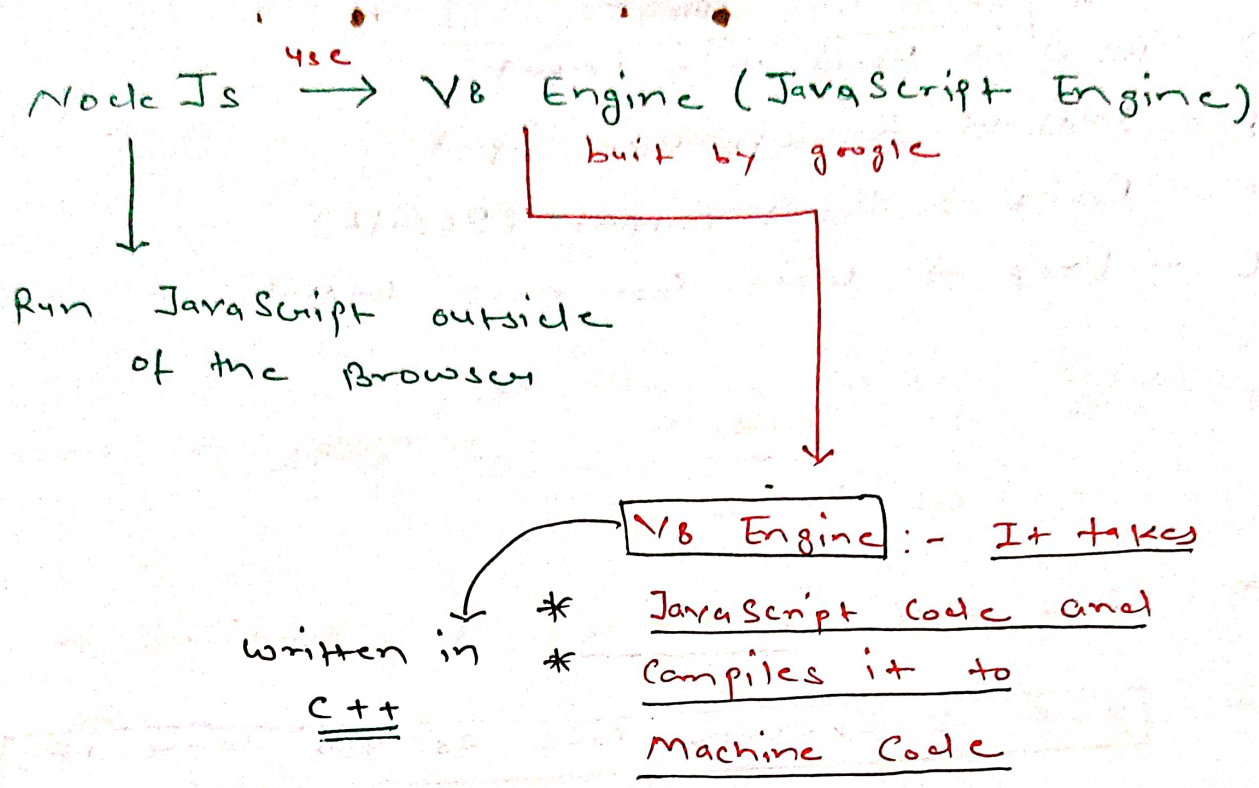
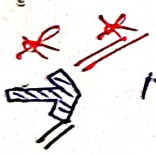
Node.js Notes

Bhupendra Singh

→ What is nodeJs ?





Single threaded

NodeJs is an open-source, Cross-platform, backend JavaScript runtime environment. It is run on the V8 Engine (that written in C++), which is used to Convert JavaScript code to Machine code (during Compilation).

* NodeJs main advantage that use JavaScript which uses so much in modern web development.

⇒ REPL → Different way to execute node code

↳ It's like playGround

R - Read → Read user input

E - Eval → Evaluate user input

P - Print → Print output (Result)

L - Loop → Wait for new Input.

* Just type "node" in terminal to start/run the code in terminal.

* Just type "ctrl + C" to terminate the program

* REPL → Execute code as you write it. *

* NPM

→ node package manager

** → It is not node core modules but we can use npm to initialize to node project or to add some extra feacture.

→ npm init (package.Json file will be created)

```
"Script" : {

    "start" : "node app.js",
    "start-dev" : "node app.js"
}
```

→ we can define our own script.

npm start

error

npm start-dev

npm run start-dev

Node.Js Basics

- HTTP → A protocol for transferring data which is understood by Browser and Server.
- HTTPS → Hyper Text Transfer Protocol Secure
  - HTTP + Data Encryption

Create Node.js Server

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

require → keyword (global)

```
http.createServer();
```

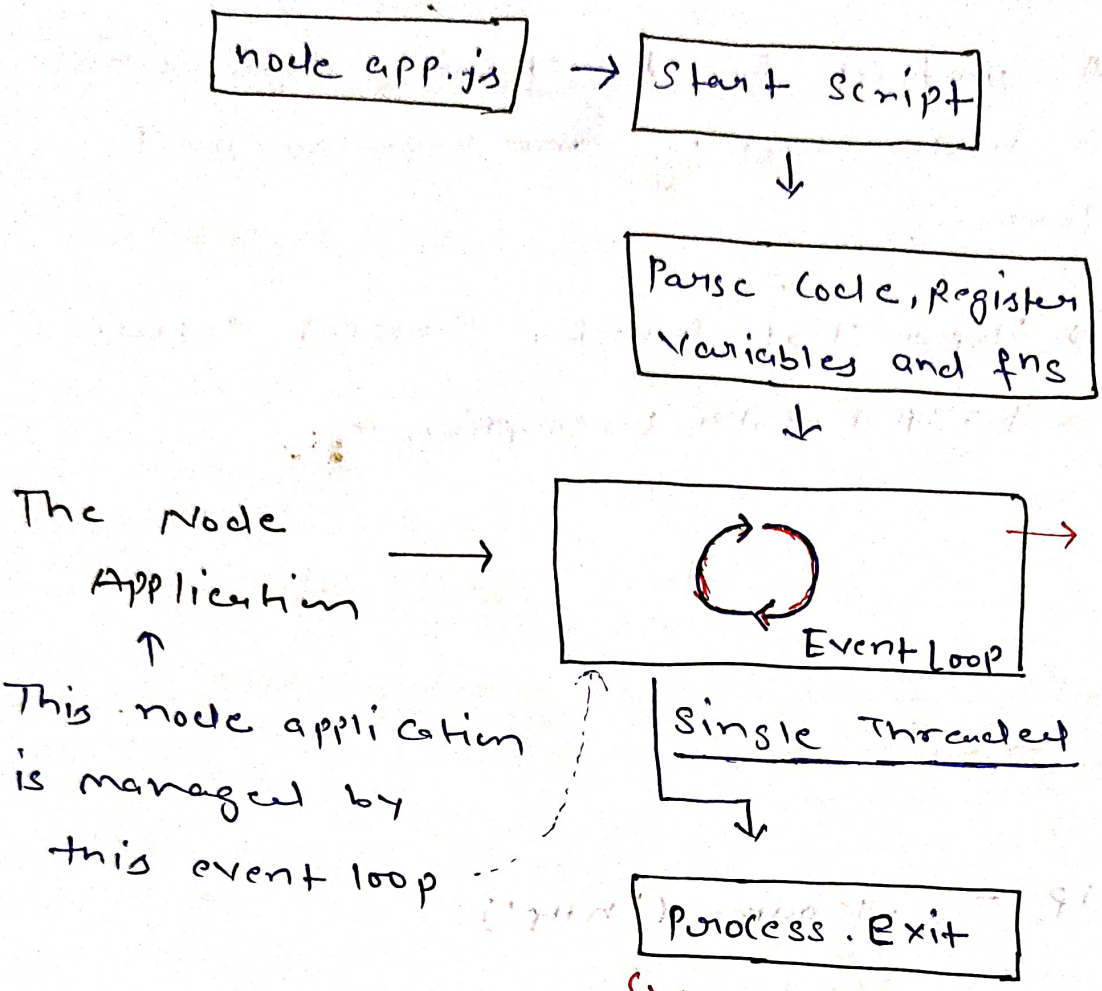
↳ takes requestListener as an argument

⇒ createServer() never finishes by default.

```
const server = http.createServer((req, res) => {
    console.log("--req--");
    // process.exit()
});

server.listen(port,
```

