Shengyi Huang

Dr. Chris Healy

Traffic Data Analysis for Better Travel Scheduling

Furman Computer Science Department

Aug 26, 2017

Furman Advantage Program Reflection Essay

Our research project successfully produces a server software named StreetTraffic that collects and analyzes traffic flow data. First, it crawls traffic flow data of some given regions or routes defined by the user; then, it produces a web page in which the user can visualize the traffic flow history of a given route within the region on which he/she crawled data. This empowers the user to visualize and understand the historical traffic pattern of his/her interested route, which could be valuable to commuters or someone who wants to plan a trip.

The project is open sourced and currently live on Github (<https://github.com/streettraffic/streettraffic> it also provides some nice GIF animations of StreetTraffic). It is also published in PyPi (The Python Package Index, <https://pypi.org/project/streettraffic/>), which means anyone who uses Python programming language can easily installed StreetTraffic.

What Did We Solve?

First, StreetTraffic provides a good solution to collect historical traffic flow data, which is, as far as we know, *unprecedented* in the open source community. Even though we are in 2017 as more and more data are publicly available, historical traffic flow data remains intangible for regular user. For example, the biggest traffic flow data collector, Google, obfuscates their code and does not provide any API to download traffic data. It only provides a raster traffic layer (<https://developers.google.com/maps/documentation/javascript/examples/layer-traffic>), which is a layer made of image pixels rather than numerical traffic data. Another important data collector is HERE, who provides both raster traffic layer as well as numerical traffic layer (The latter kind of layer/data is what we are interested). This is surprising because numerical traffic data seems to be an important business data source; some companies such as INRIX depends on it to make business(<http://inrix.com/products/traffic/>). I should guess this is also why Google does not publicly provide numerical traffic data. Despite the generosity of HERE, it only provides real-time numerical traffic data and does not provide any historical traffic data, and this is where StreetTraffic is helpful.

StreetTraffic allows you to collect real-time traffic data from HERE and store it in RethinkDB database. This could open different kinds of research possibilities since historical traffic flow data was not publicly available before. With StreetTraffic, researchers can easily setup a server to crawl traffic flow data and potentially answer some traffic flow related questions. For example, one of our question was “would two different routes in the same city experience the same traffic pattern”. Since we crawled the traffic flow data of Atlanta, we wrote some simple scripts that could answer that question. Basically, we wrote a script to randomly generate 100 routes and pick the longest 20 routes and plot the mean and maximum Jamming Factor throughout 24 hours for each one of those routes, as shown in Fig. A. The results shows that some long routes in Atlanta City could experience twice as worse traffic than some other routes, therefore different routes in the same city *could* experience drastically different traffic pattern.



Fig. A. The Plotting of Mean and Maximum Jamming Factor of 20 Longest Routes (Out of 100 Randomly Generated Routes within Atlanta City). For more information, refer to <https://github.com/streettraffic/streettraffic/blob/develop/streettraffic/research/multiple_routes_analysis/Multiple_routes_analysis.ipynb>

Second, StreetTraffic enables the user to visualize the historical traffic pattern of his/her interested route so that he/she can make better travel plan as shown in Fig. B. Basically, StreetTraffic comes with a built-in admin web UI where the user could register route, cities, and start collecting traffic data. The web UI also enables the user to randomly sample points within a geospatial region and query the average Jamming Factor (a measurement of traffic flow with 10 being the worst and 0 being free flow) of the geospatial region, as shown in Fig. C. This is just a starting point, of course, since most of our analytics are somewhat superficial. A user could write some more complex analytics based on our APIs (<http://streettraffic.org/docs/modules/index.html>)

