

Lab 4: Testing and Installing a Mobile App

Objectives of this lab session:

- Testing a mobile web app
- Making the app installable
- Distributing an app


Resources required:

- A current web browser (ideally Google Chrome)
- WebStorm IDE
- A completed version of Lab 4 (or Lab 3 if you have not completed Lab4, but Lab4 is better)

Part 1: Testing your work

Fully testing a Mobile Web App (or any mobile app for that matter) is not something that you can do without access to a huge set of Mobile devices: while there are well-defined standards for Javascript and, to a degree, HTML5, mobile browser implementations are changing continuously as new APIs are added and existing ones updated; for example, some browsers have lagged in supporting several HTML5 features while others have gone full speed ahead. Fortunately, most up to date mobile browsers will support much of HTML5, but the only way to be sure is testing *on specific devices*.

While desktop browser testing is not an ideal substitute, current browsers include features that deal with the most likely inconsistencies. Google Chrome (as of V32, currently at V39) includes an emulation mode that covers a range of mobile features – devices & user-agent setting (including screen resolutions), sensors (e.g. accelerometer), touch interfaces and network characteristics. Firefox provides a responsive view mode, which can be used to emulate a range of screen sizes. Currently, Chrome provides the best range of features, but that can change very quickly.

1. Open an existing project in WebStorm – I suggest Lab 4, since that includes a range of mobile-specific testable features, but Lab 3 will do if you've yet to complete Lab 4. It is important that you use WebStorm, since it includes a web server component that will deliver your app via HTTP. Opening the app by opening the main HTML file in a browser will mean that some of the APIs in use (e.g. geo-location) will refuse to work because of the wrong protocol
2. In WebStorm, select the app's main HTML file. The file name should be **index.html**, but if yours is not, simply right-click on the file in the Project pane, choose Refactor then Rename, and change the name. Open **index.html** and click on the Chrome icon at the top-right of the edit pane. The app should open in the Chrome browser
3. Press the key combination Ctrl+Shift+I to open the Chrome developer tools. When the tools window opens, it is worth disconnecting it from the browser window if it is not already disconnected – this icon  at the top right of the developer pane does that (see figure 1, below)
4. If it is not already visible, show the “drawer” – a tool area at the bottom of the Developers' Tools, which has a set of tabs along the top (Console, Search, Emulation and Rendering), and select the Emulation tab (see figures 1 & 2 below).

