Front End

Sridhar Alagar

Motivation

Build a Web App to create Todo list

We can write a JavaScript code using DOM properties

It can be lengthy and tedious

Code to create Todo list using React

```
function TaskList({ tasks }) {
 return (
    <ul>
      {tasks.map(task => (
        <li key={task.id}>
          <span>{task.name}</span>
          <button type="button">Delete</button>
        </li>
      ))}
    </ul>
  );
}
```

Code to create Todo list using Vue

```
<ul>
  <li v-for="task in tasks" v-bind:key="task.id">
    <span>{{task.name}}</span>
    <button type="button">Delete</button>
  </li>
</ul>
```

Brief history of Web App

Initially, just static HTML files with forms and input

JavaScript added some interactivity

Common Gateway Interface (CGI):

- Certain URLs map to an executable program that generates HTML page
- New process created to execute and exits after web page is complete
- Introduced the notion of stateless servers
  - Each request is separate and complete. No state from previous requests
- Creates a new process for every request – increased CPU and memory usage
- Not scalable

First generation web-app framework

Example: PHP, ASP.net, Java servlets

Ran persistent servers with multi-threading capability

Template-based system to insert dynamic content into predefine spots in a page

Mix code (logic) and HTML (presentation) – spaghetti hard to maintain

Web specific library packages

- URL handling
- HTML generation
- Session management
- Database operation

Synchronous communication – every interaction with the server would trigger a page reload

Second generation web-app framework

Example: Ruby on Rails, Django

Model-View-Controller: decompose application along separation of concerns

Object-Relational Mapper (ORM)

- Classes and objects instead of tables and rows
- Allowed developers to use database using objects/methods rather than raw SQL queries
- Allows developer to work with database more efficiently

Routing and clean URL

- In Rails, a URL like /posts/123 would be routed to the PostsController to display the post with the ID of 123

Asynchronous communication

- Used AJAX to update parts of webpage without a full-page reload

Third generation web-app frameworks

Example: Angular JS

Client-side Rendering (CSR)

- Instead of generating entire HTML page on the server, frameworks deliver a single page and use JavaScript to dynamically update content
- Interactive and quick responding applications – round trip to server avoided

Framework not dependent on server-side capabilities

- Separates front-end and back-end development concerns
- Allowed developers to use database using objects/methods rather than raw SQL queries
- Front-end acts as a client to use back-end services (database, authentication, etc.) through APIs

Many of the concepts from previous generations carry forward

- MVC
- Templates – HTML/CSS view description

Model-View-Controller (MVC) pattern

Model: manages the application's data

JavaScript objects. PhotoApp: Users' names, pictures, comments, etc.

View: responsible for how the webpage is displayed

HTML/CSS. PhotApp: view users, view photos with comment

Controller: fetch model and control views, handle user interactions

JavaScript code. PhotoApp: DOM event handlers, server communication

MVC pattern has been around since the late 1970s

Conceived by a scientist while working on Smalltalk project at Xerox PARC

View generation

Any WebApp should eventually generate HTML/CSS

Templates are commonly used technique. Basic idea:

- Write HTML doc with parts of the page that remain the same
- Add pieces of code to generate dynamic part of the page
- The template is expanded by executing the code fragments, substituting the results into the document

Benefits of templates (compared to DOM programming)

- Easy to visualize HTML structure
- Easy to see how dynamic data fits in
- Can be used in server or client

View template in AngularJS

```
...
<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

Controllers

In third-generation frameworks, controller is JavaScript running on browser

Responsibilities

- Connect model and views
  - May require server communication using fetch, push models
- Manage the view templates being shown
- Handle user interaction – button, menus, etc.

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 ‘userGreetingView’ 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 some other view)

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 app

Library vs Framework

Library – collection of code that provides a narrow set of functionalities

Framework – provides a complete solution to build an entire application

- Opinionated – Provides the structure and controls the flow.
- Unopinionated – provides tools. Let the user structure and control the flow

Technically, React is a library.

But React is mostly used with Redux, React Router, etc. So, it is often called as framework

Hello React…

React apps are made out of components

Components can be as small as a button or as large as webpage

React components are JavaScript functions that return a markup

Sources

1. MDN - Introduction to client-side frameworks
2. Stanford CS 142 Web Application – Lectures