

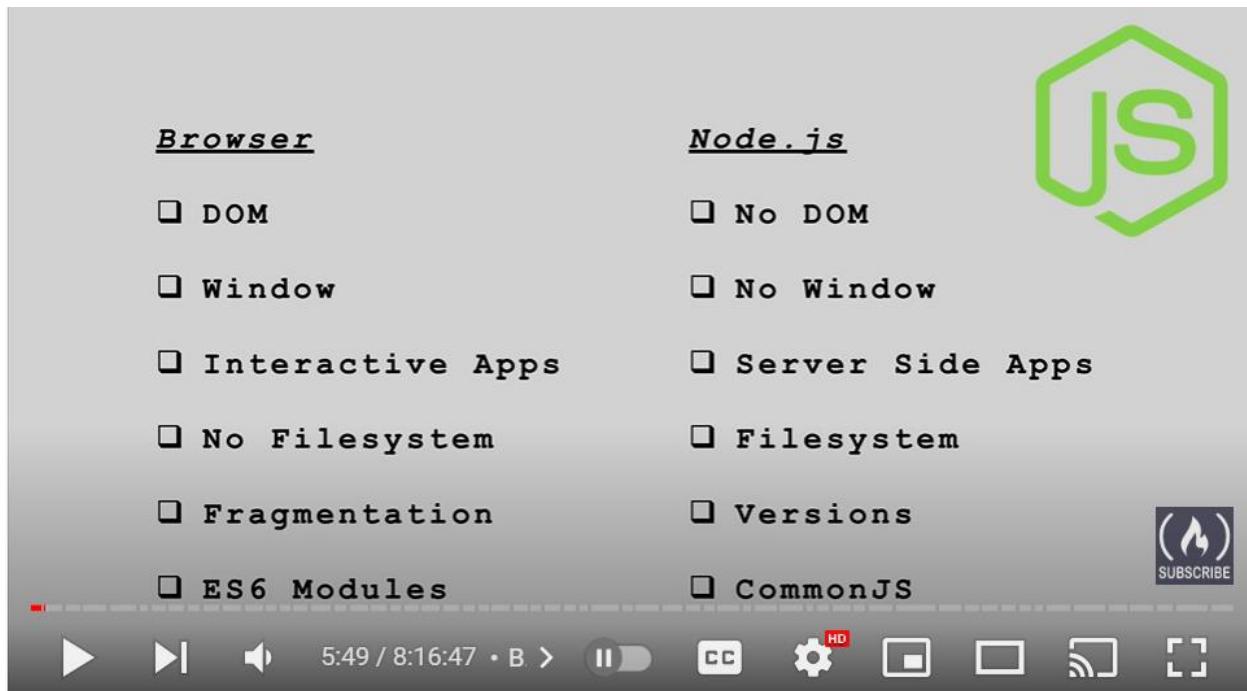
Node JS

Environment to run JS outside the browser.

Created in 2009.

Built on chrome's V8 Engine.

Every browser has an engine, a tool that compiles our code down to machine code and chrome uses it by the name V8.



Node JS uses Common JS library.

For frontend part, browser does most of the part. Our code is compiled, and browser runs our code.

How does node evaluate our code?

We have 2 options

1. REPL – Read Eval Print Loop
2. CLI Executable – running our app code in node

REPL is only for playing around but we mostly use CLI.

REPL can be accessed by opening a terminal and type node and then hit enter. Once you have entered the REPL, type const name = "Sai" and click enter.

Now, you can access the name variable which has been defined as 'Sai'.

You can close the REPL by click on Ctrl + C for two times.

How to run a Node Application in a terminal

Use the command `node <filename>` to run the node app in terminal

Ex: `node app.js`

Ex: `node app` (we can exclude the applications file extension)

The screenshot shows the Visual Studio Code interface. In the center, there is an editor window titled "app.js" containing the following code:

```
const amount = 14;
if (amount > 15) {
  console.log("Large Number");
} else {
  console.log("Small Number");
}
console.log('Hey !! this is my first node application');
```

Below the editor is a terminal window titled "TERMINAL". It shows the output of running the script in PowerShell:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
Large Number
Hey !! this is my first node application
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
Small Number
Hey !! this is my first node application
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>
```

Global Variables in Node

In Vanilla JavaScript, we have access to various global variables like window. With window variable we can have access to many useful functions like querySelector().

There is no window in Node because there is no browser for Node.

Global Variables are nothing but a variable which can be accessed anywhere over the application.

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Terminal:** 2-globals.js - Node JS Tutorial Youtube - Visual Studio Code
- Explorer:** Shows a file tree under 'NODE JS TUTORIAL YOUTUBE'. The 'content\\subfolder' directory contains files: test.txt, 1-intro.js, 2-globals.js (selected), 3-modules.js, 4-names.js, 5-utils.js, 6-alternative-flavor.js, 7-mind-grenade.js, 8-os-module.js, and app.js.
- Code Editor:** The '2-globals.js' file is open, displaying the following code:

```
// GLOBAL Variables - NO WINDOW
// __dirname - path to current directory
// __filename - file name
// require - function to use modules (CommonJS)
// module - info about current module (file)
// process - info about env where the program is being executed
// console - console is also an global variable which helps to print values on terminal

// there are many more global variables
console.log(__dirname);
console.log(__filename);
setInterval(() => {
  console.log("Hello World !!!");
}, 1000);
setTimeout(() => {
  console.log("Hello Another World !!!");
}, 2000);

// setInterval and setTimeout can also be used in node js
```
- Bottom Status Bar:** PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> | In 13, Col 20 | Spaces: 4 | UTF-8 | CRLF | {} JavaScript | ✓ Prettier | ⚡ Q

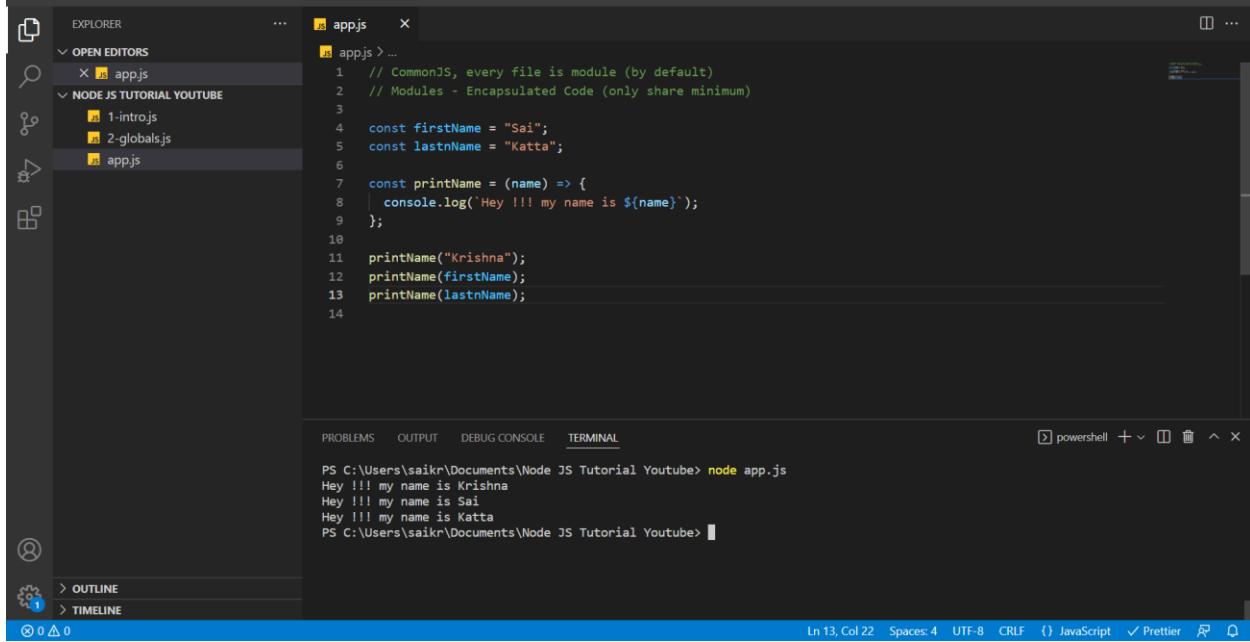
Modules in Node

Generally, our code will be split into multiple files, but all would be connected to app.js file because we run our entire application runs using one file.

Every File in node is a module.

Modules are Encapsulated code (only share minimum)

Without the concept of Module



```
// CommonJS, every file is module (by default)
// Modules - Encapsulated Code (only share minimum)

const firstName = "Sai";
const lastName = "Katta";

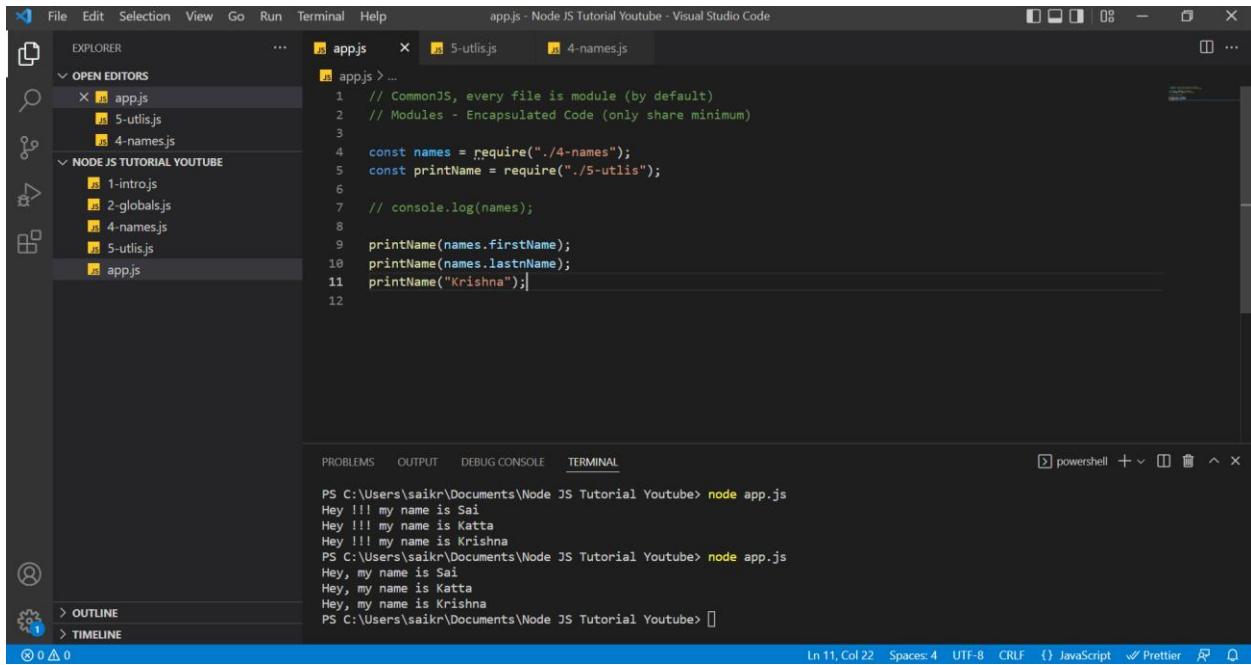
const printName = (name) => {
  console.log(`Hey !!! my name is ${name}`);
};

printName("Krishna");
printName(firstName);
printName(lastName);
```

PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
Hey !!! my name is Krishna
Hey !!! my name is Sai
Hey !!! my name is Katta
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>

With concept of Module

app.js



```
// CommonJS, every file is module (by default)
// Modules - Encapsulated Code (only share minimum)

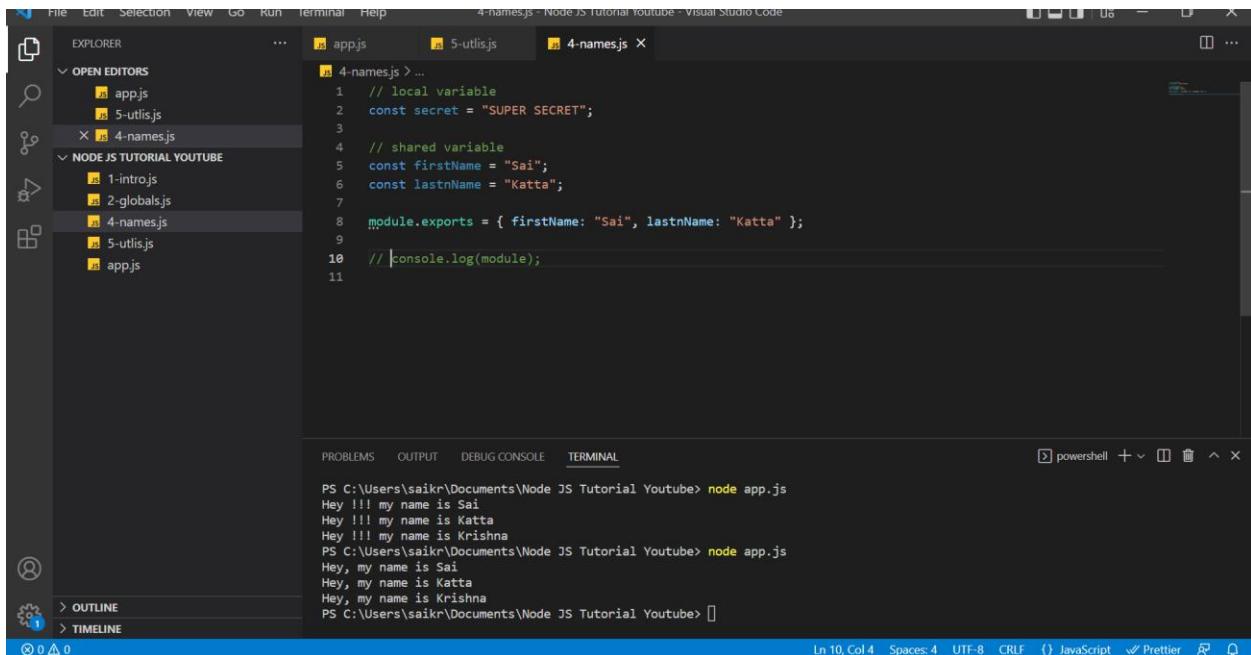
const names = require("./4-names");
const printName = require("./5-utilis");

// console.log(names);

printName(names.firstName);
printName(names.lastName);
printName("Krishna");
```

```
Hey !!! my name is Sai
Hey !!! my name is Katta
Hey !!! my name is Krishna
Hey, my name is Sai
Hey, my name is Katta
Hey, my name is Krishna
```

4-names.js



```
// local variable
const secret = "SUPER SECRET";

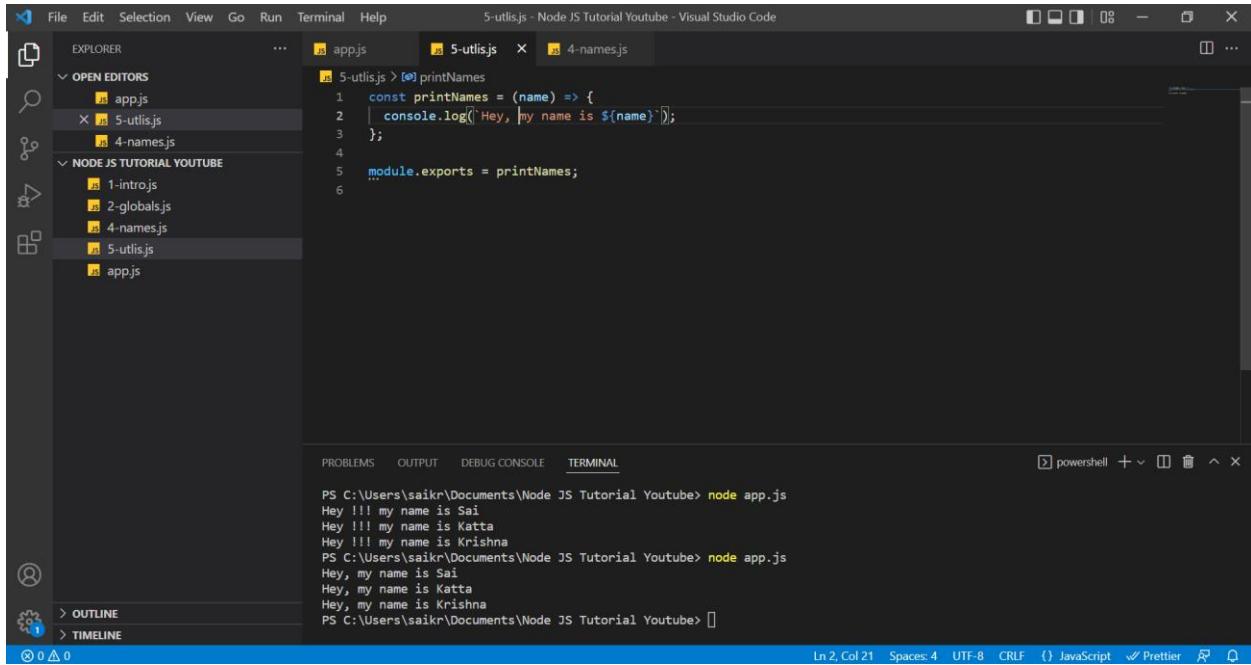
// shared variable
const firstName = "Sai";
const lastName = "Katta";

module.exports = { firstName: "Sai", lastName: "Katta" };

// console.log(module);
```

```
Hey !!! my name is Sai
Hey !!! my name is Katta
Hey !!! my name is Krishna
Hey, my name is Sai
Hey, my name is Katta
Hey, my name is Krishna
```

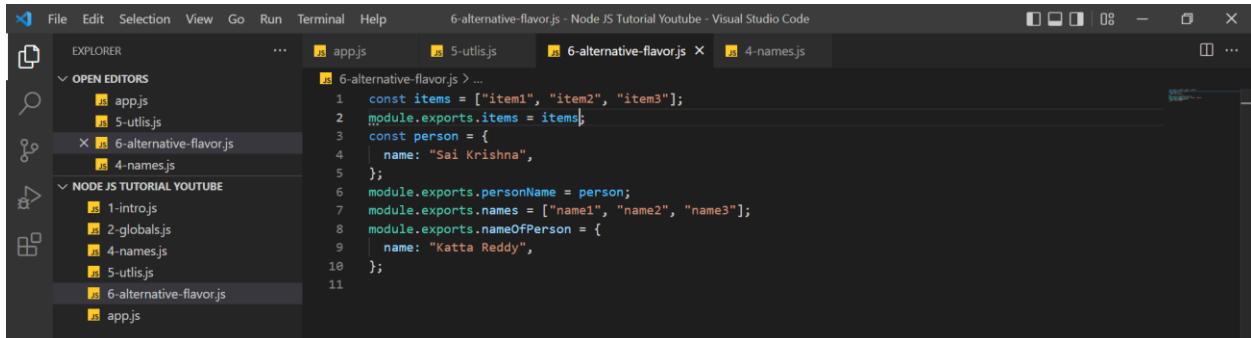
5-utils.js



```
File Edit Selection View Go Run Terminal Help 5-utils.js - Node JS Tutorial Youtube - Visual Studio Code
EXPLORER OPEN EDITORS NODE JS TUTORIAL YOUTUBE
app.js 5-utils.js 4-names.js
5-utils.js > [o] printNames
1 const printNames = (name) => {
2   console.log(`Hey, my name is ${name}`);
3 }
4
5 module.exports = printNames;
6

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
Hey !!! my name is Sai
Hey !!! my name is Katta
Hey !!! my name is Krishna
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
Hey, my name is Sai
Hey, my name is Katta
Hey, my name is Krishna
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> []
Ln 2, Col 21  Spaces: 4  UTF-8  CRLF  () JavaScript  ✓ Prettier  ⚡  ⌂
```

6-alternative-flavors.js



```
File Edit Selection View Go Run Terminal Help 6-alternative-flavor.js - Node JS Tutorial Youtube - Visual Studio Code
EXPLORER OPEN EDITORS NODE JS TUTORIAL YOUTUBE
app.js 5-utils.js 6-alternative-flavor.js 4-names.js
6-alternative-flavor.js > ...
1 const items = ["item1", "item2", "item3"];
2 module.exports.items = items;
3 const person = {
4   name: "Sai Krishna",
5 };
6 module.exports.personName = person;
7 module.exports.names = ["name1", "name2", "name3"];
8 module.exports.nameOfPerson = {
9   name: "Katta Reddy",
10 };
11

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
[object Object]
[object Object]
[object Object]
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> []
Ln 2, Col 21  Spaces: 4  UTF-8  CRLF  () JavaScript  ✓ Prettier  ⚡  ⌂
```

We can do multiple exports or else we can use the approach having only module export per file. It depends on the comfort of the person writing code.

In app.js, we can import it only as one single object because in modules we only use export object to export data from one module to another so that is the reason we can have multiple objects like items, personName, names, nameOfPerson from 6-alternative-js inside one object named data of app.js file.

```

File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS ...
app.js x 5-utiljs x 6-alternative-flavor.js x 4-names.js
app.js > ...
2 // Modules - Encapsulated Code (only share minimum)
3
4 const names = require("./4-names");
5 const printName = require("./5-utiljs");
6
7 // console.log(names);
8
9 // printName(names.firstName);
10 // printName(names.lastName);
11 // printName("Krishna");
12
13 const data = require("./6-alternative-flavor");
14 console.log(data);
15

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
{
  items: [ 'item1', 'item2', 'item3' ],
  personName: { name: 'Sai Krishna' },
  names: [ 'name1', 'name2', 'name3' ],
  nameOfPerson: { name: 'Katta Reddy' }
}
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>

Ln 11, Col 18 Spaces: 4 UTF-8 CRLF () JavaScript ✓ Prettier ⚡ Q

```

Basically, when we import a module in app.js file we invoke that module. Invoking means sometimes we may receive data through exports objects or sometimes we may invoke a function to execute some functionality in that module.

To get a good explanation about that, please have a look at the below screenshots.

```

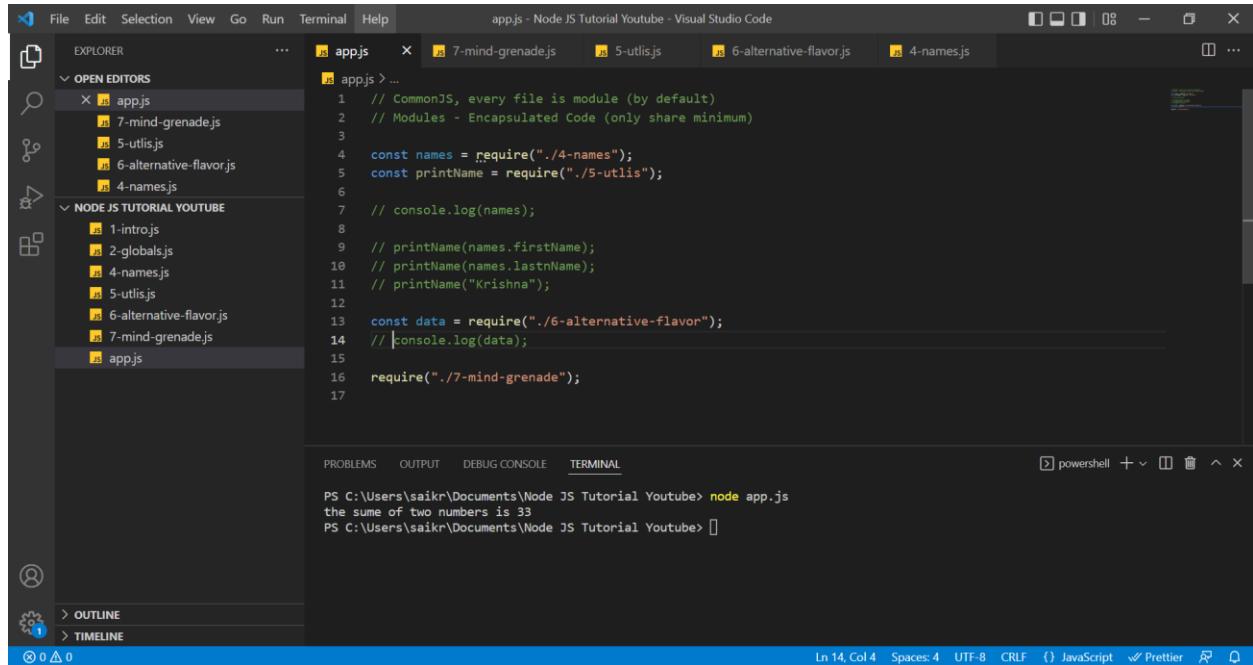
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS ...
app.js x 7-mind-grenade.js x 5-utiljs x 6-alternative-flavor.js x 4-names.js
7-mind-grenade.js > ...
1 const num1 = 15;
2 const num2 = 18;
3
4 const addValues = () => {
5   console.log(`the sum of two numbers is ${num1 + num2}`);
6 };
7
8 addValues();
9

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
the sum of two numbers is 33
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>

Ln 9, Col 1 Spaces: 4 UTF-8 CRLF () JavaScript ✓ Prettier ⚡ Q

```

In app.js, when we run node app.js, with the help of **require** global variable we are invoking the module 7-mind-grenade.js, which in turns executes a function named addValues() which calculates sum of two numbers.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a tree view of files. Under "OPEN EDITORS", there are files: app.js, 7-mind-grenade.js, 5-utilis.js, 6-alternative-flavor.js, and 4-names.js. Under "NODE JS TUTORIAL YOUTUBE", there are files: 1-intro.js, 2-globals.js, 3-arrays.js, 5-utilis.js, 6-alternative-flavor.js, 7-mind-grenade.js, and app.js.
- Code Editor:** The active file is app.js, containing the following code:

```
// CommonJS, every file is module (by default)
// Modules - Encapsulated Code (only share minimum)

const names = require("./4-names");
const printName = require("./5-utilis");

// console.log(names);

// printName(names.firstName);
// printName(names.lastName);
// printName("Krishna");

const data = require("./6-alternative-flavor");
// |console.log(data);

require("./7-mind-grenade");
```
- Terminal:** Shows the command line output:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
the sum of two numbers is 33
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>
```
- Status Bar:** Shows "Ln 14, Col 4" and other settings like "Spaces: 4", "UTF-8", "CRLF", "JavaScript", "Prettier".

