Front End Programming

CPEN320

From Static HTML Files…

- Initially: static HTML files only with HTML forms for input
- Common Gateway Interface (CGI)
    - Certain URLs map to executable programs that generate web page
    - Program exits after Web page complete
    - Introduced the notion of stateless servers: each request independent, no state carried over from previous requests. (Made scale-out architectures easier)
    - Perl typically used for writing CGI programs

First-generation web app frameworks

Examples: (PHP, ASP.net, Java servlets)

- Incorporate language runtime system directly into Web server
- **Templates**: mix code and HTML - HTML/CSS describes view
- Web-specific library packages:
  - URL handling
  - HTML generation
  - Sessions
  - Interfacing to databases

Second-generation frameworks

Examples: (Ruby on Rails, Django):

- **Model-View-Controller (MVC)**: stylized decomposition of applications
- Model and controller: server, View: client browser
- Object-relational mapping (**ORM**): simplify the use of databases (make database tables and rows appear as classes and objects)
    - Easier fetching of dynamic data

Third-generation frameworks

Example: AngularJS

- JavaScript frameworks running in browser - More app-like web apps
  - Interactive, quick responding applications - Don't need server round-trip
- Frameworks not dependent on particular server-side capabilities
  - Node.js - Server side JavaScript
  - No-SQL database (e.g. MongoDB)
- Many of the concepts of previous generations carry forward
  - Model-view-controller
  - Templates - HTML/CSS view description

Model-View-Controller (MVC) Pattern

- **Model**: manages the application's data
    - JavaScript objects. Photo App: User names, pictures, comments, etc.
- **View**: what the web page looks like
    - HTML/CSS. Photo App: View Users, View photo with comments
- **Controller**: fetch models and control view, handle user interactions
    - JavaScript code. Photo App: DOM event handlers, web server communication

MVC pattern been around since the late 1970's

- Originally conceived in the Smalltalk project at Xerox PARC

View Generation

- Web App: Ultimately need to generate HTML and CSS
- **Templates** are commonly used technique. Basic ideas:
  - Write HTML document containing parts of the page that are always the same.
  - Add bits of code that generate the parts that are computed for each page.
  - The template is expanded by executing code snippets, substituting the results into the document.
- Benefits of templates (Compare with direct JavaScript to DOM programming)
  - Easy to visualize HTML structure
  - Easy to see how dynamic data fits in
  - Can do either on server or browser

AngularJS view template   (HTML/CSS)

```
...
<body>
   <div class="greetings">
       Hello {{models.user.firstName}},
    </div>
   <div class="photocounts">
       You have {{models.photos.count}} photos to review.
    </div>
</body>
```

Angular has rich templating language (loops, conditions, subroutines, etc.)....

Controllers

- Third-generation: JavaScript running in browser

Responsibilities:

- Connect models and views
    - Server communication: Fetch models, push updates
- Control view templates
    - Manage the view templates being shown
- Handle user interactions
    - Buttons, menus, and other interactive widgets

AngularJS controller (JavaScript function)

```
function userGreetingView ($scope, $modelService) {
   $scope.models = {};
   $scope.models.users = $modelService.fetch("users");
   $scope.models.photos = $modelService.fetch("photos");

   $scope.okPushed = function okPushed() {
      // Code for ok button pushing
   }
}
```

Angular creates $scope and calls controller function called when view is instantiated

Model Data

- All non-static information needed by the view templates or controllers
- Traditionally tied to application's database schema
  - Object Relational Mapping (ORM) - A model is a table row
- Web application's model data needs are specified by the view designers

  But need to be persisted by the database
- Conflict: Database Schemas don't like changing frequently but web application model data might (e.g. user will like this view better if we add … and lose ...)