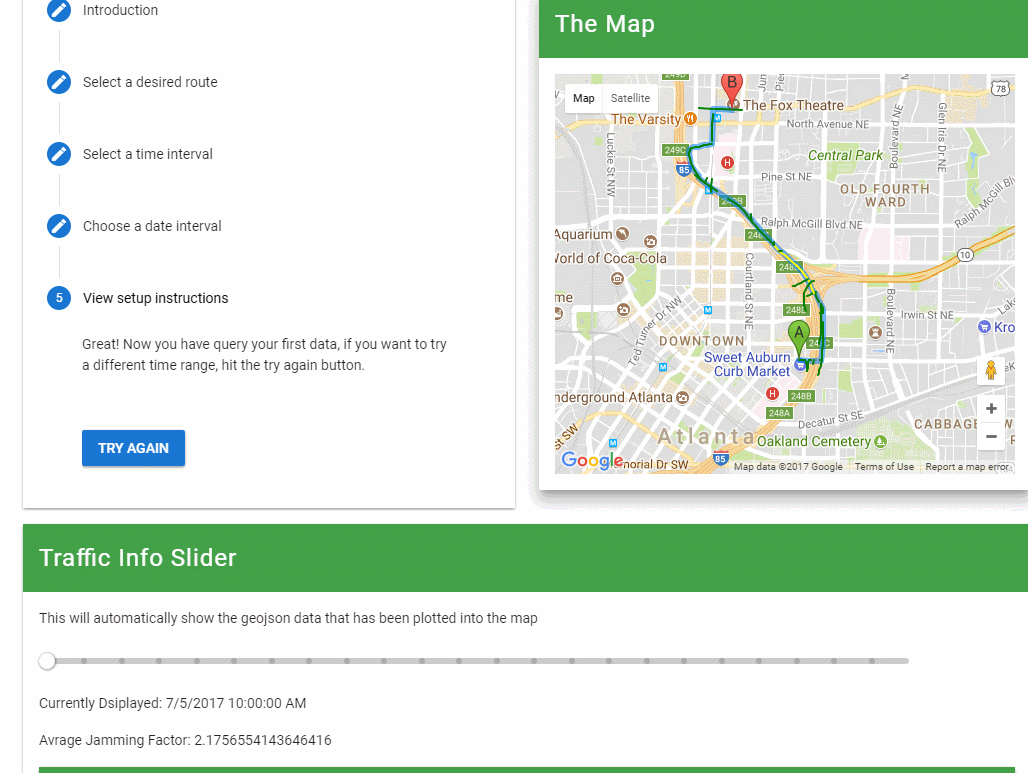


Fig. B. How StreetTraffic Visualize Traffic Flow History. See an animated version at

<https://raw.githubusercontent.com/streettraffic/streettraffic/develop/docs/source/_static/traffic_slider.gif>