This can be great example, since we use a bunch of third-party modules which executes functionalities in background, but we aren't able to see any of those methods on screen. We just extract few methods from the module using **require** global variable and use it in our code.

Built-in Modules in Node JS

Below are some of the major modules used in Node JS

OS (Operating System Module)

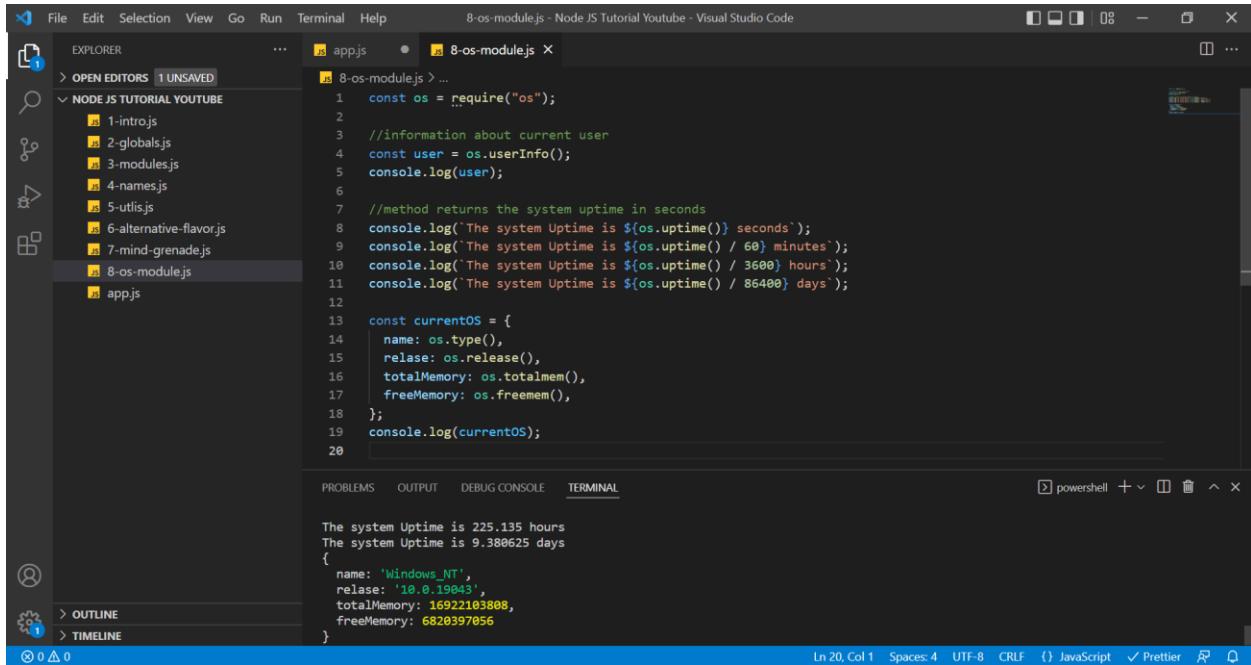
PATH Module

FS (File System Module)

HTTP Module – used to set up our HTTP server

OS (Operating System Module)

Provides methods and properties that are useful to interact with operating system and as well as server.



The screenshot shows the Visual Studio Code interface with the file `8-os-module.js` open. The code uses the `os` module to log system uptime and details about the current operating system. The terminal output shows the system uptime and the current operating system object.

```
const os = require("os");
//information about current user
const user = os.userInfo();
console.log(user);

//method returns the system uptime in seconds
console.log(`The system Uptime is ${os.uptime()} seconds`);
console.log(`The system Uptime is ${os.uptime() / 60} minutes`);
console.log(`The system Uptime is ${os.uptime() / 3600} hours`);
console.log(`The system Uptime is ${os.uptime() / 86400} days`);

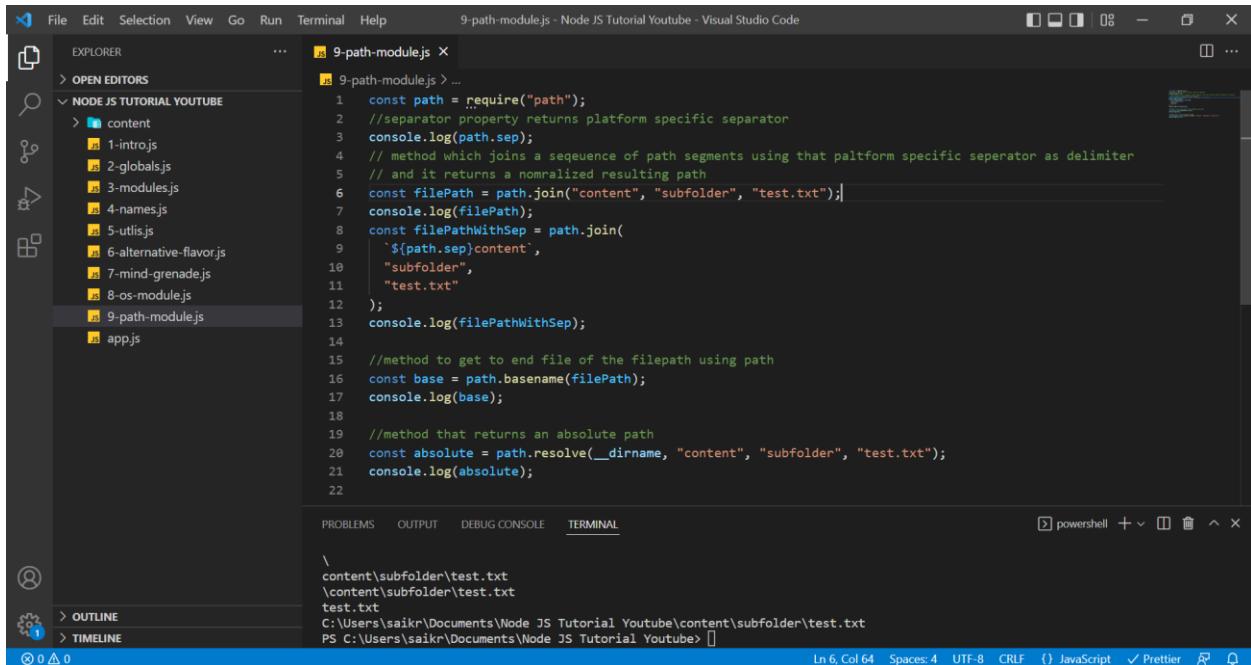
const currentOS = {
  name: os.type(),
  release: os.release(),
  totalMemory: os.totalmem(),
  freeMemory: os.freemem(),
};
console.log(currentOS);

The system Uptime is 225.135 hours
The system Uptime is 9.380625 days
{
  name: 'Windows_NT',
  release: '10.0.19043',
  totalMemory: 16922103888,
  freeMemory: 6820397056
}
```

PATH Module

With JavaScript in browser, we cannot access File System but in node we can access the File System.

PATH Module allows us to interact with file paths easily.



The screenshot shows the Visual Studio Code interface with the file `9-path-module.js` open. The code demonstrates various methods of the `path` module, such as joining paths, getting the base name, and resolving absolute paths. The terminal output shows the resulting file paths.

```
const path = require("path");
//separator property returns platform specific separator
console.log(path.sep);
// method which joins a sequence of path segments using that platform specific separator as delimiter
// and it returns a normalized resulting path
const filePath = path.join("content", "subfolder", "test.txt");
console.log(filePath);
const filePathWithSep = path.join(
  `${path.sep}content`,
  "subfolder",
  "test.txt"
);
console.log(filePathWithSep);

//method to get to end file of the filepath using path
const base = path.basename(filePath);
console.log(base);

//method that returns an absolute path
const absolute = path.resolve(__dirname, "content", "subfolder", "test.txt");
console.log(absolute);

\content\subfolder\test.txt
\content\subfolder\test.txt
test.txt
C:\Users\saikr\Documents\Node JS Tutorial Youtube\content\subfolder\test.txt
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> []
```

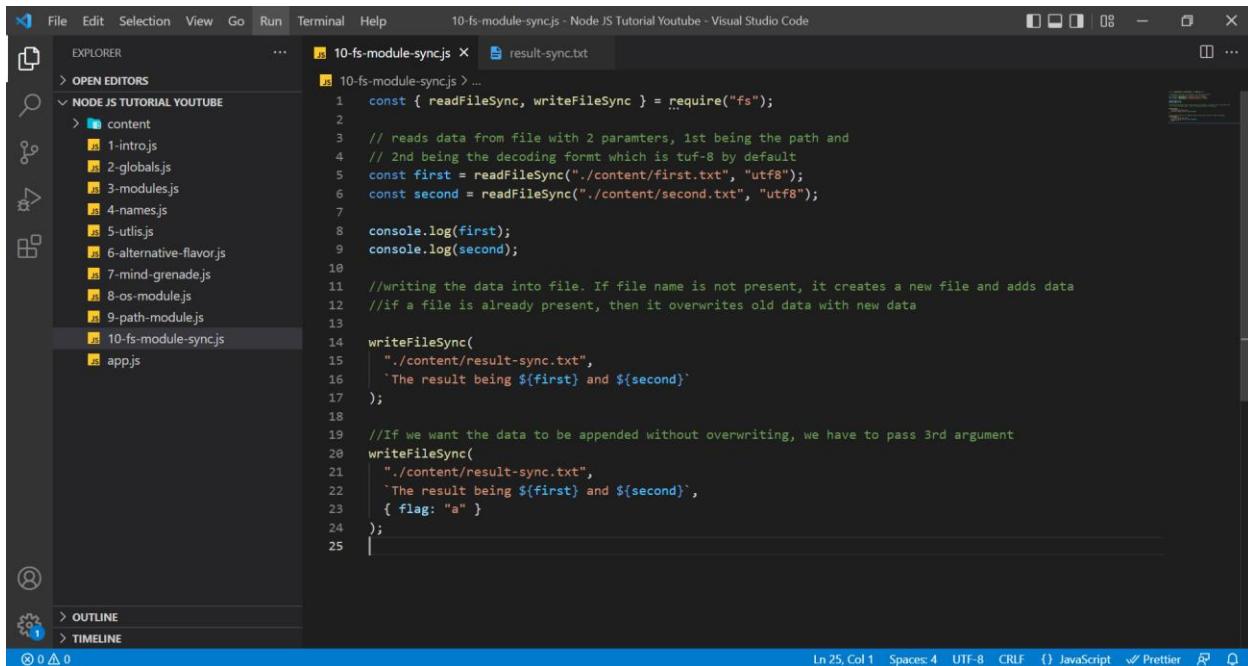
FS (File System Module)

Accessing File System Module can be done in two ways,

Asynchronously (Which is non-Blocking) and Synchronously (Which is Blocking)

Let us have a look at the **Synchronous** way of reading and writing data into a file.

10-fs-module-sync.js



```
10-fs-module-sync.js - Node JS Tutorial Youtube - Visual Studio Code
```

```
const { readFileSync, writeFileSync } = require("fs");

// reads data from file with 2 parameters, 1st being the path and
// 2nd being the decoding format which is utf-8 by default
const first = readFileSync("./content/first.txt", "utf8");
const second = readFileSync("./content/second.txt", "utf8");

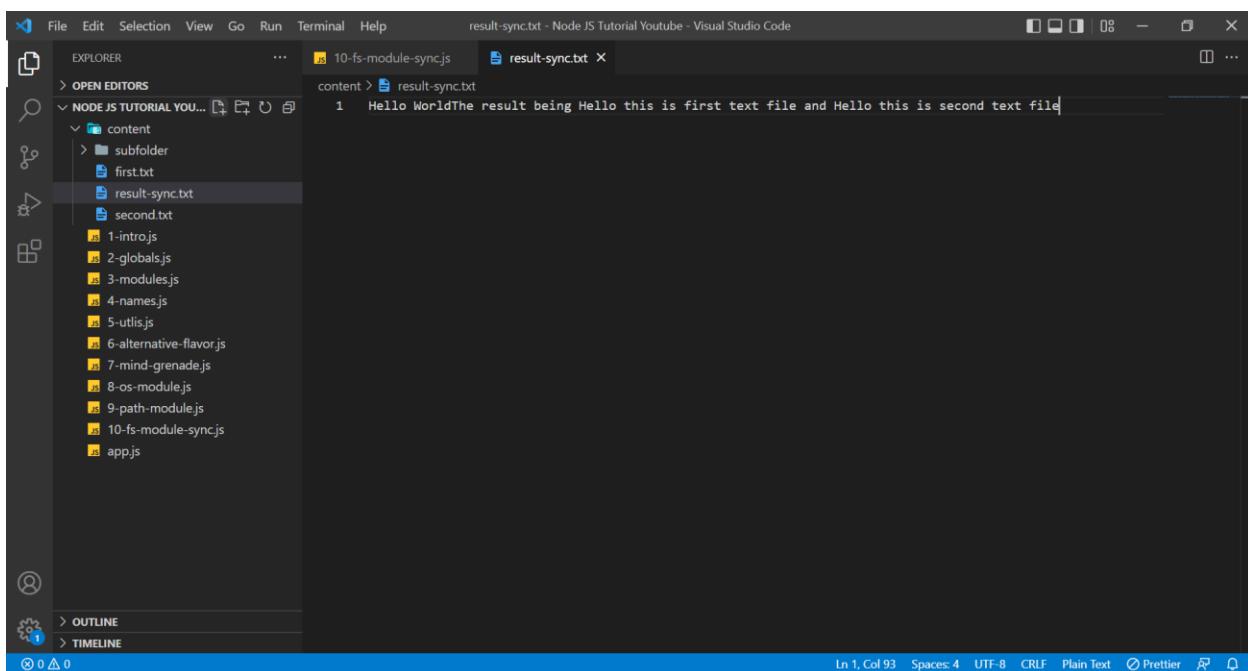
console.log(first);
console.log(second);

//writing the data into file. If file name is not present, it creates a new file and adds data
//if a file is already present, then it overwrites old data with new data

writeFileSync(
    "./content/result-sync.txt",
    `The result being ${first} and ${second}`
);

//If we want the data to be appended without overwriting, we have to pass 3rd argument
writeFileSync(
    "./content/result-sync.txt",
    `The result being ${first} and ${second}`,
    { flag: "a" }
);
```

result-sync.txt

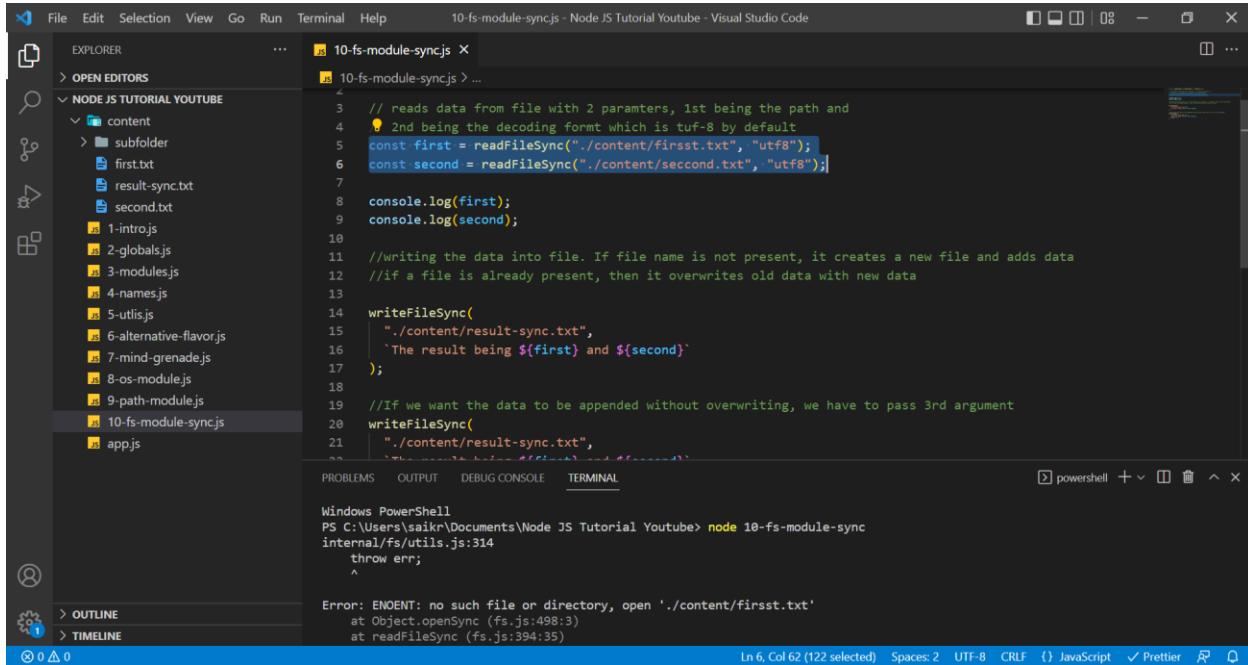


```
result-sync.txt - Node JS Tutorial Youtube - Visual Studio Code
```

```
content > result-sync.txt
```

```
Hello WorldThe result being Hello this is first text file and Hello this is second text file
```

If you try to read data from a non-existed file, then node will throw an error. Here we are taking files which are not existing. Here the node is throwing an error.



The screenshot shows the Visual Studio Code interface with the title bar "10-fs-module-sync.js - Node JS Tutorial Youtube - Visual Studio Code". The Explorer sidebar on the left shows a folder structure under "NODE JS TUTORIAL YOUTUBE" containing files like "content", "first.txt", "second.txt", and several numbered script files. The main editor area displays a JavaScript file named "10-fs-module-sync.js" with the following code:

```

1 // reads data from file with 2 parameters, 1st being the path and
2 // 2nd being the decoding format which is utf-8 by default
3 const first = readFileSync("./content/firsst.txt", "utf8");
4 const second = readFileSync("./content/seccond.txt", "utf8");
5
6 console.log(first);
7 console.log(second);
8
9 //writing the data into file. If file name is not present, it creates a new file and adds data
10 //if a file is already present, then it overwrites old data with new data
11
12 writeFileSync(
13   "./content/result-sync.txt",
14   `The result being ${first} and ${second}`
15 );
16
17 //If we want the data to be appended without overwriting, we have to pass 3rd argument
18 writeFileSync(
19   "./content/result-sync.txt",
20   `The result being ${first} and ${second}`
21 );
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
311
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
143
```

result-async.txt

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders under 'NODE JS TUTORIAL YOUTUBE'. The 'content' folder contains 'first.txt', 'result-async.txt', 'result-sync.txt', and 'second.txt'. The 'OPEN EDITORS' tab shows 'app.js' and 'result-async.txt'. The 'result-async.txt' editor has the following content:

```
Hello this is first text file and Hello this is first text file
```

The Terminal tab at the bottom shows the command 'node app.js' being run, with the output:

```
Hello this is first text file
Hello this is second text file
undefined
```

When we try to read the data and couldn't find the file.

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders under 'NODE JS TUTORIAL YOUTUBE'. The 'OPEN EDITORS' tab shows 'app.js' and 'result-async.txt'. The 'app.js' editor contains the following code:

```
const fs = require('fs');
const readline = require('readline');

const rl = readline.createInterface({
  input: fs.createReadStream('./content/first.txt'),
  output: fs.createWriteStream('./content/result-async.txt')
});

rl.on('line', (line) => {
  console.log(`Line: ${line}`);
});

rl.on('close', () => {
  process.exit(0);
});
```

The Terminal tab at the bottom shows the command 'node app.js' being run, with the output:

```
Hello this is first text file
[Error: ENOENT: no such file or directory, open 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\content\seccond.txt']
  errno: -4058,
  code: 'ENOENT',
  syscall: 'open',
  path: 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\content\seccond.txt'
]
```

result-async.txt (result will be only from first result variable which is “Hello this is first text file”)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure under "NODE JS TUTORIAL YOUTUBE" containing "content", "first.txt", "result-async.txt", "result-sync.txt", and "second.txt". Below "content" are subfolders "1-intro.js" through "10-fs-module-sync.js" and "app.js".
- Terminal:** Displays the command "node app.js" and its output:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
Hello this is first text file
[Error: ENOENT: no such file or directory, open 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\content\second.txt' {
  errno: -4058,
  code: 'ENOENT',
  syscall: 'open',
  path: 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\content\second.txt'
}]
undefined
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>
```
- Status Bar:** Shows "Ln 1, Col 64" and other standard status bar information.

Sync vs Async

In Synchronous way, JavaScript executes code in line-by-line fashion, which means it executes line after line even if task 2 takes more time, it will wait until the task 2 execution is complete and then moves on to task 3.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure under "NODE JS TUTORIAL YOUTUBE" containing "content", "first.txt", "result-async.txt", "result-sync.txt", and "second.txt". Below "content" are subfolders "1-intro.js" through "10-fs-module-sync.js" and "app.js".
- Terminal:** Displays the command "node 10-fs-module-sync.js" and its output:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node 10-fs-module-sync.js
10-fs-module-sync.js >
1  const { readFileSync, writeFileSync } = require("fs");
2  console.log("Start");
3
4 // reads data from file with 2 parameters, 1st being the path and
5 // 2nd being the decoding format which is tuf-8 by default
6 const first = readFileSync("./content/first.txt", "utf8");
7 const second = readFileSync("./content/second.txt", "utf8");
8
9 //writing the data into file. If file name is not present, it creates a new file and adds data
10 //if a file is already present, then it overwrites old data with new data
11
12 writeFileSync(
13   "./content/result-sync.txt",
14   `The result being ${first} and ${second}`);
15
16 //If we want the data to be appended without overwriting, we have to pass 3rd argument
17 writeFileSync(
18   "./content/result-sync.txt",
19   `The result being ${first} and ${second}`,
20   { flag: "a" });
21
22
23 console.log("Done with this task");
24 console.log("Starting next task");
25
```
- Status Bar:** Shows "Ln 23, Col 36" and other standard status bar information.

In Asynchronous way, JavaScript executes line-by-line fashion but if it sees any task (assume task 2) taking time it will offload the task2 and move on to another task but in the meanwhile the task 2 runs in the background and once it is done it will provide the results.

In Asynchronous way, we handle the tasks using the callback function but if we use too many callback functions it will create a Callback Hell and hence, we use Promises or Async and Await for asynchronous.

```

OPEN EDITORS 1 UNSAVED
EXPLORER 11-fs-module-async.js 10-fs-module-sync.js 11-fs-module-async.js
NODE JS TUTORIAL YOUTUBE
content
1-intro.js
2-global.js
3-modules.js
4-names.js
5-util.js
6-alternative-flavor.js
7-mind-grenade.js
8-os-module.js
9-path-module.js
10-fs-module-sync.js
11-fs-module-async.js
app.js

11-fs-module-async.js > ...
1 const { readfile, writefile } = require("fs");
2 console.log("Start");
3 readfile("./content/first.txt", "utf8", (err, result) => {
4   if (err) {
5     console.log(err);
6     return;
7   }
8   const first = result;
9   // console.log(result);
10  readfile("./content/second.txt", "utf8", (err, result) => {
11    if (err) {
12      console.log(err);
13      return;
14    }
15    // console.log(result);
16  });
17  const second = result;
18  console.log("Done with this task");
19  writefile(
20    "./content/result-async.txt",
21    `${first} and ${second}`,
22    (err, result) => {
23      if (err) {
24        console.log(err);
25        return;
26      }
27      // console.log(result);
28    }
29  );
30 });
31 console.log("Starting next task");

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node 11-fs-module-async
Start
Starting next task
Done with this task
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>

HTTP Module

HTTP Module is used to setup a web server.

