TEAM # 4

Improving <u>Reliability</u> and ***Performance***

Sergio Pichardo

Lisa Melo

Joey Guillaume

OVERVIEW

1. Promise.any() method

2. Error handling

3. Reliability

4. Performance

5. FunnyFish Demo

Promise.any

How the Promise.any static method works

Lisa Melo

OUTLINE

- Quick Review
  - `Promise.all`
  - `Promise.allSettled`
- `Promise.any`
- `Example:Promise.any`
- Error Handling
  - Empty Array
  - All promises reject
  - Custom error message

QUICK REVIEW

`Promise.all()`

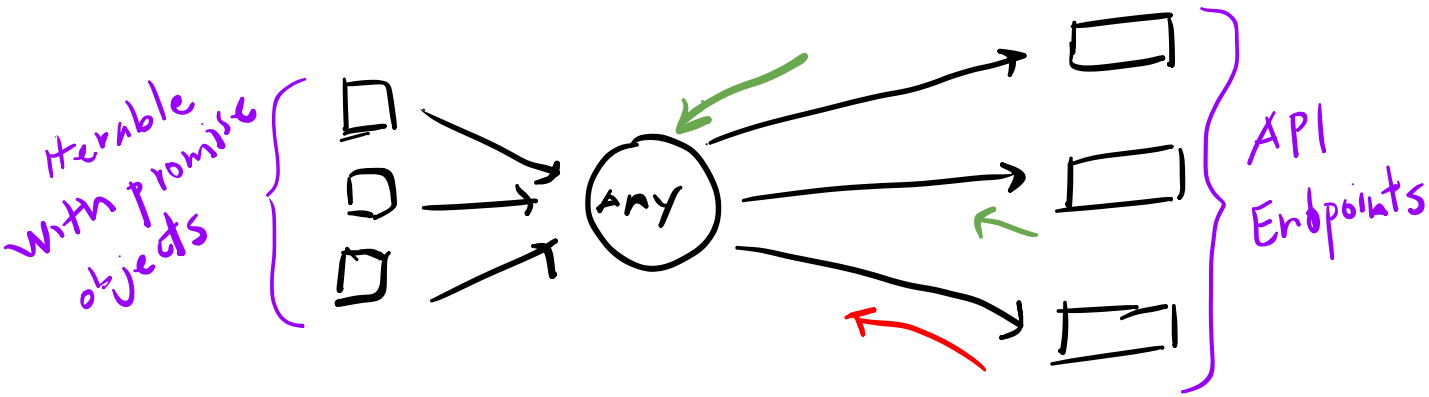
Returns a single promise that resolves if all the given promises resolve. The returned promise resolves to an array with all the fulfilled values. If one promise fails, Promise.all() stops execution and immediately rejects.

`Promise.allSettled()`

Returns a single promise that resolves when all given promises either fulfill or reject. The returned promise resolves to an array of objects, each object has a status property and a value (or rejection) reason property.

Lisa Melo

Promise.any()



- Takes an iterable of promise objects
- Returns a pending promise
- The promise fulfills as soon as **any** of the promises in the iterable resolves, **with the value of the resolved promise**.

Promise.any

```
1  const pErr = new Promise((resolve, reject) => {
2    reject("Always fails");
3  });
4
5  const pSlow = new Promise((resolve, reject) => {
6    setTimeout(resolve, 500, "Done eventually");
7  });
8
9  const pFast = new Promise((resolve, reject) => {
10   setTimeout(resolve, 100, "Done quick");
11 });
12
13 Promise.any([pErr, pSlow, pFast]).then((value) => {
14   console.log(value);
15   // pFast fulfils first
16 })
17 // expected output: "Done quick"
```

Source: MDN

ERROR HANDLING

Promise.any() **rejects** in two situations:

An **empty iterable** was passed in as an argument → Promise.any([])

All promises **reject** → Promise.any(✗, ✗, ✗, ✗)

ERROR HANDLING: EMPTY ARRAY

```
let resources = [];

let promiseAny = Promise.any(resources);

promiseAny.then(response => {})
.catch(error => {
  console.log(error); // AggregateError: All promises were rejected
  console.log(error.errors); // []
});
```