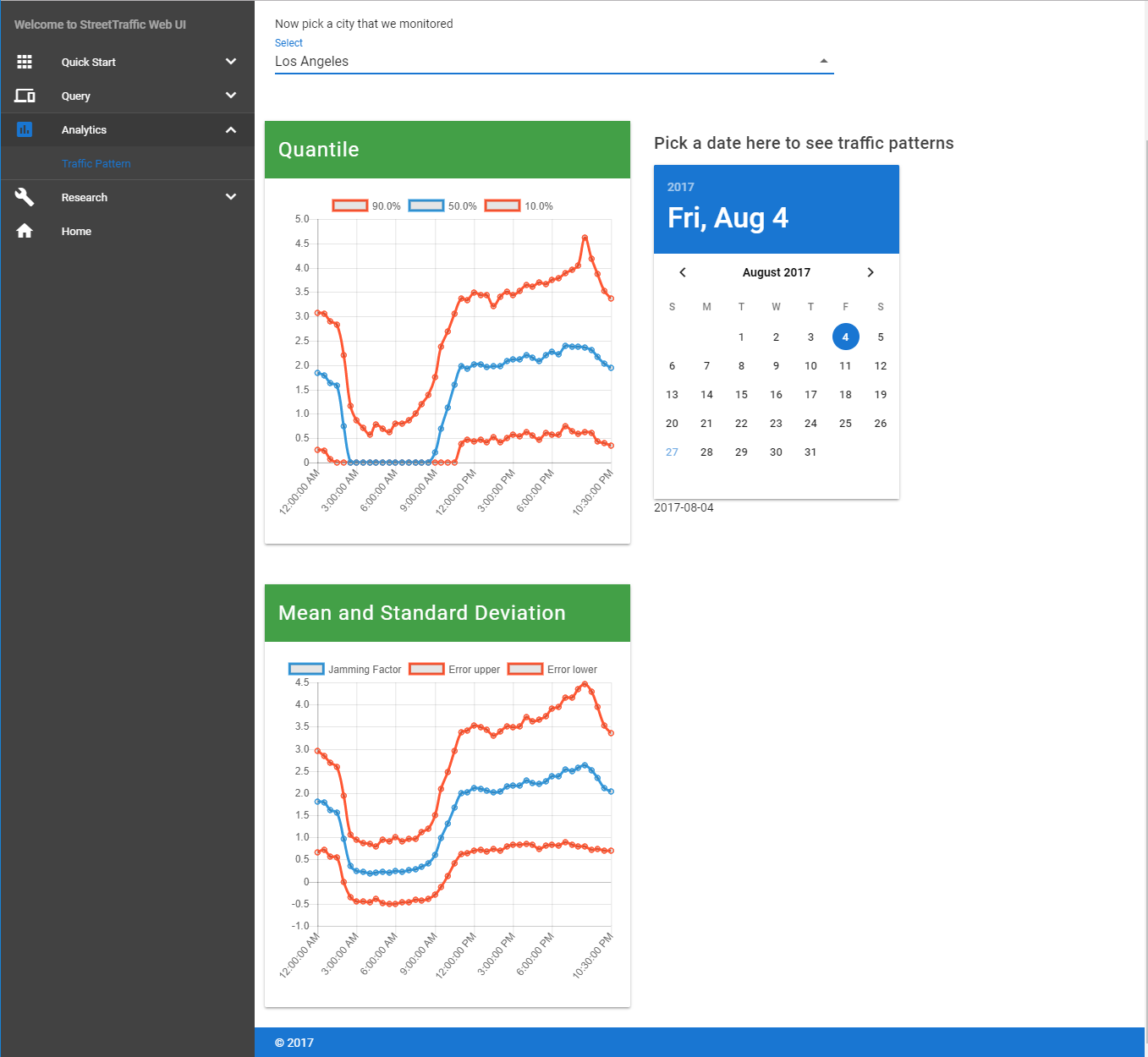


Fig. C. StreetTraffic Visualize Traffic Pattern of Los Angeles

What Have I Learned

This summer has been the most productive one that I have ever had. I have learned so much about software engineering, web programming, and doing research.

Software Engineering

First, I have learned and applied one of the most important skill of software engineering: unit-testing. Writing unit-test means you would write a verification function for an underlying function such that the verification function will tell you whether the underlying function works as expected. For example, if you write a function to fetch text data from database, a unit-testing function will verify that you fetch the correct text data. By doing unit-testing for all the major function you wrote, you can ensure the quality of your code such that you will have confidence when modifying the code. The unit-testing process is also automated as indicated in Fig. 1.

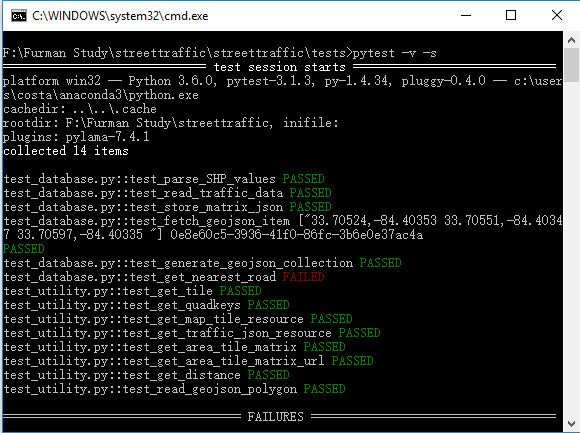


Fig. 1. Running StreetTraffic Unit-test Cases with PyTest

I have written 14 test cases that ensures the building block of my program will run correctly. This makes me feel comfortable to write larger coding project. Before the summer, I felt unmaintainable and uncomfortable after only writing 1,000 lines of code, but with the help of unit testing, I feel can manage to write 10,000 lines of code without falling into chaos.

