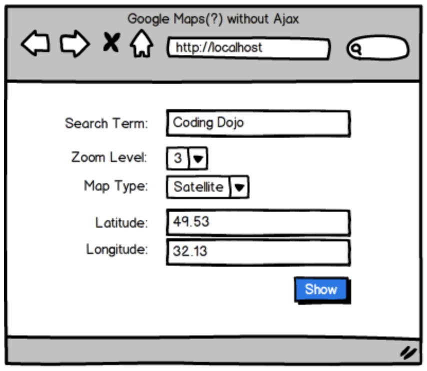
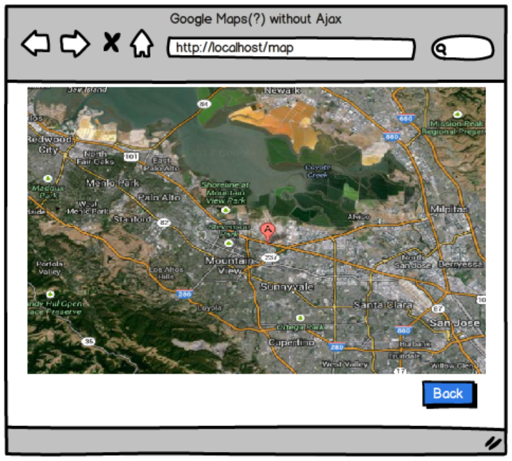
Traditional Web Applications

With traditional web apps (web apps built prior to Ajax or apps built in the early 2000's), once the browser sent an HTTP request and rendered the HTTP response returned from the web server, you did not expect what you see in the browser to change. It was impossible back then to imagine (let alone create) an app where the contents would refresh as you change some of the values on the web app. When Google first used Ajax to give "keyword suggestions" right as you were typing the search term, it shocked the whole industry! In other words, traditional web apps were designed to serve static contents and for the developers to use a form to pass data from one page to the other. It was never really designed to build web apps that dynamically change contents without the browser going to a whole new page. If you look back on some of the recent web 2.0 sites you've used, you'll know that new web apps don't function like the old apps. It doesn't have so many forms for you to fill out, but instead, as you click and do things within the page, it seems to automatically update part of the web page, without having to go to a whole new URL. Ajax has now become an integral part of building modern web apps.

If you had to build something like Google Maps without Ajax, you would have done something like below.

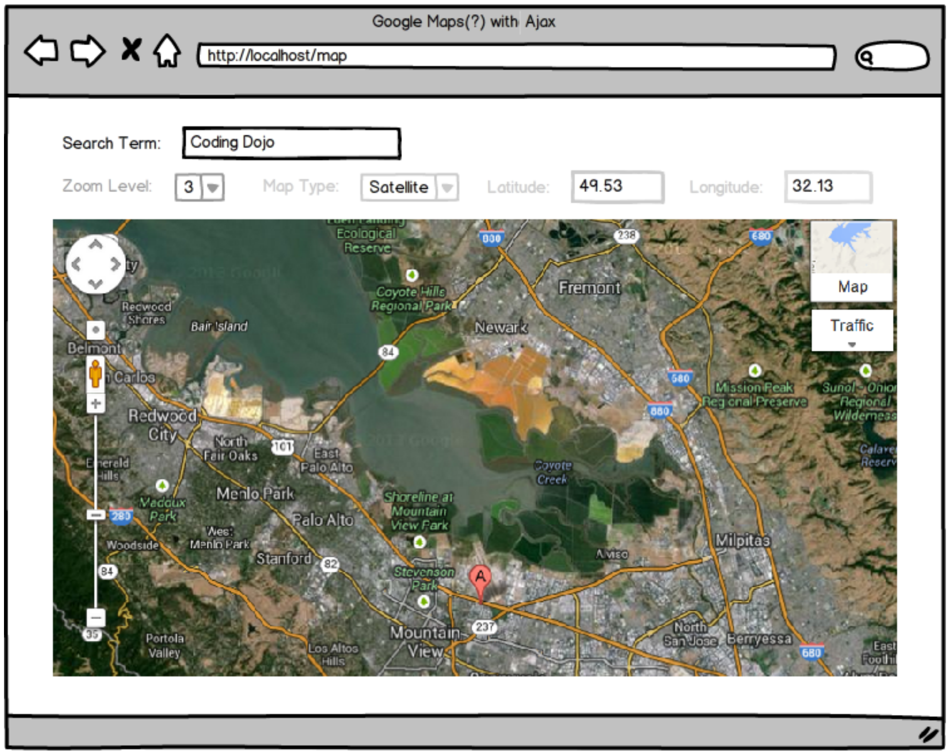


Search Term:	Coding Dojo
Zoom Level:	3
Map Type:	Satellite
Latitude:	49.53
Longitude:	32.13



After AJAX

With Ajax, our Google Map could work like something below.