Angular doesn't specify model data (JavaScript objs)

- Angular provides support for fetching data from a web server
    - REST APIs
    - JSON frequently used

On Server, JSON:

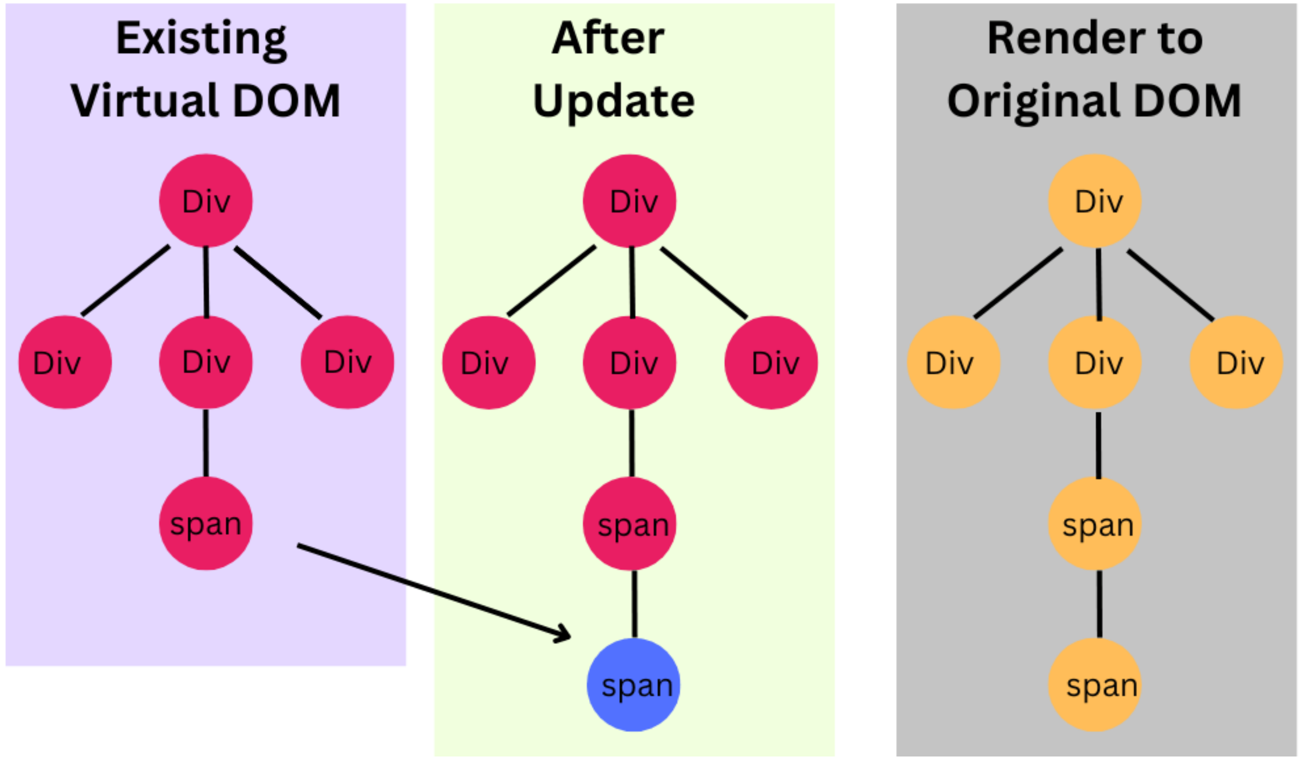
```
var userSchema = new Schema({
  firstName: String,
  lastName: String,
});
var User = model('User', userSchema);
```

Fourth-generation frameworks

Examples: React.js, Vue.js, Angular(v2)

- Many of the concepts of previous generations carry forward
  - JavaScript in browser
  - Model-view-controllers
  - Templates
- Focus on JavaScript components rather than pages/HTML
  - Views apps as **assembled reusable components** rather than pages.
  - Software engineering focus: modular design, reusable components, testability, etc.
- Virtual DOM
  - Render view into DOM-like data structure (not real DOM) using JS
  - Benefits: Performance, Native apps

Virtual DOM: In memory



Controller's role in Model, View, Controller

- Controller's job to fetch model for the view
  - May have other server communication needs as well (e.g. authentication services)
- Browser is already talking to a web server, ask it for the model
- Early approach: have the browser do a HTTP request for the model
  - First people at Microsoft liked XML so the DOM extension got called: **`XMLHttpRequest`**
- Allowed JavaScript to do a HTTP request without inserting DOM elements
- Widely used and called **AJAX** - **A**synchronous **J**avaScript **a**nd **X**ML
- Since it is using an HTTP request it can carry XML or anything else
  - More often used with JSON

XMLHttpRequest

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

XMLHttpRequest: status codes?

