Spring 2019

Data-Driven Human-Centric Web Design: Beyond Tables and Statistics

Sherry Berghefer
Iowa State University, slbergh@iastate.edu

Data-Driven Human-Centric Web Design: Beyond Tables and Statistics

by

Sherry Berghefer

A creative component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

Master of Science

Major: Human Computer Interaction

Program of Study Committee:
Dr. Andrew King, Major Professor
Dr. Joel Geske, Committee Member
Dr. Michael Dahlstrom, Committee Member

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this creative component. The Graduate College will ensure this creative component is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

ABSTRACT

Conventional web design tends to see tabular data as providing a barrier to creative graphic design. This project explores how such data can be displayed in a more abstract visual format, free from tables and the need to navigate through layers of information. The creative component reported here is the result of a personal exploration based on curriculum taught by the author in an undergraduate interdisciplinary course that merged perspectives from the fields advertising, graphic design and computer science to create personalized data-driven applications. Of particular interest to this project was determining if there are limits to the design elements that can be controlled by live data in a web page displaying weather information.

Seven data sources, connected to via application programming interfaces (APIs), were used to bring in 16 location-specific data points relating to weather, population, and geolocation. Literal text-based content displays used some of the data points, while others were used to control various visual elements (e.g., backgrounds, weather condition representations, and animation speeds). The result was a single visual web page displaying current conditions and forecasts using limited text, dynamically applied images, and Cascading Style Sheet (CSS) animations, all of which required minimal user intervention.

CHAPTER 1. INTRODUCTION

In 2015, I, along with two other university faculty members created and piloted a multi-disciplinary course called “Computational Communication: Advertising Creative.” Despite the bulky name, it was an interdisciplinary course combining advertising, graphic design, and computer science students with the goal of bringing data into creative projects, encouraging computational thinking (i.e., problem-solving methods that involve expressing solutions in ways that a computer can execute), and bridging the gap between creative expectations and technical limitations in application development.

Each faculty member had his or her own area of expertise: one was an experienced former advertising executive and one was a professor of computer science. My role in the course was to cover design topics, having an extensive background as a graphic design practitioner. I also have a significant background in web application development, putting me in the position of being able to essentially “speak” two different languages: design and code.

The project underlying this creative component was borne from that course. A variety of topics were covered in the course, including how to retrieve data from application programming interfaces (APIs) using hypertext markup language (HTML) and hypertext preprocessing (PHP). Unlike HTML, where information displayed on a web page must be explicitly coded into the page, PHP is a server-side scripting language, meaning the PHP code is automatically converted into HTML prior to page load (Christensson, 2006). Essentially, this means that the page content does not exist prior to page rendering. Additional topics included user experience (UX) and user interface (UI) design. UX defines the entire user experience from how pleasing colors are to frustrations encountered to emotional state during and after use. UI refers to the actual interface and navigability based

on understanding icons and color schemes to recognizing menus and buttons. Both UX and UI are essential considerations in website and app design.

As part of the course curriculum relating to coding, the computer science professor introduced APIs and how to make connections to them using PHP commands. Using data retrieved from free data sources (e.g. Firebase, Twitter, and Weather Underground), students were taught about authentication, endpoints, and arrays. Although I had a development background, I had not worked with APIs until this course. My background had been in developing, accessing, and displaying information from databases. This meant I understood the fundamentals, but the mechanics were new to me, so I worked along with the rest of the class on the API material.

Once we had successfully connected to the weather API, I demonstrated how to use HTML and Cascading Style Sheets (CSS) to format the data based on the values returned. For example, in class, if the incoming temperature data was below 50-degrees, the page displayed a snowy image. When the temperature was between 50-degrees and 70-degrees a springtime image displayed, and when the temperature was above 70-degrees, a beach image displayed on the page. This process occurred programmatically using computational logic, so the formatting changed without any user intervention. Obviously, the idea of formatting based on data was not new to me. This was something I had done with database data. It did, however, start me wondering exactly how far I could take the idea of data-driven design.

The project is a mixture of best practices and unique approaches. In the realm of best practices, the creative component makes use of CSS, separating content and formatting. It uses the CSS to make the website responsive, so that it adapts to different screen sizes, both in layout and in text formatting. What makes it stand out in its approach to handling weather

data is that this creative component eschews current weather app design conventions (e.g., tabular weather displays, multiple pages, and layers of weather data) in favor of visual representations of the weather beyond simple icons. Instead, of using structured elements to hold the data, this creative component uses the data to illustrate the data. Further, this project also minimizes the amount of information displayed, distilling it down to current conditions plus three days.

CHAPTER 2. LITERATURE REVIEW

We have all seen weather websites and weather apps. Many of them look and function similarly to each other, displaying detailed information relating to temperatures, barometric pressure, wind vectors, moon phases, etc. Many allow for location-based reporting to ensure the user sees relevant information first, also providing a search feature to look up information for other areas.