* Node.js program cycle



* Response

```
const http = require('http');
const server = http.createServer((req, res) => {
    res.setHeader('Content-Type', 'text/html');
    res.write('<html>');
    res.write('<head><title> Code </title></head>');
    res.write('<body><h1> Hello </h1></body>');
    res.write('</html>')
    res.end();
});
server.listen(3000);
```

attached Header to response — basically pass some meta information

default header — value

write allow us to write some data to response, In chunks

* "res.write" will return data in response with Header-value format, which we have defined in "setHeader"

* Routing Requests

```
Const url = req.url
```

this will show the url path during req,

⇒ file system

```
Const fs = require('fs');
fs.writeFileSync('filename.txt', 'dummy');
```

res from server

```
res.statusCode = 302;
res.setHeader('Location', '/');
res.end();
```

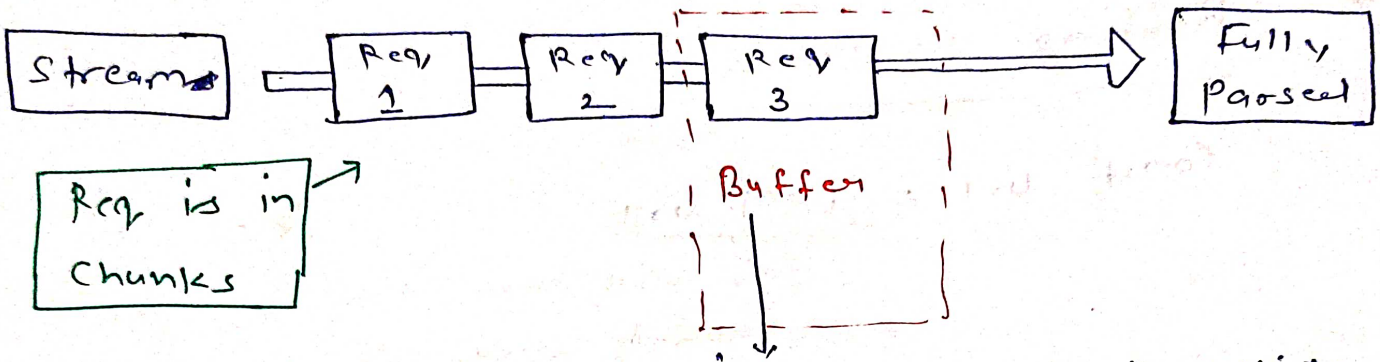
OR

error Handle

```
fs.writeFile("file.txt, data, err => {
    res.statusCode = 302;
    res.setHeader('Location', '/');
    return res.end()
});
```

* Parse the Request Body

✯ Streams & Buffers




```
const body = [];
req.on('data', (↑) => {
    body.push(↑);
});
```

Chunks

will be fired when new chunks ready to be pair

~~const body = [];~~ chunks

```
req.on('end', (
    const parse = Buffer.concat(body).tostring();
                  ↓ global
    const msg   = parse.split('=')[1];
    fs.writeFileSync('filename.txt', msg);
```

* Event Driven Code Execution

As we know that Nodejs is single threaded and it run code line by line.

But If we use Callback then the eventloop come into the picture.

The code ~~block~~ execution is never block due to that Callback.

Event loop will register internally and execute code internally, once all code is executed then only the Callback response will return from event loop.

Information

* * fs.writeFile ↓ this will not wait.

& fs.writeFileSync ↓ synchronous

this will wait/block code execution until file is created

When the server Run then Event loop Started and will Handled Event Callback.

NodeJs Behind scene

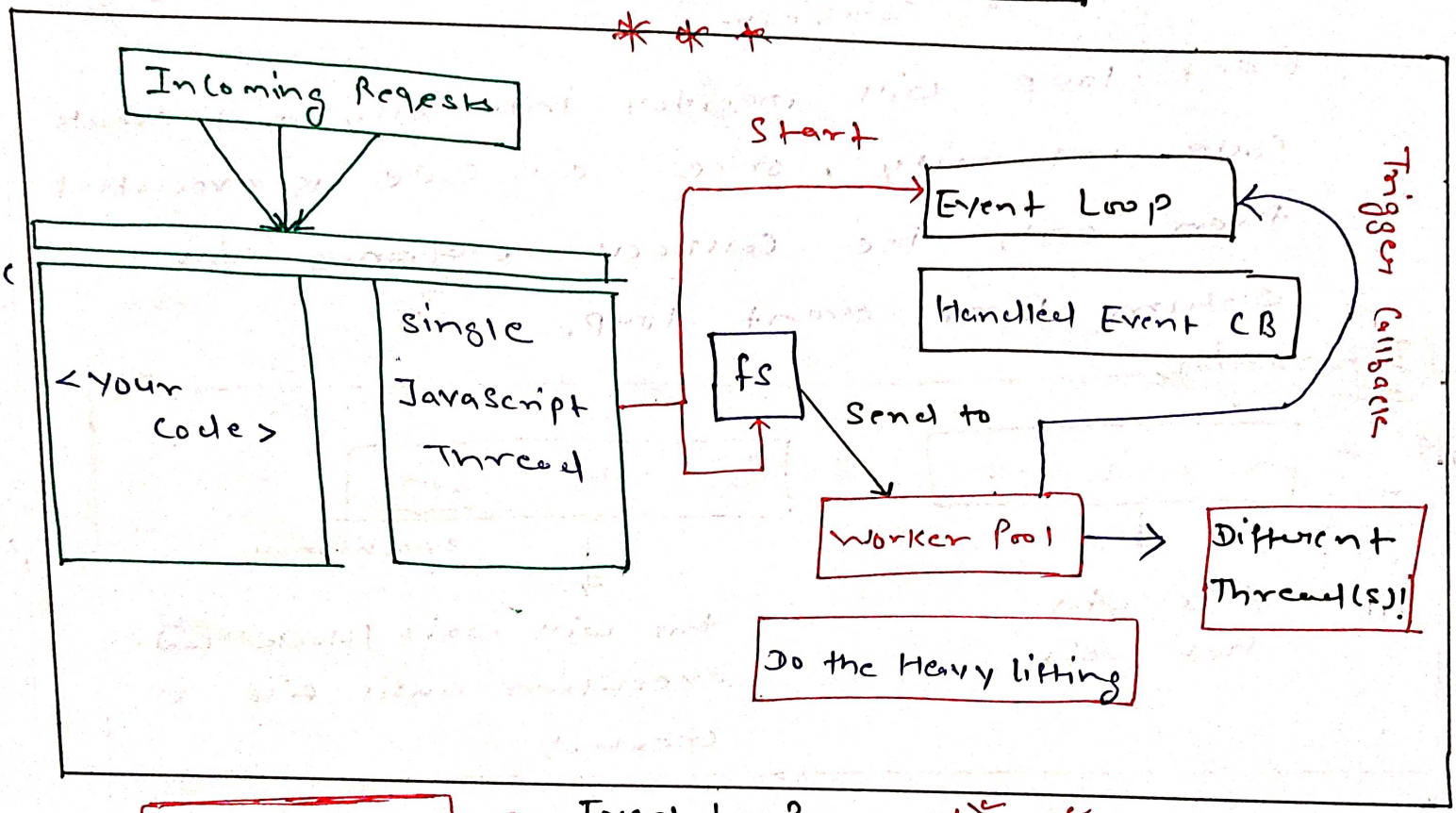
Single Thread, Event Loop & Blocking Loop

