

Namaste React Notes

Lecture 1- Inception

Hello World Program by using HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Namaste React</title>
</head>

<body>
  <div id="root">
    <h1>Hello World using HTML</h1>
  </div>
</body>

</html>
```

Hello World Program by using Javascript

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Namaste React</title>
</head>

<body>
  <div id="root">
  </div>
  <script>
    const heading = document.createElement("h1");
    heading.innerHTML = "Hello World from JavaScript"
    const root = document.getElementById("root")
    root.appendChild(heading)
  </script>
</body>

</html>
```

Injecting React into Html file using CDN(Content Delivery Network)

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Namaste React</title>
</head>

<body>
  <div id="root">
  </div>
  <script crossorigin
src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
</body>

</html>
```

Hello World Program using React

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Namaste React</title>
</head>

<body>
  <div id="root">
  </div>
  <script crossorigin
src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script>
    const heading = React.createElement("h1", {}, "Hello World from React")
    const root = ReactDOM.createRoot(document.getElementById("root"))
    root.render(heading)
  </script>
</body>

</html>
```

Separating the JavaScript Code, CSS and HTML into separate files

index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="index.css">
  <title>Namaste React</title>
</head>

<body>
  <div id="root">
  </div>
  <script crossorigin
src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script src="App.js"></script>
</body>

</html>
```

App.js

```
const heading = React.createElement(
  "h1",
  { id: "heading", "data-testid": "heading", testid: "heading" },
  "Hello World from React"
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(heading);
```

index.css

```
#root {
  background-color: aqua;
}
```

Output:

Hello World from React

If we console.log(heading) in App.js

```
const heading = React.createElement(
  "h1",
  { id: "heading", "data-testid": "heading", testid: "heading" },
  "Hello World from React"
);
console.log(heading)
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(heading);
```

Output:

```
▼ Object i
  $$typeof: Symbol(react.element)
  key: null
  ▼ props:
    children: "Hello World from React"
    data-testid: "heading"
    id: "heading"
    testid: "heading"
    ► [[Prototype]]: Object
  ref: null
  type: "h1"
  _owner: null
  ► _store: {validated: false}
  _self: null
  _source: null
  ► [[Prototype]]: Object
>
```

Creating Nested Elements in React

Trying to create

```
<div id="parent">
  <div id="child">
    <h1 id="inner-child">Hello World!</h1>
  </div>
</div>
```

```
const parent = React.createElement(
  "div",
  { id: "parent" },
  React.createElement(
    "div",
    { id: "child" },
    React.createElement("h1", { id: "inner-child" }, "Hello World!")
  )
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(parent);
```

```
▼ {$$typeof: Symbol(react.element), type: 'div', key: null, ref: null, props: {...}, ...} ⓘ
  $$typeof: Symbol(react.element)
  key: null
  ▼ props:
    ▼ children:
      $$typeof: Symbol(react.element)
      key: null
      ▼ props:
        ► children: {$$typeof: Symbol(react.element), type: 'h1', key: null, ref: null, props: {...}, ...}
        id: "child"
        ► [[Prototype]]: Object
        ref: null
        type: "div"
        _owner: null
        ► _store: {validated: true}
        _self: null
        _source: null
        ► [[Prototype]]: Object
        id: "parent"
        ► [[Prototype]]: Object
        ref: null
        type: "div"
        _owner: null
        ► _store: {validated: false}
        _self: null
        _source: null
        ► [[Prototype]]: Object
    > |
```

Notice the children in the above example

Creating Siblings in React

```
<div id="parent">
  <div id="child">
    <h1 id="inner-child1">H1 Tag</h1>
    <h2 id="inner-child2">H2 Tag</h2>
  </div>
</div>
```

```
const parent = React.createElement(
  "div",
  { id: "parent" },
  React.createElement(
    "div",
    { id: "child" },
    [React.createElement("h1", { id: "inner-child1", key:"1" }, "H1 Tag"),
     React.createElement("h2", { id: "inner-child2", key:"2" }, "H2 Tag")]
  )
);
console.log(parent);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(parent);
```

Siblings are passed inside an Array

H1 Tag

H2 Tag

```
▼ {$$typeof: Symbol(react.element), type: 'div', key: null, ref: null, props: {...}} ⓘ
  $$typeof: Symbol(react.element)
  key: null
  ▼ props:
    ▼ children:
      $$typeof: Symbol(react.element)
      key: null
      ▼ props:
        ▼ children: Array(2)
          ► 0: {$$typeof: Symbol(react.element), type: 'h1', key: null, ref: null, props: {...}, ...}
          ► 1: {$$typeof: Symbol(react.element), type: 'h2', key: null, ref: null, props: {...}, ...}
          length: 2
          ► [[Prototype]]: Array(0)
          id: "child"
          ► [[Prototype]]: Object
          ref: null
          type: "div"
          _owner: null
          ► _store: {validated: true}
          _self: null
          _source: null
          ► [[Prototype]]: Object
          id: "parent"
          ► [[Prototype]]: Object
          ref: null
          type: "div"
          _owner: null
          ► _store: {validated: false}
          _self: null
          _source: null
          ► [[Prototype]]: Object
    ►
```

It becomes extremely complex to write React Code like this. So, there came the need for JSX (HTML Like syntax inside Javascript)

Lecture-02 Igniting Our App

npm is a package manager for the JavaScript programming language maintained by npm, Inc. npm is the default package manager for the JavaScript runtime environment Node.js and is included as a recommended feature in the Node.js installer.

Project Scaffolding Steps:

1. npm init (Creates package.json)

```
PS C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th Feb> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (namaste-react-02th-feb)
version: (1.0.0)
description:
entry point: (App.js)
test command:
git repository: (https://github.com/AkashDR/Namaste-React.git)
keywords: namaste react
author: Akash D R
license: (ISC)
About to write to C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th Feb\package.json:

{
  "name": "namaste-react-02th-feb",
  "version": "1.0.0",
  "description": "Namaste React Course",
  "main": "App.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/AkashDR/Namaste-React.git"
  },
  "keywords": [
    "namaste",
    "react"
  ],
  "author": "Akash D R",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/AkashDR/Namaste-React/issues"
  },
  "homepage": "https://github.com/AkashDR/Namaste-React#readme"
}

Is this OK? (yes)
PS C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th Feb> █
```

```

() package.json > ...
1  {
2    "name": "namaste-react-02th-feb",
3    "version": "1.0.0",
4    "description": "Namaste React Course",
5    "main": "App.js",
6    "scripts": {
7      "test": "echo \\\"Error: no test specified\\\" && exit 1"
8    },
9    "repository": {
10     "type": "git",
11     "url": "git+https://github.com/AkashDR/Namaste-React.git"
12   },
13   "keywords": [
14     "namaste",
15     "react"
16   ],
17   "author": "Akash D R",
18   "license": "ISC",
19   "bugs": {
20     "url": "https://github.com/AkashDR/Namaste-React/issues"
21   },
22   "homepage": "https://github.com/AkashDR/Namaste-React#readme"
23 }
24 |

```

Package.json is configuration for npm. It contains details of all the packages/libraries the project has like version, package Name etc.

2. npm install -D parcel (Installs parcel as Developer Dependency)

The above command adds **package-lock.json** file, node modules folder, parcel package and other dependency package of the parcel

A bundler helps in creating production ready apps. Example of bundlers include webpack, vite, parcel etc.

Difference between Dev Dependency and Normal Dependency:

<https://www.geeksforgeeks.org/difference-between-dependencies-devdependencies-and-peerdependencies/>

While installing parcel or any bundler, if you get this error

npm ERR! 404 Not Found - GET https://registry.npmjs.org/create-react-app/webpack

then we have to set the registry. Only if we set the registry, then npm would download the packages from that registry. Steps to resolve the issue are listed below

Missing repository registry