```

OPEN EDITORS
EXPLORER app.js
NODE JS TUTORIAL YOUTUBE
content
1-intro.js
2-global.js
3-modules.js
4-names.js
5-util.js
6-alternative-flavor.js
7-mind-grenade.js
8-os-module.js
9-path-module.js
10-fs-module-sync.js
11-fs-module-async.js
app.js

app.js > (e) server > http.createServer() callback
1 //Instead of using write and end methods, we can use end method itself
2 //res.end("Hi! welcome to our home page");
3 res.write("Hi! welcome to our home page");
4 res.end();
5 return;
6 }
7 if (req.url === "/about") {
8   //Instead of using write and end methods, we can use end method itself
9   //res.end("Here is our Short History");
10  res.write("Here is our Short History");
11  res.end();
12  return;
13 }
14 //instead of using write and end methods, we can use end method itself
15 //res.end("<h1>Oops!!</h1>");
16 //p>The page you are looking for is not available</p>
17 //a href="/">Back to Home Page</a>
18 res.write( "<h1>Oops!!</h1>" );
19 <p>The page you are looking for is not available</p>
20 <a href="/">Back to Home Page</a>;
21 res.end();
22 }
23 //listen method is used to listen the server at particular port number
24 server.listen(5000);
25
26
27
28
29
30
31

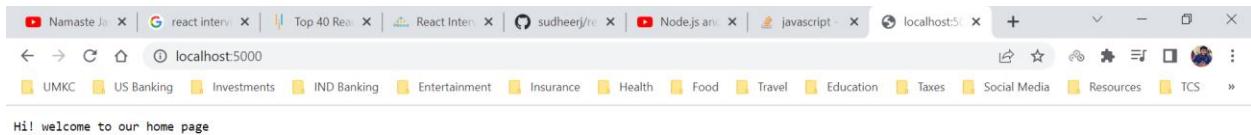
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

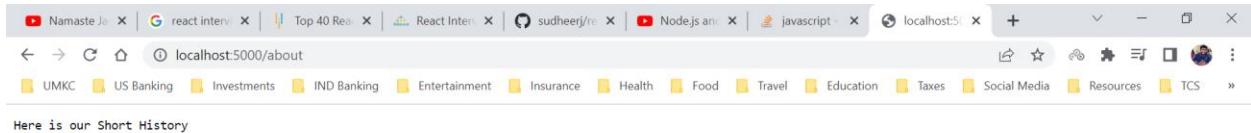
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app

Web Server is started when we run node app.js command in the terminal.

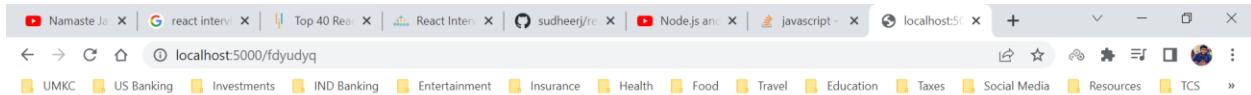
Home Page on localhost:5000



About Page on localhost:5000/about



Error Page on localhost:5000/fdyudyq

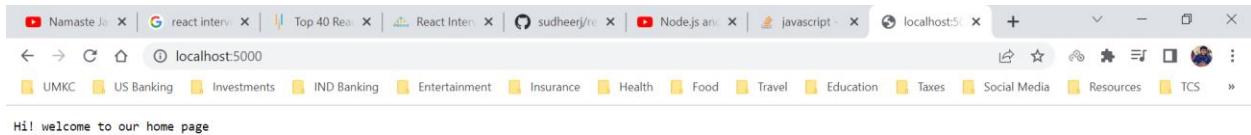


Oops!!!

The page you are looking for is not available

Back to Home Page

On Click of Home Page Link, we will be redirected to Home Page



Hi! welcome to our home page

NPM Information

When we install node, we automatically install NPM (Node Package Manager). NPM enables to do 3 things

1. Reuse our code in another project.
2. Use code written by other developers using a npm command.
3. Share our solutions with other developers as well.

NPM is hosted on npmjs.com, here we can find useful utility functions to full blown frameworks and libraries.

Example: React is available in npm using ***npx create-react-app <app-name>***

A typical node project will have more than few packages installed as dependencies.

NPM calls the reusable code, basically a package. A package is a folder that contains a JavaScript code. Package can be also called as dependency and module as well.

There is no quality control in npm registry. Anyone can publish anything on the registry. A good indication for good and secure package is the number of weekly downloads.

NPM in Node

Since NPM is already installed with node, In Node JS environment we have access to NPM global variable `npm`.

`npm -v`

`package.json` - manifest file (stores important info about project/package)

3 ways to create `package.json` file

1. manual approach (create `package.json` in the root, create properties etc.)
2. `npm init` (step by step, press enter to skip)
3. `npm init -y` (everything default)

The screenshot shows the Visual Studio Code interface. The left sidebar has 'OPEN EDITORS' expanded, showing 'NODE JS TUTORIAL YOUTUBE' which contains 'content', 'node_modules', 'bootstrap', 'lodash', '1-intro.js', '2-global.js', '3-modules.js', '4-names.js', '5-util.js', '6-alternative-flavor.js', '7-mind-grenade.js', '8-os-modules.js', '9-path-module.js', '10-fs-module-sync.js', '11-fs-module-async.js', '12-http-module.js', and 'app.js'. The right side shows two tabs: 'app.js' and 'package.json'. The 'app.js' tab contains the following code:

```
1 // npm - global command, comes with node
2 // npm --v
3
4 // local dependency - use it only in this particular project
5 // npm i <packagename>
6
7 // global dependency - use it in any project
8 // npm install -g <packagename>
9 // sudo npm install -g <packagename> (mac) (In Mac, most likely the password will be asked)
10
11 // package.json - manifest file (stores important info about project/package)
12 // 3 ways to create package.json file
13 // 1. manual approach (create package.json in the root, create properties etc)
14 // 2. npm init (step by step, press enter to skip)
15 // 3. npm init -y (everything default)
16
```

The 'package.json' tab is currently active. Below the editor, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The terminal shows the path 'PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>'. At the bottom, status bar items include 'Ln 16, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', '(JavaScript)', 'Prettier', and icons for file operations.

If you wish to publish this NPM Registry, then select a unique name for package which is not available in the NPM registry.

The screenshot shows the Visual Studio Code interface with the 'package.json' file open. The code in the editor is:

```
1 {
2   "name": "nodejs-tutorial",
3   "version": "1.0.0",
4   "description": "",
5   "main": "1-intro.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "sai krishna",
10  "license": "ISC"
11}
```

The 'package.json' tab is active. Below the editor, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The terminal shows the path 'PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>'. A message 'Is this OK? (yes)' is displayed above the terminal input field. At the bottom, status bar items include 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'LF', '(JSON)', 'Prettier', and icons for file operations.

We will try to install first package into our project. The lodash is a utility library. Now we can see dependencies object inside the package.json file. We can also see node_modules folder is installed with lodash dependency.

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a 'NODE JS TUTORIAL YOUTUBE' folder containing several files like 'content', 'node_modules', '1-intro.js', etc. The 'package.json' file is open in the editor. Its content is:

```
1 {
  "name": "nodejs-tutorial",
  "version": "1.0.0",
  "description": "",
  "main": "1-intro.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "sai krishna",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.21"
  }
}
```

In the Terminal tab, the command 'npm i lodash' is run, and the output shows:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> npm i lodash
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN nodejs-tutorial@1.0.0 No description
npm WARN nodejs-tutorial@1.0.0 No repository field.

+ lodash@4.17.21
added 1 package from 2 contributors and audited 1 package in 2.389s
found 0 vulnerabilities
```

The status bar at the bottom indicates: Line 15, Col 1 | Spaces: 2 | UTF-8 | LF | {} JSON | ✓ Prettier

node_modules is the folder where all the dependencies are stored.

Installed bootstrap library as well.

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a 'NODE JS TUTORIAL YOUTUBE' folder containing 'content', 'node_modules', 'bootstrap', and 'lodash'. The 'package.json' file is open in the editor. Its content is:

```
1 {
  "name": "nodejs-tutorial",
  "version": "1.0.0",
  "description": "",
  "main": "1-intro.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "sai krishna",
  "license": "ISC",
  "dependencies": {
    "bootstrap": "^5.1.3",
    "lodash": "^4.17.21"
  }
}
```

In the Terminal tab, the command 'npm i bootstrap' is run, and the output shows:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> npm i bootstrap
npm WARN bootstrap@5.1.3 requires a peer of @popperjs/core@^2.10.2 but none is installed. You must install peer dependencies yourself.
npm WARN nodejs-tutorial@1.0.0 No description
npm WARN nodejs-tutorial@1.0.0 No repository field.

+ bootstrap@5.1.3
added 1 package from 2 contributors and audited 2 packages in 2.016s

1 package is looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
```

The status bar at the bottom indicates: Line 15, Col 2 | Spaces: 2 | UTF-8 | LF | {} JSON | ✓ Prettier

In case of external dependencies, we need to first install them through NPM if we don't install then we won't be able to find it. In case of internal dependencies like in-built modules, we don't need to install anything.

Using loadash in our project

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders for a project named 'NODE JS TUTORIAL YOUTUBE'. The 'OPEN EDITORS' section shows 'app.js' and 'package.json'. The code editor window displays the contents of 'app.js'. The terminal at the bottom shows the output of running the script:

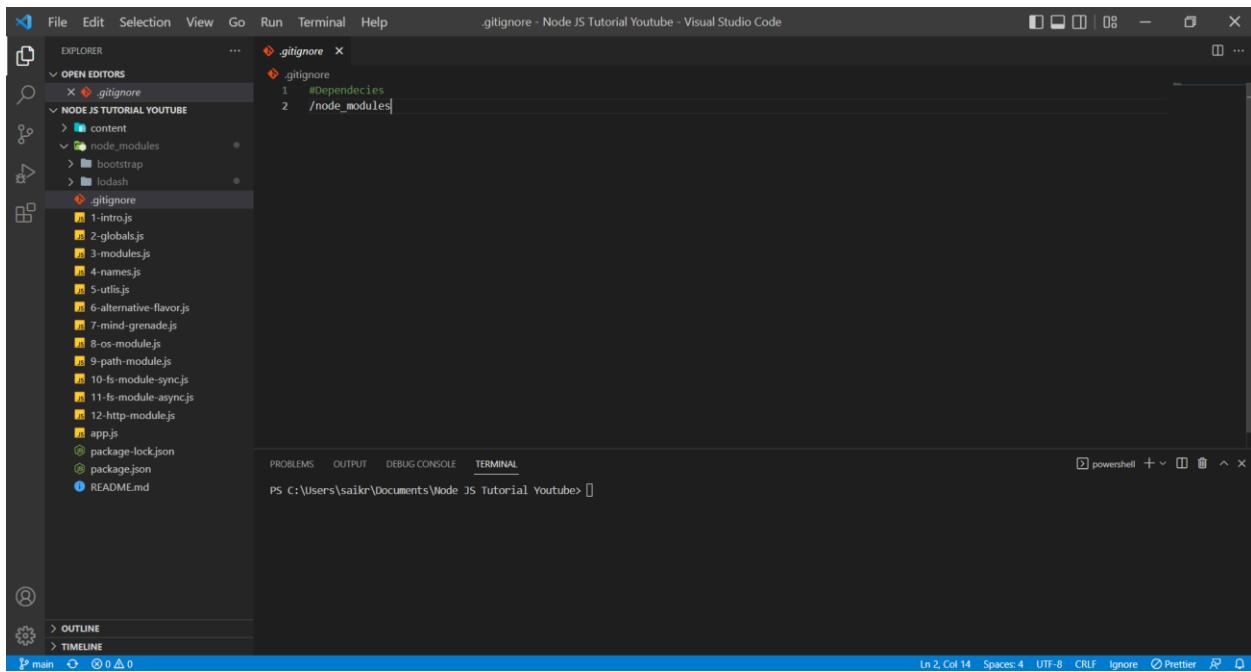
```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node app.js
[ 1, 2, 3, 4, 5 ]
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>
```

Sharing Code with Other Developers

We can share the code using the github repository. We cannot upload node_modules folder into the github repository. Hence, we create a file named .gitignore which will ignore the folders that we mention in the file.

It is very important to have package.json file because, we can take a repository from github and then use the command npm install to install all the dependencies into our project.

.gitignore file as mentioned earlier.



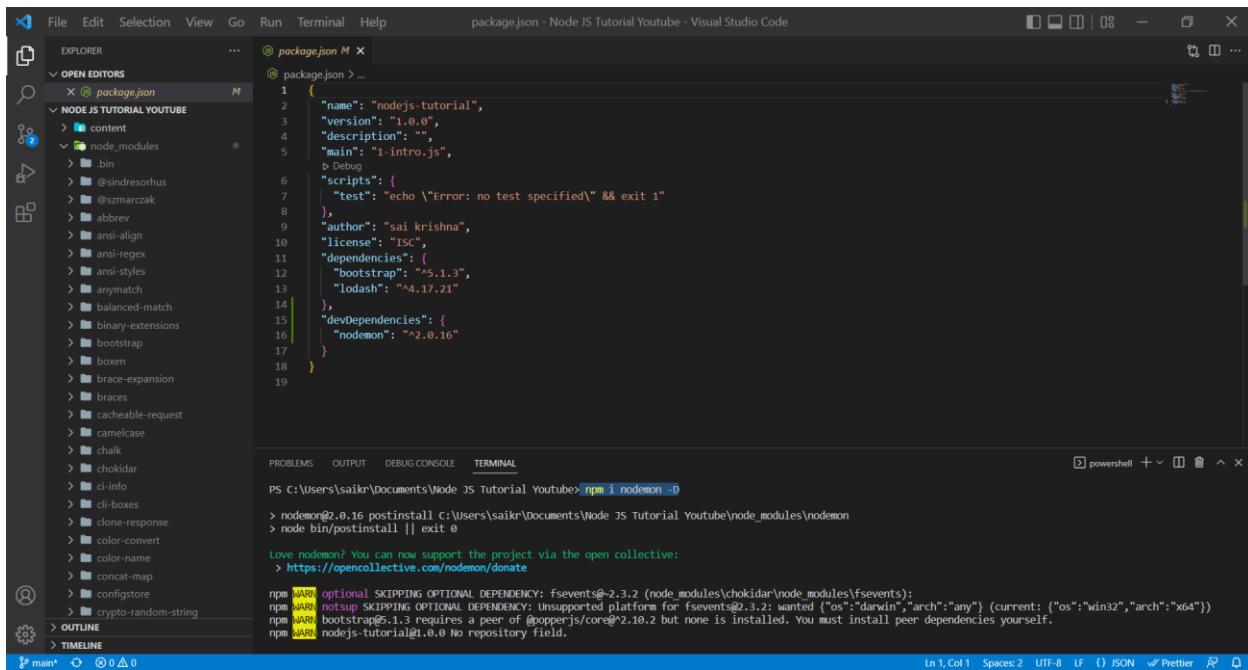
Nodemon

Nodemon is one of the packages of the npm. It is going to watch our file system and restart the application for us. In that way we don't have to type node <filename>. We can install it as dependency but here we are installing it as a dev dependency.

Command to install a package as dev dependency

npm i <package-name> -D or npm i <package-name> --save-dev

Nodemon as dev dependency In package.json file



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "NODE JS TUTORIAL YOUTUBE".
- Editor:** The "package.json" file is open, displaying the following JSON code:

```
1 {
2   "name": "nodejs-tutorial",
3   "version": "1.0.0",
4   "description": "",
5   "main": "1-intro.js",
6   "scripts": {
7     "test": "echo \\"Warning: no test specified\\\" && exit 1"
8   },
9   "author": "sai krishna",
10  "license": "ISC",
11  "dependencies": {
12    "bootstrap": "^5.1.3",
13    "lodash": "4.17.21"
14  },
15  "devDependencies": {
16    "nodemon": "^2.0.16"
17  }
18}
```

- Terminal:** Shows the command `npm i nodemon -D` being run in the terminal, followed by the output of the command.

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> npm i nodemon -D
> nodemon@2.0.16 postinstall C:\Users\saikr\Documents\Node JS Tutorial Youtube\node_modules\nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
> https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN bootstrap@5.1.3 requires a peer of @popperjs/core@^2.10.2 but none is installed. You must install peer dependencies yourself.
npm WARN nodejs-tutorial@1.0.0 No repository field.
```

- Status Bar:** Shows the current file is "package.json - Node JS Tutorial Youtube - Visual Studio Code".

In production we may not use the Nodemon, but Heroku and other services will be using other serious packages but in development stage this is an option.

Basically, devDependencies is nothing but the dependencies that we are using for the development purposes but once we get into production, we only use the dependencies used by our application.

Inside the scripts object of the package.json file we just setup some commands which we want to run. For example, we setup start which indeed is a replacement for command ***node <file-name>***.

We have run command **npm start** which in turn is starting the node application.

For some commands, we just need to use npm <name-of-command> like **npm start**.

With the **start** command which is nothing but **node app.js**, we are stopping the application after executing the command.

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there are several files and folders: .gitignore, 1-intro.js, 2-globals.js, 3-modules.js, 4-names.js, 5-utils.js, 6-alternative-flavor.js, 7-mind-grenade.js, 8-os-module.js, 9-path-module.js, 10-fs-module-sync.js, 11-fs-module-async.js, 12-http-module.js, app.js, package-lock.json, package.json, and README.md. The package.json file is open in the editor, showing the following content:

```
1  {
2   "name": "nodejs-tutorial",
3   "version": "1.0.0",
4   "description": "",
5   "main": "1-intro.js",
6   "scripts": {
7     "start": "node app.js"
8   },
9   "author": "sai krishna",
10  "license": "ISC",
11  "dependencies": {
12    "bootstrap": "^5.1.3",
13    "lodash": "4.17.21"
14  },
15  "devDependencies": {
16    "nodemon": "^2.0.16"
17  }
18}
```

In the bottom right corner of the code editor, there is a terminal window. It shows the command `npm start` being run, followed by the output: `> nodejs-tutorial@1.0.0 start C:\Users\saikr\Documents\Node JS Tutorial Youtube
> node app.js
[1, 2, 3, 4, 5]`. The terminal also shows the path `PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>`.

For some other commands, we need to run `npm run <name-of-command>` like `npm run dev`.

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there are several files and folders under the 'NODE JS TUTORIAL YOUTUBE' folder, including 'content', 'node_modules', '.gitignore', '1-intro.js', '2-globals.js', '3-modules.js', '4-names.js', '5-utilis.js', '6-alternative-flavor.js', '7-mind-grenade.js', '8-os-module.js', '9-path-module.js', '10-fs-module-sync.js', '11-fs-module-async.js', '12-http-module.js', 'app.js', 'package-lock.json', and 'package.json'. The 'package.json' file is open in the editor, showing its contents. The terminal at the bottom shows the command `node app.js` being run, which starts Nodemon. The output shows Nodemon version 2.0.16, watching for changes, and restarting the application. The application logs 'Hello World' and then 'Hello People'.

```
1 {
2   "name": "nodejs-tutorial",
3   "version": "1.0.0",
4   "description": "",
5   "main": "1-intro.js",
6   "scripts": [
7     "start": "node app.js",
8     "dev": "nodemon app.js"
9   ],
10  "author": "sai krishna",
11  "license": "ISC",
12  "dependencies": {
13    "bootstrap": "^5.1.3",
14    "lodash": "^4.17.21"
15  },
16  "devDependencies": {
17    "nodemon": "^2.0.16"
18  }
19}
```

```
[nodemon] 2.0.16
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
[ 1, 2, 3, 4, 5 ]
Hello World
[nodemon] clean exit - waiting for changes before restart
```

With the Nodemon package, since it watches the file system and then any if change appears in our file system, it immediately restarts our application.

The screenshot shows the Visual Studio Code interface with the same project structure as the previous screenshot. The 'package.json' file is open in the editor, showing the 'scripts' section. The 'start' script now contains the code `// console.log("Hello People")`. The terminal at the bottom shows the command `node app.js` being run, which starts Nodemon. The output shows Nodemon watching for changes, restarting due to changes, and then executing the new code, which logs 'Hello People'.

```
1 {
2   "name": "nodejs-tutorial",
3   "version": "1.0.0",
4   "description": "",
5   "main": "1-intro.js",
6   "scripts": [
7     "start": "node app.js",
8     "dev": "nodemon app.js"
9   ],
10  "author": "sai krishna",
11  "license": "ISC",
12  "dependencies": {
13    "bootstrap": "^5.1.3",
14    "lodash": "^4.17.21"
15  },
16  "devDependencies": {
17    "nodemon": "^2.0.16"
18  }
19}
```

```
[nodemon] 2.0.16
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
[ 1, 2, 3, 4, 5 ]
Hello World
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
[ 1, 2, 3, 4, 5 ]
Hello People
[nodemon] clean exit - waiting for changes before restart
```

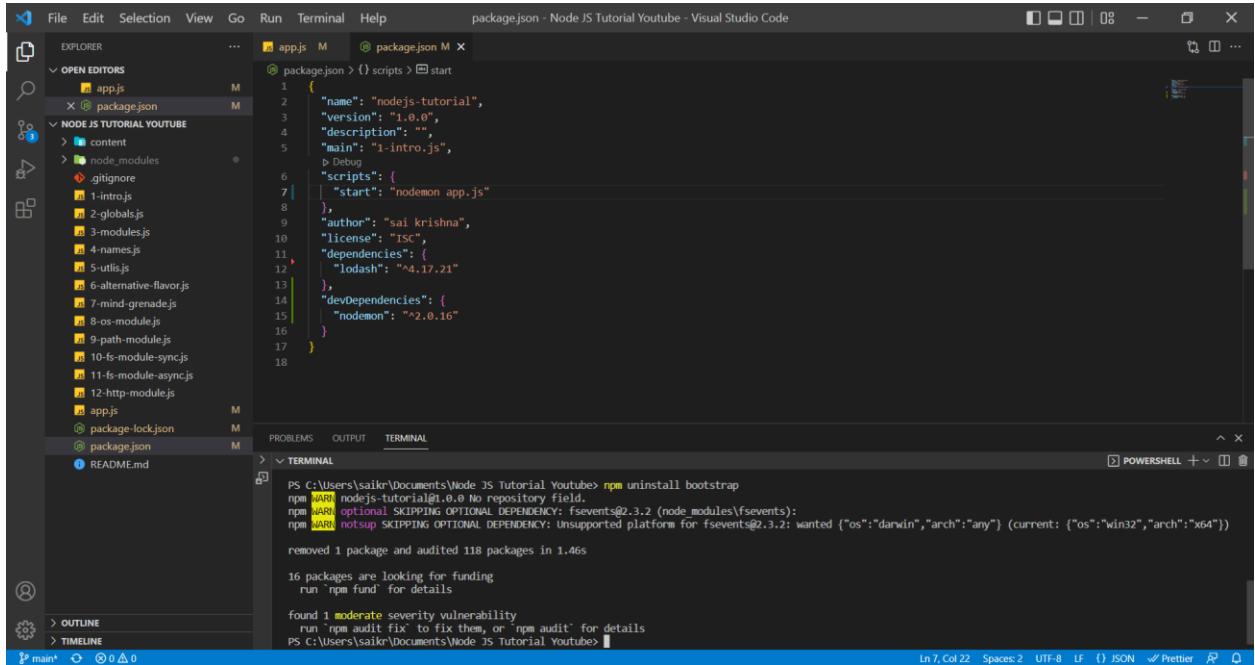
In the above example, we just changed the statement inside the `console.log` from "Hello World" to "Hello People" and Nodemon immediately recognized and restarted the application and executed the code.

Use Control + C command to stop the Nodemon.

Uninstall a Package in package.json file

There are two ways to uninstall a package.

1. Run command `npm uninstall <package-name>`



The screenshot shows the Visual Studio Code interface with the terminal tab active. The terminal window displays the command `npm uninstall bootstrap` being run in the directory `C:\Users\saikr\Documents\Node JS Tutorial Youtube`. The output shows that the package `bootstrap@1.0.0` was removed, along with 118 other packages. It also indicates that optional dependencies like `fsevents` were skipped due to unsupported platforms.

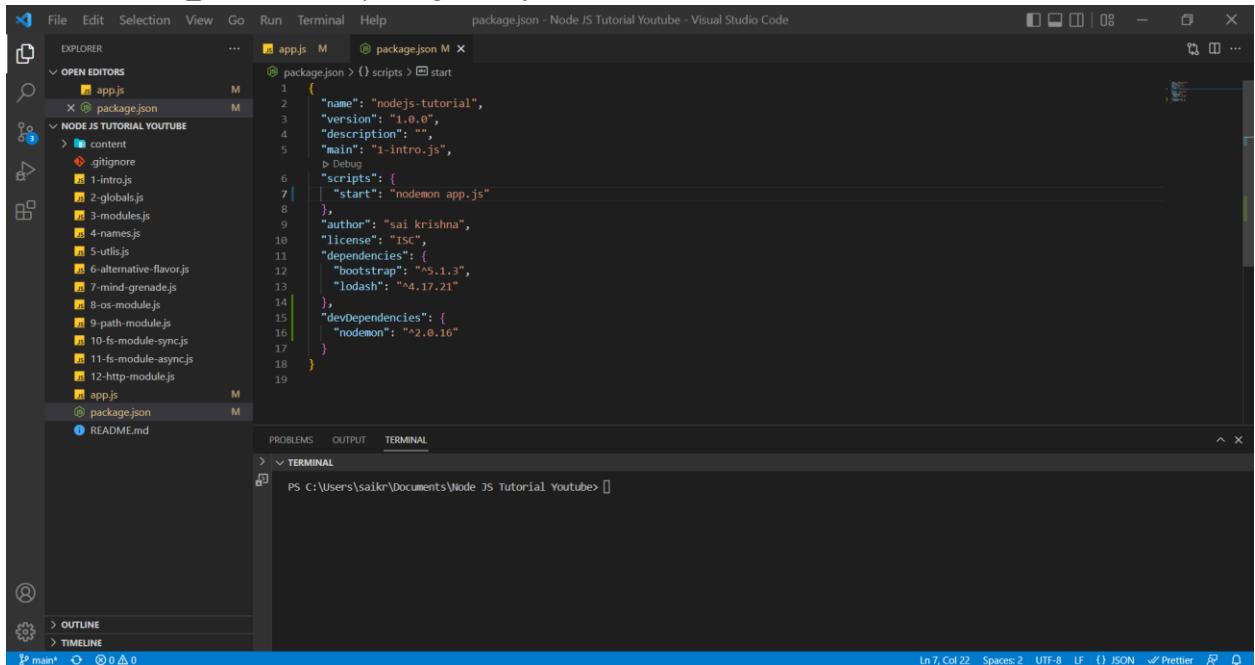
```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> npm uninstall bootstrap
npm WARN nodejs-tutorial@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup NOTSUPPLIED OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
removed 1 package and audited 118 packages in 1.46s