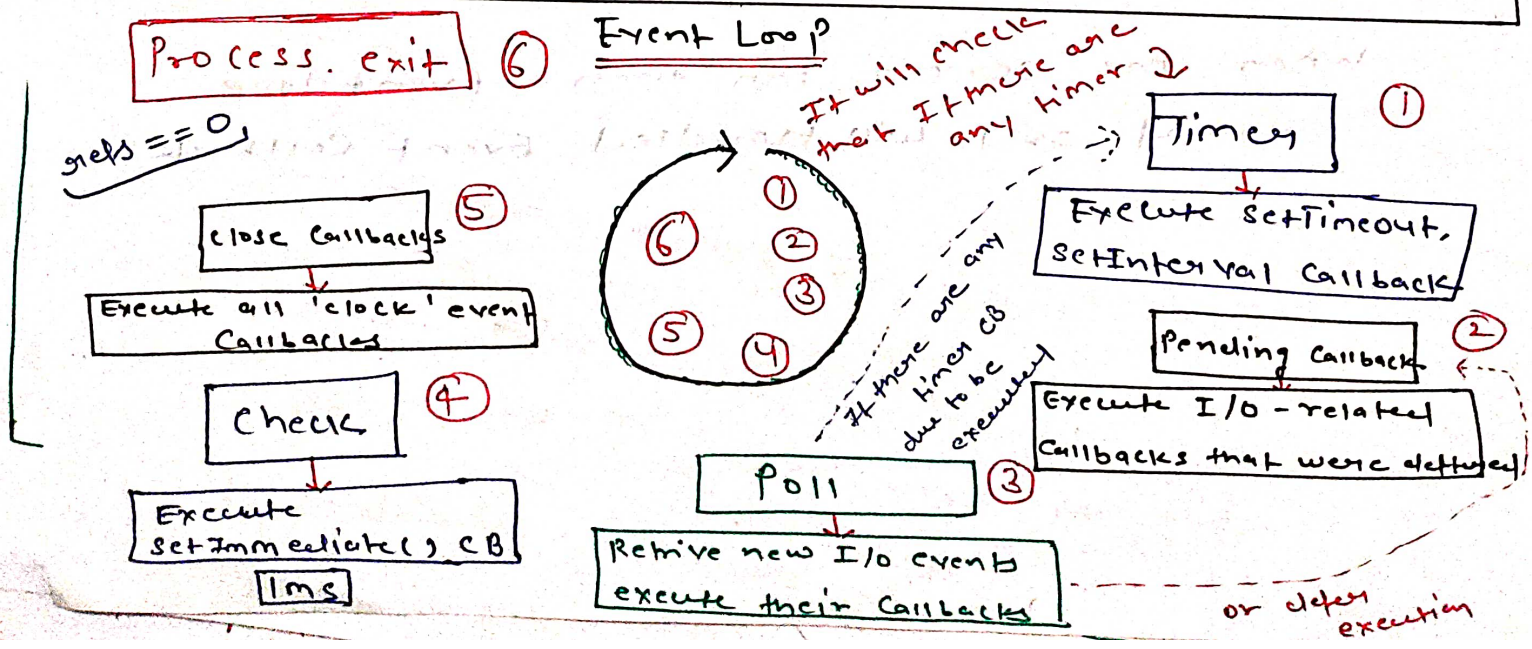
Single Thread

↓

It's a process in operating system

⇒ How It handle multiple Requests ?





* Node module System

module.exports
↓
global

→ After export we can use this file code to any where (read)

1. module.exports = functionName
2. exports.value = functionName
3. module.exports.value = functionName
4. module.exports = {
   value : functionName
   }

Same

* Debugging & Easier Development

1. Syntax error
2. Runtime error
3. Logical error

* we can change the variable value in debug console during runTime.

→ launch.json (You can restart debug config in this file)

```
{
"restart" : true,
"runtimeExecutable" : "nodemon",
"console" : "integratedTerminal"
}
```

Should be Globally

* Express.js framework

1. Server side logic is complex.
2. use a framework for Heavy lifting.
3. free and open-source web application framework for Node.js
4. minimal and flexible (doesn't add to much functionality out of box)

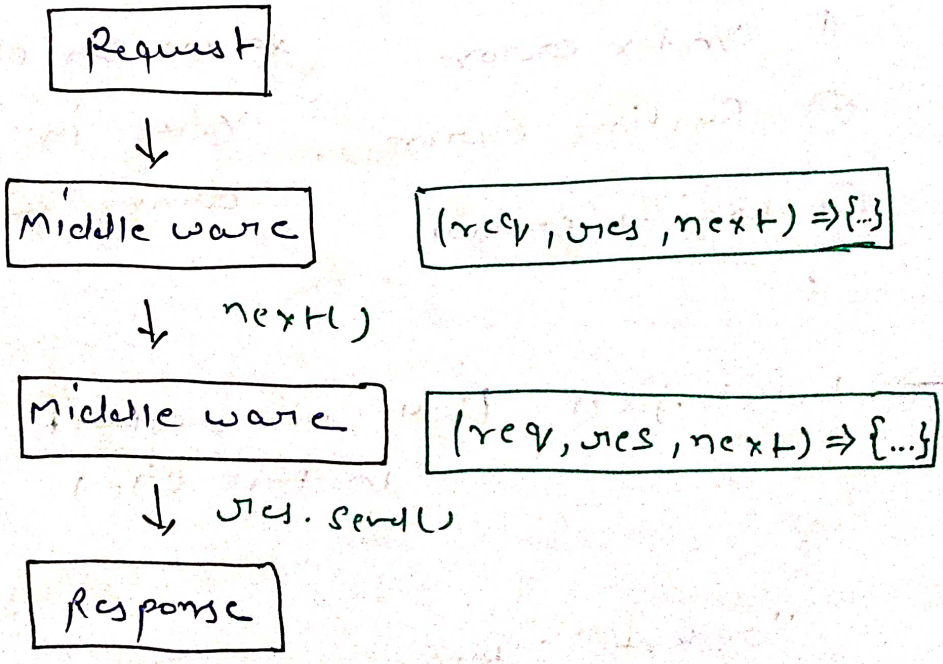
⇒ It is highly extensible and other packages can be plugged into it (middleware)

