Node Js

DATE 25 May

⊙ Node JS is Framework / Platform, Javascript is language

NodeJS → console application (Node page1.JS)

⊙ module → file having JS extension → JS code
↳ It may have functions, classes, variables, etc

⊙ page1.JS

```
function add(P1, P2){
  console.log('${P1}+${P2}=${P1+P2}')
}                    10   20   30

console.log(module)

// Module is hidden variable
   added by node for every
   page (module)

① // export required function

module.exports = { // json object
Func ← myadd : add
   //mySubstract : substract,
   //mymultiply : multiply
   // key : alias
}

console.log(module)

// in JS file not needed to
   use "" / '' not for
   keys in object
```

⊙ page2.js

```
// import required module
// require Func loads
   the module (page1.js) &
   returns exported object
   from loaded module

const page1 = require('./page1')
              Func

O/P : {
   myAdd : [Function : add],
}

page1.myAdd(10, 20)
O/P : 10+20 = 30

// Func are not defined in
   page2, these are exported
   from page1
```

```
// Import File System Modwe
const fs = require('fs')

function writeContents() { // Write Contents in File
    fs.writeFileSync('./myFile.txt', 'India is ------')
}

writeContents() // Function call

// Reading File sequentially synchronously
function readMyFileSync() {
try { //operation1: const data = fs.readFileSync('./myfile.txt')
                    console.log(data) → ASCII
                    console.log(string(data))
} catch(ex) { console.log('exception handled') }
//operatn2: const result = 123145688 93 * 614532193 * 216845
                    console.log(result)
}
readMyFileSync()
```

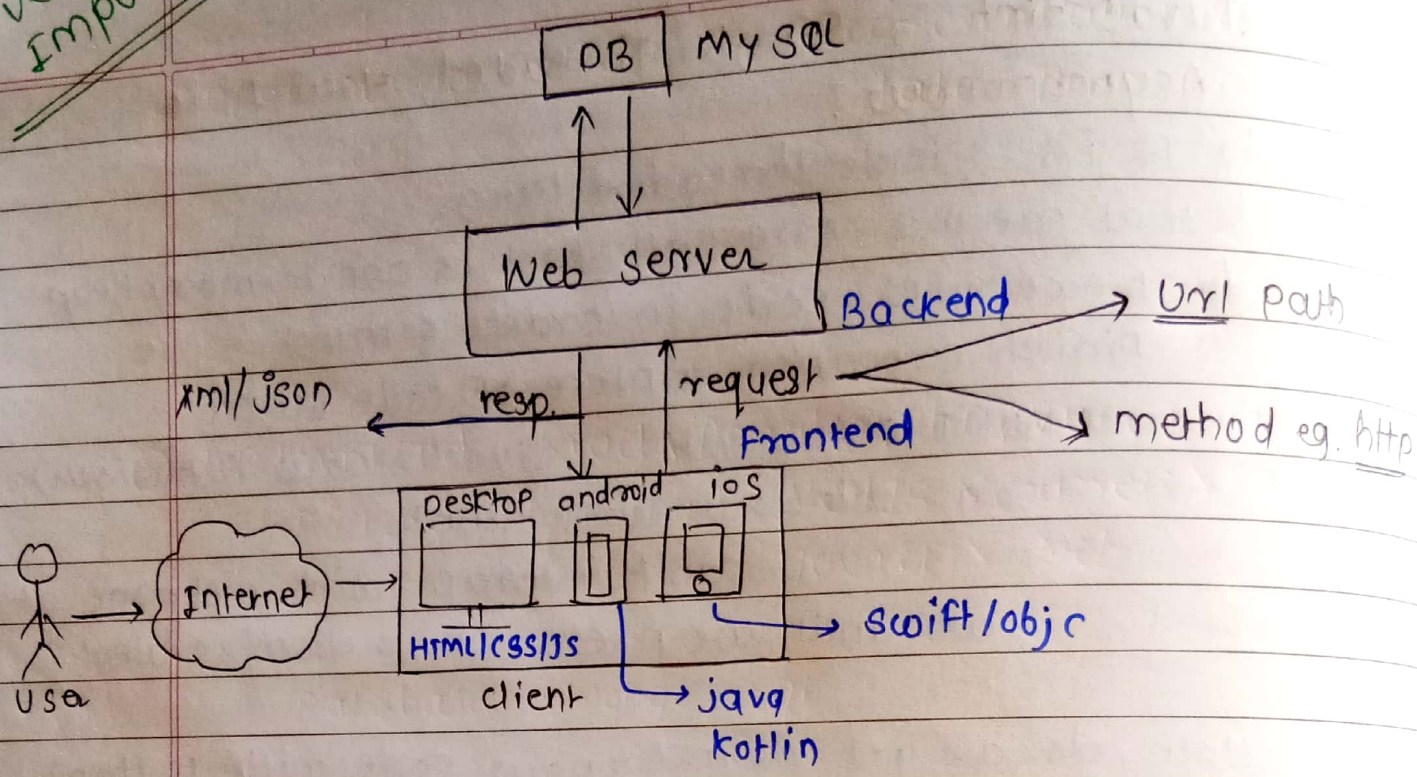
Create File
eg. myFile.txt

o/p : All the statements will executed one by one
o/p → is always predicted

// Suppose Filecontents are huge in MB's & GB's then next lines of prog. will be blocked till File is reading in progress unHill entire content read
// ∴ Sync funcns are known as Blocking Funcns / APIs

// In Synchronous Mode, if there is error in executing Funcn then code will throw an exception then do exception handling

*very Important



⊙ Backend → For Data Base Connectivity (server-client)
- → Design patterns to provide connectivity → REST, graphQL
- → Invisible to end user
- → Also known as Middleware, webserver, webservices
- → Technologies :
  - → Java Enterprise Edition
  - → Javascript → Express, fastify, hapi
  - → TS → NestJS
  - → PHP → codeIgnator, etc
  - → Python → Flask