16 packages are looking for funding
  run npm fund for details

found 1 moderate severity vulnerability
  run npm audit fix to fix them, or npm audit for details
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>
```

2. Nuclear Approach is deleting `node_modules` folder, `package-lock.json` file and removing the package name from `package.json` file and run command `npm install`

Removed `node_modules` and `package-lock.json` file



The screenshot shows the Visual Studio Code interface with the terminal tab active. The terminal window displays the command `PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>`, indicating that the working directory has been cleared of the `node_modules` and `package-lock.json` files.

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>
```

Removing the package name (`bootstrap`) from `package.json` file

```

{
  "name": "nodejs-tutorial",
  "version": "1.0.0",
  "description": "",
  "main": "4-intro.js",
  "scripts": {
    "start": "nodemon app.js"
  },
  "author": "sai krishna",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.21"
  },
  "devDependencies": {
    "nodemon": "^2.0.16"
  }
}

```

PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> **npm install**

Run command ***npm install*** (node_modules will be installed and package-lock.json will be added to project)

```

{
  "name": "nodejs-tutorial",
  "version": "1.0.0",
  "description": "",
  "main": "4-intro.js",
  "scripts": {
    "start": "nodemon app.js"
  },
  "author": "sai krishna",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.21"
  },
  "devDependencies": {
    "nodemon": "^2.0.16"
  }
}

```

PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> **npm install**

```

> nodemon@2.0.16 postinstall C:\Users\saikr\Documents\Node JS Tutorial Youtube\node_modules\nodemon
> node bin/postinstall || exit 0

npm notice created a lockfile as package-lock.json. You should commit this file.
npm warn optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\chokidar\node_modules\fsevents):
npm warn notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm warn nodejs-tutorial@1.0.0 No repository field.

added 117 packages from 56 contributors and audited 118 packages in 6.777s
16 packages are looking for funding

```

Nuclear Approach is followed in Gatsby applications.

Global Install

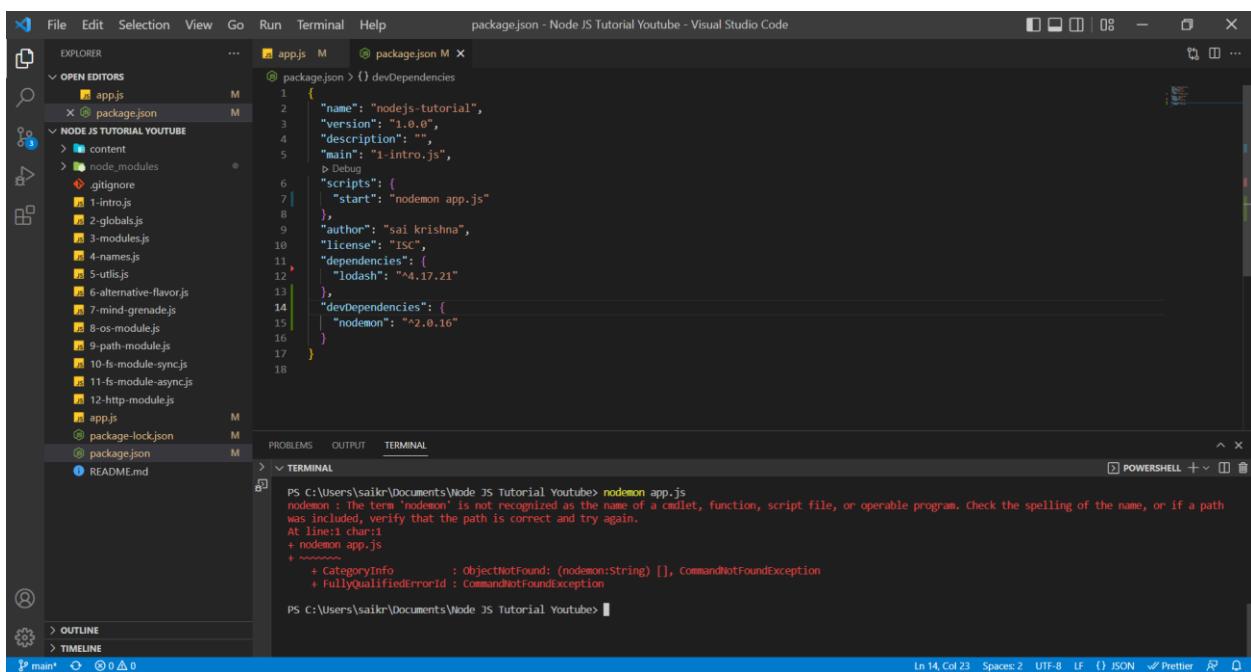
In a scenario, assume we have 20 nodal applications, and we are constantly working on node applications, so every time we can't go and change our application name in the scripts object in package.json file. To make our work easier we make nodemon package global, where we can run command **nodemon <file-name>** to run an application and restart the application whenever there is a change in the file system.

To Install a package globally, we run the command in the following way

npm install -g <file-name>

Ex: **npm install -g nodemon**

Before nodemon was installed globally, we get the error because it can't recognize it globally.

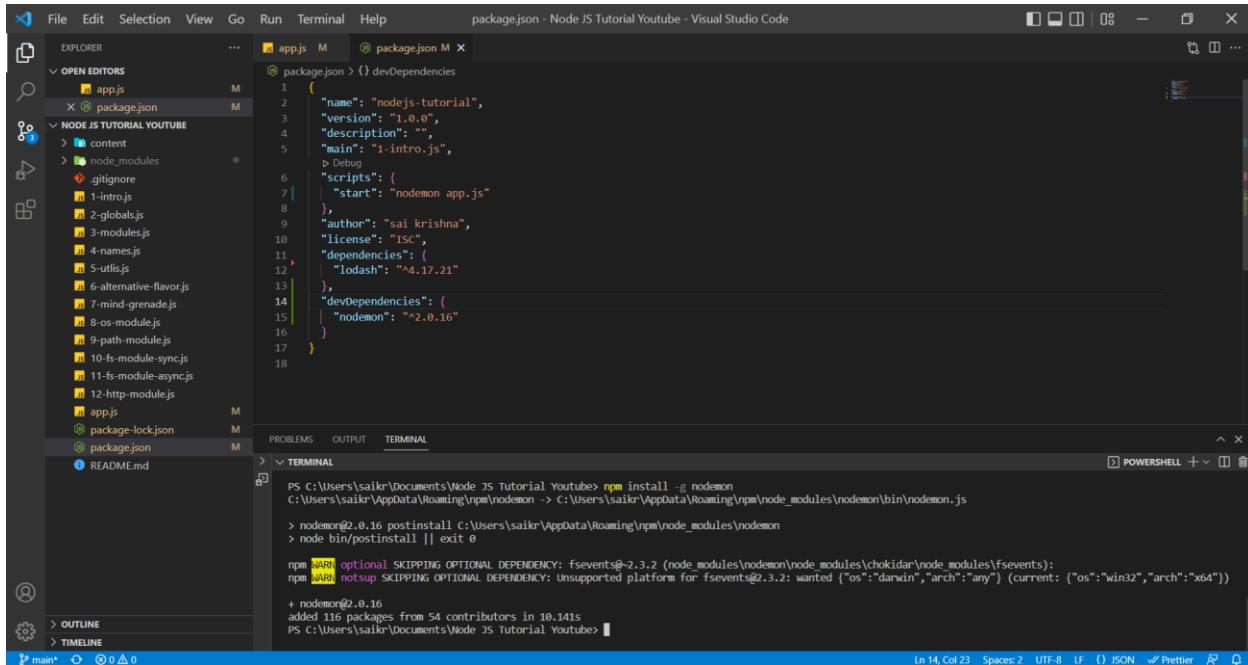


The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there are several files and folders: .gitignore, 1-intro.js, 2-globals.js, 3-modules.js, 4-names.js, 5-utilis.js, 6-alternative-flavor.js, 7-mind-grenade.js, 8-os-modules.js, 9-path-module.js, 10-fs-module-sync.js, 11-fs-module-async.js, 12-http-module.js, app.js, package-lock.json, package.json, and README.md. The package.json file is open in the main editor area, showing its contents:

```
1 {  
2   "name": "nodejs-tutorial",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "1-intro.js",  
6   "scripts": {  
7     "start": "nodemon app.js"  
8   },  
9   "author": "sai krishna",  
10  "license": "ISC",  
11  "dependencies": {  
12    "lodash": "^4.17.21"  
13  },  
14  "devDependencies": {  
15    "nodemon": "^2.0.16"  
16  }  
17}  
18
```

Below the editor, the Terminal tab is active, showing the command PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> nodemon app.js followed by an error message: nodemon : The term 'nodemon' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again. At line:1 char:1 + nodemon app.js + ~~~~~ + CategoryInfo : ObjectNotFound: (nodemon:String) [], CommandNotFoundException + FullyQualifiedErrorId : CommandNotFoundException

After installing globally,

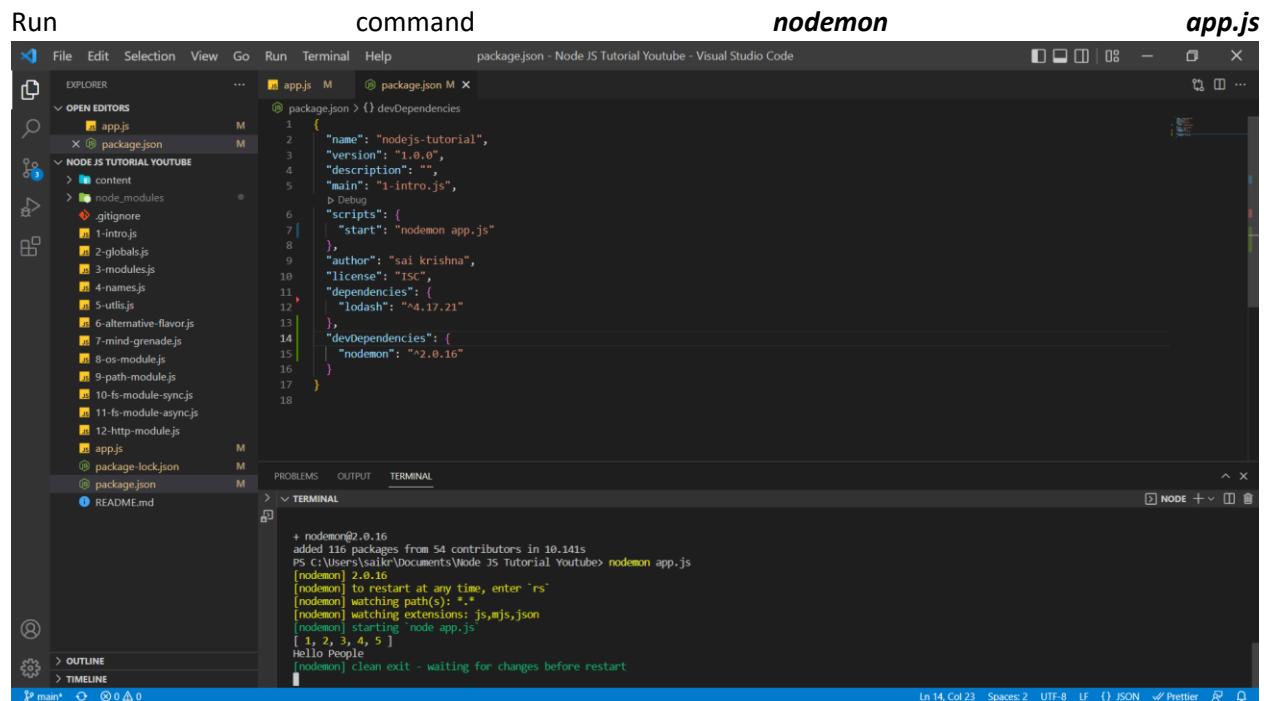


The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like app.js, package.json, and README.md.
- Editor:** Displays the package.json file content:

```
1 {
  "name": "nodejs-tutorial",
  "version": "1.0.0",
  "description": "",
  "main": "1-intro.js",
  "scripts": {
    "start": "nodemon app.js"
  },
  "author": "sai krishna",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.21"
  },
  "devDependencies": {
    "nodemon": "^2.0.16"
  }
}
```

- Terminal:** Shows the command `npm install -g nodemon` being run, followed by the output indicating the global installation of nodemon.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like app.js, package.json, and README.md.
- Editor:** Displays the package.json file content (same as above).
- Terminal:** Shows the command `npx nodemon app.js` being run, followed by the output indicating the local execution of nodemon on the app.js file.

Before installing the nodemon package globally, it won't be recognized in the terminal but when we installed it globally it can be recognized.

Earlier we used to install frontend libraries like react and Gatsby globally but when **npx** came into light, the route has shifted by installing a package globally to use **npx** command.

Ex: `npx create-react-app my-app`

npx stands for Node Package Execute and official name is Package Runner. It is a feature that is introduced in NPM 5.2. The main idea is to be running the CLI tool without installing the packages globally.

package-lock.json

If you remember some of the dependencies have their own dependencies like bootstrap which dependencies like jQuery and others etc. and those dependencies have versions as well.

Sometimes the dependencies of the packages may change as well by mistake when we handover code from developer to developer and it may introduce bugs into our application since there was change in package and dependencies versions.

Package-lock.json file contains all the versions of the packages and their dependencies as well.

```
"dependencies": {  
    "lodash": "^4.17.21"  
},
```

A package version has 3 values. This can be considered as contract between the person who is using the package and the person who developed.

The first number is the major change which means there are breaking changes. The second number is a minor one, which means backward compatible. If we change that to 18, we don't expect any breaking changes. The last one is patch for the bug fix. It is important to remember whenever we publish our own package.

Event Loops (A complex topic)

The event loop is what allows Node JS to perform non-blocking I/O operations – even though JavaScript is single threaded – by **offloading** operations to the system kernel whenever possible.

sync-js.js

The screenshot shows the Visual Studio Code interface with the title bar "sync-js.js - Node JS Tutorial Youtube - Visual Studio Code". The Explorer sidebar on the left lists files under "NODE JS TUTORIAL YOUTUBE" and "node_modules". The "OPEN EDITORS" tab shows two files: "sync-js.js" and "async-js.js". The "sync-js.js" editor contains the following code:

```
event-loop > sync-js.js > ...
1 // this piece of code is run on browser since we are using document which is DOM object
2 // Here JavaScript is Synchronous and Single Threaded
3 // It executes the next task only when it completes the second task (for loop) which is time consuming
4 console.log("first task");
5
6 console.time();
7 // we cannot offload for loops to the browser
8 for (let i = 0; i < 100000000; i++) {
9   const h3 = document.querySelector("h3");
10  h3.textContent = "Hey, Everyone is waiting on me";
11 }
12 console.timeEnd();
13
14 console.log("next task");
```

The terminal at the bottom shows the command "PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\event-loop>" followed by the output of the script execution.

async-js.js

The screenshot shows the Visual Studio Code interface with the title bar "async-js.js - Node JS Tutorial Youtube - Visual Studio Code". The Explorer sidebar on the left lists files under "NODE JS TUTORIAL YOUTUBE" and "node_modules". The "OPEN EDITORS" tab shows two files: "sync-js.js" and "async-js.js". The "async-js.js" editor contains the following code:

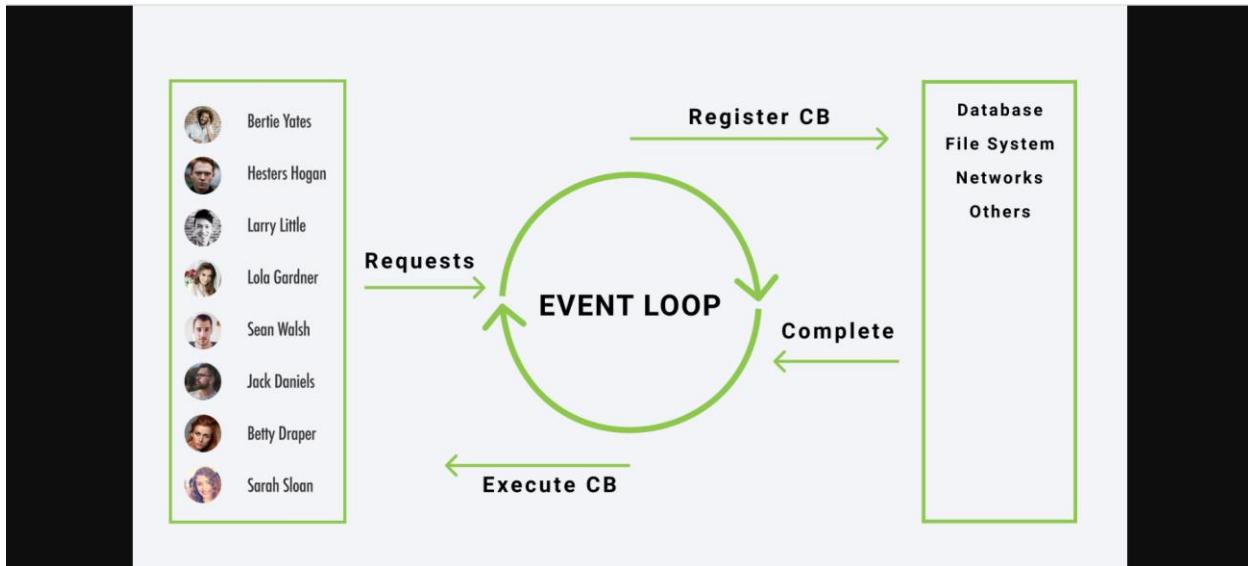
```
event-loop > sync-js.js > ...
1 // In browsers we use the API call, so we offload the task and execute the callback function when the task is completed
2 // using fetch is good example for the statement
3 // but still we can get an idea by using the below task
4 console.log("first task");
5
6 // even we set the time to be 0 seconds, callback will be executed only when the next task is completed
7 setTimeout(() => {
8   console.log("second task");
9 }, 0);
10
11 console.log("next task");
12
```

The terminal at the bottom shows the command "PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\event-loop> node async-js" followed by the output of the script execution.

By saying we cannot offload for loop, we can still write blocking code in JavaScript but the browser does provide some nice API's where we can offload the time consuming tasks.

Node JS Event Loop

Assume a scenario where our application is popular and there are 8 users who are requesting information from application and as the requests are coming in the event loop is responsible for avoiding this type of scenario.



In the above scenario, assume Larry Little requesting something from our application which is time consuming, so in this case event loop registers the callback function basically it registers what needs to be done when the task is complete because if event loop wouldn't do that then we would have the above scenario where the request are coming in and because Larry is requesting something that take a long time then rest of the users have to wait.

It's just a fact that we are wasting our time on waiting for the operation to be done and then only we can serve other users but what event loop does is it registers the callback and only when the operation is complete it executes the callback function.

Here the requests are coming in, let's say the operation is complete we first register the callback and operation is complete and instead of executing the callback right away it effectively puts away at the end of the line and then when there is no immediate code to run then we execute the callback.

With the help of Event Loop, we can offload some time-consuming operations and effectively just keep our other requests executed instead of wasting time for the time-consuming operation to complete.

Event Loop Example Codes

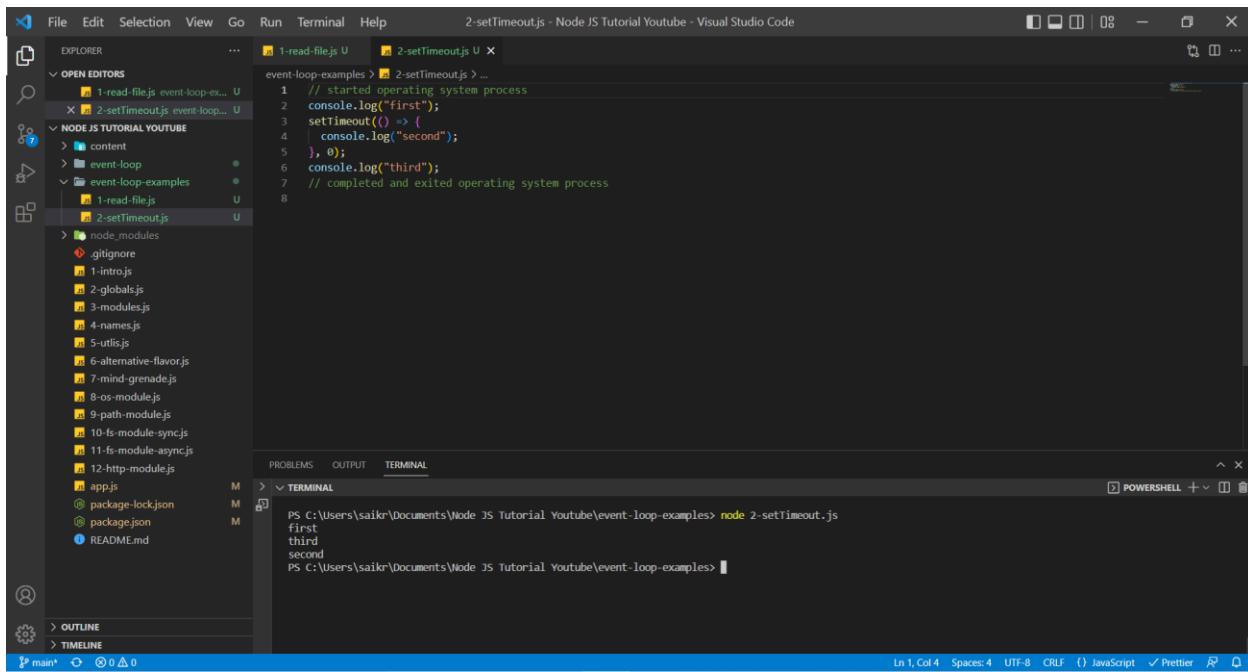
Here the Event Loop registers the callback function. Event Loop will offload this task (in this case, it will offload to file system). Event Loop will execute the next tasks and in the meanwhile if the readFile operation is complete (it may provide error or data) Event Loop puts that operation at end of the process/list and executes the remaining tasks. After all the tasks are executed and when there are no immediate lines to be executed then Event Loop executes the registered callback function. That is the reason you will see **result** and **completed first task** statement at the bottom of the screen.

1-read-file.js

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "NODE JS TUTORIAL YOUTUBE". The "event-loop-examples" folder contains several files: 1-read-file.js, 2-globals.js, 3-modules.js, 4-names.js, 5-util.js, 6-alternative-flavor.js, 7-mind-grenade.js, 8-os-module.js, 9-path-module.js, 10-fs-module-sync.js, 11-fs-module-async.js, 12-http-module.js, app.js, package-lock.json, package.json, and README.md.
- Editor:** The "1-read-file.js" file is open. The code reads a file named "first.txt" and logs its content to the console. It includes comments explaining the event loop behavior: starting tasks, offloading to the file system, and executing remaining tasks.
- Terminal:** The terminal window shows the command "node 1-read-file.js" being run, followed by the output of the program. The output shows the file content being read and then the message "Completed the first task".

