Lecture 5

Fetch & APIs

Today’s Contents

- APIs with examples
- Fetch
    - JSON & text
    - GET & POST
- Regular Expressions
- HTML Validation

Fetch & APIs

- JS `fetch`
- `fetch` & `Promise`
- Client-server architecture
- APIs

`fetch` & APIs

Allows us to use JavaScript to request resources from servers connected to the internet

```
    fetch("http://www.weather.com")
```

Can think of it as replicating the behavior of our browser address bar

`fetch` returns a Promise!

Client and Server

Analogy: You're out to get some pizza, so you:

- Request menu
- Order pizza
- Check pizza
- Eat pizza
- Pay for pizza

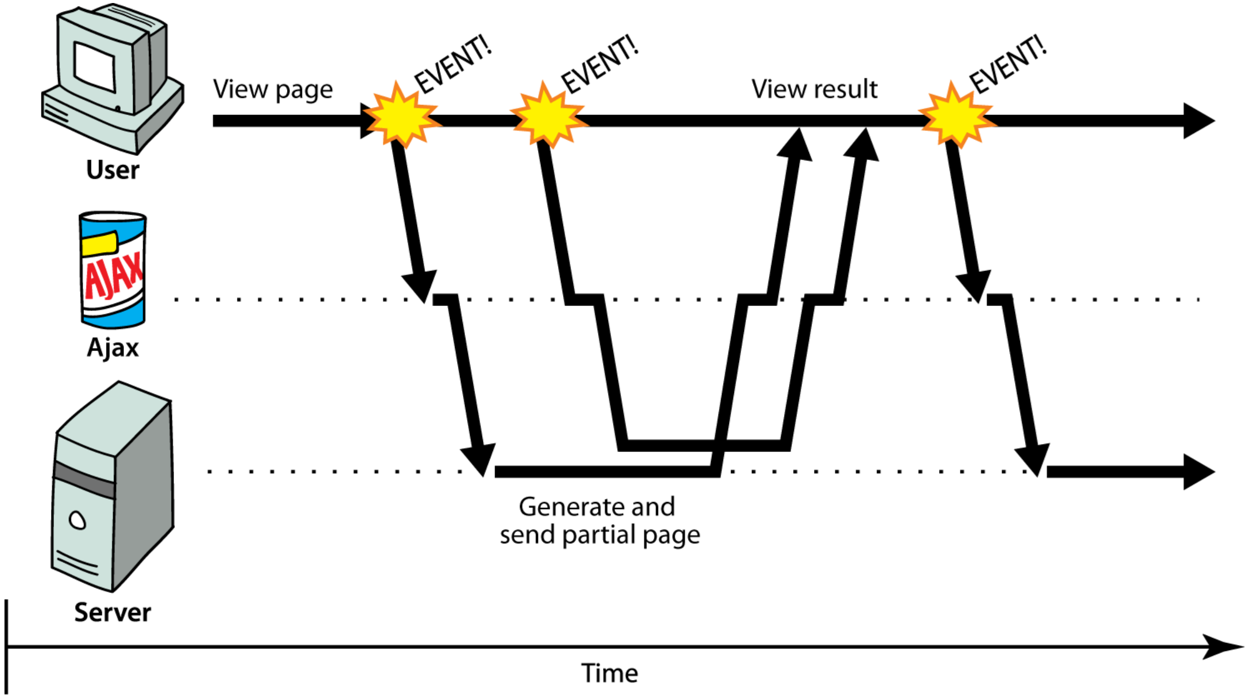
At each step, can’t continue before the previous finishes. We as the client (pizza eaters) need the server (pizza provider) to do something on our behalf.

Fetching Data Asynchronously

But we have to be careful because the Internet is an “unusual” place and full of surprises!

- What kind of data can we get from the Internet?
- How do we request data from the Internet?
- What happens when something goes wrong on the Internet?
- Can you trust everything you find on the Internet?

