Page No.
Date

=> For Prettier file configuration make file with [.Prettierrc] name and define all requirment configuration.

esc: in file:

```
{
  "singleQuote": false,
  "bracketSpacing": true,
  "tabWidth": 2,
  "trailingComma": "es5"
  "semi": true
}
```

=> there is also one it's file which name is [.Prettierignore] which is used for which file we not apply Prettier format.

esc: in file:

```
*.env
.env
.env.*
```

=> There are also, boiler generator for it all file, or but it's all are depend on our requirement.

6] How to connect database in MERN with debug.

(1) make mongoDb atlas account for cloud. to cloud.
caution:
- always at there is only one machine IP is provided where our backend served.

'allow from anywhere' is not good and caution. but a we can use debugging purpose.

→(2) after set environment variable in .env

PORT = 8000
MONGODB_URI = mongodb+srv://...... .net

→ after we give name to our app database in constant.js as below.

export const DB_NAME = "videotube"

(3) There is two approach to connect mongoDB.

① Professional and modular approach where all Sequlized connection func. code are in DB → file. and called by the index.JS file.

② directly all connection code are write in the index.JS file.

→ | dotenV Package : it's require to load env variable in our app. it use require syntax.

Tips: Database about some points...
- whenever we connecting database problems are necessary.
- make sure we use try catch or promise
- Database is always in other continent
- make sure async await used.

① → So, more use use first approach to write code in index.JS file. with IIFE func.

```
;(async () => {
    try {
        await mongoose.connect(`${process.env.MONGO_URI}/${DB_NAME}`)
```

express event listner error handling

```
        app.on('error', (error) => {
            console.log("Error", error);
            throw error
        })
```

to run on port

```
        app.listen(process.env.PORT, () => {
            console.log(`App listening on ${process.env.PORT}`)
        })
    } catch (error) {
        console.error("Error", error)
        throw err
    }
})()
```

(2) Second db comection approach.
to write all code on modular & in DB folder file.

⇒ In db ⇒ Index.js file.

```
const connectDB = async () => {
    try {
        const connInstance = await
mongoose.connect(`${process.env.URI}/${DB_NAME}`)
        console.log(`\n MongoDB connected !!
DB HOST: ${connectionInstance.connection.host}`);
    } catch (error) {
        console.log("MongoDB connn Failed", error
        process.exit(1);
    }
export default connectDB
```

mongoDB return obj for success connection we can check or see the other it host

process is from node which means to terminate process

↳ 0 means without failure 1 means with failure.

⇒ In main Index.JS file.

```
require('dotenv').config()
import ....
import ....
ConnectDB();
```

} It break code consistency so we use env statement with flag.

import ...

import dotenv from 'dotenv'

dotenv.config ({
    path : './env'
})

connectDB()

⇒ in package.json

nodemon -e dotenv/config --exec experimental-json-modules src/index.js"

—> This all things are only for dotenv package other to all the app when it used and for import statement instant require

⇒ dotenv package used for the provide other to all in app.

⇒ when ever error occure read properly because in backend even .js extensions error are problem. it we not given.

7] Custom API response & error Handling

⇒ Now, we disperse into express.js file.

after, connection DB file it return promise. so we write app listen on his then logic. like as.

⇒ index.js

```
connect.DB().
.then( () => {
   app.listen (env.PORT.., () => {
      console... ( "app listen....) }
.catch ( (e) => {
   c.lg (" Failed.... )
})
```

⇒ In app.JS file we write all express related code.

```
import express from 'express'

const app = express();

   }   }   }   }   ← Here all middleware
                    and other package config
                    are write which
                    we discuss ahed.

export { app }
```

→ In between the app.js

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

↳ It is also using for resource sharing for authentication & cookie testing.

↳ app.use(express.json({limit: "16kb"}))

For accepting the json data from users.

↳ app.use(express.static("public"))

↳ It is for static assets accessing.

↳ app.use(cookieParser())

↳ We install this cookie-parser pkg for the sending and storing cookie and performing CRUD operation. to accessing set cookies.