One necessary initial step to creating a weather website/app is to determine what it should look like. Although many of the existing solutions overlap in visual design, there are numerous methods for displaying relevant data. Design trends, such as those seen in weather displays, often become trends simply because a particular layout met with success and was emulated by others. These sites function perfectly well, for the most part, however they tend to be visually lacking. The existing sites employ a techno-centric approach: providing layers of data which can then be accessed through links, buttons, and toggles, ostensibly allowing the user to control their experience on the site.

Speaking as a practitioner, while many of those developing weather websites and apps do have technical programming proficiency, it is unlikely most also have a theoretical design background (Barraclough, 2019; Cousins, 2013; NewGenApps, 2018). Instead, web developers rely on global standards that are in a perpetual state of evolution (Achour et al., 2019; Refsnes Data, n.d.; World Wide Web Consortium, n.d.). It is then the role of the developer to maintain functionality by ensuring best practices are met and current standards are employed, all while minimizing deprecated coding techniques.

Web design trends are even more fluid than web development standards, changing not in response to a written set of rules, but rather as a result of the designer preferences and

client requests. Despite this fluidity, designers do tend to have some type of theoretical background that allows the justification of design choices, making it is possible to employ a paradigm of visual perception to give context to the importance of UX and UI.

In the 1970s, psychologist James Gibson developed his theory of affordances (Gibson, 1979). The theory of affordances essentially determines what an environment has to offer, either positively or negatively (Gibson, 2015). Gibson used the theory as an ecological means to understand visual perception and developed this theory in the context of animals in a natural environment. For example, he described a horizontal, flat surface as affording support for other behaviors, such as movement via walking (Gibson, 2015).

Extending this theory to the digital world, the construct of a surface providing support would be equivalent to the framework established through HTML standards to create web pages, which would then afford the display of information and the manipulation of information through the use of interactive elements. Norman (1988) takes an affordances approach and moves it from environmental into more tangible and digital contexts. Norman redefines affordances within the context of web or app design as “perceived” affordances. He says that the perceived affordances, what the user will “get out of” an interaction, exist independently of what is seen on the screen.

Within the realm of the design of interactive digital systems, Norman (1988) considers design to distill down to just two aspects: the user being able to figure out what to do and the user being able to tell what is going on. Norman writes: “Design should make use of the natural properties of people and the world: it should exploit the natural relationships and natural constraints. As much as possible, it should operate without instructions or labels” (p. 188).

Within the context of a weather display, the current design trend appears to require a significant amount of user interaction. I evaluated four popular mobile weather apps to make contrasts with this project. At this point, I feel it is important to note two elements. First, that these external weather apps were evaluated near the completion of this project. While I was already familiar with their basic designs, specific design conventions were not explicitly considered in the development of this project. Second, the evaluation includes mobile apps, while this creative component is a website. Because the same basic layout and functionality occurs with this creative component regardless of screen size and type, I ultimately determined the website vs web app discrepancy to be a non-issue. Since this project involved the display of current conditions and a three-day forecast, the evaluation of the commercial apps was limited to the interaction steps required to access the current conditions and forecast, as well as the overall aesthetic (Table 1.1).

Choi (2012) proposed a standardized system to evaluate apps using thirteen dimensions: typography, visual hierarchy, grid-based layout, alignment, colors, legibility, buttons, intuitiveness, simplicity, navigation, contents, attractiveness, and consistency. Given the design of this project, I chose to focus on four elements that I found the most relevant across all of the apps plus the project: visual hierarchy, buttons, simplicity and intuitiveness.

Visual hierarchy refers to the ability to identify the importance of items, particularly type, based on their size, treatment (e.g., bold, italic, etc.) and color. Visual hierarchy helps direct the viewer's eye, indicating to the user in which order items on a page should be viewed. Buttons come with the expectation that the user is able to readily identify that an item is a button and that it is available for interaction. Ideally, the user will have an indication what will occur once a button is pressed or tapped. Simplicity refers to how easy it is to

understand the content, as well as how easy it is to visually navigate a page. In visual design, the term is used to define to how much work is required from the viewer to both see and comprehend a design. Intuitiveness in interaction design is a reference to how well a design adheres to established norms. Moving a menu, for example, from the top of the page to the bottom of the page makes navigation unintuitive since most users do not expect to find a menu located there.

