

Front End Programming

Brief history of Web Applications

- 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**: stylized decomposition of applications
- 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.
- **View:** what the web page looks like
 - HTML/CSS.
- **Controller:** fetch models and control view, handle user interactions
 - JavaScript code.

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:

- Mongoose's Object Definition Language (ODL) has "models"

```
var userSchema = new Schema({  
  firstName: String,  
  lastName: String,  
});  
var User = mongoose.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)
 - Benefits: Performance, Server-side rendering, Native apps