JavaScript, Laravel, Microservices, DAX

CODE

NOV
DEC
2021

codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 8.95 Can $ 11.95

Bokeh

Understanding Core Data Structures and Algorithms

Exploring Laravel with Google Cloud

Building Resilient Microservices

1 1>
7 25274 56130 8

Microsoft Azure + AI Conference

CO-PRODUCED BY
Microsoft & DEVintersection

DEVintersection Conference

POWERED BY
Microsoft & DEVintersection

SQL Server & Azure SQL Conference

POWERED BY
Microsoft & NextGen

We invite you to join your favorite Microsoft Leaders Scott Guthrie, EVP Cloud & AI, Rohan Kumar, CVP of Azure Data and Charles Lamanna, CVP of Business Apps & Power Platform as they share their excitement about the power of technology and where the future is taking us. Microsoft engineers and industry experts take a deep dive with you into the many new releases that will empower you to stay competitive.

SCOTT GUTHRIE
Executive Vice President, Cloud + AI Platform, Microsoft

CHARLES LAMANNA
Corporate Vice President, Business Applications & Platform, Microsoft

ROHAN KUMAR
Corporate Vice President, Azure Data, Microsoft

SCOTT HANSELMAN
Partner Program Manager, Microsoft

SCOTT HUNTER
Director of Program Management, Microsoft

JEFF FRITZ
Senior Program Manager, Microsoft

JOHN PAPA
Principal Developer Advocate Lead, Microsoft

KATHLEEN DOLLARD
Principal Program Manager, Microsoft

BOB WARD
Principal Architect Microsoft Azure Data, Microsoft

ANNA HOFFMAN
Data & Applied Scientist, Microsoft

BRI ACHTMAN
Program Manager, Microsoft

DAN WAHLIN
Cloud Developer Advocate Manager, Microsoft

BEC LYONS
Senior Program Manager, Microsoft

MARKUS EGGER
President and Chief Software Architect, EPS Software Corp.

MADS TORGERSEN
Program Manager, C#, Microsoft

and many more!

MGM Grand
Las Vegas, NV

- 200+ in-depth sessions
- 150+ speakers
- 75+ Exhibitors
- 20 Optional full-day workshops
- Exciting evening events

FALL

Dec 7–9, 2021

Workshops Dec 5, 6, 10

LAS VEGAS, NV MGM GRAND

SPRING

Apr 5–7, 2022

Workshops Apr 3, 4, 8

LAS VEGAS, NV MGM GRAND

Health Protocol: Our technology community is excited to get back together as our country re-opens. Please note that proof of vaccination is required to join us in Las Vegas, or a negative Covid-19 test within 72 hours of your arrival. For more information, see our website(s).

OPTIONAL PREMIUM WORKSHOPS *See website for details*

DEC 6
Building for Desktop, Mobile and more with .NET MAUI

Attendees of this MAUI workshop will use and take home an iPad Mini

DEC 5
Virtual Reality, Mixed Reality and More with .NET and Unity

Attendees of this VR workshop will use & take home an Oculus Quest 2

3-day Certification Boot camp: AZ-220 Microsoft Certified Azure IoT Developer
Microsoft certification testing will take place on Day 3.

Here are just a few of the topics covered in sessions and workshops:

.NET 6 • .NET MAUI • BLAZOR • C# 10 • ANGULAR • AZURE • AI
AZURE SQL • MODERN DATA • SQL SERVER • KUBERNETES • DEVOPS
PROJECT DESIGN & UI • SECURITY • MACHINE LEARNING • AZURE LOGIC
BONUS: POWER PLATFORM TRACK

APRIL 2022 REGISTRATION

When you ***REGISTER EARLY for a WORKSHOP PACKAGE,*** you'll receive a choice of hardware or hotel gift card!

Go to **DEVintersection.com** or **AzureAIConf.com** for details.

