**Node js :-**

Node js is a javascript environment , in which we use javascript to write the code.

**Software Dependencies :-**

It is an external stand alone library that can be organized into packages to perform specific task.

**NPM :-** Node Package Manager

**Express js :-**

It is Prebuild Node js framework that will help us creating server side web applications faster and even smarter , and since it’s a package so it will reduce our work load.

**Mongoose :-**

Mongoose is a MongoDB object modelling tool designed to work in an asynchronous environment .

Mongoose supports both promises (like Future in dart) and callbacks ( promises and callbacks are used to overcome the delay in asynchronous calls ) .

**nodemon :-**

npm I nodemon –save-dev ( to install nodemon and save it as developer dependency )

**What is API ?**

API is software that helps application to interact with each other.

Messenger between Frontend and Backend.

**JSON** :- Javascript Object Notation

**Client Server Architecture :-**

In this the Client(frontend) generates a request to server(backend)

And server gives response to the client , this is Client server architecture.

**HTTP Protocol :-**

For the request and response we use HTTP protocol.

HTTP Protocol defines some rules and regulations to be followed while request and response for the communication .

**Commands :-**

node –version

git –version

cls => ( to clear the terminal )

npm init -y => ( to create package.json file )

npm install express jsonwebtoken nodemon bcrypt dotenv mongoose cors => ( to install the required libraries )

node src/index.js ( To run index.js file stored in src folder )

npm start ( To run the “start” script mentioned in package.json file )

npm run start ( To run the “start” script mentioned in package.json file )

**node\_modules :-**

This folder contains the code of all the libraries that we have installed

**Importing libraries in javascript :-**

const express = require("express");

**Further process :-**

const app = express();// app is an object of express.

app.listen(5000, () => {

    console.log("Server started on port no. 5000");

})

app.get("/", (req,res) => {

})

This means that when the get request will come on the root url ( “/” ) then the given function will be called ….

// Function : -

(req,res) => {

})

req is the object of Request.

res is the object of Response.

app.get("/", (req,res) => {

    res.send("Hello world");

})

send( ) function is used to send the response to the request.

app.get("/quotes\_with\_status", (req,res) => {

    res.status(200).json(quotes);

})

Here the response is given a status which tells the client or user about the request, that the request is successful or unauthorized , etc.

**200 => Request Successful.**

**404 => Not Found.**

**400 => Bad Request.**

**201 => Record is created successfully.**

**500 => Some Error occurred**

**401 => Unauthorized User.**

let index = Math.floor(Math.random() \* quotes.length);

To generate random number in the given range(quotes.length) the result will be in decimal so using floor function we can get the int value.

**package.json**

"start": "nodemon src/index.js"

Here nodemon library is used instead of node so that we do not have to restart the server after the changes we made, it will automatically restart the server.

**Routes :-**

This is used to separate or segregate the endpoints of the url, like the user url will be in another file and the notes url will be in another file and admin related url in another file.

**userRoutes.js**

const express = require("express");

const { signup, signin } = require("../controller/userController");

const userRouter = express.Router();// userRouter is the object of Router.

userRouter.post("/signup",signup);

userRouter.post("/signin",signin);

module.exports = userRouter;// we export it so that we can use it in another files

**In index.js file**

app.use("/users",userRouter);

to access the users we have used the userRouter for “/users” end point and now we can access it like :-

/users/signup

/users/signin

**noteRoutes.js**

const express = require("express");

const { getNotes, createNote, deleteNote, updateNote } = require("../controller/noteController");

const noteRouter = express.Router();

const auth = require("../middlewares/auth");

noteRouter.get("/",auth,getNotes);

noteRouter.post("/",auth,createNote);

noteRouter.delete("/:id",auth,deleteNote);

noteRouter.put("/:id",auth,updateNote);

module.exports = noteRouter;

Here , id is used in the url to delete and update(put) the note of a particular user .

auth function will be called before the getNotes , createNote , deleteNote , updateNote functions as next() & these functions will get the user id after the verification of token in req.userId(which can be used in those functions).

.

**Controllers :-**

In this folder , we define the functions that belong to the particular endpoints, which are executed when we navigate to a particular url endpoint in the routes.

**userController.js**

const userModel = require("../models/user");

const bcrypt = require("bcrypt");

const jwt = require("jsonwebtoken");

const SECRET\_KEY = process.env.SECRET\_KEY;

const signup = async (req,res) => {

    // Existing User Check

    // Hashed Password

    // user creation

    // token generate

    const {username,email,password} = req.body;

    try {

        // Existing User Check

        const existinguser = await userModel.findOne({email:email});

        /\*

        as findOne() function is going to connect with the database, so it

        will take some time to return the response or output or result, so

        that's why we have used await keyword so that program can wait till

        we get the output from the database.

        \*/

        // To use await keyword , the parent function should be async.

        if(existinguser)

        {

            return res.status(400).json({message:"user already exists"});

        }

        // Hashed Password

        const hashedPassword = await bcrypt.hash(password,10);

        //here password is the string to be hashed and 10 is the no. of rounds to be executed to hash the password.

        // await is used because bcrypt is asynchronous function.

        // user creation

        const result = await userModel.create({

            email:email,

            password:hashedPassword,

            username:username,

        });

        // token generate

        const token = jwt.sign({email:result.email, id:result.\_id},SECRET\_KEY);

        // In result.\_id , \_id is the unique id that is generated after creating the user in Database.