Second, using Git-related tools helped me to manage larger code base. Git is a version control software that allows you to store your entire working history such that you can always “roll back” to history if something goes wrong. Git is essential for building large project. Without Git, you have to manually version your code, which was what happened to me before the summer as shown in Fig. 2 where I manually gave a version number to every file in my other project. It soon became unmaintainable. This reminds of a joke about bad programming: “After I finished programming, only Gods and I know the meaning of my program. But after 10 days, only God knows my program.” This was not only funny, but also realistic as I had to spend extra hours just to understand my work. Fortunately, after using Git, everything is a lot better as I can give a meaningful summary message of what I have written and I therefore don’t need to manually version my program, as indicated by Fig. 3. I can always “go back in history” to see what I have written. Furthermore, Git also allows me to collaborate with other people with ease since I can see other people’s code history. This is another reason why I feel comfortable managing projects that has a level of 10,000 lines of code.

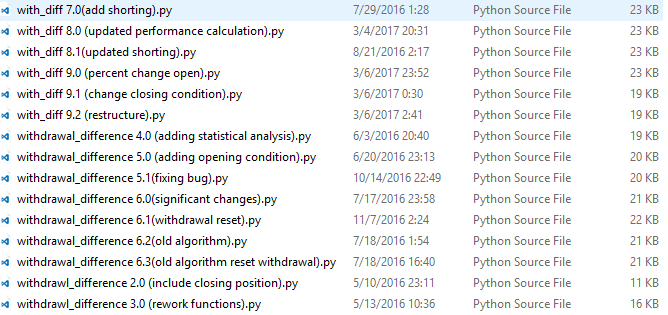


Fig. 2. Unmaintainable Manual Version Control

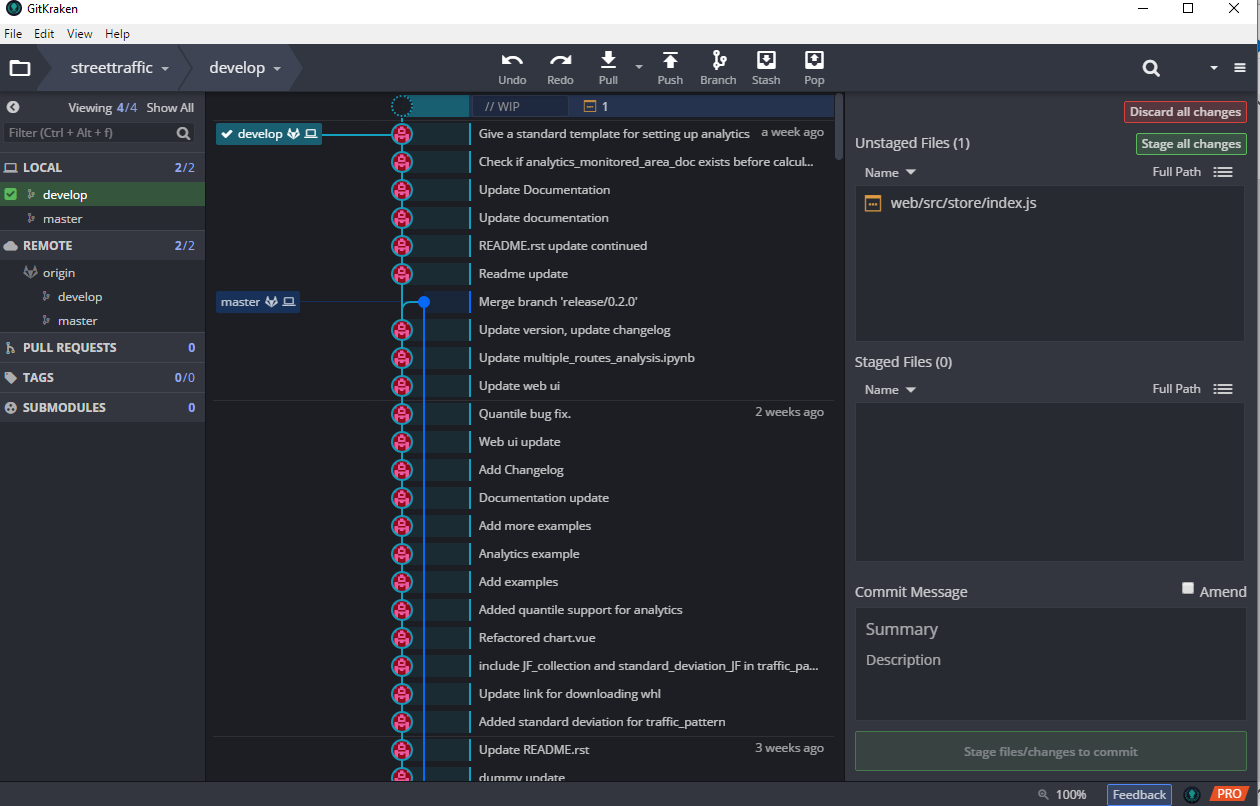


Fig. 3. Industry-level Version Control: Git

Last but not least, I have learned to write *beautiful* documentation in Python. Every modern software, especially the open-sourced ones, has a documentation if it is large enough. It is essential for providing useful introduction to its user and outlining the program for potential contributors. After much comparison, I chose to write Google style Python Docstring (<http://www.sphinx-doc.org/en/stable/ext/example_google.html>), which allows me to write much cleaner code. Furthermore, I can use Sphinx (<http://www.sphinx-doc.org/en/stable/index.html>) to automatically generate an API documentation of my code as shown in Fig. 4. This would incredibly helpful to anyone who wants to read and maintain my code, and it also improves my productivity in the long run.