Google Maps(?) with Ajax

http://localhost/map

Search Term: Coding Dojo

Zoom Level: 3 Map Type: Satellite Latitude: 49.53 Longitude: 32.13

Now, notice that the zoom level, map type, latitude and longitude (these are coordinates of where the center of the map is) are still there in the form but are grayed out. This is because the information is still within the form but are just hidden! In other words instead of *<input type="text" name="zoom_level" />*, you would now have *<input type="hidden" name="zoom_level" />*. As the user clicks on different items on the HTML page (e.g. the + or the -) or as the user changes the search term or the map type, you would change the value of the form input, send the form information to the server, retrieve some information from the server (maybe a new map?) and then update part of the original HTML page. Also, notice that with Ajax, we don't even need a submit button as we can tell the browser to submit this form automatically behind the scene when certain events occur (when some value gets changed, etc).

So let's review what's now happening with the new Google Maps example (with Ajax):

When the user clicks on +/-, it updates the value of *<input type="hidden" name="zoom_level" />*

When the user changes the Map Type, it updates the value of *<input type="hidden" name="map_type" />*

When the user clicks on the map and moves left/right/top/bottom, it updates the value of both longitude/latitude as these are the coordinates where the center of the map should be.

Whenever any of these changes occur, the client sends this form information to the server (e.g. localhost/process or whatever URL you choose) and tell the browser not to actually go to that page nor reload the page.

. The client waits for the server to send some information back, and once this data is received (say the new map info), it displays the new map somewhere in the HTML (using JavaScript, jQuery, or any other JavaScript library that allows HTML manipulations).

With this new approach of submitting the form information WITHOUT reloading the page, we can create cooler apps! Let's look at more examples to see how Ajax has been changing how web apps are built.

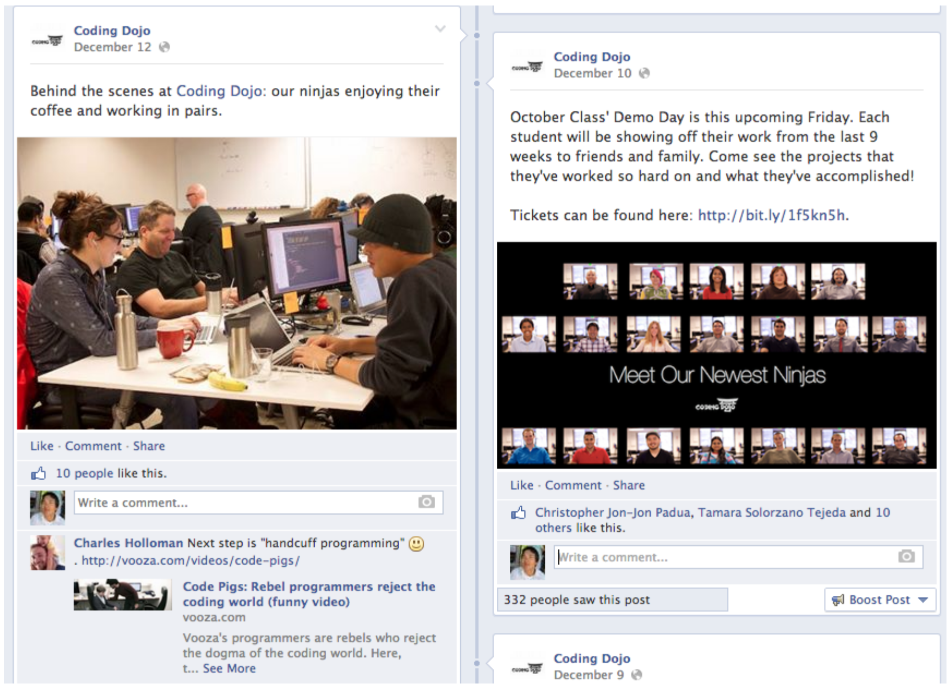
AJAX In Facebook

Again back in the days where Ajax was not available, every single HTML form built once submitted went to a new page (wherever the form's action attribute specified). For example, say you had a form like this somewhere in your HTML:

```
<form action="/post/message" method="post">
   <textarea name="message"></textarea>
   <input type="submit" value="Post a message" />
</form>
```

Whenever someone clicks on the submit button, the browser would go to *"/post/message"*. If you wanted to come back to the original page, you needed to press the back button or type the previous URL manually in the browser.

Imagine how posting a message on Facebook could have been without Ajax. Imagine that you wanted to post a comment in Facebook in a page that looked like below.



Coding Dojo
December 12

Behind the scenes at Coding Dojo: our ninjas enjoying their coffee and working in pairs.

Like · Comment · Share

10 people like this.

Write a comment...

Charles Holloman Next step is "handcuff programming" 🙂 . http://vooza.com/videos/code-pigs/

Code Pigs: Rebel programmers reject the coding world (funny video)
vooza.com
Vooza's programmers are rebels who reject the dogma of the coding world. Here, t... See More

Coding Dojo
December 10

October Class' Demo Day is this upcoming Friday. Each student will be showing off their work from the last 9 weeks to friends and family. Come see the projects that they've worked so hard on and what they've accomplished!