→ npm install --save express

↓

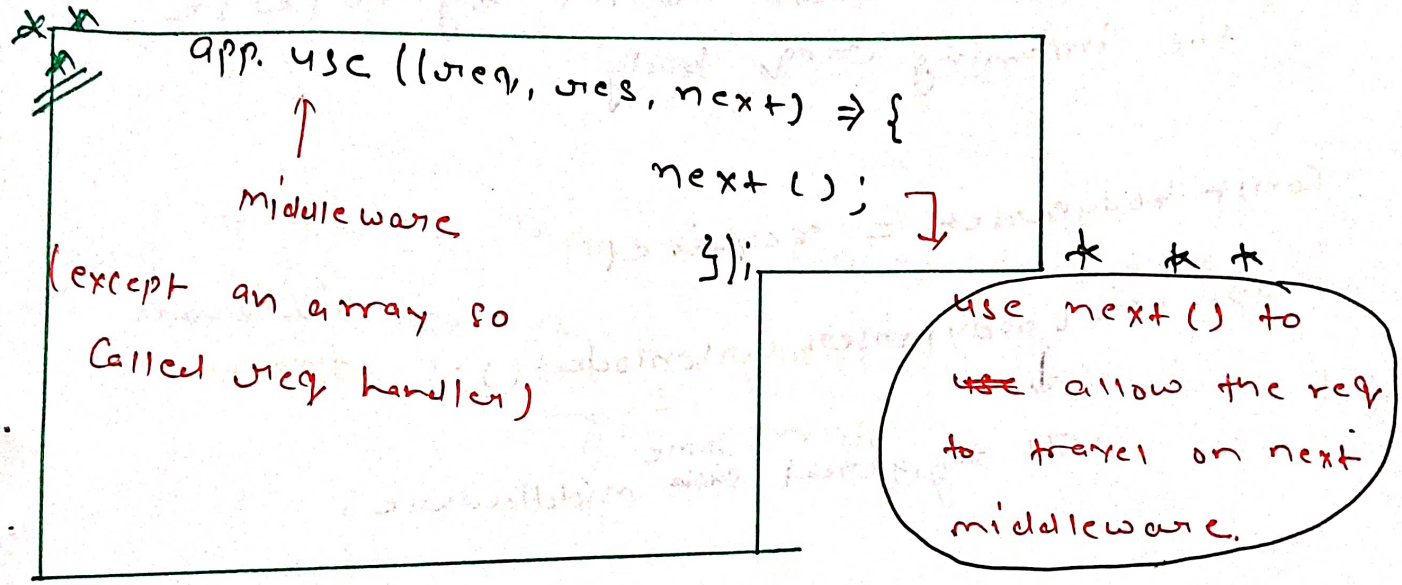
prod

* Middleware



⇒ Middleware means that the incoming request automatically funneled through the bunch of functions by express.js.

```
Const app = express();
```



* express.js doesn't send any default response.

~~app.use((req~~ app.use((req, res, next) ⇒ {

next();

Should be present for next middleware.

or

res.send('<h1> Hello Express </h1>');

});

→ middleware functions are handling the req, res and next() functions for our applications.

→ Handling Routing

```
app.use('/', (req, res, next) => {
    next();
});
```

* (req.body)
  ↳
  By default request doesn't try to parse the incoming req body.

```
const bodyparser = required
app.use(bodyparser.urlencoded());
```

→ middleware function

↓

It registered some ~~such~~ middleware

urlencoded({extend: false})

↓

Non-default

For only get request

```
app.get('/')
```

For only post request

```
app.post('/')
```

→ Express Router

```
Const Router = express.Router();
```

`Router.post('/')`

```
Module.exports = Router
```

In another file

```
const route = require('./Routes');
app.use(route)
```

** app.use → These are calling / finding any route.

app.get
post
: → Strict, will check only route that you provide.

→ Adding 404 Error page (Route)

At the bottom

```
app.use((req, res, next) => {
    res.status(404).send('<h1> page not found </h1>')
});
```

→ Send HTML file to response

```
app.get('/', (req, res, next) => {
    res.sendFile('./HTML/a.html')
```

absolute module.

for the absolute path to send the file from another folder.

→ Const path = require ('path');

~~app.use~~

(E.g) app.get ('/', (req, res, next) => {

** res.sendFile (path.join (__dirname, 'folder', 'a.html'));

(global variable which hold the absolute path on our OS to Project folder)

Also it will point to the current folder for other folder

3)

** res.sendFile (path.join (__dirname, '..', 'folder', 'a.html'))

→ Error 404 during file send

app.use ((req, res, next) => {

res.status (404).sendFile (path.join (__dirname, 'view', 'error.html'))

→ Helper function for Navigation

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

module.exports = path.dirname(process.mainModule.filename);
```

This will return main module (app.js)

Variable available in all file

Import in another file

```
app.use get('/', (req, res, next) => {
res.sendFile(path.join(pathImport, 'folder', 'a.html'));
```

also import path file

root folder

```
app.use(express.static(path.join(__dirname, 'folder')));
```

→ Server file statically (not handled by the express Router or middleware)

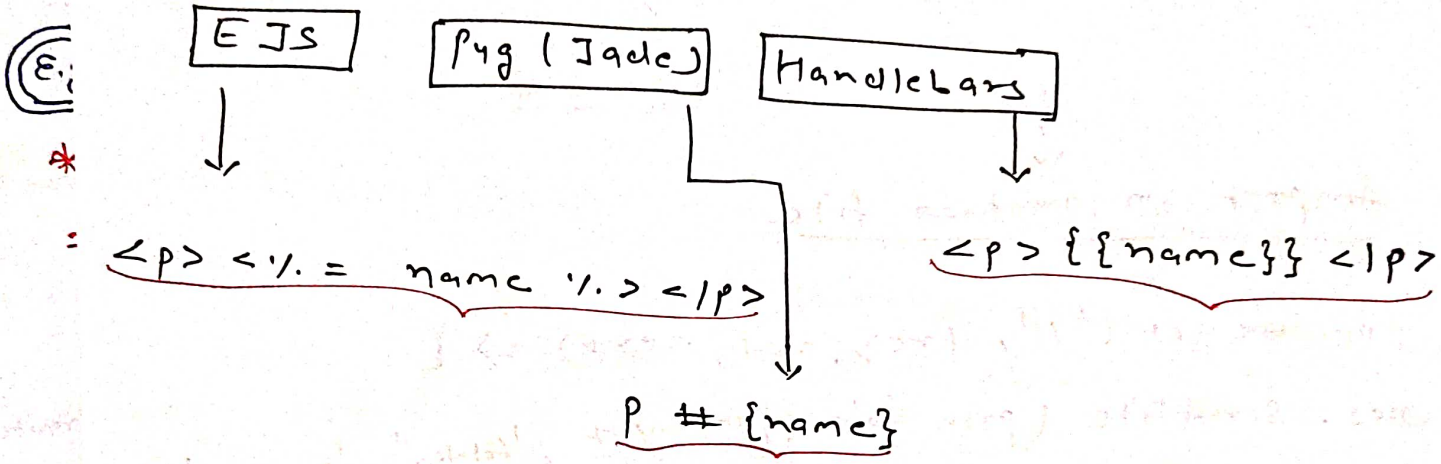
It will handle directly.

→ read access but still more

for → Dynamic Content & Templates

→ sharing data across users

→ Templating Engines



→ npm install --save ejs pug express-handlebars

(Prod)

pug ↓

```
app.set('view engine', 'pug');
```

allow us to set any value globally in our express application key names

```
app.set('views', 'views');
```

allow us to tell express where to find these dynamic views.