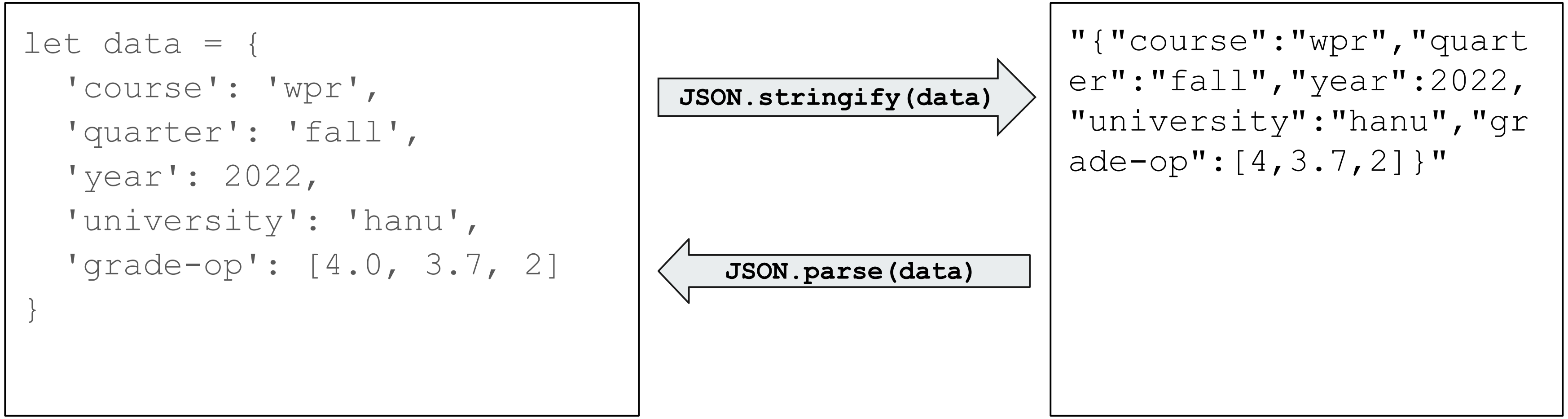
Asynchronous JavaScript and (XML)* -- "AJAX"



JavaScript calls where we just don't know when it'll come back, but we're sure it'll be a request sent to a server with some data sent back to us (the client).

*Nobody really uses XML anymore.

Recall: JSON



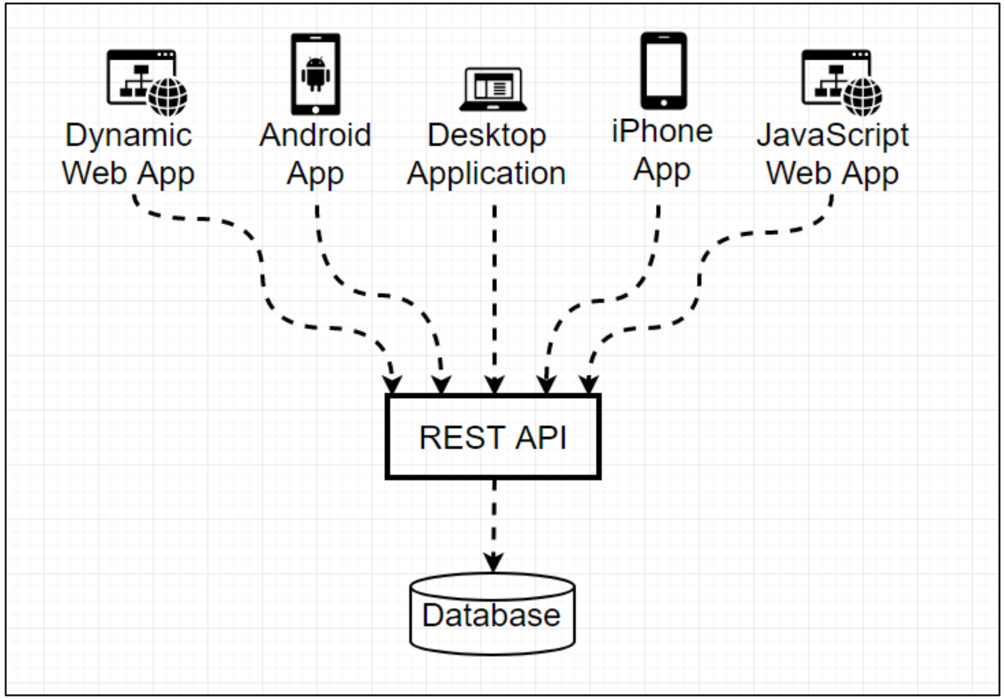
```
let data = {
  'course': 'wpr',
  'quarter': 'fall',
  'year': 2022,
  'university': 'hanu',
  'grade-op': [4.0, 3.7, 2]
}
```

```
"{"course":"wpr","quart
er":"fall","year":2022,
"university":"hanu","gr
ade-op":[4,3.7,2]}"
```

Application Program Interface (APIs)

An application programming interface (API) is a communication protocol that allows two pieces of software to communicate.

A *Web* API is a set of predefined URLs with parameters that allow a user to get information from a web server. Can be used by many types of clients.



API: Application Programming Interface

APIs come in all shapes and sizes:

- REST
- SOAP
- RPC/gRPC
- JSON over HTTP
- GraphQL
- … And many others

Most of what we'll see is based on or built on top of HTTP.

Promises (and async/await) + API calls

When calling an API, we have no idea when it’ll return.

- Where’s the server located?
- Where is it located in relation to us?
- Are there any traffic jams on the Internet?
- Is the server running slow today?
- Did the server change something about what it expects in requests?