```
$ npm set registry https://registry.npmjs.org/
```

Clean cache

```
$ npm cache clean  
$ npm rebuild
```

Difference between Caret and Tilde

Caret(^) consider only patch and minor version update automatically. Caret(^) is less safer than Tilde(~) for production app. because here minor feature will also update automatically .

<https://www.geeksforgeeks.org/difference-between-tilde-and-caret-in-package-json/>

<https://www.linkedin.com/pulse/difference-bw-tilde-notation-caret-alok-tiwari/>

3. cat > .gitignore

Create .gitignore file by using above command and add all the files which should not be committed

```
❖ .gitignore  
1  # Excluding following files/item from commit  
2  /node_modules  
3  .parcel-cache/  
4  dist/
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
no changes added to commit (use "git add" and/or "git commit -a")  
PS C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th Feb> git status  
On branch main  
Your branch is up to date with 'origin/main'.
```

```
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
        modified:   App.js  
        modified:   Namaste React Notes.docx  
        modified:   Namaste React Notes.pdf  
        modified:   index.html
```

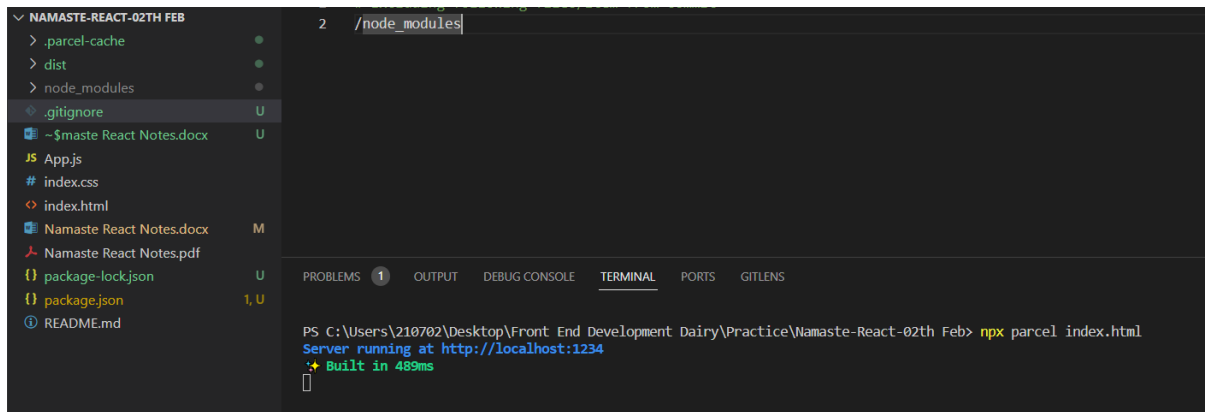
```
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
        .gitignore  
        package-lock.json  
        package.json  
        ~$maste React Notes.docx
```

```
no changes added to commit (use "git add" and/or "git commit -a")  
PS C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th Feb> █
```

How to create .gitignore file

https://www.youtube.com/watch?v=ErJyWO8TGoM&ab_channel=codebasics

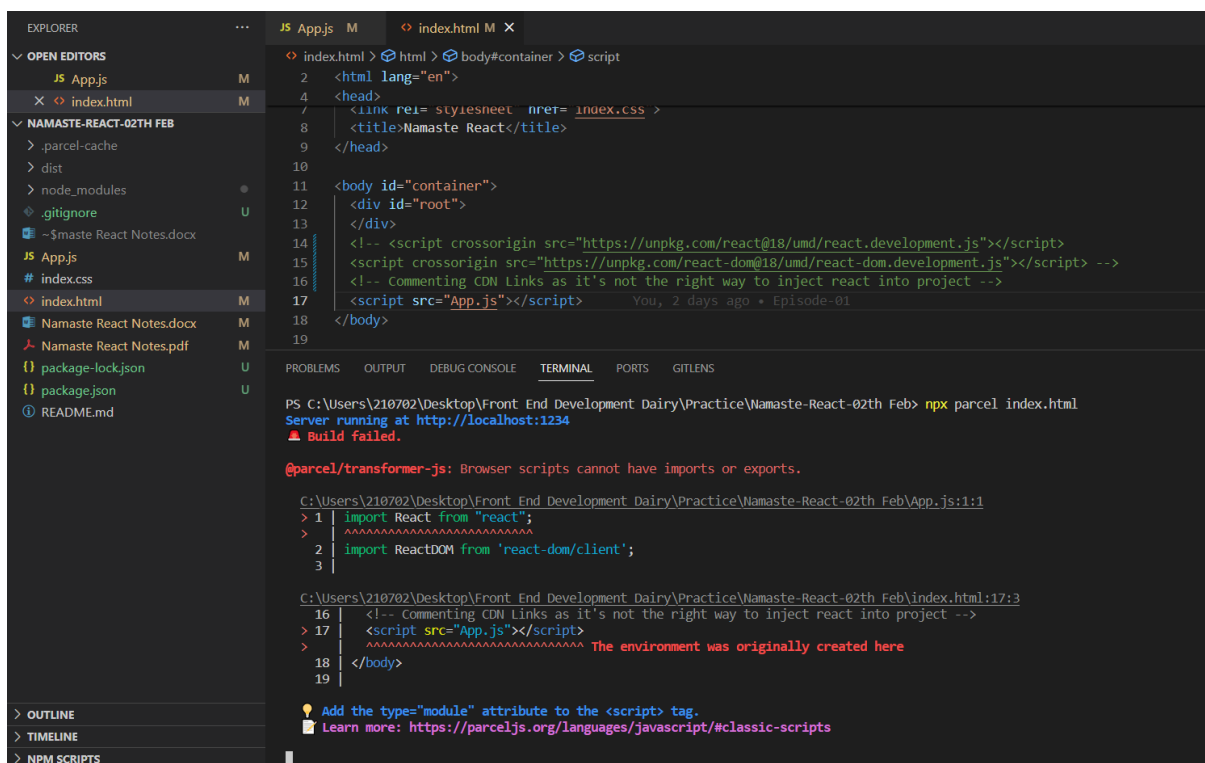
4. `npx create index.html`



The screenshot shows the VS Code interface with a file explorer on the left listing files like `.parcel-cache`, `dist`, `node_modules`, `.gitignore`, and `index.html`. The terminal at the bottom displays the command `npx parcel index.html` and its output: `Server running at http://localhost:1234` and `Built in 489ms`.

Start/Index the application using the above command. Notice that it has created `.parcel-cache` and `dist` folder inside the project. Project starts on port 1234

npx is used to execute the package.



The screenshot shows the VS Code interface with the `index.html` file open. The terminal at the bottom displays the command `npx parcel index.html` and its output, which includes an error: `@parcel/transformer-js: Browser scripts cannot have imports or exports.` The error points to the `App.js` file, which contains `import React from 'react';` and `import ReactDOM from 'react-dom/client';`. The terminal also shows the `index.html` file content, which includes a `<script src='App.js'></script>` tag. A hint suggests adding the `type="module"` attribute to the `<script>` tag.

Error: Browser scripts can't have imports or exports. Solution would be mention attribute type as module in index.html. As `App.js` is not a normal file it is a module.

