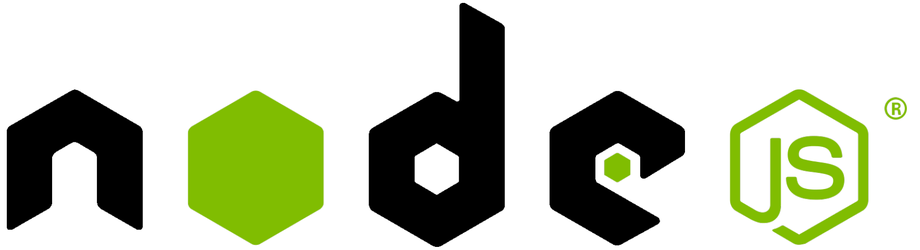
JAVASCRIPT

PROMISE, ASYNC-AWAIT, REQUEST

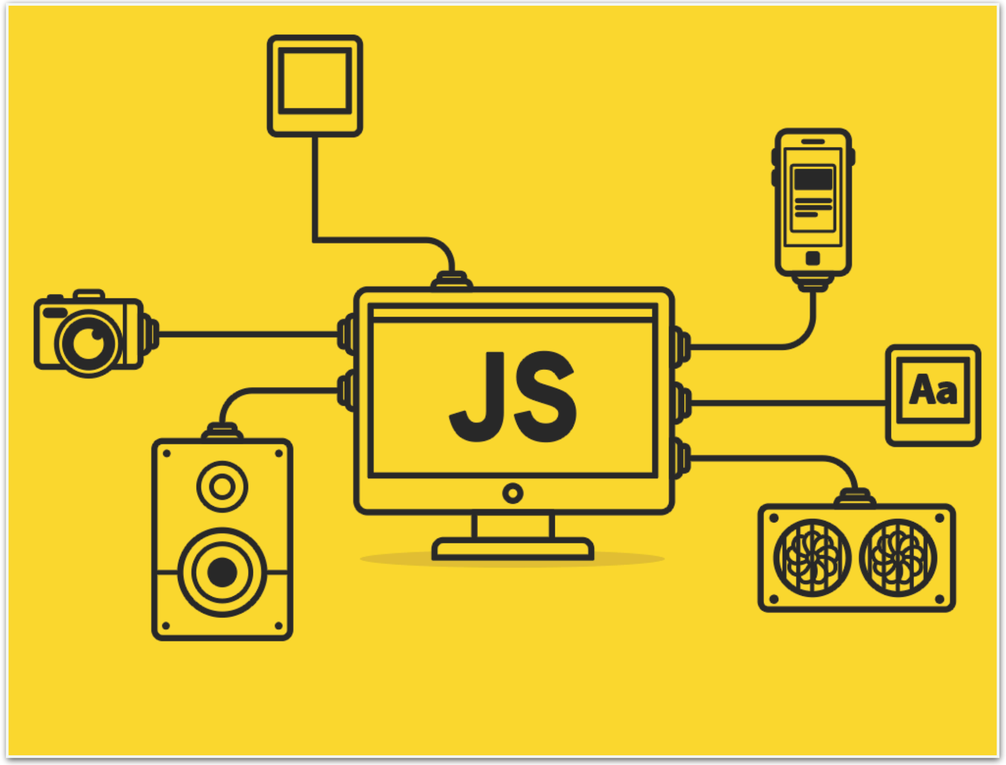
Anirach Mingkhwan

Anirach.m@fitm.kmutnb.ac.th



OUTLINE

- Asynchronous
- Promises
- Async-Await
- Request
- NPM



ASYNCHRONOUS

CERTAIN OPERATION COULD TAKE TIME

File I/O
REST calls
Database operations
Complex computations

Most applications are single threaded
One worker process
One activity at a time
User interface or application could appear frozen
No other tasks can execute

Credit:https://sec.ch9.ms

ORIGINALLY USING CALLBACK

```
function callback() {
  console.log('Timeout completed');
}

setTimeout(callback, 3000); // wait 3 seconds
```