Database operation

1. Create → eg. new user
2. Read → eg. Log in
3. Update → eg. change password
4. Delete → eg. Remove user

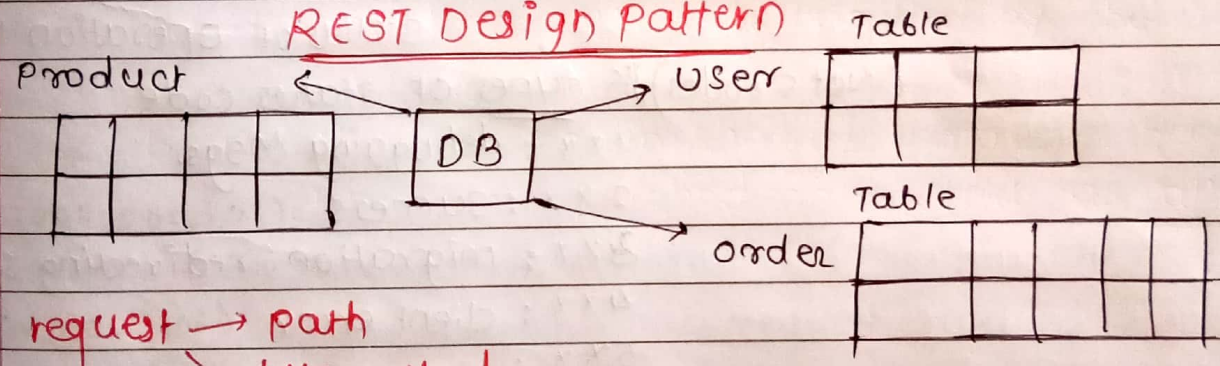
client needs to tell server which opern to perform using http methods

http Methods
- → POST
- → GET
- → PUT, PATCH
- DELETE

- **Frontend**
  - User Interaction
  - visible to end user
  - Provides Backend connectivity
  - Also known as Client → Desktop, android, ios
  - Technologies
    - JS, HTML, CSS
    - VueJs
    - Angular
    - React
    - NextJs
    - jquery
    - python → Django

- **Fullstack** → MERN → Mongo/Mysql + Express + React + NodeJs
  MEAN
  WISA
  LAMP

- REST Design Pattern



request → path
→ http method

client requests server to perform operation
using http method
↓
server will get request with method = POST / GET / PUT
along with path

Create → POST → insert INTO
Read → GET → Select From
Update → PUT / PATCH → UPDATE
Delete → DELETE → delete From

eg post/order : it means insert into order Table

REST Design Pattern → Representational State Transfer

Request to Server → Path & Method

Response of Request → Json / XML Format / html / text

① Top Frontend : React, Angular, PhP, Vue, Java

Backend : Typescript : NestJS > express > FastHfy > PHP

Java EE > .Net

Database : Mongo, SQl

Response : may contain html, json, xml, text

↳ contains:

Header

① status code : For client to check the status of operation perform

(Not 5 codes) // 5 types of status code

1XX : debugging Msgs.

2XX : success (201, 202, 200 ...)

3XX : migration (redirecting server)

4XX : client error (401, 403, 404)

5XX : Server error (500, 502)

console.log (http.STATUS_CODES)

response.setHeader ('Content-Type', 'text/html')