```
JS App.js M    <> index.html M X
<> index.html > html
2  <html lang="en">
4  <head>
7  <link rel="stylesheet" href="index.css" >
8  <title>Namaste React</title>
9  </head>
10
11 <body id="container">
12   <div id="root">
13   </div>
14   <!-- <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
15   <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script> -->
16   <!-- Commenting CDN Links as it's not the right way to inject react into project -->
17   <script type="module" src="App.js"></script>
18 </body>
19
20 </html>    You, 2 days ago • Episode-01
```

Error: Expected content key de1e4a02ec63c4eb to exist

```
EXPLORER    ...    JS App.js M X    <> index.html M
v OPEN EDITORS
  X JS App.js M
  <> index.html M
  v NAMASTE-REACT-02TH ...
    v .parcel-cache
      7d3d872b02d671a6-AssetGraph-0
      9e063c879a3e4ec8.txt
      44c3568f9ed7534a-AssetGraph-0
      d2762124c2adb2f0-RequestGraph-0
      data.mdb
      df5040dab5c2f433-BundleGraph-0
      lock.mdb
    > dist
    > node_modules
    > .gitignore
    > ~$maste React Notes.docx
  JS App.js M
  # index.css
  <> index.html M
  Namaste React Notes.docx M
  Namaste React Notes.pdf
  {} package-lock.json U
  {} package.json U
  @ README.md

> OUTLINE
> TIMELINE
v NAMASTE-REACT-02TH ...
  JS App.js > ...
  You, 1 second ago | 1 author (You)
  1 import React from "react";
  2 import ReactDOM from 'react-dom/client';
  3
  4 const heading = React.createElement("h1", { id: "heading" }, "Hello World!");
  5 const root = ReactDOM.createRoot(document.getElementById("root"));
  6 root.render(heading);
  7

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS
Feb\node_modules\nullthrows\nullthrows.js:7:15)
  at AssetGraph.getNodeIdByContentKey (C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th
Feb\node_modules\parcel\graph\lib\ContentGraph.js:67:38)
  at C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th
Feb\node_modules\parcel\core\lib\SymbolPropagation.js:52:82
  at Array.map (<anonymous>)
PS C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th Feb> npx parcel index.html
Server running at http://localhost:1234
Build failed.
Error: Expected content key de1e4a02ec63c4eb to exist
Server running at http://localhost:1234
Build failed.
Error: Expected content key de1e4a02ec63c4eb to exist
PS C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th Feb> npx parcel index.html
Server running at http://localhost:1234
Build failed.
Error: Expected content key de1e4a02ec63c4eb to exist
Server running at http://localhost:1234
Build failed.
```

If you are using parcel then try to delete ".parcel-cache" folder. And then Rerun the build to solve the above issue

What does Parcel do?

Read about these concepts in this page (1st page itself).

<https://parceljs.org/>

```
# Parcel
- Dev Build
- Local Server
- HMR = Hot Module Replacement
- File Watching Algorithm - written in C++
- Caching - Faster Builds
- Image Optimization
- Minification
- Bundling
- Compress
- Consistent Hashing
- Code Splitting
- Differential Bundling - support older browsers
- Diagnostic
- Error Handling
- HTTPs
- Tree Shaking - remove unused code
- Different dev and prod bundles
```

Read about few of the definitions from below link

<https://legacy.reactjs.org/docs/code-splitting.html>

Differential bundling is the concept of sending various copies of your code to different targets and letting the browser decide which one to download

How to create dev build?

Code: `npx parcel index.html` ("Notice the keyword build missing")

How to create production ready build?

Code: `npx parcel build index.html`

When you run this, the production build gets created in dist folder after all the optimization (Done by parcel).

```
PS C:\Users\210702\Desktop\Front End Development Dairy\Practice\Namaste-React-02th Feb> npx parcel build index.html
🚀 Built in 2.73s

dist\index.html          366 B    1.18s
dist\index.a28d1165.css   98 B     70ms
dist\index.b0ff1e7e.js  138.77 KB  1.52s
```

Error: @parcel/namer-default: Target "main" declares an output file path of "App.js" which does not match the compiled bundle type "html".

```
package.json > {} repository
1  {
2    "name": "namaste-react",
3    "version": "1.0.0",
4    "description": "This is Namaste React by Akshay Saini",
5    "main": "App.js",
6    "scripts": {
7      "test": "jest"
8    },
9    "repository": {
10     "type": "git",
11     "url": "git+https://github.com/namastedev/namaste-react.git"
12   },
13   "keywords": [],
14   "author": "Akshay Saini",
15   "license": "ISC",
16   "bugs": {
17     "url": "https://github.com/namastedev/namaste-react/issues"
18   },
19   "homepage": "https://github.com/namastedev/namaste-react#readme",
20   "devDependencies": {
21     "parcel": "^2.8.3",
22     "process": "^0.11.10"
23   },
24 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** GITLENS COMMENTS

```
4 | "description": "This is Namaste React by Akshay Saini",
> 5 | "main": "App.js",
> |      ^^^^^^^^^ Did you mean "App.html"?
6 | "scripts": {
7 |   "test": "jest"
8 | }
9 |
```

💡 Try changing the file extension of "main" in package.json.

To solve this error remove "main" in package.json

```
{ } package.json > ...
1  {
2    "name": "namaste-react",
3    "version": "1.0.0",
4    "description": "Namaste React Course",
5    "main": "App.js",
6    "scripts": {
```

Like this

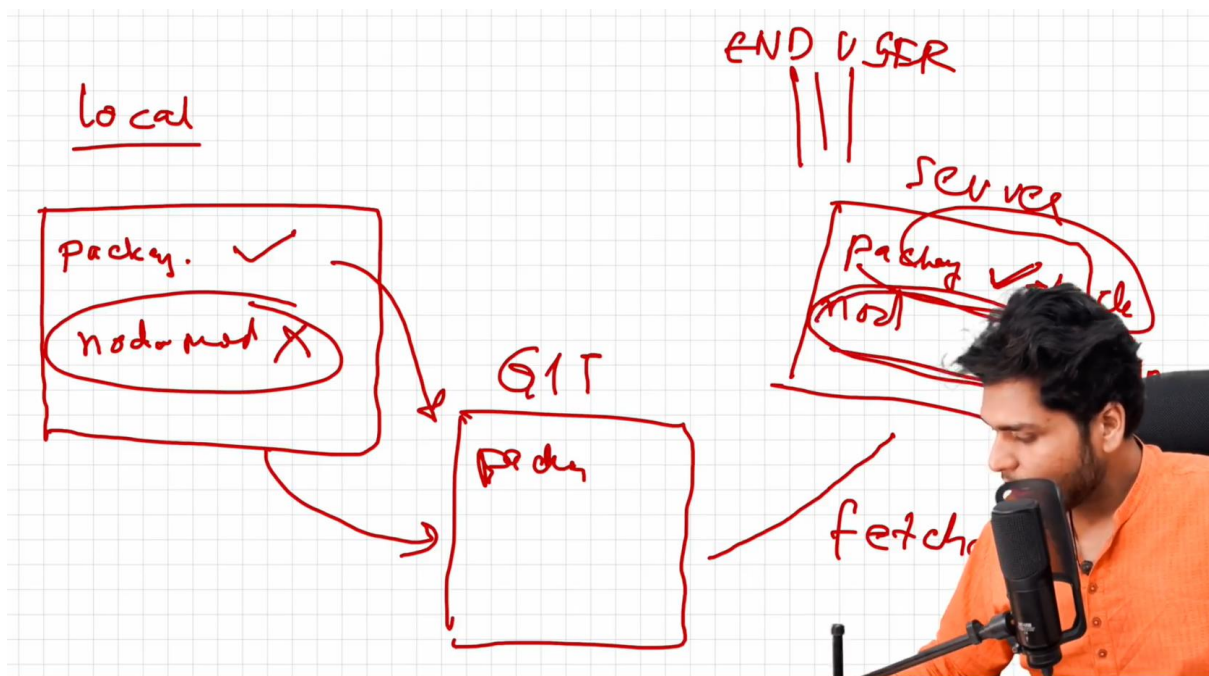
```

{} package.json > ...
1  {
2    "name": "namaste-react",
3    "version": "1.0.0",
4    "description": "Namaste React Course",
   ▶ Debug
5    "scripts": {
6      "test": "echo \\\"Error: no test specified\\\" && ex

```

Flow of Application:

Server fetches package.json & package-lock.json from Git and executes the command to create production build. And hosts that build to the end user



How to make your app compatible to older versions of browser?

Use browserlist package for achieving compatibility.