Credit:https://sec.ch9.ms

CASCADING PROBLEM

```
longRunningOperation(() => {
     anotherLongRunningOperation(() => {
          yetAnother(() => {
               oneMore(() => {
                    lastOne(() => {
                         console.log('Where are we?');
                    });
               });
          })
     })
});
```

Credit:https://sec.ch9.ms

PROMISE

ENTER PROMISE

Common development pattern

Cleaner version of callbacks

Recent versions of JavaScript have built-in Promise object

Long running operations typically return a Promise

Credit:https://sec.ch9.ms

PROMISE SYNTAX

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Credit:https://www.w3schools.com/js/js_promise.asp

STATES OF A JAVASCRIPT PROMISE

A JavaScript `Promise` object can be in one of three states: `pending`, `resolved`, or `rejected`.

While the value is not yet available, the `Promise` stays in the `pending` state. Afterwards, it transitions to one of the two states: `resolved` or `rejected`.

A resolved promise stands for a successful completion. Due to errors, the promise may go in the `rejected` state.

In the given code block, if the `Promise` is on `resolved` state, the first parameter holding a callback function of the `then()` method will print the resolved value. Otherwise, an alert will be shown.

```
const promise = new Promise((resolve, reject) => {
  const res = true;
  // An asynchronous operation.
  if (res) {
    resolve("Resolved!");
  } else {
    reject(Error("Fatal Error"));
  }
});

promise.then(
  (res) => console.log(res),
  (err) => console.log(err)
);
```

Credit:https://sec.ch9.ms

PROMISE STATE

Lecture-05-Asychronous > JS statepromise.js > ...

```
1   const promise = new Promise((resolve, reject) => {
2     const res = true;
3     // An asynchronous operation.
4     if (res) {
5       resolve("Resolved!");
6     } else {
7       reject(Error("Fatal Error"));
8     }
9   });
10
11  promise.then(
12    (res) => console.log(res),
13    (err) => console.log(err)
14  );
15
```

```
> node statepromise.js
Resolved!

~/C/JavaScript/
>
```

Credit:https://sec.ch9.ms

STATES OF A JAVASCRIPT PROMISE

A JavaScript `Promise` object can be in one of three states: `pending`, `resolved`, or `rejected`.

While the value is not yet available, the `Promise` stays in the `pending` state. Afterwards, it transitions to one of the two states: `resolved` or `rejected`.

A resolved promise stands for a successful completion. Due to errors, the promise may go in the `rejected` state.

In the given code block, if the `Promise` is on `resolved` state, the first parameter holding a callback function of the `then()` method will print the resolved value. Otherwise, an alert will be shown.

```
const promise = new Promise((resolve, reject) => {
  const res = true;
  // An asynchronous operation.
  if (res) {
    resolve('Resolved!');
  }
  else {
    reject(Error('Error'));
  }
});

promise.then((res) => console.log(res), (err) =>
alert(err));
```

Credit:https://sec.ch9.ms

PROMISE DEMO

Lecture-05-Asychronous > JS promise.js > ...

```
1   function promiseTimeout(ms) {
2     return new Promise((resolve, reject) => {
3       setTimeout(resolve, ms);
4     });
5   }
6
7   promiseTimeout(2000)
8     .then(() => {
9       console.log("Done!!");
10      return promiseTimeout(1000);
11    })
12    .then(() => {
13      console.log("Also done!!");
14      return Promise.resolve(42);
15    })
16    .then((result) => {
17      console.log(result);
18    })
19    .catch(() => {
20      console.log("Error!");
21    });
22
```

```
> node promise.js
Done!!
Also done!!
42

 ~/C/JavaScript
>
```

Credit:https://sec.ch9.ms

ASYNC-AWAIT

ASYNC/AWAIT

Make asynchronous code look synchronous

While promises are cleaner, they're not perfect

Can add bloat to code

async/await

Standard in many languages

Syntax closer to synchronous code

Credit:https://sec.ch9.ms

ASYNC/AWAIT IN ACTION

Lecture-05-Asychronous > JS asyncawait-1.js > ...

```
1   function promiseTimeout(ms) {
2     return new Promise((resolve, reject) => {
3       setTimeout(resolve, ms);
4     });
5   }
6
7   async function run() {
8     // logic
9     console.log("Start!!");
10    // try take of await and compare
11    await promiseTimeout(2000);
12    console.log("Stop!!");
13  }
14
15  run();
16
```

```
> node asyncawait-1.js
Start!!
Stop!!

 ~/C/JavaScript
>
```

