Session 2 - Scaling the ui (Put on your gloves it’s time to box)

We will cover:
1) Creating scalable layouts that automatically fill the screen with the -webkit-box,
2) Styling without images.
3) Styling the text in menus.

Step 2a - Boxing the layout

Getting the basic layout sorted was one of the most time consuming parts of my web app. I wanted a titlebar at the top an area in the middle for content and a menu bar at the bottom. The titlebar and menu bar should have a fixed height and the content section in the middle should expand to fit the screen. This is pretty much the default layout for all native iPhone apps.

None of the frameworks that were available at the time or books helped that much in solving this problem (we're talking 18 months ago). The lack of position fixed from css also made things tricky. I resorted to using javascript to solve this problem. It worked but it was quite a mess and required everything to be recalculated when the user switched orientation, which meant event listening for orientation change and so on. Then I discovered the css box model. It's awesome and one of the best bit's about the new css. When you get the hang of it you're going to love it. It's just a shame that only the latest desktop browsers support it. Luckily mobile browsers basically support the cutting edge of everything, so we're sorted.

1) Open the webpage step2a.html in your editor.
2) Look at the html. We have taken the template we created earlier and added titlebar, content and menu bar divs, to the body.
3) Looking at the inline css.

Now lets see how the page renders in your desktop safari or chrome browser. (Not great)

What we want is the content to fill the page forcing our menu bar to the bottom. We want this to work even if we rotate the screen.

So lets start boxing by adding some inline css wizardry

4) Add a div with a class of "box" to the body so it encapsulates our other divs.

```
 <div class="box">
       <header class="titlebar">titlebar</header>
       <div class="content">content</div>
       <nav class="menubar">menubar</nav>
</div>
```

5) Create a css style for our box class and set it to:

```
.box {
      height: 100%;
      display: -webkit-box;
      -webkit-box-orient: vertical;
}
```

Note: display: -webkit-box; This tells the renderer to stack the child elements like boxes. -webkit-box-orient: vertical; This determines how they should be stacked; vertical = on top of each other; horizontal = side by side.

Now we have to make our content div a little more flexible, so it flexes into the space available.

6) Add the following css.

```
        .content{
              -webkit-box-flex: 1;
        }
```

Note: -webkit-box-flex means fill the available space. We'll play more with this later.

That should be it, save and preview. Looks great doesn't it....

What??? it doesn't work? No way.

Welcome to a little html5 gotcha.

7) Let's try removing the `<!DOCTYPE html>` from the top of the page so our renderer views the page as html4. Try again. Try resizing your browser. Everything should work as we want. Titlebar at the top, menubar at the bottom, content flexing to fill the available space.

Great but we want HTML5. We're doing cutting edge stuff here.
With html5 we have to go one step higher in our css when defining height:100%. We need to make sure our html tag fills 100% of the screen size not just the body.

8) Put back the `<!DOCTYPE html>` to the top of the page.

9) Add the following to our css section.

```
       html{
         height:100%;
      }
```

10) Load in your desktop safari or chrome.
11) Try it on some of your mobile devices. Try changing screen orientation.

PLAYTIME:

a) Try switching between horizontal and vertical layouts.
b) Add another div into the box div.
c) Add css to make that div flex too.
d) What happens if one of the flexible divs is set to -webkit-box-flex:1; and the other to -webkit-box-flex:2; ?
e) What do you think will happen when you use add "-webkit-`box-direction: reverse;`" to .box? Try and see.

The box layout will drastically simplify and speed up your development when you need a custom mobile layouts. Everything from photo albums to lists can be made to look beautiful on any screen size with the help of box.

Note: as we're using -webkit and box isn't stable yet we always refer to it as -webkit-box. In mozilla it is box and opera it is box. No guesses which browser doesn't yet support this model. Apparently it was partially supported in an early alpha version of IE 9, but not in the current. It will be included along with other CSS3 goodies in IE 10. Hopefully the spec will be finalised and we will have one box to rule them all (box).

Step 2b - Boxing the menubar.

Lets create a horizontal box for our menubar items.

1) Alter the menubar style like this:

```
.menubar {
  height: 40px;
  background-color: blue;
  display: -webkit-box;
  -webkit-box-orient: horizontal;
}
.menubar > div {
  -webkit-box-flex: 1;
}
```

2) Add some menu items to our menubar as follows.

```
 <nav class="menubar">
        <div>item 1</div>
        <div>item 2</div>
        <div>item 3</div>
</nav>
```

3) Save it and look at it. Add a few more buttons and watch how it automatically rescales. Try switching orientation on your device too.

