AJAX
Brief History of Web Applications

• Early days (pre-1996): static HTML files only

• Common Gateway Interface (CGI)
  • Some URLs map to executable programs
  • Program exits after the web page is sent to client

• Introduced the notion of stateless servers: each request is independent

• Perl was commonly used for writing CGI programs (also Python and other scripting languages).
First-generation web app frameworks

- Examples: PHP, ASP.net, Java servlets
- Templates: mix code and HTML
- Web-specific library packages:
  - URL handling
  - HTML generation
  - Sessions
  - Interfacing to databases
Second-generation frameworks

• Examples: Ruby on Rails, Django, Java Spring

• Model-view-controller: decomposition of applications

• Object-relational mapping (ORM) simplifies database access
Third-generation frameworks

- Examples: AngularJS, ReactJS
- JavaScript frameworks running in browser
  - Much more of the application runs in the browser
  - Interactive, responsive applications
- Frameworks not as dependent on particular server-side features
  - Server primary stores and serves data
- Many concepts of previous generations still apply
Ajax

• **Asynchronous Javascript** and **XML**

• 1996, Microsoft used iframe tags to load data asynchronously
  • dynamically update news stories and stock quotes

• 1999 Microsoft created the **XMLHttpRequest** JavaScript object
• Google made extensive use of standards-compliant cross browser Ajax in Gmail (2004).

• XMLHttpRequest allowed JavaScript inside web pages to do something they could never really do before: get more data. (http://www.aaronsw.com/weblog/ajaxhistory)
Ajax

1. User interaction invokes an event handler.
2. The event handler's code creates an XMLHttpRequest object.
3. The XMLHttpRequest object requests a page from the server.
4. The server retrieves appropriate data based on the page requested and sends it back.
5. The XMLHttpRequest object fires an event (a callback) when the data arrives, often a function.
6. The callback event handler processes the data and updates the DOM accordingly.
JSON

• JavaScript Object Notation is a language-independent convention for formatting data as a set of JS objects.

• Made up of a collection of name/value pairs (object) that can also include an ordered list of values (array).
Example

{
  "private": "true",
  "from": "Alice Smith (alice@example.com)",
  "to": [
    "Robert Jones (roberto@example.com)",
    "Charles Dodd (cdodd@example.com)"
  ],
  "subject": "Tomorrow's "Birthday Bash" event!",
  "message": {
    "language": "english",
    "text": "See you at my place!"
  }
}
Ajax and JSON

• If you’re using Ajax to access a page that serves JSON-formatted data, you can use these JavaScript methods to parse and convert.

`JSON.parse(string)`

• Converts the given string of JSON data into an equivalent JavaScript object and returns it.

`JSON.stringify(object)`

• Converts the given object into a string of JSON data (the opposite of `JSON.parse`).
$.ajax({
    url: "/api/getWeather",
    data: {
        zipcode: 97201
    },
    success: function( data ) {
        $( "#weathertemp" ).html( "<strong>" + data + "</strong> degrees" );
    }
});
• You should tell jQuery AJAX what dataType you’re expecting: text, html, xml, json ...

• Same-origin policy: Unless it’s ajsonp type (from another domain), must come from same protocol (http or https), same port, and same domain as request page.
Fetch API

- Replaces XMLHttpRequest
- Uses promises

- Example:
  ```javascript
  fetch('/names')
  .then(status)
  .then(jjjkkkjjson)
  .then(function(data) {
    console.log('Request succeeded with JSON response', data);
    addList(data);
  }).catch(function(error) {
    console.log('Request failed', error);
  });
  ```