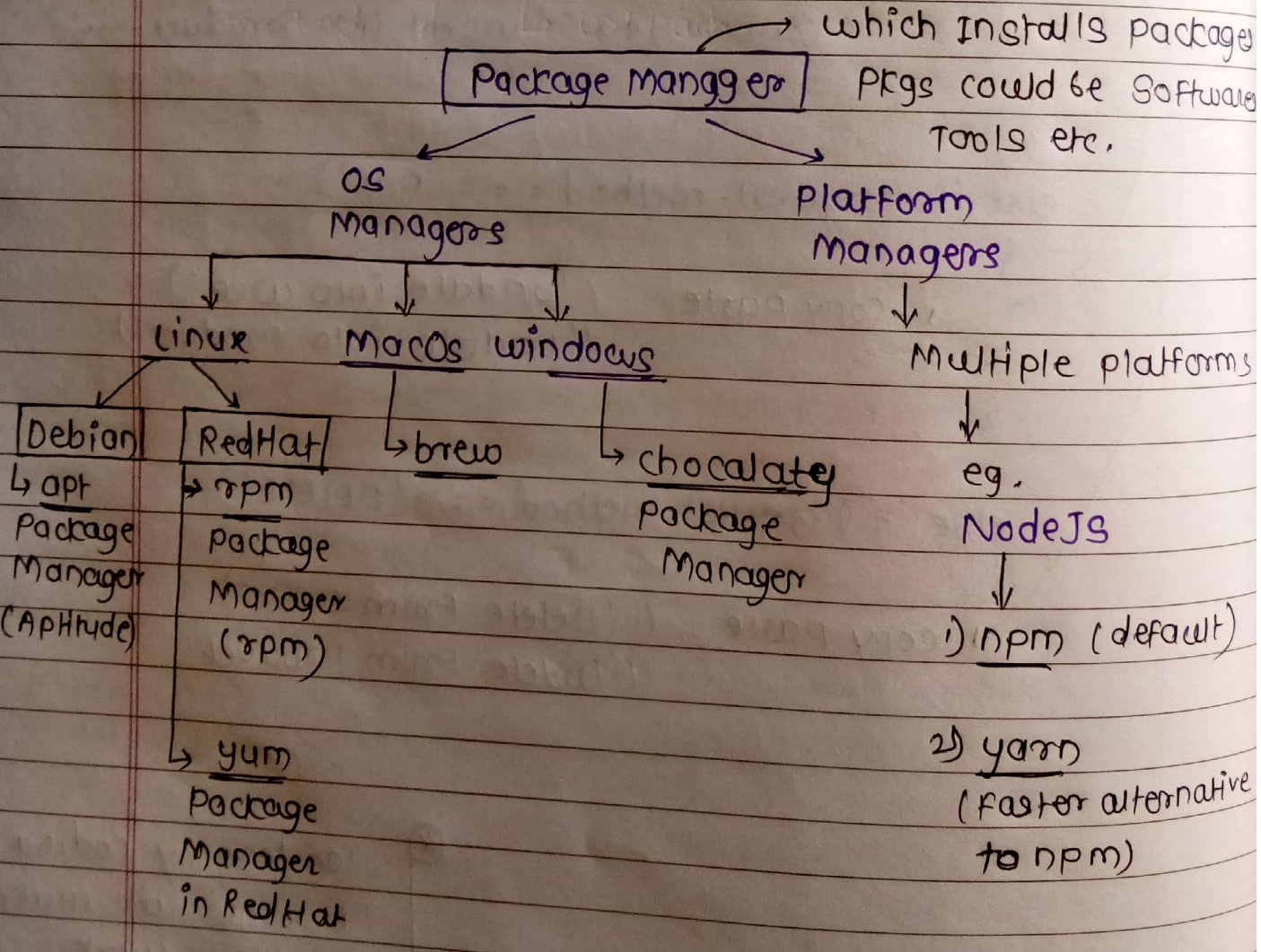
② content Type : text / plain / json

http built in module in
express is third party utility

⊙ **Express** : Fast, unopiniated, minimalist web framework for Node.js

```
const express = require('express')
```

Framework / pkg. developed on TOP OF NodeJS
// By default Not available in NodeJS
// So Download express
// Node Pkg. Manager (NPM) downloaded already with Node JS
// you can ask NPM to download express it for you
$ npm install express



```
// import express
const express = require('express')

// create express application (express server)
const app = express()

// start express application
app.listen(4000, '0.0.0.0', () => {
    console.log('application started on port 4000')
})

// route → mapping of http method & url (path)
// app.get('/')
method ↓   ↳ url path

app.get('/', (request, response) => {
    console.log('request recieved')
    response.end('GET/')
})

app.post('/', (request, response) => {
    console.log('request recieved')
    response.end('POST/')
})
```

eg:

product

① app.get('/', (request, response) => {
    // code here to process request
    response.~~GET~~ send ('GET /')
})

product

app.post('/', (request, response) => {

    ~~ret~~ response.send ('POST /')
})

const product = [{id: 1, title: 'product-1', price: 100}]

① New Directory → open in vs code → New file
② yarn init -y → for package.json
③ yarn add express → node_modules

```
const express = require ('express')

    const app = ('express') express()
// routes for products
app.get ('/product', (request, response) => {
        console.log ('get all products')
    response.send ()
})
app.post ('/product', (request, response) => {
        console.log ('insert into product')
    response.send ()
})
app.put (
        // code
})

app.delete (
        // code
})




app.listen (4000, '0.0.0.0', ()=> {
        console.log ('server started.... port 4000')

})
```

// Middleware → Feature provided by express
↳ Application to call automatically everytime
↳ Before calling actual route
↳ application will pass request, response and next parameters automatically
↳ request : request sent by client
response : response to be sent to the client
next : function reference / alias of next function to be executed (mostly refering actual route)
↳ it is parameter name, you can say anything myNext or else

⊙ can we have multiple Middlewares ? → yes

eg:
```
function mylog1 (request, response, myNext) {

    console.log ('request Method: ${request.method},
                  URL : ${request.url})

    myNext()
}

app.use (mylog1)

// Another Middleware
const mylog2 = (request, response, next) => {

    console.log ('----Mylog2')
    next()
}

app.use (mylog2)
```

⊙ Anonymous Middleware → directly written in app.use

```
app.use ((request, response, next) => {
        console.log ('---- Anonymous Middleware')

        next()
});
```

// then write all route and then run server

eg.
```
app.get ('/ ', (request, response) => {
        console.log ('inside GET /')
        response.send ()
})
```

eg.
```
app.post ('/product', (request, response) => {
        Console.log ('Inside /POST')

        response.send ()
})
```

eg.
```
app.put (.....
app.delete (.....
```

```
app.listen (4000, '0.0.0.0', () => {

        console.log ('server started on port 4000')

})
```

DATE 27 May

⑥ Eg // Routes for Person, Mobile, Product in server.js

```
const express = require ('express')
    Const app = express
//① person
app.get ('/person', (request, response) => {
        console.log ('GET')

        response.send()
})
app.post ('/person', (request, response) => {

        Console.log ('post /Person')
        response.send()
})
app.PUT ('/person', (request, response) => {

        console.log ('PUT/person)
        response.send()
})
app.delete ('/person', (request, response) => {

('delete/person)
response.send()
})
//② Route for mobile
app.get ('/mobile - - - -
app.post ('/mobile - - - -
app.put ('/mobile - - - - -
app.delete ('/mobile - - - - -
//③ Route for product
app.get ('/product - - -
app.post ('/product - - -

app.listen(4000, '0.0.0.0', () => {
// prog. will be lengthy so use
})
```

server.js	person.js	mobile.js	product.js
Create server	route Person	route mobile	route product
Run server			