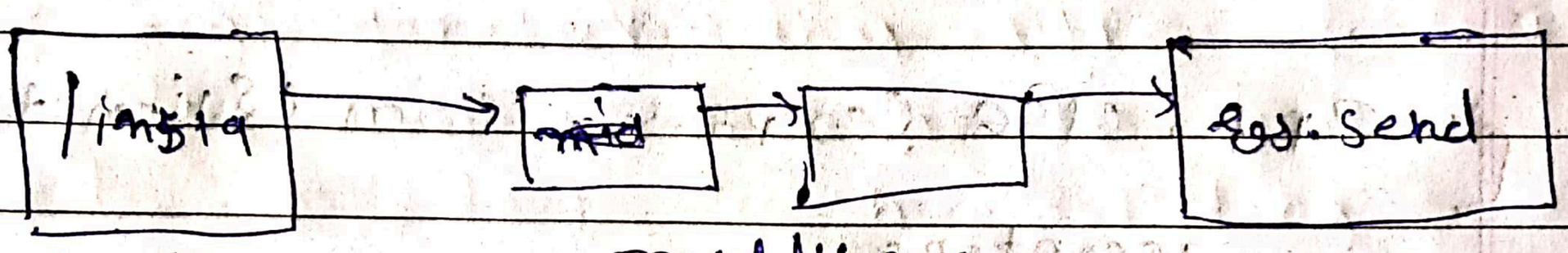
app.use(express.urlencoded({extended: true, limit: "16kb"}))

↳ for handelling the url data from the user.

⇒ Middleware overview :-

→ middleware work is the process between req & res. to performing some task.

→ exc: user hit the /instagram url but we wants to check is logged user so we made middleware to check before sending response.



⇒ There is next argument for passing to the next middlewares.

→ There is always 3 argumen not only (req, res. but (error, req, res, next

→ middlewares are always write in sequences.

⇒ Utils Folder use & implementation :-

→ In utils all utility fils and reusable code are written.

⇒ For try catch or promises we use in all connection & request so we use it handler.

→ we make file in utels → asyncHandler.js

① with try catch.

```
const asyncHandler = (fn) => async (req, res,
  next) => {
    try {
        await fn(req, res, next)
    } catch (error) {
        res.status(err.code || 500).json({
          success: false,
          message : err.message
        })
    }
}
```

② with Promises.

```
const asyncHandler = (requestHandler) => {
    return (req, res, next) => {
        Promise.resolve(requestHandler(req,
        res, next))
        .catch((error) => next(error))
    }
}
export {asyncHandler}
```

=> In general this type of writing code better approach to reduce the code of try catch repeting.

→ for attaching try catch we made wrapper.

⇒ Same way for catch or error part there is also one better approach to customize the error code.

→ Node provide error class for Handeling error.

→ b with help of it we make our custom error API, how like as.....

→ In utils → ApiError.js file.

```
class ApiError extends Error{
    constructor(
        statusCode,
        message = "something went wrong",
        errors = [],
        stack = ""
    )
    {
        super(message)
        this.statusCode = statusCode
        this.data = null;
        this.message = message;
        this.success = false
        this.errors = errors

        if (stack) {
            this.stack = stack
        }else{
            Error.captureStackTrace(this,
            this.constructor)
        }
    }
}

export {ApiError}
```

Page No.
Date

⇒ Also same task as we make file for success response.

→ for sending the response with the proper manner & esponse we write this code as error we made file.

→ In, utils → ApiResponse.js file.

```
class ApiResponse{

constructor(statuCode, data, message="Sucess"){
this.statusCode = statusCode
this.data = data
this.message = message
this.success = statusCode < 400
}
}

export {ApiResponse}
```

Page No.
Date

8] User & Video model with hooks & JWT

⇒ in the user & Video model we need first it's model file and define all regular field, name, email, Pass etc. but the main things are written below of that model and myths.

⇒ index: true, it's property in define in our model schema field for searching optimizing way. So we define index true for that field.
→ remember give more index is affect the app.

⇒ avatar or Thumbnail or any image or Video we use third party API to Storing this and they provide us link and some data for accessing it. ex: cloudinary, AWS,

⇒ So in model we just define it's type as a string but it's link. of cloudinary or third party api.

Page No.
Date

⇒ mongoose-aggregate-Pagimate-V2 :-

- it's allows us to make the aggregate query efficiently and give more power to write advance aggregates queries.

→ It's inject like Plugin in our Project. (in Schemas)

→ OC : with explanation.

import mongooseAggregatePagimate from "mongoose-aggregate-....";

const VideoSchema = new Schema({
    our property with field.
});

VideoSchema.plugin(mongooseAggregatePagimate)

Schema variable we give to add plugin and other PRE, POST method to perform task.