```
app.get('/', (req, res, next) => {
    res.render('shop')
});
```

this will work for pug template.

Ex div. main → in shop.pug file

not required optional

↑ class

Header. new

↑ class

a. active (href = "/") SHOP

class attribute text

⇒ const product = admindata. product

res. render ('shop', { prods: products });

allow us to pass in data that should be added in to our view

, title: 'shop'

pass more than one arguments

a. active (href = "/") #{title}

dynamic data

** We can re-use the code (from pug file) by using "Layout"

extends foldername/a.pug

block content

h1 Hello world

block styles

link (rel = "stylesheet", href = "/")

a.pug

block styles

block content

Handlebars

for

- `const expressHbs = require('express-handlebars');`
- `app.engine('handlebars', expressHbs());`
- `app.set('view engine', 'handlebars');`
- `app.set('views', 'views');`

folder name

`<title>{{title}}</title>`

`{{#}}`

Special blocks statements

it's actually wrap some content, o/p – conditionally or in loop

In Handlebars we can't run any logic in our Handlebars template

`{{#if array.length > 0}}` ~~array.length > 0~~

if else

`{{#else}}`

`{{/if}}` → close

We can use only true/false

`{{#if hasValue}}` etc.

→ Layout Handlebar

```
app.engine('handlebar', expressHbs({layoutDir:
    'view/layout/', defaultLayout: 'name'
    , extname: 'handlebar'
    }));
```

'view/layout/' → folder where file exist

'name' → file name

extname → for only handlebars

for Extended

{{{ body }}}

placeholder

→ The Model View Controller

Models

→ objects / part of our code that is responsible for representing your data, it allowing to work with data.

E.g save data, fetch data, update data

Views

→ is responsible for what the user seeing on the end, rendering the write content in our html document and sending that back to user.

Controllers

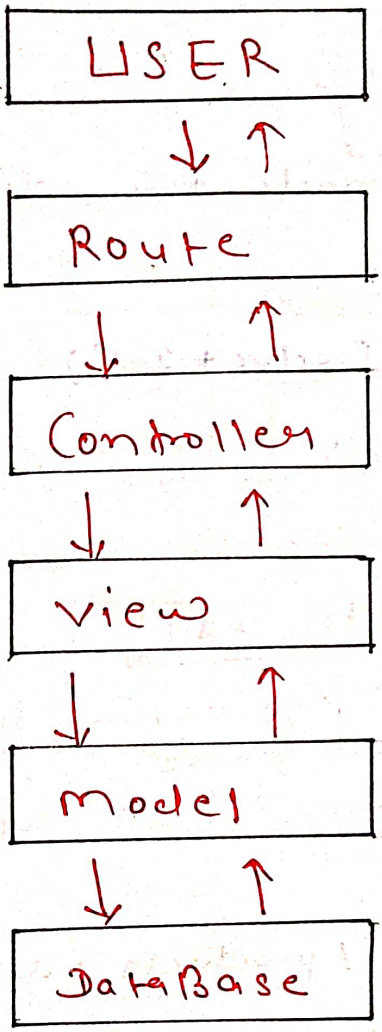
→ are the connections point between the models and views.

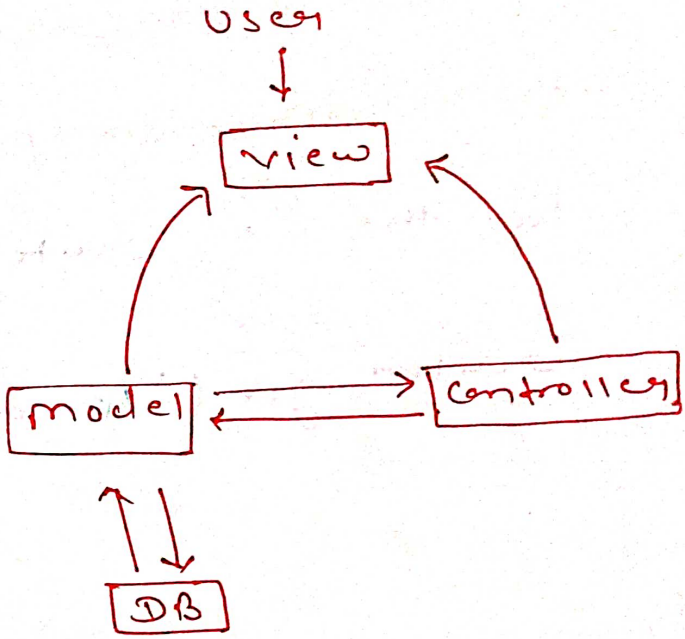
→ Views shouldn't care about the application logic and models do care about how to fetch data n all.

Route

→ defined that which http method or which controller should be Executed.

MVC pattern


USER
Route
Controller
view
Model
DataBase


User
view
model
Controller
DB

→ Dynamic Routes

https://localhost:3000/products/1

for the "1" route, we provide the diff route

⇒ router.get('/products/:productId')

It should not look for route but it can be anything

"/products/1234"

get the information through that name

→ Always write the dynamic route after specific route

normal

E.g router.get('/products/delete')

→ Then dynamic route

In Controllers

exports.product = (req, res, next) ⇒ {

const prodId = req.params.productId

→ Dynamic Routes

https://localhost:3000/products/1

For the "1" route, we provide the diff route

⇒ `router.get('/products/:productId')`

It should not look for route but it can be anything

"/products/1234"

get the information through that name

→ Always write the dynamic route after ~~specific~~ specific route

normal

E.g `router.get('/products/delete')`

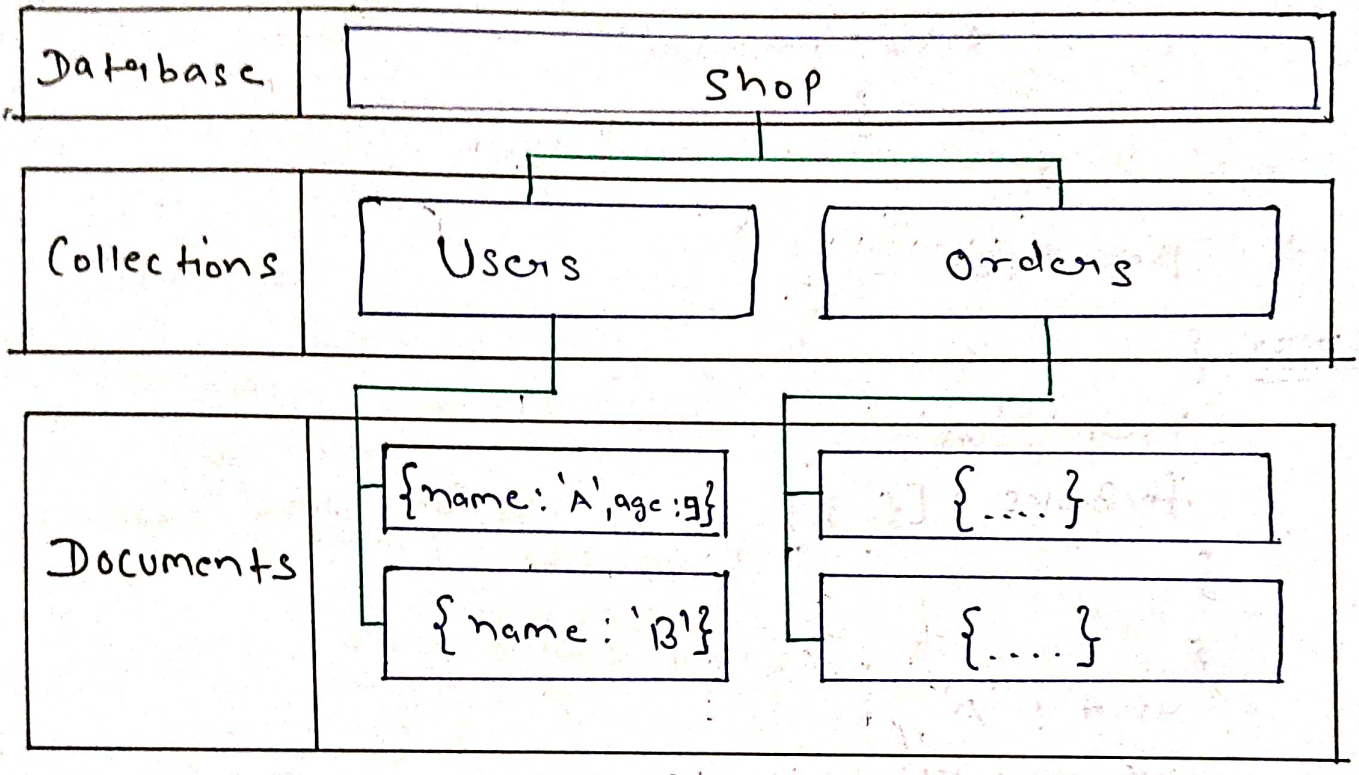
→ Then dynamic route

In Controller

