AJAX with Fetch

AJAX

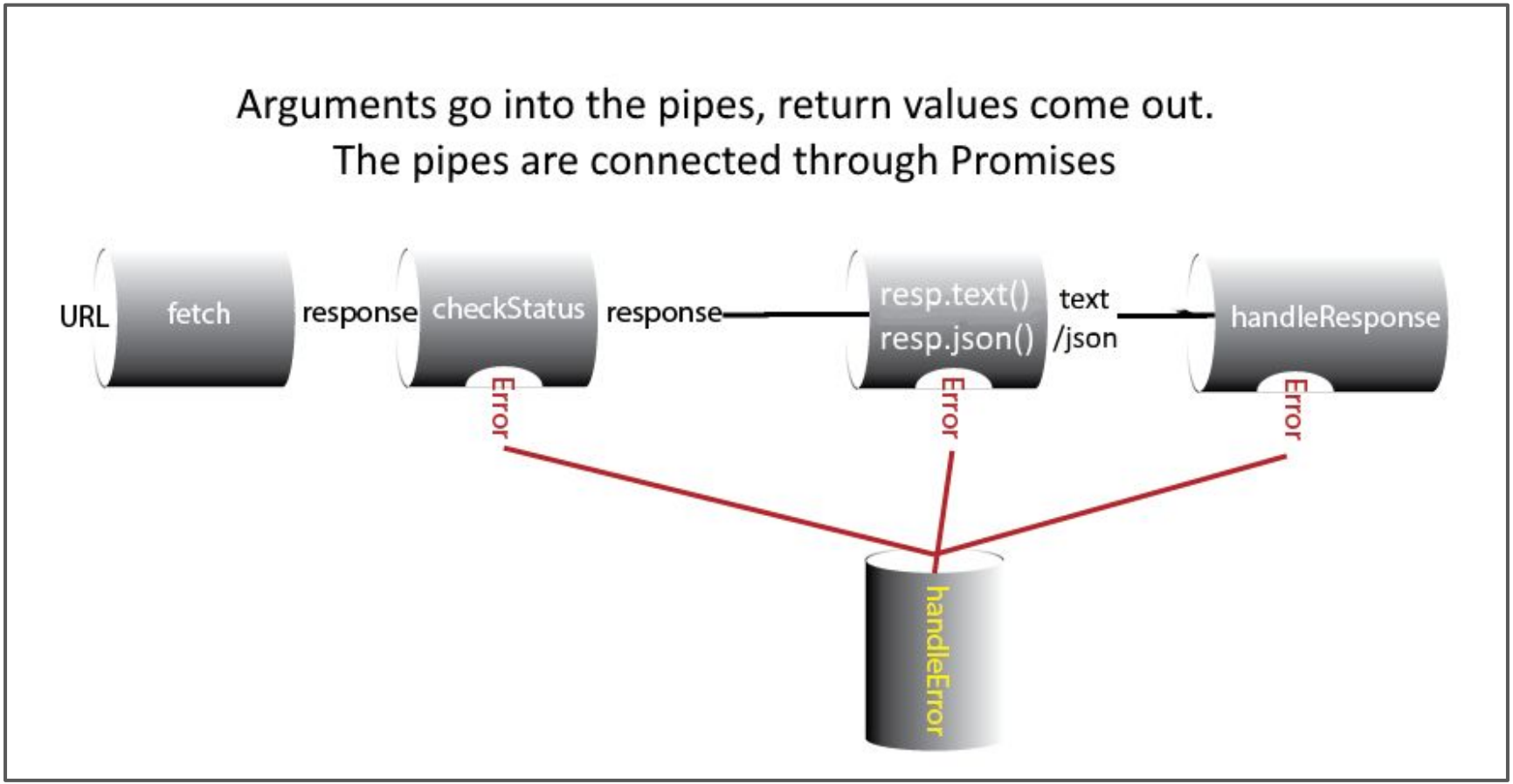
Why is it useful?

- The web is full of data - often, websites "ask" for data from servers which hold different types of data (txt, json, images, databases, etc.)
- What we know about JS so far does not give us any way to process data outside of our JS program. That's where AJAX comes in!

How do we use it?

- `fetch` (a built-in JavaScript function)
- A touch of Promises to elegantly control success (200) vs. error (non-200) responses from a server

The Promise Pipeline



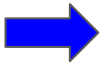
Mechanics

We initiate a `fetch` of a URL

- A `fetch` call returns a `Promise` object
- The `.then` method on a `Promise` object returns a `Promise` object
- Our first `.then(checkStatus)` checks the status of the response to makes sure the server responded with an OK. The result of that first `.then` is another `Promise` object with the response as the value of the Promise.
- We `.then(resp => resp.json())` which also returns a `Promise` object with a JSON object as the value
- We `.then(processData)` which will do something with the response from the server.
- If at any time there is an error, the execution falls down to the `.catch` method on the `Promise` chain

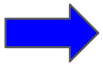
Chaining of Promises gives us a nice data flow, like down a pipe!

Making a request and then chaining with .then/.catch



```
function makeRequest() {
  fetch(BASE_URL + "?query=params")
    .then(checkStatus)
    .then(res => res.json())
    //.then(res => res.text())
    .then(processData)
    .catch(handleError);
}
```

Making a request and then using async/await



```
async function makeRequest() {
  try {
    let res = await fetch(BASE_URL +
"?query=params");
    await checkStatus(res);
    res = await res.json();
    //res = await res.text();
    processData(res);
  } catch(err) {
    handleError(err);
  }
```

Exercise 1: Ajax Pets

Given these starter files, create an AJAX-powered gallery of pet images that allows you to switch between kitty and puppy images without reloading the page. You can view the finished product here.

Exercise 1: Ajax Pets API URL

Service URL:
https://courses.cs.washington.edu/courses/cse154/webservices/pets/ajaxpets.php

Query Parameters (required):

?animal=<value>

Details: animal is the name of the query parameter you need to assign a value to. This API recognizes either a value of puppy or kitty.

Example Request (with puppy as the value):

Exercise 1: Ajax Pets API Response Format

Response Format: Plain Text

```
https://path/to/pet/img0.jpg
https://path/to/pet/img1.jpg
https://path/to/pet/img2.jpg
https://path/to/pet/img3.jpg
...
```

Exercise 1: Ajax Pets Implementation

The provided starter code includes a module-pattern template we've been using to get you started, named `ajaxpets.js`. You will need to implement the JavaScript to incorporate AJAX and make a request with the Ajax Pets API URL with the parameter `animal` of value `kitty` or `puppy`, depending on which radio button is selected.

When a request returns a response successfully with the plain text response of image paths, write JS to add `img` tags as children to the `#pictures` div for each image path returned on a new line.

Hint: you should listen for the `change` event to know when to be making the corresponding fetch request

Solution

https://courses.cs.washington.edu/courses/cse154/19sp/sections/week05-tues/code/solution/ajaxpets/ajaxpets.js