Table 2.1 *Brief evaluation of select weather apps*

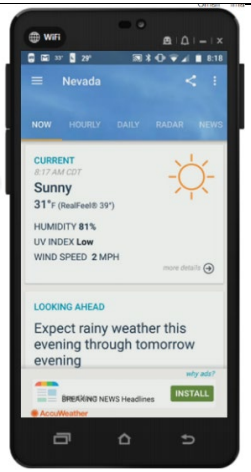
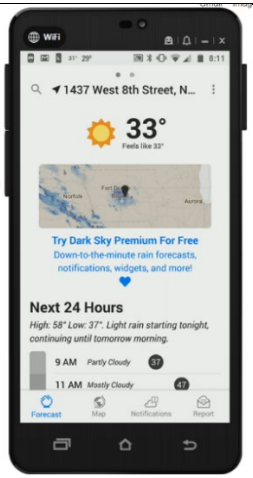
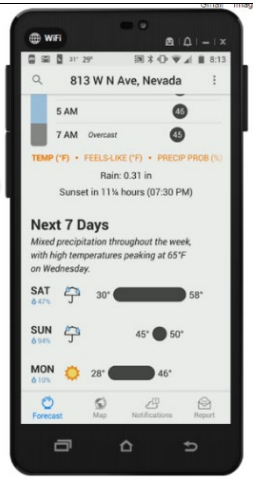
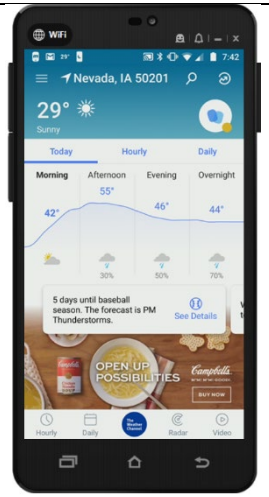
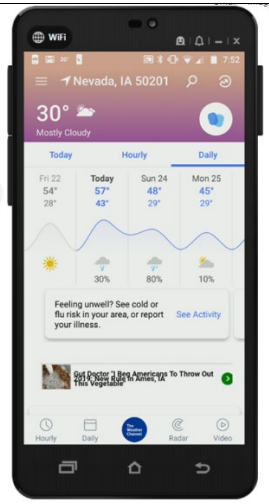
App Name	Home screen	Forecast screen	Interaction steps	Overall aesthetic
Accuweather			*Current conditions:* A small amount of swiping down on the home screen to view the Looking Ahead. A "more info" link provides details. *Forecast*: Tap to access the forecast. Toggles provide access to predicted temperature OR predicted precipitation. Another toggle access nighttime data as well.	*Visual hierarchy:* The larger icon immediately draws the eye to the middle of the screen where the current conditions exist. *Buttons:* The buttons are evident and easily identified. *Simplicity:* It is evident what is expected of the user. *Intuitiveness:* Overall the design is fairly intuitive, though the toggles were not immediately evident.
Dark Sky			*Current conditions:* The temperature is easy to find, but it took 3 swipes down in order to find the forecast. *Forecast:* No button exists to indicate where the forecast is. The user must swipe past the 24-hour forecast to get to the daily forecast.	*Visual hierarchy:* The advertising carries more visual weight than the current conditions. *Buttons:* There are minimal buttons and not ones that are expected. *Simplicity:* Confusing. This app has two home screens, one for the user and one for Dark Sky. *Intuitiveness:* Not intuitive. The home screen is 10 swipes long.

Table 2.2 *continued*

App Name	Home screen	Forecast screen	Interaction steps	Overall aesthetic
Weather Channel			*Current conditions:* The most basic of conditions are easy to find. The user must swipe to the right to access more information. This app uses stories similar to social media stories for content. *Forecast:* A button takes the user to the daily forecast. More stories are available on this screen. The user must swipe to the left to view more information.	*Visual hierarchy:* The trendline in the center has more visual weight than the conditions but does not convey much information. *Buttons:* The stories icon was not easily recognizable. The times of day appear as if they are buttons, but they are not. *Simplicity:* Not simple to understand. *Intuitiveness:* Not intuitive.
WeatherBug			*Current conditions:* Basic current conditions are easy to find. Detailed information is difficult to find. The home screen is 8 swipes long. *Forecast:* The forecast was easy to locate. Days can be tapped for more information. The arrow is a toggle to show both day and night predictions.	*Visual hierarchy:* There is a clear hierarchy at the very top, but it is not carried through. *Buttons:* It is not always clear what is a button and what is not. In many cases, tapping on text is the only way to identify buttons. *Simplicity:* Fairly simple to understand. *Intuitiveness:* Not very intuitive. The home screen required multiple swipes.

Each of the evaluated weather apps tend to require some degree of user interaction to access the basic weather conditions and forecast. Each of the evaluated apps also contain a significant amount of data beyond conditions and forecast, i.e. weather news and weather stories. Zabini (2016) says that the use of multimodal forecast information that includes tables, symbols, maps, text, etc. may actually make it more difficult for users to correctly interpret the forecasts. Ryan (2003) agrees that the visual information conventionally used in weather apps may make interpretation uncertain, especially since different apps may use different visualizations, icons and variables.

In 2015, Tech Crunch broke down a Nielsen report regarding the popularity of, and time spent using, various smartphone apps (Perez, 2015). Weather apps were lumped into a category that also contained news and sports apps. This entire category only accounted for 3% of all minutes used on the devices in the study, with news being the most used app in that category (Perez, 2015). Given that so little time during a day is being spent with weather apps, is it actually necessary to provide so many different options for how to view the data?

CHAPTER 3. THE WEATHER DISPLAY

My decision to create a data-driven weather display was driven by two goals: to control as much as possible with data and to have the fact that data was controlling the display be as invisible as possible. As a graphic designer, I knew I wanted the display to be very visual. So many of the current weather sites and apps are very structured with tabular displays of temperature and windspeed. I wanted to create something very different. Typically, UI – at its simplest – has essentially been thought of as how to make a better button. With this project I questioned whether buttons were even necessary.

