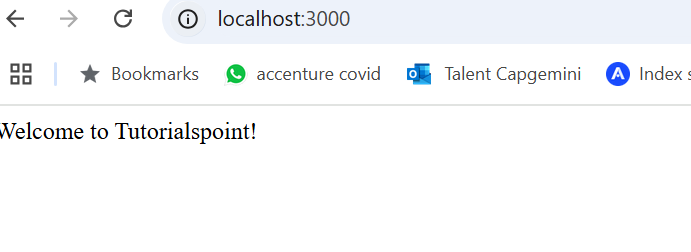
**If you’ve made changes to the database (e.g., inserted, updated, or deleted documents) and want to see the latest data in a GUI tool like MongoDB Compass:**

* **In MongoDB Compass:**
  + **Navigate to the collection you want to view.**
  + **Click the "Refresh" button (usually found near the top of the interface) to reload the latest data.**

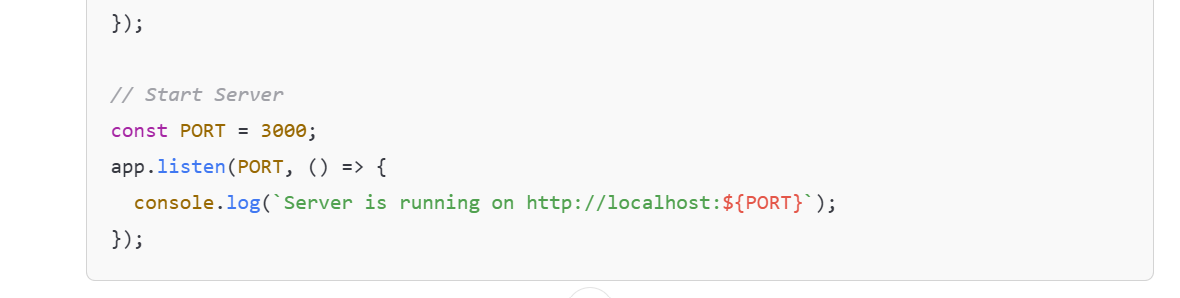
**This is for outer example:**

node server.js









The **server** and **database** are two distinct components in a software system, each serving a specific purpose. Here's a clear comparison:

**1. Definition**

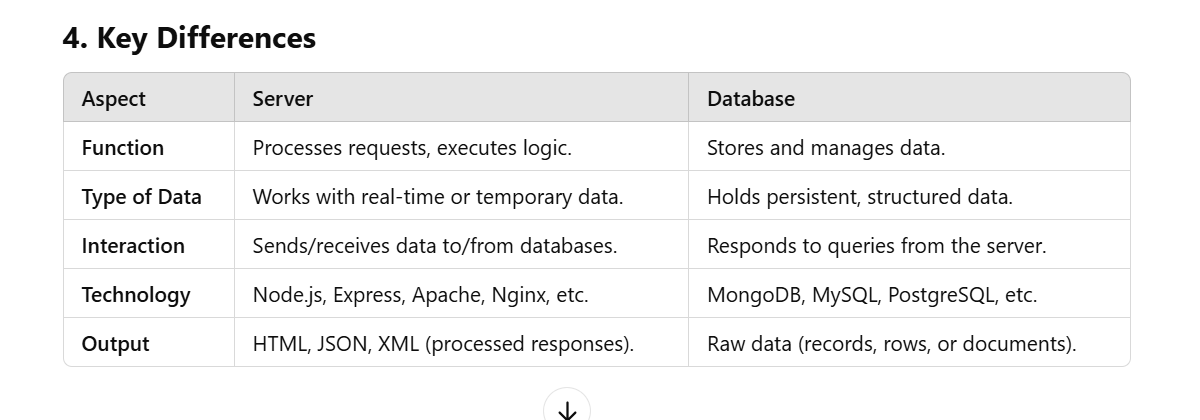
* **Server**:
  + A server is a software or hardware system that provides services or resources to other systems (clients) over a network.
  + Examples: Web servers, application servers, email servers, **cloud servers**, etc.
* **Database**:
  + A database is a structured collection of data that is stored electronically and is designed for easy access, management, and updating.
  + Examples: MySQL, MongoDB, PostgreSQL, **cloud databases** etc.

**2. Purpose**

* **Server**:
  + Facilitates communication between clients (e.g., browsers, mobile apps) and various backend resources (like databases or APIs).
  + Executes logic, processes requests, and sends responses.
* **Database**:
  + Stores, organizes, and retrieves data efficiently.
  + Ensures data integrity, supports queries, and performs CRUD operations (Create, Read, Update, Delete).

**3. Roles in the MEAN Stack**

* **Server** (Node.js with Express):
  + Handles the business logic of the application.
  + Routes client requests (from Angular) to appropriate services or databases.
  + Example: **When a user submits a form, the server processes the data and interacts with the database to save it.**
* **Database** (MongoDB in MEAN):
  + Stores data persistently in a structured format (collections and documents for MongoDB).
  + Example: Stores user data submitted through the form.



**5. Example**

**Scenario: A User Logs In**

1. **Client (Browser or App)**:
   * Sends login credentials (username and password) to the **server**.
2. **Server**:
   * Receives the credentials, validates them, and queries the **database**.
3. **Database**:
   * Checks if the username and password exist and sends the result to the **server**.
4. **Server**:
   * Processes the result and sends a response to the client (e.g., success or error message).

**In summary:**

* The **server** acts as the brain of the system, handling communication, processing, and business logic.
* The **database** serves as the memory, securely storing and retrieving data when needed.

**Server on the Cloud**

* **Cloud Servers**: These are virtual servers hosted on cloud platforms (like AWS, Google Cloud, Azure). You can run applications, APIs, and any server-side logic on cloud servers.
* **Examples**:
  + AWS EC2 (Elastic Compute Cloud)
  + Google Cloud Compute Engine
  + Azure Virtual Machines

**Database on the Cloud**

* **Cloud Databases**: These are managed database services hosted on cloud platforms. You don't have to worry about infrastructure or maintenance (e.g., updates, backups) since the cloud provider manages it for you.
* **Examples**:
  + AWS RDS (Relational Database Service) for SQL databases (MySQL, PostgreSQL, etc.)
  + Google Cloud SQL
  + Azure SQL Database
  + MongoDB Atlas (Cloud-hosted NoSQL database)
  + Firebase (Real-time NoSQL database by Google)

On-prem (i.e on premise)

physical servers and maintains its own database hardware.