<https://browserslist.dev/?q=bGFzdCAyIHZlcuNpb25z>

<https://github.com/browserslist/browserslist#query-composition>

Configuration is as follows:

{ } package.json 1, U X

{ } package.json > ...

```
1  {
2    "name": "namaste-react",
3    "version": "1.0.0",
4    "description": "Namaste React Course",
5    "scripts": {
6      "test": "echo \"Error: no test specified\" && exit 1"
7    },
8    "repository": {
9      "type": "git",
10     "url": "git+https://github.com/AkashDR/Namaste-React.git"
11   },
12   "keywords": [],
13   "author": "Akash D R",
14   "license": "ISC",
15   "bugs": {
16     "url": "https://github.com/AkashDR/Namaste-React/issues"
17   },
18   "homepage": "https://github.com/AkashDR/Namaste-React#readme",
19   "devDependencies": {
20     "parcel": "^2.11.0",
21     "process": "^0.11.10"
22   },
23   "dependencies": {
24     "react": "^18.2.0",
25     "react-dom": "^18.2.0"
26   },
27   "browserlist": [
28     "last 10 Chrome version",
29     "last 10 Firefox version",
30     "last 10 versions"
31   ]
32 }
33
```

Promises: How to extract data from Promises?

```
const cart = ["Shoes", "Pants", "Watches"];

function createOrder(cart, proceedToPayment) {
  console.log("Order Created", cart);
  console.log("Lets Wait");
  setTimeout(() => {
    proceedToPayment();
  }, 5000);
}

function proceedToPayment() {
  console.log("Proceeded to Payment");
}

createOrder(cart, proceedToPayment);

const URL1 = "https://api.github.com/users/mojombo";
fetch(URL1)
  .then((res) => {
    return res?.json();
  })
  .then((data) => {
    console.log(data, "data");
  });

let promise = new Promise((resolve, reject) => {
  reject("Hello JavaScript Failed!");
});

promise.then((result) => console.log(result)).catch(
  res=>{
    console.log(res)
  }
);

const URL2 = "https://api.github.com/users/mojombo";
const user = fetch(URL2)
  .then((res) => {
    return res?.json();
  })
  .then((data) => {
    console.log(data, "data");
  });
```


Check the state of Promise

```
function createOrder(cart) {
  const promise = new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("12345");
      console.log(promise, "promise2");
    }, 5000);
  });
  console.log(promise, "promise1");
  return promise;
}

createOrder(cart)
  .then((res) => {
    console.log(res, "res");
  })
  .catch((err) => {
    console.log(err, "err");
  });
```

```
▼ Promise {<pending>} i 'promise1'
  ► [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "12345"
```

```
▼ Promise {<fulfilled>: '12345'} i 'promise2'
  ► [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "12345"
```

12345 res

>

Usage of finally in Promises.

```
function createOrder(cart) {
  const promise = new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("12345");
      console.log(promise, "promise2");
    }, 5000);
  });
```

```

});
console.log(promise, "promise1");
return promise;
}

createOrder(cart)
  .then((res) => {
    console.log(res, "res");
  })
  .catch((err) => {
    console.log(err, "err");
  }).finally(err=>{
    console.log("Just Print - No Matter What")
  });

```

► Promise {<pending>} 'promise1'

► Promise {<fulfilled>: '12345'} 'promise2'

12345 res

Just Print - No Matter What

> |

Promise Chaining:

Notice the word return inside then block. You should always return if you want to create Promise chain. Else it would lead to Pyramid kind of structure similar to Callback hell.

```

const cart = ["shoes", "pants", "kurtas"];

function createOrder(cart) {
  return new Promise((resolve, reject) => {
    resolve("Cart Creation Successful");
  });
}

function proceedToPayment(orderId) {
  return new Promise((resolve, reject) => {
    resolve("Payment Successful");
  });
}

createOrder(cart)

```

```

.then((res) => {
  console.log(res);
  return res;
})
.then((res) => {
  console.log(res);
  return proceedToPayment(res);
})
.then((res) => {
  console.log(res);
  return proceedToPayment(res);
})
.then((res) => {
  console.log(res);
  return proceedToPayment(res);
})
.then((res) => {
  console.log(res);
})
.catch((err) => {
  console.log(err, "err");
});

```

Cart Creation Successful

Cart Creation Successful

Payment Successful

Payment Successful

Payment Successful

>

Catch checks only the errors that come above it. It won't check below it. If we cart creation is failed, we can still move to payment. See Details below.

```

const cart = ["shoes", "pants", "kurtas"];

function createOrder(cart) {
  return new Promise((resolve, reject) => {
    reject("Cart Creation Failed");
  });
}

function proceedToPayment(orderId) {
  return new Promise((resolve, reject) => {
    resolve("Payment Successful");
  });
}

```

```

createOrder(cart)
  .then((res) => {
    console.log(res);
    return res;
  })
  .catch((err) => {
    console.log(err);
    return err
  })
  .then((res) => {
    console.log(res);
    return proceedToPayment(res);
  })
  .then((res) => {
    console.log(res);
    return proceedToPayment(res);
  })
  .catch((err) => {
    console.log(err, "err");
  });

```

Notice first Catch Block above. It catches only failures in Create Cart Function

Cart Creation Failed	index.js:67
Cart Creation Failed	index.js:71
Payment Successful	index.js:25

```

47 const cart = ["shoes", "pants", "kurtas"];
48
49 function createOrder(cart) {
50     return new Promise((resolve, reject) => {
51         reject("Cart Creation Failed");
52     });
53 }
54
55 function proceedToPayment(orderId) {
56     return new Promise((resolve, reject) => {
57         reject("Payment Failure");
58     });
59 }
60
61 createOrder(cart)
62     .then((res) => {
63         console.log(res);
64         return res;
65     })
66     .catch((err) => {
67         console.log(err);
68         return err
69     })
70     .then((res) => {
71         console.log(res);
72         return proceedToPayment(res);
73     })
74     .then((res) => {
75         console.log(res);
76         return proceedToPayment(res);
77     })
78     .catch((err) => {
79         console.log(err, "err");
80     });
81

```

Notice now, the Promise failed at both places, but first catch block caught only Cart Failure error.

Cart Creation Failed	index.js:67
Cart Creation Failed	index.js:71
Payment Failure err	index.js:79

> |

Async & Await

Syntax for Async function

```

async function getData() {
    return "Namaste Javascript";
}

const data = getData();

console.log(data, "data");

```

Async function always returns a Promise. If you return any value like string or anything, it will wrap the values inside a Promise and return it.

```
▼ Promise {<fulfilled>: 'Namaste Javascript'} 'data'  
  ► [[Prototype]]: Promise  
    [[PromiseState]]: "fulfilled"  
    [[PromiseResult]]: "Namaste Javascript"
```

[index.js:81](#)

Extracting data out of Promise

```
async function getData() {  
  return "Namaste Javascript";  
}  
  
const data = getData();  
  
console.log(data, "data");  
  
data  
  .then((response) => {  
    console.log(response, "response");  
  })  
  .catch((err) => {  
    console.log(err);  
  });
```

```
► Promise {<fulfilled>: 'Namaste Javascript'} 'data'  
Namaste Javascript response
```

[index.js:81](#)

[index.js:85](#)

Example when returning a Promise

```
async function getData() {  
  const p= new Promise((resolve,reject)=>{  
    resolve("Hello World")  
  })  
  return p;  
}  
  
const data = getData();  
  
console.log(data, "data");  
  
data  
  .then((response) => {  
    console.log(response, "response");  
  })  
  .catch((err) => {  
    console.log(err);  
  });
```

It won't wrap with another Promise when returning a Promise. It juts returns it.

```
▼ Promise {<pending>} ⓘ 'data'
  ► [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "Hello World"
```

Hello World response

>

Using Async and Await together

```
const promise = new Promise((resolve, reject) => {
  resolve("Hello World");
});

console.log(promise, "promise");

async function handlePromise() {
  const value = await promise;
  console.log(value, "value");
}

handlePromise()
```

```
▼ Promise {<fulfilled>: 'Hello World'} ⓘ 'promise'
  ► [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "Hello World"
```

Hello World "value"

>

Await keyword can only be used inside an Async Function. It resolves Promise

How Async and Await is different from Normal Promises?

What will be printed first (Using Normal Promises)?