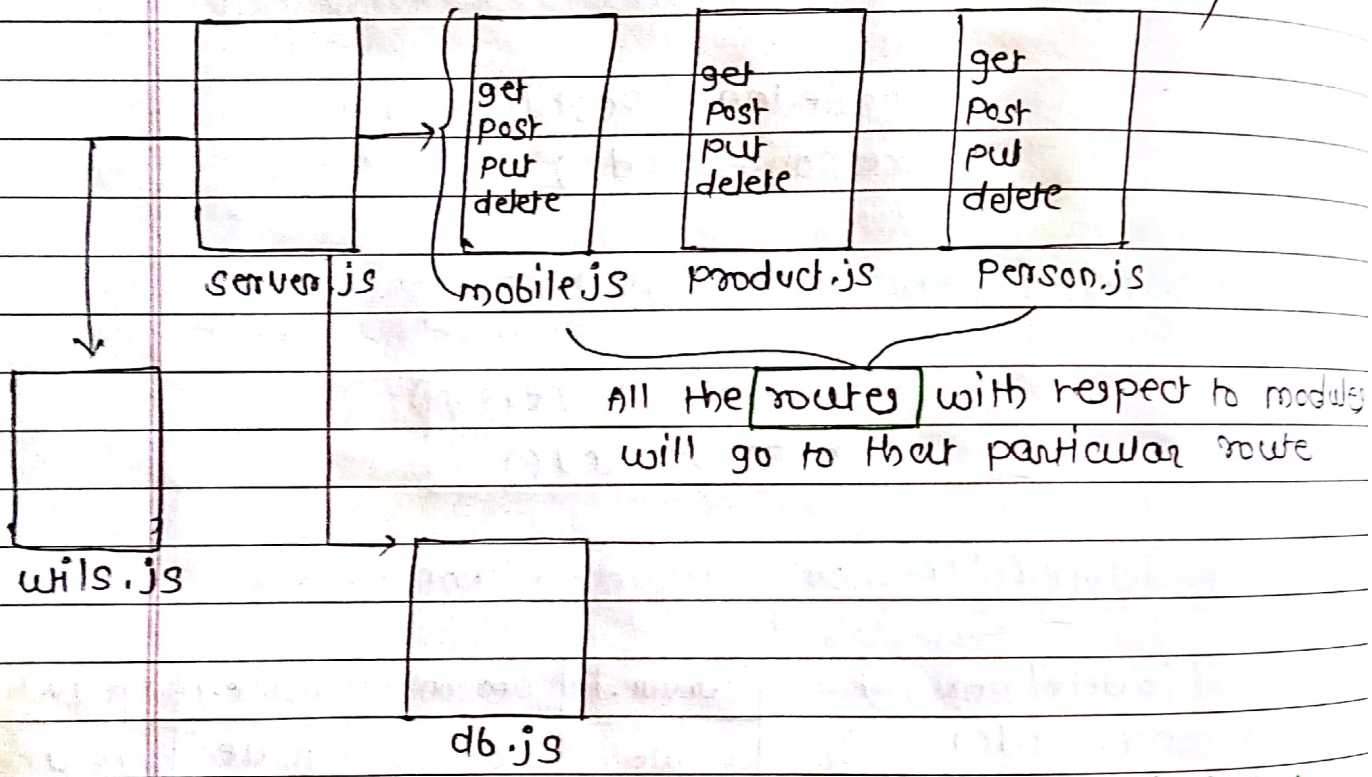
Different Modules (Modularity)

⊙ Install express : yarn init -y
yarn add express

Achieve

How to ~~add~~ Modularity in Program?

↳ To eliminate bulkiness / lengthiness of code
↳ write seperate files for modules / routes & then export it on server
↳ new directory → Routes → create multiple modules to represent routes of multiple entity



⊙ Instead write everything in same file, it is better to write seperate modules

⊙ Working of express behind scene:

server created

app = express()

This express provided something called Router

app.get('/', ---)

app.post('/', ---)

Router → maintains Routing Table

Path	Call Back
GET/-	
GET/person	
post/mobile	

Routing Table

server.js

```
const express = require('express')
 const app = express()

// add the Routes
const personRouter = require('./routes/person')

const mobileRouter = require('./routes/mobile')

const productRouter = require('./routes/product')



// use the Router        // middleware feature
 app.use(personRouter)

 app.use(mobileRouter)

 app.use(productRouter)



app.listen(4000, '0.0.0.0', () => {
        console.log('server started on port 4000')
})
```

person.js

```
const express = require('express')

// get the router
const router = express.Router()

// routes for person

1) router.get('/person', (request, response) => {
        console.log('GET / person')
        response.end()
   })

2) router.post('/person', (request, response) => {
        console.log('POST / person')
        response.end()
   })

3) router.put('/person', (request, response) => {
        console.log('PUT / person')
        response.end()
   })

4) router.delete('/person', (request, response) => {
        console.log('DELETE /person')
        response.end()
   })

// Export router with all routes related to person

module.exports = router
```

mobile.js

```
const express = require('express')

const router = express.Router()

router.get('/mobile', (request, response) => {

router.post('/mobile', (request, response) => {

router.put('/mobile', (request, response) => {

router.delete('/mobile', (request, response) => {


module.exports = router
```

Product.js

```
const express = require('express')
const router = express.Router()

router.get('/product', (request, response) => {

router.post('/product', ----------

router.put('/product', ----------

router.delete('/product', ----------

module.exports = router
```

⊙ Data Persistance → Using DataBases for persisting our Data
- mysql
- RDBMS
- mongo

} anything

⊙ Make a new Directory → Install 2 modules
1) mysql module
2) express module,

```
> yadd yarn add express mysql
```

or

```
> npm install mysql express
```

Packages

```
> yarn add express mysql
```

database (sql)

```
> create database mern_db;
> use mern_db;
> Create table Product(
                id integer primary key auto_increment,
                title varchar(100),
                description varchar(1000),
                price float,
                );
> Select id, title, description, price from product;
> insert into product (title, description, price) values
        ('product-1', 'quality is good', 1000);
```

⊙ server.js

```
const express = require('express')
const app = express()

app.listen(4000, '0.0.0.0', () => {
        console.log('server started on port 4000')
})

const productRouter = require('./routes/product')
app.use(productRouter)

// add middleware after creating app for request's body
app.use(express.json())
```

New Directory for Routes

product.js

```
Const express = require('express')
Const router = express.Router()
Const db = require('../db') // import database connectivity
router.get('/product', (request, response) => {
// statement to get all product
Const statement = 'select id, title, description, price
                   from product;'
//response.send('GET/product')
// get the connection → Const connection = db.connect()
                       connection.query(statement, (error, products) =>
// Execute the query  {
                       // close connection
* This is Ansynchro.    connection.end()
  function             if(error){ console.log(error) }
                       else{ response.send(products) }
                       })
})
// Product data will be sent in terms of string & then will get converted into json on client side
// export Router
module.exports = router
```

Mysql Module (Not in Routes directory)

(db.js)

```
// import mysql
const mysql = require('mysql')

Function connect() {
    // create connection to the database
    const connection = mysql.createConnection({
        host: 'localhost',
        user: 'root',
        password: 'manager',
        database: 'mern-db',
        port: 3306,
    })
    // open the connection
    connection.connect()

    return connection
}

// export the Function for other modules
module.exports = {
    connect: connect
}
```

product.js

```
const express = require('express')
const db = require('../db')

const router = express.Router()
// Read Product Details From Product table / Database
router.get('/product', (request, response) => {

    const statement = " Select id, title, description,
                        price from product; "

    const connection = db.connect()

    connection.query ( statement, (error, products) =>
        {
            connection.end()
            if (error) {
                response.send(error)
            } else {
                response.send (products)
                console.log (products)
            }
        })

})
```

