
JAVASCRIPT PROMISE, ASYNC-AWAIT, REQUEST

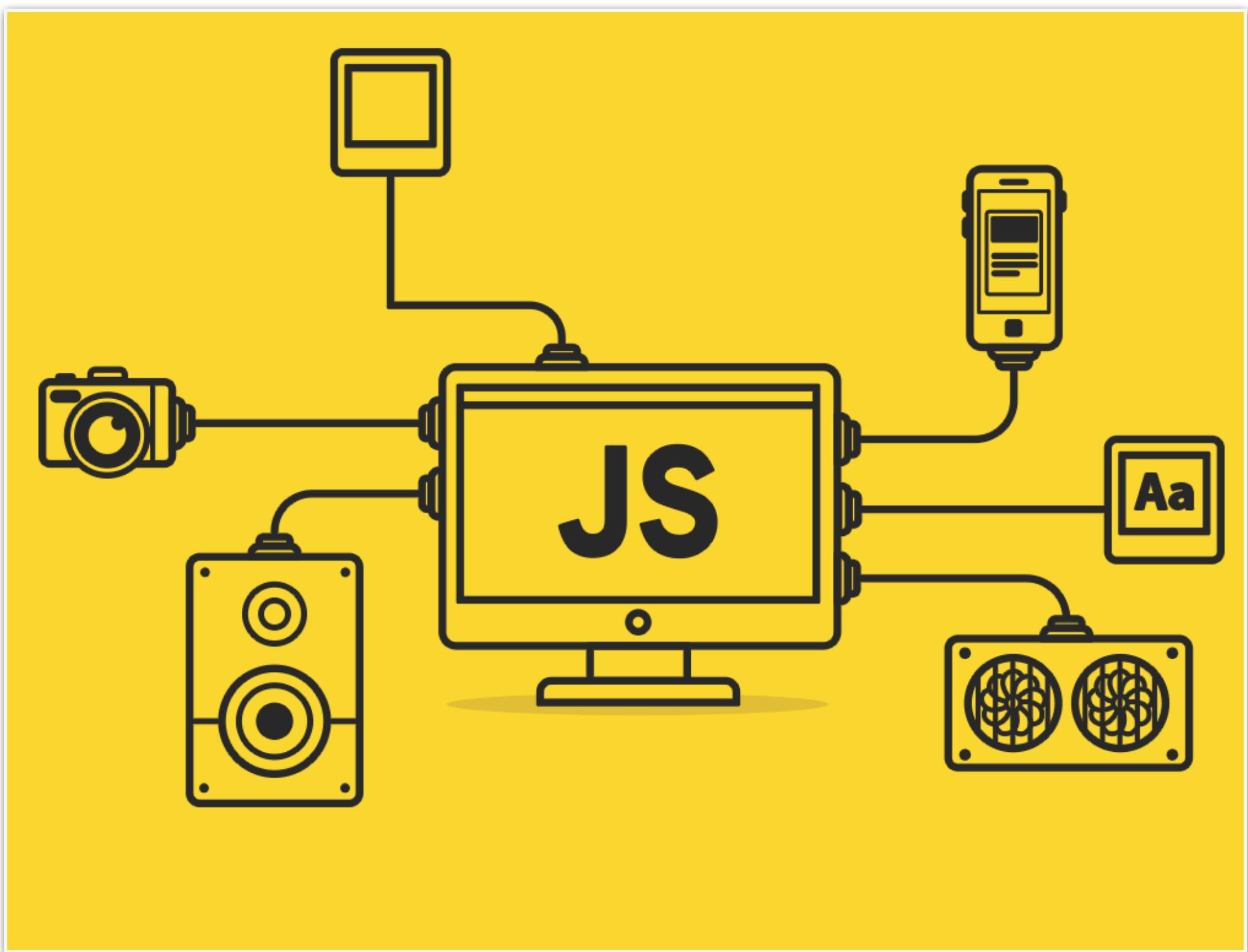
Anirach Mingkhwан

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

ORIGINALLY USING CALLBACK

```
function callback() {  
    console.log('Timeout completed');  
}  
  
setTimeout(callback, 3000); // wait 3 seconds
```

CASCADING PROBLEM

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

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

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 */ }  
);
```

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)
);
```

PROMISE STATE

Lecture-05-Asynchronous > 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!
```

```
MacBook-Pro:~/C/JavaScript/
```

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));
```

PROMISE DEMO

Lecture-05-Asynchronous > `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
>
```

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

ASYNC/AWAIT IN ACTION

Lecture-05-Asynchronous > 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!!
```

```
MacBook-Pro:~/C/JavaScript
```

ASYNC/AWAIT IN ACTION-1

Lecture-05-Asynchronous > 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!!  
  