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Am a Promise");
  }, 10000);
});

function handlePromise() {
  promise.then((res) => {
    console.log(res, "Promise");
  });
  console.log("Namaste JavaScript");
}

handlePromise();
```

Namaste JavaScript
Am a Promise 'Promise'

[index.js:85](#)
[index.js:83](#)

Using Async and Await to handle Promises

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Am a Promise");
  }, 10000);
});

async function handlePromise() {
  // JS Engine waits for Promise to resolve when using Async and Await
  const value = await promise; // Program wait for 10 Sec here
  console.log(value, "value");
  console.log("Namaste JavaScript");
}

handlePromise();
```

When using await JS Engine waits for Promise to be resolved. But this is not the case when using Normal Promise. Below values will be printed at a time after the timeout of 10 seconds


```
Am a Promise "value" index.js:83
Namaste JavaScript index.js:84
>
```

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Am a Promise");
  }, 10000);
});

async function handlePromise() {
  // JS Engine waits for Promise to resolve when using Async and Await
  const value = await promise; // Program wait for 10 Sec here
  console.log(value, "value");

  const value2 = await promise; // Program wait for 10 Sec here
  console.log(value2, "value2");

  console.log("Namaste JavaScript");
}

handlePromise();
```

Here also the whole values will be printed after a gap of 10 seconds (Not 20 Seconds)

```
Am a Promise "value" index.js:84
Am a Promise "value2" index.js:87
Namaste JavaScript index.js:89
>
```

Even if you create 2 separate Promises like below with different Timeouts. The response will be printed after 10 seconds. At promise1, 10 seconds wait time. At promise2, no wait time because it had only 5s wait time which is already taken place when executing promise1

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Am a Promise");
  }, 10000);
});

const promise2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Am a Promise 2");
  }, 5000);
});
```

```

});

async function handlePromise() {
  // JS Engine waits for Promise to resolve when using Async and Await
  const value = await promise; // Waits for 10 seconds
  console.log(value, "value");

  const value2 = await promise2; // Won't wait as it has already resolved in
  first 5 seconds (While waiting for 10 Seconds in first Promise)
  console.log(value2, "value2");

  console.log("Namaste JavaScript");
}

handlePromise();

```

Am a Promise "value"	index.js:90
Am a Promise 2 "value2"	index.js:93
Namaste JavaScript	index.js:95

Reversing the Timeouts. Note the difference. First Promise gets resolved in 5s and then in another 5s the second Promise gets resolved.

```

const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Am a Promise");
  }, 5000);
});

const promise2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Am a Promise 2");
  }, 10000);
});

async function handlePromise() {
  // JS Engine waits for Promise to resolve when using Async and Await
  const value = await promise;
  console.log(value, "value"); // Will wait for 5 Seconds

  const value2 = await promise2;
  console.log(value2, "value2"); // Will wait for another 5 seconds not 10
  seconds

  console.log("Namaste JavaScript");
}

handlePromise();

```

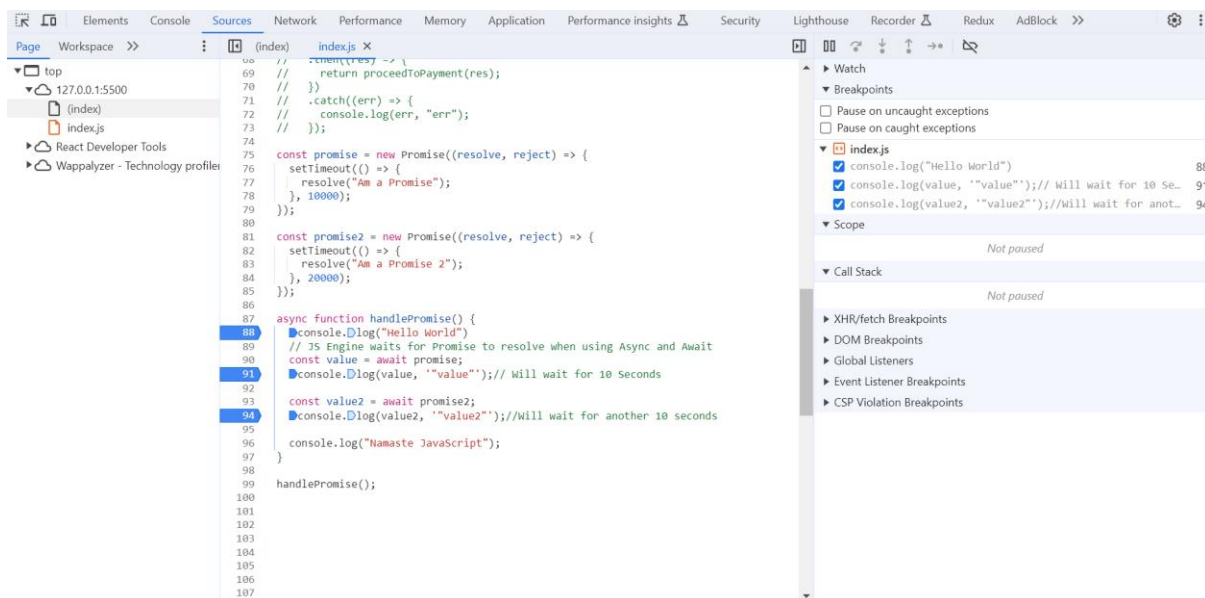
```

Am a Promise "value"                                     index.js:90
Am a Promise 2 "value2"                                   index.js:93
Namaste JavaScript                                       index.js:95

```

Time, Tide and JavaScript waits for None. JS Engine appears waiting (Only looks like), but it is not actually waiting. If it is waiting, then screen would Freeze.

When it sees await keyword, it actually suspends execution of handlePromise() function. It won't block the call stack (JS has only one call stack). Once the promise1 is resolved, handlePromise() again comes into action. It will start execution from where it left. Again, it will check for promise 2, as it has **await keyword**. So it would be suspended again. It will move out of call stack. Comes back when promise2 is resolved. It is not JS Engine that is waiting, it is the handle Promise function that gets suspended.



You can notice suspend of handlePromise() function in Call Stack. JS Engine, separately tracks the function.

<https://api.github.com/> Can be used for sample APIs

How fetch works?

Fetch Returns a Response Object. Response object has a body which is a readable stream, if you want to convert Readable Stream into a json you have to do **Response.json()**.

Response.json() is a promise which must be resolved to get the data.

▼ *Promise {<pending>}* ⓘ 'data'

- ▶ *[[Prototype]]*: Promise
- [[PromiseState]]*: "fulfilled"
- ▼ *[[PromiseResult]]*: Response
 - body*: (...)
 - bodyUsed*: false
 - ▶ *headers*: Headers {}
 - ok*: true
 - redirected*: false
 - status*: 200
 - statusText*: "OK"
 - type*: "cors"
 - url*: "https://api.github.com/users/amogh"
 - ▶ *[[Prototype]]*: Response

```
const API_URL = "https://api.github.com/users/akshaymarch7";

// await can only be used inside an async function
async function handlePromise() {

  const data = await fetch(API_URL);

  const jsonValue = await data.json()

  fetch().then(res => res.json()).then(res => console.log())

  // fetch() => Response.json() => jsonValue

}
handlePromise();
```

Await gives resolved Promise

```
async function getData() {
  const data = await fetch("https://api.github.com/users/amogh");
  const userData = await data.json(); // Await would give resolved Promise
  console.log(userData, 'userData')
}

getData()
```

You will have data as below

```

▼ {login: 'amogh', id: 592374, node_id: 'MDQ6VXNlcjU5MjM3NA==', avatar_url: 'https://avatars.githubusercontent.com/u/592374?v=4', gravatar_id: '', ...} userData ind
  avatar_url: "https://avatars.githubusercontent.com/u/592374?v=4"
  bio: null
  blog: ""
  company: null
  created_at: "2011-01-31T09:01:04Z"
  email: null
  events_url: "https://api.github.com/users/amogh/events{/privacy}"
  followers_url: "https://api.github.com/users/amogh/followers"
  following_url: "https://api.github.com/users/amogh/following{/other_user}"
  gists_url: "https://api.github.com/users/amogh/gists{/gist_id}"
  gravatar_id: ""
  hireable: null
  html_url: "https://github.com/amogh"
  id: 592374
  location: null
  login: "amogh"
  name: null
  node_id: "MDQ6VXNlcjU5MjM3NA=="
  organizations_url: "https://api.github.com/users/amogh/orgs"
  public_gists: 0
  public_repos: 0
  received_events_url: "https://api.github.com/users/amogh/received_events"
  repos_url: "https://api.github.com/users/amogh/repos"
  site_admin: false
  starred_url: "https://api.github.com/users/amogh/starred{/owner}/{repo}"
  subscriptions_url: "https://api.github.com/users/amogh/subscriptions"
  twitter_username: null
  type: "User"
  updated_at: "2015-04-09T20:21:10Z"
  url: "https://api.github.com/users/amogh"
  [[Prototype]]: Object
>