I, like many designers, have been guilty of putting form over function. It is easy to decide what something looks like then write code to fit the design. Conversely, I have also been guilty of simply using a design as a container for data. By exploring data-driven design, I encouraged myself to look beyond obvious design conventions. Going into the project, I had an idea of what I wanted the end product to look like. Given that I wanted the code to control the visuals, I knew that I needed to be flexible in my design. Buxton warns that in the design of interactive systems, to which websites and apps belong, code should be the last element considered (Buxton, 2007). Ordinarily, I would agree; however, in this instance the code and design went hand in hand. This presented a flexibility that allowed me to use the data to explore relationships with the visuals I might have otherwise ignored.

Data sources

My goal was to use only free data sources, which were not always as easy to find as I had hoped. I would also like to note that the API I had initially used for this creative component was discontinued toward the end of the project, leaving me in the position of having to find a new API. Unfortunately, this led to a massive recoding of the project, as the

API data structures and keys were different. For this project, I ended up using the following data sources:

- Server variables for the user's IP address
- Geocoding from geoPlugin for latitude and longitude from the user's IP address
- Geocoding from MapQuest for latitude and longitude from the search form
- Weather from Dark Sky (original data came from Weather Underground)
- Moon phase information from the United States Naval Observatory
- Geocoding from GeoNames for population
- Device screen size

I started very simply, using the weather API to return current weather information and forecasts. The way the weather API works, I needed to include information a city and state in my query, so I began by hard coding the information into my query string. Most APIs have a documentation page that indicates the endpoints and keys available for access. They are also often available in multiple formats. For this project, I chose to work in the JSON format simply because I was already familiar with it.

JSON or JavaScript Object Notation is an open-standard format meant for data exchange. Because of its structure, it is as easy for humans to read and understand as it is for machines to parse (Refsnes Data, n.d.). JSON data is returned as text contained in arrays. These arrays contain data returned as strings, numbers, objects, or other arrays. Because of the structure of JSON data, it is a fairly simple process, in most cases, to create variables that can then hold and display dynamic information.

Once I was able to return the correct information for the hard-coded location, which was tested by changing the location used, I began the process of automating that through the use of variables. The weather API relies on latitude and longitude provided in decimal

degrees. Rather than manually looking up values each time, I created latitude and longitude variables to hold that information. To bring my location in automatically, I turned to server variables, specifically $_SERVER['REMOTE_ADDR'], which returns the IP address sending requests to a web server. Using that IP address, I was able to input that information programmatically into the geoPlugin. GeoPlugin is able to return the country, state, city, latitude, and longitude associated with an IP address. Each of those elements was stored in a variable for later use. These five variable names ended up being at the core of the functionality.

The next step was to develop a search form for both domestic and international cities. The search form inputs were used to create variables for city, state, and country. The reason for utilizing variables is due to the ability to use programming logic to assign values to each variable. For example, when the page loads into the web browser, it checks to see if any information has been entered into the city field in the form. If no information has been entered, the user's IP address is used to define city, state or country, latitude, and longitude. If, on the other hand, either the domestic city or the international city fields have been used as part of a search, the page will recognize that and store those input values in the appropriate variables, which are then sent to the appropriate MapQuest API call for geocoding. Two MapQuest calls had to be made: one using city and state and one using city and country.

With values in place for latitude and longitude, the next step was to call the Dark Sky API forecast endpoint. Despite its name, the forecast endpoint contains both current observations and forecast information. Dark Sky returns a large amount of data. After looking through the result options, I chose several pieces of information I felt were the most relevant. Again, with data and design being so intertwined in this project, I was very

cognizant of the limitations of human visual information processing. I wanted the page to contain the most relevant information, without overwhelming the user. I ended up returning the following pieces of weather information:

- Current temperature in degrees Fahrenheit
- Current wind speed in miles per hour
- Current wind gusts in miles per hour
- Current apparent or “feels like” temperature
- Current time
- Time zone
- Current brief weather description for icon display
- Current cloud cover from 0 to 1
- Current weather alerts
- Daily summary
- Forecast summary
- Forecast precipitation probability
- Forecast brief weather description for icon display
- Forecast high temperature in degrees Fahrenheit
- Forecast low temperature in degrees Fahrenheit
- Forecast wind speed in miles per hour
- Forecast wind gusts in miles per hour

With all of this data, it was now time to determine how to display it. The most important pieces of information were the location being queried, the current temperature, the apparent temperature, current conditions, and wind speed. The challenge became how to display the other pieces of data. The forecast was relatively simple. It makes sense to display that type of data in a tabular format. Not only is it easy to format and read, but it is a format

with which many people are familiar in a weather context. For the remaining data, I started looking at each piece in the context of visual opportunities.

An obvious start was to have the background image of the page match the current conditions. Again starting simply, I created images to represent a daytime blue sky, a daytime gray sky, a dusky nighttime sky, and a starry nighttime sky. Using the time and time zone data values, I converted the time to the current location time. I was then able to use the SUNFUNCS built into PHP to return the sunrise and sunset times for the currently displayed latitude and longitude. With the local time, the local sunrise time and the local sunset time, I used conditional formatting to determine whether it was daytime or nighttime and to display either a daytime or nighttime background image (Figure 3.1). In order to display the most appropriate image, the logic also evaluated whether it was clear or cloudy based on the cloud cover. Additionally, if it is a clear day, a sun image will display on the screen. Because I was displaying a sun, I chose to also display a moon. This required me to use latitude and longitude to return the current moon phase from the Unite States Naval Observatory API. Since I was able to return the phases, I created graphics to represent each moon phase so the appropriate moon will display.