2-setTimeout.js



```
// started operating system process
console.log("first");
setTimeout(() => {
  console.log("second");
}, 0);
console.log("third");
// completed and exited operating system process
```

PROBLEMS OUTPUT TERMINAL

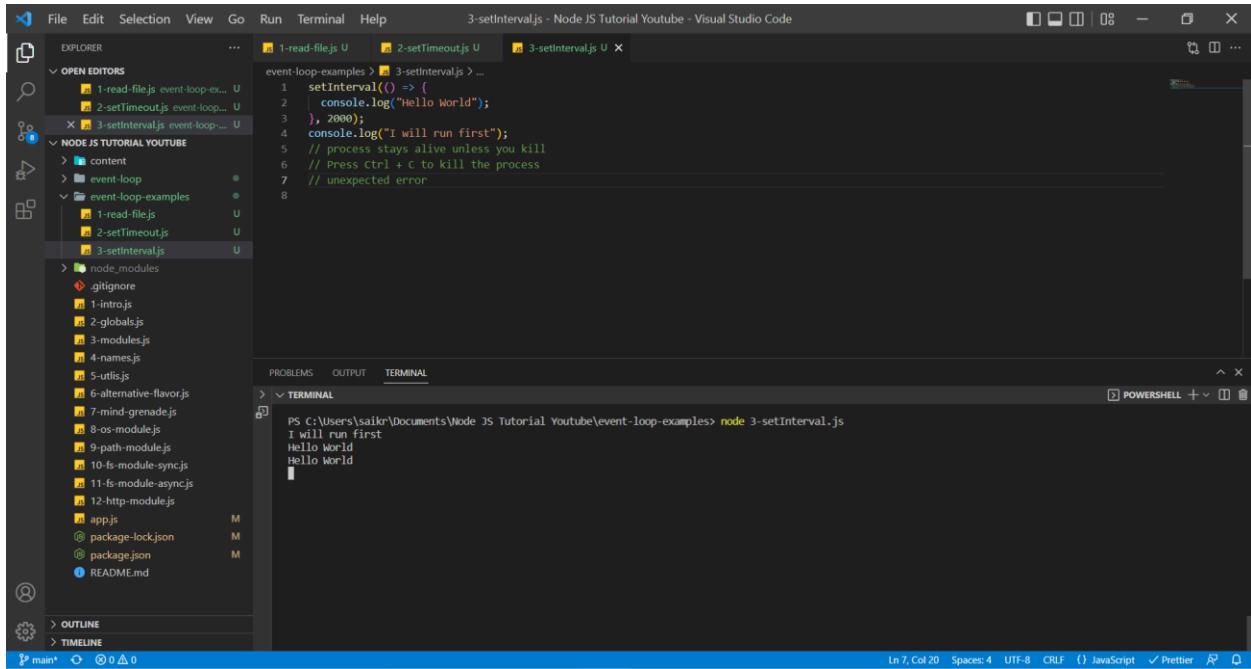
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\event-loop-examples> node 2-setTimeout.js

first
third
second

PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\event-loop-examples>

Even though `setTimeout` has 0 seconds (it is asynchronous code), the Event Loop will offload the task (in this case it will be offloaded to operating system) and execute the immediate tasks (synchronous code) and then there is no immediate task to execute then Event loop will execute the offloaded task (asynchronous code).

3-setInterval.js



```
event-loop-examples > 3-setInterval.js > ...
1 setinterval(() => {
2   | console.log("Hello World");
3 }, 2000);
4 console.log("I will run first");
5 // process stays alive unless you kill
6 // Press Ctrl + C to kill the process
7 // unexpected error
8
```

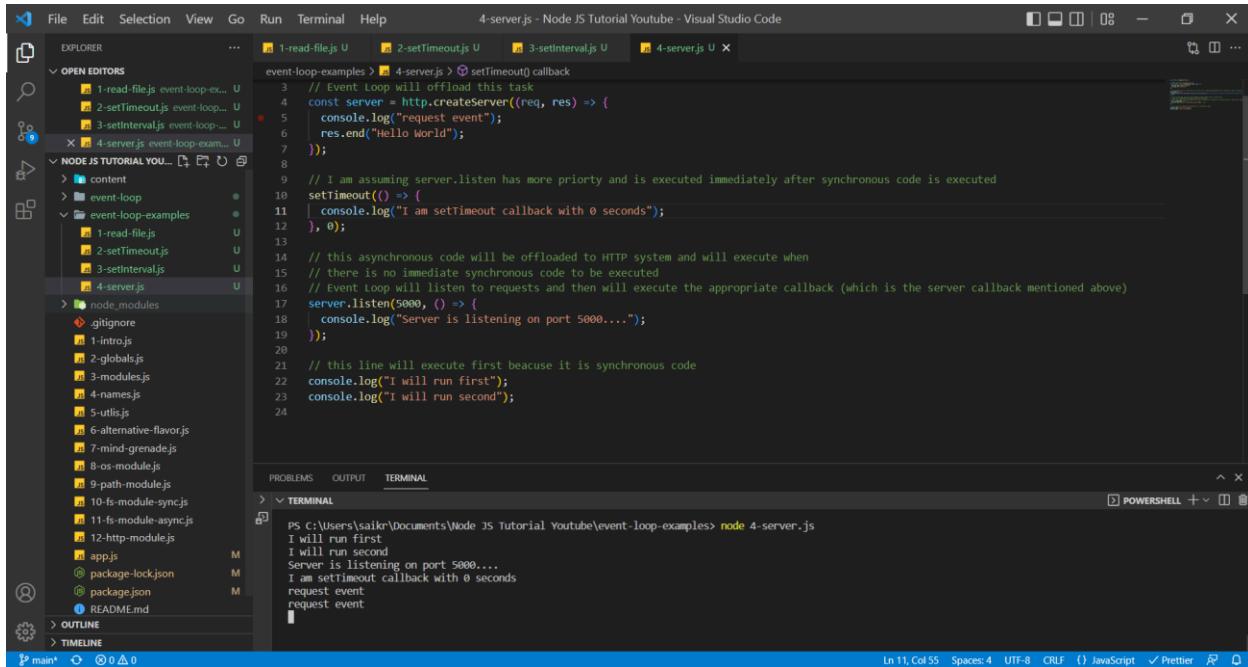
PROBLEMS OUTPUT TERMINAL

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\event-loop-examples> node 3-setInterval.js
I will run first
Hello World
Hello World
```

Ln 7, Col 20 Spaces: 4 UTF-8 CR LF { } JavaScript ✓ Prettier

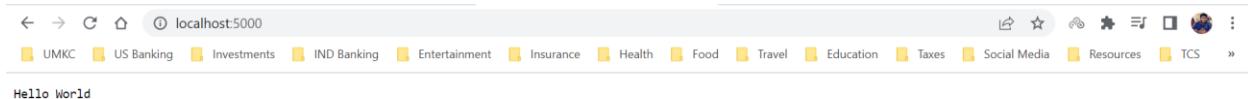
setInterval runs in intervals. So that's why we see the console statement after 2 every seconds. It will be offloaded by Event Loop and then executed when there is no immediate synchronous code.

4-server.js



```
event-loop-examples > 4-server.js > setTimeout() callback
3 // Event Loop will offload this task
4 const server = http.createServer((req, res) => {
5   console.log("request event");
6   res.end("Hello World");
7 });
8
9 // I am assuming server.listen has more priority and is executed immediately after synchronous code is executed
10 setTimeout(() => {
11   console.log("I am setTimeout callback with 0 seconds");
12 }, 0);
13
14 // this asynchronous code will be offloaded to HTTP system and will execute when
15 // there is no immediate synchronous code to be executed
16 // Event Loop will listen to requests and then will execute the appropriate callback (which is the server callback mentioned above)
17 server.listen(5000, () => {
18   console.log("Server is listening on port 5000....");
19 });
20
21 // this line will execute first because it is synchronous code
22 console.log("I will run first");
23 console.log("I will run second");
24
```

Request from Webpage

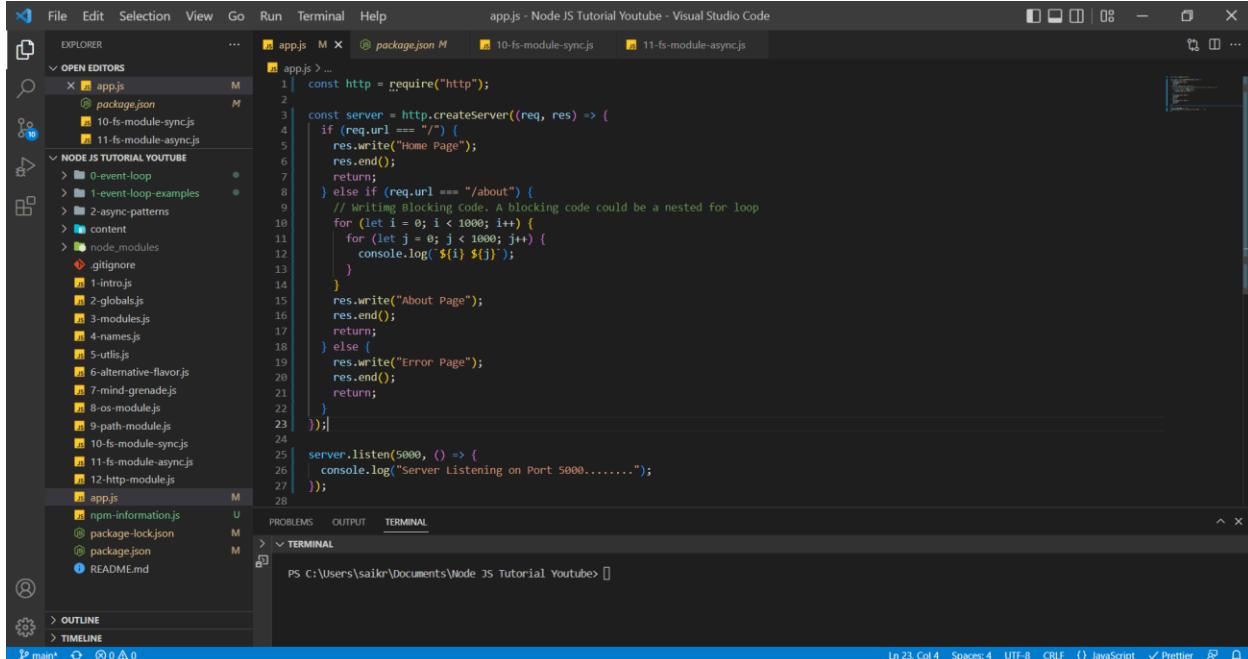


Here, we are starting the server and then listening to the server in port 5000 and whenever we listen a request from the server, we execute the appropriate callback function and return the response.

server.listen() process stays alive because listen is an asynchronous function and the moment, we set it up, Event Loop is just waiting for those requests to come in and once they come in Event Loop runs the appropriate callback function and sends the response back to the server. This process will end when we terminate or close the terminal.

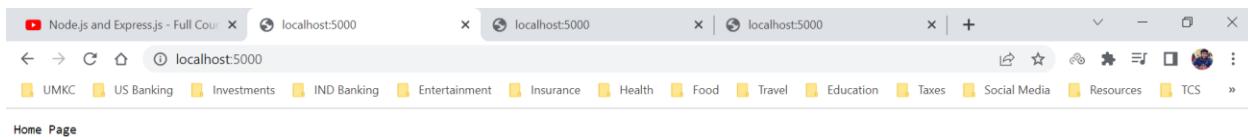
Async Patterns in Node JS

As we already discussed callback hell in File System Module. Callback Hell is nothing but implementing one callback function inside another callback function and going on. Callback Hell will mess up the code and hence we use different approaches to solve the code blocking in JavaScript.



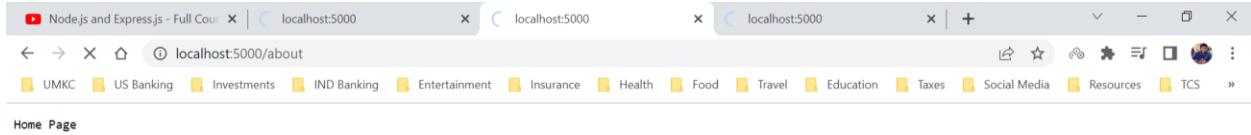
```
const http = require("http");
const server = http.createServer((req, res) => {
  if (req.url === "/") {
    res.write("Home Page");
    res.end();
    return;
  } else if (req.url === "/about") {
    // Writing Blocking code. A blocking code could be a nested for loop
    for (let i = 0; i < 1000; i++) {
      for (let j = 0; j < 1000; j++) {
        console.log(`${i} ${j}`);
      }
    }
    res.write("About Page");
    res.end();
    return;
  } else {
    res.write("Error Page");
    res.end();
    return;
  }
});
server.listen(5000, () => {
  console.log("Server Listening on Port 5000....");
});
```

In the above code, a server is created and listening to the port at 5000 and responds to the request for three URLs. We can login from multiple tabs and hit the server with a request and we will be served. This works fine until there is no blocking code, but once we have blocking code in any of the request block then we assume that only the user requesting URL but not. In real time, all the users request the server will be blocked since server is servicing the most time-consuming request which is in a synchronous code format.

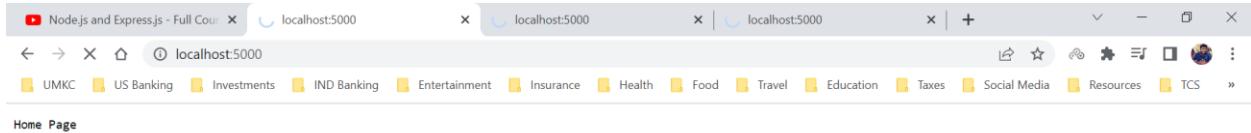


We have 3 different tabs or users requesting the server a response. Server is proving them with a response. In the code we have a blocking piece code for **localhost:5000/about** URL and when a user requests a response from that URL (`localhost:5000/about`) the requests from other users will also be blocked and response will be provided only that time-consuming request is completed.

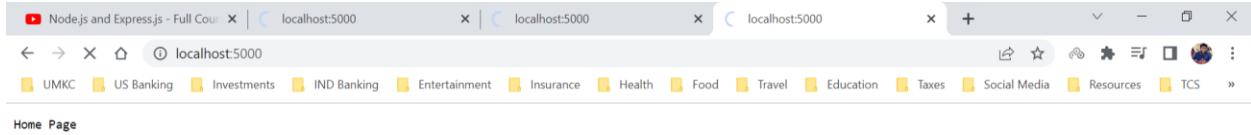
From User 2/ Tab 2 we are requesting About Page (which contains the blocking code)



After that, we requested Home Page from User 1/ Tab 1 and our page is still loading even though Home Page block doesn't contain any blocking code.



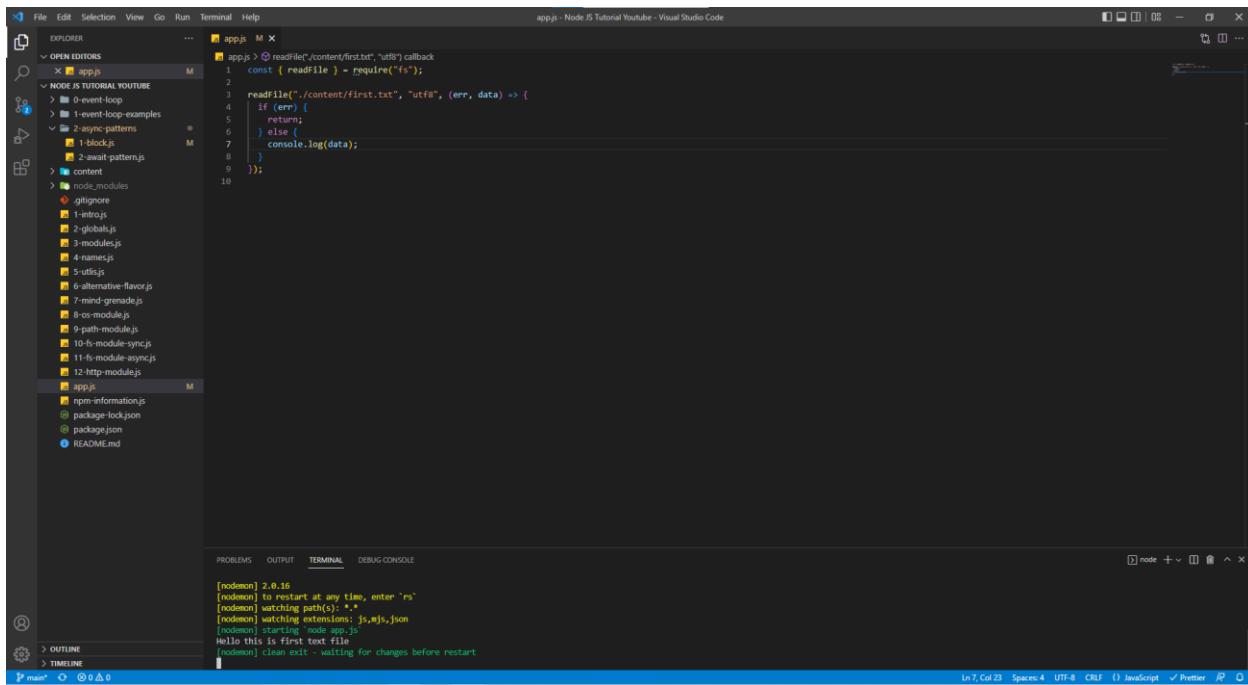
In the same way, User 3/ Tab 3 request Home Page but still page will be in loading state.



User 1 and User 3 request will be resolved only User 2 request is resolved.

This is the representation why prefer the asynchronous approach. We should always strive to set our code asynchronously.

Async Pattern – Promise Setup



The screenshot shows the Visual Studio Code interface with the following details:

- File Structure:** Explorer sidebar shows a project structure with files like `app.js`, `1-block.js`, `2-await-pattern.js`, etc.
- Code Editor:** The main editor window contains the following code for `app.js`:

```
const.readFile = require('fs').readFile;
readFile("./content/first.txt", "utf8", (err, data) => {
  if (err) {
    return;
  } else {
    console.log(data);
  }
});
```
- Terminal:** The terminal shows the output of running the script:

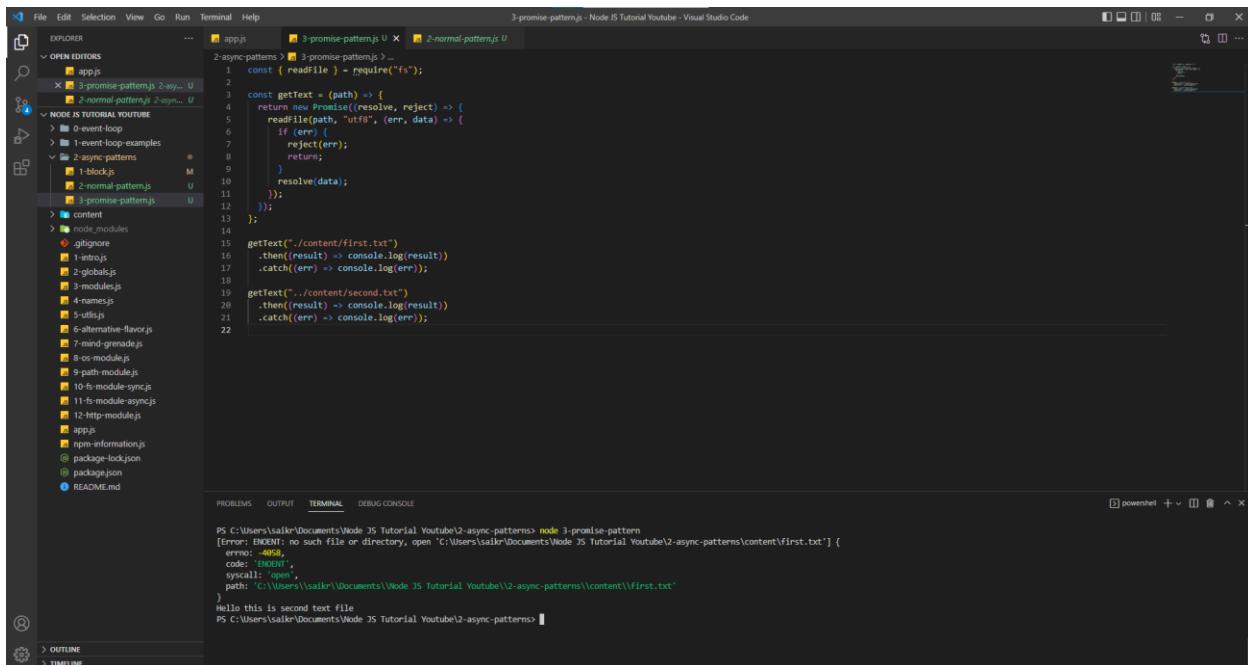
```
[nodemon] 2.0.16
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node ./app.js`
Hello this is first text file
[nodemon] clean exit - waiting for changes before restart
```
- Status Bar:** Shows the current file is `node`, line 7, column 23, with 23 spaces, and the encoding is UTF-8.

The above code works well but the real problem starts when we want to read multiple files and write data into one file.

The better solution would be turning that into a promise.

This is how an asynchronous function inside a promise looks like. Promise is divided in 3 stages which are

Pending, Reject and Resolve



The screenshot shows the Visual Studio Code interface with the following details:

- File Structure:** Explorer sidebar shows a project structure with files like `app.js`, `1-block.js`, `2-normal-pattern.js`, `3-promise-pattern.js`, etc.
- Code Editor:** The main editor window contains the following code for `3-promise-pattern.js`:

```
const readFile = require('fs');
const getText = (path) => {
  return new Promise((resolve, reject) => {
    readFile(path, "utf8", (err, data) => {
      if (err) {
        reject(err);
      } else {
        resolve(data);
      }
    });
  });
};

getText("./content/first.txt")
  .then(result => console.log(result))
  .catch(err => console.log(err));

getText("./content/second.txt")
  .then(result => console.log(result))
  .catch(err => console.log(err));
```
- Terminal:** The terminal shows the output of running the script:

```
PS C:\Users\salikr\Documents\Node JS Tutorial Youtube\2-async-patterns> node 3-promise-pattern
[Error: ENOENT: no such file or directory, open 'C:\Users\salikr\Documents\Node JS Tutorial Youtube\2-async-patterns\content\first.txt' ]
  errno: -4058,
  code: 'ENOENT',
  syscall: 'open',
  path: 'C:\Users\salikr\Documents\Node JS Tutorial Youtube\2-async-patterns\content\first.txt'
)
Hello this is second text file
PS C:\Users\salikr\Documents\Node JS Tutorial Youtube\2-async-patterns>
```
- Status Bar:** Shows the current file is `powershell`, line 7, column 23, with 23 spaces, and the encoding is UTF-8.

If Promise is fulfilled, it is Resolved. If it isn't fulfilled or failed, then it is considered as Reject. If it is in either of those two stages, then it would be in Pending State.

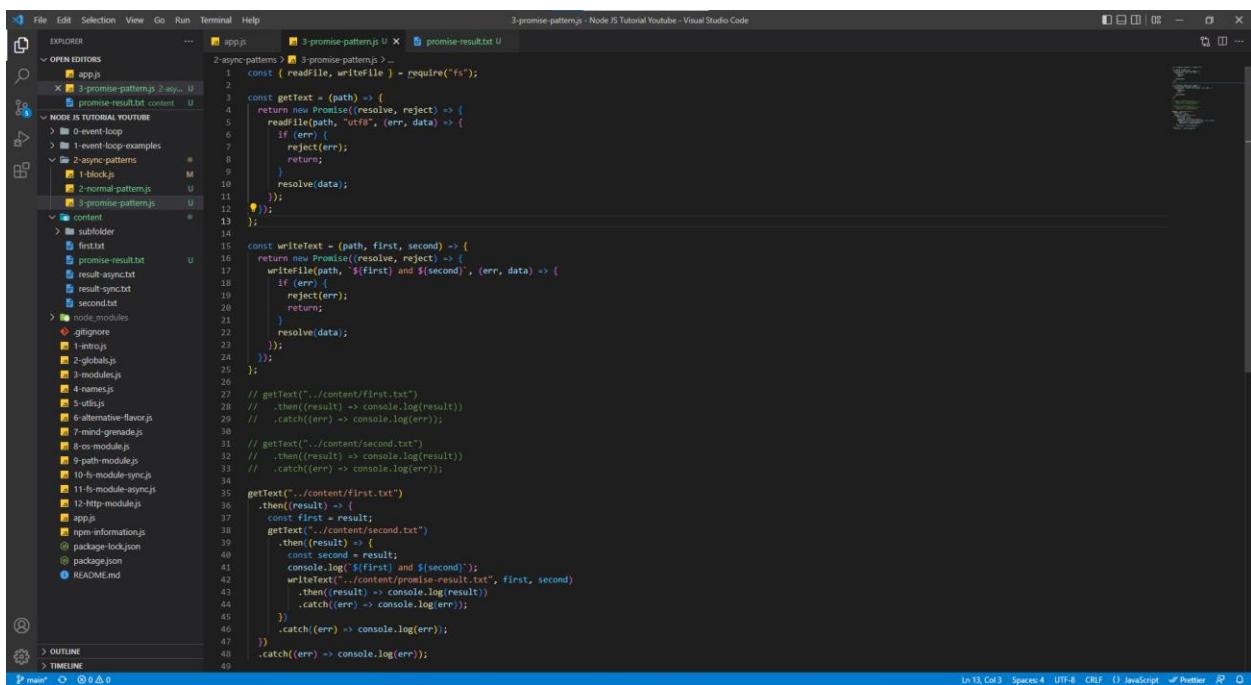
If a Promise is resolved it will enter then block, if it rejected it will enter catch block. The above screenshot depicts Resolved and Rejected states of Promise.

The above code is still not cleaner and that is why we follow different approach of setting up Async and Await.

If we want to do 2 file reads and 1 write still it would be messier because we need to chain promises one inside another, hence we use the approach Async and Await.

Chaining of Promises can be found below,

3-promise-pattern.js



```
const { readFile, writeFile } = require("fs");

const getText = (path) => {
  return new Promise((resolve, reject) => {
    readFile(path, "utf8", (err, data) => {
      if (err) {
        reject(err);
        return;
      }
      resolve(data);
    });
  });
};

const writeText = (path, first, second) => {
  return new Promise((resolve, reject) => {
    writeFile(path, `${first} and ${second}`, (err, data) => {
      if (err) {
        reject(err);
        return;
      }
      resolve(data);
    });
  });
};

// getText("../content/first.txt")
// .then(result) => console.log(result)
// .catch(err) => console.log(err);

// getText("../content/second.txt")
// .then(result) => console.log(result)
// .catch(err) => console.log(err);

// getText("../content/first.txt")
// .then(result) => {
//   const First = result;
//   getText("../content/second.txt")
//     .then(result) => {
//       const Second = result;
//       console.log(`${First} and ${Second}`);
//       writeText("../content/promise-result.txt", first, second)
//         .then(result) => console.log(result)
//         .catch(err) => console.log(err);
//     }
//   .catch(err) => console.log(err);
// }
// .catch(err) => console.log(err);

// console.log(`Result: ${result}`);
```

and the result can be seen in terminal and promise-result.txt file

terminal

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists various files and folders related to a 'Node JS Tutorial YouTube' project. In the center, the code editor displays '3-promise-pattern.js'. The terminal at the bottom shows the following output:

```
PS C:\Users\saikir\Documents\Node JS Tutorial YouTube\2-async-patterns> node 3-promise-pattern
Hello this is first text file and Hello this is second text file
undefined
```

promise-result.txt

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists various files and folders related to a 'Node JS Tutorial YouTube' project. In the center, the code editor displays 'promise-result.txt'. The terminal at the bottom shows the following output:

```
PS C:\Users\saikir\Documents\Node JS Tutorial YouTube\2-async-patterns> node 3-promise-pattern
Hello this is first text file and Hello this is second text file
undefined
```

Async Patterns – Refactor to Async

Now we are refactoring the code using the `async` and `await`. With the help of `async` and `await`, we will wait till the promise is settled and then decide whatever we want to do.

The screenshot shows the Visual Studio Code interface with two files open: `3-promise-pattern.js` and `4-async-await-pattern.js`. The `4-async-await-pattern.js` file contains the following code:

```
const start = async () => {
  try {
    const first = await getText("../content/first.txt");
    console.log(first);
  } catch (err) {
    console.log(err);
  }
};

start();
```

The terminal below shows the output of running the script:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\2-async-patterns> node 4-async-await-pattern
Hello this is first text file
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\2-async-patterns>
```

In the `async await` approach we wait for the promise to be settled and then follow the next steps. We always try to write code in `try` and `catch` blocks to capture any unhandled exceptions.