```

But in case of Promise, you have to resolve separately **using .then**. Notice the difference below

```

// Using Async and Await
async function getData() {
  const data = await fetch("https://api.github.com/users/amogh");
  const userData = await data.json(); // Await would give resolved Promise
  console.log(userData, 'userData')
}

getData()

// Using Promise

function getData2() {
  fetch("https://api.github.com/users/amogh")
    .then((res) => {
      return res?.json();
    })
    .then((data) => {
      console.log(data, "data");
    });
}

getData2()

```

Error Handling in Async and Await has to be done using try and catch

```
// Using Async and Await
const URL = "https://invalidUrl";
async function getData() {
  try {
    const data = await fetch(URL);
    const userData = await data.json();
    console.log(userData, "userData");
  } catch (err){
    console.log(err, "err");
  }
}

getData()

// Using Promise

function getData2() {
  fetch("https://api.github.com/users/amogh")
    .then((res) => {
      return res?.json();
    })
    .then((data) => {
      console.log(data, "data");
    })
    .catch((err) => console.log(err, err));
}

getData2();
```

✖ GET <https://invalidurl/> net::ERR_NAME_NOT_RESOLVED

[index.js:92](#) ⓘ

TypeError: Failed to fetch
at [getData \(index.js:92:24\)](#)
at [index.js:100:1](#) "err"

[index.js:96](#)

Another way is as follows:

As Async function always returns a Promise

```
// Using Async and Await
const URL = "https://invalidUrl";
async function getData() {
  const data = await fetch(URL);
  const userData = await data.json();
  console.log(userData, "userData");
}

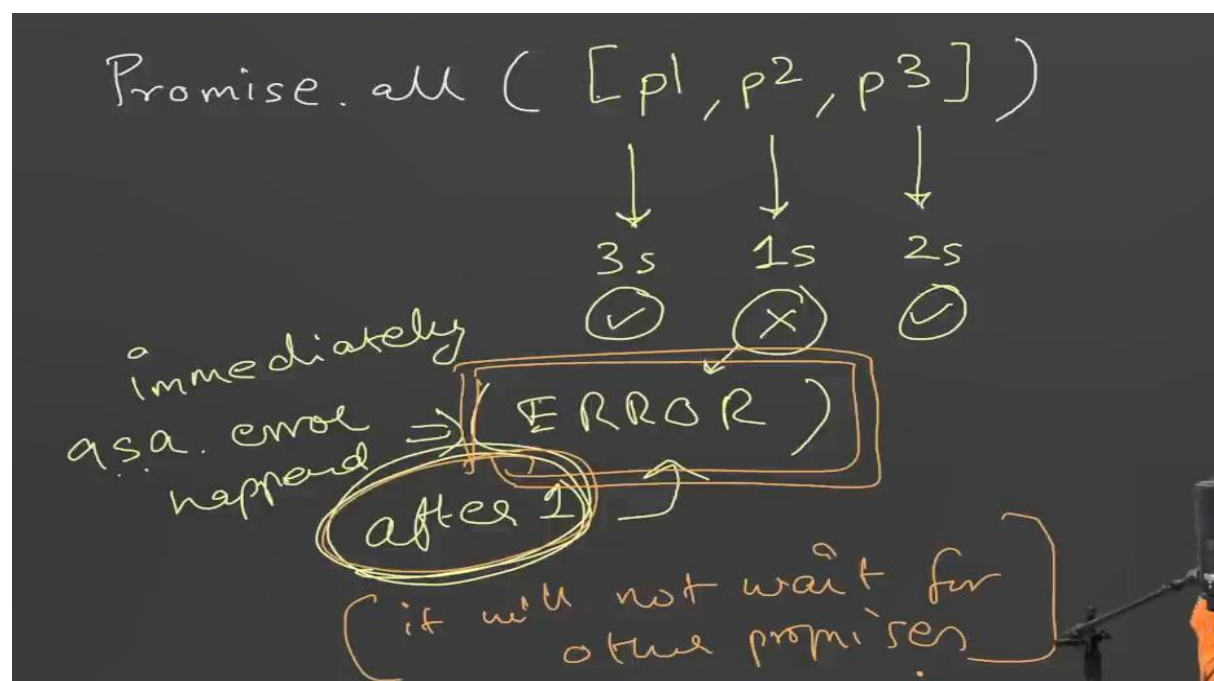
getData().catch((err) => console.log(err, "err"));
```

✖ GET <https://invalidurl/> net::ERR_NAME_NOT_RESOLVED

TypeError: Failed to fetch
at getData ([index.js:91:24](#))
at [index.js:96:1](#) 'err'

Async-Await is just syntactical sugar for Promises. Behind the scenes it also uses then and catch. In general, Async-Await solves the complexity of Promise Chaining. It's always a personal choice which to use.

Promise.all



```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p2");
  }, 1000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p3");
  }, 2000);
});

Promise.all([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

► (3) ['p1', 'p2', 'p3'] 'res'

>

```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p2");
  }, 1000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p3");
  }, 2000);
});

Promise.all([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

p2 err

>

```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p2");
  }, 1000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p3");
  }, 2000);
});

Promise.all([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

p2 err

>


```

const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p2");
  }, 1000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p3");
  }, 2000);
});

Promise.all([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));

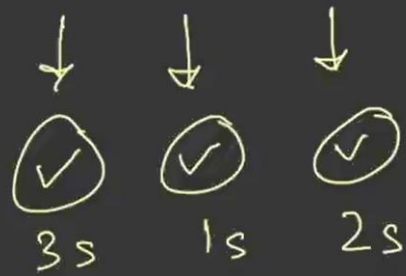
```

p3 err

>

Promise.allSettled

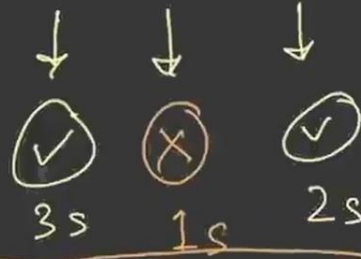
Promise.allSettled([p1, p2, p3])



after 3s [val1, val2, val3]

Promise.allSettled([p1, p2, p3])

wait for all promises
to settled



after 3s [val 1, err 2, val 3]

```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p2");
  }, 1000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p3");
  }, 2000);
});

Promise.allSettled([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

▼ (3) [...], [...], [...]