iPad Mini

Apple Watch SE 44mm case

Surface Earbuds

Surface Headphones

Surface Go 2

Apple Watch SE 40mm case

DEVintersection.com AzureAIConf.com 203-527-4160

Features

Columns

Departments

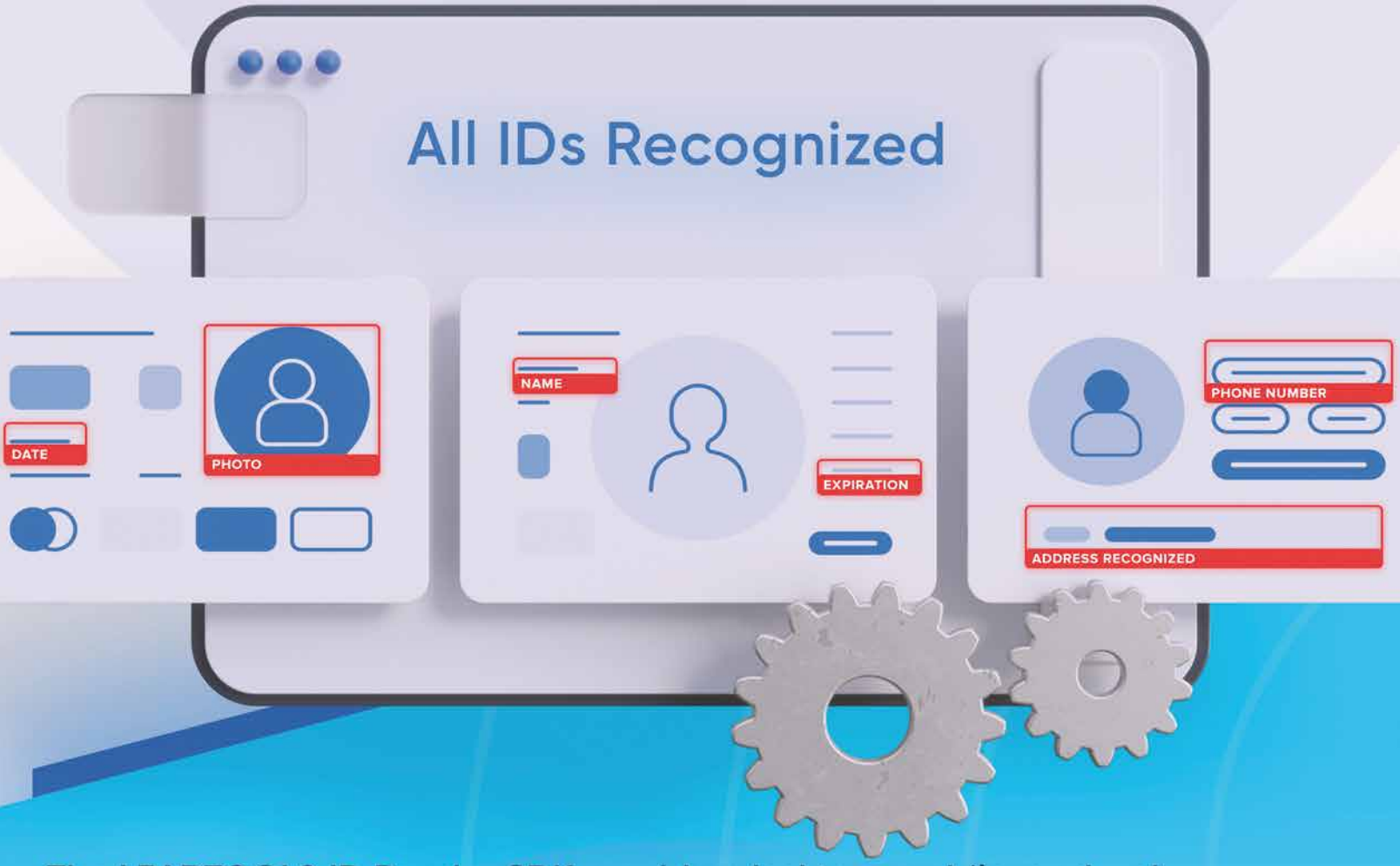
US subscriptions are US $29.99 for one year. Subscriptions outside the US pay $50.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards are accepted. Bill Me option is available only for US subscriptions. Back issues are available. For subscription information, send e-mail to subscriptions@codemag.com or contact Customer Service at 832-717-4445 ext. 9.

Subscribe online at www.codemag.com

CODE Component Developer Magazine (ISSN # 1547-5166) is published bimonthly by EPS Software Corporation, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A. POSTMASTER: Send address changes to CODE Component Developer Magazine, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A.

LEADTOOLS®

ID Reader SDK



The LEADTOOLS ID Reader SDK provides desktop, mobile, and web developers tools to extract data from driver's licenses and similar forms of identification. The state-of-the-art algorithms can extract data in real-world conditions, including glare on lamination and unsteady hands.

EVERY ID | **EASY TO INTEGRATE** | **NO TEMPLATE REQUIRED** | **INDUSTRY LEADER**

Microsoft .NET | Xamarin | Windows | Linux | iOS | macOS | Android | HTML5

Get Started Today

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM

Beating the Doldrums

Welcome to the final issue of 2021. I can say without any doubt that the last two years have been some of the roughest I've ever lived through, and to be 100% honest, I'm tired. I have a LOT of words for how I feel on a given day. Here are just a few: anxious, anxiety ridden, tired, blocked, fragged, happy, elated, morose, blah, rough, exhausted. Every day seems to have its own set of emotions.

The worst days are what I call "black cloud" days. These are the days when a black cloud flies over, ready to drop a steady drum of emotional rain down on me. These days are the worst of them. Some days I catch the cloud flying in early and others, it takes me all day to realize that the cloud has been overhead all along. My preference is to catch it early so I can mitigate the day or to just call it and take a mental health day. Back in the before times, my normal therapy on black cloud days was to shut down the laptop and head to the movies. Movies always lift me out of my doldrums but with COVID, taking a trip to a dark movie theater is far from a stress-free way to relax.

You're probably thinking "Gee, thanks Rod. Way to start my day on such a cheery note." It's my sincere hope that this editorial doesn't stress you out but rather, helps you realize that you're not alone. I know for a fact that I'm not unique in my experience with these feelings, and that helps. It helps to know that other people are experiencing the same feelings and maybe, if we talk about it, we can help each other on our journeys. We're all in this boat together and it's up to us to help each other along.

So how do I deal with these "black cloud" days? I have several things that seem work for me. Here are a few of them.

Over the years, listening to music seems to have worked well in helping me get my stuff together. In the early 2000s, I went through a rough divorce and listening to music helped keep me sane. I have no idea how this works, but it does. Sometimes the fix was a dose of Santana's "Supernatural" or a steady stream of Foo Fighters' "The Colour and the Shape" or simply some Slipknot at 100db. For me, a song or two can change my attitude.