Credit:https://sec.ch9.ms

ASYNC/AWAIT IN ACTION-1

Lecture-05-Asychronous > JS asyncawait-2.js > ...

```
1   function promiseTimeout(ms) {
2     return new Promise((resolve, reject) => {
3       setTimeout(resolve, ms);
4     });
5   }
6
7   async function longRunningOperation() {
8     // logic
9     return 42;
10  }
11
12  async function run() {
13    // logic
14    console.log("Start!!");
15    await promiseTimeout(2000);
16    //try to take await out and see
17    const response = await longRunningOperation();
18    console.log(response);
19
20    console.log("Stop!!");
21  }
22
23  run();
24
```

```
> node asyncawait-2.js
Start!!
42
Stop!!

 ~/C/JavaScript/
>
```

Credit:https://sec.ch9.ms

ASYNC/AWAIT IN ACTION-2

Lecture-05-Asychronous > JS moreasyncawait.js > ...

```
1   function who() {
2     return new Promise((resolve) => {
3       setTimeout(() => {
4         resolve("🤡");
5       }, 200);
6     });
7   }
8
9   function what() {
10    return new Promise((resolve) => {
11      setTimeout(() => {
12        resolve("lurks");
13      }, 300);
14    });
15  }
16
```

```
17  function where() {
18    return new Promise((resolve) => {
19      setTimeout(() => {
20        resolve("in the shadows");
21      }, 500);
22    });
23  }
24
25  async function msg() {
26    const a = await who();
27    const b = await what();
28    const c = await where();
29
30    console.log(`${a} ${b} ${c}`);
31  }
32  console.log('We are looking for:')
33  msg(); // 🤡 lurks in the shadows <-- after 1 second
34  console.log('Hello')
35
```

```
> node moreasyncawait.js
We are looking for:
Hello
🤡 lurks in the shadows

~/C/JavaScript/Le
>
```

Credit:https://sec.ch9.ms

ASYNC/AWAIT IN ACTION-3

Lecture-05-Asychronous > JS asynccatch.js > ...