ERROR HANDLING:
ALL PROMISES REJECT

```
function rejectPromise(reason, delay) {
  return new Promise(
    (resolve, reject) => setTimeout(() => reject(reason), delay)
  );
}

let rejectedOne = rejectPromise('Something went wrong', 1000);
let rejectedTwo = rejectPromise('Wrong URL', 2000);

let resources = [rejectedOne, rejectedTwo];

let promiseAny = Promise.any(resources);

promiseAny.then(response => {})
.catch(error => {
  console.log(error); // AggregateError: All promises were rejected
  console.log(error.errors); // ['Something went wrong', 'Wrong URL']
});
```


CUSTOM
ERROR MESSAGE

```javascript
function rejectPromise(reason, delay) {
  return new Promise(
    (resolve, reject) => setTimeout(() => reject(reason), delay)
  );
}

let rejectedOne = rejectPromise('Something went wrong', 1000);
let rejectedTwo = rejectPromise('Wrong URL', 2000);

let resources = [rejectedOne, rejectedTwo];

function getData() {
  return Promise.any(resources)
  .catch(() => {
    return Promise.reject(
      new Error('Unable to access the API')
    );
  });
}

getData().then(
  (response) => console.log(response.title),
  (error) => console.error(error)
  //error.errors is undefined
);

```

Quick Review
Promise.any
`Example:Promise.any`
Error Handling

Lisa Melo

...Questions?

THANK YOU!

Next, **Sergio** will talk to you about another important topic, **reliability.**

Lisa Melo

IMPROVING RELIABILITY

How **Promise.any** can help us build more reliable applications

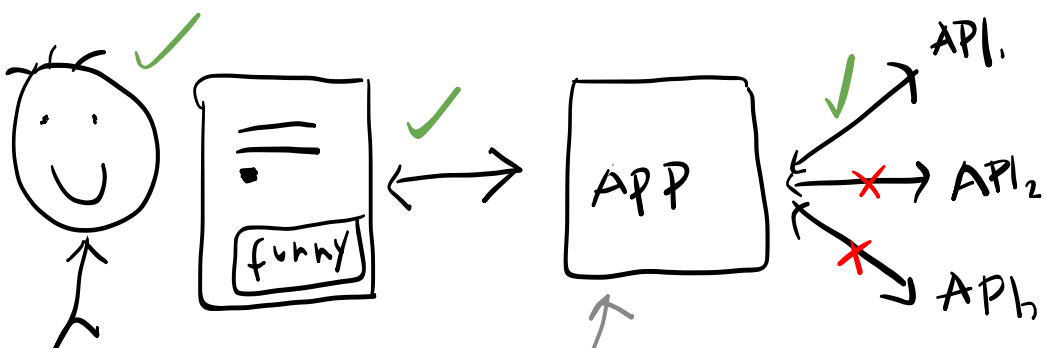
Sergio Pichardo

OUTLINE

- Reliability
- Single Point of Failure
- Reliability through Redundancy

RELIABILITY

The ability of an app to **avoid** failure and to **recover** quickly from it when it occurs.