```
exports.product = (req, res, next) => {
    const prodId = req.params.productId
```

→ NOSQL using MongoDB [Fast]

*→ MongoDB is built for large scale application
→ it can store lots and lots of data



→ It is schemaless.
→ Uses JSON to store data in Collections.
→ We can fetch data without having to combined multiple collections behind the scenes on the server.

A collection
↓
{ id: 1,
name: 'A'
}
{ id: 2,
name: 'B'
}

B collection
↓
{ id: 2,
name: 'OK'
}

→ Nested / Embedded Documents

```
{ id: 1,
  name: 'You'
    address: {
        city: 'mumbai'
            }
    }
```

→ References

Customer {
user : 'A',
favBooks : [{...}, {...}] (lots of data duplication)
}

{
user : 'A',
favBooks : ['id1', 'id2'] → It's managed with different collections
}

Book {
_id : 'id1',
name : 'Harry Potter'
}

→ You can relate documents but you don't have to (and you shouldn't do it too much or your queries become slow)

→ try to Install mongoDB on ~~cloud~~ Cloud, which is good for practice.

Connection

```
Const mango = require ('mongoDB');
Const Connection = mongo.MongoClient();
Connection.Connect('url').then(result => {});
                          .Catch(err => {});
```

```
let _db;
Connection.Connect('_____//'database'/_____')
                    URL
  .then(client => {
    _db = client.db()
  })
  .Catch(err => {
    throw err
  });
```

this will call in app.js

here we can pass new database name

In app.js

```
mongoConnect(() => {
  app.listen(3000)
});
```

→ fetch all records from mongodb)

```
Const _db = db.client();
return _db.collection('').find().toArray()
    .then(A => {});
    .catch(err => {});
```

To return all documents in js array (for less records) otherwise Pagination

It will not return immediately promises instead return so called cursor (object), which allows elements or documents step by step.

```
new
_id = mongoDB.ObjectId(prodId)
```

In c

Working with Mongoose

→ Mongoose is object-document mapping library (ODM), which is similar to sequilize (ORM) object relational mapping.

ODM → object data modelling
object document mapping

→ Manage relationships between data, provides schema validation, and is used to translate between objects in code

→ It creates model abstraction which makes it easier to work with, so it look's like you are working with just objects rather than pure data.

Modals

```
Const mongoose = require('mongoose');
Const schema = mongoose.schema;
Const q = new schema({
    name: String,   name : {
                        type : string,
                        required : true
                    }
});
```

module.exports = mongoose.model('product',); which we defined for schema

↑ (28)

important for mongoose behind the scenes to connect a schema blueprint with a name.

→ In mongoose if we pass id, it automatically connect with an objectId.

* Relations in mongoose

→ for the ref of schema, You need to add below ref/relation.

```
Const schema = new schema ({
    name: {
      type: string
      required: true
        },
    UserId: {
      type: Schema.Types.ObjectId,
      ref: 'User'
        }
});
```

In

Product.find().populate('User Id').then(res => {})

↓
for all the details info
You can pass nested path, if it is there

User Id.user

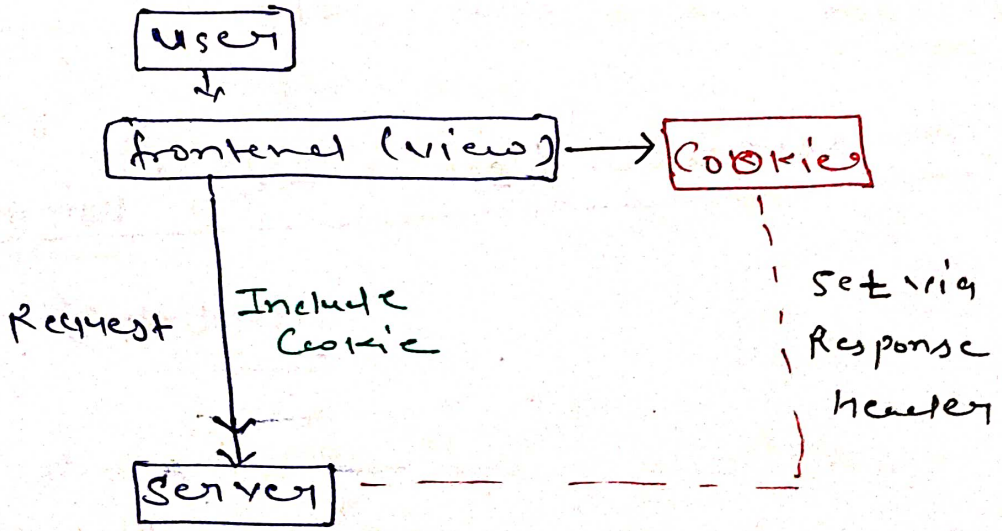
- Select('name') or Select('name age -_id')

↑ which field you want to select

exclude Id (from nested)

Session & Cookies

Cookies



→ Cookies are stored on the client-side.

res.setHeader('Set-Cookie', 'login=true');

↑ reserved name

↑ key value

↓ set

req.get('cookie')

⇒ we can manipulate / change the value of cookie from inside the browser.

You can set multiple cookies.

res.setHeader('Set-Cookie', 'login=true; Expire=date');

'HttpOnly' ↓

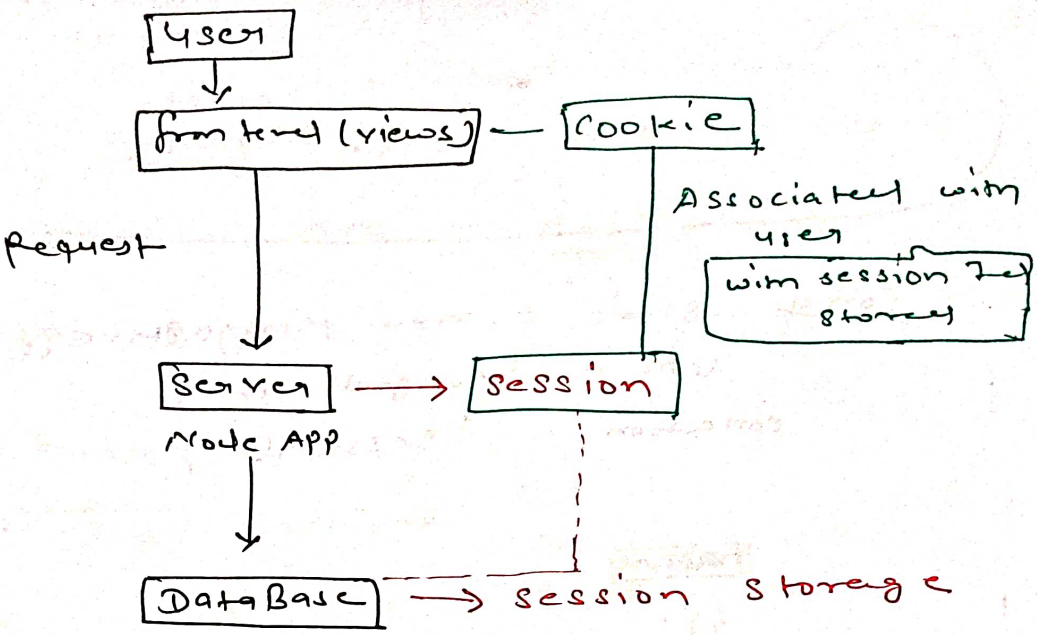
After use this "You can't read the value in Browser" (JS)

↓

Secure

only with HTTPS

Sessions



* session are stored in server side.

→ npm install expression-session

→ app.use(session({ secret: 'my secret',
resave: false, → default not save on every req
saveUnitialized: false
}));

* By default session is save in memory in server side

→ req.session

Note: for more data like 1m, 1mm of user the memory will be overflow.

So store in database that is the best way.

npm install --save [redacted] connect-mongodb-session