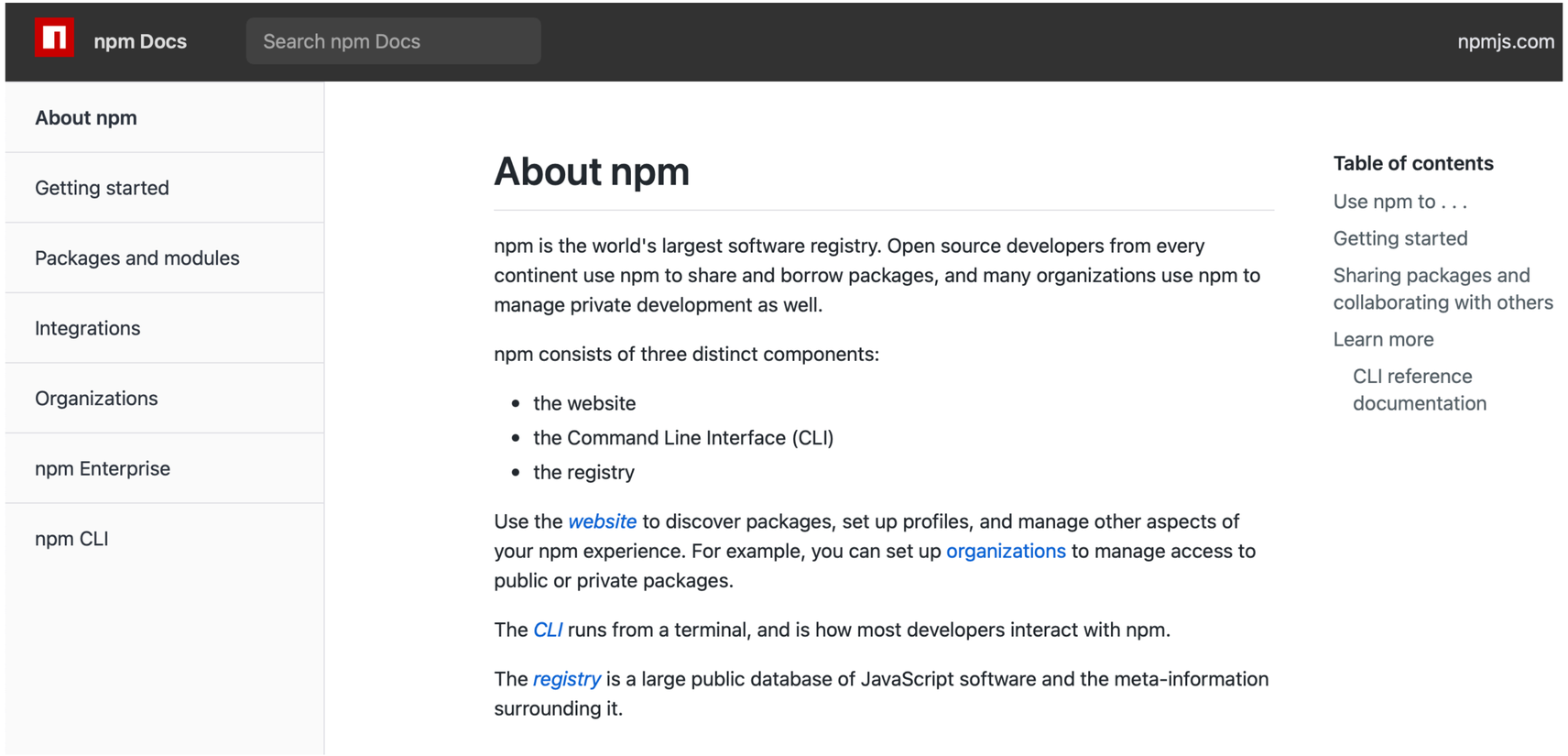
```
1   function yayOrNay() {
2     return new Promise((resolve, reject) => {
3       const val = Math.round(Math.random() * 1);
4       // 0 or 1, at random
5       val ? resolve("Lucky!!") : reject("Nope 😠");
6     });
7   }
8
9   async function msg() {
10    try {
11      const msg = await yayOrNay();
12      console.log(msg);
13    } catch (err) {
14      console.log(err);
15    }
16  }
17
18  msg(); // Lucky!!
19  msg(); // Lucky!!
20  msg(); // Lucky!!
21  msg(); // Nope 😠
22  msg(); // Lucky!!
23  msg(); // Nope 😠
24  msg(); // Nope 😠
```

```
❯ node asynccatch.js
Lucky!!
Lucky!!
Lucky!!
Lucky!!
Lucky!!
Lucky!!
Nope 😠
Lucky!!
Lucky!!
Lucky!!
❯ node asynccatch.js
Lucky!!
Nope 😠
Nope 😠
Nope 😠
Nope 😠
Nope 😠
Lucky!!
Lucky!!
Nope 😠
Lucky!!

~/C/JavaScript
❯
```

