Evolution of the "Web App"

@HenrikJoreteg
@Hoarse_JS
THIS USED TO BE SIMPLE!
1. WRITE SOME HTML
2. LAY IT OUT WITH FRAMES OR TABLES
3. FTP IT TO A SERVER!
4. BAM!
CONGRATULATIONS,
YOU’RE A WEB DEVELOPER!
THEN IT GOT HARDER
WHO’S WRITTEN THEIR OWN BLOG SOFTWARE?
OVER ENGINEERING A BLOG WAS A RITE OF PASSAGE
1. **WRITE SOME PHP/PYTHON/ASP/COLDFUSION**
2. **SET UP RELATIONAL DATABASE**
3. **WRITE SOME BAD SQL**
4. **SHOVE DYNAMIC DATA INTO OUR HTML**
5. **LAY IT OUT WITH CSS (NO TABLES THIS TIME)**
6. **RUN IT ON SHARED HOSTING SOMEWHERE**
CONGRATULATIONS, YOU’RE A WEB DEVELOPER!
PHEW!!
THEN WE GOT "SMART"
USE A FRAMEWORK!!
1. RAILS
2. ALL THEM PHP FRAMEWORKS
3. DJANGO
4. ETC.
OUR EXCESSIVELY DYNAMIC BLOG... IS NOW BETTER ORGANIZED AND HAS MORE DEPENDENCIES
CONGRATULATIONS,
YOU’RE A WEB DEVELOPER!
WHAT ABOUT TODAY?
CONGRATULATIONS
YOU MIGHT BE A
"WEB DEVELOPER"
WHAT DOES "WEB DEVELOPER" EVEN MEAN?
WHAT DOES "WEB APP" EVEN MEAN?
THE BROWSER ISN’T OUR RENDERER
THE BROWSER IS OUR RUNTIME
THE WEBVIEW IS OUR RUNTIME
MORE LOGIC MOVING TO FRONT-END JS
DEVs WITH DIFFERENT BACKGROUNDs CONVERGING ON JS
BRINGING THEIR PATTERNS AND PREFERENCES
BACK-END FRONT-END ISOMORPHIC RESPONSIVE ES6(2015) BABEL TRACEUR AMD COMMONJS MVC MVVM FLUX RELAY GULP GRUNT API-GATEWAY CORS JSON-WEB-TOKENS NODE.JS HTTP 2.0 OFFLINE-FIRST MOBILE-FIRST WEBGL WEBRTC WEBSOCKET GRAPHQL AMPERSAND EMBER REALTIME REDIS RIAK LEVELDB RABBITMQ PUSH-NOTIFICATION ENABLED BACKBONE APP WITH POLYFILLED POLYMER WEB COMPONENTS
SINGLE PAGE APPS
SINGLE PAGE APPS
"NATIVE"
I’M A WEB DEVELOPER
NATIVE WEB APP
1. JAVASCRIPT
2. HTML
3. CSS
4. BROWSER APIS
FUNDAMENTAL DISTINCTION...
THE BROWSER IS YOUR RUNTIME
Send the app itself to the browser instead of the result of running it.
<!doctype>
<script src="app.1.3.7.js"></script>
SHOULD WE EVEN BE DOING THIS?
SHOULD WE BUILD APPS THAT REQUIRE JAVASCRIPT?
Improving performance on twitter.com

Tuesday, May 29, 2012 | By Twitter (@twitter) [21:23 UTC]

To connect you to information in real time, it's important for Twitter to be fast. That's why we've been reviewing our entire technology stack to optimize for speed.

When we shipped #NewTwitter in September 2010, we built it around a web application architecture that pushed all of the UI rendering and logic to JavaScript running on our users' browsers and consumed the Twitter REST API directly, in a similar way to our mobile clients. That architecture broke new ground by offering a number of advantages over a more traditional approach, but it lacked support for various optimizations available only on the server.

To improve the twitter.com experience for everyone, we've been working to take back control of our front-end performance by moving the rendering to the server. This has allowed us to drop our initial page load times to 1/5th of what they were previously and reduce differences in performance across browsers.

On top of the rendered pages, we asynchronously bootstrap a new modular JavaScript application to provide the fully-featured interactive experience our users expect. This
SHOULD WE BUILD APPS THAT REQUIRE JAVASCRIPT?
YES!
WHAT SERVICE ARE WE PROVIDING?
CONTENT?
Emergency rule in Burkina Faso

Burkina Faso's military announces emergency measures after violent protests against President Compaoré's bid to extend his 27-year rule.

