Future Web App Technologies

Mendel Rosenblum

MERN software stack

- **R**eact.js
  - Browser-side JavaScript framework - Only View/controller parts
  - Javascript/CSS with HTML templates embedded (JSX)
  - Unopinionated: leaves much SPA support to 3rd parties - 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 model data
  - Storage system support scale out (sharing and replication), queries, indexes
- Commonly listed alternatives: Angular(2) and Vue.js

Angular (AngularJS Version 2)

- Very different from AngularJS (First version of Angular)
  - Doubled down on the AngularJS Directive abstraction - focus reusable components
- Components written in extended Typescript (ES6 + Typescript + annotations)
  - Got rid of AngularJS 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
- Vue.js - Done by former AngularJS developer, community supported
  - Similar component architecture
  - Mostly big companies in China

Current generation: React.js, Angular, Vue.js

- Common approach: Run in browser, component building blocks, etc.
  - A sign the area is maturing?
- Specification using traditional web programming languages
  - Advanced JavaScript - Babel
  - Templated HTML
  - CSS for layout (e.g. grid) and styling
- Decoupling from the browser's DOM with a Virtual DOM

Virtual DOM

- Component render() functions results are places in a Virtual DOM
  - Optimized one-way binding process
    - Only components whose props or state change are updated
  - Much faster access than the real DOM
- Efficiently pushes the Virtual DOM to the Browser's DOM
  - Only the parts of the Browser's DOM that change are updated
- Decoupling from the browser DOM enabled component use other places
  - Server-side rendering
  - Native client

React.js and the future of Web Apps

- Choice of describing UI using HTML/CSS/JavaScript is surprising
- Large advantage to be on the dominate platform
  - Available components
- React.js used of JSX embedded in Javascript is problematic
  - Lots of gotchas when learning (`this`, iteration, etc.)
  - Loses ability to use compiler technology on templates (e.g. Svelte)
    - Declarative languages are more popular this days

State management

- Reactive programming paradigm
- Example: Redux - A Predictable State Container for JS Apps
  - Put all web app browser state in a common abstraction: a state store
  - All inputs (user, network, components, etc.) go into store
  - Components get their inputs from the store
  - Eases support for offline operation
- Example: Relay - The production-ready GraphQL client for React
  - Model fetching and caching using Graphql - local state store
  - Specify as part of React.js component - render method specifies model data query
  - Uses compiler to bunch together all component queries into a single GraphQL query to backend

Browser extension: ServiceWorker

- Use browser "service workers" to cache web application
  - JavaScript Web Workers - JavaScript extension to run code in background
    - Runs isolated but in parallel with the other JavaScript
    - Communicate with using postMessage/events
    - Can stick around after web app exits
  - Network proxy that allows worker to interpose on web app's request to web server
  - Cache - storage mechanism for Request / Response object pairs
    - Store contents of web app request/responses so they can replayed without backend
- Supports:
  - Super fast web app startup - All components and even model data already in the web app
  - Offline operation - Can run out out of the server worker cache

Progressive Web Applications

- Leverage ServiceWorkers to get native app characteristics
  - Fast startup, view switching, and offline support
- Lots of other web app niceties rolled in
  - HTTPS support for protection
  - Responsive design for different size displays
  - Deep linking
  - Push notifications
  - Google search support
  - Cross-browser
  - Etc.

Browser extension: Web Assembly

- Web Assembly (Wasm) -
  - Binary instruction format for a stack-based virtual machine
  - Portable target for compilation of high-level languages like C/C++/Rust/Go etc
  - Uses a just-in-time compiler to native instructions
- Runs in isolated environment in parallel with JavaScript
  - Like JavaScript Web Workers except with near-native CPU performance
- Allows performance critical legacy code to run in browser
  - Example: Game engines

Web App programming is being used all over

- Mobile environments (iOS and Android)
  - React Native - Supports using React components
  - Ionic - Supports using Angular, React, or Vue
- Desktop environments
  - Electron - Build cross platform desktop apps with JavaScript, HTML, and CSS
    - Extend Node.js with browser functionality (chromium)
    - Example app: Atom - A hackable text editor for the 21st Century
  - Ionic

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 - special devices and services
- Backend can be largely the same for both - (e.g. REST/GraphQL/RPC APIs)
  - Need legacy support

Web Servers

- Express.js functionality
  - Code handlers to process requests from clients
  - Routing URLs/verbs to handlers
  - Middleware for common processing
- Functionality pretty fundamental
  - Alternatives basically use the same functions just different languages
  - Callbacks/Promises vs threads is a big difference

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 - Can use Typescript now.
- Concurrency support (e.g. crypto operations) & Performance overheads
- Node.js V2 - Deno - TypeScript and smaller trusted computing base

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

```
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?
- Many other databases
  - Cloud storage offerings are getting better
  - Example: Spanner (Globally consistent, scalable, SQL database)
- Open source infrastructure company in a SaaS world

Alternatives to building your own backend

- 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 (e.g. no programming) 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

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 vs gRPC
- Backend - Serverless, perhaps Go language, Microservices
- Storage - SQL query language - relational-like database