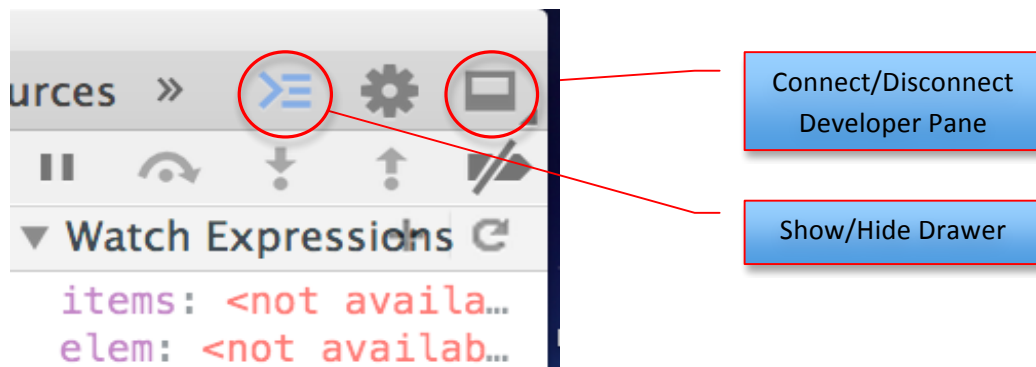


Figure 1: Control buttons in the Developer's Tools window

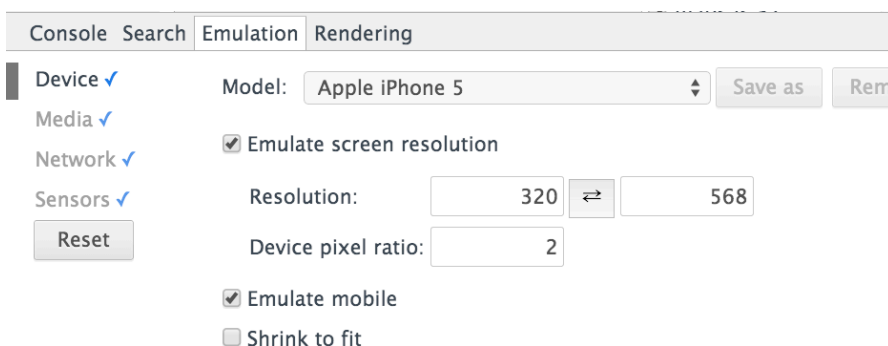


Figure 2: The "Drawer" in the Developers' Tools Window

5. Click on Device at the top-left of the drawer and select a Model. I usually start with the iPhone 5 as that is fairly up to date but will also highlight any major problems with your display. The browser window will change when you do this, and you may see a warning that you need to refresh the browser window. Refresh the browser and you will see your app running on the equivalent of an iPhone 5's screen. Press the rotate button (see below) and the pixel width and height will swap as if you had rotated the device

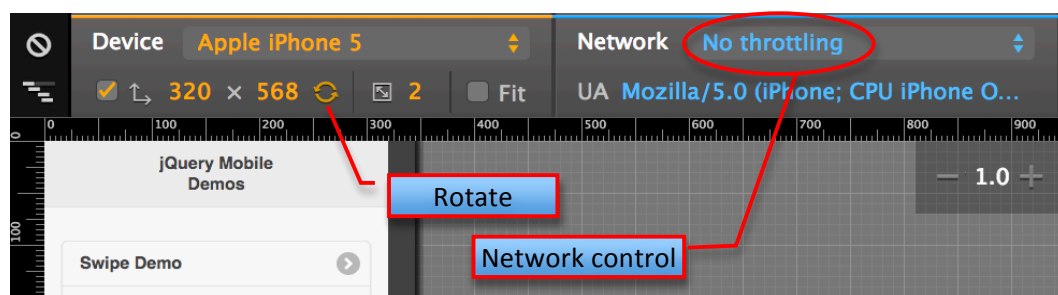


Figure 3: Device Emulation more

6. Take the (virtual) device offline by selecting "**Offline**" from the **Network** control box. Now refresh the display, and you should find that the emulated screen displays an "offline" message. We will fix this later.
7. If you are currently emulating the implementation of Lab 4, go to the maps page. It should show your location as close to Glasgow Airport (if you are doing this in the labs), or the location of your ISP's servers if not. In the Developer Tools window, click on the Sensors

setting (see below). You can now enter values for latitude and longitude – for example, the coordinates shown in figure 4, below, pinpoint a location near the centre of Paris. If you enter these coordinates and check “Emulate geo-location coordinates” and then refresh the page, the map should show your location as just north of the Seine.

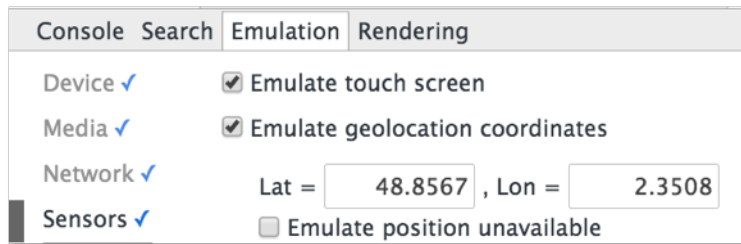


Figure 4: The "Sensors" page of the drawer

If you look further down the Emulation page of the drawer you will find that you can also emulate the Accelerometer sensors of devices that have them. We'll look into that in a future lab.

Using these Developer Tools you can test most features of a web-app (including many that many developers assume to be accessible only by native mobile apps). Of course, testing an app using an emulator is only a first stage; for a commercial app, it is important that full hardware testing is carried out on as wide a range of devices as possible.

2. Packaging a Web App

Once a web app is ready for deployment, it will need to be set up so that it can be installed on devices. That means that the app will need to be hosted on a web server, since web-apps cannot be installed by copying the project files to the device. The most important facet of this is that the app should be available offline; all major mobile platforms have browsers that are able to configure a web app to work offline. The two crucial components in the app are that it uses localStorage to store data that needs to be preserved while the app is not running, and that the app itself must be stored in the browser's application cache.