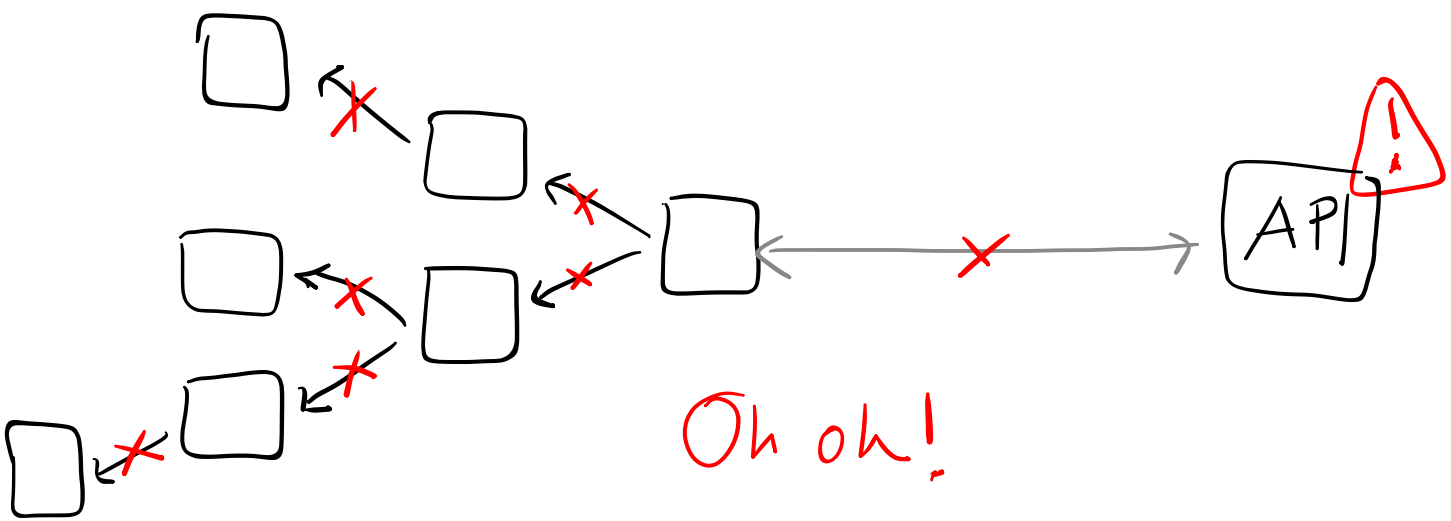


A reliable app performs the functionality the user expects.

Not all applications are built with reliability in mind

SINGLE POINT OF FAILURE (SPOF)

A component that if it fails or stops working the entire system is affected.



```javascript
const apis = [
  'https://eloux.com/todos/1',
];

async function fetchData(API) {
  const response = await fetch(API);
  if (response.ok) {
    return response.json();
  } else {
    return Promise.reject(new Error('Request failed'));
  }
}

function getData() {
  return Promise.any([
    fetchData(apis[0])
  ])
  .catch(() => {
    return Promise.reject(
      new Error('Unable to access the API');
    );
  });
}

getData()
  .then((data) => {
    console.log(data.title);
  });
```

SINGLE POINT OF FAILURE (SPOF)

Because we're using only one endpoint we have a component that creates a Single Point of Failure (SPoF). This is risky, especially if your app performs some critical functionality. (e.g., health, finance).

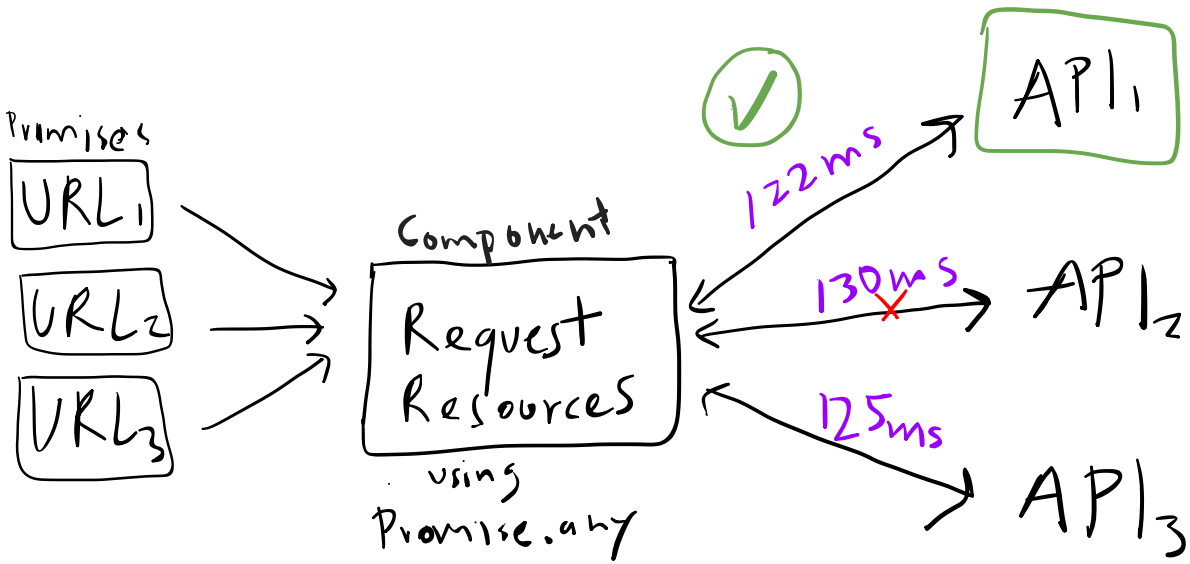
Attempts to fetch some resources from the internet. If the response is successful (200 OK), then it will return a fulfilled promise with a javascript object. Otherwise, it returns a rejected promise with an error object.

