Rant warning
Slava in 2012

Built apps in JS

- Backbone.js
- Underscore.js templates
- UMD.js loader
- Grunt.js
Web Development in the past

Make HTML pages
Add some styles with CSS
Add some JS for interactivity
Publish the pages
Later

Make HTML templates

Have a library to turn the templates into HTML

Hook up the library to the server
Later again

CSS is getting out of control

Use a library to generate CSS out of templates

Hook up the library into the build process
Later again

Images are too large

Use imagemagick tool to compress the images

Hook up a command to run imagemagick on build
Later later

New version of JavaScript spec is released

But most browsers are behind and still use the old version of JavaScript

Use a set of compilers and libraries to setup the translation

Hookup the compilers into the build process
The list goes on

- Compile CSS
- Compile JS
- Compile Templates
- Compress Images
- Minimize and compress JS
- Upload everything to the website
- etc
Solutions people built

Task runners like Grunt

Assets processors like Gulp

All in one solutions like Webpack
GRUNT
The JavaScript Task Runner

Why use a task runner?
In one word: automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes. After you’ve configured it through a Gruntfile, a task runner can do most of that mundane work for you—and your team—with basically zero effort.

Why use Grunt?
The Grunt ecosystem is huge and it’s growing every day. With literally hundreds of plugins to choose from, you can use Grunt to automate just about anything with a minimum of effort. If someone hasn’t already built what you need, authoring and publishing your own Grunt plugin to npm is a breeze. See how to get started.
Automate and enhance your workflow

$ npm install gulp-cli -g
$ npm install gulp -D
$ touch gulpfile.js
$ gulp --help
bundle your assets
CSS Processors

CSS lacks some nice features

- Variables
- Functions
- Nested rules, remove duplication

All of these are possible with an additional step of compilation
Getting Started

Less is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and many other techniques that allow you to make CSS that is more maintainable, themeable and extendable.

Less runs inside Node, in the browser and inside Rhino. There are also many 3rd party tools that allow you to compile your files and watch for changes. The quickest place for first experiments with Less is our online editor.

For example:
@base: #f938ab;

.box-shadow(@style, @c) when (iscolor(@c)) {
  -webkit-box-shadow: @style @c;
  box-shadow: @style @c;
}

.box-shadow(@style, @alpha: 50%) when (isnumber(@alpha)) {
  .box-shadow(@style, rgba(0, 0, 0, @alpha));
}

.box {
  color: #fe33ac;
  border-color: #fdceda;
}

.box div {
  -webkit-box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
}
CSS with superpowers

Sass is the most mature, stable, and powerful professional grade CSS extension language in the world.
JavaScript Preprocessors

JavaScript used to be stagnating until 2015

There were a lot of quirks, different things used to be ugly or complicated

A lot of alternatives arised:
- Coffeescript
- Typescript
CoffeeScript is a little language that compiles into JavaScript. Underneath that awkward Java-esque patina, JavaScript has always had a gorgeous heart. CoffeeScript is an attempt to expose the good parts of JavaScript in a simple way.

The golden rule of CoffeeScript is: “It’s just JavaScript.” The code compiles one-to-one into the equivalent JS, and there is no interpretation at runtime. You can use any existing JavaScript library seamlessly from CoffeeScript (and vice-versa). The compiled output is readable, pretty-printed, and tends to run as fast or faster than the equivalent handwritten JavaScript.
Account = (customer, cart) => {
  @customer = customer
  @cart = cart

  $('shopping_cart').on('click', (event) => {
    @customer.purchase @cart
  });
};

var Account;

Account = function(customer, cart) {
  this.customer = customer;
  this.cart = cart;

  return $('shopping_cart').on('click', (event) => {
    return this.customer.purchase(this.cart);
  });
};
Coffeescript

Ruby-like syntax (somewhat resembling Python)

Shorthands for Classes, bound functions, etc

Removed weak comparison, prototypes, other confusing things
TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Any browser. Any host. Any OS. Open source.

