Node JS

→ Backend development framework
→ asynchronous, event-driven

eg
```
var generateName = require
                   ("sillyname");

var sillyName = generateName();
console.log (sillyName);
```

→ Express framework
  ↳ ① readibility
    ② less code
    ③ middleware.

express interacts with node

eg
```
import express from "express"
const app = express();
app.listen (3000, () => {
    console.log("Server running on
                 port 3000");
});
```

→ localhost → our own computer

→ port → no. of doors in our computer
  when we specify the port, control searches the port and uses it

netstat -ano | findstr "LISTENING"

→ HTTP → hyper text transfer protocol.
  computers communicate each other.

GET → resource request from server.

POST → resource sending to server.

PUT → replace entire resource with our data.

PATCH → replace a part of resource with our data.

DELETE → delete resource

→ req - request
  res - response

→ nodemon → removes need to stop and restart the server continuously

eg npm install -g nodemon
  -g to install a package globally

→ Status messages.
  info (100 - 199) "hold on"
  success (200 - 299) "here you go" you fetched it
  redirect (300 - 399) "go away"
  client error (400 - 499) "you fucked up"
  server error (500 - 599) "I fucked up"

→ Middleware
  ↳ Server → middleware ⇄ post, delete, get, put, patch

(post request)
body-parser middleware
[app.use(bodyParser.urlencoded
({extended: true}));]

→
```
<form action='/login'
      method = 'POST'>
  <input type='text'
         name="name"
         required>
  <input type=submit'
         value='submit'>
</form>
```

→ public folder
  ↳ contains files not to be edited and are global to code

→ morgan middleware
↳ decides how detailed logging has to be

```
app.use((req, res, next) => {
    console.log("Request method:",
                req.method);
    next(); [decides who to go next fn that is in line]
});
function logger (req, res, next){
  console.log("Requested url:", req.url);
  console.log("Requested method",
              req.method);
  next();
});
```

→ EJS (embedded JS)
↳ helps to embed JS in html file.
extension '.ejs'

eg
```
app.post("/submit", (req, res) => {
    res.render("index.ejs", {
        name: req.body['name']
    });
});
```

```
<body>
  <h1> Hello <%= name %></h1>
</body>
```

- `<%= variable %>`
- `<% console.log("hello") %>` → executable JS code.
- `<% <h1>Hello</h1> %>`
- `<% % % %>` → shows ejs tags.
- `<% // This is a comment %>`
- `<%- include("header.ejs") %>`

eg
```
let bowl = ["Apples", "Oranges", "Pears"]
app.get("/", (req, res) => {
    res.render("index.ejs", {fruits: bowl});
});
```

```
<UL>
  <% for (let i = 0 ; i < fruits.length ; i++) { %>
    <LI> <%= fruits[i] %> </LI>
  <% } %>
</UL>
```

→ locals → keyword to check for existence passed to ejs from .js

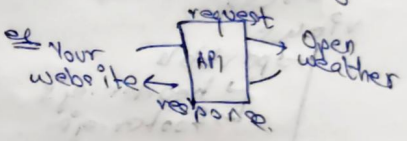
→ HTML no ejs layouts, template

```
app.use(express.static('public'));
```

public → all static files.
- image, css → static files
- html, javascript → dynamic files.

→ Partials.
`<%- include('header.ejs') %>`
`<%- include('footer.ejs') %>`

→ API
↳ helps in communication of two diff programs



→ API endpts
↳ baseurl/endpt

eg
bored-api.appbrewery.com/endpoint?query1=value&query2=value&query3=value

→ JSON vs JS object

1) {
"name": "Jack Bauer",
"age": 48,
"city": "Santa Monica",
}
2) sent across internet. difficult to read.

1) const jack = {
name: "Jack Bauer",
age: 48,
city: "Santa Monica",
}
2) more relaxed notation

→ JSON → JS object

```
const data = JSON.parse(
                jsonData);
```

JS object → JSON

```
const jsonData = JSON.
        stringify(data);
```

→ import https from "https";

```
app.get("/", (req,res) => {
  const options = {
    hostname: "bored-api.
        appbrewery.com",
    path: "/random",
    method: "GET",
  };
```

→ tree

```
node_modules
public
  ↳images
  ↳css
views
  ↳partials
      ↳header.ejs
      ↳footer.ejs
  ↳about.ejs
  ↳contact.ejs
  ↳index.ejs
```

→ some index.js code

```
import express from "express";
const app = express();
const port = 3000;
app.use(express.static("public"));
app.get("/",(req,res) => {
  res.render("index.ejs");
});
app.get("/about",(req,res) =>{
  res.render("about.ejs");
});
app.get("/contact",(req,res) =>{
  res.render("contact.ejs");
});
app.listen(port, () => {
  console.log("Server running on
        port 3000");
});
```

→ Axios → JS library to make HTTP requests from a browser or Node.js It is isomorphic that is it can run in browser and Node.js with same codebase

```
import express from "express";
import bodyParser from
        "body-parser";
import axios from "axios";

const app = express();
const port = 3000;
app.use(express.static("public"))
app.use(bodyParser.urlencoded
        ({extended: true}));

app.get("/", async(req,res)=>{
  try{
    const response = await
      axios.get("https://
        bored-api.appbrewery.com/
          random");
    const result = response.data;
    console.log(result);
    res.render("solution.ejs",
        {data: result});

  }
  catch(error){
    console.log("error");
    res.render("solution.ejs",
        {error: error.
          response});
  }
});
app.post("/", async(req,res)=>{
  try{
    console.log(req.body);
    const type = req.body.type;
    const participants = req.body.
        participants;
    const response = await
      axios.get("https://
        bored-api.appbrewery.com/
        filter?type=${type}&
        participants=${participants}
        ");
    const result = response.data;
    res.render
```

```
res.render("solution.ejs", {
    data: result[Math.floor(
        Math.random()*
            result.length)],
});

}
catch(error){
    res.render("solution.ejs", {
        error: error.message});
}
});
app.listen(port, () => {
    console.log("Server running
        on port 3000")
});
```

→ REST → representative state transfer.

rules

1) HTTP → (GET, POST, PUT, PATCH, DELETE)
2) JSON → API must respond in JSON format
3) Client-server must be completely different
4) Statelessness → each req from client to server must contain all info for processing request.

→ API authentication

Level	Type
0	No
1	Basic
2	API key
3	Token

No Authentication
↳ API puts a limit on no. of requests in a paticular time

Basic Authentication
↳ provides username, password while GET request

username: password
authentication : xxxxxxxx

→ authorization vs authentication
client allowed to use your API with API key
users can authenticate with username and password.

→ Token based
↳ once login is done, a token is generated which is used for authentication.

```
import axios from "axios";
app.get("/", async(req,res)=>{
    try{
    await axios.get("URL",
                    config);
    res.sendStatus(200);
    }
    catch(error){
    res.status(400).send
            (error.response.
                data);
    }
});
```

for get delete

await axios ("URL", body, config);

put, post, patch