If the file we are trying to read is not found then we will handle the exception using the `catch` block.

The screenshot shows the Visual Studio Code interface with the same two files. The `4-async-await-pattern.js` file now includes a `catch` block:

```
const start = async () => {
  try {
    const first = await getText("../content/first.txt");
    console.log(first);
  } catch (err) {
    console.log(err);
  }
};

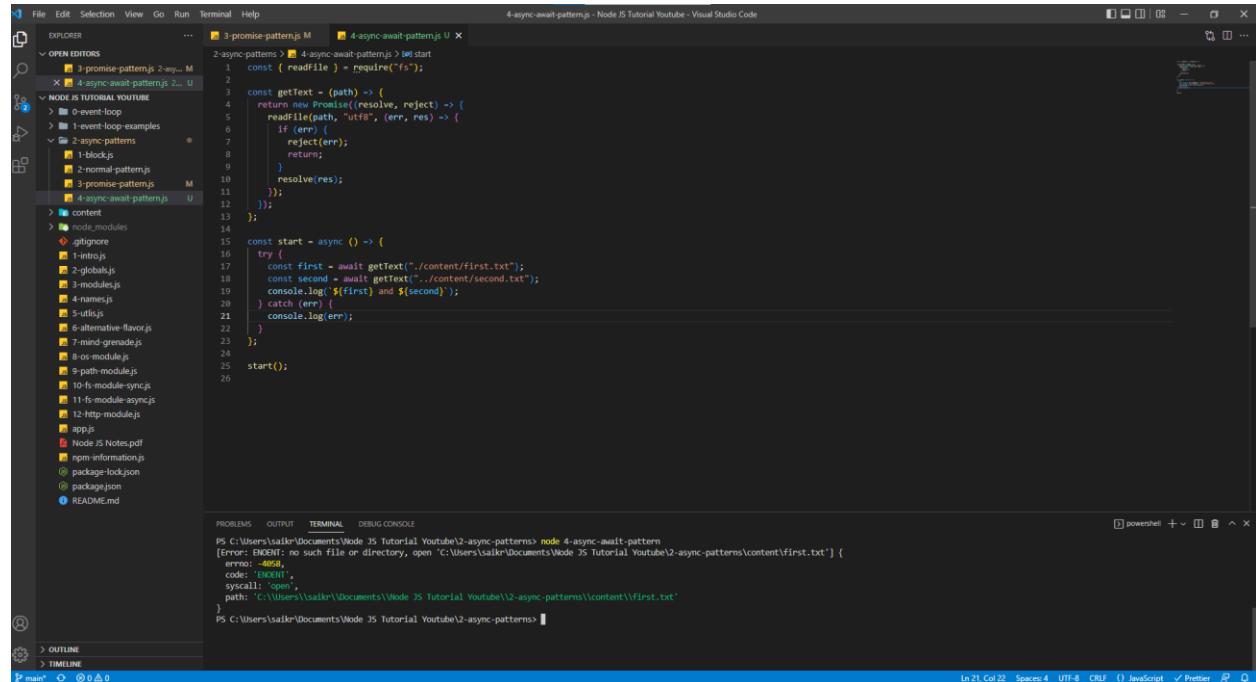
start();
```

The terminal output shows an error for a missing file:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\2-async-patterns> node 4-async-await-pattern
[Error: ENOENT: no such file or directory, open 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\2-async-patterns\content\first.txt']
  errno: -4098,
  code: 'ENOENT',
  syscall: 'open',
  path: 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\2-async-patterns\content\first.txt'
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\2-async-patterns>
```

Using try and catch blocks, if any one promise is rejected then try block won't be execute and it will immediately enter the catch block to handle the error.

When there is an error in first file path.



The screenshot shows the Visual Studio Code interface with two open files: '3-promise-pattern.js' and '4-async-await-pattern.js'. The '4-async-await-pattern.js' file contains the following code:

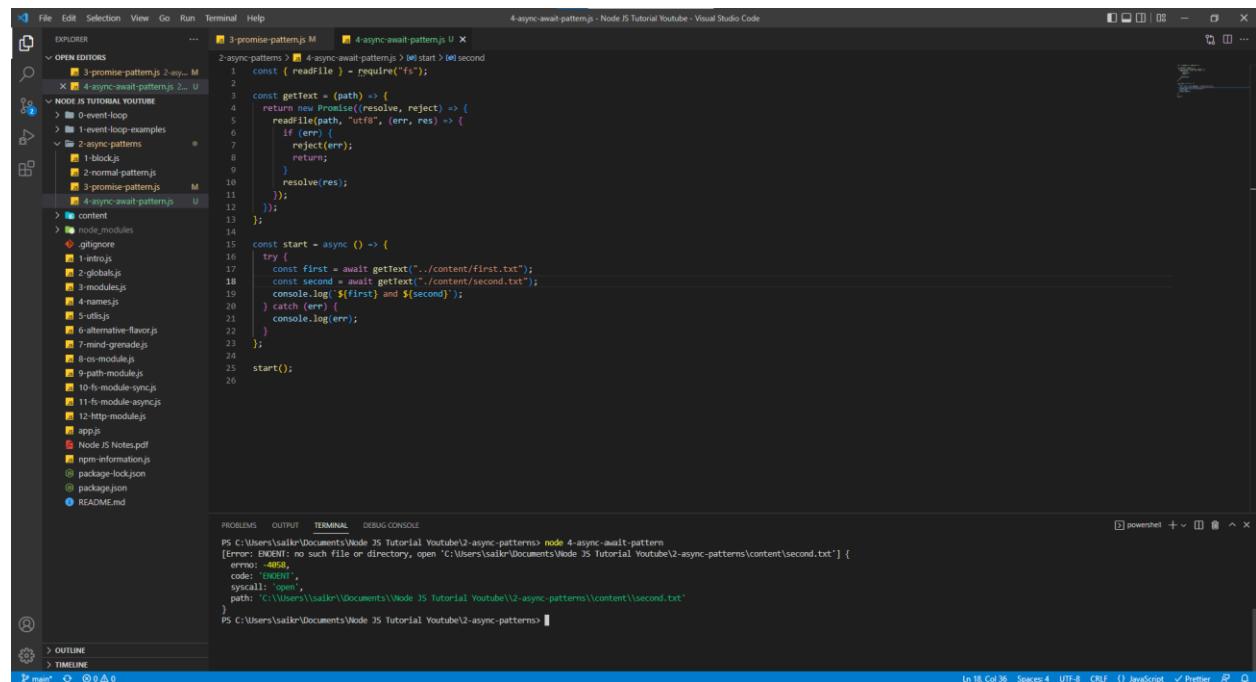
```
const start = async () => {
  try {
    const first = await getText("./content/first.txt");
    const second = await getText("./content/second.txt");
    console.log(`${first} and ${second}`);
  } catch (err) {
    console.log(err);
  }
};

start();
```

The terminal window shows an error message:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\2.async-patterns> node 4-async-await-pattern
[Error: ENOENT: no such file or directory, open 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\2.async-patterns\content\first.txt'] {
  errno: -4058,
  code: 'ENOENT',
  syscall: 'open',
  path: 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\2.async-patterns\content\first.txt'
}
```

When there is an error in second file path



The screenshot shows the Visual Studio Code interface with two open files: '3-promise-pattern.js' and '4-async-await-pattern.js'. The '4-async-await-pattern.js' file contains the same code as the previous screenshot:

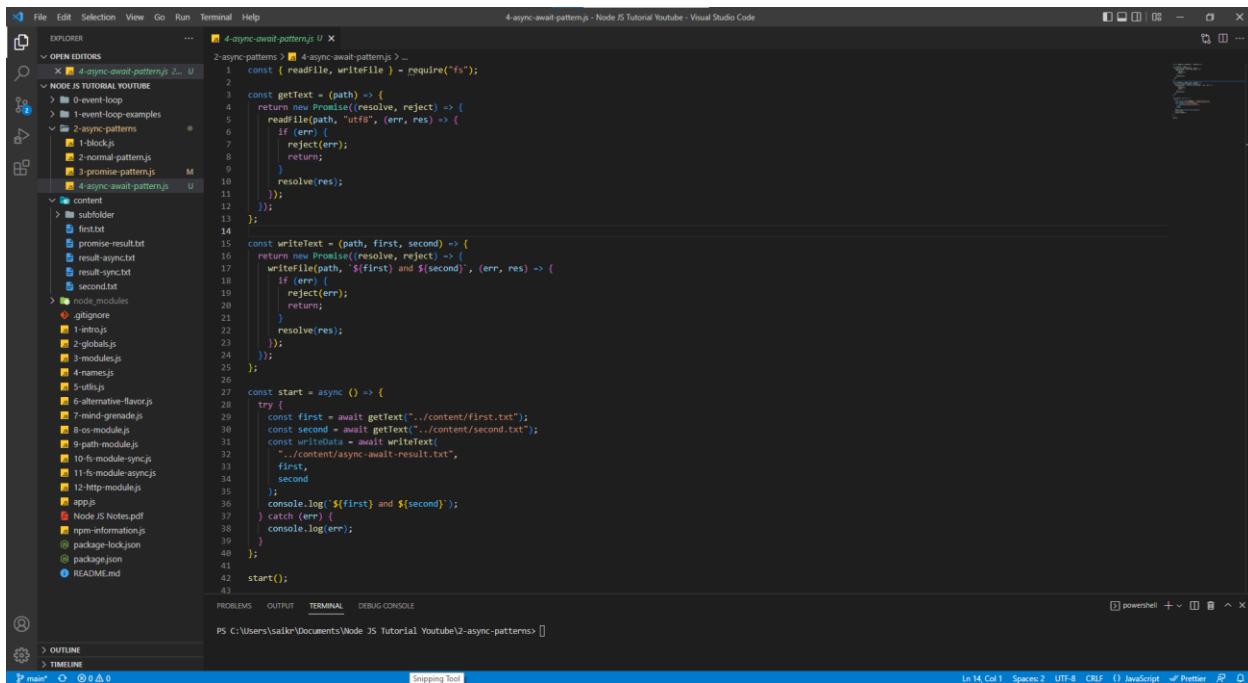
```
const start = async () => {
  try {
    const first = await getText("./content/first.txt");
    const second = await getText("./content/second.txt");
    console.log(`${first} and ${second}`);
  } catch (err) {
    console.log(err);
  }
};

start();
```

The terminal window shows an error message:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\2.async-patterns> node 4-async-await-pattern
[Error: ENOENT: no such file or directory, open 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\2.async-patterns\content\second.txt'] {
  errno: -4058,
  code: 'ENOENT',
  syscall: 'open',
  path: 'C:\Users\saikr\Documents\Node JS Tutorial Youtube\2.async-patterns\content\second.txt'
}
```

With the help of `async` and `await` pattern, this is how we implement two reads and one write

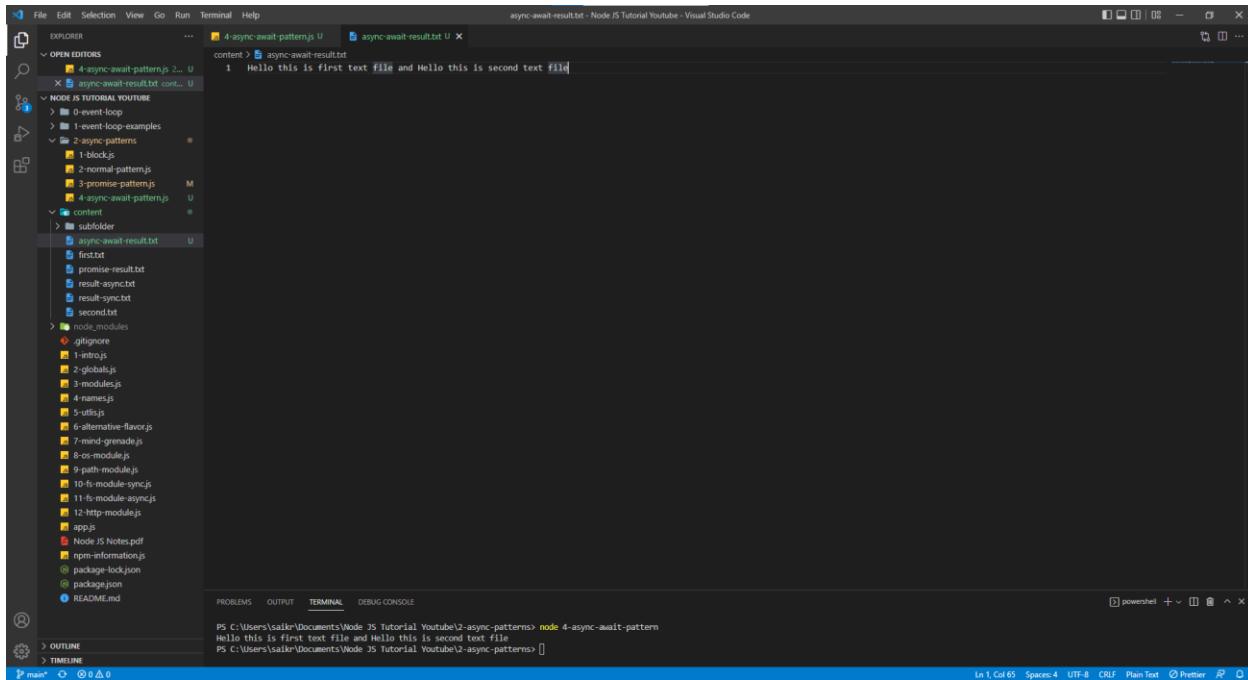


```

File Edit Selection View Go Run Terminal Help
OPEN EDITORS 4-async-await-pattern.js U ...
NODE JS TUTORIAL YOUTUBE
0-event-loop
1-event-loop-examples
2-async-patterns
1-block.js
2-normal-pattern.js
3-promise-pattern.js M
4-async-await-pattern.js U
content
subfolder
first.txt
promise-result.txt
result-async.txt
result-sync.txt
second.txt
node_modules
.gitignore
1-intro.js
2-global.js
3-modules.js
4-names.js
5-utils.js
6-alternative-flavor.js
7-mind-grenade.js
8-os-module.js
9-path-module.js
10-fs-module-sync.js
11-fs-module-async.js
12-http-module.js
app.js
Node JS Notes.pdf
npm-information.js
package-lock.json
package.json
README.md
4-async-await-pattern.js U ...
2-async-patterns > 4-async-await-pattern.js ...
const (readfile, writefile) = require("fs");
const getText = (path) => {
  return new Promise((resolve, reject) => {
    readfile(path, "utf8", (err, res) => {
      if (err) {
        reject(err);
        return;
      }
      resolve(res);
    });
  });
};
const writeText = (path, first, second) => {
  return new Promise((resolve, reject) => {
    writefile(path, `${first} and ${second}`, (err, res) => {
      if (err) {
        reject(err);
        return;
      }
      resolve(res);
    });
  });
};
const start = async () => {
  try {
    const first = await getText("../content/first.txt");
    const second = await getText("../content/second.txt");
    const writeData = await writeText(
      "../content/async-await-result.txt",
      first,
      second
    );
    console.log(`${first} and ${second}`);
  } catch (err) {
    console.log(err);
  }
};
start();

```

`async-await-result.txt`



```

File Edit Selection View Go Run Terminal Help
OPEN EDITORS 4-async-await-pattern.js U ...
NODE JS TUTORIAL YOUTUBE
0-event-loop
1-event-loop-examples
2-async-patterns
1-block.js
2-normal-pattern.js
3-promise-pattern.js M
4-async-await-pattern.js U
content
subfolder
async-await-result.txt U ...
1 Hello this is first text file and Hello this is second text file
4-async-await-pattern.js U ...
content > 4-async-await-result.txt ...
1 Hello this is first text file and Hello this is second text file

```

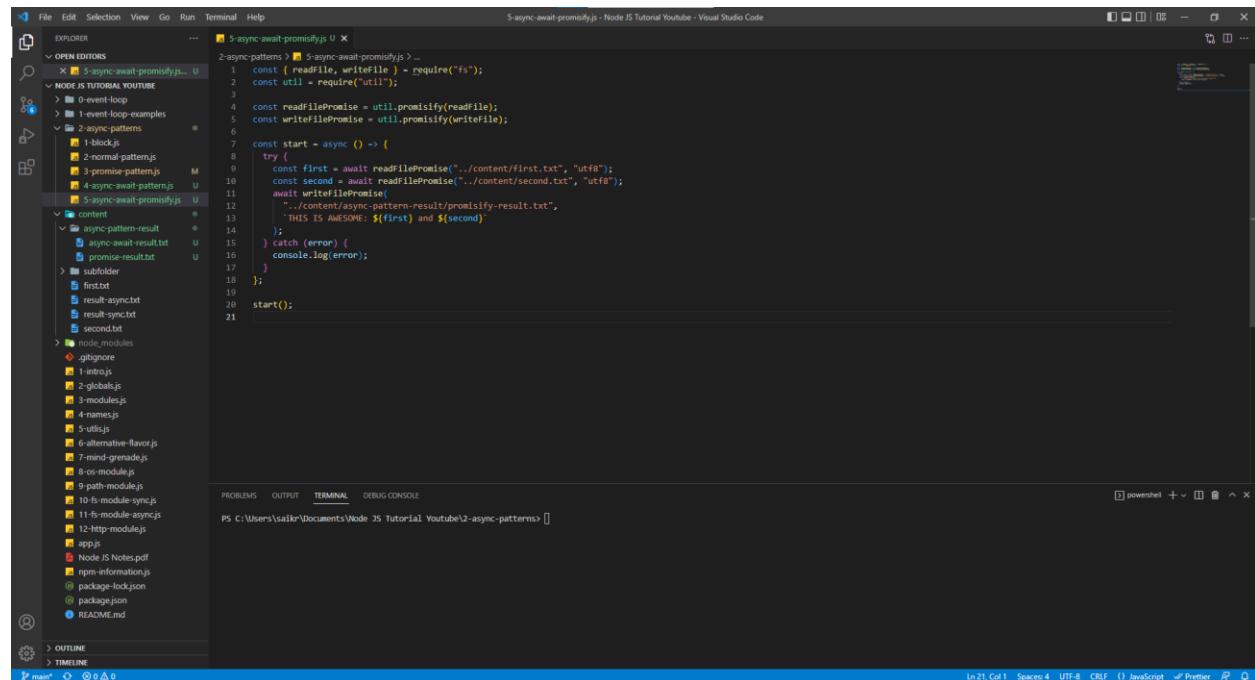
Till now we have used to wrap the function (`getText` and `writeText`) with Promises and worked with them using `then()` and `catch()` blocks (which are nothing but a promise pattern), `async` and `await` pattern. Now we will try to remove the wrapping function ad use the Node JS Native pattern using util module in Node.

Async Patterns – Node's Native Option

Using the node native option, we have Util module in Node which provides the `promisify` function which in turn provides the promise state (resolved, rejected, or pending). This approach rapidly reduces the line of code to be written and provides a cleaner code.

Using Promisify

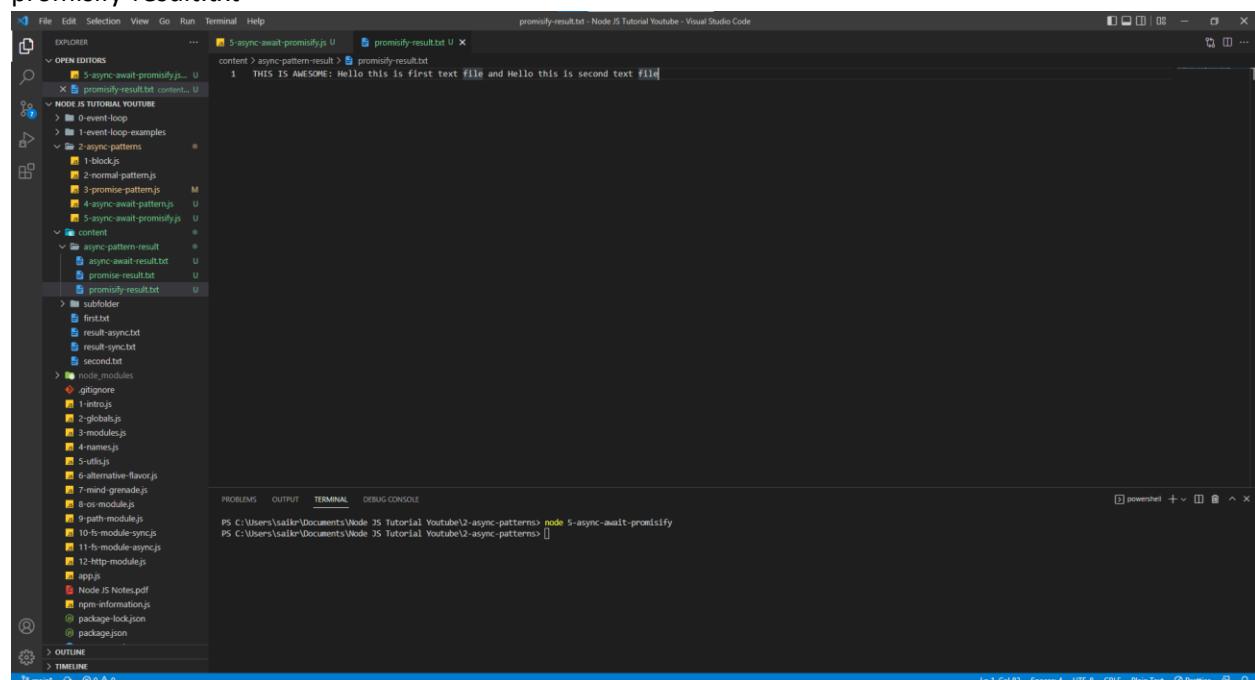
5-async-await-promisify.js



