The Modern Developer's Workflow

- Step-by-step, concept-by-concept, "why"
- Git, node.js, CSS, Mobile-first, JavaScript, Go-live, Jobs
  - Git: collaborate with other dev., Git, GitHub, command-line basics
- Node.js: Turn our computer into our personal assistant robot, NPM, Webpack
  - NPM: a grocery store
  - Webpack: a tool for node.js
- CSS:
  - Thoughts process, organized & manageable CSS, recycle-able
    - BEM + PostCSS = magic (philosophy)
- Mobile-first Design & Performance
  - Quick load times
  - making sure that users don't download any more data needed for their particular device
  - (eg. a mobile visitor shouldn't download a huge image that fits for 30inch desktop monitor)
  - Lazy loading for images: images only get loaded in once user scrolls down
- JavaScript
  - Single most valuable skill in terms of what employers are looking for
  - Basic OOP
  - Module pattern: To break up codes into modular, super well-organized chunks
  - Babel & Webpack
    - to write ES6
    - let us use NPM packages in our front-end code
- Go-live
  - Github Pages service to host site
  - Netlify & Cloud (AWS Lambda) Functions
- Jobs
  - Get More Interviews
  - Do well in Interviews
  - Next: what to learn
  - Career progression: continually move forward & avoid career burnout/stagnation

Git -

What is Git, Why, Where to begin ?

See Git in Action

grow Git vocab

* main.css in css dir.

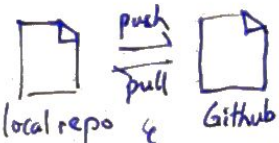
- Project = Repository (Repo) : where git saves all of the data, history, & changes.
- Working Directory = The folder
- Commit : Git doesn't save or store changes we make into its history until we actively (commit those changes
  - = Git's way of "save"
  - database
- Staging : our chance to have fine grained control on what we are committing
  - eg. 1) Working dir (24 files) → 2) Modified files (12 files) → 3) Staging Area (4 files) 4) Committed (4 files)

git status : To get a recap of changes
git add . : Add all changes to staging area
git status
git commit -m 'm' : commit these changes in git history
git checkout -- . : right back to the previous commit stage ⇒ Cat-proof ✓

Git stores all of its history data in a hidden folder in the root of your git repo

- locally (on your hard drives)
  - Loses ur comp, lose your project
  - Collaboration (only u have access to ur comp)
  - ⇒ Solution : Repo Hosting ⇒ GitHub

Git ≠ Github → Yn services that can host ur repo (most popular)



& work with the most updated file (check commits)

Summary
* Nothing is saved, until `commit` in git history

git config --global user.name "~"
git config --global user.email "~"
pwd
cd ← drag folder in # Change Directory
mkdir ← Make Directory
cd folder
git init ← To begin tracking anything that happens within the folder
touch index.html → add <p> ~ * .DS_Store : system file on Mac
git status
git add index.html
git commit -m "My ~"
* To restore to last commit : git checkout --

---

git clone ~

---

1) Setup github
   - * bitbucket offers private repo for free
   1) repo travel-site
   2) git clone https://github.com/LearnWebCode/travel-site-files.git
   3) rename to travel-site
   4) cd travel-site
   5) git remote -v # check location of ~~pre-cloned~~ origin
   6) git remote set-url origin ~ → git remote -v
   7) git push origin master
   8) change index.html ⇒ git add . ⇒ git commit -m "~" ⇒ git push origin master
   9) ~~check this~~ Check changes by clicking commits.

3) Intro to Node.js & NPM
- The need for Automation & Organization
- While we build our site, we want automation & organization
  - Automation: Take a task & find a way to have your computer do it for you. & saves us time & saves us sanity :)
    - eg. small-tasks (3-sec) ⇒ hours you can save with automation
    - eg1 Automatic browser refreshing
    - eg2. CSS Autoprefixer
      - -moz-columns ... / -webkit-columns ...
  - Organization
    - CSS Organization
      - Don't want to place all of the code within one single huge CSS file & stay organized & keeping CSS separated in multiple bite-sized file (eg. header.css, footer.css, banner.css, features.css)
      - & but, ✗ litter the <head> with all .css files; serve the web browser & visitor of our website 1 single css file.
      - & We can setup our computer to watch for any changes to any .css, & when a save is detected, automatically combine all little .css into one single file. eg. (header.css, ...css ⇒ styles.css)
        - ↳ Website will load quickly & we as developer can still stay organized.
  - Package Management
    - Making Bootstrap, Lodash, Normalize.css faster.