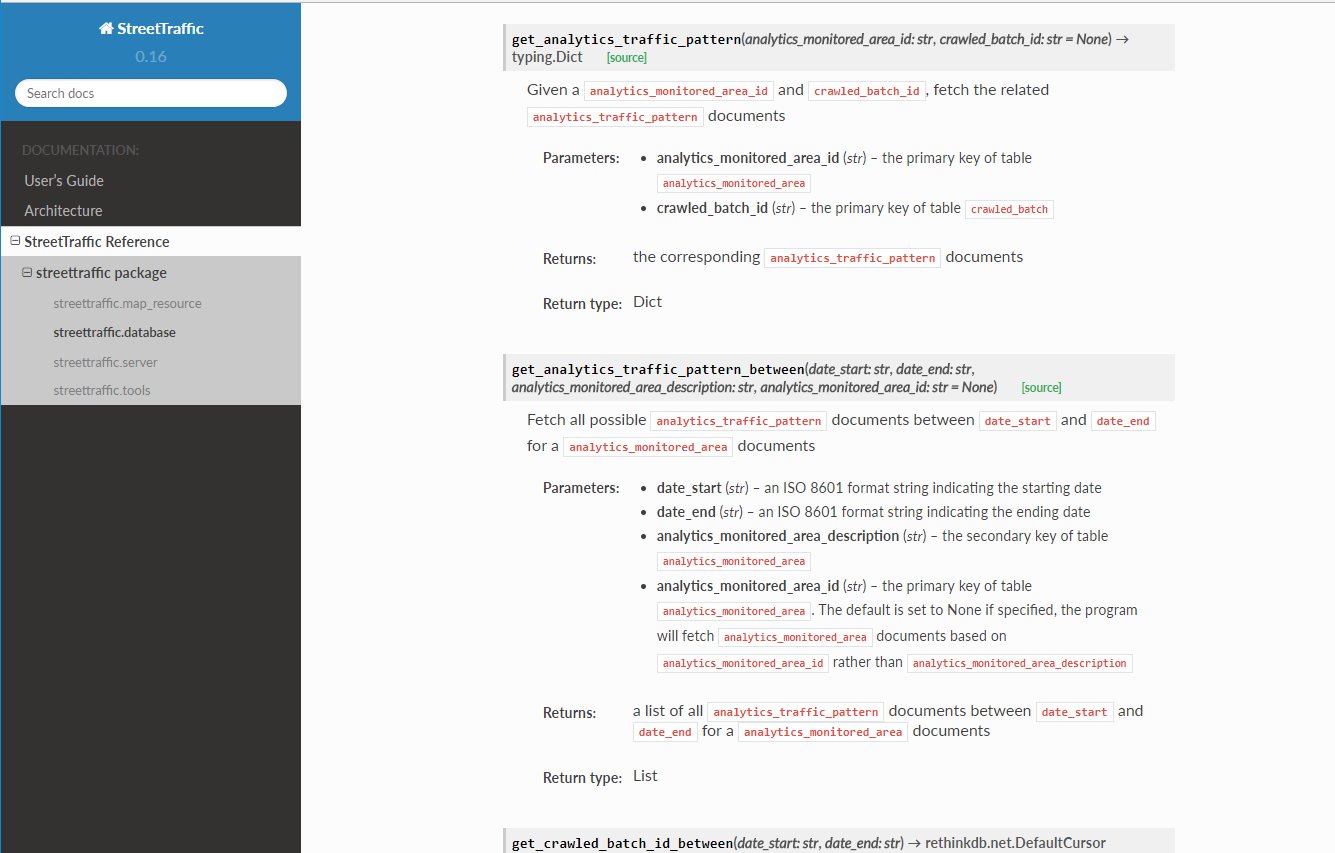


Fig. 4. Industry-level Documentation Generated by Sphinx. It is currently live at <http://streettraffic.org/docs/modules/index.html>

In conclusion, during the research period, I have learned and applied many best practices in software engineering such as unit-testing, version control, and automatic documentation. This is an invaluable experience that has prepared me to write larger codebases and projects. Generally speaking, it has made me a much better programmer.

Web Programming

Being able to elegantly communicate ideas has always been my dream. Some pieces of information are always presented in a better way. Specifically, some of them are presented much more visually appealing than the others. For example, when I tried to learn Riemann Hypothesis at the first