NodeJS 101

•••

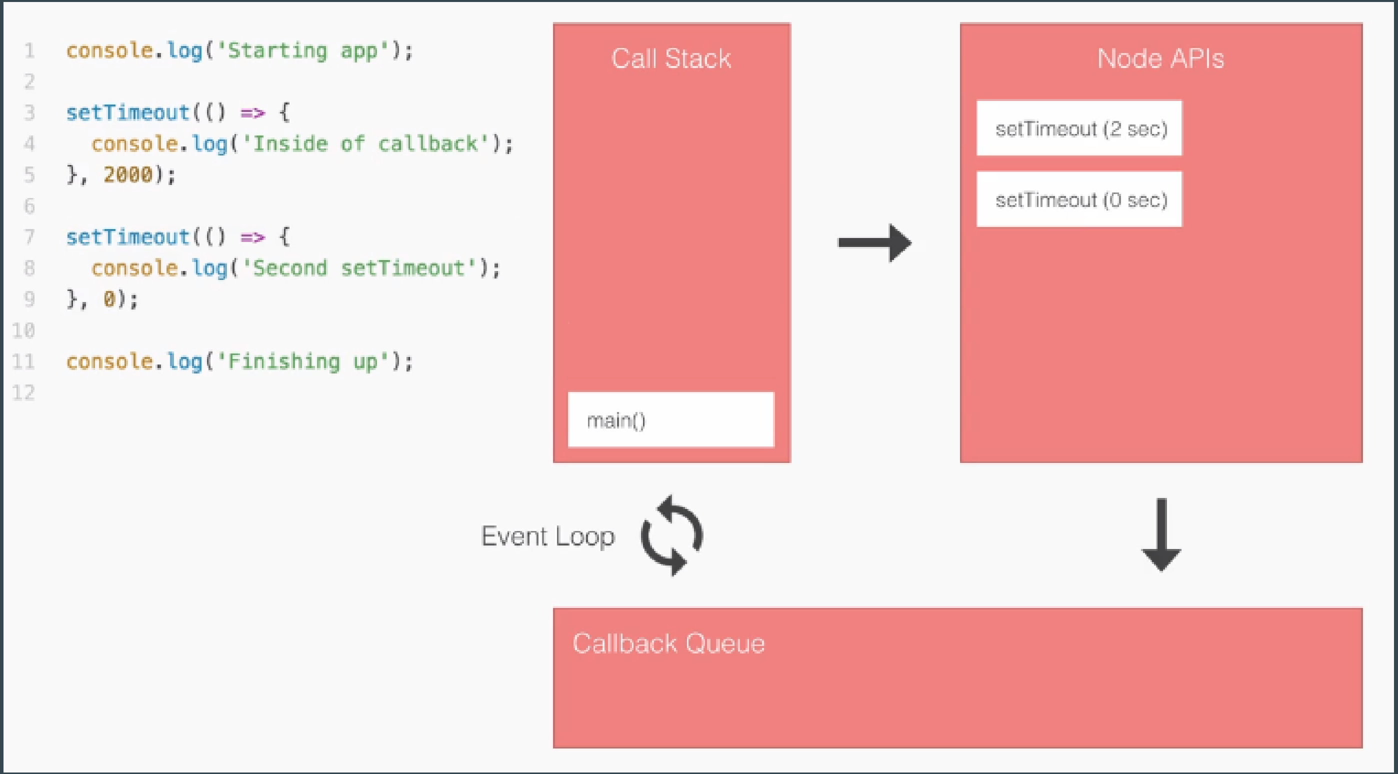
Intro - Installation - NPM - Server Setup - Express - PM2 - Sessions - Templating - Debugging - Best Practices

What is NodeJS ?

Standard Google Definition :

- Node.js® is a JavaScript runtime built on Chrome’s V8 JavaScript engine.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

How JS Event Loop Works ?



```
console.log('Starting app');

setTimeout(() => {
  console.log('Inside of callback');
}, 2000);

setTimeout(() => {
  console.log('Second setTimeout');
}, 0);

console.log('Finishing up');
```

Explanation

1. Push main() onto the call stack.

2. Push console.log() onto the call stack. This then runs right away and gets popped.

3. Push setTimeout(2000) onto the stack. setTimeout(2000) is a Node API. When we call it, we register the event-callback pair. The event will wait 2000 milliseconds, then callback is the function.