19) Node.js intro.

└ JS

Web-browserish tasks	Computer'ish / server'ish tasks
- Opening a modal window / lightbox - Open / close a toggle menu - Adding content to a page without browser refresh Language: JS.	- Accessing computer's file system - Interacting directly with a database - Sending an automated email. Language: PHP, Ruby, Python, ... until Node.js came out

Node.js ⇒ A JS runtime that let us do about anything with JS.

- Ways to use Node

Camp #1	Camp #2
Install Node.js on a server ↳ taking up a request, go hitting up database / API, retrieving data & sending back to user ↳ powering & serving up our public website / app.	Install Node.js on our own personal computer ↳ Using node as a development tool. ↳ To create our own little robot assistant ↳ To automate all sorts of web dev tasks

Install

- node --v

Hands-on demo Ex: Use node to create file

1) mkdir node-experiment
2) cd n
3) touch test.js → open
4) console.log("Hello World!");
5) node test.js // force node to run test.js.
6) console.log(2+2) → step (5) ⇒ 4

↳ whatever we wrote in the file isn't merely get repeated / echoed out to command line, instead it means that node is actually interpret & evaluate JS.

test.js

```
var fs = require('fs'),   // → file system
fs.writeFile(__dirname + "/index.html", "<h1>HTML is great</h1>", function(error) {
  if (error) {
    return console.log(error);
  } else {
    return console.log("Congrats");
  }
});
```

(CL) node test.js → Congrats. → in dir, index.html is created. containing <h1>...

Ex: Use node to programmatically download & save an image from internet.

↳ * Get an img url (public img)

In test.js

```
var https = require('https');   // → not necessary but best practices.
                                // > var myPhotoLocation = 'www';
https.get(myPhotoLocation, function(response) {
  response.pipe(fs.createWriteStream(__dirname + "/mydog.jpg"));
});
```

(CL) node test.js ⟹ the img in dir named mydog.jpg

* We can leverage code that other super intelligent people have already written & shared to the world. ⟹ npm

NPM - Node Package Manager.

- Q: What is NPM? What is package mgt? Why?
- Q: What types of files/packages do we hope to find on NPM?
- Hands-on demo

NPM? a grocery store of code.

- Eg. bake a cake ← flour, sugar, ... or get from grocery store
  - Auto compile CSS, auto concat FS...
- A centralized place where developers share their code with the world for free!
  - If we come together & share our code with each other, we can all create better work & avoid constantly reinventing the wheel.
- A place where bunch of codes lives.

Package mgt?

~~buying things from grocery store~~

- Just sit at home & create a grocery list → personal robot assistant → fetch all ingredients.
- food rot/expire ≈ code bugs/outdated: With PM, we can automatically grab fresh copies of all our ingredients.

Types of files to get.

- Types of Packages:
  - Node packages: automation, build tools, server tasks
  - Other things (nth to do with node): Lodash, Bootstrap, normalize.css

- NPM is bigger than just node, ≈ one-stop shop for any code we need.

Hands-on

1) (CL) cd ~~site~~ travel-site
2) npm install lodash → see node_modules & ~~package-lock.json~~
   ↓ lodash
3) del node_modules
4) npm init -y // more organised way → package.json → projects grocery list / recipe, keep track of all of the packages
5) npm install lodash ⟹ ≈ ②
   ↳ go view dependencies inside package.json.
6) npm install normalize.css
7) del node_modules (rm), > test, because we have dependencies inside package.json, we can
8) npm install ✓ to install back
   ↳ node looks in the directory & look for a file called package.json., then npm will automatically download all dependencies in one fell swoop.
9) git add . git commit -m "Added package.json" git push origin master

---

Q & A:

1) How do I know the exact name of packages?
   ↳ npmjs.com search & browse for packages ~~or~~
   ↳ google for what you're hoping to achieve with the package.
2) How do we actually use the packages JS & CSS files?
   ↳ leave where they are, will learn elegant ways of importing the package code into our own code.
3) Why are the new package files not tracked by git? (try git status)
   ↳ .gitignore (★★★ use that template)
      ↳ eg. thumbs.db (windows.OS will create for any folders with images)
      ↳ node_modules/ (configured our git repo to completely ignore the node modules folder; it's best practices)
         ↳ best practices: As long as we have package.json file, no sense to bloating to repo

✱ (12) read.

(13) Webpack : - a package, the heart of all automation.