Uses the **Promise.any** method to return a single promise with the resolved value of the first fulfilled promise. In this case we only have one promise.

Chain a **.then** method that accepts a callback that logs a todo title to the console

RELIABILITY THROUGH REDUNDANCY

Promise.any can help us enhance the reliability of our apps



RELIABILITY THROUGH REDUNDANCY

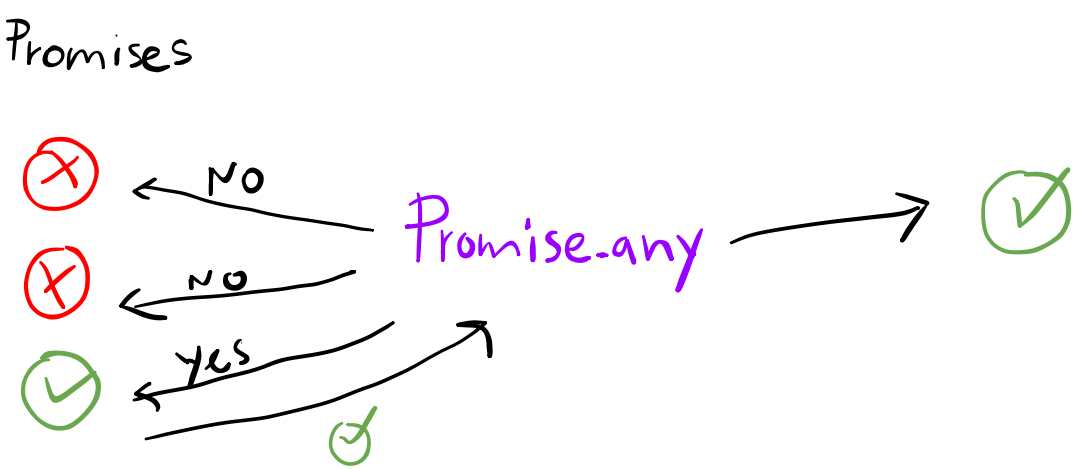
```javascript
const apis = [
  'https://eloux.com/todos/1',
  'https://jsonplaceholder.typicode.com/todos/1',
  'https://yetanotherapi.com/todos/1'
];

async function fetchData(API) {
  const response = await fetch(API);
  if (response.ok) {
    return response.json();
  } else {
    return Promise.reject(new Error('Request failed'));
  }
}

function getData() {
  return Promise.any([
    fetchData(apis[0]), fetchData(apis[1]),
    fetchData(apis[2])
  ])
  .catch(() => {
    return Promise.reject(
      new Error('Unable to access the API');
    );
  });
}

getData()
  .then((data) => {
    console.log(data.title);
  });
```

We can enhance the reliability of our application by using multiple API endpoints instead of only one.

Promise.any will skip over any rejected promises and return the very first **fulfilled promise**.

KEY TAKEAWAY

Promise.any allows us to create more reliable applications because it will ignore any rejected promises, and will wait until it gets a fulfilled promise.



...Questions?

@sergiojpichardo

THANK YOU!

Next, **Joey** will talk to you about another important topic, **performance**.

@sergiojpichardo

PERFORMANCE

Improving application performance with Promise.any

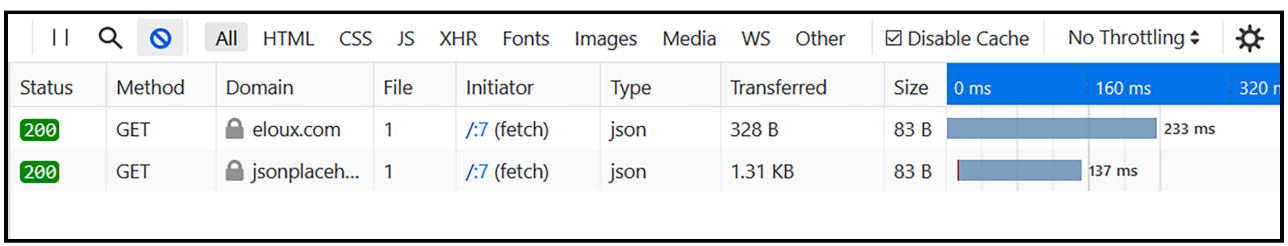
Joey Guillaume

OUTLINE

- Improving Response Times
- Participation Time
- Scenario
- Other Performance Considerations
- React App Demo: funnyFish