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

XMLHttpRequest: status codes?

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

200 OK
request succeeded, requested object later in this message

301 Moved Permanently
requested object moved, new location specified later in this message (Location:)

400 Bad Request
request message not understood by server

404 Not Found
requested document not found on this server

505 HTTP Version Not Supported

XMLHttpRequest

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

Event handling

```
function xhrHandler(event) {
  // this === xhr
  if (this.readyState != 4) { // DONE
      return;
  }
  if (this.status != 200) { // OK
      return; // Handle error ...
  }
  ...
  let text = this.responseText;
  ...
```

XMLHttpRequest event processing

- Event handler gets called at various stages in the processing of the request

0	UNSENT	open() has not been called yet.
1	OPENED	send() has been called.
2	HEADERS_RECEIVED	send() has been called, and headers and status are available.
3	LOADING	Downloading; responseText holds partial data.
4	**DONE**	**The operation is complete**.

- Response available as:

raw text	- `responseText`
XML document	- `reponseXML`

- Can set request headers and read response headers

XMLHttpRequest

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

Event handling

```
function xhrHandler(event) {
  // this === xhr
  if (this.readyState != 4) { // DONE?
      return;
  }
  if (this.status != 200) { // OK?
      return; // Handle error ...
  }
  ...
  let text = this.responseText;
  ...
```

Traditional AJAX uses patterns

- Response is HTML

```
elem.innerHTML = xhr.responseText;
```

- Response is JavaScript

```
eval(xhr.responseText);
```

Neither of the above are the modern JavaScript framework way:

- Response is model data (JSON frequently used here)

```
JSON.parse(xhr.responseText);
```

Fetching resources with XMLHttpRequest via REST

- Can encode model selection information in request in:

URL path:

```
xhr.open("GET","userModel/78237489/fullname");
```

Query params:

```
xhr.open("GET","userModel?id=78237489&type=fullname");
xhr.send();
```

Posting resources with XMLHttpRequest via REST

- Can encode model selection information in request in:

url = “userModel");

Request body:

```
xhr.open("POST", url);
xhr.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded");
xhr.send("id=78237489&type=fullname");
```

Other Transports: HTML5 WebSockets

- Rather than running over HTTP, HTML5 brings sockets to the browser
  - TCP connection from JavaScript to backend Web Server - Bidirectional pipes
- Event-based interface like XMLHttpRequest:

```
 let socket = new WebSocket("ws://www.example.com/socketserver");
socket.onopen = function (event) {
  socket.send(JSON.stringify(request));
};

socket.onmessage = function (event) {
  JSON.parse(event.data);
};
```

Trending approach: GraphQL

- Standard protocol for backends from Facebook
  - Like REST, server exports resources that can be fetched by the web app
  - Unlike REST
    - GraphQL is a **query language** for APIs and a **runtime** for executing those queries by using a type system you define for the data.
    - Exports a "schema" describing the resources and supported queries.
    - Client specifies what properties of the resource it is interested in retrieving.
    - Unlike REST, which uses multiple endpoints to retrieve different data, GraphQL typically exposes a **single** endpoint.
- Gaining in popularity particularly compared to REST
  - Gives a program accessible backend - **Application Programming Interface (API)**

Building a Web app

You are hired as a software engineer

The goal is to construct a web application that **recommends movies to users based on their profiles and hobbies**

In pairs, devise a plan in which you outline how you would go about achieving the goal.