```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 5-async-await-promisify.js U ...
NODE JS TUTORIAL YOUTUBE
0-event-loop
1-event-loop-examples
2-async-patterns
1-block.js
2-normal-pattern.js
3-promise-pattern.js M
4-async-await-pattern.js U
5-sync-await-promisify.js U
content
  1-first.txt
  2-second.txt
  result-async.txt
  result-sync.txt
  second.txt
node_modules
  .gitignore
  1-intro.js
  2-global.js
  3-module.js
  4-names.js
  5-utils.js
  6-alternative-flavor.js
  7-mind-grenade.js
  8-os-module.js
  9-path-module.js
  10-fs-module-sync.js
  11-fs-module-async.js
  12-http-module.js
  app.js
  Node JS Notes.pdf
  npm-information.js
  package-lock.json
  package.json
  README.md
OUTLINE
TIMELINE
5-async-await-promisify.js 5-async-await-promisify.js ...
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\2-async-patterns> [powershell + v]
In 21, Col 1  Spaces:4  UTF-8  CRLF  ( ) JavaScript  ⚡ Prettier  ⚡
TERMINAL
In 21, Col 1  Spaces:4  UTF-8  CRLF  Plain Text  ⚡ Prettier  ⚡
```

```
2-async-patterns > 5-async-await-promisify.js > ...
2-async-patterns > 5-async-await-promisify.js - require('fs');
1 const { readFile, writeFile } = require('fs');
2 const util = require('util');
3
4 const readFilePromise = util.promisify(readFile);
5 const writeFilePromise = util.promisify(writeFile);
6
7 const start = async () => {
8   try {
9     const first = await readFilePromise("../content/first.txt", "utf8");
10    const second = await readFilePromise("../content/second.txt", "utf8");
11    await writeFilePromise(`../content/async-pattern-result/promisify-result.txt`,
12      `THIS IS AWESOME: ${first} and ${second}`);
13  } catch (error) {
14    console.log(error);
15  }
16}
17
18;
19
20 start();
21
```

promisify-result.txt



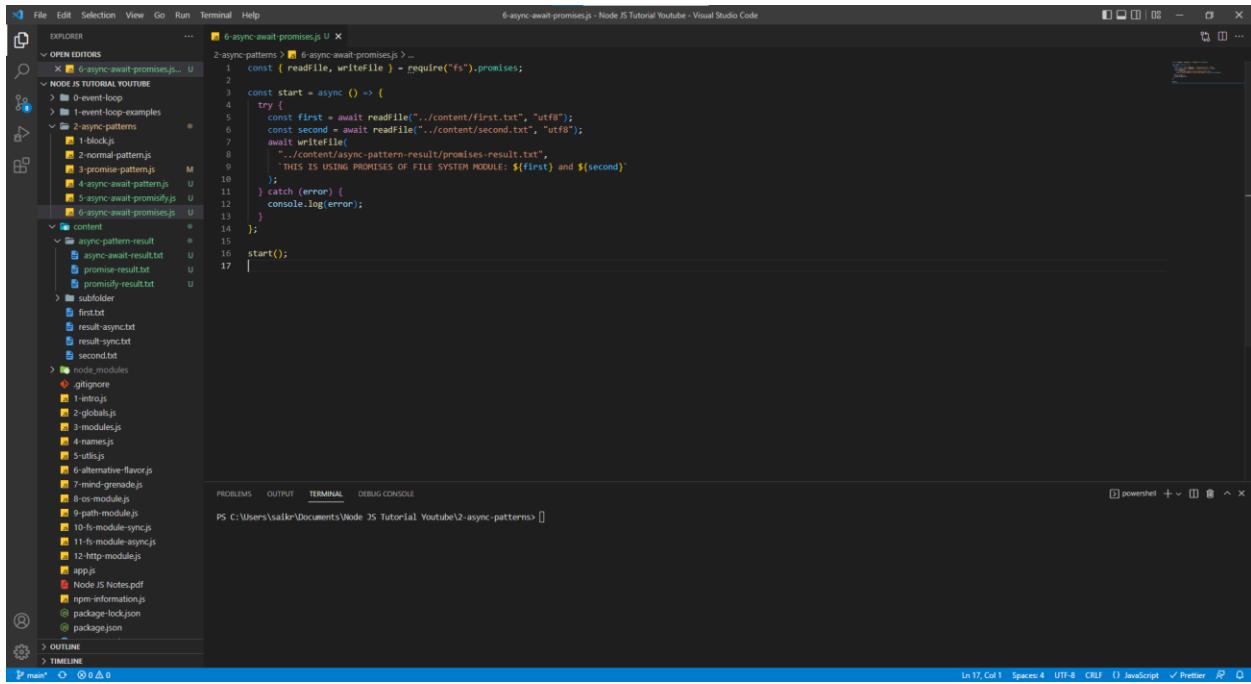
```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS 5-async-await-promisify.js U  promisify-result.txt U ...
NODE JS TUTORIAL YOUTUBE
0-event-loop
1-event-loop-examples
2-async-patterns
1-block.js
2-normal-pattern.js
3-promise-pattern.js M
4-async-await-pattern.js U
5-sync-await-promisify.js U
content
  1-first.txt
  2-second.txt
  result-async.txt
  result-sync.txt
  second.txt
  promisify-result.txt
node_modules
  .gitignore
  1-intro.js
  2-global.js
  3-module.js
  4-names.js
  5-utils.js
  6-alternative-flavor.js
  7-mind-grenade.js
  8-os-module.js
  9-path-module.js
  10-fs-module-sync.js
  11-fs-module-async.js
  12-http-module.js
  app.js
  Node JS Notes.pdf
  npm-information.js
  package-lock.json
  package.json
  README.md
OUTLINE
TIMELINE
5-async-await-promisify.js 5-async-await-promisify.js ...
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\2-async-patterns> node 5-async-await-promisify
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube\2-async-patterns> [powershell + v]
In 1, Col 82  Spaces:4  UTF-8  CRLF  Plain Text  ⚡ Prettier  ⚡
TERMINAL
In 1, Col 82  Spaces:4  UTF-8  CRLF  Plain Text  ⚡ Prettier  ⚡
```

```
content > async-pattern-result > 5-async-await-promisify.js
1 THIS IS AWESOME: Hello this is first text file and Hello this is second text file
```

Using Promises of File System Module in Node

We can remove the `promisify` function of Util Module and just use the promises available in File System Module. The code looks much cleaner.

6-async-await-promises.js

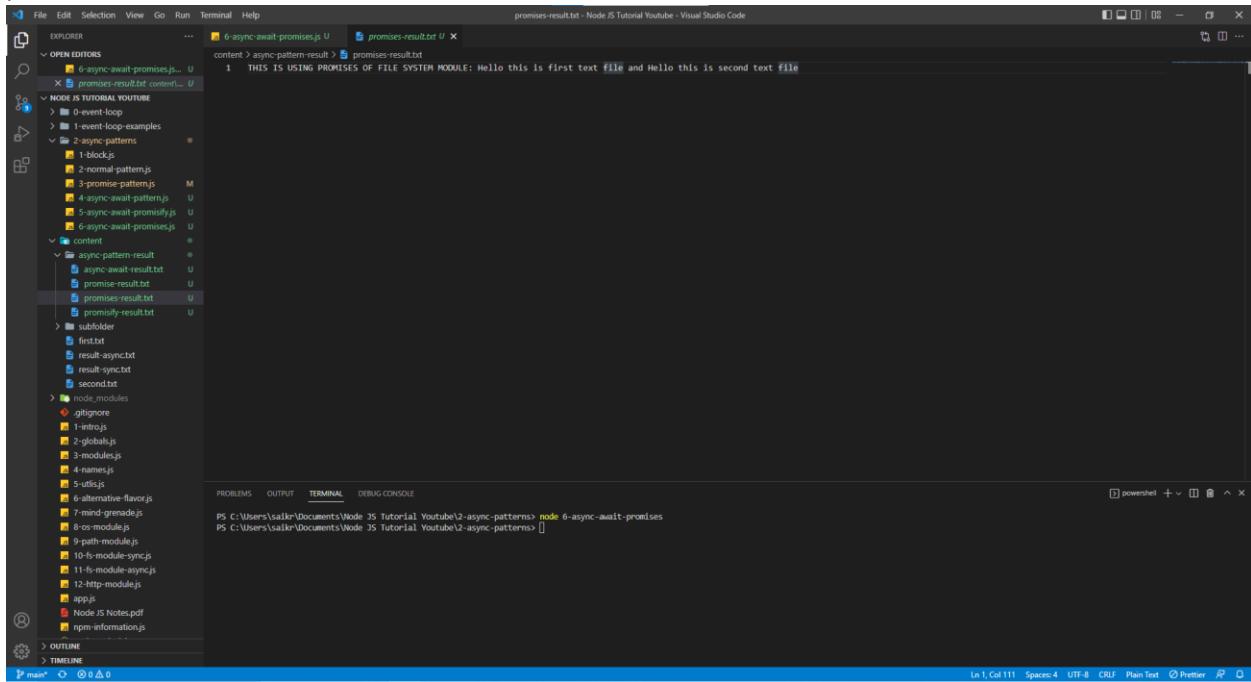


The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "6-async-await-promises.js".
- Editor:** Displays the code for `6-async-await-promises.js`. The code uses `fs.promises` to read files and write to a result file.
- Terminal:** Shows the command `node 6-async-await-promises` being run.
- Output:** Shows the terminal output where the program reads from `first.txt` and `second.txt` and writes to `promises-result.txt`.

```
const { readfile, writefile } = require("fs").promises;
const start = async () => {
  try {
    const first = await readfile("../content/first.txt", "utf8");
    const second = await readfile("../content/second.txt", "utf8");
    await writefile("../content/promises-result.txt",
      `THIS IS USING PROMISES OF FILE SYSTEM MODULE: ${first} and ${second}`);
  } catch (error) {
    console.log(error);
  }
};
start();
```

promises-result.txt



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "6-async-await-promises.js".
- Editor:** Displays the contents of `promises-result.txt`, which contains the text "THIS IS USING PROMISES OF FILE SYSTEM MODULE: Hello this is first text file and Hello this is second text file".
- Terminal:** Shows the command `node 6-async-await-promises` being run.
- Output:** Shows the terminal output where the program reads from `first.txt` and `second.txt` and writes to `promises-result.txt`.

```
THIS IS USING PROMISES OF FILE SYSTEM MODULE: Hello this is first text file and Hello this is second text file
```

Events

- Event Driven Programming
- Used Heavily in Node JS

In browser applications, events are emitted through actions performed by a user on web page like click on a button or hovering etc., Essentially our program executes at least in part it is controlled by events. In browser apps those events are mostly external. Now that style of programming is called Event Driven Programming.

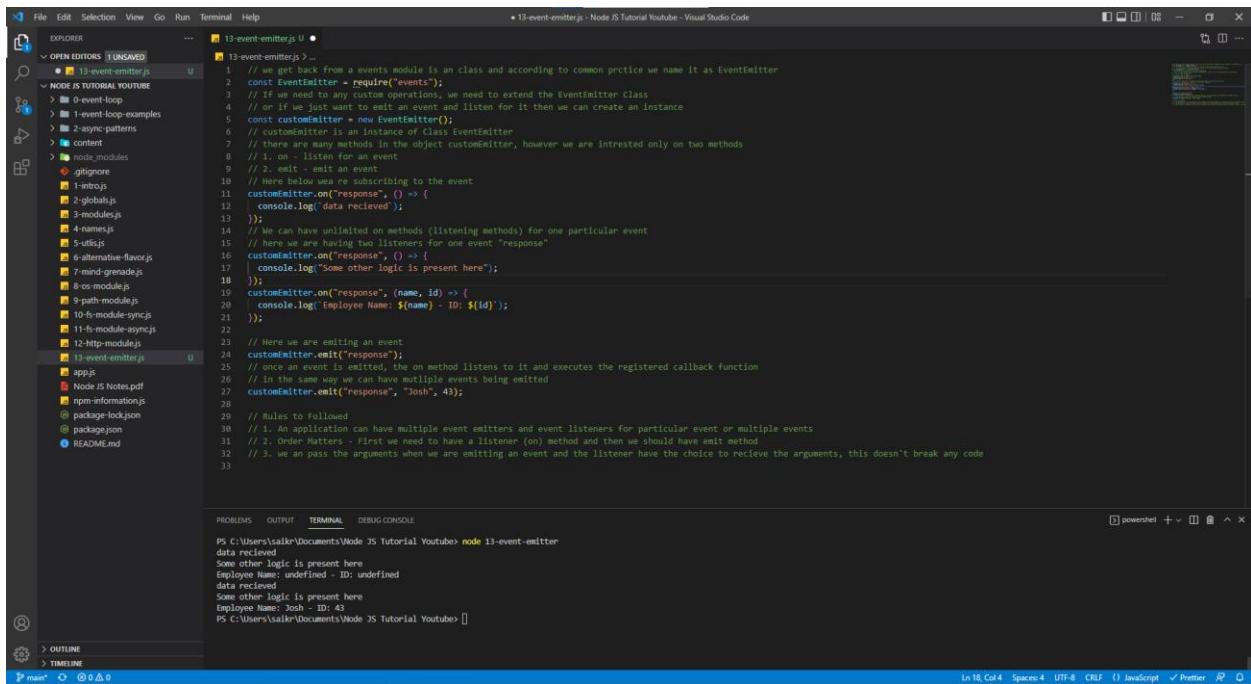
Server-side applications also use events and Event Driven Programming is heavily used in Node JS (Server Side).

Basically, we listen to specific events and register functions that will execute in response to those events. So, once an event takes place, a callback function is fired.

Many built-in modules in Node JS do use events under the hood and therefore making events an event driven programming a big part of Node JS.

Even though we don't write our own events, events are core building block of Node JS. Lot of built-in modules rely on Events.

13-event-emitter.js



```
File Edit Selection View Go Run Terminal Help
EXPLORER 13-event-emitter.js U
OPEN EDITORS 13-event-emitter.js U
NODE JS TUTORIAL YOUTUBE
  1-event-loop
  1-event-loop-examples
  2-async-patterns
  content
  node_modules
    .gitignore
    1-intro.js
    2-globals.js
    3-modules.js
    4-names.js
    5-utils.js
    6-alternative-flavor.js
    7-mind-grenade.js
    8-os-module.js
    9-path-module.js
    10-fs-module-sync.js
    11-fs-module-async.js
    12-http-modules.js
    app.js
    Node JS Notes.pdf
    npm-information.js
    package-lock.json
    package.json
    README.md
13-event-emitter.js U
13-event-emitter.js
1
2 // This will look back from a events module is an class and according to common practice we name it as EventEmitter
3 const customEmitter = require("events");
4 // If we need to any custom operations, we need to extend the EventEmitter Class
5 // or if we just want to emit an event and listen for it then we can create an instance
6 const customEmitter = new EventEmitter();
7 // customEmitter is an instance of Class EventEmitter, however we are interested only on two methods
8 // 1. on - to listen for an event
9 // 2. emit
10 // Here below we are subscribing to the event
11 customEmitter.on("response", () => {
12   | console.log("data received");
13 });
14 // We can have unlimited on methods (listening methods) for one particular event
15 // here we are having two listeners for one event "response"
16 customEmitter.on("response", () => {
17   | console.log("Some other logic is present here");
18 });
19 customEmitter.on("response", (name, id) => {
20   | console.log(`Employee Name: ${name} - ID: ${id}`);
21 });
22
23 // Here we are emitting an event
24 customEmitter.emit("response");
25 // Once an event is emitted, the on method listens to it and executes the registered callback function
26 // In the same way we can have multiple events being emitted
27 customEmitter.emit("response", "Josh", 43);
28
29 // Rules to Follow
30 // 1. An application can have multiple event emitters and event listeners for particular event or multiple events
31 // 2. Order Matters - First we need to have a listener (on) method and then we should have emit method
32 // 3. we pass the arguments when we are emitting an event and the listener have the choice to receive the arguments, this doesn't break any code
33
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS C:\Users\valku\Documents\Node JS Tutorial Youtube> node 13-event-emitter
data received
Some other logic is present here
Employee Name: undefined - ID: undefined
data received
Some other logic is present here
Employee Name: Josh - ID: 43
PS C:\Users\valku\Documents\Node JS Tutorial Youtube>
Ln 18, Col 4  Spaces: 4  UFT-8  CRLF  { JavaScript  ✓ Prettier  ⚡
```

In Node JS, we import the module named “events” to get access to a Class named EventEmitter. If we need to perform any operations on EventEmitter then we need to extend the EventEmitter class or if we just want to listen and emit an event, we can create an instance or object of Class EventEmitter.

There are many methods in EventEmitter class but as of we need only 2 methods namely

1. on method – listens for events
2. emit method – emits an event

Once an event is emitted, the listen method listens the event and executes the registered callback function.

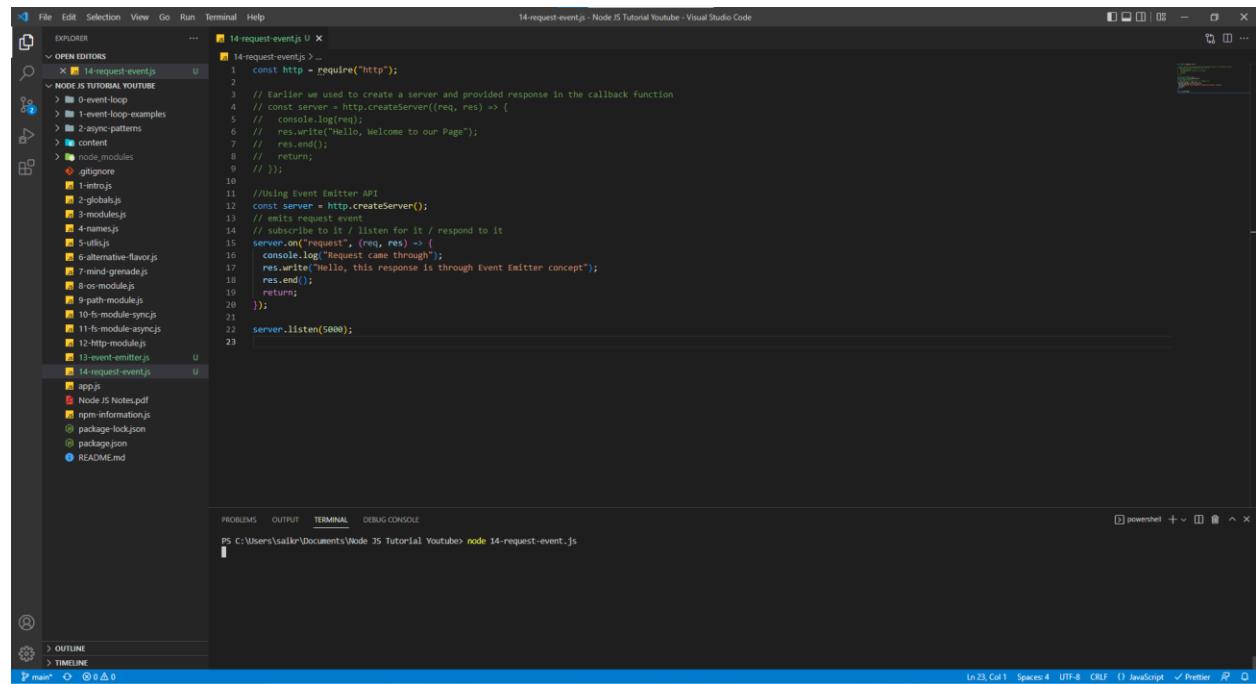
We can have multiple event listeners for one event and in the same way we can emit many events for a particular event.

Rules to be Followed in Events

1. An application can have multiple event emitters and event listeners for event or multiple events.
2. Order Matters - First we need to have a listener (on) method and then we should have emit method.
3. we can pass the arguments when we are emitting an event and the listener have the choice to receive the arguments, this doesn't break any code.

Events Emitter – Http Module Example

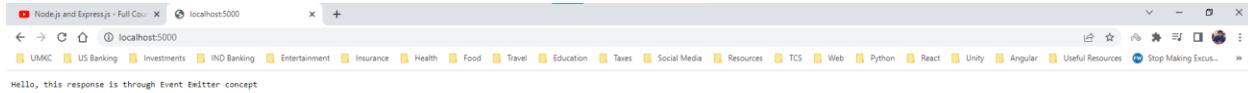
14-request-event.js



```
14-request-event.js
1 const http = require("http");
2 // Earlier we used to create a server and provided response in the callback function
3 // const server = http.createServer((req, res) => {
4 //   console.log(req);
5 //   res.write("Hello, Welcome to our Page");
6 //   res.end();
7 //   return;
8 // });
9 //
10 //Using Event Emitter API
11 const server = http.createServer();
12 // emits request event
13 // subscribe to it / listen for it / respond to it
14 server.on("request", (req, res) => {
15   console.log("Request came through");
16   res.write("Hello, this response is through Event Emitter concept");
17   res.end();
18   return;
19 });
20
21
22 server.listen(5000);
```

Earlier we used to create a server using the HTTP module and then have written a callback function inside the server but in the Event Emitter approach, Node listens to the event/ subscribe to the event/ responds to that event emitted by the server (**request event**) and then executes the registered callback.

At Port 5000, when server sends a request (emits an event), on method in the code listens to emitted event and executes the registered callback function.



Streams in Node JS

Introduction

Streams are used to read and write data sequentially. When we must handle and manipulate streaming data for ex: a continuous source or a big file, streams come handy. There are 4 types of streams.

1. Writeable – used to write data sequentially.
2. Readable – used to read data sequentially.
3. Duplex – used to both read and write data sequentially.
4. Transform – used to modify data when reading or writing.

Just like Events, many built-in modules implement streaming interface. Streams extend EventEmitter Class which simply means that we can use events in streams.

Streams – Read File

When we use readfile or readFileSync we are reading the entire file and storing it in a variable but if we read a big file then we would be eating up a lot of memory for the variable and if the file size increases the variable won't be good enough for storing the file and we would be getting an error for the size.

The solution for the above problem would be readStream option.

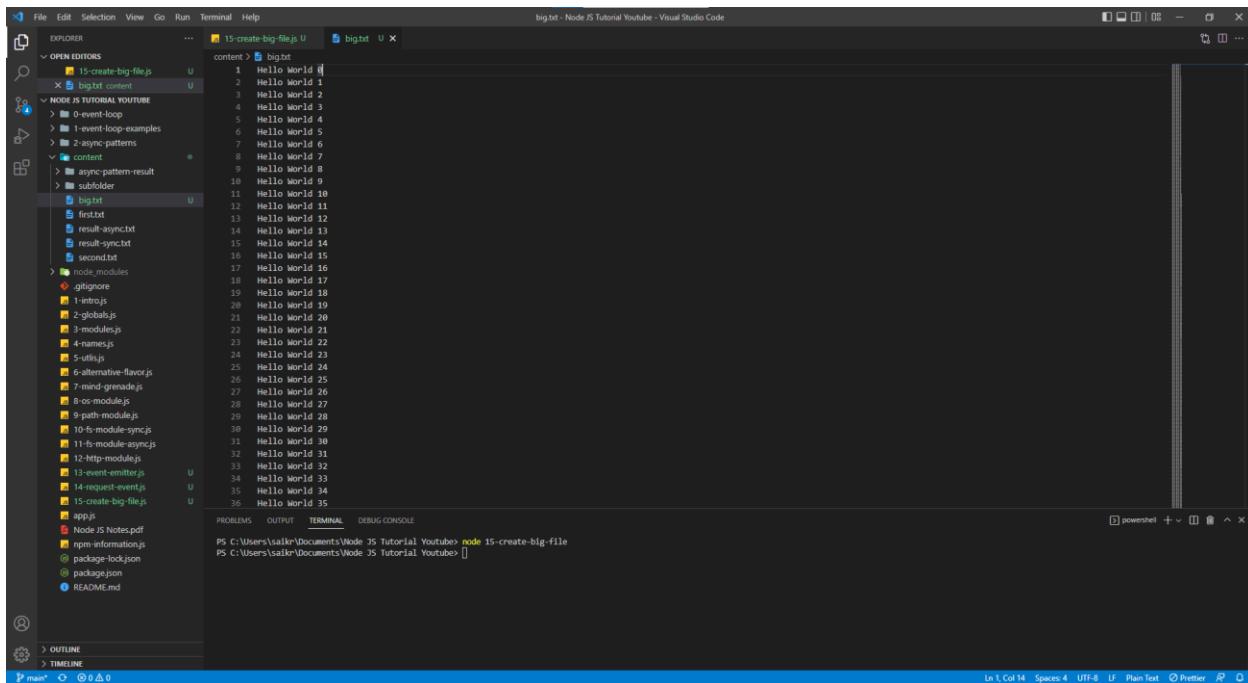
Now, let us set up a big text file which contains a lot of data.

15-create-big-file.js