4. After registering it in the APIs, setTimeout(2000) gets popped from the call stack.

5. Now the second setTimeout(0) gets registered in the same way. We now have two Node APIs waiting to execute.

6. After waiting for 0 seconds, setTimeout(0) gets moved to the callback queue, and the same thing happens with setTimeout(2000).

7. In the callback queue, the functions wait for the call stack to be empty, because only one statement can execute a time. This is taken care of by the event loop.

8. The last console.log() runs, and the main() gets popped from the call stack.

9. The event loop sees that the call stack is empty and the callback queue is not empty. So it moves the callbacks (in a first-in-first-out order) to the call stack for execution.

Installation

- Visit the Official Node Link to grab the installer for NodeJS:

  https://nodejs.org/en/ - (install v10.0.0 +)

- Check node version in terminal/cmd : node -v

Why NodeJS over other programming Languages ?

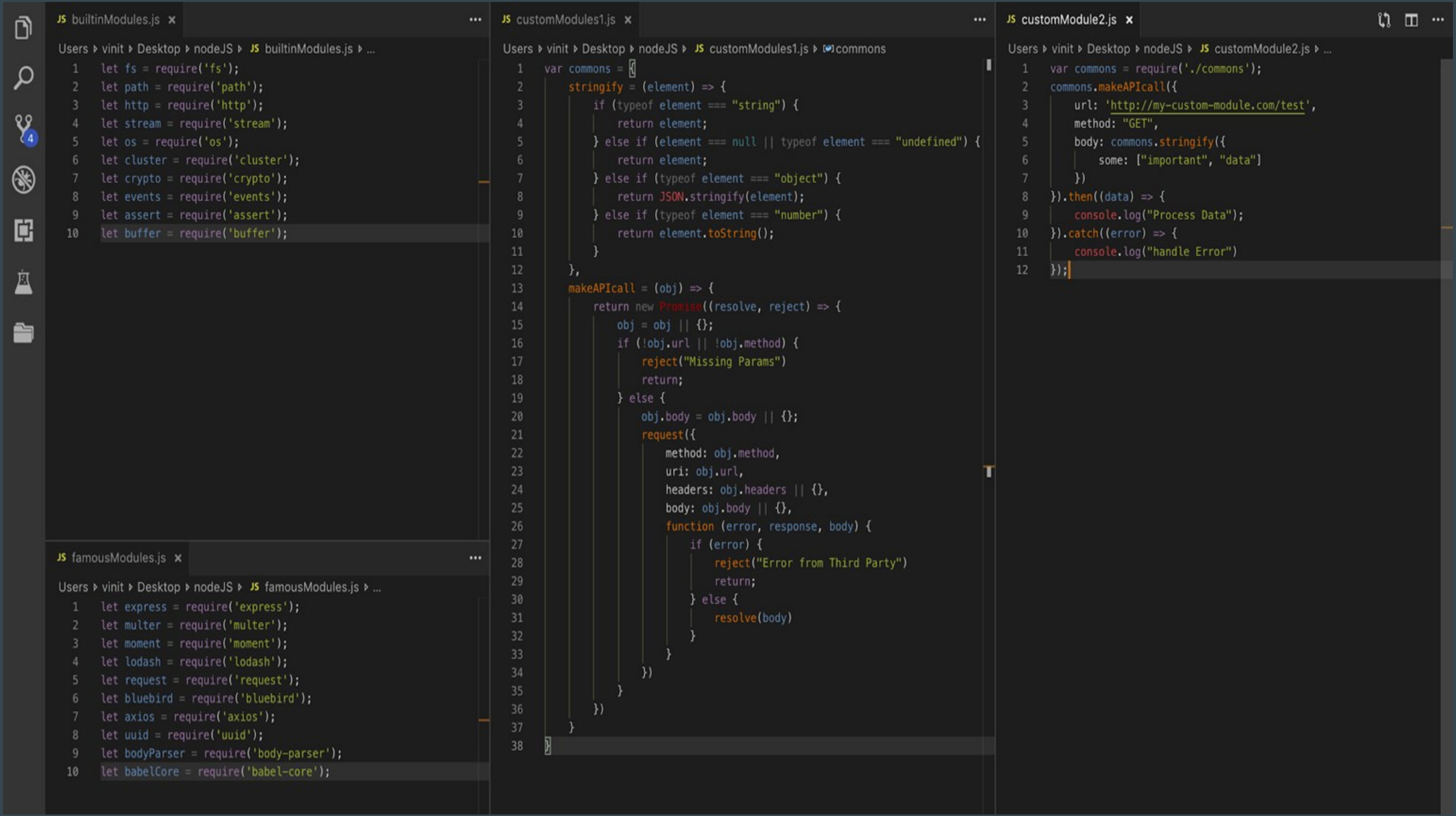
- It’s your Homework.

