HTML5

Mobile Local

due by Wed 2/15, noon ET

Ingredients.

- HTML5
- JavaScript
- JSON
- JSONP
- `localStorage`
- `navigator.geolocation`
- RSS
- XML
- YQL

Help.

Help is available throughout the week at `http://help.cs76.net/`. We'll do our best to respond within 24 hours!

Academic Honesty

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed by some project. Viewing, requesting, or copying another individual's work or lifting material from a book, magazine, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student.

Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this course that you have submitted or will submit to another. Nor may you provide or make available your or other students' solutions to projects to individuals who take or may take this course (or CSCI S-76) in the future.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. You may also turn to the Web for instruction beyond the course's lectures and sections, for references, and for solutions to technical difficulties, but not for outright solutions to problems on projects. However, failure to cite (as with comments) the origin of any code or technique that you do discover outside of the course's lectures and sections (even while respecting these constraints) and then integrate into your own work may be considered academic dishonesty.

If in doubt as to the appropriateness of some discussion or action, contact the staff.

All forms of academic dishonesty are dealt with harshly. If the course refers some matter to the Administrative Board and the outcome for some student is disciplinary action, the course reserves the right to impose local sanctions on top of that outcome for that student that may include, but not be limited to, a failing grade for work submitted or for the course itself.

Grades.

Your work on this project will be evaluated along four primary axes.

Correctness. To what extent is your code consistent with our specifications and free of bugs?
Design. To what extent is your code written well (*i.e.*, clearly, efficiently, elegantly, and/or logically)?
Scope. To what extent does your code implement the features required by our specification?
Style. To what extent is your code readable (*i.e.*, commented and indented with variables aptly named)?

Getting Started.

☐ Let's first dive into HTML5 by having you curl up with a free book. Spend some time with `http://diveintohtml5.info/`. You should find that it's an easy read and provides a nice overview of HTML5. Rest assured that you won't need to leverage all of HTML5's features for this or future projects, so it's probably okay to skim any sections that don't really interest you!

☐ Next curl up with another free book, this one at `http://ofps.oreilly.com/titles/9780596805784/`. (Agreed, the first book had a cooler URL.) This book focuses rather specifically on building web apps for iPhones, but you should find that its lessons apply to most any Webkit-based browser. This book, too, is worth reading (or skimming!) in its entirety, but, at a minimum, read chapters 1 through 5.

☐ If new to JavaScript, you might also like to read over `https://developer.mozilla.org/en/JavaScript/Guide`.

☐ For this project and others, it's ideal to install on your desktop or laptop the latest versions of:

- ☐ Google Chrome from `http://www.google.com/chrome`, if you run Linux, Mac OS, or Windows.
- ☐ Safari from `http://www.apple.com/safari/`, if you run Mac OS or Windows. Once installed, select **Safari > Preferences…** (in Mac OS) or **Edit > Preferences…** (in Windows), then click **Advanced**, check **Show Develop menu in menu bar**, then close the **Advanced** window.
- ☐ Xcode 4.2.1 from `http://itunes.apple.com/us/app/xcode/id448457090?mt=12`, the latter of which comes with iOS Simulator, if you run Mac OS.

No need to install the Android SDK (and emulator) just yet, as it's a bit more involved. And no worries if you don't yet have (access to) a Mac. Installing at least one of Google Chrome and Safari suffices for this project.

Be sure, though, that your project indeed renders and behaves properly in a Webkit-based browser before you submit.