```
const session = require('express-session');
const mongostore = require('connect-mongodb-session')(session);

const store = new MongoStore({
    uri = 'mongodb uri',
    collection = 'sessions',
});
```

↑ any name

```
app.use(session({
    secret: 'abcde',
    resave: false,
    saveUninitialized: false,
    store: store
}));
```

How to clear cookie

```
req.session.destroy(() => {
    req.session → out
});
```

⇒ Authentication

For encrypt password

→ npm install --save bcryptjs *

Const bcrypt = require ('bcryptjs');

let pass = bcrypt.hash (password, 12);

↑ How many round hasing will be applying

Password : pass

↓ during save password

⇒ You cannot decrypt that password

→ After login How to match password ?

bcrypt. compare (password, user. password);

return promise

* Create a diff folder and write session authentic code there then used that auth code to route.

router. get ('/', auth, Controller);

→ CSRF Attack

Cross - site Request Forgery

→ If some fake page will be show on screen which is almost same as your page.

→ But due to session they can get your data from server.

→ CSRF TOken

npm install --save csurf

↘ CSRF token

app.js

```
Const csrf = require('csurf');

Const csrfProtection = csrf();
                       ↳ default
```

after session initialised

```
app.use( csrfProtection
```

Should have

You to pass that csrf token to payload

like `csrfToken: req.csrfToken()`

↓ which is used all post payload

↓ method

* Instead of passing same value in payload do one thing just pass a middleware for all the variables / same values.

E.g. app.use((req, res, next) => {

res.locals. —— properties,

↑ allow us to set local variables are passed into the views

});

** → Authentication means that not every visitor of the page can view and interact with everything.

→ Authentication has to happens on the server-side and builds up on sessions.

* Send mails

→ We will use third-party service, Because Nodejs you can't create mail servers.

↓

because Handling is not easy for many emails at the same time

(Complex) (Security)

→ Sendgrid

* npm install --save nodemailer nodemailer-sendgrid-transport

```
Const nodemailer = require ('nodemailer');
Const transport = require ('nodemailer-sendgrid-transport')
Const transporter = nodemailer.createTransport(
                    transport ({
    auth: {
    api_key: '12345'
        }
});
```

* Create key from sendgrid account e.g.:- '12345'