NPM - Node Package Manager

- Adapt packages of code for your apps, or incorporate packages as they are.
- Download standalone tools you can use right away.
- Run packages without downloading using npx.
- Share code with any npm user, anywhere.
- Restrict code to specific developers.
- Create Orgs (organizations) to coordinate package maintenance, coding, and developers.
- Form virtual teams by using Orgs.
- Manage multiple versions of code and code dependencies.
- Update applications easily when underlying code is updated.

Modules

- NodeJS Built In Modules - (Installed along with NPM installation)
    - https://nodejs.org/api/
- Public Modules
    - Modules developed by other developers and freely available for use on NPM repository.
- Custom Modules
    - Modules / Packages / Applications developed by you.


builtinModules.js
Users › vinit › Desktop › nodeJS › builtinModules.js › ...
customModules1.js
Users › vinit › Desktop › nodeJS › customModules1.js › commons
customModule2.js
Users › vinit › Desktop › nodeJS › customModule2.js › ...
famousModules.js
Users › vinit › Desktop › nodeJS › famousModules.js › ...

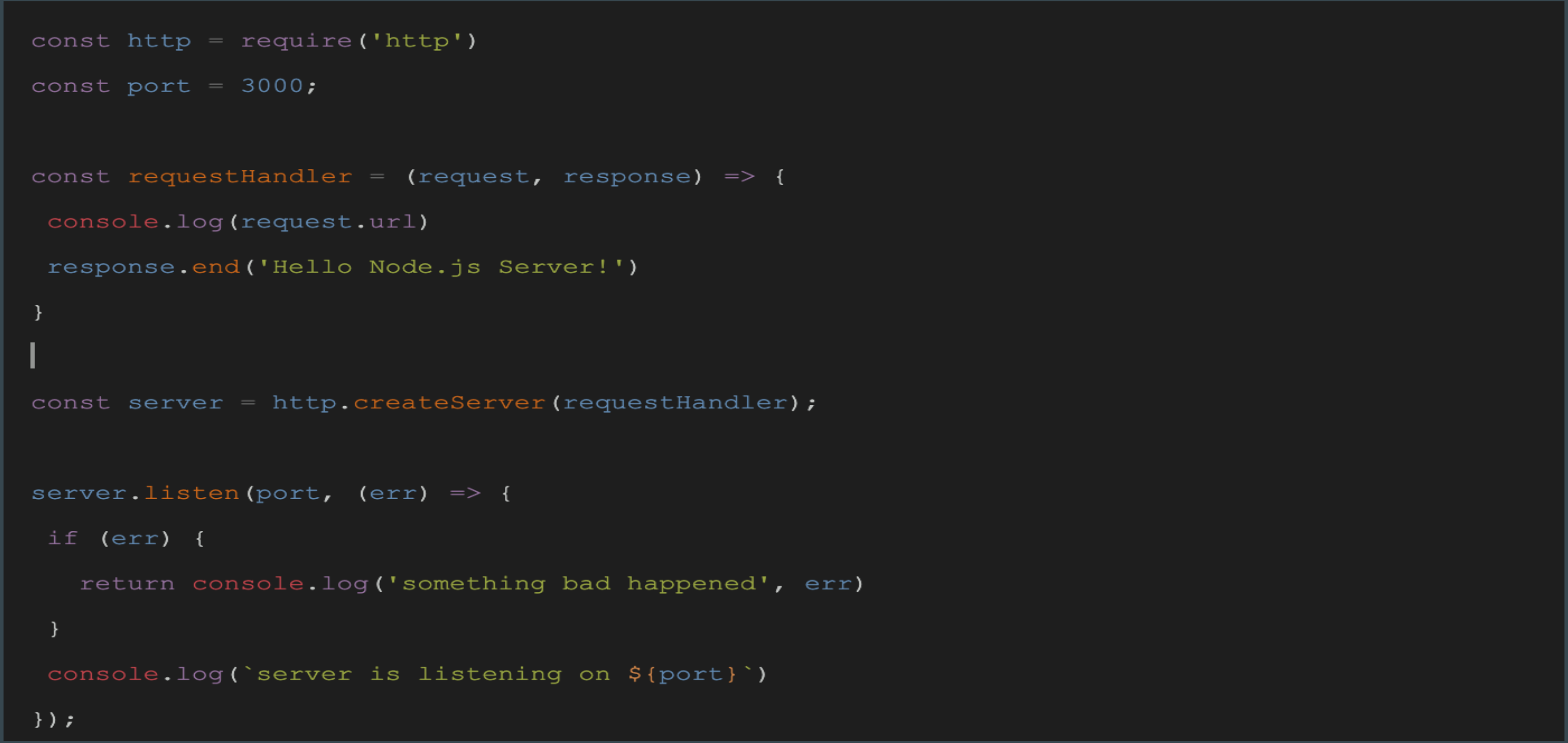
```js
let fs = require('fs');
let path = require('path');
let http = require('http');
let stream = require('stream');
let os = require('os');
let cluster = require('cluster');
let crypto = require('crypto');
let events = require('events');
let assert = require('assert');
let buffer = require('buffer');
```

```js
let express = require('express');
let multer = require('multer');
let moment = require('moment');
let lodash = require('lodash');
let request = require('request');
let bluebird = require('bluebird');
let axios = require('axios');
let uuid = require('uuid');
let bodyParser = require('body-parser');
let babelCore = require('babel-core');
```

```js
var commons = {
    stringify = (element) => {
        if (typeof element === "string") {
            return element;
        } else if (element === null || typeof element === "undefined") {
            return element;