Another form of therapy for me is simply getting into the car and going for long drives. I kind of knew this about myself already, but during COVID, it became more apparent to me that this was a form of therapy. During COVID, I spent many weekends taking long road trips with my wife. We'd get into the car and point it in a random direction and spend the day checking out the sites. My random travels aren't limited to Texas (where I live). My random drives happen elsewhere, as many of my friends can attest. I'm known for my famous "Rod's Reality Tour" of Hollywood. Ask fellow CODE Magazine author John Petersen. He can tell you about these epic drives.

Another form of therapy, and one that's been elusive of late, is vacationing. It's nearly 2022 and I haven't taken a "real" vacation in some time. Like many of you, I had big plans for 2020. We had our passports all ready for a trip to London, Gen-Con, Reaper Con, SDCC, and a Motley Freaking Crue concert! 2020 was going to go down in the books as EPIC. Well, you know what happened. It WAS epic, but 2020 turned into a bummer and here we are. This lack of real downtime has probably had the biggest impact on me and my family. The strange thing is that it took until September of 2021 to come to this realization. I was in the process of cancelling many concerts and conferences that I'd scheduled for fall of 2021 (because of the Delta variant) and I was having a "bitchy" kind of day. It was this bitchy kind of day that gave me the realization that I hadn't had any down time for over a year and a half. I'm trying to rectify this now. It's difficult, because things are really ramping up work-wise as we head into the end of 2021. Wish me luck.

Finally, there's writing. Writing is a form of therapy for me. The biggest help for me was when I discovered Morning Pages. Check out the Jan/Feb 2021 issue of CODE Magazine for my discussion of how Morning Pages helps me. But it's not just the Morning Pages that helps when it comes to writing. These editorials of mine are therapeutic to me as well. When I first started doing these editorials, I was a bit snarky in some of them. One day, Markus Egger messaged me and recommended I use these pages in a more positive way. I took that to heart and have tried my best to use these pages to serve our readers in a positive way. I hope I've succeeded.

As I conclude this editorial, I want to make clear that these are just a few things I use to help get me through. FWIW, I also include many creative endeavors to help get me through. A good example is the Dungeons and Dragons game I DM. This has been an invaluable outlet, as it involves creativity and getting together with friends, albeit virtually for most of the pandemic.

There are numerous ways to help make your mental health better and, in some cases, it might take professional help (something I'm looking into). One of the first steps is to realize that you're not alone and that these feelings are universal. Remember that we're all in this together—and I thank you for being here for me, too!

Rod Paddock
CODE

Xceed WORDS for .NET

Xceed's Words for .NET allows you to **create** or **manipulate** Microsoft Word documents directly from your .NET applications.

Coming soon: Xceed Workbooks for .NET, which allows you to create, manipulate and export Microsoft Excel files.

Xceed

www.xceed.com

Data Structures and Algorithms

I love working in the IT industry. I can't think of any other industry that has had such a massive impact on the planet. I'm not just talking about technological progress. I'm talking about equity, opportunity, and progress. When I was growing up, I didn't have access to world-class information, resources, or people. There was really nobody to guide me on the things I wanted to achieve,

Sahil Malik
www.winsmarts.com
@sahilmalik

Sahil Malik is a Microsoft MVP, INETA speaker, a .NET author, consultant, and trainer.

Sahil loves interacting with fellow geeks in real time. His talks and trainings are full of humor and practical nuggets.

His areas of expertise are cross-platform Mobile app development, Microsoft anything, and security and identity.

and I didn't even know what I wanted to achieve because I couldn't see that far or where to look. You can imagine that things are pretty much the same today back home. For example, somebody in Silicon Valley has a much better opportunity of networking with people in the IT industry. Somebody in Washington DC can perhaps rub shoulders with politicians. Somebody from a wealthy family New York is probably connected to the finance industry. All of them have a better chance then somebody who must walk 15 kilometers each way in a poor village in Africa to get fresh water. Your location matters; your circumstances still matter.

I'm not saying that the industry has solved this problem 100%. But I can argue that with the advent of things such as the Internet, and social networking, putting information and people together is so much more within reach than ever. And because of the pandemic and remote work being the norm, the world is flatter than ever before. It's to the extent that it has created a reverse problem: Information is no longer the issue; the deluge and quality of information is.

What does all this have to do with algorithms? Well, one of the things that I've heard over and over again in the IT business is that you don't need a CS degree to do well in this industry. And indeed, there are lots of examples of very successful people in our industry that have done well without a CS degree. Sadly, in the modern Google-driven development world, it's also becoming very evident that the quality of software has gone down. One of the key reasons for this is a change in focus to getting the task done without focusing on the fundamentals. This may get you to some distance, but it won't let you build world-class systems.

Ask anybody that's been around for a while. The concepts you see in service bus in Azure are the same concepts you see in MSMQ on premises, which are very similar to the concepts around window-messaging in Win32. The concepts around load balancing in the most complex systems are pretty much the same as sharding concepts in a modern distributed database or a complex global caching infrastructure, which are shockingly parallel to any other scenario where consistent hashing is applied.

I thought, therefore, it would be worth it to create a super back-to-basics article, which is language agnostic, and focuses on some basic algorithms that I wish everyone in the IT industry knew, whether or not they have a CS degree.

In this article, I'll go through these concepts one by one, and where applicable, I'll try to explain a practical application for these boring data structures and algorithms.

