Handling DB Connection: Naive and Professional

Two important points

- DB se baat karne par problems aa sakti hai to **try-catch** me wrap karo humesha. Ya phir **promises** bhi le sakte hai.
- DB is always in another continent - means time lagega - **async-await**

Naive way

`.env` PORT = 8000 MONGODB_URI = mongodb+srv://:@practice.u5g2cwm.mongodb.net

`constants.js` export const DB_NAME = "youtube-twitter";

`index.js`

```
import express from 'express'
const app = express()

import dotenv from 'dotenv'
dotenv.config()


;(async ()=>{
    try{
        await mongoose.connect(`${process.env.MONGODB_URI}/${DB_NAME}`)
        app.on("error",(error)=>{

            console.log("ERR: ",error);
            throw error;
        })
        app.listen(process.env.PORT,()=>{
            console.log(`App is listening on port ${process.env.PORT}`)
        })
    }catch(error){
        console.error("ERROR: ",error);
        throw error;
    }
})()
```

Professional

- `.env` : isme bas environment variables daalo
- `constants.js` : isme constants if needed
- `index.js` : isme dotenv ko import or config karenge. `app.js` ko import karenge jisme express server ka code hoga, aur `db/connection.js` ko import karenge, aur jo function jisme connection ka

code hoga use call kar denge.

- `db/connection.js` : isme db connection ka code hoga aur jo function hoga use export kar lenge.
- `app.js` :express ka code yaha hoga.

`.env`

PORT = 8000 MONGODB_URI = mongodb+srv://:@practice.u5g2cwm.mongodb.net

`constants.js`

export const DB_NAME = "youtube-twitter";

`index.js`

> dotenv ko import aur config karo, and connection.js me se connection wale function ko import kar ke call kar do.

```
import dotenv from "dotenv";
dotenv.config();
import connectDB from "./db/connection.js";
import app from "./app.js";

connectDB()
  .then(() => {
    app.on("error", (error) => {
      console.log("ERR: ", error);
      throw error;
    });
    app.listen(process.env.PORT || 8000, () => {
      console.log(`Server is running at ${process.env.PORT}`);
    });
  })
  .catch((err) => {
    console.log("Mongo DB connection FAILED!!!!", err);
  });
```

`connection.js`

```
import mongoose from "mongoose";
import { DB_NAME } from "../constants.js";

const connectDB = async () => {
  try {
    const connectionInstance = await mongoose.connect(
      `${process.env.MONGODB_URI}/${DB_NAME}`
    );
    console.log(
      `\n MongoDb Connected !! DB HOST: ${connectionInstance.connection.host}`
    );
  } catch (error) {
    console.log("MONGODB connection FAILED", error);
    process.exit(1);
  }
};
```

```
export default connectDB;
```

`app.js`

```
import express from "express";
import cors from "cors";
import cookieParser from "cookie-parser";

const app = express();

app.use(
  cors({
    origin: process.env.CORS_ORIGIN,
    credentials: true,
  })
);

app.use(express.json({ limit: "16kb" }));
app.use(express.urlencoded({ extended: true, limit: "16kb" }));
app.use(express.static("public"));
app.use(cookieParser());

export default app;
```

- Explaination of `app.js`

  ```
  import express from "express";
  ```

  - This line imports the Express.js framework

  ```
  import cors from "cors";
  ```

  - CORS (Cross-Origin Resource Sharing) is a mechanism that allows or restricts the resources on a web page to be requested from another domain outside the domain from which the first resource was served.

  ```
  import cookieParser from "cookie-parser";
  ```

  - This middleware parses cookies attached to the client's request and makes them available in the request.cookies object.

  ```
  const app = express();
  ```

  - Creates an instance of express() application

```
app.use(
cors({
  origin: process.env.CORS_ORIGIN,
  credentials: true,
})
);
```

- `app.use` : This method is used to mount middleware functions in the Express application pipeline.
- `cors` : This middleware is used to enable Cross-Origin Resource Sharing. It allows or restricts cross-origin HTTP requests based on the configuration provided.
- `origin: process.env.CORS_ORIGIN` : This sets the allowed origin for CORS requests based on the value specified in the `CORS_ORIGIN` environment variable. It's common to set this to the domain or origins that are allowed to make requests to your server.
- `credentials: true` : This indicates that the server should include credentials (such as cookies or HTTP authentication) in the CORS request.

```
app.use(express.json({ limit: "16kb" }));
app.use(express.urlencoded({ extended: true, limit: "16kb" }));
```

- `express.json` : This middleware parses incoming JSON payloads and makes the parsed data available in `request.body` .
  - The `limit` option is set to control the maximum size of the JSON payload (16kb in this case).
- `express.urlencoded` : This middleware parses incoming URL-encoded payloads (usually from HTML forms) and makes the parsed data available in `request.body` .
  - The `extended: true` option allows parsing of nested objects.
  - The `limit` option is set to control the maximum size of the URL-encoded payload.

```
```
app.use(express.static("public"));
```
  - **`express.static`**: This middleware is used to serve static files, such as images, CSS, and J
    - In this case, it serves files from the "public" directory.

```
app.use(cookieParser());
```
  - This middleware parses cookies attached to the client's request and makes them available in the
    - It's used to handle cookies sent by the client.
```

Important

In our code, the database connection is established using the mongoose.connect method, and the connection string is constructed by concatenating process.env.MONGODB_URI and DB_NAME. If

DB_NAME is not present in your MongoDB database, MongoDB will **not create** a new database with that name unless you perform some write operations.

MongoDB will connect successfully even if the specified database (youtube-twitter in this case) does not exist at the time of connection. MongoDB will create the database when we perform our first write operation.