        } else if (typeof element === "object") {
            return JSON.stringify(element);
        } else if (typeof element === "number") {
            return element.toString();
        }
    },
    makeAPIcall = (obj) => {
        return new Promise((resolve, reject) => {
            obj = obj || {};
            if (!obj.url || !obj.method) {
                reject("Missing Params")
                return;
            } else {
                obj.body = obj.body || {};
                request({
                    method: obj.method,
                    uri: obj.url,
                    headers: obj.headers || {},
                    body: obj.body || {},
                    function (error, response, body) {
                        if (error) {
                            reject("Error from Third Party")
                            return;
                        } else {
                            resolve(body)
                        }
                    }
                })
            }
        })
    }
}
```

```js
var commons = require('./commons');
commons.makeAPIcall({
    url: 'http://my-custom-module.com/test',
    method: "GET",
    body: commons.stringify({
        some: ["important", "data"]
    })
}).then((data) => {
    console.log("Process Data");
}).catch((error) => {
    console.log("handle Error")
});
```

Create a Node Server



```
const http = require('http')
const port = 3000;


const requestHandler = (request, response) => {
 console.log(request.url)
 response.end('Hello Node.js Server!')
}


const server = http.createServer(requestHandler);


server.listen(port, (err) => {
 if (err) {
   return console.log('something bad happened', err)
 }
 console.log(`server is listening on ${port}`)
});
```

Some more important points

- process keyword in node.
  - The process object is a global that provides information about, and control over, the current Node.js process
  - console.log(**process**);   // process module
  - console.log(**process.argv**); // command Line arguments
  - console.log(**process.env**); // Environment Variables