The current app (Lab 4) does not need to store any offline data; it is unusual in that respect, and you should not rely on this being a common thing. We can store the APP itself offline by providing a Manifest file: the important features of this are:

- It must list EVERY file required by the app by URL – this includes library code, which can be listed as local files by relative URL or as files in a content delivery network as full URL, graphics, style-sheets, HTML files and Javascript code files. If the app relies on any other files (e.g. text files containing data), these must also be listed
- Files must be listed individually – it is not possible to use a wildcard; for example **images/*.png**, to specify a set of files
- If the app contains files that **MUST** be accessed via the Internet – for example, weather report data – either it will not work offline, or you must cache local versions for use offline (e.g. yesterday's weather forecast for today will not be as accurate, but it may well be better than no weather report. You can specify NETWORK: files, provided you also specify FALLBACK: files that are cached local versions or store the data in localStorage

The Lab 4 app is a good example of a mix of local and remote files that must be cached to allow it to work offline. It has a specific feature that means it cannot be used offline in every situation – the Google Maps page requires a network connection to download map tiles from Google's servers, and these cannot all be downloaded in preparation for offline use. Instead, will can only recognise that mapping may not be available and deal with it by informing the user of this. Note, however, that a native mobile app could not do any better in this situation.

The file structure of Lab4 is shown below:

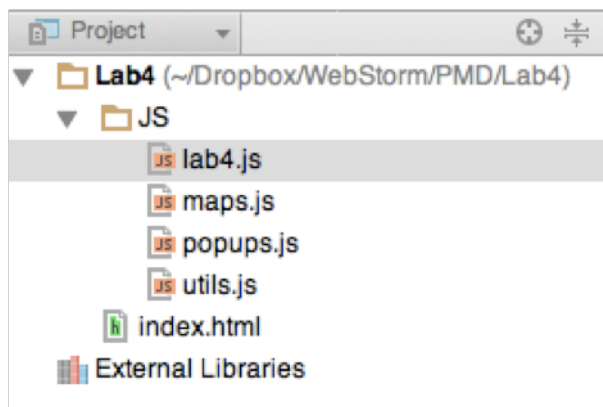


Figure 5: The local files that make up Lab 4.

Given this, we might conclude that the manifest file for this app needs to include only the following: index.html, lab4.js, popups.js, maps.js. However, if the app is to work offline, we need to recognise that it requires the availability of the jQuery Mobile code and CSS libraries. For the app to work properly offline, we need to include these files in the manifest by adding their URL to it. A suitable manifest for Lab 4 is:

```
CACHE MANIFEST

# 01-02-2015    Version 0.1

NETWORK:
*

CACHE:
index.html
js/lab4.js
js/popups.js
js/maps.js
js/utils.js
http://code.jquery.com/mobile/1.4.0/jquery.mobile-1.4.0.min.css
http://code.jquery.com/jquery-1.10.2.min.js
http://code.jquery.com/mobile/1.4.0/jquery.mobile-1.4.0.min.js
images/favicon.ico
```

Listing 1: a manifest file suitable for Lab4

1. Add a file to the project, and give it the name **cache.manifest**. (You can use any name you like provided it matches the name you add in the next step.)
2. You may notice that I've added a file to the list: **images/favicon.ico**. This defines an icon file that will be delivered with the app if we link it properly to the **index.html** page. Add the following two lines to the <head> section of the index file:

```
<link rel="icon" href="images/favicon.ico" type="image/x-icon">
<link rel="shortcut icon" href="images/favicon.ico" type="image/x-icon">
```

You will also need to find or create a suitable icon file – there is one in the folder for this lab on Moodle, although you can use any icon file you like

3. For the next part, it is important that you have fully tested the app – if there are any errors,

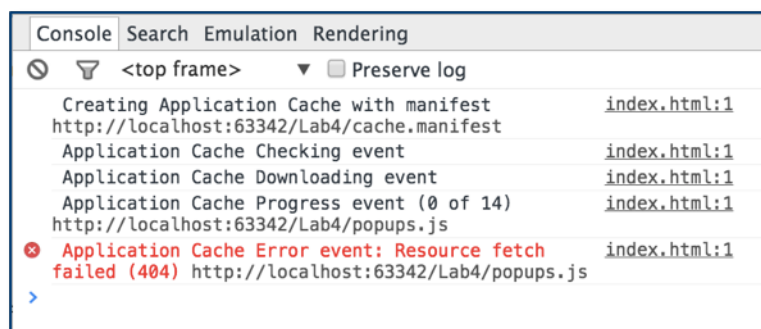
you will find that browsers will make it difficult for you to fix them because they will hang on to the files in the browser cache rather than download new ones – this happens even when the files are local. Consequently, you ought to do the following before you move on to step 4:

