Future Web App Technologies

Mendel Rosenblum

MEAN software stack

- Stack works but not the final say in web app technologies
- **A**ngular.js
  - Browser-side JavaScript framework
  - HTML Templates with two-way binding
  - Directives and services for modular design
  - Much single page application support - routing, model fetching, etc.
- **N**ode.js / **E**xpress.js web server code
  - Server side JavaScript
  - High "concurrency" with single-thread event-based programming
- **M**ongoDB "document" storage
  - Store frontend models
  - Storage system support scale out (sharing and replication), queries, indexes

Angular criticisms

- Digest cycle overheads on pages with large numbers of items
    - Consider the watches on a large data table with multiple columns
    - HTML template with two-way binding
- DOM access overhead
    - Access to the browser DOM is slow
- Large size of JavaScript
    - Needs to download, initialize, and digest before anything appears
    - Problematic on mobile
- Software engineering problems programming at scale
    - Scope inheritance, JavaScript lack of typing, interface definitions, etc.
- Considered too opinionated by some

Front-end alternative - ReactJS from Facebook

- JavaScript framework - Does view component only
  - Less opinionated than Angular - Need model fetch, routing, etc. packages to work
- View declared in JavaScript (more accurately a lang translated to JavaScript)
  - Angular: HTML with JavaScript embedded
  - React: JavaScript with HTML embedded
- Basic building block: **Components**
  - Have a function **`render()`** that returns HTML-like structure
  - Accepts inputs (**`this.props`**)
  - Have an internal state (**`this.state`**)
- Components are reusable pieces composed to form view

React in JavaScript

```
var CommentBox = React.createClass({displayName: 'CommentBox',
  render: function() {
    return (
      React.createElement('div', {className: "commentBox"},
        "Hello, world! I am a CommentBox."
      )
    );
  }
});
ReactDOM.render(
  React.createElement(CommentBox, null),
  document.getElementById('content')
);
```

React using JSX

- Encourage to use JSX (XML-like language translated to JavaScript)

```
var CommentBox = React.createClass({
  render: function() {
    return (
      <div className="commentBox">
        Hello, world! I am a CommentBox.
      </div>
    );
  }
});
ReactDOM.render(
  <CommentBox />,
  document.getElementById('content')
);
```

`this.props` input to components

```
var Comment = React.createClass({
  render: function() {
    return (
      <div className="comment">
          {this.props.author}
          {this.props.children}
      </div>
    );
  }
});

<Comment author="Mendel Rosenblum">This is one comment</Comment>
```

Virtual DOM

- React component render() functions results are places in a Virtual DOM
    - Highly optimized one-way binding process
        - Only components whose `this.props` or `this.state` change are updated
    - Much faster access than the real DOM
- React efficiently pushes the Virtual DOM to the Browser's DOM
    - Only the parts of the Browser's DOM that change are updated
- Key feature of React
    - Decouples React from the browser DOM

ReactJS input

- Like AngularJS: Components can JavaScript event handlers on clicks and form/textarea changes that update `this.state` or `this.props`
  - More efficient that the AngularJS binding to a JavaScript variable and then using watch/digest to detect changes: one vs two way binding.
- Encourages a more holistic view of web app state. Example: Redux
  - Put all web app browser state in a common abstraction: a state store
  - All inputs (user, network, components, etc.) go into store
  - Components take their inputs from the store
- More elegant solution that AngularjS with $on/$broadcast

ReactJS benefits over AngularJS

- High performance for rapidly changing views
  - Less time calling into Browser's DOM
- Server-side rendering
  - Can run React either on server or browser
  - Faster startup by pushing HTML from server
- React Native
  - Have native mobile apps for iOS and Android that speak React

Angular Version 2 - Renamed **Angular**

- Very different from AngularJS
  - Doubled down on the AngularJS Directive abstraction - focus reusable components
- Components written in extended Typescript (ES6 + Typescript + annotations)
  - Got rid of scopes, controllers, two-way binding
  - Directives are components with a HTML template and corresponding controller code
- Similar architecture to ReactJS
  - Faster rendering and can support server-side rendering

Node.js criticisms

- Callback hell - TJ Holowaychuk's why Node sucks:
  1. you may get duplicate callbacks
  2. you may not get a callback at all (lost in limbo)
  3. you may get out-of-band errors
  4. emitters may get multiple “error” events
  5. missing “error” events sends everything to hell
  6. often unsure what requires “error” handlers
  7. “error” handlers are very verbose
  8. callbacks suck

- JavaScript lack of typing checking
- Concurrency support (e.g. crypto operations)
- Performance overheads

Go Language

- System programming language released in 2007 by Google
  - Done by original Unix authors (Reacting to complexity of C++/Java and Python at scale)
  - From Wikipedia:

    A **compiled**, **statically typed** language ..., with **garbage collection**, **memory safety** features and **CSP-style concurrent programming** …