> [Terminal Icon] ~/C/JavaScript/
```

ASYNC/AWAIT IN ACTION-2

Lecture-05-Asynchronous > 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

ASYNC/AWAIT IN ACTION-3

Lecture-05-Asynchronous > `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!!
Lucky!!
Nope 😞
Lucky!!
Lucky!!
Lucky!!
Lucky!!
> node asynccatch.js
Lucky!!
Nope 😞
Nope 😞
Nope 😞
Nope 😞
Nope 😞
Lucky!!
Lucky!!
Nope 😞
Lucky!!

```



Credit:<https://sec.ch9.ms>

NPM PACKAGE

NPM

The screenshot shows the 'About npm' page from the npm Docs website. The top navigation bar includes the 'npm Docs' logo, a search bar, and the URL 'npmjs.com'. A sidebar on the left lists several sections: 'About npm', 'Getting started', 'Packages and modules', 'Integrations', 'Organizations', 'npm Enterprise', and 'npm CLI'. The main content area features a large heading 'About npm' and a paragraph explaining that npm is the world's largest software registry. It then details the three components of npm: the website, the Command Line Interface (CLI), and the registry. Below this, it describes the website and CLI, and concludes with information about the registry.

About npm

npm is the world's largest software registry. Open source developers from every continent use npm to share and borrow packages, and many organizations use npm to manage private development as well.

npm consists of three distinct components:

- the website
- the Command Line Interface (CLI)
- the registry

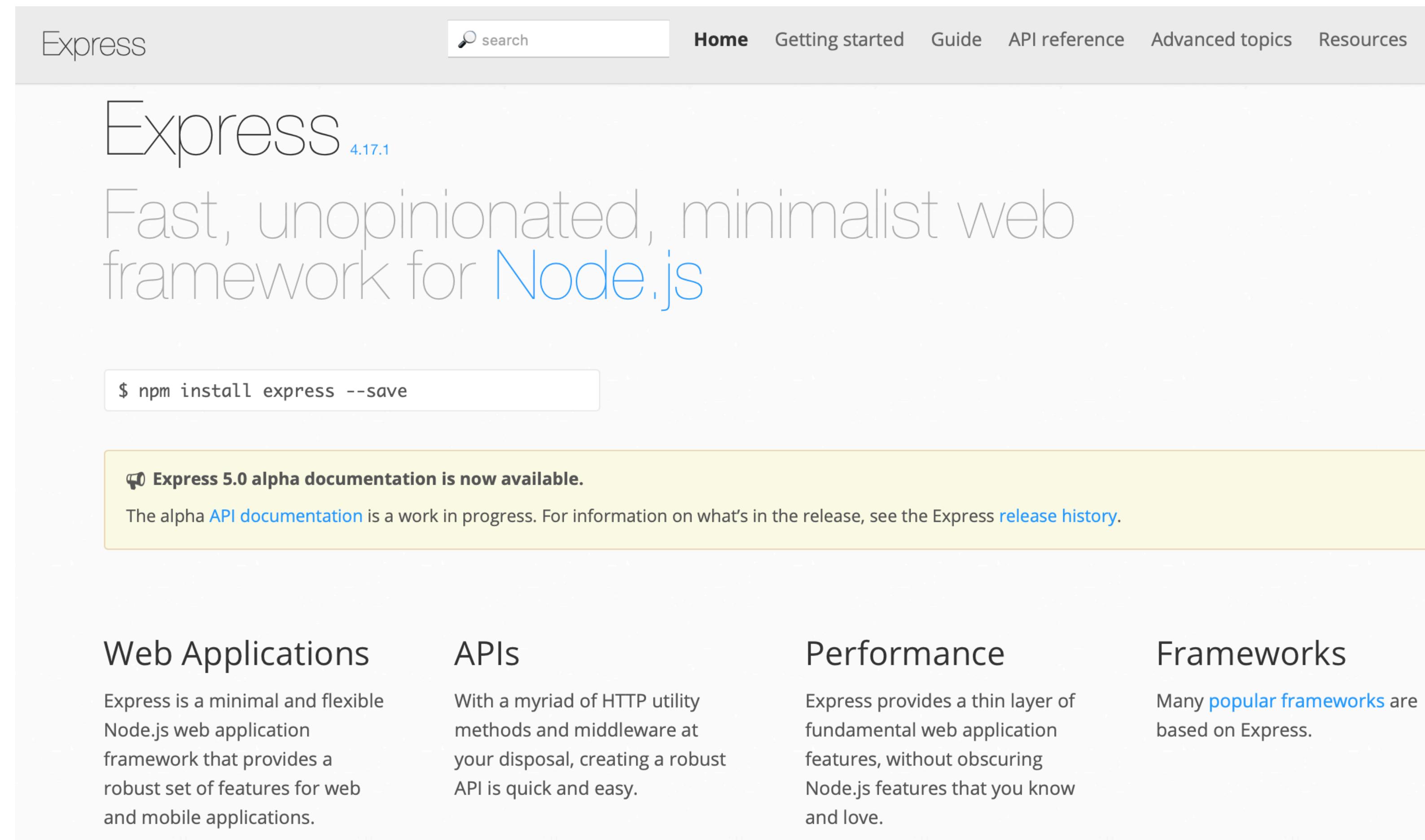
Use the [website](#) to discover packages, set up profiles, and manage other aspects of your npm experience. For example, you can set up [organizations](#) to manage access to public or private packages.

