```
// any module that we require need to be imported and give to local va
riable
const fs= require('fs')

// used to write into file
// 1st argumet file name and second argument is content that we want t
o write
fs.writeFileSync("sample.txt",'first line')

//  append to the exesting file
fs.appendFileSync("sample.txt",'\n third line')
```

test.js

```
const send_data= function()
{
    return "data sent"
}

name ="shivam parve"

// this will make the local data global to access to other files
module.exports =send_data
```

first.js

```
// we are importing the file
const test= require('./test.js')

// calling the function
console.log(test())
```

1. we can use some predefined packages form npm

npm init
npm install validator@10.8.0
validator is just one example

```
 const validator= require('validator')

console.log(validator.isEmail("shivam.p.parve@gmail@com"))

a= "patil"
b="patil"
console.log(validator.equals(a,b))
```

npm install nodemon@1.18.5 -g

here -g indicate that the package is installed globally and present in os.

Up till now we are running the script each and every time but nodemon app.js will countinoly run in background and make changes as change in code

Now we will e commond to run

nodemon app.js

```
const fs= require('fs')
const data={
    name:"shivam",
    age:20
}

// to send data to file convert into string

const data_string= JSON.stringify(data)

// write data to file
fs.writeFileSync('data.json',data_string)

// data accepted from file is not string
```

```
const buffer_data =fs.readFileSync('data.json')

// convert into string
const receive_data= buffer_data.toString()

// convert into json

const data_parse= JSON.parse(receive_data)
console.log(receive_data)

//JSON -> string ->file_store -> buffer_recieve->string ->json
```

2. Passing multiple data using module.export

```
 module.exports= {
      "addNotes" :addNotes,
      "loadNotes":loadNotes
  }
```

Where addNotes ND loadNotes are the functions

```
// we are importing the file
const test= require('./test.js')

// calling the function
Test.addNotes(title,body)
```

3. **ES6 function syntax**

Const square =function (x) {

Return x*x

}

The above function can be modified as

By removing the function name and adding the => operator

const square =() => {

return x*x

}

When the function have only one line to execute then it can be again modified

```
const duplicate= note.filter(function (notes) {
        return notes.title != title_received  // this is the
condiditon to take    particular element in an new array
     })
```

In the above function we are creating the duplicate array of given array which will have elements base on given condition

This can be modified as follows

```
const duplicate= note.filter((notes) => return notes.title != title_re
ceived )
```

4. **Accessing the array elements**

```
[{"title":"attend the college ","body":"i am lack of attendance "}]

// this is the array of object
notes= loadNotes()  // this function will return the array of objects

notes.forEach(function(data){

console.log(data.title)  // data will represent each object in that
array
}
)
```

5. **To debugg the code**

This will print the message in red or another color so that it is easy to identifiy

```
const chalk = require('chalk');

console.log(chalk.red('in remove notes'))
```

Asynchronous

The one line will execute one after another not caring how much time each line will require to execute

6. **HTTP request**

```
var request = require('request');

request('http://www.google.com', function (error, response, body
) {

  console.log('error:', error); // Print the error if one occurr
ed

  console.log('statusCode:', response && response.statusCode); /
/ Print the response status code if a response was received

  console.log('body:', body); // Print the HTML for the Google h
omepage.

});
```

All the information send from the server is in response.body further we can access each object and each field from body

Eg.

```
{
    "name":"shivam",
    "age":20,
    "marks":[
        {"year ":"10th",
        "percentage":95.4
        },
        {"year ":"12th",
            "percentage":88.33
            },
            {"year ":"engg",
                "percentage":9.99
                }
    ]
}
```

7. We can access marks array as

```
console.log(response.body.marks[0])
```

in this way we can access each and evey element of body by . operator

Asynchronous function: A function which takes more time to execute then it will send to backend and next statement will be executed . if any function has some aynchrous activity then it might happen that it will not send the proper return value.

So to avoid this we use callback we send the data to the callback function and accept it in the calling function so that when ever we get output we send to callback function

```
const add=(a,b, callback)=>{
setTimeout(() => {
  //  return (a+b)
    callback(a+b)
}, 2000);  // asynchronous function takes more time

}
console.log('before add')
add(3,5,(sum)=>   // 3ed paramenter is callback function
{
console.log(sum)
})
console.log('after addd')
```

this can be applied when we have to receive data from server which takes time

so we send data to callback function

Express

Get request using express

```
const express =require('express')
// this of ojbect of express using which we canaccess the all the methods
const app= express()

// 1st argument is url
// 2nd arg is function which will work when response from resrver came
// it has two parameters req,res
// req data send by user using request
// res the response e want to send
app.get('/help',(req,res)=>
{
    res.send('this is help page')
})

// this will activate the port 3000
app.listen(3000,()=>
{
    console.log('listning on port 3000')
})
```

Git

git init : this will create the github repository in our project for version

git status : gives status of our project and all untracked files

git add src/ : this will all all files in src folder means they are ready to commit

git add . : all the files in the main folder means they are ready to commit

git commit -m " this is the message with each commit that explain what we have done" :commit the **added** folder

git checkout (1st 5 letters of id of commit) : move to that commit

git checkout master : move to master commit

Mongodb

```
var MongoClient = require('mongodb').MongoClient;

const dbase='shivam'
const url='mongodb://127.0.0.1:27017'

MongoClient.connect(url ,{ useUnifiedTopology: true },{useNewUrkParser:true},(error,client)=>
{
    if(error)
     return console.log('unable to connect to database')

     console.log('connected to db')
   const  db =client.db(dbase)

        db.collection('nodejs').insertOne({"name":"Ankitapatil","grno":"17u113"},
(error,result)=>{

            if(error)
            return console.log('data not inserted')

            // ops will show inserted document
            console.log(result.ops)
        })

})
```

We use mongoose for database connectivity .