// Update a Product in the DataBase

③ router.put('/product/:id', (request, response) =>
{
↳ // to know which product we want to update

```
const { id } = request.params

const { title, description, price } = request.body

const statement = `UPDATE product
                   SET
                   title='${title}',
                   description='${description}',
                   price='${price}'
                   WHERE id = ${id};
                   `

const connection = db.connect()

connection.query(statement, (error, result) =>
{
    connection.end()

    if(error){
        response.send(error)
    }else{
        response.send(result)
    }
})
```

});

* NOTE : Take care of
1) path / : etc
2) commas ' '
3) SQL query
4) send
5) Back quotes in statement

// Delete a Product from DataBase

④
```
router.delete('/product/:id', (request, response) =>
{
    const {id} = request.params

    const statement = `
                        DELETE from product
                        WHERE id = ${id};
                      `

    const connection = db.connect()

    connection.query(statement, (error, result) =>
    {
        connection.end()

        if (error) {
                response.send(error)
            }
        else {
                response.send(result)
            }
    })
```

③

NOTE :

In postman URL :

DELETE http://localhost:4000/product/

eg. # Develop Application → For NOTES

TABLES

① USER
- id userid (primary key)
- email
- name
- password

② NOTES
- id
- content
- title
- userid (Foreign key)
- date

BACKEND

User's (functionality)
- signUp → Register user
- signIn → Log In

Note's (functionality)
- add note
- delete note
- delete all notes
- update note
- get all Notes

Database

```
> create database notes_db;
> use notes_db;

> create table user (
        id integer primary key auto_increment,
        firstName varchar(20),
        lastName varchar(20),
        email varchar(50),
        password varchar(50),
        createdTimestamp timestamp default
                                CURRENT_TIMESTAMP
    );
```

```
> Create table note (
        id integer Primary key auto_increment,
        title varchar(20),
        content varchar(1000),
        userId integer,
        createdTimeStamp timestamp defaut
            (date)                    CURRENT_TIMESTAMP,
    );
```

⊙ Install express & mysql on Terminal

```
> yarn add express mysql
```

⊙ New directory → Routes → user.js
                          → note.js

// NOTE*: create database carefully on MySql
check spelling errors*

server.js

```
const express = require('express')
const app = express()

const userRouter = require('./routes/user')

app.use(userRouter)     // app.use('/user', userRouter)

app.use(express.json()) // middleware to get body

const noteRouter = require('./routes/note')

app.use(noteRouter) // app.use('/note', noteRouter)

app.listen(4000, '0.0.0.0', () => {
    console.log('server started on port 4000')
})
```

```
const CryptoJS = require('crypto-js')

# Routes → user.js (signup)

const express = require ('express')
const router = express.Router()
const db = require ('../db')

router.post('/user/signup', (request, response)
  => {

const { firstName, lastName, email,
    password } = request.body
const encryptedpassword = String (cryptoJs.MD5(password))
const connection = db.connect()

const statement = `
        INSERT INTO user
        VALUES
        (firstName, lastName, email, Password)
        VALUES
        ('${firstName}', '${lastName}', '${email}',
          '${password}')   → visible password instead
          `${encryptedPassword}`   use encrypted
connection.query (statement, (error, result) =>
  {
    connection.end()

    if (error){
        response.send (error)
      }
    else {
        response.send (result)
      }
  })                    */ problem in this is in
                        database password is
                        visible */
})

module.exports = router
```

* GET is NOT associated with body request
* you can NOT send body along GET request
* you have to have POST request
∴ SignUP & SignIn POST request

⊙ For sign In I want to send id & password through body
∴ method will NOT be GET and it will be POST

⊙ password will be visible to database admin, this is security concern, can be cured by encrypted password → Hashing password

⊙ crypto.js package
> npm install crypto-js
> yarn add crypto-js

⊙ what changes to be done in code?
1) add crypto-js module / package
const cryptoJs = require('crypto-js')

2) const encryptedPassword = String(cryptoJs.MD5(Password

3) Pass '${'encryptedPassword}' in sql statement

⊙ changes in Signup Module

```
const statement = `INSERT INTO user
                  (fileName, lastName, email, password)
// used to prevent  VALUES (?, ?, ?, ?)
sql injection
    // use question marks as placeholder

    db.pool.query (statement,
// Array of all values   [firstName, lastName, email,
(replacements of ?)                encryptedPassword],
                  (error, result) =>
    {
        if (error) {
                  response.send(error)
                } else {
                  response.send(result)
                }
    })
})
```

db.js (For Database/ mysql connectivity)

```
const mysql = require('mysql')

Function Connect() {

  const Connection = mysql.createConnection({
        host: 'localhost',
        user: 'root',
      Password: 'manager',
      database: 'notes_db',
        Port : 3306
  })

  Connection.Connect()

  return Connection

}

module.exports = {

      Connect,

}

// module.exports = {
    Connect: Connect,
}
```

// In JS when you are creating an object if you know key & value both of them are same, then No need to say connect: connect you can just say Connect

Another way to opening & closing connection

① mysql → Pooling connections → create / manage connections

```
function connect() {
    const connection = mysql.createPool({
        host: 'localhost',
        user: 'root',
        password: 'manager',
        database: 'notes-d6',
        connectionLimit: 10
    })
    return pool
}
module.exports = {
    connect,
}
```

// no. connections opened at a time

using this way we do not need to end connection everytime

OR (Instead writing function)

// Refer this code for Day 30 may codes

```
const pool = mysql.createPool({
    host: 'localhost',
    user: 'root',
    password: 'manager',
    database: 'notes-d6',
    port: 3306,
    connectionLimit: 20
})

module.exports = {
    pool,
}
```