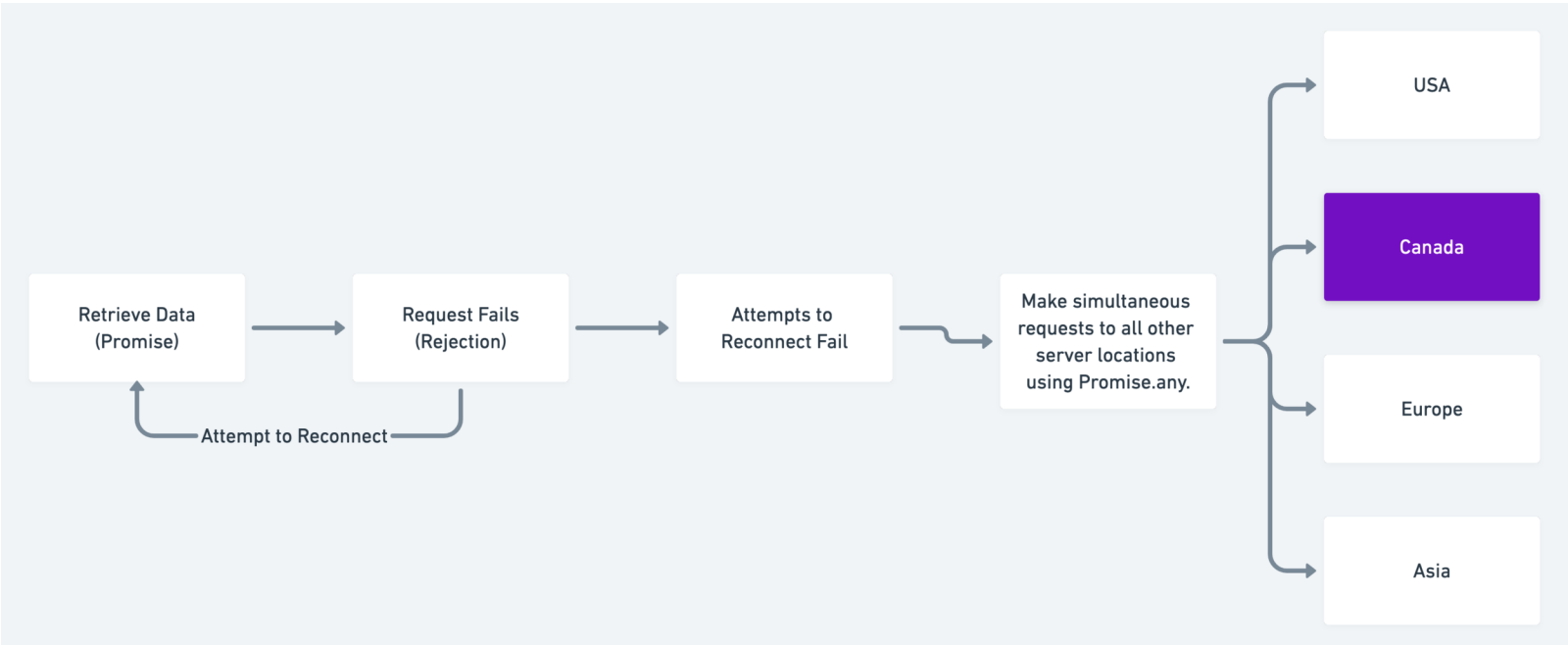
IMPROVING RESPONSE TIMES

By using ***Promise.any***, we can focus on retrieving our data as fast as possible. As a result, we can pass those savings onto the user by improving our response times!



Can you think of a use-case where ***Promise.any*** would be appropriate?

POSSIBLE SCENARIO



Using ***Promise.any*** here would allow us to recover from service degradation as fast as possible!