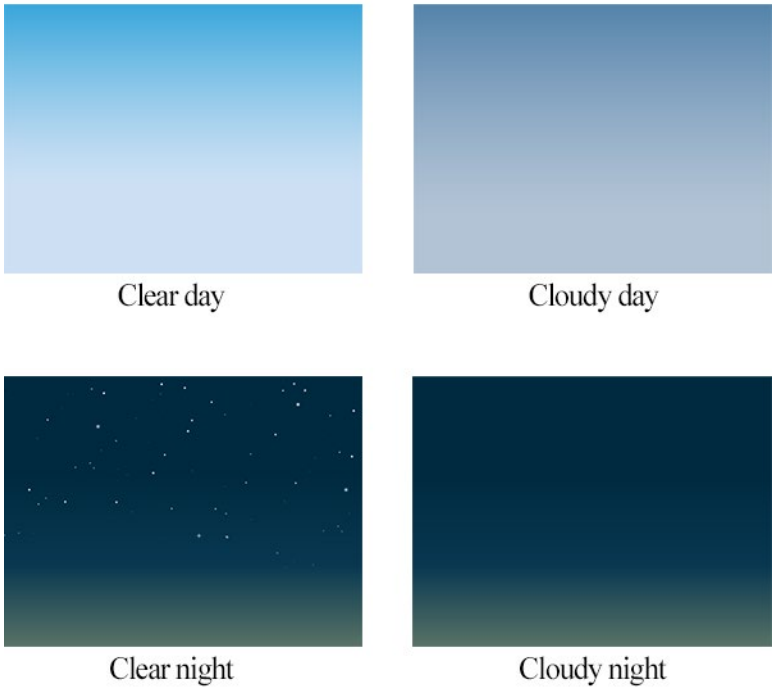


Figure 3.1. *Condition-based backgrounds.*

The next design element I decided to include that could be controlled by data was a cityscape. I already had variables for latitude and longitude and the GeoNames API is able to return population for cities all around the world. Using definitions regarding city sizes from the U.S. Census Bureau, I created simple and generic silhouettes to represent metropolitan, micropolitan, small town, and rural locales (Office of Management and Budget, 2010). Any city with a population smaller that 2,500 people displays as a farm. Locations between 2,500 and 10,000 people will display as a small town. Micropolitan areas are between 10,000 and 70,000 and metropolitan areas make up the remainder.

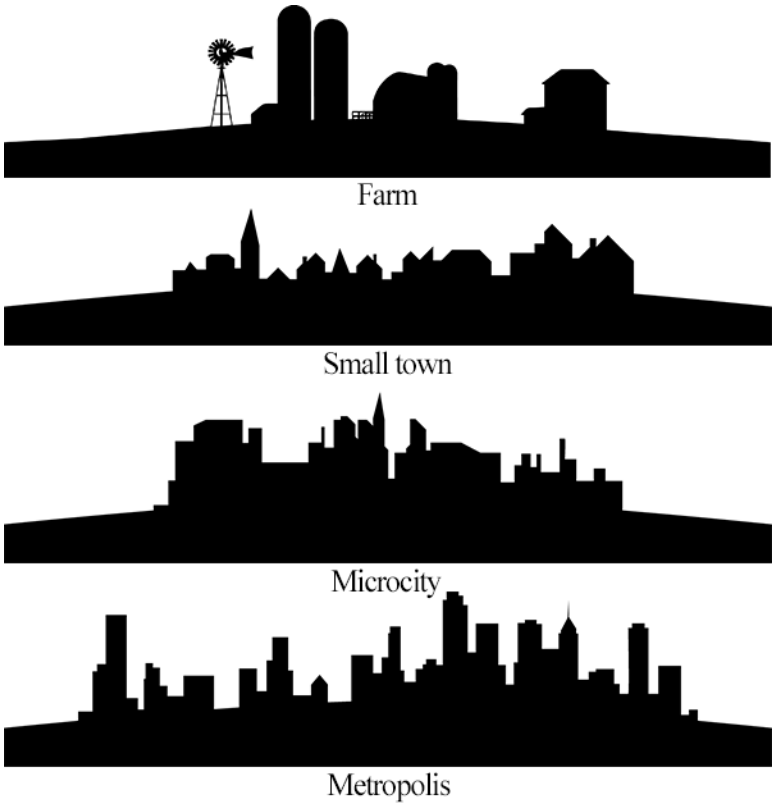


Figure 3.2. *Cityscapes.*

While not a perfect solution, it provides a clear visual representation that doesn't detract from the purpose of the weather page. I later revisited the basic cityscapes and added palm trees to all but the farm. The more "tropical" cityscapes do not just rely on population, but also latitude and only display within 35 degrees north and south of the equator. I have also included location-specific cityscapes for several notable locations, each of which is known for its skyline and/or very distinctive landmarks.