Array

An array is a data structure consisting of a collection of elements, each identified by an index or key. For example, let's say you want to store some numbers.

```
[1,5,7,0]
```

This could be an array. Think of how this would be represented in memory. You'd find a contiguous block of memory that could hold this array. You'd want to put some dimensions on how big of an object you can put in this array. Let's say you're dealing with integers. And you give yourself enough room that a 32-bit integer is the largest you need to deal with. For this array, you'd need a 32 bits-times-4 size of space. That's not so bad. But what if you wish to resize this array? Let's say you wish to add four more elements to the end. Now you have to find 8 x 32 bits of contiguous memory, and copy stuff around—assuming that the fifth 32-bit block after the fourth element was taken up. This is why, typically, in languages when we allocate arrays, we're forced to pick the size of it, so our program can allocate it and not have to keep hunting for space for it.

Another curious question you may be thinking: What if you need to stuff objects inside the array? Well, then you'd simply store locations (pointers) to where the actual object resides. In this case, the pointer would be 32-bits long. But aren't most computers 64-bit now? Should a pointer be 64-bits long, and how many programs would actually benefit from such a huge addressing space? A 64-bit address space can address 16-exabytes of memory. That is 17,179,869,184 gigabytes. The laptop I'm writing this on has 32 GB of RAM and it's more than enough for most of the programs I need. Should every array by default by 64-bit? Clearly, no. I'm oversimplifying here, but you get the idea.

Array List

As you can see, an array is quite useful, but having to preallocate what I may need ahead of time can be a real pain. In languages such as C++, you have methods such as calloc, malloc, and delete to play with memory. If you want 100% control of what your program is doing, that's indeed the way to go about it. But in the languages most people use (sorry C++), such as Java, C#, etc., we have higher level constructs to help us deal with managing memory in shape of arrays. That concept is an array list. An ArrayList is simply like an array, but you can dynamically increase its size as necessary. For instance, you can create an ArrayList and it gives you four spaces. When you need to add a fifth element, the ArrayList, implemented as a class, hunts for another four blocks somewhere, and manages all this for you. The number "4" is dependent on the framework. Some frameworks, such as .NET, allow you to instantiate an Array List with a predefined initial capacity.

Additionally, an Array list gives you convenient methods for stuff you'd usually do with an array, such as sort, BinarySearch, reverse etc. The code snippet below shows the usage of an ArrayList in C#. Java is quite similar.

```
ArrayList myAL = new ArrayList();
myAL.Add(1);
```

```
myAL.Add("Hello");
myAL.Add(new object());
```

Isn't an ArrayList wonderful? You no longer have to deal with malloc and friends. Umm, not so fast. There's no free ride here.

Behind the scenes, there's a bunch of magic going on, around allocating memory, juggling memory. You don't have to write that code, but you're paying the price for it. Additionally, because an ArrayList can hold heterogenous objects, such as the example above holds a string, a number, and an object, it won't offer you the best performance. Why would that be? Because storing every object requires special consideration. Pointers to objects behave differently than integers do. So you end up making worst-case assumptions, and you pay boxing and unboxing costs.

Most languages therefore offer generic implementations where you're required to pick a type during initialization, and that allows your program to allocate memory more effectively. In C#, this is the List<T> class, which you should almost always prefer over ArrayList. C++ had a very clever way of getting around this problem that was widely used in COM. It was the void** data type. It was a null pointer to a pointer, effectively letting you store anything without paying the additional memory cost. Still, the tradeoff there was that you had to be careful of what was on the other side of that void **, and it was easy to shoot yourself in the foot.

But let's admit it, arrays have one huge advantage. Once your array is set up, accessing any element via an indexer is fast.

Linked List

Arrays are useful, but they suffer from a pretty big disadvantage. As the array increases or decreases in size, you have to do all sorts of memory gymnastics. As you increase the size of the array, newer chunks of memory still need to be allocated. What happens when you delete stuff? Well, you're stuck with the poor choice of either leaving holes in your array, or moving stuff to the left once an element is deleted.

A linked list gets around these problems. It allows you to add elements as needed and delete elements without having to reshuffle your entire data structure. Here's how a linked list is represented in code.

```
public class Node {
    public Node next;
    public Object data;
}

public class LinkedList {
    private Node head;
}
```

As you can see, a LinkedList is simply a placeholder that lets you hold the data you care about, and a pointer to where you can find the next element in this linked list. This looks like **Figure 1**.

Now you can grow this list on demand without having to allocate huge chunks of memory on the fly. For instance, if you were to add a fourth element, you'd simply update element 3's Next pointer to the location of the fourth element.

What happens if you wish to delete an element? For instance, what if I wish to delete element #2? Well, that's easy, you simply repoint element 1's **next** pointer to element 3. This is not computationally intensive.

Figure 1: A LinkedList

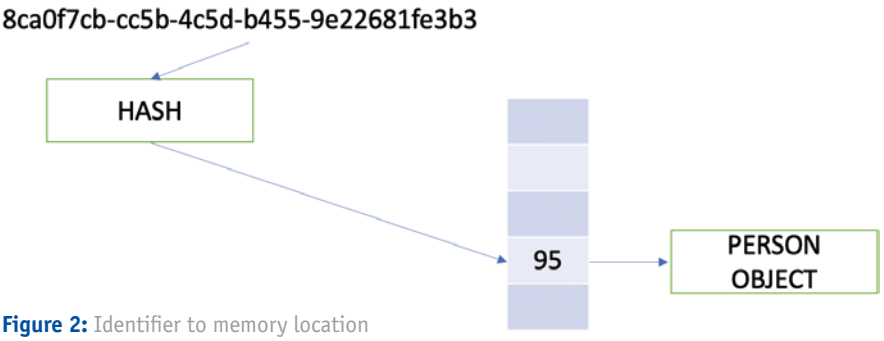


Figure 2: Identifier to memory location

What if you wanted to go backward in the linked list, i.e., find the parent of an element? You can simply add a "previous" pointer. This would be a doubly linked list.