Express & its few Amazing features

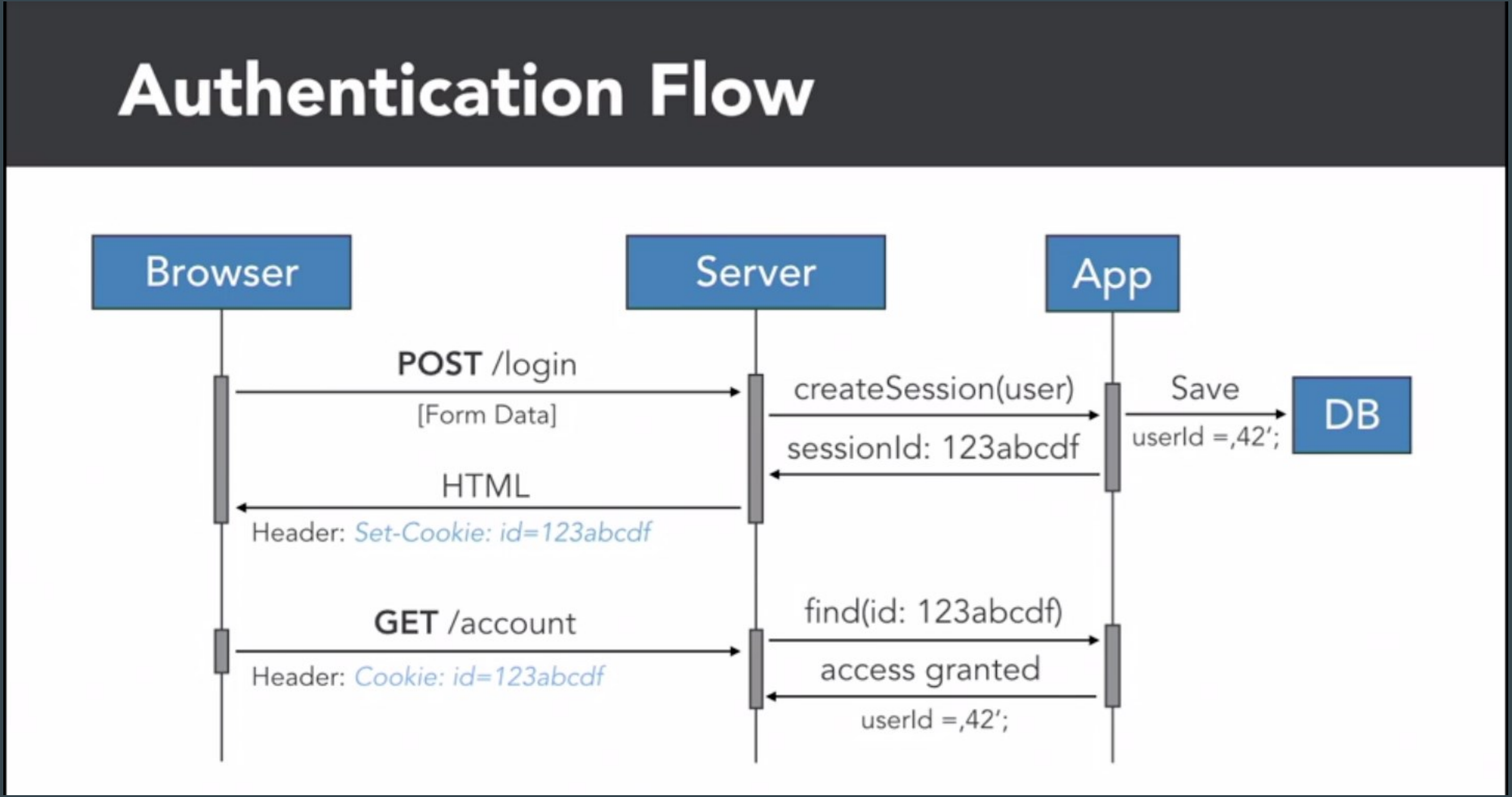
- Express is a minimal and flexible Node.js web application **framework**. Simply . . .
- Routing
- Middlewares
- Templating
- Error Handling

- Guide Section in https://expressjs.com/

Running the Application

- Forever
- PM2
  - pm2 start app.js
  - pm2 stop app.js
  - pm2 logs
  - pm2 status
  - pm2 delete app.js
  - pm2 kill
  - pm2 start app.js --watch
- Nodemon