AJAX with `fetch`

```
(function(){
    ...
    function doWebRequest() {
        const url = "....."; // put url string here
        fetch(url); // returns a Promise!
    }
    ...
})();
```

AJAX with `fetch`

`fetch` was created in 2014 and incorporated into the global `window`.
`fetch` takes a URL string as a parameter to request data (e.g. menu category JSON) that we can work with in our JS file.

```
function populateMenu() {
  const URL = "https://hanustartup.org/wpr/api";
  fetch(URL + "/getCategories.php") // returns a Promise!
    .then(...)
    .then(...)
    .catch(...);
}
```

We need to do something with the data that comes back from the server.
But we don't know how long that will take or if it even will come back correctly!
The `fetch` call returns a `Promise` object which will help us with this uncertainty.

AJAX `fetch` Code Skeleton

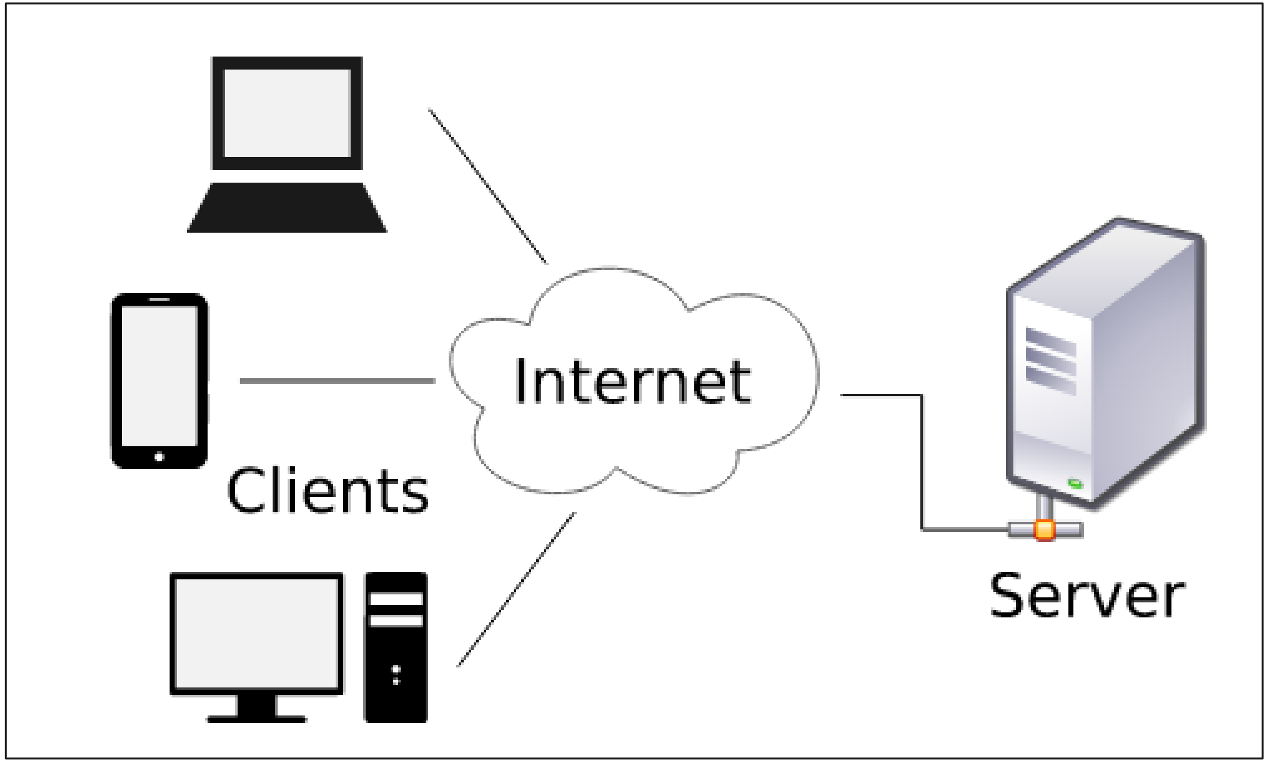
- The following is much easier to read, and factors out larger callback functions into named functions. Remember that the return of each callback function passed to a then is exactly the argument passed into the callback of the next then.

- For short callbacks, we also see the motivation with arrow function syntax!

