Misc topics and wrap-up

Michael Chang

Spring 2020

Plan for today

A few misc / advanced topics

   Stuff you probably should know about

   Stuff that's just cool/new/different

Publishing (deploying) your web app

General advice and conclusions

   Where to go from here, etc.

Misc topics

Browser compatibility: Babel

A JS compiler--input is JS, output is also JS

Input code can use newer or even non-standard features

Output will be broadly compatible

Can customize to define target browsers/environments

Bundling: Webpack

"Bundles" JavaScript files together

Plugins allow you to "transform" the JS

E.g. Babel, minification

Tools simplify development workflow

E.g. `webpack-dev-server`

Misc topics

Testing and robustness

- In production, make sure your code handles errors
  - User errors, code errors, malicious attempts to break your page
- Testing frameworks like Jasmine or Mocha can help

Type checking with TypeScript

- Most syntax is the same as JS
- Add types, interfaces to check for errors
- Compiles to standard JS output

Misc topics

Bidirectional communication: WebSockets

- Allows server to notify client when events occur
- Wrapper libraries like socket.io can make using this easier
  - Uses multiple techniques to ensure compatibility
  - Very simple API
- This is what causes the page to auto-reload when you save your files
  - See files in lib

Frontend frameworks

What are they

Frameworks change how you develop your web app

More than just providing some functions

Often come with their own concepts, best practices, pitfalls

Examples: React, Vue.js

Frontend frameworks

Pros

For larger applications, can simplify repeated code

- E.g. showing/hiding elements, handling events

Many (**many**) reusable components you can drop into your app

- E.g. calendars, text editors, style templates, …

Cons

Very large footprint (>1k dependencies)

- For smaller apps, often not worth it

Framework-specific patterns aren't portable

- E.g. moving from React to Vue may be hard

Plugin/component quality highly (**highly**) variable

Frontend frameworks

My advice

Foremost, web fundamentals will always be important

  Everything eventually comes back to HTML/CSS/JS

For small applications, it's probably not worth it

  You may end up spending a lot of time fighting the framework to get something simple done

But you should learn a framework

  If only to understand different paradigms

  Follow a tutorial, try converting your small app

Be thoughtful about their communities

  Many opinions, some very strong

  Many people are good, helpful, thoughtful

So far

A few misc / advanced topics

Stuff you probably should know about

Stuff that's just cool/new/different

Publishing (deploying) your web app

General advice and conclusions

Where to go from here, etc.

Terminology

Web hosting: the web server

If running an API or database, they would often be running here too (but could be separate)

Related: cloud hosting

- You don't get a physical server
- Can be moved between machines for load balancing
- This is basically every web host these days

Domain name

Can be bought (registered) separate from web hosting

Some hosts provide domain name registration services too

Publishing options

Static web pages

Github pages

If you keep your files on github, you can publish them directly

Stanford web site

Accessible via web.stanford.edu

Publishing options

Node.js servers

Heroku

Instructions on project page

Cloud hosting (AWS, GCP, Azure)

Many options here, from hosting your code to having a full server you can run (almost) anything you want on

Nontrivial initial setup

Requires familiarity with command line, Linux/UNIX

May require some system admin/maintenance

So far

A few misc / advanced topics

Stuff you probably should know about

Stuff that's just cool/new/different

Publishing (deploying) your web app

General advice and conclusions

Where to go from here, etc.

Common questions

How do I keep up with web tech?

Common questions

How do I keep up with web tech?

Short answer: don't stress about it

Common questions

How do I keep up with web tech?

Web tech changes frequently

But tech doesn't disappear overnight

E.g. many apps still use Java, PHP, ASP.NET

Should you learn about the new stuff? Yes, absolutely

Often solves long-standing problems, simplifies your life

Do you need to learn every technology the day it comes out? Not at all

Totally okay to let things settle, work out bugs

Common questions

Which language/framework will win?

Or: I'm seeing lots of jobs asking for ___. Should I use that for all my apps?

Or: What if I learn a framework and it becomes obsolete?

Common questions

Which language/framework will win?

Short answer: no need to jump on the hype train

Common questions

Which language/framework will win?

Start with fundamentals

HTML, CSS, JS, APIs... these aren't going anywhere

Focus on concepts, not syntax

How to organize components? How to use data from API in the app? How are page reloads handled?

Use tools suited for the task at hand

E.g. consider project size, other dependencies

It's not actually a competition

There's plenty of space on the web for many different frameworks

General thoughts

Web development as software engineering

The more code you read and write, the better your judgment will be

  This takes times; there's really no shortcut

Set small, achievable, testable goals

Beware of unnecessary complexity

  Simpler is always better

He was just putting the finishing touches on the optimization when it was time to fill out the management form for the first time. When he got to the lines of code part, he thought about it for a second, and then wrote in the number: -2000.

I'm not sure how the managers reacted to that, but I do know that after a couple more weeks, they stopped asking Bill to fill out the form, and he gladly complied.

-- Source

What's next?

Other cool classes

User interfaces, design, HCI (CS147)

Databases (CS145)

Networking (CS144)

Programming languages, types, compilers (143, 242)

Security, cryptography (CS155, CS255)

What's next?

Read lots of code

Find examples, see how people do things

Good and bad examples are each valuable experience

Build something fun

Set reasonable expectations

Pick tools that require some learning, but are somewhat familiar

Develop incrementally

What's next?

Read lots of code

Find examples, see how people do things

Good and bad examples are each valuable experience

Build something fun

Set reasonable expectations

Pick tools that require some learning, but are somewhat familiar

Develop incrementally

...Then build something useful

Don't get too attached to your first few projects

But even very small projects can have a huge impact

Thank You!