One drawback to using images on the web is their penchant for pixelating. Each of the cityscapes was drawn in Adobe Illustrator as vector graphics. Vector graphics are algorithmically generated and scale up and down without any degradation. In contrast, raster graphics, which are the most common, do not scale well. They are formed by either filling

individual pixels in or not. I deliberately created the images larger than I needed to ensure they would cover all of the available allotted space. Although these were created as vector, they were then converted to raster, as raster images are web standard. I chose the Portable Network Graphics (PNG) format because it supports transparency and the degradation is much less with this format.

I determined the background color was not enough to fully convey current conditions, so I began adding in other elements, based on information contained in the weather data arrays. I added clouds at the top of the screen with the cloud cover image determined by the amount of cloud cover returned in the current conditions. Using CSS animations to control the speed of the clouds, they move across the screen based on the reported wind speed.

The sun graphic animates depending on whether the day is clear or partly cloudy. For clear days, the sun rotates and pulses slightly. On partly cloudy days, it is stationary with the clouds passing over the top of it. The sun's position on the screen is controlled by the time of day and creative math that takes into account the length of the day for a given location and what time the sun is scheduled to be at its zenith for the day.

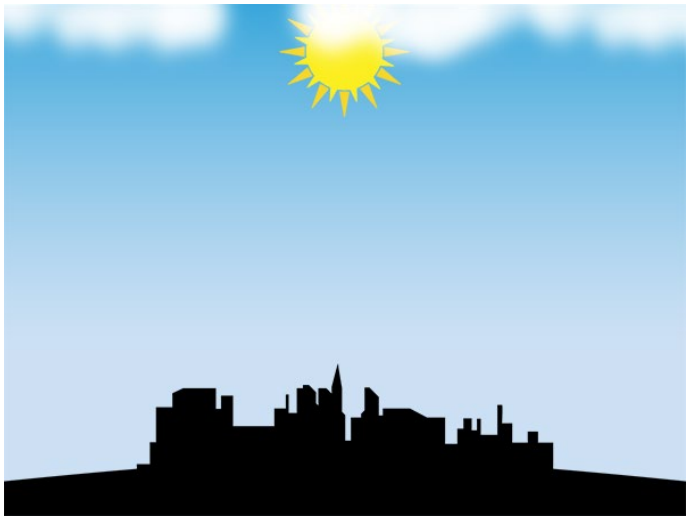


Figure 3.3. *Basic page design.*

Determining that clouds were not enough on days it was either raining or snowing, I utilized CSS animations and the incoming data and logic to add rain or snow to the screen. When thunderstorms are reported, an animated lightning image periodically flashes onto the screen.

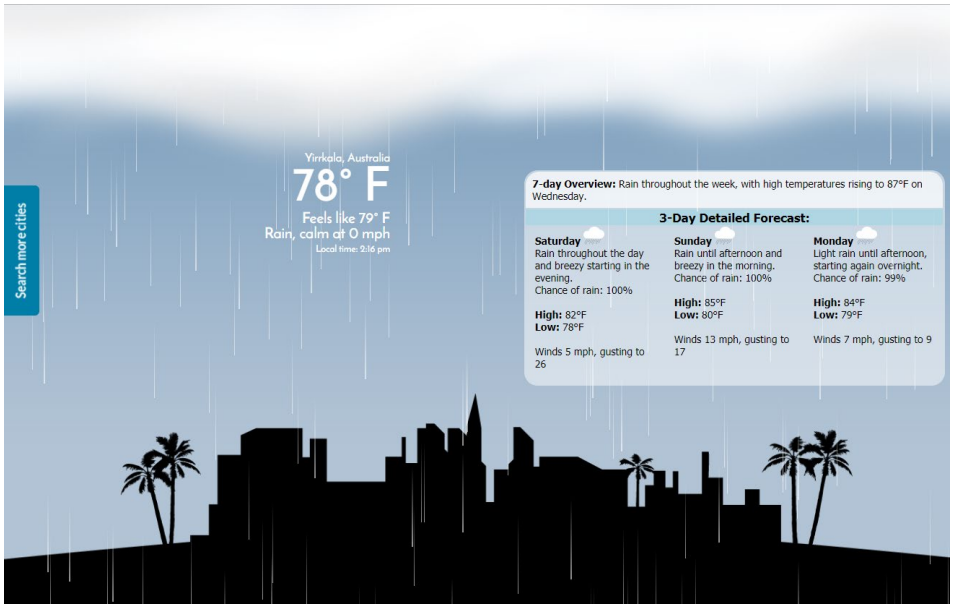


Figure 3.4. *Full layout with rain.*

While this was developed as a web page, I did take into account that it might be viewed on a variety of devices. Using @media queries, which are a part of the CSS3 standard (the most current version of CSS), the browser reads the width of the screen in pixels and automatically adjusts the size and placement of different elements. The text displaying the current conditions and temperature scales to appropriate sizes as the browser size changes. The forecast table moves from the right size of the screen to the bottom of the screen to accommodate mobile devices, without any required user intervention.