Credit:https://sec.ch9.ms

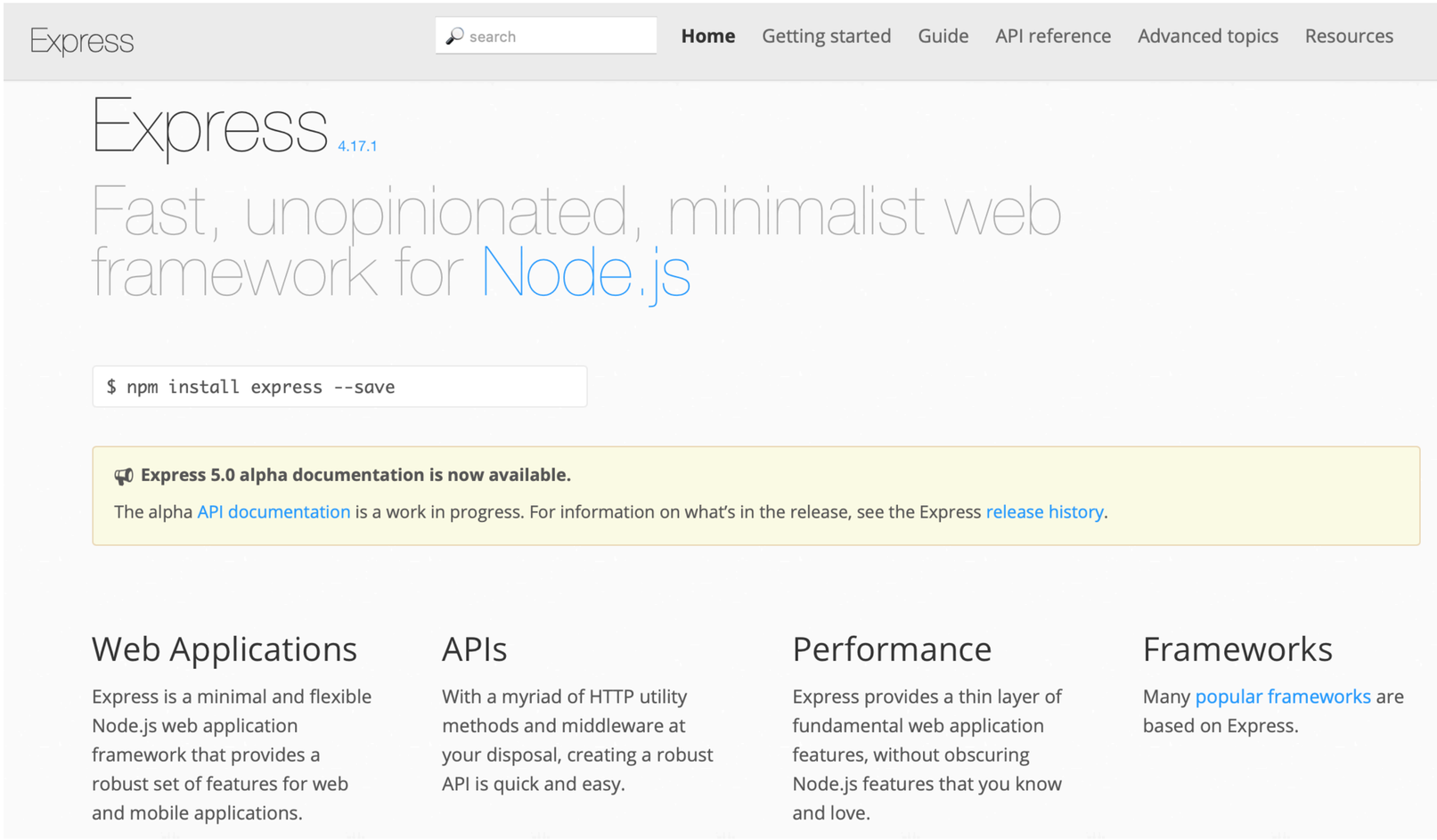
NPM PACKAGE

NPM



https://docs.npmjs.com/about-npm

EXPRESS



http://expressjs.com

DOTENV

dotenv

8.2.0 • Public • Published a year ago

Readme | Explore BETA | 0 Dependencies | 19,175 Dependents | 38 Versions

dotenv

Dotenv is a zero-dependency module that loads environment variables from a `.env` file into `process.env`. Storing configuration in the environment separate from code is based on The Twelve-Factor App methodology.