place, the introduction video from 3Blue1Brown (as indicated in Fig. 5) made a everlasting impression to me: the elegant and beautiful animation, the peaceful and melodious music, and the clear and interactive explanation. This made me to ponder the beauty of Mathematics. While in comparison, there are many introduction Riemann Hypothesis paper or articles that do not have animations at all. Despite they are all trying to convey the same information, despite I really appreciate their effort, and despite I understand if I spend efforts to read their articles I might experience the same intellectual joy, I have to admit I am absolutely fond of the style of 3Blue1Brown.

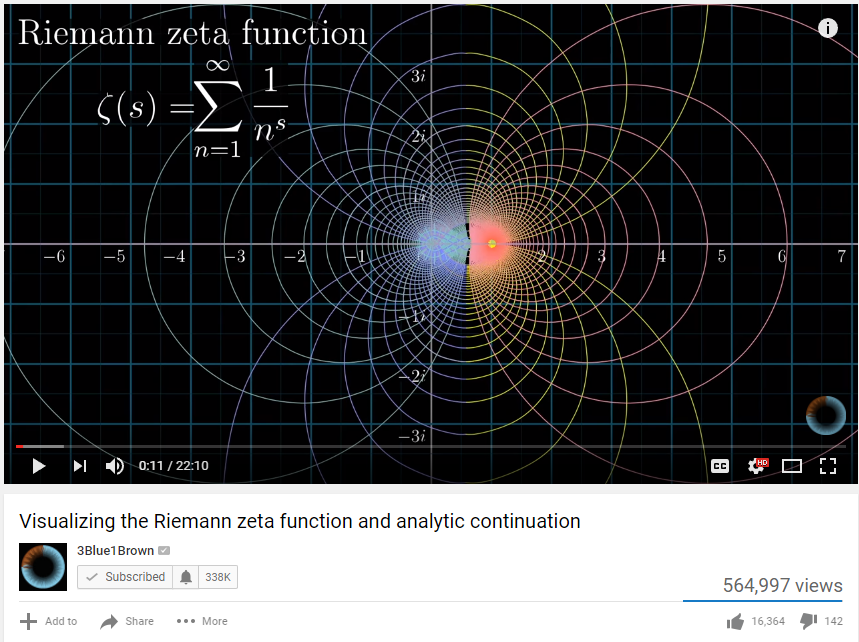


Fig. 5. Visualizing the Riemann zeta function and analytic continuation (<https://youtu.be/sD0NjbwqlYw>)