- ▶ 0: {status: 'fulfilled', value: 'p1'}
- ▶ 1: {status: 'fulfilled', value: 'p2'}
- ▶ 2: {status: 'fulfilled', value: 'p3'}

length: 3

▶ [[Prototype]]: Array(0)

```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p2");
  }, 1000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p3");
  }, 2000);
});

Promise.allSettled([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

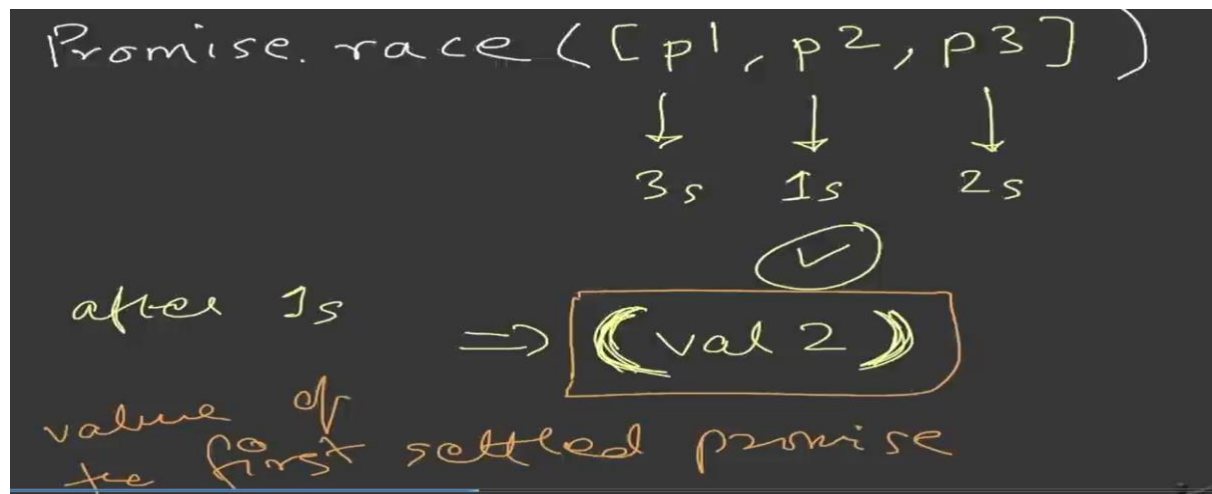
▼ (3) [...], [...], [...]

- ▶ 0: {status: 'fulfilled', value: 'p1'}
- ▶ 1: {status: 'rejected', reason: 'p2'}
- ▶ 2: {status: 'rejected', reason: 'p3'}

length: 3

▶ [[Prototype]]: Array(0)

Promise.race



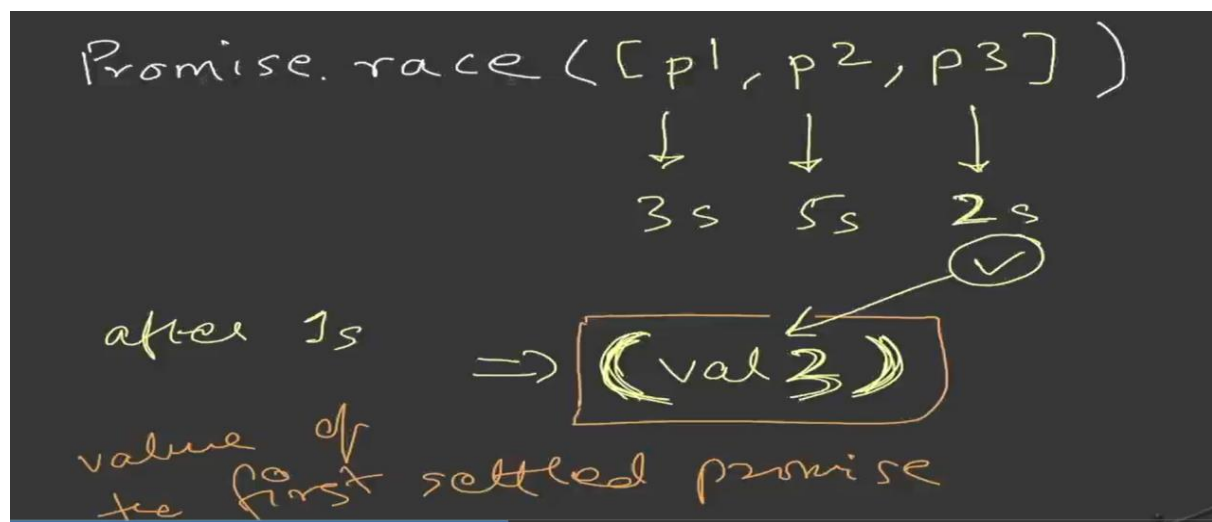
```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p2");
  }, 1000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p3");
  }, 2000);
});

Promise.race([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

p2 res



```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p2");
  }, 5000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p3");
  }, 2000);
});

Promise.race([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

p3 res

> |

Promise.race([p1, p2, p3])

↓ ↓ ↓
3s 5s 2s

after 2s

⇒ (ERROR)
 (p3)

return
 result of
 first settled promise

```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p2");
  }, 5000);
});

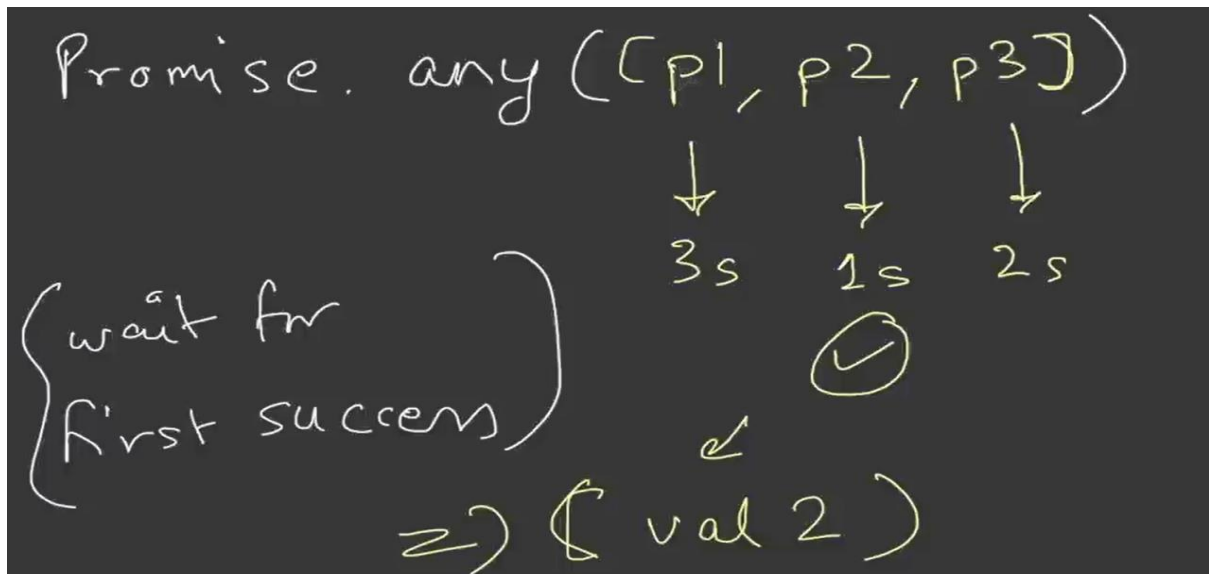
const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p3");
  }, 2000);
});