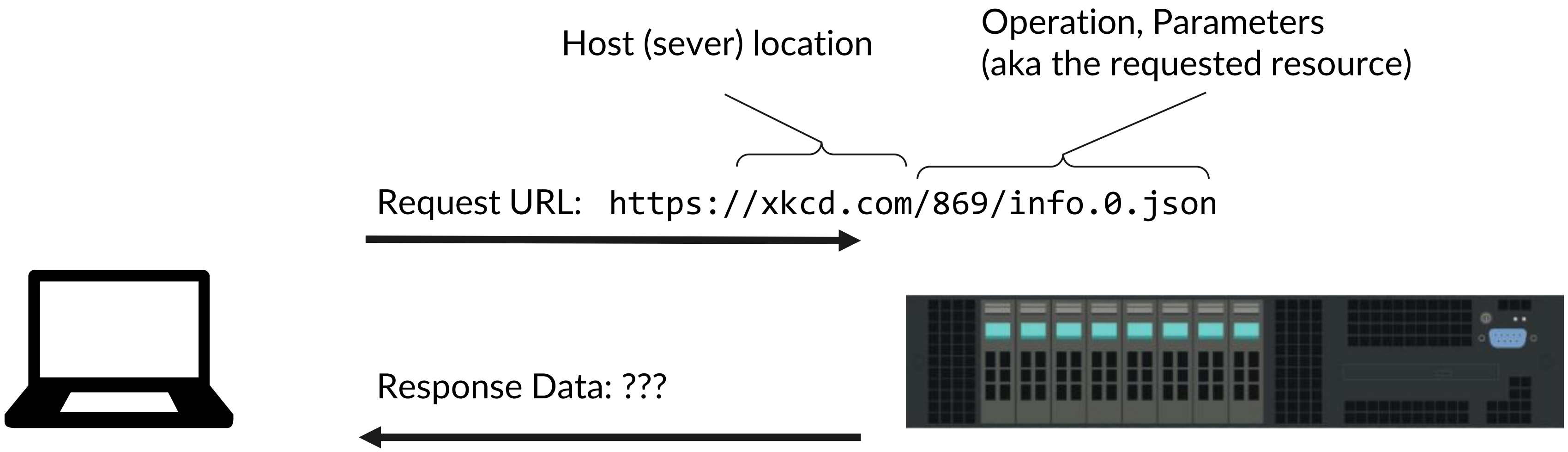
```
function fetchTheData() {
  fetch(url + possibleParameters)
    .then(statusCheck)
    .then(response => response.json())    // if JSON
    // .then(response => response.text()) // if text
    .then(processResponse)
    .catch(handleError);
  }
}

function processResponse(data) { ... }
function handleError(err) { ... }
function statusCheck(response) { ... }
```

The Client-Server Model



The Request/Response Transaction with Web APIs


Host (sever) location
Operation, Parameters
(aka the requested resource)
Request URL: `https://xkcd.com/869/info.0.json`
Response Data: ???

Various formats: JSON, XML, Protobuf, HTML, or anything that can be serialized to plain text

Generic `fetch` template

- When we make a call to fetch data (whether it's HTML, txt, JSON, etc) we use a function in JavaScript called `fetch`, which takes a URL path to fetch data from.

- Here's the general template you will use for most AJAX code:

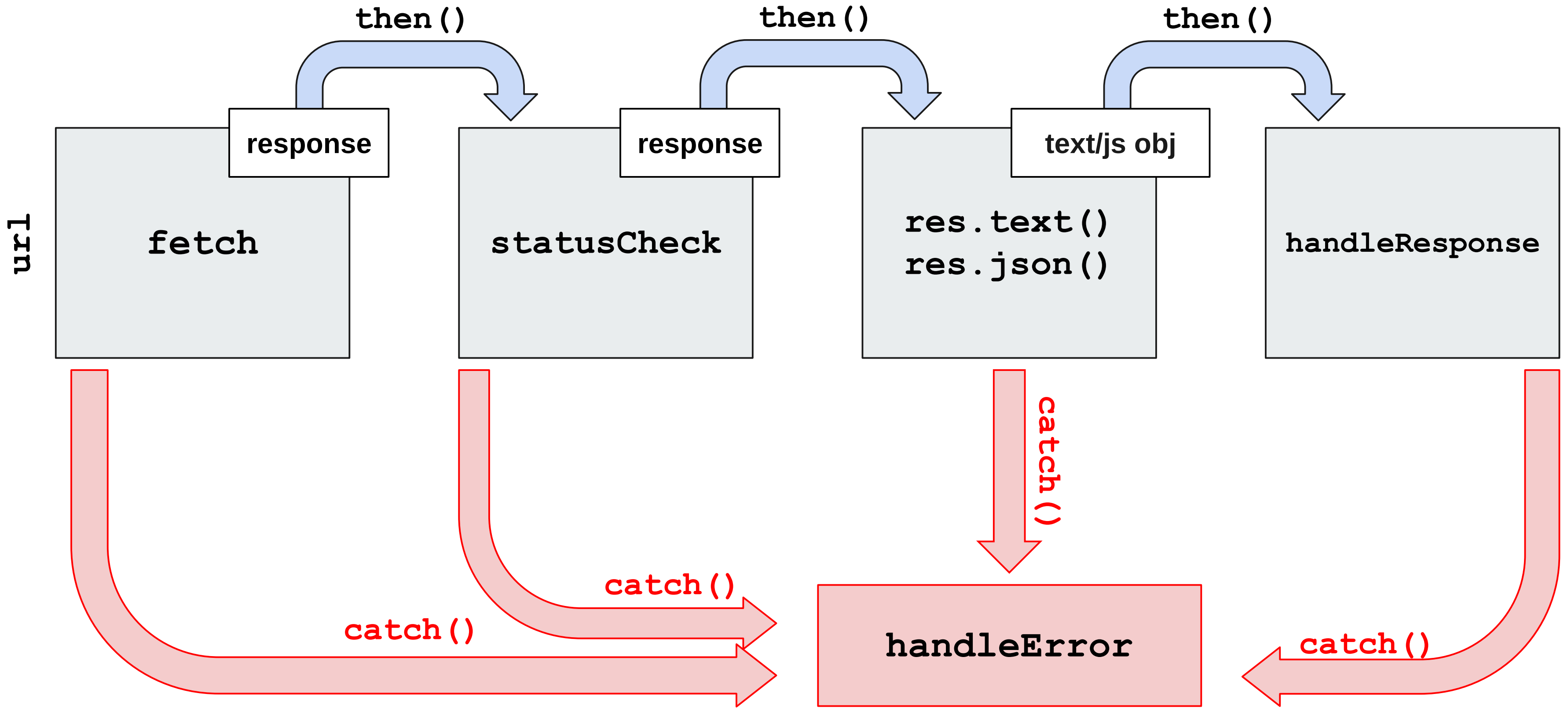
```
const BASE_URL = "some_base_url.com"; // you may have more than one

// Step 1. Write a function to "fetch" data from a URL
function makeRequest() {
  let url = BASE_URL + "/api?query0=value0"; // some requests require parameters
  fetch(url)
    .then(statusCheck)
    //.then(resp => resp.text()) // use this if your data comes in text
    //.then(resp => resp.json()) // or this if your data comes in JSON
    .then(processData)
    .catch(handleError); // define a user-friendly error-message function
}

// Step 2: Implement a function to process the data. You may want to break this apart
// into multiple functions.
function processData(responseData) {
  // Do something with your responseData! (build DOM, display messages, etc.)
}

// Step 3. Include the statusCheck function
async function statusCheck(res) {
  if (!res.ok) {
    throw new Error(await res.text());
  }
  return res;
}
```