How does the anecdote relate to web programming? Well, some good front-end web designs elegantly convey ideas. For example, Facebook News Feed (<https://newsfeed.fb.com/>) is almost like telling a story, and Jet Edge (<https://www.flyjetedge.com/>) gives an interactive experience of their its plane. The front-end web design has some valuable skills to learn and I have been trying to learn it so that I could convey my ideas well.

During the summer, I tried to learn the front-end web programming, which is basically website design and website interactivity. I have learned some of the major CSS frameworks like Bootstrap, UIKIT, Semantic UI and etc, then I realize the modern web programming workflow has somehow become incredibly complex. There is an interesting and ironic article that might conveys some feeling about the modern front-end web programming: *How it feels to learn JavaScript in 2016* (<https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f>). Basically, you have to learn a huge collection of JavaScript libraries that are interdependent and somehow messed up together. Nevertheless, I worked hard to understand those concepts and at some point, I was finally over the steep learning curve. After that, I could create some relatively beautiful front-end design without too much difficulties. For example, I created the website design in Fig. 6 within an hour.

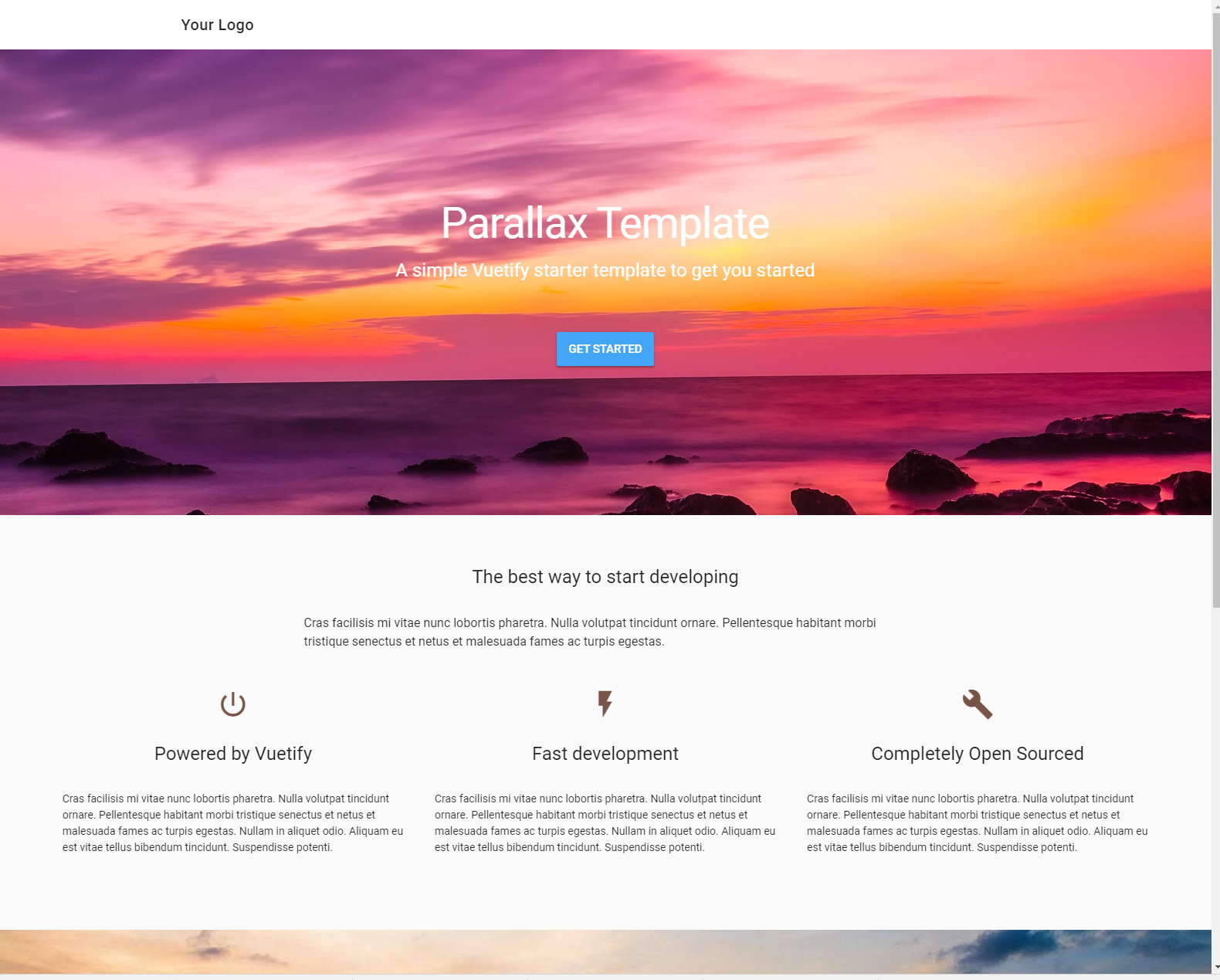


Fig. 6. A Parallax Website Written in Vuetify (<https://codepen.io/vwxyzjn/full/LjrVzW/>)

After learning front-end, I have also dived into back-end programming (also known as server-side programming). I tried to compare different back-end languages (Python, Java, Go, Php) and their related web framework (Flask, Play, Gin, Laravel). Each one of them has their own advantage and shortfalls. For example, Python is notorious for its GIL (Global Interpreter Lock) because GIL cripples Python’s ability to do multi-threaded programming; Go is famous for its concurrency model (so called green-thread or M:N threading) because you can write very clear and understandable concurrent code in Go, while in other languages such as JavaScript, concurrent programming is somewhat more difficult.

What is interesting for me is that at the first place, I thought I only took what is needed without extra complexities, but soon I realized some of those best practices are really good in the long-run and I just kept googling for different solutions, languages, and technologies. I found myself excited when I saw comparison articles between different technologies. And at some point, I found myself embracing various complex technologies without too much difficulties.