Routes → note.js

```
const express = require('express')
const db = require('../db')
const router = express.Router()

// get all the notes of user having userId

router.get('/note/:userId', (request, response) =>
{
    const { userId } = request.params

    const statement = `SELECT
                        id, title, content, date
                        FROM note
                        WHERE
                            userId = ${userId}
                        `

    const connection = db.connect()

    connection.query(statement, (error, notes) =>
    {

        connection.end()
        if (error) {
            response.send(error)
        } else {
            response.send(notes)
        }

    })

})

module.exports = router
```

```
// Add Note For a Particular User
router.Post('/note/', (request, response) =>
{

    const {title, content, userId} = request.body;

    const statement = `INSERT into note
                        (title, content, userId)
                       VALUES
                       ('${title}', '${content}',
                            '${userId}')
                       `

    const connection = db.connect()

    connection.query(statement, (error, notes) =>
    {
        connection.end()

        if (error) {
                response.send(error)
            } else {
                response.send(notes)
            }

    })

})
```

// Update Note using noteId

```
router.put('/note/:noteId', (request, response) =>
{
    const {noteId} = request.params

const {title, content, userId} = request.body

const statement = `UPDATE note
                   SET
                   title = '${title}',
                   content = '${content}',
                   //userId = '${userId}',
                   WHERE
                   id = ${noteId}
    const connection = db.connect()
connection.query(statement, (error, notes) =>
{
        connection.end()

        if (error) {
            response.send(error)
        } else {
            response.send(notes)
        }

})

})
```

why commented ?
Because ← userid we do not want to change only updating note title & content

```
// Delete the Particular Note with noteId
router.delete('/note/:noteId', (request, response) =>
    {
        Const {noteId} = request.parrams

    Const statement = ` DELETE FROM note
                        Where
                         id = ${noteId}

    Const connection = db.connect()

connection.query (statement , (error, notes) =>
    {
            connection.end()

            if (error) {
                response.send(error)
            } else {
                response.send (notes)
            }

    })

})
```

⊙ Routes: note.js

```
const express = require('express')
const db = require('../db')
const router = express.Router()
const utils = require('../utils')

// Get all notes for particular user
router.get('/:userId', (request, response) =>
{
    const {userId} = request.params

    const statement = `SELECT
                        id, title, content, date
                        FROM note
                        WHERE userId = ?
                        `

    db.pool.query(statement, [userId], (error, notes)
        => {                          ↳ // Array of note records
                                        of particular user
    if (error) {
        response.send(error)
    } else {
        response.send(notes)
    }
```

*Instead:

```
// response.send
(utils.createResult
  (error, notes))
```

```
})

})
```

// create a new Note

No need of userId because we are not taking it from params but taking from Body

```
router.post('/:userId', (request, response) =>
{
    const {title, content, userId} = request.body

    const statement = `INSERT INTO note
                       (title, content, userId)
                       VALUES (?, ?, ?)
                       `

    db.pool.query(statement, [title, content, userId],
                  (error, notes) =>
    {
        if (error) {
            response.send(error)
        } else
        {
            response.send(notes)
        }
    })
        // response.send(
            utils.createResult(error, notes)
})
```

// update a note using particular noteId

```
router.put('/:noteId', (request, response) =>
    {
        const {noteId} = request.params

const {title, content} = request.body

const statement = ` UPDATE note SET
                    title = ?, content = ?
                    WHERE
                     id = ?
                    `

db.pool.query(statement, [title, content, ~~user~~ noteId],
                    (error, notes) =>
{
    if (error) {
        response.send(error)
    }
    else {
        response.send(notes)
    }
})
})

// response.send(utils.createResult
                    (error, note))
```

// Delete a particular Note using particular noteId

```
router.delete('/:noteId', (request, response) =>
{
    const {noteId} = request.params

const statement = `DELETE FROM note
                   WHERE
                     id = ?
`

db.pool.query(statement, [noteId], (error, notes) =>
{
    if (error) {
        response.send(error)
    }
    else {
        response.send(notes)
    }
})
    // response.send(utils.createResult
                     (error, notes))
})

module.exports = router
```

utils.js

```
function createResult (error, data) {
    // create an emtty object of result
    const result = { }

    if (error) {  // something wrong
        result ['status'] = 'error'
        result [ 'error'] = error
    }
    else { // everything good
        result ['status'] = 'success'
        result ['data'] = data
    }

    return result
}

module.exports = {

        createResult,
}
```

⊙ db.js

```
const mysql = require('mysql')

function connect() {
    const connection = mysql.createConnection({
                host: 'localhost',
                user: 'root',
                password: 'manager',
                database: 'notes-db',
                port : 3306

    })
    connection.connect()
    return connection
}
module.exports = { connect, }
```

OR

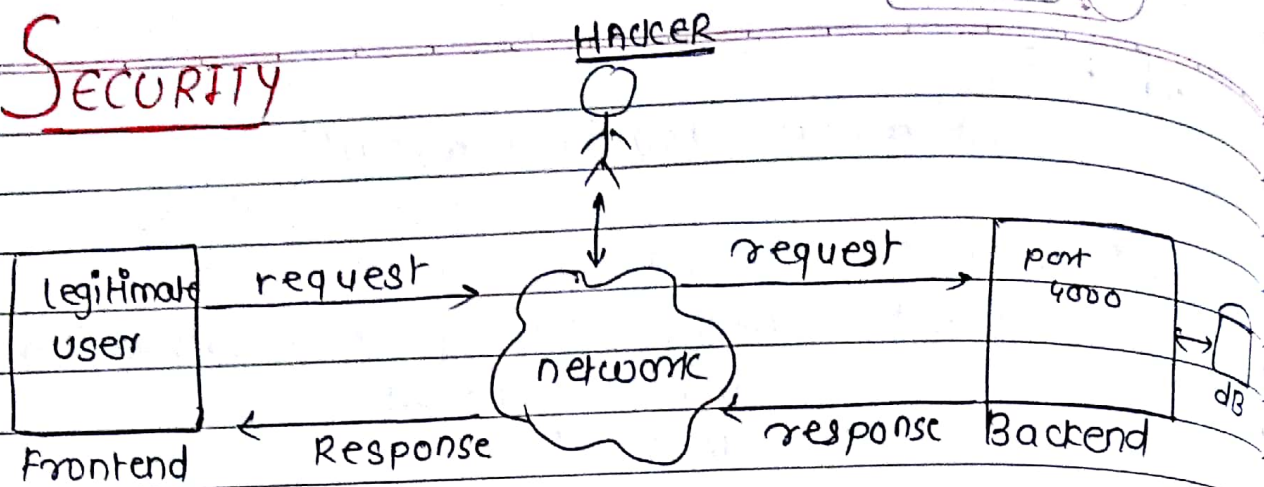
// opening connection & closing connection is not suitable everytime for Real time world scenario, its time consuming