↳ A bundler

↳ ~~helps~~ To bundle up those packages that our website depends on in a way that is for visitors of our website to download & consume.

↳ 8m dl weekly. ⇒ Industry standard ; not just an obscure tool I'm big fan of'

1) code.

2) ~~Web~~ Without configuration, webpack only understands JS files.

3) Create `scripts` folder inside assets.

↳ App.js.

4) alert("Hello")

// How to actually use webpack? → Command Line / View > Terminal (control + `)

5) npm install webpack webpack-cli --save-dev
↳ command line interface

& ✓ have webpack in our project

Q: How to use / leverage webpack?

↳ Create a configuration file within folder root

6) At root folder, touch webpack.config.js.

& webpack will know & find for this file

7) In webpack.config.js

↳ {} ⇝ JS object

↳ entry: ie⇝ path that points towards our JS file (App.js) that we want to bundle

↳ ./app/assets/scripts/App.js

ie⇝ which webpack watch, process & bundle

Q: How to tell webpack to use App.js as instructions & actually run, process & bundle files.

8) before {} ⇝ module.exports = ie⇝ when webpack loads this file it knows {} is what to be exported & used

9) In package.json

~~After~~ Look for scripts, or after "version" ...;

"scripts": {},
↳ to create many different NPM scripts
a command that able to be run from CL

{
10) "dev": "webpack", → we don't want to install webpack globally
↑ anyname

11) ↳ (CL) npm run dev → a folder `dist` ─ main.js
↳ Webpack look at our App.js file, process & bundle it & create this new main.js file

Q: ~~What if we~~ Can it be called as other name / locate in 'app' folder instead of main.js inside a new dist folder?

12) ↳ In webpack.config.js
└ entry: 'app',
output: {}
↳ filename: '↳ bundled.js → name.
path: path.resolve(__dirname, 'app') → set path to app folder
↳ webpack requires absolute path for the output
Node.js path package

13) └ At top:
const path = require('path') → part of the node library
↳ At (CL) got a WARNING → didn't set mode of the web pack.

14) after output: {},
mode: 'development' → we're still developing things, not ready to deliver them live to public yet.

15) delete 'dist' folder

16) npm run dev → app ↳ bundled.js

17) In index.html
before </body>
↳ <script src="bundled.js"></script>

18) edit App.js → npm run dev → refresh index.html

Q: Very inconvenient?

Ans) In webpack.config.js
19) under mode: 'development',
watch: true

20) npm run dev → webpack will stay running & catch & detect any time we save a change to our source file/entry file

21) ctrl + c / command + . to stop

Not impressive? Because our App.js file is so simple. If we do sth complex, webpack would be a lifesaver, because every time we save, we have an automated workflow tool that magically makes everything happen in a matter of milliseconds.

* read 14)

15) CSS With Webpack
↳ use webpack to setup an automated CSS workflow
↳ CSS workflow? What we're trying to setup & why? Add CSS support to webpack configuration

↳ Autoprefixer
& automation tool
column: w → -webkit-columns: w;
-moz-columns: w;
columns: w;
& lot others that we can't do ∵ browsers didn't support it
& eg sass, Less, Stylus

eg. CSS Variables ~~to re~~
  eg. $mainblue : #2f5572 ; → to reuse / modify that color

Nested CSS (0% web browser supports).

eg:
```
.feature-box a {
  }
}
.feature-box b {
  }
}
```
→ C (Sass, Less, Stylus) →
```
.feature-box {
  a {
  }
}
  b {
  }
}
```

PostCSS : - Trendy, popular post-processor
  └ Fastest to compile out of all pre/post-processors.

1) In app/assets
  └ styles
    └ styles.css
      ↳ body {
          color : orange
        }

Q: linking this CSS in <head> ?
  ↳ this setup won't leave any room for any automated workflow.
We gonna load this CSS file from within our main JS file

2) In App.js
import '../styles/styles.css'
* if npm run dev → error ∵ without config, webpack only understands JS but not CSS

3) (CLI) npm install css-loader style-loader --save-dev

4) In webpack.config.js (configuration)
  under watch: 'true',
+ → module : {}
    ↳ rules : [ ]
      ↳ {}
        ↳ test : /\.css$/i, ← regex
          use : [ ] → 'css-loader'
                         ↳ to understand & process CSS files

5) npm run dev

⇒ Successfully bundled css into bundled.js, but not actually be used by browser.

↳ To fix: In webpack.config.js.