Note: .menubar > div, this is a css3 “child combinator”. It means all div elements that are a child of main.

You should now be comfortable with the box model and have a basic layout template.

Reference:
There is a lot more cool stuff you can do with box check out these links:
http://hacks.mozilla.org/2010/04/the-css-3-flexible-box-model/
http://www.html5rocks.com/tutorials/flexbox/quick/

For more new ways for selecting elements in css3 see http://www.w3.org/TR/css3-selectors/

Step 2c - Making things look pretty without images.

Let’s face it our layout might be good, but it looks like crap. In this section we will fix that. We want our wiki app to have the native look and feel.

One area that is really important with a mobile web app. is keeping the app size down and reducing the number of requests made to the server.

A great way to achieve this is by using webkits css features to generate user interface components and not bother with images. This not only increases the download speed of the app it often also makes it look better. A great example of this is css buttons on the iPhone. The browser generates an image based on the css definition. Since the bowser is sitting on the device if you have a seriously high definition screen i.e. iPhone 4 with 300dpi, the browser generates an image with that resolution, if you’re on a older machine with a 96dpi resolution it generates that resolution. This means your app is going to look fantastic on new devices without having to change the resolution of masses of static images and using phone detection etc. for deploying them to the right device.

Let’s start out by tackling the horrible red titlebar. What we want is a beautiful gradient background. So let’s take a look at -webkit-gradient.

-webkit-gradient generates a gradient image, you just need to specify the parameters. It can be used anywhere in your css where you can use an image url.

We are going to use it with the background-image property of titlebar.

1) Replace `background-color: red;` **with** `background-image:-webkit-linear-gradient(bottom, rgb(176,14,54) 15%, rgb(214,17,17) 65%);` **from the titlebar css.**

This generates a linear gradient starting at the left top corner of our element to the bottom left.

2) Increase the size of the titlebar to 400px. Try changing the from and to colors.

3) When you feel you understand how it works choose a nice gradient color and return the titlebar size to 40px.

4) Add a border-bottom as the finishing touch.

```
    border-bottom: 1px solid #666;
```

Now let’s fix the menubar.

Unlike the titlebar the standard iPhone black menu bar has two parts to it’s gradient. No problem we can achieve this too.

We’ll start with what we know:

5) Replace `background-color: blue;` **with** `background-image: -webkit-linear-gradient(top, #27282b 0%,#000000 100%);`

Take a look at your page with your desktop browser.

Note: If you wan’t to get really creative you can add transparent colors by using rgba to specify colors e.g. `to(rgba(228, 100, 100,50)).`

6) Take a look at your page.

Note: The gradient is almost flat, and that is how it is in iOS too.

Still not convinced of the power of gradients: checkout the awesome css only buttons @cubiq created. buttons.html

Reference:
http://webkit.org/blog/175/introducing-css-gradients/
http://gradients.glrzad.com/
http://leaverou.me/css3patterns/

Step 2d - Styling text

Now we've sorted out the gradient let's style our text so it looks nice.

1) Open session2/step2d.html in your editor.

2) Browse to the site in Chrome.

It's ok but we want the highlight of our active menu item to have rounded corners.

3) Add a border-radius to the `.menuActive` css rule:

```
    -webkit-border-radius: .4em;
```

We can add a little text shadow to our text to make it look like it engraved.

4) In the css for the `h1` element we'll add `text-shadow: 0px 1px 0px #555;`

This drops a shadow around are text. Try experimenting with its parameters.
Note: This is not supported on Android yet, only iOS.

5) Staying with `h1`, we have a really long title, our app should be scalable for multiple screen sizes, see what happens with the text if you make the browser size too small for the text to display in it's entirety.

6) Add the following style to `h1`

```
     overflow: hidden;
     text-overflow: ellipsis;
```

7) Try resizing again and note what happens to the end of the text as the screen size get's too small for to show.

8) Add a max-width if you want total control "max-width: 200px;"

If you have any extra time try creating a nice looking button, gather inspiration from cubiq's examples session2/buttons.html

That's it for section two.

You now should be comfortable with the box model and have a basic layout template. You should understand how -webkit-gradients, text-overflow and text-shadow.