The Promise Pipeline



Every API has four components:

1. Where is the server that's running the API? (I.e., what's the hostname?)
2. Are we allowed to access it?
3. What content does the API give us?
4. What format should the request for content be in? What format does the response come back in?

Fetching data from a web service

To make a request to a web service (API), we need to know:

1. Who to ask (the url)
    -
2. Do we have permission to ask (any API keys/tokens)
3. What to ask (any query parameters)
    -
    -
4. What format we get answers in (the response data format, preferably text or JSON)

Fetching data from a web service

To make a request to a web service (API), we need to know:

1. Who to ask (the url)
    - Example: https://elephant-api.herokuapp.com/elephants/random
    - Example: http://numbersapi.com
2. Do we have permission to ask (any API keys/tokens)
    - Not for elephants, but NASA/TMDB require them; add as additional query parameter
3. What to ask (any query parameters)
    - Differ per API, some have specific order of parameter values, separated by /, others have explicit key/value pairs started with ? and combined with &
    - Example: A Recipes API might have `recipe.php?recipe=cupcake&vegan=true`
    - Example: https://quandd.42web.io/wpr/api/cafe/menu.php?menu=cafe-menu (also supports menu=pizza-menu)
    - Example: Numbers API: http://numbersapi.com/154/math
4. What format we get answers in (the response data format, preferably text or JSON)

Example API Requests

A few examples of breaking down request URLs for various APIs.

- NASA APOD
- Datamuse
- XKCD

API1: NASA APOD (Astronomy Photo of the Day) API

1. What kind of format is the response data?
2. Do you need an API key?
3. What is the base url?

API1: NASA APOD API Solution

1. What kind of format is the response data? **JSON**
2. Do you need an API key? **Yes** (but you can make 50 calls per day with "DEMO_KEY" as your api key parameter!)
3. What is the base url? **https://api.nasa.gov/planetary/apod**

API2: Datamuse API

An word-querying API: https://www.datamuse.com/api/

1. What kind of format is the response data?
2. Do you need an API key?
3. What is the base url?

API2: Datamuse API Solution

An word-querying API: https://www.datamuse.com/api/

1. What kind of format is the response data? **JSON**
2. Do you need an API key? **No**
3. What is the base url? **https://www.api.datamuse.com**

API3: XKCD

"Documentation": https://xkcd.com/json.html

1. What kind of format is the response data?
2. Do you need an API key?
3. What is the base url?

API3: XKCD Solution

"Documentation": https://xkcd.com/json.html

1. What kind of format is the response data? **JSON**
2. Do you need an API key? **No**
3. What is the base url? **https://xkcd.com/**

A closer look at `statusCheck()`

```
async function statusCheck(res) {
  if (!res.ok) {
    throw new Error(await res.text());
  }
  return res;
}
```

Why do we place a `.catch()` at the end of our fetch chain rather than directly after the `fetch()`?

Why don’t we place it after `statusCheck()`?

An answer can be found here.