6) ↳ rules [ ]
↳ { }
↳ use: ['style-loader', 'css-loader']
↳ let webpack understand & bundle css files
↳ Applies the CSS in the browser itself.

7) ctrl+c / command+. > npm run dev ⇒ refresh ✓

8) del alert in App.js ✓

Q: Why are we loading CSS from within our JS?
↳ Don't worry. When it comes time to actually serve our website to public, we'll set things up, so that our CSS lives in its own traditional seperate CSS file that's included in <head>.

↳ During development, there are speed & performance advantages to having CSS handled by JS.
↳ We'll setup webpack dev. server ⇒ x refresh, just save.

9) ctrl+c to stop. > npm install postcss-loader --save-dev

10) In webpack.config.js
↳ rules: [ ]
↳ { }
↳ use: [... { }] ⇒ to give few options / plug ins
↳ loader: 'postcss-loader', options: { }
↳ plugins: postCSSPlugins
↳ list all postCSS features we want

11) + const postCSSPlugins = [ ]

12) (CL) npm install postcss-simple-vars postcss-nested autoprefixer --save-dev

13) In (11) ↳ require('postcss-simple-vars'), & save
require('postcss-nested'),
require('autoprefixer')

(4) In styles.css , test these postCSS features.

~~└ Vars~~

a) Variables :

```
$mainBlue : #2fb5d2;

.large-hero
```

b) nested CSS :

```
large-hero {
  h2 {
    color : $mainBlue ;
  }
}
```

(5) save & > npm run dev ✓ successfully configured postCSS plugins. ✓.

* read 16) So far . git , Node, NPM , Webpack.

---

(7) CSS File Architecture

└ Keys to success : 1) Good Attitude 2) Create amazing websites

└ CSS Organization

└ 1) File Architecture : organize CSS code into multiple, small, specific-purpose files.

2) Identify patterns in Design :

3) Rules to follow for creating class names & writing our CSS selectors

1) ~~open~~ travel-site > index.html

2) styles.css , del.

```
└ body {
    font-family : 'Roboto' , sans-serif ;
    color : #333
  }
```

3) npm run dev → start webpack, it will watch & recreate bundled.js for every save

4) In styles.css

```
img {
  max-width: 100%;
  height: auto;
}
```

~~large-hero {~~ Imports come into play !!

5) travel-site > app > assets > styles > modules

└ _large-hero.css

↳ underscore = partial file (import)

In _large-hero.css :

```
.large-hero {
  position: relative;
}
```

In styles.css :

```
@import 'modules/_large-hero';
```

↳ Http : A native CSS feature, but we don't want the web browser to download multiple CSS files. Instead, we want webpack & postCSS to see this import like & replace this line with the contents

6) ctrl+c ; npm install postcss-import --save-dev

7) In webpack.config.js,

```
const postCSSPlugins = [
  require('postcss-import'),
];
```

In _large-hero.css :

```
a {
  color: purple; //testing
}
```

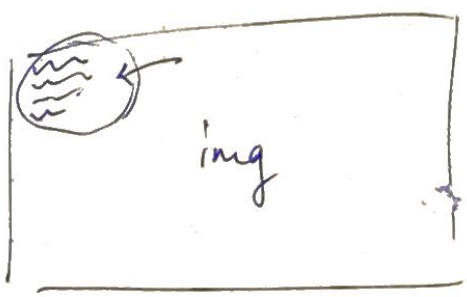
(8) npm run dev ✓

del color: purple. ✓

8) move body {} & img {} & paste in
styles > base > _global.css
& @import "base/_global"; in styles.css
& main css file doesn't contain any CSS itself, instead like a recipe pointing towards other ingredients with own specified purpose each.
& Keep our code super organized

9) @import 'normalize.css'
& an alternative to CSS resets. It adjusts the styles of certain elements in an effort to make all browsers render things more consistently.
& we it on all of ur projects.

10) ~~In large-hero.css~~
enclose <h1> & <p> into
* .large-hero__text-content {
position: absolute;
top: 0
left: 0
}



width: 100%
+ text-align: center

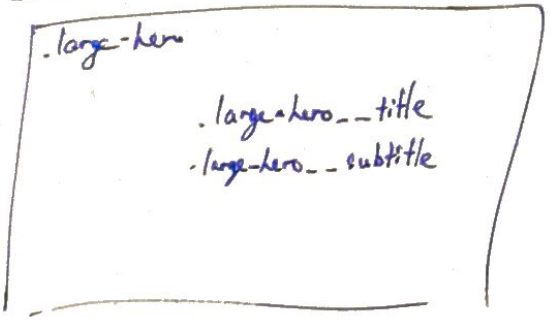
+ top: 50%;
transform: translateY(-50%)