So why don't we always use linked lists? Why do we even bother with arrays? The answer: Traversing a linked list is expensive. To get to element #5949, I have to go through 5948 elements first. But hey. At least I don't have to pay a huge up front and ongoing cost for allocating and managing all this memory.

> Traversing a linked list is expensive.

Where would a linked list be useful? Think of any real-world representation of information. It's all about connections. I'm Sahil. I have a computer, a phone, a tablet. My phone has apps, memory, mail, photos. See? I am traversing a linked list. Actually, I'm traversing a graph, because the photos might be of my computer. More on this shortly.

Hash Table

I'm trying to store key value pairs. For instance, I have (person name, age) as a possible data structure I wish to store. Or maybe I want to store (identifier, person object) as a key value pair. So given an identifier, I want to quickly retrieve the person object.

Well, that's where a hash table fits. Much to my chagrin, it wasn't a table from Amsterdam. Let's first define what a hash is. A hash is a function that you pass into an object and it reliably, for the same object, gives you the same hash every time. Given that, if I were to pick a hash function and pass in an identifier, it would yield me a number where I can store a pointer to the person object.

For example, an identifier of "8ca0f7cb-cc5b-4c5d-b455-9e22681fe3b3" could perhaps resolve to a hash value of 95, and on location #95, I can store a pointer to the person object, as shown in **Figure 2**.

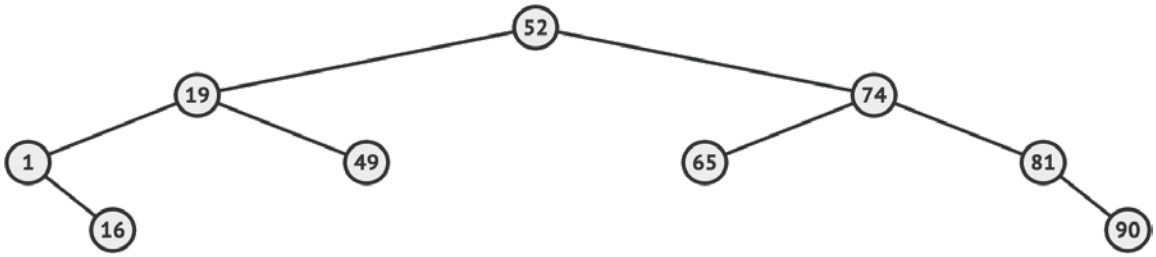


Figure 3: A binary tree

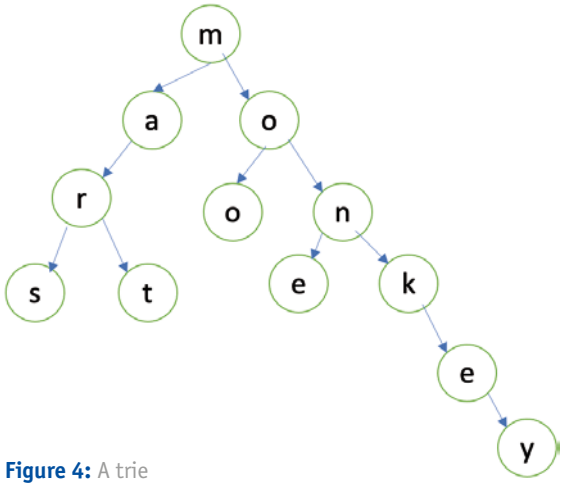


Figure 4: A trie

Problem solved, right? Nope. The problem is, the inputs are seemingly infinite, but the output of the hash function space is finite. So, you're bound to have collisions. There are various strategies to this problem. You could start storing linked lists at individual locations. Or you could use other structures, such as finding an empty bucket closest to the already filled location, or you could have a separate hash on top of a hash to increase the address space.

But either way, in a very large set of data, one advantage is clear: speed. If the number of entries can be roughly predicted and allocated for, you can pick just the right hash function that's the right balance of space and performance, and create a really fast lookup. The disadvantage, of course, is that every object is seemingly random. For instance, you may want to traverse through the objects and where your previous pick could effectively limit your address space, you gain no advantage by using a hash table. For example, I wish to represent all cars. But what if I want to narrow down to Toyota, and then Corolla. The fact that I've picked Toyota should help me speed up my lookup for Corolla. But in a hashtable, all elements are equally fast, or equally slow. And picking a poor hash function can really slow you down.

Now, you may be thinking, why does this all this even matter? Imagine that you're writing a cloud scale system, and every optimization translates to millions of dollars saved—suddenly, all this matters. But let's pick a more mundane example. You're writing a globally distributed system and you need a cache. Caches could be implemented as hashtables and you want to pick the right algorithm to avoid naming collisions or the appropriate data structures to speed up your lookups. What if one of your tasks was the search through your data? For example, you're implementing an autocomplete-like functionality. If I type "car," I want card, carpet, and carry to be matched. I could do this using a hashtable, but would that be the best approach? Hashablets are great when I want consistent or almost consistent performance for each element. It's important to find the right tool for the right job.

Trees

Okay, so hashtables are very useful. But every element is equal to every other element. I wish I had a mechanism to bake some intelligence into my data. For instance, let's say that frequently I need to perform calculations against my data. I have a bunch of invoices, and I wish to easily sort them. Or I want to find the min/max of them. Or do stuff like find the average of all invoices less than $500. Are you aware of a program that lets you do this kind of magic? Of course. Excel does this. All the time.