Promise.race([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

p3 err

>

Promise.any



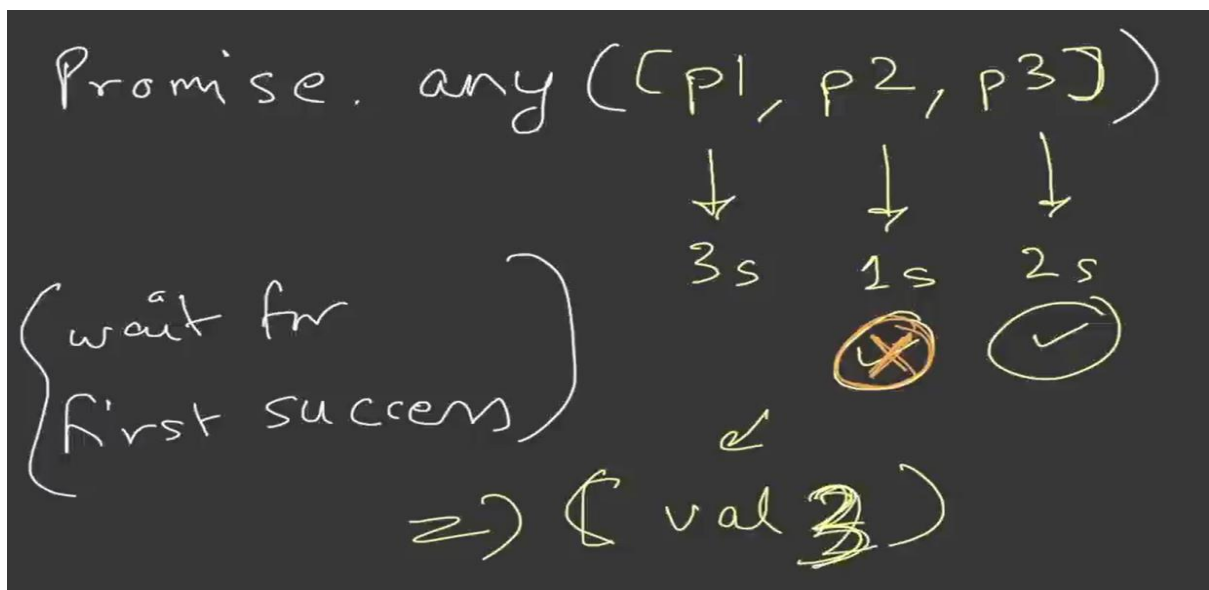
```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p2");
  }, 5000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p3");
  }, 2000);
});

Promise.any([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

p3 res
> |



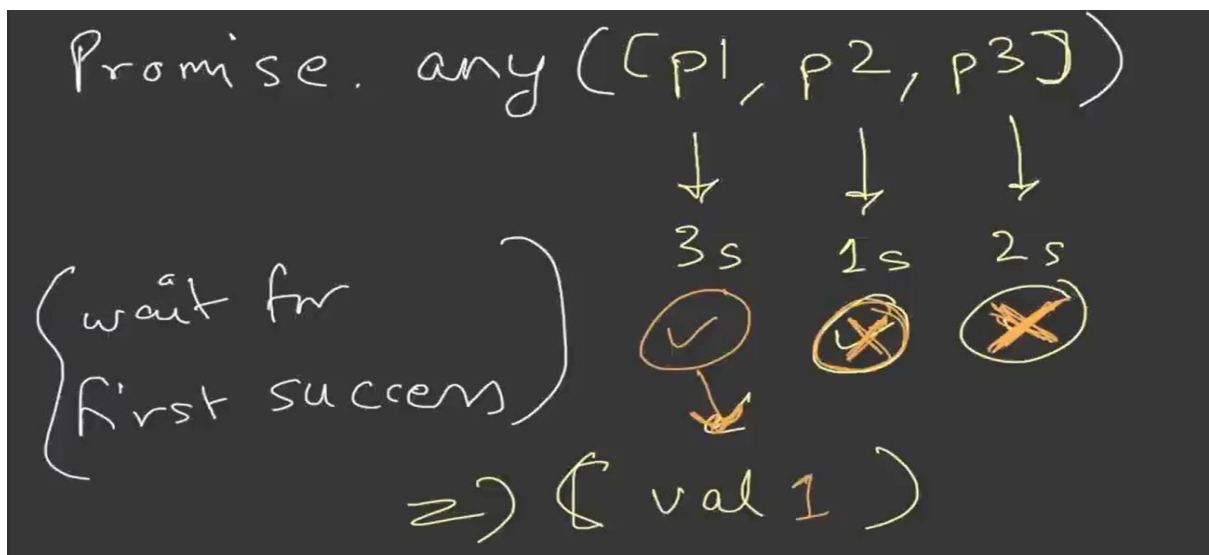
```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p2");
  }, 5000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p3");
  }, 2000);
});

Promise.any([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

p1 res
>



```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p1");
  }, 3000);
});

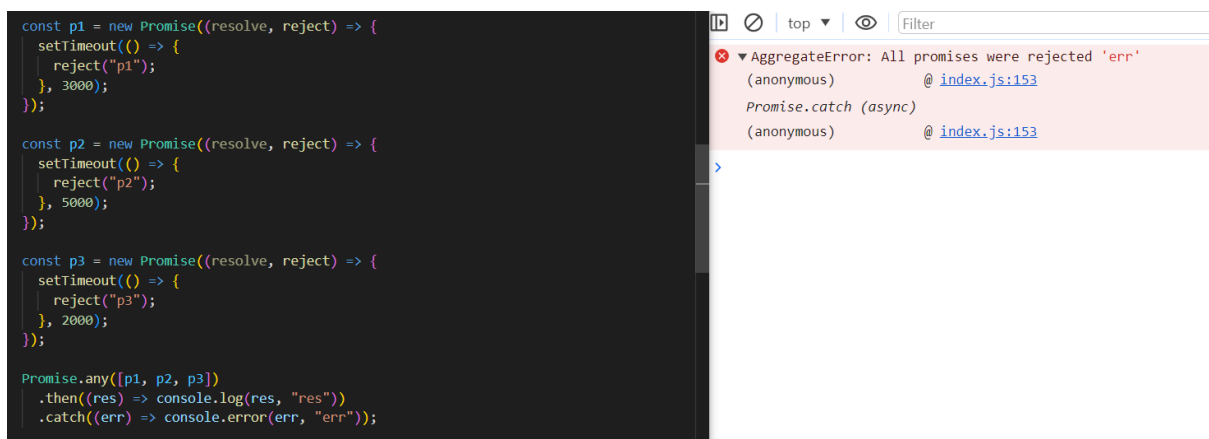
const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("p2");
  }, 5000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p3");
  }, 2000);
});

Promise.any([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.log(err, "err"));
```

p2 res
> |

If all the promises are rejected, then it gives AggregateError



```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p2");
  }, 5000);
});

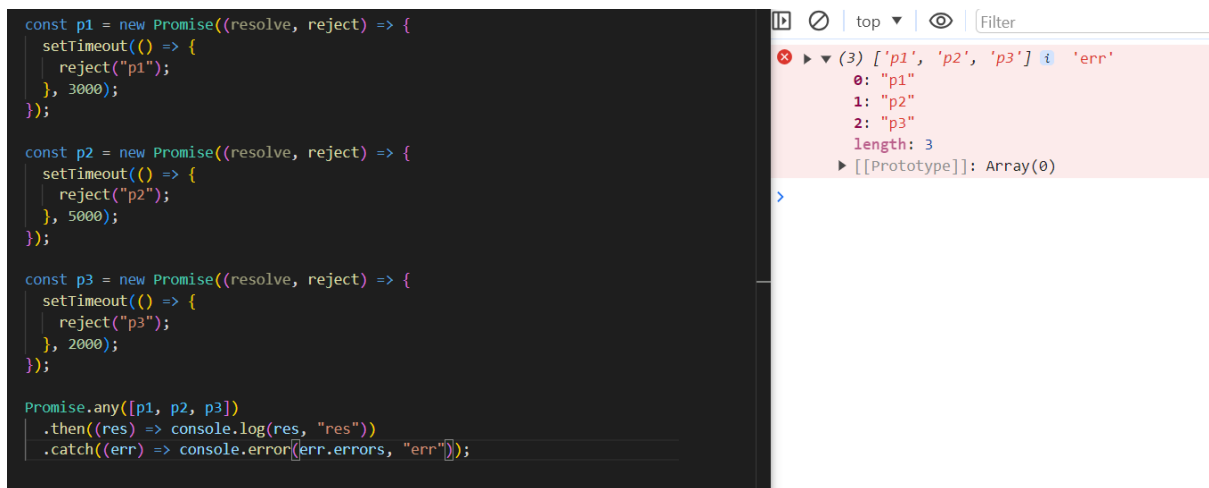
const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p3");
  }, 2000);
});

Promise.any([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.error(err, "err"));
```

The console shows an **AggregateError** with the message "All promises were rejected 'err'". The stack trace includes the following frames:

- (anonymous) @ index.js:153
- Promise.catch (async)
- (anonymous) @ index.js:153

Handling Aggregate Error (You have to use err.error)



```
const p1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p1");
  }, 3000);
});

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p2");
  }, 5000);
});

const p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject("p3");
  }, 2000);
});

Promise.any([p1, p2, p3])
  .then((res) => console.log(res, "res"))
  .catch((err) => console.error(err.errors, "err"));
```

The console shows an **AggregateError** with the message "All promises were rejected 'err'". The error details are expanded, showing an array of three error objects:

- 0: "p1"
- 1: "p2"
- 2: "p3"
- length: 3
- [[Prototype]]: Array(0)

Lecture-03 Laying the Foundation