- Cross C & scripting languages
  - Productive and readable programs
  - C-like but got rid of unnecessary punctuations
  - Super fast compiler

Go language features

- Like dynamic languages, types are inferred

```
intVar := 3;
stringVar := "Hello World";
```

- Functions can return multiple values

```
func vals() (int, int) {
    return 3, 7
}
a, b := vals()
```

  - Common pattern: `return result, err`

Go language features

- Can declare types and allocate instances

```go
type person struct {
    name string
    age  int
}
s := person{name: "Sean", age: 50}
```

- Automatic memory management using garbage collection

Go concurrency - threads

- goroutine is a lightweight thread of execution

```
go processRequest(request);
```

  - Encourages using tons of threads. Example: per request threads

- Has channels for synchronization

```
messages := make(chan string)
go func() { messages <- "ping" }()
msg := <-messages
```

  - Also locks for mutual exclusion

MongoDB criticisms

- Lots - Pretty lame database
    - Loses data, doesn't scale well
    - Large space overheads for objects and indexes
    - Query language: Not SQL?
    - Limited concurrency control (only single object transactions)
- Many other databases
    - Cloud storage offerings are getting better
    - Example: Spanner (Globally consistent, scalable, relational database)

Alternatives to building your own full stack

- Frontend centric: Model storage approach
  - Firebase
    - Develop your web app (MVC) and store models in the cloud services
    - Pushes new models to the web app when things change
    - Example sweet spot: Top scorer list for a game
- Backend centric: Schema driven approach
  - Describe data of application
  - Auto generate schema and front-end code
    - Limited to form-like interface
- Various systems that promises to take a specification of your web app and deliver it

Full stack engineering

- Tall order to fill
    - Make pretty web pages by mastering HTML and CSS
    - Architecture scalable web service
    - Layout storage system system sharding, schema, and indexes
- Typically people specialize
    - The expert in CSS is different than expert in database schema is different from the ops team

Looking to the future

- Cloud providers will offer a platform that most web applications can just build off
  - LIke people don't write their own operating system anymore.
  - Technologies and app demands have been changing so much we still in the roll your own phase.
- Pieces are coming together
  - World-wide scalable, reliability, available storage systems (e.g. Google's spanner)
  - Serverless computing platforms (e.g. Amazon Lambda)
  - Cloud services - Pub/sub, analytics, speech recognition, machine learning, etc.

Example Cloud Offering: Google Firebase

- Client library for most app platforms (web, ios, android, etc.)
  - App focus - No backend programming
- Storage
  - Realtime Database - Shared JSON blob (noSQL) with watches and protection
    - Client directly queries database (no web servers needed)
  - Cloud Storage - Blob storage for bigger things like files
    - Use for unstructured data you don't want to encode into JSON in the realtime database
- Authentication - Let users login
  - Supports accounts/passwords, Google, Facebook, OAUTH, etc.

Google Firebase (continued)

- Hosting
  - Global content distribution network (CDN)
    - Distribute read-only parts (e.g. HTML, CSS, JavaScript) with low-latency
  - Remote Config - Distribute different versions
    - A/B testing, customize versions
  - Cloud Function - Serverless computing - Triggers on network or storage events
    - Allows for backend functionality without needing servers

- Application monitor - Provides a dashboard
  - Google Analytics - Track application usage (e.g which routes, etc.)
  - Performance Monitoring - Track request timings, etc.
  - Crash reporting - Upload information about failures
  - Crashlytics - Classify crashes and provide alerts

Google Firebase (continued)

- User Communication
  - Cloud Messaging - Send messages or notifications to app users
  - Invites - Allow users to point other users at your app
- Dynamic Links - Deep linking support
  - Direct users to native mode apps
- Google Integration
  - Admob - Show ads in your app
  - Adwords - Advertise your app on Google
  - App Indexing - Have your app show up in Google Search

Google Cloud offerings

- Everything is an Application Programming Interface (API)
  - REST commonly used
- Language Translation
- Information extraction services:
  - Video Analysis
  - Speech Analysis
  - Text Analysis
- Conversational user interface support (e.g chatbot)

Trending Web App Frameworks - CS142?

- View - JavaScript/TypeScript/CSS or Native app
  - React.js, Angular (2), Vue.js
    - View-only: Components packaging HTML/Templates
- State Management
  - Reactive programming
  - Observable pattern
  - Becoming similar to old distributed system consistency issues
- Backend communication - Graphql vs REST
- Backend - Serverless, perhaps Go language
- Storage - SQL query language - relational-like database

Web Apps versus Native Apps

- Web Apps advantages:
  - Available on all platforms - Smaller, faster development
  - Easy "update" of application
  - Customize application per user
- Native apps
  - Native look and feel user interface
  - Integrate with host platform
- Hybrid approach: Embed browser in native app
- Backend can be largely the same for both - (e.g. REST APIs)