Tickets can be found here: http://bit.ly/1f5kn5h.

Like · Comment · Share

Christopher Jon-Jon Padua, Tamara Solorzano Tejeda and 10 others like this.

Write a comment...

332 people saw this post

Boost Post

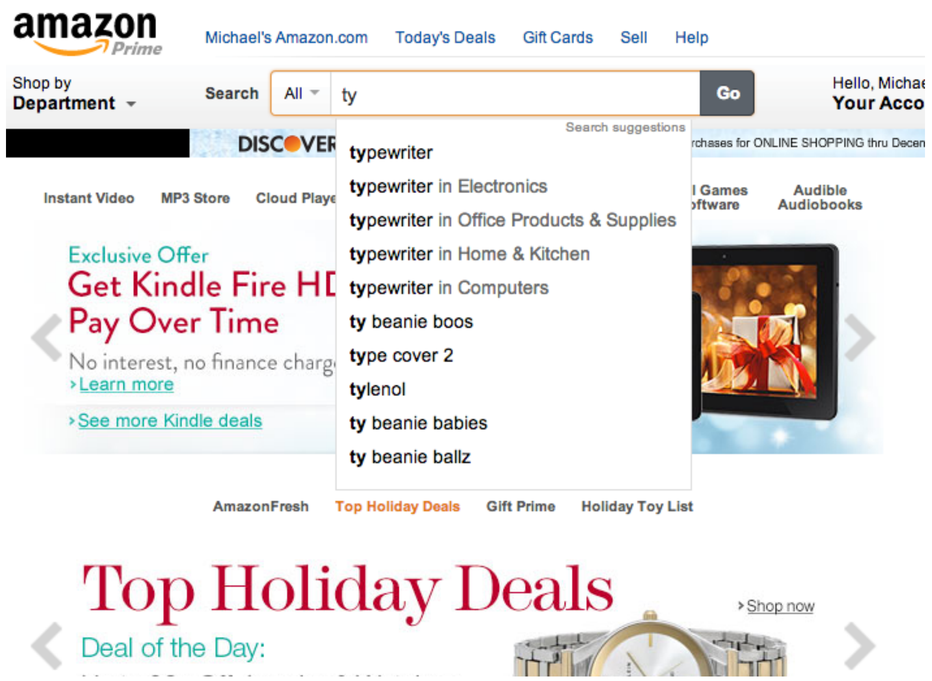
Coding Dojo
December 9

Say you went to the text area where you could post a message/comment and pressed enter. If this resulted in loading a new page every single time you post a comment to any picture/post/event, it would drive you crazy! (Imagine you were posting a comment in one of the timelines way back and once you posted the comment, it reloaded the page where you had to scroll all the way down to find the things you were reading about.) Back before the days of Ajax, this was the norm; there was no way of having your browser send another HTTP request behind the scene and just update part of the HTML on the page. With Ajax, though, now what's happening is something like below

1. You fill out a message/comment.
2. You press enter.
3. It submits the form but does NOT have the browser redirect to that page.
4. The server that received the form information does whatever it wants to do (e.g. save the information in the database, return some type of message indicating whether it was successfully posted, etc), and sends some type of response back to the client/browser
5. The client waits for this information and once it receives it, updates part of the HTML with whatever it wants (e.g. maybe the new comment that just got submitted).
6. Maybe while the client is waiting for some of this information to be returned from the server, it displays an icon that looks like below.

Auto-complete Ajax Example

When you go to sites like Google or Amazon, as you type in the search term, it displays a drop-down of possible search terms, like below.



How is this done? Using Ajax! Any time the browser fetches new information from the server, the page that has been loaded is all Ajax. Let me repeat. Any time the browser gets new information from the server, ONCE it renders the page, is all Ajax (assuming the sites were built in PHP, Python, Ruby/Rails, .NET, Java, or basically everything except Node). Now, let's see how this auto-complete drop-down menu in Amazon could have worked.

. The user types something in the search box.
. Whenever the user changes or types something in the search box, the browser sends a HTTP request to say */auto_complete_suggestions* and sends data (maybe it sends data that was stored in a form like *<form ...><input type="text" name="search_term" /></form>*). Again the browser does NOT go to this new page or reload the page but just waits. In other

words, this data submission from the client/browser to the server is happening in the background (or without going to another page).

. The browser waits for the server to send some information back.

·. Once the server sends some information back to the client/browser (say top 10 search suggestions), the browser displays these 10 search suggestions in a drop-down format using JavaScript.

Summary

Say you want to create a page that can make a ***remote HTTP request*** (when a button is clicked, when a form is submitted, or when someone pressed a key) and want to have your app ***use the HTTP response from that request*** to update ***part of your page***. You can do this using Ajax. You do NOT always have to submit a FORM to do an Ajax call but as we often have to send information to a specific URL to retrieve certain information, we're introducing the FORM concept as it would make it easier for you to group your data in preparation for making a new HTTP request.