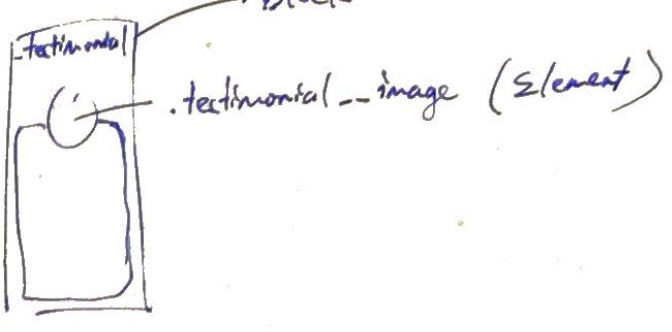
Q: why not background-image instead of <img>?
& ~~less~~ responsive issue

18) BEM ?

- styling <h1> ; How?
  a. .large-hero h1 {
       font-weight: 300,
     }
  ↳ In recent years, there has been a move away using type & descendant selectors.
  ↳ Instead, moving towards a more systematic way of targeting elements.
    ↳ A popular system/methodologies for selectors : BEM
  ↳ ~~give~~ <h1 class="large-hero__title">
    ↳ .large--hero__title {...}
- ↳ Why? We were misguided in the past about what it means to write semantic HTML.
  ↳ __ ? meaning of it?
- ↳ BEM :
  - an abstract way of thinking our design / interface
  B : Block - an independent, reusable part of our design
  E : Element - belongs to a block. It can't be used outside of the block that it belongs to.
  M : Modifier - can be used on a block or an element to indicate a change to the default state of an object.

eg Element





eg. <div class="mobile-menu mobile-menu--is-open" ..>

Modifier :

↳ modifier class, toggled by JS when tap/click

Modifier

Eg.

[btn1] : <a class = "btn" .. > default state of button.

[btn2] : <a class = "btn btn--orange" .. >

↳ modifier class

[btn 3] : <a class = "btn btn--orange btn--large" .. >

BEM Overview :

- CSS selectors should target elements directly with classes, instead of relying on type selectors, descendent selectors, and the cascade.
  ↳ We are somewhat eliminating 'C' in CSS.
- Because we are limiting cascade we are free to move blocks around & reuse them throughout the page.
  ↳ we are creating 1-on-1 relationship between a block of HTML & its CSS
  ↳ we don't have to worry about any parent elements trickling their styles down to our blocks & hurting the predictability of our block styles.
  ↳ Our blocks are independent, self-contained & we are free to move them around & reuse them throughout the page.
  ↳ Blocks can be nested inside other blocks.
  ↳ Identify patterns, & then create single-responsibility blocks.
  To
  eg. columns, are patterns identified in our design.
  ↳ There's no reason to code a pattern more than once
  └ Create a block for creating col. layouts, & use it to create 3 empty cols, & with each col. block I'd nest a testimonial block.
  ↳ Makes the relationship between our HTML & CSS clear.

Q. What about the semantics of our HTML? Is adding all these extra classes with presentational names making our HTML less semantic?

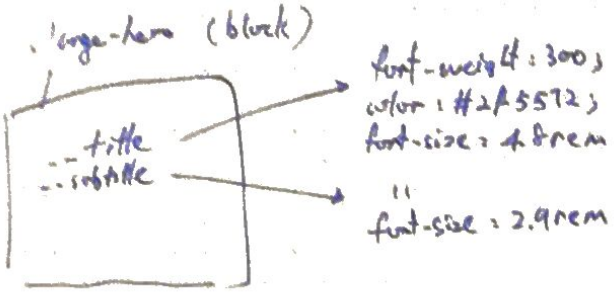
↳ No. - By Nicholas Gallagher, creator of normalize.css

└ " class names cannot be unsemantic."

└ "Content-layer semantics are already served by HTML elements "

└ "Class names impart little or no useful semantic info to machines or human visitors"

"The primary purpose of a class name is to be a hook for CSS and JavaScript."
"Class names should communicate useful info to developers"
— Nicholas Gallagher



* Avoid descendants selectors: they often create specificity wars
* If we instead select elements with direct class selectors, we can maintain a very flat CSS architecture or a flat specificity tree.

How can we write nested CSS to help us stay organized conceptually without having webpack compile things down to a descendant selectors?