        // here sign() function is a function of jwt , that will generate the token for the given payload or values.

        /\*

        => Tokens are used to validate the user.

        => While signup , the token will be generated according to the information provided.

        => Then , the user can access the notes using the token by decrypting the token for userId.

        \*/

        res.status(201).json({user:result,token:token});//200 => record is created successfully.

    } catch (error) {

        console.log(error);

        res.status(500).json({message:"Something went wrong"});//500 => Some Error occured.

    }

}

const signin = async (req,res) => {

    const {email,password} = req.body;

    try {

        const existinguser = await userModel.findOne({email:email});

        if(!existinguser)

        {

            return res.status(404).json({message:"user not found"});

        }

        const matchPassword = await bcrypt.compare(password,existinguser.password);

        if(!matchPassword)

        {

            return res.status(400).json({message:"Invalid credentials"});

        }

        const token = jwt.sign({email:existinguser.email, id:existinguser.\_id},SECRET\_KEY);

        res.status(200).json({user:existinguser,token:token});

    } catch (error) {

        console.log(error);

        res.status(500).json({message:"Something went wrong"});

    }

}

module.exports = {signin,signup};

**Index.js**

app.use(express.json());// it is used to pass the request body into json form., so that we can access the response directly.

**noteController.js**

const noteModel = require("../models/note");

const createNote = async (req,res) =>

{

    console.log(req.userId);

    const {title,description} = req.body;

    const newNote = new noteModel({

        title:title,

        description:description,

        userId:req.userId,

    });

    try {

        await newNote.save();

        res.status(201).json(newNote);

    } catch (error) {

        console.log(error);

        res.status(500).json({message:"Something went wront"});

    }

}

const updateNote = async (req,res) =>

{

    const id = req.params.id;

    const {title,description} = req.body;

    const newNote= {

        title : title,

        description: description,

        userId:req.userId,

    }

    try {

        await noteModel.findByIdAndUpdate(id,newNote, {new:true});

        res.status(200).json(newNote);

    } catch (error) {

        console.log(error);

        res.status(500).json({message:"Something went wront"});

    }

}

const deleteNote = async (req,res) =>

{

    const id = req.params.id;

    try {

        const note = await noteModel.findByIdAndDelete(id);

        res.status(202).json(note);

    } catch (error) {

        console.log(error);

        res.status(500).json({message:"Something went wront"});

    }

}

const getNotes = async (req,res) =>

{

    try {

        const notes = await noteModel.find({userId:req.userId});

        res.status(200).json({notes});

    } catch (error) {

        console.log(error);

        res.status(500).json({message:"Something went wront"});

    }

}

module.exports = {

    createNote,

    updateNote,

    deleteNote,

    getNotes,

}

**SQL :-**

In SQL , there is table in the relation among the different columns and rows in the table or among the tables.

**NOSQL :-**

In NOSQL , there is Document (Row) and Collection (Table).

So the Collection is the collection of Documents

**mongoose :-**

mongoose is a package which is used to connect to mongodb.

const mongoose = require("mongoose");

mongoose.connect(process.env.MONGO\_URL\_NOTES)//("mongodb+srv://saininaveen933:naveen1234@cluster0.0gsvxzs.mongodb.net/?retryWrites=true&w=majority")

.then(()=>{

    app.listen(PORT , ()=>{

        console.log("Server started at port no. "+PORT);

    });

})

.catch((error)=>{

    console.log(error);

})

Here then( ) is a callback function which is executed when the connection is successful in this case.

And , if any error then catch( ) function is executed.

**Schema :-**

It defines the properties to be saved in the object.

Schema is defined in models folder.

**user.js**

const mongoose = require("mongoose");

const UserSchema = mongoose.Schema({

    username:{

        type:String,

        required:true,

    },

    password:{

        type:String,

        required:true,

    },

    email:{

        type:String,

        required:true,

    }

},{timestamps : true} // it will add 2 properties in the schema :- created at and modified at.

);

module.exports = mongoose.model("User",UserSchema);

**note.js**

const mongoose = require("mongoose");

const NoteSchema = mongoose.Schema({

    title:{

        type:String,

        required:true,

    },

    description:{

        type:String,

        required:true,

    },

    userId:{

        type:mongoose.Schema.Types.ObjectId,// we have to refer the object id of the user into the note collection.

        ref:"User",//ref property defines that the object id is of User type or User.

        required:true,

    }

},{timestamps : true}

);

module.exports = mongoose.model("Note",NoteSchema);

**Middlewares :-**

Middlewares are used to implement authorization or validation.

Middlewares are the security guard that authorizes the user before initiating the request.

Middlewares also represents the processes or steps between the pipeline of request and response.

Eg :- app.use(express.json());

const jwt = require("jsonwebtoken");

const SECRET\_KEY = process.env.SECRET\_KEY;

const auth = (req,res,next) => {

    try {

        let token = req.headers.authorization;

        if(token)

        {

            token = token.split(" ")[1];

            let user = jwt.verify(token , SECRET\_KEY);// after verifying the token , we will get all the information of the user.

            req.userId = user.id;

        }

        else

        {

            return res.status(401).json({message:"Unauthorized user"});

        }

        next();// if verification successful.

    } catch (error) {

        res.status(401).json({message:"Unauthorized user"});// 401 => Unauthorized User.s

    }

}

module.exports = auth;