☐ For this project, it suffices to develop your web app locally (on your own hard drive), viewing and testing it locally as well (whether with a browser, emulator, or simulator). However, you (and your friends!) might enjoy accessing your work on an actual phone, in which case you need to make it available on your LAN or the Internet at large via HTTP. To do so, you have a few options:

- ☐ If you already have (access to) a webserver, you're welcome to upload your app (as with SFTP) to that webserver in order to try out your app on an actual phone. Odds are you already know how to do so!
- ☐ If you would like to host your app on one of Harvard's webservers, create an FAS account (if you haven't one already) at `https://idm.fas.harvard.edu/new`. You'll need to know your Harvard ID (HUID) at PIN, the same credentials that you use to log into the course's own website. Once you have an FAS account (*i.e.*, username and password), you can SSH or SFTP to `nice.fas.harvard.edu`, create a directory called `public_html` in your home

directory, `chmod` it and your home directory `711`, upload your files to `public_html`, `chmod` all of them `644`, then visit your app at `http://www.people.fas.harvard.edu/~username/`, where `username` is your own username.

☐ If you would like run a webserver on your own computer, making your app accessible via HTTP on your own LAN (but not the Internet at large), install version 3 of the CS50 Appliance, per `https://manual.cs50.net/Appliance#How_to_Install_Appliance`, and then activate `eth2`, per `https://manual.cs50.net/Appliance#How_to_Access_Appliance_from_Another_Computer`. SSH or SFTP to `w.x.y.z`, where `w.x.y.z` is the IP address bound to `eth2`, create a directory called `public_html` in John Harvard's home directory, `chmod` it and that home directory `711`, upload your files to `public_html`, `chmod` all of them `644`, then visit your app at `http://w.x.y.z/~jharvard/`. If your phone supports Wi-Fi and has an IP address on your LAN, you should be able to access your app on your phone via that same URL.

RSS.

☐ If unfamiliar with XML, head to `http://en.wikipedia.org/wiki/XML` to read up on the subject!

☐ If unfamiliar with RSS, head to `http://en.wikipedia.org/wiki/RSS` to read up on that subject as well!

☐ It turns out that many sites syndicate news via RSS, including Google News. In fact, head to `http://news.google.com/news/search?geo=02138` and you should the day's news for Cambridge, MA. Scroll down to that page's bottom, and you should see an **RSS** link. Click it, and you should find yourself at a URL like the below (though, depending on your browser, yours might begin with `feed://` instead of `http://`).

```
http://news.google.com/news?pz=1&cf=all&ned=us&hl=en&geo=02138&output=rss
```

Notice, again, that the zip code you typed appears in this URL, per the boldfacing above. The implication is that we can request news for any city we'd like by constructing URLs like these. In the interests of clarity, we can simplify this URL also to just the below.

```
http://news.google.com/news?geo=02138&output=rss
```

Realize that you don't have to provide zip codes. You can instead refer to geographies by name, as in the below.

```
http://news.google.com/news/search?geo=Cambridge%2C+MA
```

as before. Notice, though, that if your input contains punctuation or whitespace, it should be URL-encoded, per `http://en.wikipedia.org/wiki/Percent-encoding`.

Neat, eh? Keep this trick in mind as we proceed.

YQL.

☐ If unfamiliar with browsers' same-origin policies, head to `http://en.wikipedia.org/wiki/Same_origin_policy` to read up on those. If unfamiliar with JSON or JSONP, head to `http://en.wikipedia.org/wiki/JSON` to read up on those subjects as well. In a nutshell, JSONP enables you to integrate data from one domain into a DOM whose HTML was served up by another domain.

☐ If unfamiliar with Yahoo! Query Language (YQL), head to `http://developer.yahoo.com/yql/` to read up on that subject too. In a nutshell, YQL allows you to retrieve data from Yahoo and webservers more generally in machine-readable formats, namely XML and JSON. In fact, head to YQL's console at `http://developer.yahoo.com/yql/console/`, replace

```
show tables
```

under **YOUR YQL STATEMENT** with

```
select * from rss where url = 'http://news.google.com/news?geo=02138&output=rss'
```

and then click **TEST**. After retrieving that URL, Yahoo will present the items within in its own XML format. Hm, that doesn't feel like much progress, since RSS is already XML. Let's do one better: this time around, select **JSON** instead of **XML** (the default) to the left of **TEST**, then click **TEST** once more. Yahoo will now present that same news as JSON, padded by a callback function called `cbfunc` by default. In other words, it appears that you can convert RSS to JSONP via this YQL service. In fact, you don't need the console at all. Notice that, beneath **THE REST QUERY**, is a long URL like the below.

```
http://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20rss%20where%20url%20%3D%20'http%3A%2F%2Fnews.google.com%2Fnews%3Fgeo%3D02138%26output%3Drss'&format=json&callback=cbfunc
```

Encoded within that URL is the query you typed. If you visit that URL in a browser, you should see the raw JSONP, without the console's GUI around it. Bit of a mess, eh? That's because, to save bytes, the service only pretty-prints its output with indentation and whitespace when you're using the console.