`&` eg
```
.large-hero {
  &__title {
  }
}
```
webpack → `.large-hero__title {}`

∴ our main CSS file imports in different single responsibility CSS files, and within app > styles > modules folder, we have a new file for each block of the design.

Within a block file:
```
.large-hero {          → base block class
  &__title {}          → Nested BEM element
}
```

19) Complete 2 blocks. : .large-hero block & .btn block

```
.large-hero {
  &__title {
    margin: 0;
  }
  &__subtitle {
    margin: 0
  }
  &__description {
    color: #FFF
    font-size: 1.875; (30px)
    font-weight: 400;
    text-shadow: 2px 2px 0 rgba(0,0,0,0.1);
```

* Why rem? Not everyone in the world has their web browser configured exactly the same. To honor the user font preferences. All spaces will be scaled accordingly to the user's font size preferences.

* Whenever we declare a font-size, padding, margin value, do so in terms of rem, instead of px or em.
* rem: everything is relative to the root of the page — <html> (default font-size of html = 16px)

Q: How can we find out the exact value, & eliminate guesswork?
↳ Told by designer lol.

```
&__description {
  max-width: 30rem; // 480 px
  margin-left: auto;
  margin-right: auto;
}
```
} margin: 0 auto;

app > assets > styles > modules > _btn.css

```
.btn {
  background-color: #2f5572;
  color: #FFF;
```

@import "modules/_btn";

- Create variables that can be reused in all blocks.
  color
  ~~$mainBlue: #2f5572~~

app > assets > styles > base > _variables.css

$mainBlue: #2f5572;

@import "base/_variables";

& update all colors.

```
.btn {
  ~~back~~
  text-decoration: none; // remove underline
  padding: 0.25rem 1.2rem;
  display: inline-block; // : its parent & surrounding elements will be aware of its
                            vertical padding.
```

* Error: saved _btn.css before creating _variables.css.
restart npm run dev.

```
.btn {
  &__orange {
    background-color: $mainOrange
  &__large {
    fontsize: 1.25rem
    padding: 1.1rem 1.9rem
}
```

or $mainOrange: #d59541;

20) Webpack Timeout: Configure webpack to automatically refresh the browser for us when save on html. ~~Add~~ Additionally, webpack won't need to refresh the browser, just auto-inject the new CSS code into the browser's memory.
↳ + productivity & speed boost.

Webpack dev Server
1. CSS & JS updates in the browser without a full reload.
2. Reloads browser on ≤HTML.
3. View site on any device connected to the same W.Fi as the computer.
~~use~~ - weekly 5m dl

1) npm install webpack-dev-server --save-dev

2) In webpack.config.js:

```
module.exports = {}
  └ output: {},
    devServer: {},
      └ contentBase: path.join(__dirname, 'app'),   // To point the folder that we want webpack to serve up (where index.html is)
        hot: true,                                  // allow webpack to inject new CSS & JS into the browser's memory without reload. - "hot module replacement"
        port: 3000
        80.80 by default
```

3) ~~let watch: true~~

4) In package.json:

```
"scripts": {
  "dev": "webpack-dev-server",
}
```

5) In app > assets > scripts > App.js

```
if (module.hot) {}
  └ module.hot.accept()
```

6) npm run dev.; /localhost:3000

* ⌘/ctrl+R = Full-page reload; very expensive in terms of performance & speed. It forces the whole browser to completely reload & rerender & re-paint the entire page.
* To check, highlight a text, disappear = ~~Auto-fast~~ Refresh.
* eg. Able to change menu without refresh & keep re-open it

1) Reload on .html: (full page reload)

In webpack.config.js:

```
devServer: {}
  └ before: function() {},   → app, server
      └ server._watch('./app/**/*.html')
```

restart npm run dev

2) Other device

In webpack.config.js:

```
devServer: {}
  └ port: 3000,
  → host: '0.0.0.0'
```

allow devices on the same network to be able to access the webpack dev server from this computer.

↳ get local IP address: 192.168.3.58

Mac: System preferences > Network → IP address (your local IP)

windows: ipconfig (on CL)
↳ look for IPv4 address.

On phone, visit 192.168.3.58:3000
↳ boost ur productivity

→ del bundled.js → still works!: webpack dev server does not actually output the bundled file to our file disk, it just keeps the bundled file in RAM/memory so it's super fast.
↳ The dev server makes the file available from our address, file can be accessed from localhost:3000/bundled.js but it doesn't actually exist on our hard drive
✓ Optimal workflow for web dev.