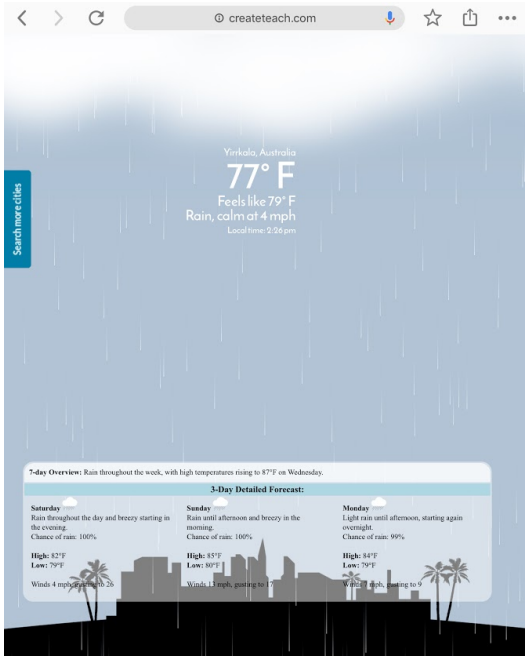


Figure 3.5. *Full layout displayed on an iPad.*

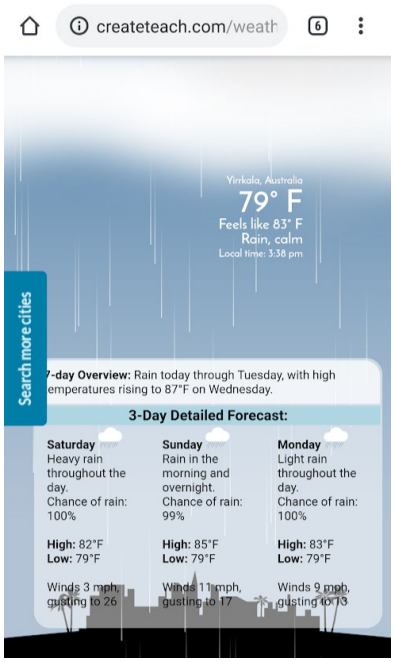


Figure 3.6. *Full layout displayed on an Android phone.*

Archived Project URL:

https://web.archive.org/web/20190416235820/http://createteach.com/weather/

CHAPTER 4. DISCUSSION

Data visualization is a growing discipline. With data visualization, data is returned with the goal of telling a story, that is, the data itself is put out in such a way that it can be visualized for greater understanding. This is often done using a recognized infographic format (e.g., charts, word clouds, networks, etc.). I propose that this creative component employs more of a data illustration approach that could be used beyond a weather display. While still using data, the literal display of individual data points is less of a focus than the overall story experience. Not all data projects would be conducive to this type of reimagining; however, there may be some use cases where using data to control the display is a valid option.

Example 1: User sentiment

One example of how data illustration could be used outside of a weather display could involve user sentiment. There may be some benefit in having an internal page for a business that would monitor social media channels via APIs. Using computational logic, a comment coding scheme, and conditional formatting, the color theme could change based whether aggregate sentiment is positive or negative, while an embedded feed displayed individual customer interactions. If sentiment started to trend negatively, certain menu options could then activate (i.e., become visible) that would allow the business to mitigate the negativity. With this approach, rather than relying on individuals to monitor the channels and report potential issues, anyone with access to that page could effectively check the sentiment at any time.

Beyond business, this type of usage could also be applicable to political candidates. As a mobile app, rather than a web page, the candidate could get an idea about public

sentiment on a more local level. Tying in a component that parsed the comments in order to identify keywords being used in that area would allow the candidate to tailor the message for the locale, rather than repeating the same speeches addressing generic issues.

Conceptually, each of these examples would likely involve multiple APIs and it has the added challenge of developing a coding system for comments. Because of the nature of social media, the coding scheme would also need to be able to apply to memes and animated GIFs.

Example 2: Realtor helpers

A second example for using data illustration may be in the case of realty websites. Currently, shopping for a home involves significant research beyond visiting realty sites. Potential homebuyers often look at crime rates, cost of living, unemployment rates, weather information, and the quality of schools. This requires reading through multiple websites and reports.

A different solution might be to use APIs to provide this type of information and present it in tandem with home listings. For example, if an individual is interested in finding a home in Des Moines, Iowa, a realty site or app would offer a map that not only shows the locations of homes for sale, but also overlays color-coded proximity circles indicating, for instance, how safe a neighborhood is based on crime reports. It could also overlay where schools are located, along with color-coding based on each school's rating. This would be an easy-to-understand way to allow homebuyers to visually assess the neighborhood for any home in which they are interested.

Upon viewing individual home listings, the APIs could provide information allowing for the display of icons that provide information regarding the area: an icon for a family, an icon indicating average income for the area, or an icon indicating how safe the area is. Rather

than presenting statistics, which often mean little or are difficult for a viewer to conceptualize, using the data to control the display of colors and icons could provide a significantly increased amount of information without overwhelming the viewer.