Sessions & Protected Routes

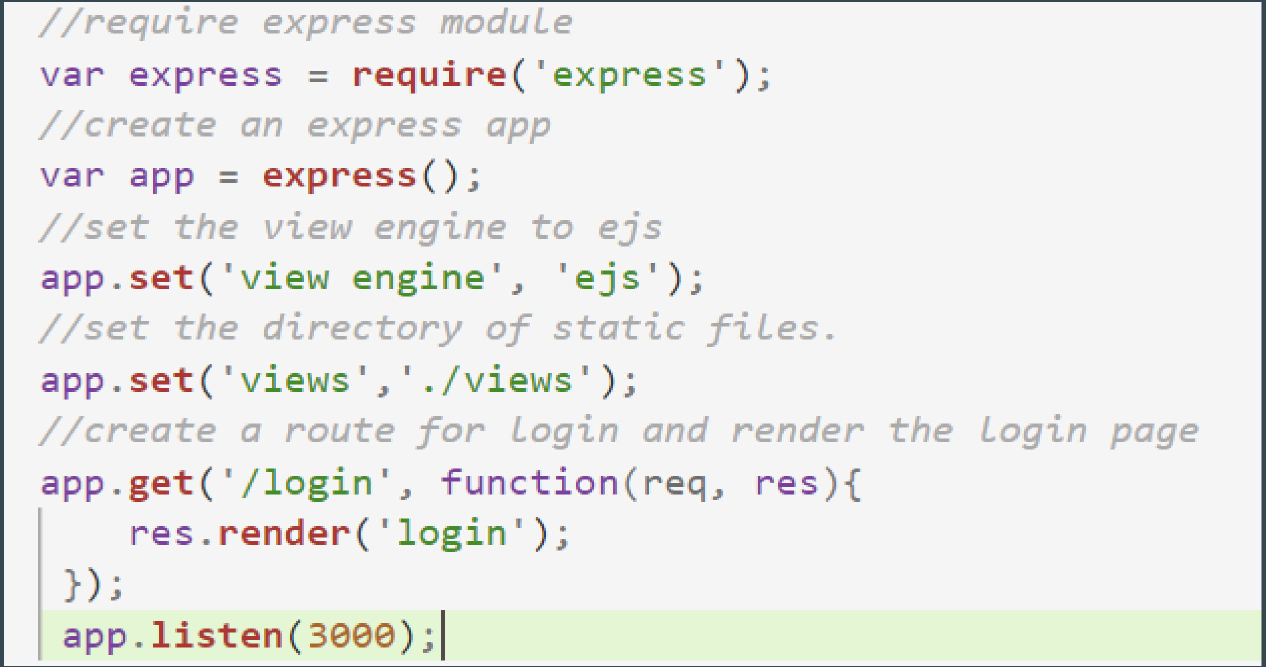


Example

```
var express = require('express')
var parseurl = require('parseurl')
var session = require('express-session')

var app = express()

app.use(session({
  secret: 'keyboard cat',
  resave: false,
  saveUninitialized: true
}))

app.use(function (req, res, next) {
  if (!req.session.views) {
    req.session.views = {}
  }

  // get the url pathname
  var pathname = parseurl(req).pathname

  // count the views
  req.session.views[pathname] = (req.session.views[pathname] || 0) + 1

  next()
})

app.get('/foo', function (req, res, next) {
  res.send('you viewed this page ' + req.session.views['/foo'] + ' times')
})

app.get('/bar', function (req, res, next) {
  res.send('you viewed this page ' + req.session.views['/bar'] + ' times')
})
```

Templating

- Mustache
- Handlebars
- EJS
- Underscore
- Pug



```
//require express module
var express = require('express');
//create an express app
var app = express();
//set the view engine to ejs
app.set('view engine', 'ejs');
//set the directory of static files.
app.set('views','./views');
//create a route for login and render the login page
app.get('/login', function(req, res){
    res.render('login');
 });
 app.listen(3000);
```

Debugging

- console.log(“I am here”);
- console.log(“I am here too”);

- Make use of Network Section in Browser inspector

- npm‘s **node-inspect**
- Allows you to debug on browser similar to a web-page using chrome inspector

Best Practices

- Follow a fixed folder structure.
- Make your Code as modularized as possible - (more readable / reusable)
- Use latest available JS features - es6 / es7
- Keep committing and pushing code changes into your GitRepos.
- Error Handling is the **Most Valuable Feature** in a project.
- Unit Test every functionality / API
- Make use of HTTP Client like **Postman** to test your API.
- Refer/follow Open source GitHub Repos for reference.
- Code Everyday ;)

Thank You !

Sample Demo files can be found here :

https://drive.google.com/open?id=1udU2sLxSzek1e5Ppsk_eX-8EKkphkHsJ