Mongoose.model is used to create objects and provide validations to them.

Mongoose

Async function: it always return the promiss and in that promiss it contains the value coder wants to return

step 0:

we have created conncetion with database using mongoose and stoed in seperate folder called db.

we will export this and use when ever necessary.

step 1:

first we have to create the model using express to sotre the data into data base.

we have seperate folder model in src which contains the model database schema for each collection

we use that model by exporing to create object and store into database

step 2:

when any get/post/delete/patch request for particular model come then we have created seperate

roter for each model and stored in src/ model folder.

we have created route object and using which we will mange all request related to that model.

we have expoted that route from each file

step 3:

index.js is our main server.

to avoid laod on single file we have devided the models into each file and then finally we use each model

in index file.

so that the routing takes place.

STEP 0

```
const mongoose =require('mongoose')
// this will connect to database with name nodejs
mongoose.connect('mongodb://127.0.0.1:27017/nodejs',{
    useNewUrlParser:true,
    useCreateIndex:true,
    useUnifiedTopology: true
})
```

STEP 1

```
const mongoose =require('mongoose')
const validator =require('validator')

// this is the model for user
const User =mongoose.model('User',
{
    name: {
        type:String,
        required:true,
        trim:true
    },
    email:{
        type:String,
        required:true,
        validate(value)
        {
                if(!validator.isEmail(value))
                {
                    throw new Error('Email is invalid')
                }
        }

    },
    age :{
        type:Number,
        default:0,
    }
})

module.exports =User
```

STEP2

```
const express =require('express')
const User =require('../models/user.js')
const router = new express.Router()


router.post('/user',(req,res)=>
{
console.log(req.body)

// object for user created now it will be stored into database
var user =new User(req.body)

        user.save().then(()=>
        {
        res.send('user registered successfully')
        }).catch((e)=>
        {
        res.status(400).send(e)
        })

})

module.exports= router
```

STEP3

```
// this is main file of server
// we have distributed overhead in model files
// and finall use that router in this file

const express =require('express')
require('./db/Mongoose.js')
const app=express()
// the data coming in the request is parsed as json
app.use(express.json())



// we are importing the router from each file
const user_router =require('./routes/user_routes.js')
const task_router=require('./routes/task_router.js')

// by this comnd it will use that router
app.use(user_router)
app.use(task_router)



app.listen(3000,()=>
{
    console.log('listning no port 3000')
})
```

Mongodb can be act as a middle ware which is used to perform operations.

To store the password we have to store in the encrypted format. User may give password password at the time of registration or updating.

Mongoose provide the facility to act as middleware.

We normally create model but to use mongoose as middle ware we have to use schema.

npm i bcryptjs

this is used to hash the password and we match the given password with the hashed password before saving to database.

We have two methods pre or post means before saving or validation we can run the function.

The update function by pass the middleware of mongoose so that we have to split the function.

Initially we were using the findByIdAndUpdate but now we will find separately and update separately.

When user login we can generate the token related to that user

This is done by `'jsonwebtoken'`

```
const mongoose =require('mongoose')
const validator =require('validator')
const bcrypt= require('bcryptjs')
const jwt =require('jsonwebtoken')

const userSchema = new mongoose.Schema({
    name: {
        type:String,
        required:true,
        trim:true
    },
    email:{
        type:String,
        required:true,
        validate(value)
        {
                if(!validator.isEmail(value))
                {
                    throw new Error('Email is invalid')
                }
        }

    },
    age :{
        type:Number,
        default:0,
    },
    password:{
        type:String,
        required:true,
    },
    Tokens:[{
        token:{
            type:String,
            required:true
        }
    }]
})

// this are the methods we are applied on the object
userSchema.methods.tokenGenrator =async function()
{
const user = this
const token = jwt.sign({_id:user._id.toString()},'AnkitaPatil')
user.Tokens= user.Tokens.concat({token})

await user.save()
return token
}



// statics are applied on model
userSchema.statics.findCredetials =async (email,password)=>
{
    console.log("email from staits",email)
    const user = await User.findOne({ email })

    if(!user)
     throw new Error('unable to login')


     const isMatch = await bcrypt.compare(password,user.password)

     if(!isMatch)
     throw new Error('unable to login')

     return user

}

// this is the function that call before evry object saved to data base wich used
 User as class
// to create object
// do not use => syntax in this function
userSchema.pre('save',async function (next)
{
const user=this

console.log('in userSchema',user)

if(user.isModified("password"))
{
    user.password= await bcrypt.hash(user.password,8)
    console.log('in if')
}

next()
})

// this is the model for user
const User =mongoose.model('User',userSchema)
```

```
// instead of using the call back method we have used asyn function which provide
the promiss with the responce


router.post('/user/login',async(req,res)=>{

try {
        const user =await User.findCredetials(req.body.email,req.body.password)
        const token =await user.tokenGenrator()
        console.log('user from try ',token)
        res.status(200).send({"message":"login successfully","toekn":token})

} catch (error) {
        console.log('error',error)
        res.status(200).send({"error":"uable to login"})

}
})


module.exports =User
```

Middle ware

index.js

```
//uptill now we have used app.use() for registering default function or router
// but we can use this to set middle ware

// request ----> middleware -----> router

app.use((req,res)=>
{
if(req.method=='GET')
      res.send('GET requests are blocked')

})
```