- President sees MPs 'homes on fire'
- As it happened: Burkina Faso unrest
- Protesters invade parliament
- Compaoré: 'What I'm doing is legal'

US police killing suspect captured

A US man who has eluded police in the forests of Pennsylvania for six weeks after the fatal shooting of a police officer has been found.
NO REASON TO COMPLICATE THINGS THAT CAN BE SIMPLE
But the web is no longer just about linked content!
There are cases where clientside functionality is the core value provided by service.
HIGH PERFORMANCE

1. RENDERING
2. NETWORKING
3. FILE READ/WRITE
4. STORAGE
5. WEB AUDIO APIS
6. WEBGL
7. VOICE/VIDEO
BROWSERS ARE NOT DUMB DOCUMENT VIEWERS
MOST CAPABLE
UBIQUITOUS
RUNTIMES
ON THE PLANET
I’M JUST GOING TO SAY IT:
THERE ARE TWO TYPES OF APPLICATIONS ON THE WEB
1. NATIVE WEB APPS
2. SERVER-SIDE WEB APPS
THEY ARE
FUNDAMENTALLY
DIFFERENT
AND THAT’S O.K.
Anything we **can** build with web tech, I think we **should**.
EVEN IF WE CAN'T SUPPORT OLDER BROWSERS
THE WEB IS INFINITELY MORE OPEN THAN NATIVE PLATFORMS
USER EXPECTATIONS HAVE EVOLVED
THE WEB IS DOING PRETTY WELL ON DESKTOPS
THE WEB IS LOSING ON MOBILE
THE WEB IS LOSING ON EXPERIENCE
WE OFTEN PREFER NATIVE APPS TO THE WEB
QUALITY AND POLISH OF USER EXPERIENCE IS OFTEN MUCH BETTER
LET’S FIX THIS!
WE’RE TOO FOCUSED ON THE PAST INSTEAD OF COMPETING ON EXPERIENCE
Saying there’s a distinction makes some people mad.
“Everything should be an enhancement!”
WE’RE ON THE SAME TEAM!
WE WANT THE OPEN WEB TO WIN!
HOW COULD WE EVEN BUILD A PROGRESSIVELY ENHANCED VERSION OF TALKY?
SHOULD WE NOT HAVE BUILT IT?
WHERE’S THE DOWNSIDE?
LET’S LOOK A BIT CLOSER AT TWITTER
WHAT IS TWITTER?
IS IT A WEB APP?
NO.
IT’S A SERVICE
APP != SERVICE
I DIDN’T REALLY CARE HOW THEY BUILT THEIR WEBAPP
BECAUSE I DIDN’T USE IT ANYWAY!
I WAS USING AN iOS APP!
THEIR WEB APP HAD ALREADY FAILED ME!
LET’S THINK ABOUT THIS
WHEN I FOLLOW A LINK TO A RANDOM TWEET ON MY PHONE...
JUST LET ME READ IT!
I DON’T MIND IF IT’S plain text
DON'T MAKE ME DOWNLOAD 2MB OF JS TO READ 140 CHARACTERS OF TEXT!
THIS IS THE PROBLEM THEY FIXED WITH NEW NEW NEW TWITTER
BUT...
CATCHING UP WITH 
ALL THINGS TWITTER 
IS A FUNDAMENTALLY 
DIFFERENT USE CASE
Failing to recognize distinction makes us flounder.
A SERVICE CAN PROVIDE BOTH!
TWITTER.COM
&
TWEETDECK.COM
THERE’S STILL SOME GAPS BETWEEN WEB AND NATIVE
REAL OFFLINE SUPPORT
PLATFORMS THAT TREAT NATIVE WEB APPS AS FIRST CLASS CITIZENS
THOSE THINGS ARE CHANGING
1. SERVICE WORKER
1. SERVICE WORKER PROGRAMMABLE CACHE LAYER CAN INTERCEPTS ALL NETWORK REQUESTS
THIS IS HUGE!
2. INSTALLABLE WEB APPS
JSON-BASED WEB MANIFEST

CHROME M39+

FIREFOX

https://developer.chrome.com/multidevice/android/installtohomescreen

https://w3c.github.io/manifest/
1. SIGNAL INTENT
2. UNINSTALL
3. DEEPER DEVICE APIS
"What about performance?"
WHITE PAGE
OF DEATH
TIME TO FIRST PAINT
A PRIMED CACHE LARGELY INVALIDATES THIS ARGUMENT
1. GIVE IT A UNIQUE NAME