Evaluating this project

This project started as an exercise in data-driven design, with the goal of seeing if there was a limit to what could be controlled with data. By incorporating several different data sources and connecting to them using live APIs to ensure the data is always fresh, I was able to design a weather display that was entirely controlled by the incoming data. Although there are images in use, programming logic is used to determine which image to display in various locations based on data values being returned. In the end, I was surprised at how many elements were able to be controlled in this manner.

There are some limitations to this project. It is currently only available as a web page, not as a native app as were the others that were evaluated. Because it uses free data sources, there are limitations on the number of calls to the data that can be made in a 24-hour period. As I discovered, free data sources do sometimes disappear with little warning, as was the case with this project, and it can be difficult to locate quality APIs that are available at no charge. Not all data sources organize their data in the same way, so if a source is no longer available, major code rewrites are not only possible but likely.

This project is also constrained to a single page with limited data being returned. In order to fully explore this project, I believe it would be necessary to have a better understanding of what a majority of users expect out of a weather app. The visuals used in the project were not the result of any type of decision justified by user preferences; rather the visuals were selected simply because they are what I wanted to use. For the select locations that have custom cityscapes, those were chosen somewhat arbitrarily by me.

One omission that does not fit with best practices in web development is the lack of error checking. Generally, code that is intended to display data from a data source should have a fallback in the event that something goes wrong. With this project, I planned for it, but it proved more difficult than expected. The problem lies with the APIs. If I search for a particular location, even if it is well known, there is a chance that the geocoding APIs will return information, just not always the correct information. For instance, if I search for Tokyo, Japan, the geocode may read just one location within Tokyo (e.g., a bus station, a shopping mall, etc.) and return a population of zero. In this case, the cityscape will not display, but weather data will because those pieces are coming from different data sources. Ideally, a default cityscape would appear, regardless of population. In the event no weather data can be displayed, the default error message is ugly and does not tell the average user much of anything. This is something that should be addressed.

This type of project is one that I am not certain actually has an end. There are so many elements that could still be added. For example, with the sunrise and sunset data, the background could conceivably change to represent dawn and dusk. Through the use of this logic combined with CSS animation, there is the potential to have one background fade into another in near real-time. I would also like to have the wind affect more than just the cloud speed. I contemplated, but did not attempt, to have the rain and snow angles change in relation to the wind direction and speed. On very windy days, leaves, hats or newspapers could blow across the screen. If snow is forecast to accumulate, the cityscapes could have the "snow" build up on them. Astronomical events, such as eclipses, could also be incorporated.

User controls could be added that would allow the user to toggle between metric and imperial units. Controls could also be added to allow the user to increase the text size, without disrupting the layout.

From an in-class introduction to accessing live data to its current state as a completely data-driven design, this project is now employed as an example in the course from which it was borne. Students are challenged to identify which parts of the display are data or controlled by data. The goal of the exercise is to encourage students to think about data not just as numbers and words to be displayed on a page, but as controls for display elements. Coming from a background where data is either forced to fit a design or designs are limited to emphasize data, I would now argue that data can actually help free design by forging creative relationships between data and design elements.

REFERENCES

Achour, M., Betz, F., Dovgal, A., Lopes, N., Magnusson, H., Richter, G., … Vrana, J. (2019). PHP Manual. Retrieved from php website: https://www.php.net/manual/en/

Barraclough, D. (2019). Web Designer vs Web Developer: Do You Know The Difference? Retrieved from Website Builder Expert website: https://www.websitebuilderexpert.com/designing-websites/web-designer-vs-web-developer/

Buxton, B. (2007). *Sketching User Experiences: Getting the Design Right and the Right Design*. San Francisco: Morgan Kaufmann.

Christensson, P. (2006). PHP definition. Retrieved April 1, 2019, from https://techterms.com/definition/php

Cousins, C. (2013). What's the Difference Between a Web Designer and Web Developer? Retrieved from designmodo website: https://designmodo.com/designer-vs-developer/

Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin.

Gibson, J. J. (2015). The Theory of Affordances. In *The Ecological Approach to Visual Perception* (Classic ed, pp. 119–136). New York: Taylor & Francis.

NewGenApps. (2018). Web Designers Vs. Web Developers - Do You Know the Difference? Retrieved from NewGenApps website: https://www.newgenapps.com/blog/web-designers-vs-web-developers-the-difference

Norman, D. A. (1988). *The Design of Everyday Things* (2002 editi). https://doi.org/10.2307/3106094

Office of Management and Budget. (2010). 2010 Standards for Delineating Metropolitan and Micropolitan Statistical Areas. *Federal Register*, *75*(123), 37246–37252.

Perez, S. (2015). Consumers Spend 85% Of Time On Smartphones In Apps, But Only 5 Apps See Heavy Use. Retrieved March 31, 2019, from Tech Crunch website: https://techcrunch.com/2015/06/22/consumers-spend-85-of-time-on-smartphones-in-apps-but-only-5-apps-see-heavy-use/

Refsnes Data. (n.d.). JSON - Introduction. Retrieved from w3schools.com website: https://www.w3schools.com/js/js_json_intro.asp

World Wide Web Consortium. (n.d.). Standards. Retrieved from W3C website: https://www.w3.org/standards/