In conclusion, learning front-end and back-end web programming is a non-trivial effort in 2017 as there are so many complicated options and technologies available for you. Fortunately, I was able to work hard and learn many of those concepts and it was a tremendously important experience.

Doing Research

One of the biggest gain from working with Dr. Healy is that I have realized one of my shortfall, which is the ability to ask research questions. I found myself interested in researching different technologies like playing with toys but I seem to have some difficulties with coming up with some interesting research question. For example, the question of “would two different routes in the same city experience the same traffic pattern” came from Dr. Healy, and I would probably never have thought of it. Furthermore, when I wrote the script to randomly generate 100 routes for analysis, he mentioned that I should take the longest 20 so that some extremely short routes would be eliminated from our samples.

Another example was how I generated the traffic pattern of a given city. At the first place, there were neither standard deviation nor quantile section on the Fig. C. I only generated the mean Jamming Factor of a given geospatial area. Nevertheless, after we finished the experiment that answers “would two different routes in the same city experience the same traffic pattern”, we realize we should generate a standard deviation for traffic pattern because it will tell the user about the error range of the mean. After finishing that, Dr. Healy also suggested I add a quantile section that picked the 90%, 50%, and 10% of the Jamming Factor since the mean and standard deviation section even showed negative Jamming Factor, which is theoretically possible but nonsense in real life.

Dr. Healy has helped me tremendously in understanding and doing research. He is much experienced and can give me research questions from statistical and innovative perspective. This ability to ask questions from a researcher’s perspective is something I would like to practice and learn more about. I would also want to learn more about probability and statistics so that I could understand data results in a better way.

Final Notes

The summer research experience at Furman was an invaluable one for me. I have learned much and feel more comfortable of being a researcher and programmer. I really appreciate the help from Dr. Healy as well as the support from Furman Advantage program.