Check your Understanding

Consider the following request we might make with `fetch`:

```
function makeRequest() {
  let requestString = "https://api.nasa.gov/planetary/apod?" +
                      "api_key=DEMO_KEY&date=2022-02-22";
  fetch(requestString)
    .then(statusCheck)                  // 1
    .then(resp => resp.json())          // 2
    .then(processData)                  // 3
    .catch(handleError);                // 4
}
```

1. Who are we asking for data from (the base url)?
2. What query parameters are we using? What are the values of the parameters?
3. What is the purpose of each step (1 through 4)?

A note on CORS

- CORS: Cross-Origin Resource Sharing (CORS resource - MDN)
- Security measure browsers have implemented that restricts one site from accessing resources in another *unless explicitly given permission*.
- Most of the APIs that are demoed in this lecture have explicitly allowed *, meaning anyone.
- Many APIs you run into around the Internet will be locked more tightly.

Regular Expressions!

```
/^[a-zA-Z_\-]+@(([a-zA-Z_\-])+\.)+[a-zA-Z]{2,4}$/
```

Regular expression ("regex"): a description of a pattern of text

- Can test whether a string matches the expression's pattern
- Can use a regex to search/replace characters in a string

Regular expressions are extremely powerful but tough to read

(the above regular expression matches... what?)

Regular expressions occur in many places:

- Java: Scanner, String's `split` method
- Supported by HTML5, JS, Java, Python, PHP, and other languages
- Many text editors (Notepad++, Sublime, etc.) allow regexes in search/replace
- The site RegEx101 is useful for testing a regex

Basic Regular Expression

```
/abc/
```

In JS, regexes are strings that begin and end with `/`

The simplest regexes simply match a particular substring

The above regular expression matches any string containing `"abc"`

- Matches: `"abc"`, `"abcdef"`, `"defabc"`, `".=.abc.=."`, ...
- Doesn't match: `"fedcba"`, `"ab c"`, `"PHP"`, ...

Wildcards, Case sensitivity

A `.` matches any character except a `\n` line break

- `/.ax../` matches "Faxes", "Jaxes", "Taxes", "maxie", etc.

A trailing `i` at the end of a regex (after the closing `/`) signifies a case-insensitive match

- `/cal/i` matches "Pascal", "California", "GCal", etc.

Quantifiers: `*`, `+`, `?`

`*` means 0 or more occurrences

- `/abc*/` matches "ab", "abc", "abcc", "abccc", ...
- `/a(bc)*/` matches "a", "abc", "abcbc", "abcbcbc", ...
- `/a.*a/` matches "aa", "aba", "a8qa", "a!?xyz__9a", ...

`+` means 1 or more occurrences

- `/Hi!+ there/` matches "Hi! there", "Hi!!! there!", ...
- `/a(bc)+/` matches "abc", "abcbc", "abcbcbc", ...

`?` means 0 or 1 occurrences

- `/a(bc)?/` matches only "a" or "abc"

Character ranges: `[start-end]`

Inside a character set, specify a range of characters with -

- `/[a-z]/` matches any lowercase letter
- `/[a-zA-Z0-9]/` matches any lowercase or uppercase letter or digit

Inside a character set, - must be escaped to be matched

- `/[+\-]?[0-9]+/` matches an optional + or –, followed by at least one digit

Practice exercise

Write a regex for Student ID numbers that are exactly 7 digits and start with 1

Practice exercise

Write a regex for Student ID numbers that are exactly 7 digits and start with 1

Solution: `/1\d{6}/`

Regular Expressions in JavaScript

Regex can be a very handy tool with JS as well, from validation to fun find/replace features. There are two common ways regex can be used:

- With the RegExp constructor (either with a literal or string, useful because you don't always know what the pattern is, such as user input)
- With a literal (when you know exactly what the regex pattern is, evaluated exactly once)

```
let pattern1 = new RegExp(/fit3wpr/, "i");
let pattern2 = new RegExp("fit3wpr", "i");
let pattern3 = /fit3wpr/;
```

(*) Note that we don't use `"/"` when using strings for patterns in the second RegExp constructor. This can be useful when we want to search for a particular pattern given as text input (e.g. a word replacer tool)!

Some Regex Functions in JS

Regex Objects have a few useful functions that take strings as arguments

`regex.test(string)` returns a boolean if a string matches the regex

```
let namePattern = /[A-Z][a-z]+ [A-Z][a-z]+/;
namePattern.test("Andrew Fitz Gibbon"); // false

let studentIdPattern = new RegExp("1\d{6}");
studentIdPattern.test("-123");          // false
```

Some String Functions in JS

Some JS string methods can take Regular Expressions, like `match`, `search`, and `replace`
`string.match(regex)` returns an array of information about a match, including the index of the first match

```
"Hello world".match(/wo.l/); // [0: "worl", index: 6, input: "Hello world"]
```

`origStr.replace(regex, replStr)` returns a new string replacing a `pattern` match in `origStr` with the string `replStr`

```
let newStr = "My cats are good cats".replace(/cat/, "kitten");
// newStr === "My kittens are good cats"
```

Question: Where might you need to be careful with a RegEx replace?

```
let newStr = "There is a category of cats that catch
cattle".replace(/cat/, "kitten");
newStr === "There is a kittenegory of cats that catch cattle"
```

And what if we replaced **all** instances?

```
let newStr = "There is a category of cats that catch
cattle".replace(/cat/g, "kitten");
newStr === "There is a kittenegory of kittens that kittench
kittentle"
```

HTTP Request Methods

- By default, `fetch` makes a GET request
- But there are other methods: GET, POST, PUT, DELETE, and others
- PUT and DELETE are less common, and we won't use them in this class.