OTHER PERFORMANCE CONSIDERATIONS

```
1 async function timeTest() {
2   await timeoutPromise(3000);
3   await timeoutPromise(3000);
4   await timeoutPromise(3000);
5 }
```

VS

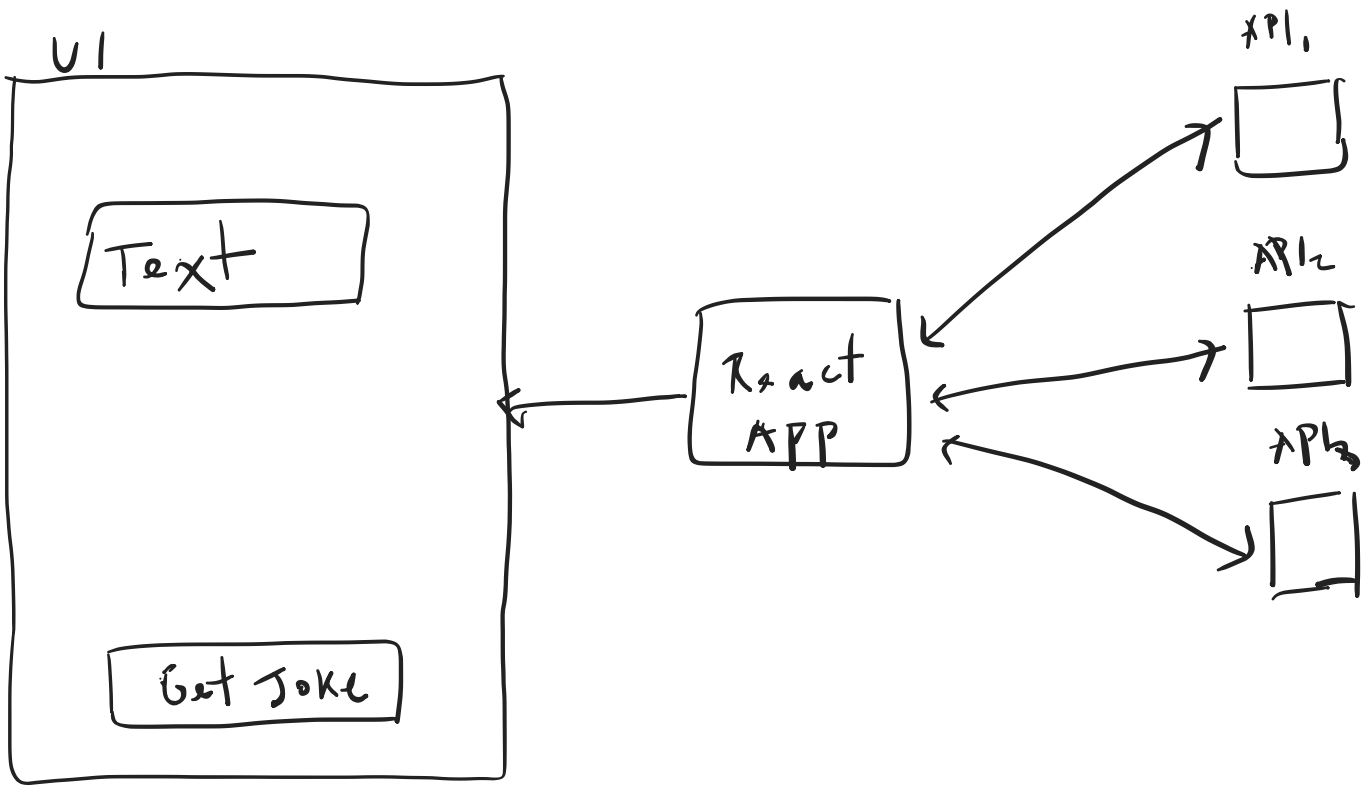
```
1 async function timeTest() {
2   const timeoutPromise1 = timeoutPromise(3000);
3   const timeoutPromise2 = timeoutPromise(3000);
4   const timeoutPromise3 = timeoutPromise(3000);
5
6   await timeoutPromise1;
7   await timeoutPromise2;
8   await timeoutPromise3;
9 }
```

OTHER PERFORMANCE CONSIDERATIONS

```
1 const catFacts = fetch('https://catfact.ninja/fact');
2
3 const ronSwansonQuotes = fetch('https://ron-swanson-quotes.herokuapp.com/v2/quotes');
4
5 const response = await Promise.any([
6   Promise.reject(),
7   Promise.reject(),
8   ronSwansonQuotes,
9   catFacts]);
```

APP DEMO

FUNNY FISH



Improving Response Times
Participation Time
Possible Scenario
Other Performance Considerations
React App Demo: funnyFish

Joey Guillaume

THANK YOU!

That's all Folks!

Sergio Pichardo

Lisa Melo

Joey Guillaume