I have no idea how Excel has been implemented behind the scenes. But I hope in a system optimized for traversing data, you'd use a structure such as trees.

For a moment, glance at the linked list data structure as shown in **Figure 1**. A tree is quite similar, except, in a linked list, you can link to only one other node. In a tree, you can link to multiple other nodes. A binary tree can link to exactly two other nodes. Also, a tree cannot be a loop. This means that element 1 cannot point to element 2 and then to 3, and 3 point back to 1.

Figure 3 shows a binary search tree. As you can see, every element has two possible children. The smaller is on the left, the larger is on the right. One or both elements can be null.

Now, if I gave you a problem, let's say: Find element #49. This problem in a hash table would be computationally expensive. You'd have the run the hash function for 49 (and assume 49 was a big complex object, such as an invoice). Then find the location, deal with collisions, and eventually land on the object and pass it back. What if I made the problem a little more complex: Find me the sum of all invoices less than 52. This would be very expensive in a hashtable.

Let's see how this works in a tree. To find element #49 (assuming it exists), I simply evaluate 49 < 52, and go left. The next element I arrive at is 19. Because 49 > 19, I go right and find what I'm looking for.

How would I do a sum? I'd use recursion or dynamic programming. This makes my job so much simpler. Even better, the amount of time it would take to compute any such problem will also become quite predictable.

Tries

A trie is a specific kind of a tree and it works great with strings. Imagine if you're building the spell check feature in MS Word. You want the feedback to the user to be fast and fluid. Let's say the user types "monkr". This is clearly a spelling mistake. Let's think through the logic here.

First the user types "m". Well, there are plenty of words that start with m. This is not a spelling mistake.

Then the user types "mo". The user may be typing money, most, Monday, monkey, moo, mom. Again, not a spelling mistake so far.

Now the user types, mon. Still, this could be Monday, money, moniker, etc.

Finally, the user types "monk." This is a word. But it could be the prefix for monkey, or monkfish, or monks.

Finally, the user types "monkr"—and we know for sure that this is a spelling mistake.

One thing is clear: at the type of every keystroke, you wish to very quickly evaluate whether or not this is leading to a real word or not. I guess I could have a huge look-up of words, such as a dictionary or a hash table. Every time the user types something, I could search. This is why my MacBook fan runs so fast. My MacBook Pro literally is MacBook hot air. Developers write poor code. I wish I had a data structure like that shown in **Figure 4**.

This kind of structure makes my job so much easier. Now when the user types in "m," I quickly look up the "m" node and pass it back to the application. The next time I receive a key stroke, I expect the application to pass me the node it was working with. So I can quickly look at the children of "m" and easily evaluate "a" and "o" as valid children, but perhaps not "mt" as a valid word combination. The moment I run into an invalid word combination, i.e., no child found, I can show a red squiggly.

Imagine how useful this would be on a cellphone. What if you had approximate matching algorithms? For instance, that touch-type thing on your iPhone can look for word proximity to improve your typing. If you type "m" followed by "r" but you could then expand to the characters immediately around "r" as a possible mistype, "e," "d," "f," "g," and "t," you can very quickly do a lookup in your trie, and immediately know that the user meant "me." You can pair this with even more intelligence by adding probabilistic words and grammatical intelligence. Imagine how cumbersome it would be to write on a touch screen without all this?

It wasn't long ago that we had to deal with all these typos. In fact, in 2008, I remember using a Windows CE phone that had a stylus, and it had none of this intelligence built into it. It took an hour to compose an email. What boggles my mind is that we haven't brought all this intelligence to desktop programs, such as word processing applications, or rebooting IntelliSense to AI IntelliSense in Visual studio. One can dream. Sigh! But have you noticed recently that some email applications have started making use of this? As you compose an email, they prompt you to fill a sentence. A huge time saver, and really hides my poor grammar too.

Graphs

Find me a word as overused as "graph" in our industry. No, I'm not talking about MSGraph or GraphQL, etc. This is a nonlinear data structure called "graph." A graph, like a tree, is a collection of objects, called nodes or vertices.

The big difference between a tree and graph is that in a tree, the rules are a bit more rigid. Specifically with n nodes, you must have exactly n-1 edges, one edge for each parent-child relationship. Each node has exactly one parent. All nodes are reachable from the root and you can always traverse to the parent from any node.

In a graph, it's a bit more freeform. You have a bunch of nodes and a bunch of edges, and you connect them as you wish. Think of it this way: A tree is a specific kind of a graph.

Now this is getting interesting. Isn't information in the real world much more like a graph? Think of streets on a map. They all connect to each other—not like a tree, but more like a mesh, or, what's the word I'm looking for? A graph! What about cooking recipes? Imagine how boring life would be if you always had to think of recipe as a tree.

If you must have your pudding, you must finish your meat, so how can you have your pudding if you don't finish your meat? <g>

Boy, have we heard that before. Real world is much more colorful, and experimental. Graphs represent a much more natural way of information we see and need to represent in our data structures. And you can do all sorts of interesting things, such as adding weights to the edges—imagine how useful that would be when calculating the shortest driving distance between two points on a map. Or how about routing data between servers on the World Wide Web?

Now, how would you go about searching in this graph? For instance, calculating driving directions? This becomes a simple recursive function, where you can ask each child, "hey, do you have a path to such and such," and you keep going until you reach that node. But this would be computationally expensive, you'd have to worry about loops, etc. So you can just do a breadth first search, where you ask all your children if they have a path until such and such node. And then keep storing the results in a stack until you get there.