GET vs. POST

GET

- Requesting data from a server using filters (URL parameters)
- Should never be used when dealing with sensitive data
- Can be cached and bookmarked, remains in browser history

POST

- Data sent in the HTTP message body, not the URL
- Not cached or bookmarked, no restrictions on data length
- POST is commonly used with forms! However, not exclusively with forms

Learn more about GET & POST Methods here.

Modifying `fetch` for POST requests

```
function requestMenu() {
  let data = new FormData();
  data.append("key1", "value1"); // repeat for as many params as necessary
  data.append("key2", "value2");

  fetch(API_URL, {method: "POST", body: data})
    .then(statusCheck)
    .then(resp => resp.json()) // or res.text() based on response
    .then(handleResponse)
    .catch(handleError);
}
```

Posting data with Forms

There are two ways you'll commonly see forms used with POST

- **Older**: With method/action attributes in HTML form tag
- **Better**: With JS using validation and AJAX

Posting with method/action attributes in `<form>` tag

- As soon as the submit button is clicked on this page, the data is sent to the web service and the page is refreshed with the response (sometimes redirecting).
- This is becoming less common because we lose the asynchronous features of requests.

```
<form id="input-form" method="post" action="url-path-to-api">
    City: <input name="city" type="text">
    State: <input name="state" type="text">
    ZIP: <input name="zip" type="number">
    <button id="submit-btn">Submit!</button>
</form>
```

Approach 2: Posting data through JS/AJAX

```html
<form id="input-form">
    City: <input name="city" type="text">
    State: <input name="state" type="text">
    ZIP: <input name="zip" type="number">
    <button id="submit-btn">Submit!</button>
</form>
```

```javascript
let url = "url-path-to-api";
let params = new FormData(id("input-form")); // pass in entire form tag
fetch(url, { method : "POST", body : params })
  .then(statusCheck)
  .then(...)
  .catch(...);
```

Approach 2: Posting data through JS/AJAX

When an input is in a form along with a button, clicking the button automatically verifies the input and does a POST request. If you do not want the page to refresh, you must use preventDefault to override default form submission behavior (used in previous example).

```
function someFunction() {
  id("input-form").addEventListener("submit", function(e) {
    // Fires when submit event happens on form
    // If we've gotten in here, all HTML5 validation checks have passed
    e.preventDefault(); // prevent default behavior of submit (page refresh)
    submitRequest(); // intended response function
  });

  // rest of your code
}
```

Validation with HTML/CSS/JS

Many websites offer features that allow users to interact with the page and request/submit data to servers. Unfortunately, not all users will behave as expected.