- a. Run the app in the Chrome browser if it is not already running, and open the developer tools if they are not open
 - b. Press the Shift key as you click on the refresh button next to the URL bar on the browser – this forces a reload, and you should see the icon file in the tab above the URL bar
 - c. Check the console in Developer Tools – if there were any problems relating to downloading the app, there should be error messages here – missing files show up in the console in red. If there are none and the app is fully working, proceed to step 4: otherwise, fix any problems before you proceed
4. Edit the opening `<html>` tag in the app to appear as follows:

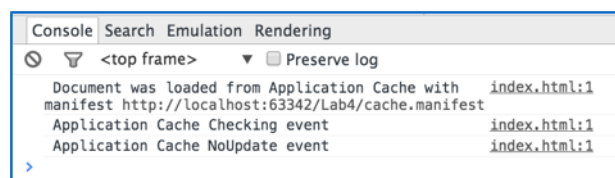
```
<html manifest="cache.manifest">
```

Note that the filename must match the name you gave to the manifest file, and the manifest file must be in the same folder as **index.html**. If it is in a different folder, use a relative address (e.g. **"folder_name/cache.manifest"**)

5. Re-load the app by pressing the browser refresh button, and then check the console again. The console should show messages relating to the progress in downloading the app to the cache. If there were problems in the download, there will be an error message in the console, for example:



If any files missing files are listed, check your manifest file before re-loading the app. A successful download will appear as:



Note – the Cache NoUpdate event indicates that files listed in the cache were not updated, either because there was no need (the manifest was unchanged) or the app was offline.

3. Delivering a Web App

The final hurdle (at least for a web-app that is not supported by custom web-services) is to get the

app on to your mobile phone, iPad or other device. This will allow more thorough testing, and the app will be able to take advantage of all of the features we have simulated during development using the emulator features in Chrome.

Ideally, you would need a website – for example, the free server space provided by your home ISP, or space you have hosted for some other purpose. The good thing is that the site that delivers the app does not need to provide any dynamic content – a static website that allows JS and CSS files will do, and most web hosting services provide that. If you have access to static web hosting via your ISP, just upload the app files in the way you normally do.

Of the free web-app hosting available, Divshot (divshot.com) is currently among the easiest to use.

To host an app at Divshot :

1. Visit divshot.com and sign up for a free account. This account will limit you to 1GB of bandwidth per month and 100 Mbytes of hosting space. This ought to accommodate several web apps
2. Log in to your new account and click on Dashboard (at the bottom of the screen)
3. Press Create App, enter a name for the app (all lower-case, e.g. my-app) and press Create App
4. Your new app domain should appear on the Apps page. There should be three rectangles on the App's page Development, Staging and Production. This set-up lets you work on new versions of the app while developing it further. The three separate areas correspond to a version you can work on (Development), one that is best used for final stage testing (Staging) and a fully working version (Production). You can promote an app through these three stages as it reaches the appropriate level of maturity (Development for a version you can freely work on, Staging to allow the last few bugs to be shaken out, and Production for a fast delivery version (hosted on a Content Delivery Network)
5. Press the Upload button in the Development area – the area should expand and a drop-area should appear in it
6. To upload your app, it will have to be in Zip format: in an Explorer window, open the app's folder and select all of the app files (don't try to zip up the whole folder – the individual files and folders within the main folder should be zipped. Select all of the files and folders you wish to upload, right click and select to compress to a Zip file
7. Drag the Zip file over to the Divshot browser window and drop it on the drop area. The app should upload and a message should appear, indicating the URL that the app is deployed at. I found this message appeared too briefly to copy it; just go back to the Apps page and you will find a Visit button has been added to the app if it was uploaded successfully
8. Click on the Visit button – your app should appear in the browser

At this stage, your app is deployed. I usually copy the URL from the browser window and email it to myself. You can then open the email in a smartphone and click on the link to visit the app. It would be a **very good idea** to read through the documentation at <http://docs.divshot.com>. Divshot provides a good set of features, such as version control, custom domains, clean URLs (i.e you can leave off .html from the URL) and other useful facilities.

End of Lab Sheet