And for the most commonly requested driving directions, you could pre-cache the results. You could also have varying weights on the edges, like one for speed and one for environmental impact. So you could do cool stuff like, "find me driving directions with the least environmental impact."

Stacks and Queues

Stacks are exactly what they sound like—like a stack of plates but for data. Think of a data structure where you can add items one on top of another. If you wish to remove an item from a stack, you pop an item and the last inserted item is popped off. If you want to add an item, you push an item into the stack—akin to setting a plate on top of the stack.

Now you may wonder where this data structure is actually used. Well, this is used all the time, like when you call a function from another function. When you go from one function to another function, the state of the calling function must be preserved. All the variables of that function are pushed into the stack. When you return, everything can be popped off the stack.

It's useful, in fact, in any backtracking kind of application where, say, you're traversing a graph, and you want to back track your steps. Imagine you did a Google map search and you have driving directions. Now you wish to go step by step and decide to revert back from step 6 back to 5,4,3,2,1. This is an example of a stack.

Queues, on the other hand, are slightly different. Queues are first in, first out. Just like you'd imagine in real-world queues, the person who stands in the queue first gets served first. Queues are incredibly useful in real life. For example, let's say you're creating an ecommerce website. You quickly want to accept all orders as fast as you can while offering the user a very good user interface. Processing the order may take some time. You wish to accept the order and then put the fact that the order has been accepted and the details of the order into a queue. Now, a worker process, or more than one worker process, can start dequeuing orders from that queue in a first in, first out basis.

Expand this a little bit further. The message sent on that queue can have specific characteristics. Those characteristics may define the appropriate recipient or recipients for that message. For example, you can tap into the queue for logging purposes. Or you could say that "this kind of order" shall be processed by "that kind of worker."

You can even offer transactional semantics on a queue. For instance, when you receive an order, you want to make sure that that order is processed. This means that when a worker dequeues a particular message, you want to make sure that their order is fulfilled. Instead of dequeuing a message, the worker can simply lock that message with a timestamp. The worker is required to renew their lock for a given time span. If the lock isn't renewed, you assume that the worker crashed. If the worker is still working on a particular order, the worker simply extends their lease. Essentially, this gives you some confidence that no messages are lost and they're all appropriately processed.

Summary

This article was extra fun. Usually whenever you're learning a concept, you're trying to solve a problem. Almost always, you're learning about some concept offered by Azure or AWS. Or maybe you're programming against an SDK, such as the .NET Framework. Someone else has done a lot of the thinking for us.

There's nothing wrong in targeting and thinking at that level. But if that's all you do, you're bound to make mistakes. It's akin to learning how to drive a car but knowing nothing about how the engine operates. How would you know, when your car is making a funny sound, that you're not driving with the parking brake on?

Yes, we all have such a friend, don't we? I hope in the programming world, this brief introduction to some of the common algorithms and data structures piqued your curiosity about an entire underworld that you want to know about.

I hope you found this article fun, and I hope to show you many other such practical applications in future articles.

Sahil Malik

CODE

Enhance Your MVC Applications Using JavaScript and jQuery: Part 2

In this article, I'm continuing my series on how to enhance the user experience (UX) of your MVC applications, and how to make them faster. In the first article, entitled "Enhance Your MVC Applications Using JavaScript and jQuery: Part 1" (https://www.codemag.com/Article/2109031/Enhance-Your-MVC-Applications-Using-JavaScript-and-jQuery-Part-1), you learned about starting the MVC application, which was coded using all server-side C#. You then added JavaScript and jQuery to avoid post-backs and to enhance the UX in various ways. If you haven't already read that article, I highly recommend that you read it to learn about the application you're enhancing in this series of articles. The previous article contains installation instructions for the MVC application and how to set up the database.

You're going to continue to add additional client-side code to the MVC application to further enhance the UX as you work your way through this article. You'll learn to expand search areas after the user performs a search, hide certain HTML elements when printing a Web page, and create custom jQuery validation rules to enforce business rules on the client-side. Download the sample that accompanies this article and install it, to follow along step-by-step with this article.

The Problem: Duplicate Form Submission Code

In the previous article, you wrote code to respond to the form being submitted and to display a "please wait" message while waiting for the page to return. The problem is that this code is duplicated on the product, customer maintenance, promotional code, and vehicle type pages. If you look at these pages, you'll find the same three lines of code within the $(document).ready() function.

```
$("form").submit(function () {
  mainController.pleaseWait(this);
});
```

The Solution: Move Code to Main Controller

You should move these three lines of code to the **mainController** closure in the site.js file. Open the **wwwroot\js\site.js** file and add a new method to the mainController closure named formSubmit().

```
function formSubmit() {
  $("form").submit(function () {
    pleaseWait(this);
  });
}
```

Make this method public by adding it to the return object of the closure.

```
return {
  "pleaseWait": pleaseWait,
  "disableAllClicks": disableAllClicks,
  "setSearchValues": setSearchValues,
  "isSearchFilledIn": isSearchFilledIn,
  "setSearchArea": setSearchArea,
  "formSubmit": formSubmit
}
```

Open the **Views\CheckOut\Index.cshtml file** and modify the code at the bottom to look like the following code snippet.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();
});
```

Open the **Views\CustomerMaint\CustomerMaintIndex.cshtml** file and modify the code at the bottom to look like the following code snippet.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();
});
```

Open the **Views\Product\ProductIndex.cshtml** file and modify the code at the bottom to look like the following code snippet.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();
});
```

Open the **Views\PromoCode\PromoCodeIndex.cshtml** file and modify the code at the bottom to look like the following code snippet.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();
});
```