// Search mysqlJS → pool → create pool → it will automatically create or manages all the sessions

```
const mysql = require('mysql')

Function connect() {    // Pool (collection) of connections
const pool  ~~const connection~~ = mysql.createPool({
                host: 'localhost',
                user: 'root',
                password: 'manager',
                database: 'notes-db',
                connectionLimit : 20
    })
    return ~~connection~~ pool
}
module.exports = { connect, }
```

SECURITY



⇒ suppose, user has requested "GET/note/4" from front end it will send to network & from network it will send to Backend/server.

⇒ server will see what kind of user details you are sending and from these user details it will select all the notes from database where userId = 4

⇒ response of server with details of all notes send back to network & henceforth on frontend

⇒ suppose there is a Hacker also connected to network, if user requested and hacker read that request from network, he can access the details with that request from Backend

⇒ From server's point of view there is no way to validate who is sending this query/request (server can not verify the user → legal one/Hacker)

Solution : ① 1st login → email & password (Assume password secret)

② After login, token with signature created (token → sign. of logged in user it is some code) token can be created only by server as server has got secrete used to create token

NOTE'S Application Using Security

user.js

```
const express = require('express')
const cryptoJS = require('crypto-js')
const db = require('../db')
const utils = require('../utils')
const jwt = require('jsonwebtoken')
const config = require('../config')

const router = express.Router()

router.post('/signup', (request, response) =>
{
  const {firstName, lastName, password, email} = request.bod
// encrypt password
const encryptedPassword = String(cryptoJs.MD5(password))
// use ? marks as placeholders to prevent sql injections
 const statement =
`INSERT INTO user (firstName, lastName, email, password)
    VALUES (?, ?, ?, ?)
`
  // Fire the query
db.pool.query(statement,  // Array of all values (at ?)
              [firstName, lastName, email, password]
                encryptedPassword],
    (error, result) =>
  {
    response.send(utils.createResult(error, result))
  })

})
```

```
router.post('/signin', (request, response) =>
{
    const { password, email } = request.body
// encrypt password
const encryptedPassword = String(cryptoJs.MD5(password))

const statement = `SELECT id, firstName, lastName, email
                   FROM user
                   WHERE email = ? AND password = ?
`

db.pool.query(statement, [email, password],
              (error, users) =>
{
    if (error) { result['status'] = 'error'
                 result['error'] = error
    } else { if (users.length === 0)
             { result['status'] = 'error'
               result['error'] = 'user does NOT exist'
             } else { // Authentication is successful
               const user = users[0]
               result['status'] = 'success'
    // create token which will verify user using token's signature
               const token = jwt.sign({ userId : user['id'] },
    // payload (to sent to all remaining API's)   // secret (used to generate token)
                                      config.secret)

    // let's not pass userId from signin so that user won't able to find userId anywhere
               result['data'] = { firstName: user['firstName'],
                                  lastName: user['lastName'],
                                  email: user['email'],
                                  token : token
                                }
             }
    }
    response.send(result)
})
})

module.exports = router
```

⊙ config.js

```
module.exports = {
                    secret: '2m368tHMu26mG0cm',
                 }
```

⊙ utils.js

```
function createResult (error, ~~result~~ data)
  {
      // create an empty result
    const result = { }

  if (error) { //Something has gone wrong
            result ['status'] = 'error'
            result ['error'] = error
          }
      else { // everything is correct
            result ['status'] = 'success'
            result ['data'] = data
          }

    return result
}

module.exports = {
                    createResult,
                 }
```

REACT JS

31 May

① ReactJs → efficient & flexible Javascript library
- for building reusable UI components
- Declarative – declare custom tags
- Flexible – complete or part of ~~fast~~ React in program
- Front end library responsible for view layer of application
- created by Jordan Walke, (Facebook)
- single page application / multi page application

② JSX → Javascript XML, Javascript syntax extension
- its an XML or HTML like syntax used by ReactJs
- embed XML / HTML tags in JS code
- It is not necessary to use JSX, but it is recommended to use in ReactJS

③ Components → React has multiple, reuseable components each component has its own logic & controls

④ One Way Data Binding → Unidirection / one-way data flow or data binding
- It is because components are supposed to be immutable & data within them cannot be changed

⊙ Alternatives to ReactJs (Why you chose ReactJs)

→ Angular VueJs ReactJs

Angular	React
1) google	1) Facebook
2) Misko Hevery	2) Jordan walke
3) 2010	3) 2013
4) Language: Typescript HTML	4) Language: Javascript (JSX)
5) open source MVC framework	5) open source JS library
6) client side Rendering	6) client side or server side
7) Data binding: Bidirectional	7) Data binding: uni directional
8) DOM: Regular DOM	8) DOM: virtual DOM
9) Slow performance	9) Fast
10) Single Page Applicn	10) SPA & MPA

React	React Native
→ used for Web Development	→ used for Mobile Development
→ executed on all platforms	→ executed only on Mobile Platform
→ HTML / CSS for Animation	→ Built-in animation libraries
→ uses React-Rower for navigation	→ built in navigation

⊙ Packages

Developer
- ① database - MongoDB, MySQL / Postgres
- ② platform - Node JS
- ③ client Application
  - → web applicn - React
  - → Mobile applicn - React Native
- ④ Web server (backend) - Express, Fastify

operations
- ⑤ Cloud - aws, azure, gcp
- ~~Docker~~ Containerization - docker
- orchestration - kubernetes, docker swarm
- CI/CD pipeline - Jenkins / Cloud CI/CD

⊙ Write New Application → update Text of div tag

1) open vs code → Folder Name → APP1
2) File Name → Page1.html
3) Folder Name → APP2 (Better Approach Using React)