Download Documentation
function greeter(person: string) {
    return "Hello, " + person;
}

let user = [0, 1, 2];

document.body.innerHTML = greeter(user);

Re-compiling, you'll now see an error:

error TS2345: Argument of type 'number[]' is not assignable to parameter of type 'string'.
Typescript

JavaScript with extra features like Classes, bound functions, etc

Added types support, similar to Java/C#/C++

Static types help to find mistakes early

Especially good on a big codebase working on a big team
ES2015, ES2016, ES2017

In 2015 JavaScript got a major update:

- Classes became part of the official syntax
- Anonymous, bound functions
- Generators
- Promises
- Async/Await
- Map, Set and other data structures
- Nice syntactic additions
- Basically eliminated the need of CoffeeScript
- Available in most major browsers (rip IE) and node
Babel is a JavaScript compiler.

Use next generation JavaScript, today.

<table>
<thead>
<tr>
<th>Put in next-gen JavaScript</th>
<th>Get browser-compatible JavaScript out</th>
</tr>
</thead>
<tbody>
<tr>
<td>const x = [1, 2, 3];</td>
<td>var x = [1, 2, 3];</td>
</tr>
<tr>
<td>foo([...x]);</td>
<td>foo([].concat(x));</td>
</tr>
</tbody>
</table>

Check out our REPL to experiment more with Babel!
Front-end Frameworks

Instead of using basic templates, organize the code into “components”

Each component is responsible for managing its own state and representation

The site doesn’t degrade to unbearable slow over time

There are literally tens of frameworks, we are going to show you names of the most popular ones but they are too complicated to go into details in one lecture
React
A JavaScript library for building user interfaces

Get Started  Take the Tutorial
The Progressive JavaScript Framework

Approachable
Already know HTML, CSS and JavaScript? Read the guide and start building things in no time!

Versatile
An incrementally adoptable ecosystem that scales between a library and a full-featured framework.

Performant
20KB min+gzip Runtime
Blazing Fast Virtual DOM
Minimal Optimization Efforts
Apache Cordova

Takes your front-end app and packages it into a native iOS/Android app with a webview

Allows you to use plugins to access native features like camera, geolocation, gyroscope, etc

Old and super buggy

But still gets the work done

cordova.apache.org
Build cross platform desktop apps with JavaScript, HTML, and CSS
React Native

Allows you to write native iOS/Android apps

But treating iOS/Android native views as if they were React.js components

A lot more complicated than Cordova and requires understanding of the actual mobile platforms, but can achieve much higher quality of apps

https://facebook.github.io/react-native/
Backend Frameworks

Give a structure to your project

Simplify common things like Accounts

Some of them do more work with databases

Some can have a very opinionated approach to the build process
Express

Fast, unopinionated, minimalist web framework for Node.js

$ npm install express --save
THE FASTEST WAY TO BUILD

JAVASCRIPT APPS

Meteor is an open source platform for web, mobile, and desktop.

INSTALL NOW
Version 1.8

TUTORIAL
Blazing-fast static site generator for React

Get Started →
START

Learning Next.js

To build server rendered JS web apps with React

Start Now
Package managers

NPM is a defacto package manager for node

dep was stagnating for a while, didn’t have good conflict resolution, would duplicate a lot of code unnecessarily

Several competitors came up but only one made a difference: yarn
FAST, RELIABLE, AND SECURE DEPENDENCY MANAGEMENT.

GET STARTED  INSTALL YARN

Stable: v1.3.2
Node: ^4.8.0 || ^5.7.0 || ^6.2.2 || ^8.0.0

Ultra Fast.
Yarn caches every package it downloads so it never needs to download it again. It also parallelizes operations to maximize resource utilization so install times are faster than ever.
Testing frameworks

Like anything else in JavaScript ecosystem, there was a huge competition for the best testing framework but in the end of the day probably only one or two matter.

The function of the testing framework is to provide a system for:

- Unit testing
- End-to-end testing
- Mocking services
Jest

Delightful JavaScript Testing

TRY OUT JEST  GET STARTED  WATCH TALKS  LEARN MORE

Easy Setup

Complete and easy to set-up JavaScript testing solution. Works out of the box for any React project.

Instant Feedback

Fast interactive watch mode runs only test files related to changed files and is optimized to give signal quickly.

Snapshot Testing

Capture snapshots of React trees or other serializable values to simplify testing and to analyze how state changes over time.
Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing *simple* and *fun*. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub.
A lot of things have happened

- 30-80 front-end frameworks
- Dozens of backend frameworks
- Tens of build tools
- Hundreds of libraries doing the same thing: promises, requests, utilities, etc
- At least 3 JavaScript compilers
- Around 4 similar looking compile-to-CSS languages
- Many many more to come

Innovation or a distraction?