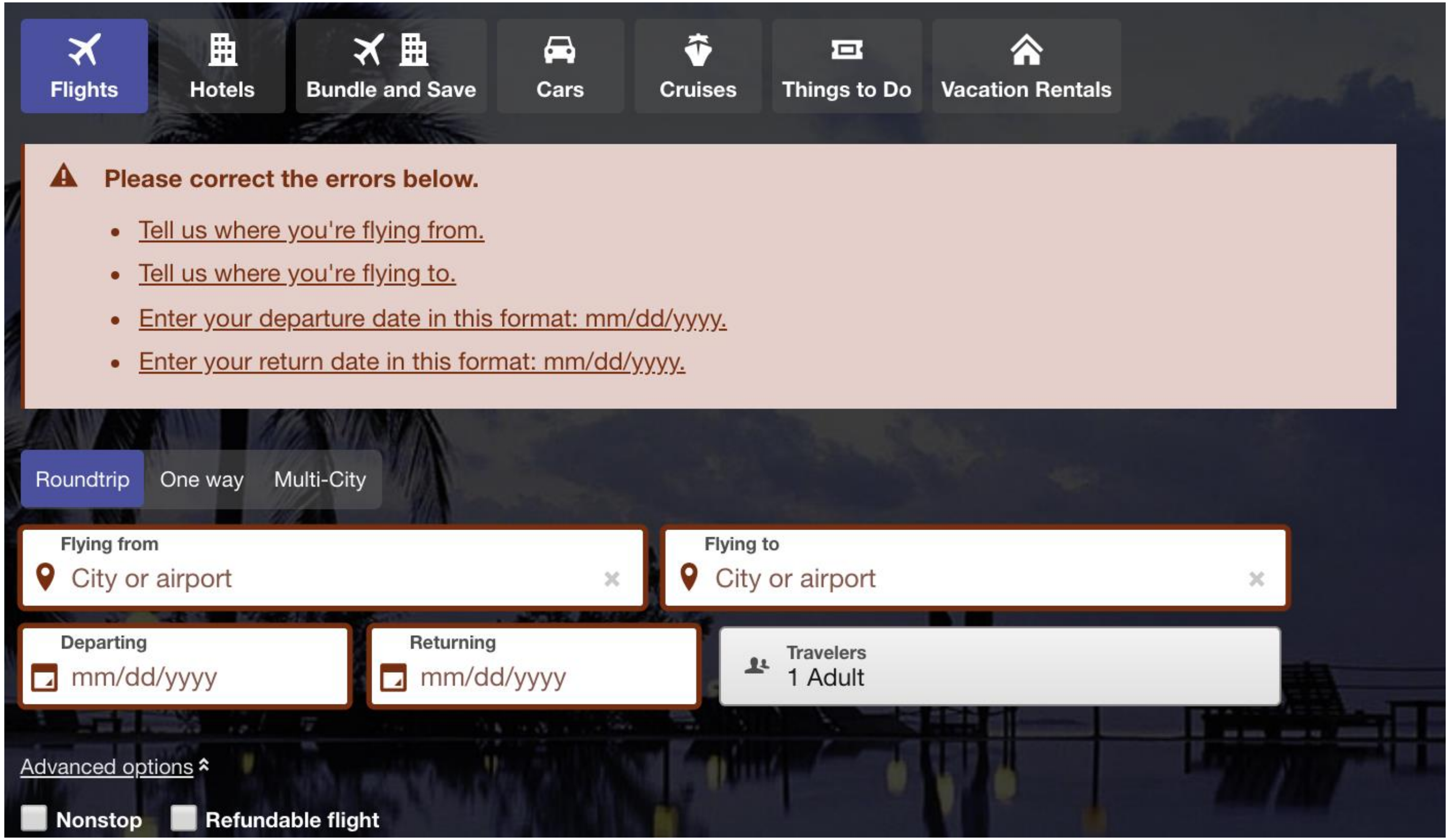
User Input Validation

User input validation is the process of ensuring that any user input is well-formed and correct (valid).

What are some examples of input validation you came up with?

- Preventing blank values (e-mail address)
- Verifying the type of values (e.g. integer, real number, currency, phone number, Social Security Number, postal address, email address, data, credit card number, ...)
- Verifying the format and range of values (ZIP code must be a 5-digit integer)
- Extra layer of confirmation (e.g. user types email twice, and the two must match)

Real-World Example with Validation Feedback



Please correct the errors below.

- Tell us where you're flying from.
- Tell us where you're flying to.
- Enter your departure date in this format: mm/dd/yyyy.
- Enter your return date in this format: mm/dd/yyyy.

Flying from: City or airport

Flying to: City or airport

Departing: mm/dd/yyyy

Returning: mm/dd/yyyy

Travelers: 1 Adult

☐ Nonstop ☐ Refundable flight

Importance of Validation in Web Development

Prioritizing validation is important as web developers so that the websites we build are:

- User-friendly
- Secure(ish)
    - If you're interested in learning more, MDN has a good quick introduction to web security, and OWASP is a fantastic resource for all things related to web security.
    - You can also find a good article on how to write user-friendly form UIs here.

Most importantly, ***don't trust that users will provide correct/safe input!***

A Form Example

```
<form>
  <label for="city">City: </label>
  <input name="city" id="city" type="text">
  <label for="state">State: </label>
  <input name="state" id="state" type="text">
  <label for="zip">ZIP: </label>
  <input name="zip" id="zip" type="number">
  <button id="submit-btn">Submit!</button>
</form>
```

We can validate this input in a few different ways:

1. Client-side: HTML5 form elements and input tag attributes
2. Client-side: JS before sending this form data to the server
3. Server-side: (later)!

HTML5 Input Validation

We've already seen some ways to use HTML5 tags to require certain types of input by adding attributes to your `<input>` tags to help with validation

```
<input type="number">
```

We can limit the up and down arrows with `min` (and `max` if we choose)

```
<input type="number" min=0>
```

To insist that there is a value in the input field we can add `required`

```
<input type="number" required>
```

To prevent a user from being able to type in erroneous values, we can add a regular expression to the `required` attribute

```
<input type="number" required="\d+">
```

Basic HTML5 Validation with a `form`

Forms are HTML elements that can be used to "package" user input values based on the name attribute, often used with POST requests. There are many neat ways to perform validation with the right choice of form elements!

```
<form>
  <label for="city">City: </label>
  <input name="city" id="city" type="text" required>
  <label for="state">State: </label>
  <input name="state" id="state" type="text" size="2" maxlength="2" required>
  <label for="zip">ZIP: </label>
  <input name="zip" id="zip" type="number" size="5" min=10000 max=99999 required>
  <button id="submit-btn">Submit!</button>
</form>
```