⊙ search ~~cupackage.com~~ UNPKG.com → content delivery network

use min.js → /react@16 ——— min.js
/react-dom@16 ——— min.js

4) File name: index.html
   - ↳ page1.js
   - ↳ page2.js

page1.js

```
const firstName = 'steve'
// binding → getting value of that variable & putting it inside
const div = <div> welcome {firstName} </div>

const root = document.getElementById('app')
ReactDOM.render(div, root)
```

element or tags

index.html

```
<html>
<head>

</head>
<body>
    <div id='app'>      </div>
<script src = "——— unpkg.com/react@16 ——— "></script>
<script src = "——— unpkg.com/react-dom ——— "></script>
```

page2.js

```
const mobile = { model: 'iphone 13 pro',
                 company: 'Apple',
                 price: 179000,
               } // Binding Mobile object with an element
const div = <div>
              <div> Model: {mobile.model} </div>
              <div> company: {mobile.company} </div>
              <div> price: {mobile.price} </div>
            </div>
const root = document.getElementById('app')
ReactDOM.render(div, root)
```

// command to execute code :

```
>npx babel page2.js --out-File public/script.js --presets=
                                    react --watch
```

page3.js

```
const countries = ['India', 'USA', 'UK', 'Japan']
// Bind an array with an element
```

manual way of indexing

```
const div = <div>
<div> {countries[0]} </div>
<div> {countries[1]} </div>
<div> {countries[2]} </div>
<div> {countries[3]} </div>
</div>
```

```
const root = document.getElementById('app')
ReactDOM.render(div, root)
```

o/p: India, USA, UK, Japan

If I will add new value in array I have to write new For binding

```
const countries = ['India', 'USA',
  'UK', 'Japan', 'France']
// Binding Array elements using
  MAP func^n

const div2 = <div>
             {countries.map
             (country => {
return <div> {country} </div>
})}

const root = ————————
ReactDOM.render ————————
```

// Another way (Binding Array with unordered list)

```
const ul=(<ul> {countries.map((country)=>{
        return <li> {country} </li>
    })}
</ul>
)
```

// Another way (Binding an Array with table)

```
const table = ( <table border='1'>
        <thead> <tr>
            <th> Name </th>
        </tr>
    </thead>
    <tbody>
    { countries.map((country) =>
        { return (
            <tr>
            <td> {country} </td>
            </tr>
        )
    })
}
    </tbody>
    <tFoot> </tFoot>
    </table>
)

const root = ____
reactDOM.render ____
```

Owpw:

Name
India
USA
UK
JAPAN
France
Nepal

① page4.js

```
const notes = [
    { id: 1, title: 'note-1', content: 'this is note1' },
    {                                                  },
    {       // copy-past & add multiple elements       },
    {                  in an array as you wish         },
]

const div1 = <div> {notes.map( note => {
                    return <div> id : {note.id}
                           </div>
                    <div> title : {note.title} </div>
                    <div> content : {note.content}
                    </div> <hr>
                    </div> )
                })}

const root = ——————————————
reactDOM.render (div1, root)
// Another way : (Binding array with table)
const table = ( <table border='1' >
                <thead> <tr> <th> id </th>
                             <th> title </th>
                             <th> content </th>
                </tr>
                </thead>
                <tbody> {notes.map (note) => {
                return ( <tr>  <td> {note.id} </td>
                               <td> {note. title} </td>
                               <td> {note.content} </td>
                </tr>
                )
                })}
                </tbody>
                </table>
```

id	title	content
1	note-1	
2	note-2	
3		
4		
5		

⊙ Create React APP

>npm install -g create-react-app

>npx create-react-app demo

⊙ For Airbnb website

Backend : server : express
database : mysql

> create database airbnb_db;

> use airbnb_db;

-- Tables
user, home, booking

>create table user (
id integer primary key auto_increment,
firstName varchar(20),
lastName varchar(20),
email varchar(20), password varchar(100),
birthDate date, ProfileImage varchar(100),
createdTimestamp date DEFAULT,
CURRENT_TIMESTAMP
);

Types of house
1: flat, 2: house,
3: ---, 4: --
5: --

> create table home(
id integer primary key auto_increment,
type integer,
addressline1 varchar(50),
addressline2 varchar(50),
city varchar(50),
state varchar(50),
pincode varchar(6),
guests integer, bedroom integer, bathroom integer,
);

Frontend : library - react

⇒ functionality → user → signup
→ signIn
→ forgetPassword
→ ResetPassword

→ home → host home
→ search home
→ add/remove wishlist
→ book home
→ cancel booking

state variables { const [password, setPassword] = useState('')

① login.js

const login = () => { let email = ' ' let password = ' '

write progm. logic before return

return (

<h1 style={{ textAlign: 'center' }}> Login </h1>

~~ <div className=~~

margin bottom

<label> Email </label>

// Event key Handler

onKeyUp = {(event) => {

setEmail(event.target.value)

}}

<input className = 'form-control'

type = 'email' > </input>

email:

<label> password </label>

password :

~~<button~~ <input onKeyUp={ } >

<button onClick = {login}

className = "btn btn-success" > Login

Login Cancel

<button onClick = {cancel} ⟶

cancel </button>

I want to take i/p from user email & password but this has to be changed so you have to add it in state.

onKeyUp = {(event) => {

setPassword (event.target.value)

}}

(X): let email = ' ' is wrong here

Airbnb Project

① How to start project ?

① wire frame → design / Requirement of project
gives idea about what client is looking for
↳ concepts, Functionality

② frontend → UI

③ Backend → server + database

eg. We are developing clone of Airbnb project
Refer earlier pages for database, etc.

src → + components (create Folder)
↳ + pages (create Folder)

src → contains source code of application

components
↳ ~~Signup~~
↳ ~~SignIn~~
↳ navbar

pages
↳ signup.js
↳ signin.js
↳ Reset password.js
↳ forgot password
↳ User
↳ House
↳ home.js
↳ searchHomes.js
↳ homeDetails.js
↳ hostHome.js

services
↳ services
used to connect frontend to Backend
↳ App.js