The [CLI](#) runs from a terminal, and is how most developers interact with npm.

The [registry](#) is a large public database of JavaScript software and the meta-information surrounding it.

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

EXPRESS



The screenshot shows the official Express.js website. At the top, there's a navigation bar with links for Home, Getting started, Guide, API reference, Advanced topics, and Resources. A search bar is also present. The main title "Express" is displayed in a large, light blue font, followed by the version "4.17.1". Below the title, a subtitle reads "Fast, unopinionated, minimalist web framework for Node.js". A command-line interface (CLI) command "\$ npm install express --save" is shown in a code block. A yellow callout box contains the text "Express 5.0 alpha documentation is now available." and "The alpha API documentation is a work in progress. For information on what's in the release, see the Express [release history](#)." At the bottom, there are four sections: "Web Applications", "APIs", "Performance", and "Frameworks".

Express

search

Home Getting started Guide API reference Advanced topics Resources

Express 4.17.1

Fast, unopinionated, minimalist web framework for Node.js

```
$ npm install express --save
```

Express 5.0 alpha documentation is now available.
The alpha API documentation is a work in progress. For information on what's in the release, see the Express [release history](#).

Web Applications

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

APIs

With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

Frameworks

Many popular frameworks are based on Express.

<http://expressjs.com>

DOTENV

dotenv

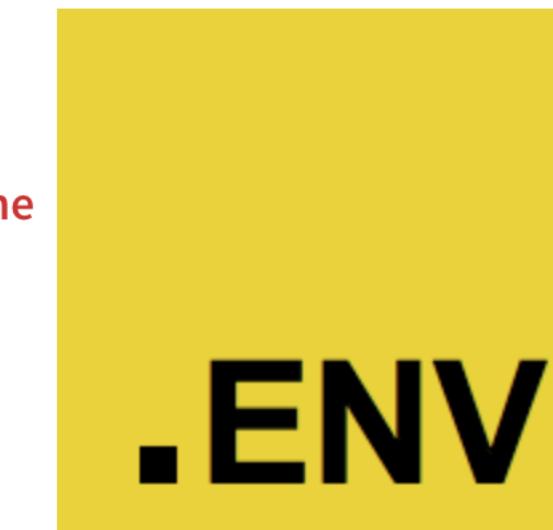
8.2.0 • Public • Published a year ago



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
```

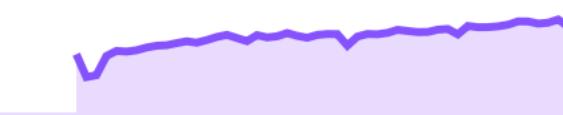
Usage

Install

```
> npm i dotenv
```

Weekly Downloads

13,364,266



Version

8.2.0

License

BSD-2-Clause

Unpacked Size

23.1 kB

Total Files

12

Issues

29

Pull Requests

10

Homepage

github.com/motdotla/dotenv#readme

Repository

github.com/motdotla/dotenv

```
1  npm init -y
2
3  npm install --save-dev prettier
4
5  npm run format
6
7  npm install express
8
9  node index.js
10
11 npm start
12
13 npm install dotenv
14
15 npm start
16
17
```

NPM GUIDE

A Beginner's Guide to npm, the Node Package Manager

By Michael Wanyoike, Peter Dierx JavaScript March 9, 2020

Share:

Node.js makes it possible to write applications in JavaScript on the server. It's built on the [V8 JavaScript runtime](#) and written in C++ – so it's fast. Originally, it was intended as a server environment for applications, but developers started using it to create tools to aid them in local task automation. Since then, a whole new ecosystem of Node-based tools (such as [Grunt](#), [Gulp](#) and [webpack](#)) has evolved to transform the face of front-end development.

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