build passing | build failing | npm v8.2.0 | code style standard | coverage 100%

license BSD-2-Clause | Conventional Commits 1.0.0

Install

```
# with npm
npm install dotenv

# or with Yarn
yarn add dotenv
```

Usage

Install

```
npm i dotenv
```

Weekly Downloads

13,364,266

Version	License
8.2.0	BSD-2-Clause

Unpacked Size	Total Files
23.1 kB	12

Issues	Pull Requests
29	10

Homepage

github.com/motdotla/dotenv#readme

Repository

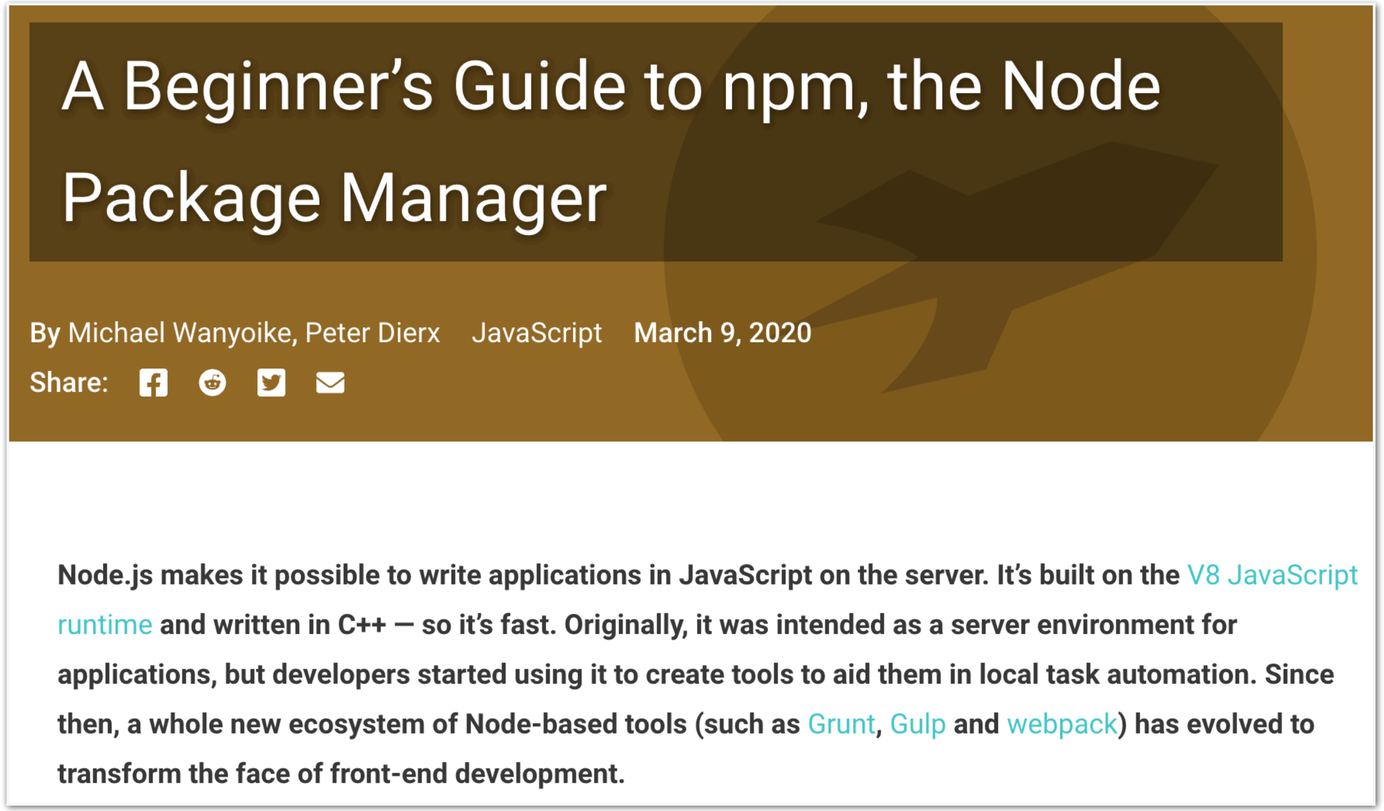
github.com/motdotla/dotenv

https://www.npmjs.com/package/dotenv

```
npm init -y

npm install --save-dev prettier

npm run format

npm install express

node index.js

npm start

npm install dotenv

npm start

```

https://learn.microsoft.com/en-us/shows/beginners-series-to-javascript/demo-package-management-50-of-51

NPM GUIDE



Let's assume you've cloned your project source code to a another machine and we want to install the dependencies. Let's delete the node_modules folder first, then execute npm install:

https://www.sitepoint.com/npm-guide/

THANK YOU

- Asynchronous
- Promises
- Async-Await
- Package