Interesting, eh?

Geocoding.

☐ Whereas *geolocating* involves finding a real-world object's geographic coordinates, *geocoding* involves converting a string (*e.g.*, a zip code or city and state) to geographic coordinates. Exactly where in the world is a place like Cambridge, MA? The Google Geocoding API knows! If unfamiliar, read up on the service at `http://code.google.com/apis/maps/documentation/geocoding/`. Notice that the API can return answers to you as XML or JSON (though not JSONP). And the API

can also reverse-geocode, in case you have some coordinates and want to find out the corresponding zip code or city and state.

Also interesting, eh? So many interesting (and free!) services out there…

Specification.

☐ Your mission for this project is to implement Mobile Local, a mobile web app with which users can check local news and weather. The overall design and aesthetics of this app are ultimately up to you, but we require that your app meet some requirements. All other details are left to your own creativity and interpretation.

Features.

- ☐ Your app's UI should be designed for a smartphone whose width is defined by `device-width`; its actual resolution might be anywhere from 320×480 to 760×1280.
- ☐ Your app must present users with an HTML form into which they can input a zip code or a city and state. Upon submitting that form, users should be shown a weather forecast for that locale as well as a list of links to recent news articles about that locale. For each link, it suffices to display the article's title, though you may also include its description or a portion thereof. Exactly how many links to show is up to you to decide.
- ☐ Your app must provide users with a button or link via which they can provide a locale via geolocation so that they can see weather and news for their current location without typing anything.
- ☐ Your app must remind users of their most recent searches, if any, by way of a list of clickable buttons or links so that they can re-check some locale's weather and news with a single touch. Exactly how many searches to show is up to you to decide.

Implementation Details.

- ☐ The entry point to your app should be a file called `index.html`. Code that you write may live within that file or external JavaScript and/or CSS files.
- ☐ You must use Google News for your news.
- ☐ You must use YQL to convert RSS from Google News to JSONP.
- ☐ You must use the Google Geocoding API as needed to convert geographic coordinates to zip codes or cities and states.
- ☐ We leave it to you to find a weather service whose data you can integrate into your app. You are welcome to discuss options with classmates.
- ☐ You may not embed weather and news in your app via iframes: you must integrate services' data into your app's own DOM.
- ☐ Your app must depend only on third-party services, not on any server-side scripts of your own. Any code that you yourself write must be client-side JavaScript. Accordingly, you must intercept submissions of your app's form and handle user's input with Ajax.
- ☐ You must use `localStorage` to store users' recent locales.

- ☐ You are encouraged, but not required, to integrate jQuery into your app, particularly since it facilitates use of JSONP, per `http://api.jquery.com/jQuery.getJSON/`. You are encouraged, but not required, to integrate jQuery Mobile, jQTouch, or Sencha Touch into your app. Use of any other JavaScript libraries must be approved by your TF.
- ☐ Your HTML must be valid HTML5 per `http://validator.w3.org/`, but your CSS does not need to be valid per `http://jigsaw.w3.org/css-validator/`.
- ☐ Your CSS and JavaScript must not be minified.
- ☐ Under no circumstances should we be able to trigger runtime errors in your JavaScript code. Be sure that you handle unwanted inputs and HTTP failures elegantly, as by reporting such errors or silently handling. Under no circumstances should your code trigger errors in Webkit's own console.

How to Submit.

To be announced via email prior to this project's deadline.