<!doctype>
<script src="app-1.2.7.js"></script>

2. CACHE IT FOREVER

HTTP/1.1 200 OK
Cache-Control: max-age=REALLY BIG NUMBER!
Content-Encoding: gzip
MOST OF THESE TYPES OF APPS REQUIRE YOU TO BE LOGGED IN
PRE-FETCH APP ON PUBLIC PAGES OR LOGIN PAGE
NATIVE WEB APPS CAN STILL HAVE SMALL JS PAYLOADS!
ALL JS IN THE AMPERSAND.JS APP ON TODOMVC.COM COMBINED 28kb min + gzip
SMALLER THAN JQUERY 2.0
THE OTHER ASPECT OF PERFORMANCE...
ONCE LOADED, PERFORMANCE IS WAY BETTER!
IF I’M GOING TO LEAVE APP OPEN ON MY DESKTOP I CARE WAY LESS ABOUT LOAD TIME
"What about dual rendered a.k.a. isomorphic apps?"
JUST A CLIENTSIDE APP WITH AN OPTIMIZED INITIAL RENDER
GOING FULL-ISOMORPHIC
RENDERING, WITH USER DATA
AND ALL...
OFTEN REQUIRES DRAMATICALLY MORE COMPLEX CODE
THERE ARE SOME CASES WHERE IT MAKES SENSE
WITH STATE OF TOOLING TODAY OFTEN NOT WORTH THE COMPLEXITY
HOWEVER...
DOESN’T HAVE TO BE ALL OR NOTHING
WE CAN PRE-RENDER EVERYTHING
THAT’S NOT USER-SPECIFIC DATA
WE CAN DO THIS AS A FULLY STATIC SITE
<table>
<thead>
<tr>
<th>Route</th>
<th>Asset</th>
</tr>
</thead>
<tbody>
<tr>
<td>site.com/pic.png</td>
<td>-&gt; pic.png</td>
</tr>
<tr>
<td>site.com</td>
<td>-&gt; index.html</td>
</tr>
<tr>
<td>site.com/page</td>
<td>-&gt; page.html</td>
</tr>
<tr>
<td>site.com/asdf</td>
<td>-&gt; 404.html</td>
</tr>
<tr>
<td></td>
<td>or 200.html</td>
</tr>
</tbody>
</table>
WOAH!
APPS USUALLY HAVE:
1. public/marketing pages
2. all the stuff behind the login
WRITE “PUBLIC” PAGES AND APP LAYOUT HTML AS ISOMORPHIC COMPONENTS
PRE-RENDER THEM TO STATIC PAGES AT BUILD TIME!
SLIP IN THE BUILT JS BEFORE:
</body>
index.html: public home page
200.html: application layout
COULD POTENTIALLY EVEN DO THIS FOR DYNAMIC/PUBLIC DATA
THINK ABOUT WHAT WE GET
pixels on the screen immediately
TOTALLY CRAWLABLE (SEO)
JS TAKES OVER ROUTING WHEN LOADED
DEPLOYMENT AND OPS BECOME AS SIMPLE AS FTP
WRITE 1 VERSION OF YOUR APP
GET 90% OF BENEFIT FROM
ISOMORPHIC RENDERING
Users will end up with a primed cache just by visiting your marketing pages.
READY FOR:
PHONEGAP/CORDOVA
DESKTOP APP
NOW WE HAVE AN APP WITH A SINGULAR CONCERN: PRESENTATION
I’VE STARTING BUILDING ALL MY APPS AS STATIC NATIVE WEB APPS
TOTALLY <3 IT!
FOR SO LONG THE TREND HAS BEEN TOWARD COMPLEXITY
If you don’t actively fight for simplicity in software, complexity will win.

…and it will suck.
WHAT’S THE NEXT STEP IN THE EVOLUTION OF THE "WEB APP"?
GOING BACK TO SIMPLE
GOING BACK TO THE STATIC WEB
STATIC NATIVE WEB APPS
POWERED BY SERVICES
SOME WHICH WE BUILD
MANY OF WHICH WE RENT
surge.sh
hood.ie
firebase.com
auth0.com
divshot.com
SIMPLE OPEN SOURCE EXAMPLE: HubTags.com
• React
• Ampersand.js
• Webpack (hjs-webpack)
• GitHub API
• Surge.sh
HOW CAN WE BE SURE WE’RE BUILDING WITH THE RIGHT TOOLS?!
WE CAN’T!
WHAT DO WE KNOW?
THINGS WILL CHANGE
BUILD MODULAR SYSTEMS THAT STRIVE TO BE AS SIMPLE AS THEY CAN BE
OFFLOADING PRESENTATION
STATIC NATIVE WEB APPS
BUILDING MICROSERVICES TO ENABLE THAT TYPE OF APP
OPTIMIZE FOR **CHANGE**.
IT IS THE ONLY CONSTANT.
LET’S KEEP PUSHING FOR SIMPLICITY
LET’S BUILD FOR THE FUTURE OF THE WEB, NOT ITS PAST
THANKS!

@HenrikJoreteg, andyet.com