```
15-create-big-file.js
const { writeFileSync } = require("fs");
for (let i = 0; i < 10000; i++) {
  writeFileSync("./content/big.txt", "Hello World $i\n", { flag: "a" });
}
```

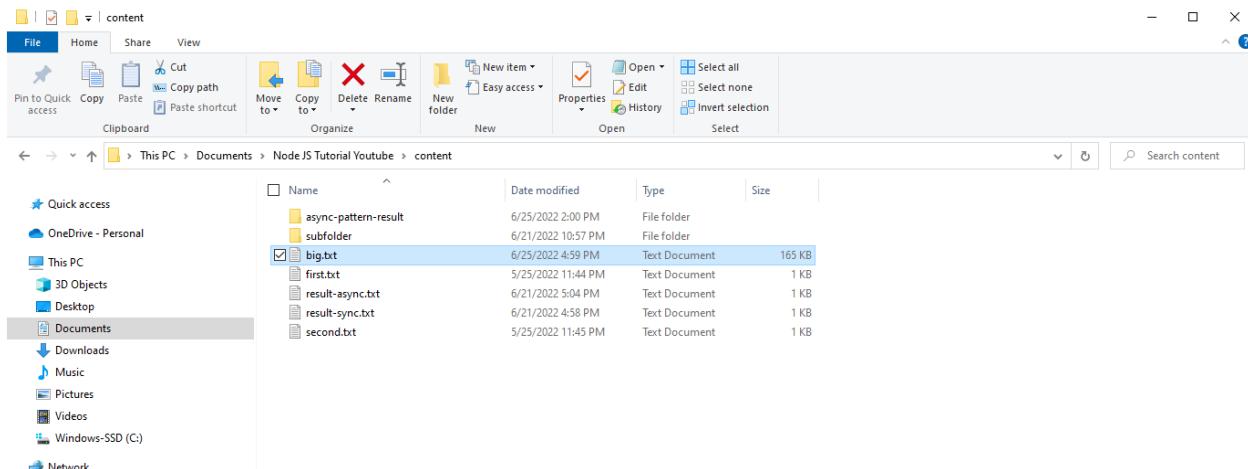
The screenshot shows the Visual Studio Code interface with the file '15-create-big-file.js' open in the editor. The code is a simple for loop that writes 'Hello World' followed by a line number to a file named 'big.txt' 10,000 times. The file is located in a folder named 'big.txt content' under 'NODE TUTORIAL YOUTUBE'. The terminal at the bottom shows the command 'node 15-create-big-file' being run, and the output shows the file being created and populated with data. The status bar at the bottom indicates the file has 6 lines and 4 spaces.

big.txt



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists several files and folders, including '15-create-big-file.js', 'big.txt', 'content', 'first.txt', 'result-async.txt', 'result-sync.txt', 'second.txt', and 'subfolder'. The 'big.txt' file is open in the editor, displaying 36 lines of text: 'Hello World 1' through 'Hello World 36'. The bottom status bar indicates the file is 165 KB in size.

The size of big.txt file is 165 KB



When we use the `createReadStream`, we are reading the data in chunks and by default each chunk of data is 64KB. We can increase size of the chunk. Instead of using the variable we are using the stream to get data in chunks. The event emitted by stream is “data”.

`createReadStream` invokes the `readStream` and returns `readStream`. `readStream` extends from `stream.Readable`. `stream.Readable` has the event named “data”. With options of encoding, we can use `console.log` the data in buffer format.

The screenshot shows a Visual Studio Code interface. The Explorer sidebar on the left lists a project structure with files like `16-streams.js`, `app.js`, and `README.md`. The terminal at the bottom shows the command `node 16-streams.js` being run, followed by several lines of binary data representing file contents. The status bar at the bottom right indicates the code is in JavaScript mode.

```

File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
16-streams.js
NODE JS TUTORIAL YOUTUBE
0-event-loop
1-event-loop-examples
2-async-patterns
content
node_modules
1-.gitignore
2-global.js
3-modules.js
4-name.js
5-util.js
6-alternative-flavor.js
7-mind-grenade.js
8-os-module.js
9-path-module.js
10-fs-module-sync.js
11-fs-module-async.js
12-http-module.js
13-event-emitter.js
14-request-events.js
15-create-big-files.js
16-streams.js
app.js
Node JS Notes.pdf
npm-information.js
package-lock.json
package.json
README.md
16-streams.js
16-streams.js > ...
1 const { createReadStream } = require('fs');
2
3 // the size of the file is 105 KB
4 const stream = createReadStream("./content/big.txt");
5
6 stream.on("data", (result) => {
7   console.log(result);
8 });
9

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS C:\Users\saikiran\Documents\Node JS Tutorial Youtube> node 16-streams.js
dBuffer 48 65 6c 6c 6f 20 57 6f 72 6c 64 20 38 0a 48 65 6c 6f 20 57 6f 72 6c 64 20 32 0a 48 65 6c 6f 20 57 6f ... 65486 more bytes>
dBuffer 57 6f 72 6c 64 20 33 39 32 0a 48 65 6c 6f 20 57 6f 72 6c 64 20 33 39 32 32 0a 48 65 6c 6f 20 57 6f ... 65486 more bytes>
dBuffer 6f 72 6c 64 20 37 37 35 0a 48 65 6c 6f 20 57 6f 72 6c 64 20 37 37 36 0a 48 65 6c 6f 20 57 6f 72 6c 64 20 37 37 37 0a 48 65 6c 6f 20 ... 37768 more bytes>
PS C:\Users\saikiran\Documents\Node JS Tutorial Youtube>

```

About “data” event in Node JS Docs

The screenshot shows a browser window displaying the Node.js documentation for the `Stream` API. The page is titled "Class: stream.Readable". It includes sections for the `'close'` event, the `'data'` event, and the `'end'` event. Each section provides a brief description and examples of how to use the event listeners. The left sidebar contains a navigation menu with various Node.js modules and tools.

Class: stream.Readable

Added in: v0.9.4

Event: 'close'

The `'close'` event is emitted when the stream and any of its underlying resources (a file descriptor, for example) have been closed. The event indicates that no more events will be emitted, and no further computation will occur.

A `Readable` stream will always emit the `'close'` event if it is created with the `emitClose` option.

Event: 'data'

Added in: v0.9.4

• `chunk: <Buffer> | <string> | <any>` The chunk of data. For streams that are not operating in object mode, the chunk will be either a string or `Buffer`. For streams that are in object mode, the chunk can be any JavaScript value other than `null`.

The `'data'` event is emitted whenever the stream is relinquishing ownership of a chunk of data to a consumer. This may occur whenever the stream is switched in flowing mode by calling `readable.pipe()`, `readable.resume()`, or by attaching a listener callback to the `'data'` event. The `'data'` event will also be emitted whenever the `readable.read()` method is called and a chunk of data is available to be returned.

Attaching a `'data'` event listener to a stream that has not been explicitly paused will switch the stream into flowing mode. Data will then be passed as soon as it is available.

The listener callback will be passed the chunk of data as a string if a default encoding has been specified for the stream using the `readable.setEncoding()` method; otherwise the data will be passed as a `Buffer`.

```

const readable = getReadableStreamSomehow();
readable.on('data', (chunk) => {
  console.log(`Received ${chunk.length} bytes of data.`);
});

```

Event: 'end'

Added in: v0.9.4

The `'end'` event is emitted when there is no more data to be consumed from the stream.

The `'end'` event will not be emitted unless the data is completely consumed. This can be accomplished by switching the stream into flowing mode, or by calling `stream.read()` repeatedly until all data has been consumed.

```

const readable = getReadableStreamSomehow();
readable.on('data', (chunk) => {

```

When we control the size of the chunk.

The screenshot shows the Visual Studio Code interface with the following details:

- File Structure:** The Explorer sidebar shows a folder named "Node JS TUTORIAL YOUTUBE" containing various files like 0-event-loop, 1-async-patterns, 1-streams.js, etc.
- Code Editor:** The main editor window contains the following JavaScript code:

```
16-streams.js
1 const { createReadStream } = require("fs");
2
3 // the size of the file is 105 KB
4 const stream = createReadStream("./content/big.txt", { highWaterMark: 90000 });
5
6 // default 64kb
7 // last buffer - remainder
8 // highWaterMark - control size
9 // const stream = createReadStream("./content/big.txt", { highWaterMark: 90000 });
10 // const stream = createReadStream("./content/big.txt", { encoding: "utf8" });
11
12 stream.on("data", (result) => {
13   console.log(result);
14 });
15
```

- Terminal:** The terminal at the bottom shows the output of running the script:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node 16-streams.js
<buffer 48 65 6c 6c 6f 20 57 6f 72 6c 64 20 30 0a 48 65 6c 6f 20 57 6f 72 6c 64 20 32 0a 48 65 6c 6f 20 57 6f ...
dBuffer 6f 72 6c 64 20 35 33 39 0a 48 65 6c 6f 20 57 6f 72 6c 64 20 35 33 36 30 0a 48 65 6c 6f 20 57 6f 72 6c 64 20 35 33 36 31 0a 48 65 6c 6f 20 ... 78840 more bytes>
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>
```

When we encode the data in utf8 format.

The screenshot shows the Visual Studio Code interface with the following details:

- File Structure:** The Explorer sidebar shows a folder named "Node JS TUTORIAL YOUTUBE" containing various files like 0-event-loop, 1-async-patterns, 1-streams.js, etc.
- Code Editor:** The main editor window contains the following JavaScript code:

```
16-streams.js
1 const { createReadStream } = require("fs");
2
3 // the size of the file is 105 KB
4 const stream = createReadStream("./content/big.txt", {
5   highWaterMark: 90000,
6   encoding: "utf8",
7 });
8
9 // default 64kb
10 // last buffer - remainder
11 // highWaterMark - control size
12 // const stream = createReadStream("./content/big.txt", { highWaterMark: 90000 });
13 // const stream = createReadStream("./content/big.txt", { encoding: "utf8" });
14
15 stream.on("data", (result) => {
16   console.log(result);
17 });
18
```

- Terminal:** The terminal at the bottom shows the output of running the script:

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node 16-streams.js
Hello World 9978
Hello World 9979
Hello World 9988
Hello World 9981
Hello World 9982
Hello World 9983
Hello World 9984
Hello World 9985
Hello World 9986
Hello World 9987
Hello World 9988
Hello World 9989
Hello World 9990
Hello World 9991
Hello World 9992
Hello World 9993
Hello World 9994
Hello World 9995
Hello World 9996
Hello World 9997
Hello World 9998
Hello World 9999
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>
```

event emitter emits error event when there is an error while reading the data stream.

```
16-streams.js U X
16-streams.js > stream
1 const { createReadStream } = require("fs");
2
3 // the size of the file is 1.8 MB
4 const stream = createReadStream("./content/big.txt", {
5   highWaterMark: 90000,
6   encoding: "utf8",
7 });
8
9 // default 64kb
10 // last buffer - remainder
11 // highWaterMark controls size
12 // const stream = createReadStream("./content/big.txt", { highWaterMark: 90000 });
13 // const stream = createReadStream("./content/big.txt", { encoding: "utf8" });
14
15 stream.on("data", (result) => {
16   console.log(result);
17 });
18
19 stream.on("error", (err) => {
20   console.log(err);
21 });
22
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node 16-streams.js
[Error: ENOENT: no such file or directory, open 'C:\Users\saikr\Documents\content\big.txt'] {
  errno: -4049,
  code: 'ENOENT',
  syscall: 'open',
  path: 'C:\Users\saikr\Documents\content\big.txt'
}
PS C:\Users\saikr\Documents\Node JS Tutorial Youtube>
```

OUTLINE TIMELINE

Streams – Http Example

Let's see a practical example when streams come in handy. Assume a situation where we are trying to send the data from big.txt file we created. We have increased the size of bg.txt file to 1.8 MB and it is not ideal to transfer data of 1.8MB over the internet.

```
17-http-streams.js U X
17-http-streams.js > _readfile
1 const http = require("http");
2 const { readfile } = require("fs");
3
4 const server = http.createServer((req, res) => {
5   const data = readfile("./content/big.txt", (err, result) => {
6     res.end(result);
7   });
8 });
9
10 server.listen(5000);
11
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

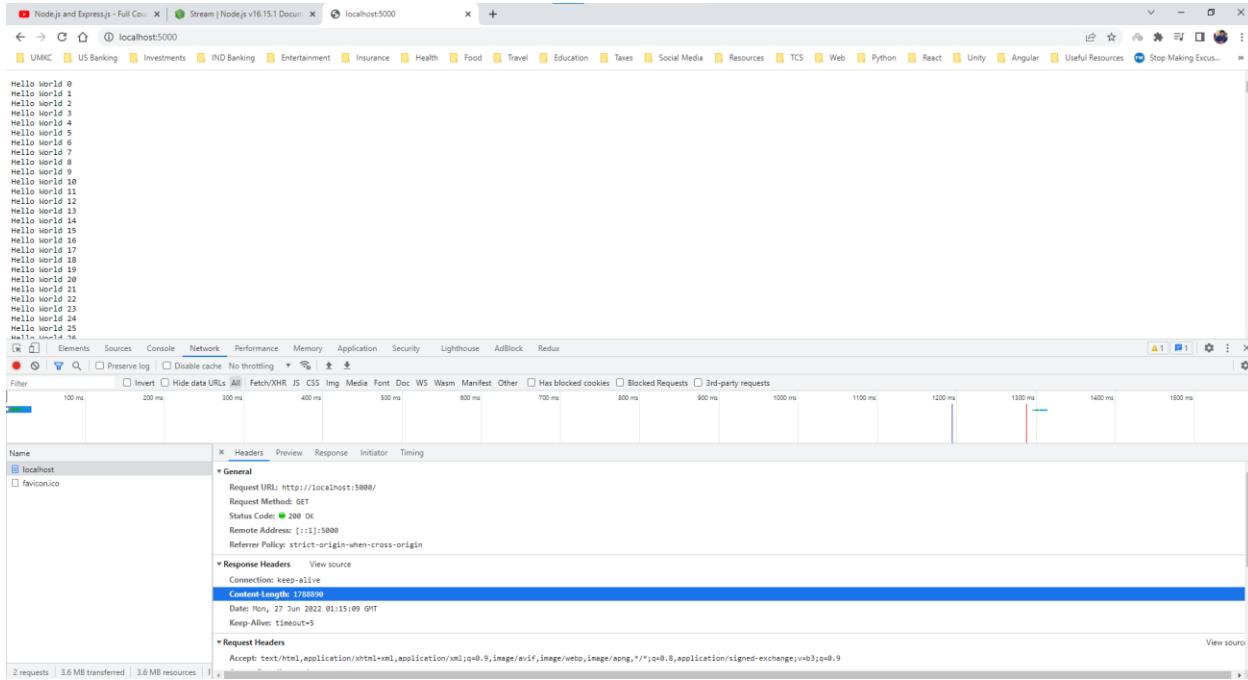
Try the cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\saikr\Documents\Node JS Tutorial Youtube> node 17-http-stream

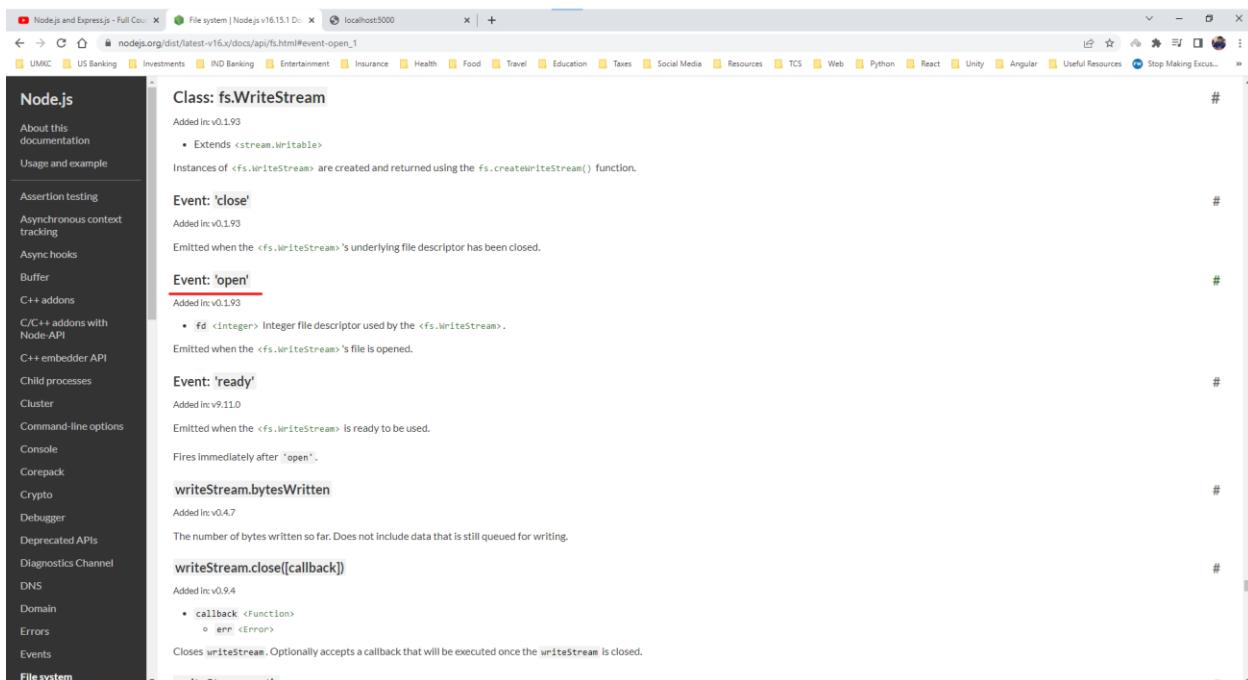
```

OUTLINE TIMELINE

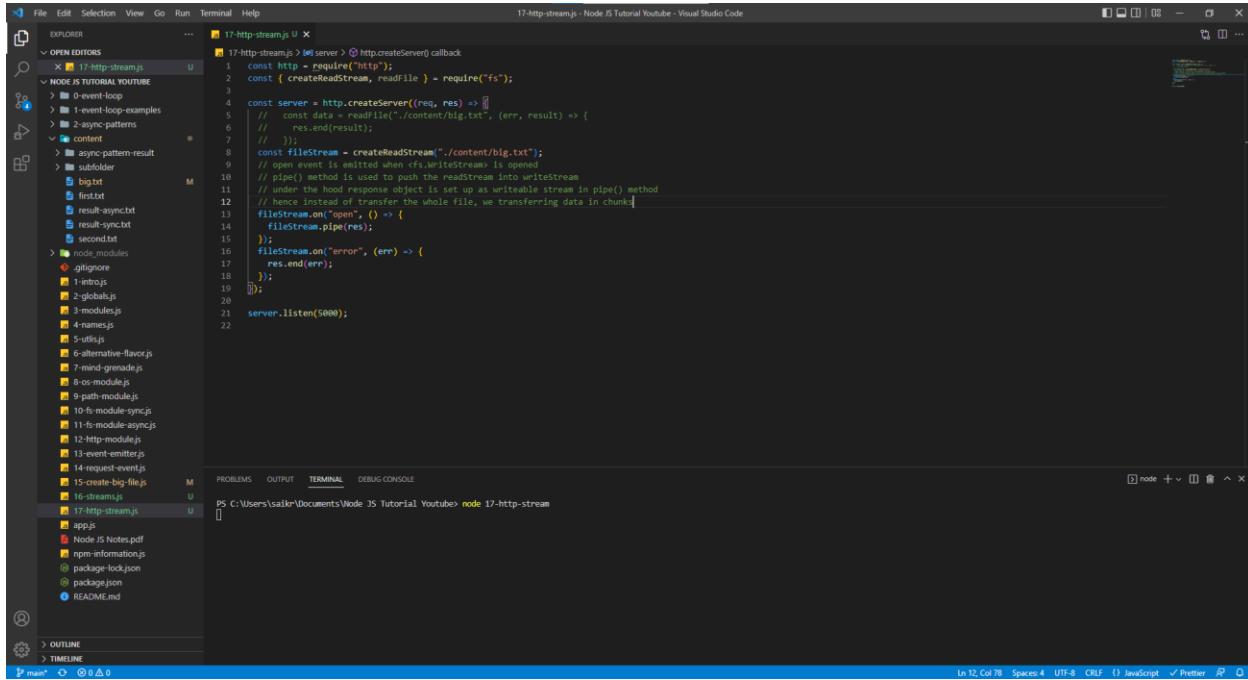
In the below screenshot we can see the size of file being approximately equal to 1.8 MB and hence we want to refactor the code and send the data in chunks.



To send the data in chunks, we use `createReadStream`. The `pipe()` method is used to push the `readStream` into `writeStream`. So, we can imagine we read data in chunks then we can also write data in chunks. What happens under the hood `response` object is set up as `writable stream` in `pipe()` method. ***Some information about "open event in writeStream.***



Hence in this case we are reading the data in chunks and writing the data in chunks using the pipe() method without transferring the data of large file at once.

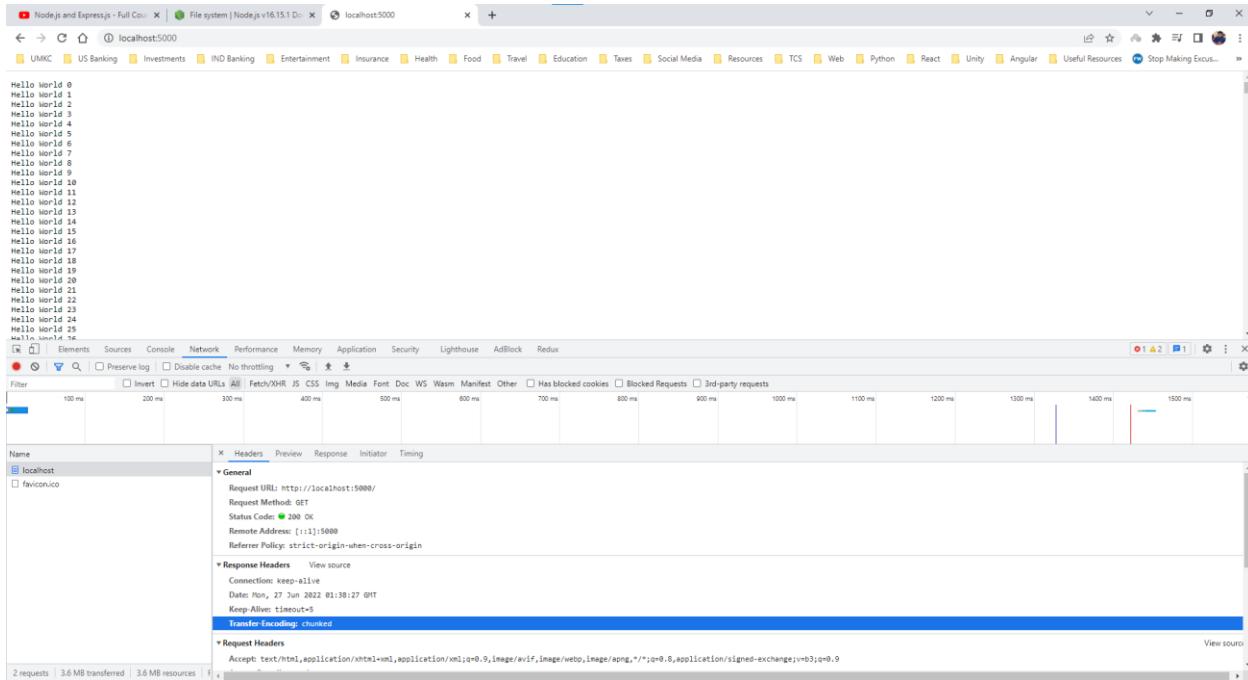


```

17-http-stream.js U
17-HTTP-STREAM.js > HTTP server > [HTTP.createServer] callback
1    const http = require('http');
2    const { createReadStream, readfile } = require('fs');
3
4    const server = http.createServer((req, res) => {
5        // const data = readfile("./content/big.txt", (err, result) => {
6        //     res.end(result);
7        // });
8
9        const fileStream = createReadStream("./content/big.txt");
10       // pipe() method is used to push the readStream into writeStream
11       // under the hood response object is set up as writable stream in pipe() method
12       // hence instead of transfer the whole file, we transferring data in chunks
13       fileStream.on("open", () => {
14           fileStream.pipe(res);
15       });
16       fileStream.on("error", (err) => {
17           res.end(err);
18       });
19   });
20
21   server.listen(5000);
22

```

We are receiving the data in chunks.



End of Node Tutorial Module

Until Here we are done with Node Fundamentals