```
transporter.sendMail({
    to: 'abc@abc.com',
    from: '____@.com'
    subject: 'signup successful',
    html: <h1> OK </h1>
});
.catch (err => { });
```

* Authentication & Authorization

Const crypto = require('crypto');

crypto.randomBytes(32, (err, buffer) => {

if (err) { — }

Const token = buffer.toString('hex');

↓

dec value

Convert in string

In model

- resetToken : String,
- resetTokenExpiration : Date

during reset pass

User.find → email

if (present) {

user.resetToken = token

user.resetTokenExpiration = Date.now() + 3600000;

return user.save();

.then(res => {

// Send mail using transporter sendmail

** find token

Const token = req.params.token;

User.findOne({ resetToken: token,
resetTokenExpiration: { $gt: Date.now() } }).
then(user

→ data

Send to the react form with user id

Catch(error)

* Validations

Const Validator = require('express-validator');

express-validator / check

OR

Const { check } = require('express-validator');

check('email').isEmail()

→ const error = Validator(req);

* noValidate → Client side validation

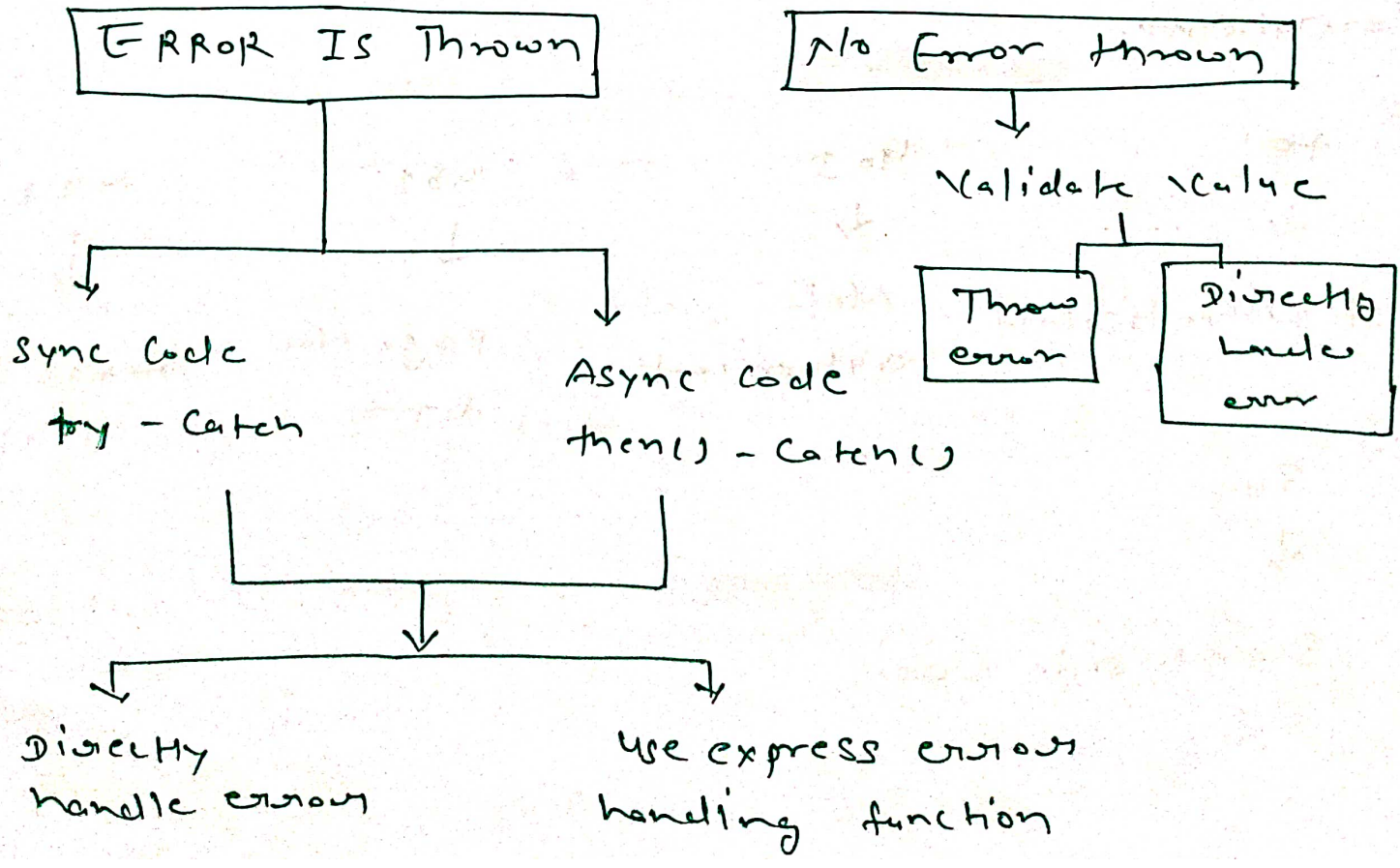
{ check, body }

⟹ ERROR Handling

* Errors are not necessarily the end of your app.

↓

Just to need handle



☆ Errors and HTTP codes ☆

2xx (success)

3xx (Redirect)

4xx (client side error)

5xx (server side error)

200
↓
operation succeeded

201
↓
success, resource created

301
↓
moved permanently

401
↓
Not authenticated

403
↓
Not authorized

404
↓
page not found

422
↓
invalid input

500
↓
Server side error.

* File upload & Downloads

npm install --save multer -it's like middleware

↑

{ Parse incoming req for files }

use enctype = "multipart/form-data"

⇒ for file upload, this contain not plain text but will contain mix data.

app.js Const multer = require('multer');

** app.use(multer().single('image'));

↑ input

or multer(

another file Const image = req.file

({ dest : 'images' })

↳ Convert buffer to path

multer is accept incoming data, Extract file and stored in for us and store the info about the file uploaed.

const fileStorage = multer.diskStorage();

{ destination: , filename: } >next page.

** * fileStorage

destination : (req, file, cb) => {

cb(null, 'image');

},

filename : (req, file, cb) => {

cb(null, file.originalname)

// two images with the same name, they donot override each other.

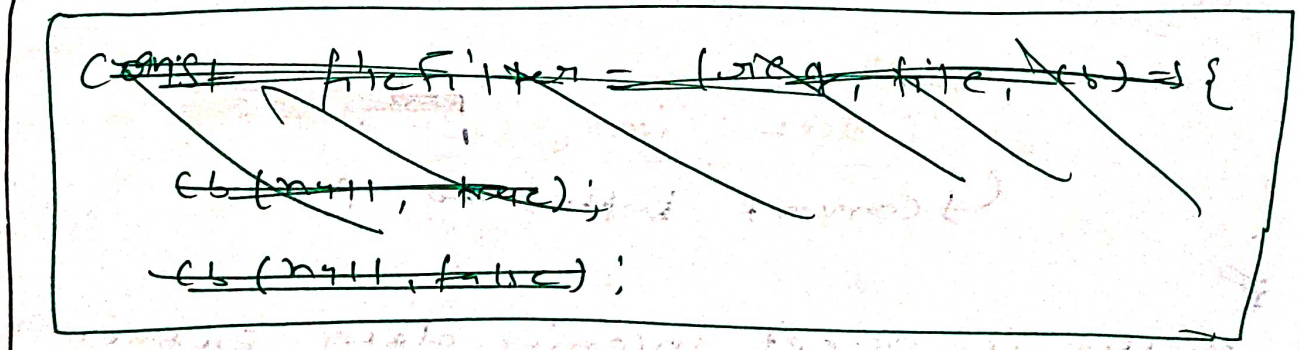
or

cb(null, file.filename + '-' + file.originalname)

or you can use date

}

app.use(multer({ storage : fileStorage, fileFilter : filter }).single('image'));



const filter = (req, file, cb) => {

if(file.mimetype = 'image/png' || 'image/jpg' || 'image/jpeg')

{

cb(null, true);

} else {

cb(null, false)

}};

* app.use(express.static(path.join(__dirname, 'images')));

'/images', {statically}

* Download file with Authentication

→ Set up your separate route

router.get('/orders/:invoiceId')

- Const invoiceId = req.params.orderId;
- Const invoiceName = 'invoice-' + ~~order~~ invoiceId + '.pdf';

('fs') require

Const path = require('path')

it works on all the OS

~~fs~~ Const invoicePath = path.join('data', 'invoice', filename)

fs.readFile(invoicePath, (err, data) => {

if (err) {
return next(err);
}

res.setHeader('Content-type', 'application/pdf');

res.send(data);

res.setHeader('Content-Disposition', 'inline');

or attachment

How this content should be served to the client

filename = "1_"

* Or

A => fs. createReadStream

A. pipe (res)

① pdfkit

→ Pagination

Mongoose have a method

→ .skip((page - 1) * total)

→ .limit(total)

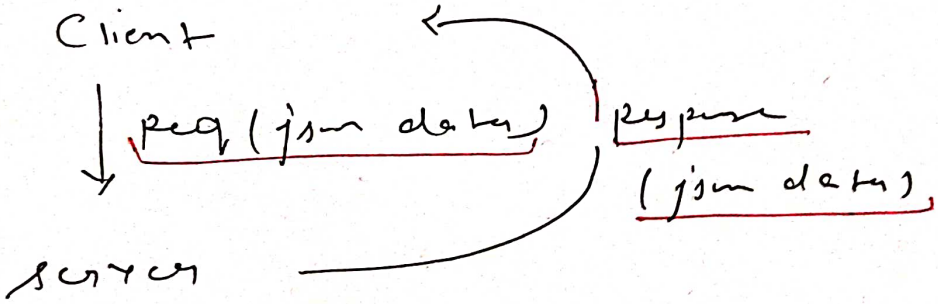
find the total Count of data ⟹ Product.find().countDocument().

then(Abc => {

skip limit

})

→ Understanding Async Request



for delete Router.delete('/product/:id', () —)

In express.json → you can return

⇒ res.json()

→ res.status(200).json()