Open the **Views\VehicleType\VehicleTypeIndex.cshtml** file and modify the code at the bottom to look like the following code snippet.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();
});
```

Try It Out

Run the sample MVC application and click on the **Admin > Products** menu to display the product page. Expand the "Search for Products" area and fill in the word "Auto" in the Product Name (or Partial) field. Click the Search button and

Paul D. Sheriff
www.pdsa.com
psheriff@pdsa.com

Paul has been in the IT industry over 34 years. In that time, he has successfully assisted hundreds of company's architect software applications to solve their toughest business problems. Paul has been a teacher and mentor through various mediums such as video courses, blogs, articles and speaking engagements at user groups and conferences around the world. Paul has 28 courses in the www.pluralsight.com library (http://www.pluralsight.com/author/paul-sheriff) on topics ranging from LINQ, JavaScript, Angular, MVC, WPF, ADO.NET, jQuery, and Bootstrap.

you should see still the "please wait" message appear. Try out some of the other pages you just changed to ensure that they still display the "please wait" message as well.

The Problem: After Searching, the Search Area Closes

When you ran the MVC application and clicked on the **Admin > Products** menu to display the product page (**Figure 1**), did you notice that when the page returned, the search area was closed? You can see the Filter Applied telling the user what the current filter is, but a better UX is to have the search area automatically open if there are values present in any of the search fields. You could accomplish this functionality with server-side C# code, but a better technique is to add a few lines of JavaScript code.

The Solution: Look for Values in Search Fields

There are many pages that have a search area: the product, customer maintenance, promotional code, and vehicle type pages to be exact. Each page's search area has the same **ID** attribute to identify the element that contains the unique search fields for each page. You need to write code to determine if any search values have been filled in, and whether to expand the card body or not. Part of this code can be made into a set of generic methods and placed into the **mainController** closure located in the site.js file. Open the **wwwroot\js\site.js** file and add a new private variable into the **mainController,** as shown in the following code snippet.

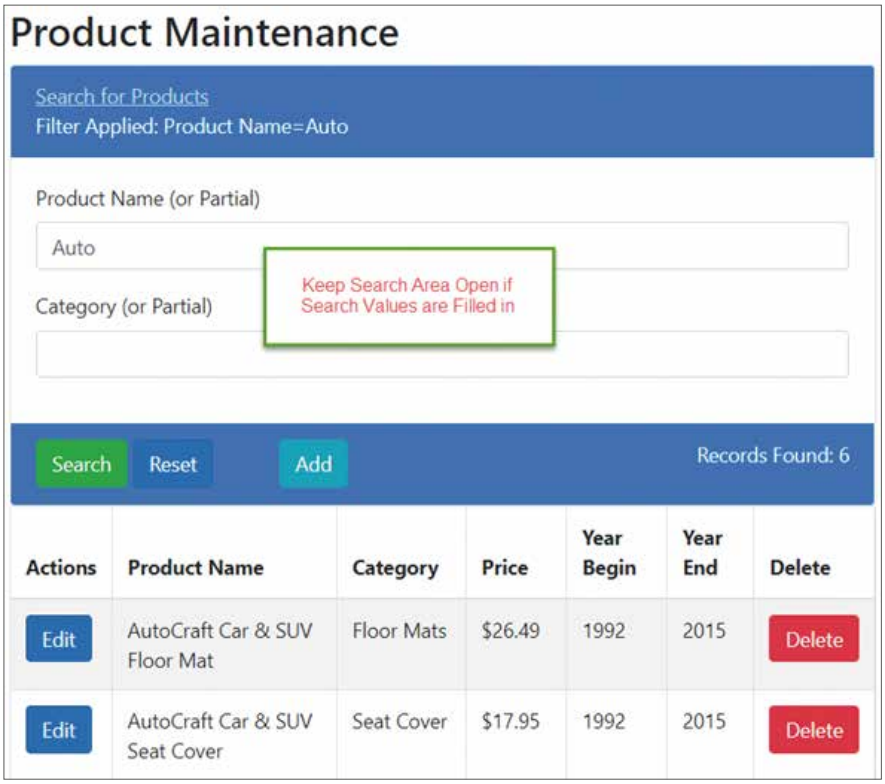


Figure 1: Keep the search area open if the user submitted values to search upon

```
// ************************************
// Private Variables
// ************************************
let searchValues = null;
```

Add three new methods to this closure, as shown in the code snippet below. The first method setSearchValues(), is called by each page after gathering the unique search field values for that page. If any value is filled into the **searchValues** variable, then isSearchFilledIn() returns a true, otherwise it returns a false value. The setSearchArea() method calls the Bootstrap collapse() method on the tag with the **ID** attribute set to "searchBody". The value "show" or "hide" is passed to the collapse() method depending on the return value from the isSearchFilledIn() method.

```
function setSearchValues(value) {
  searchValues = value;
}

function isSearchFilledIn() {
  return searchValues;
}

function setSearchArea() {
  $("#searchBody").collapse(
    isSearchFilledIn() ? "show" : "hide");
}
```

Add each of these three methods to the return object to make them public from the mainController, as shown in the following code snippet.

```
return {
  "pleaseWait": pleaseWait,
  "disableAllClicks": disableAllClicks,
  "setSearchValues": setSearchValues,
  "isSearchFilledIn": isSearchFilledIn,
  "setSearchArea": setSearchArea
}
```

Let's now use these new methods and learn how to set the values from the search area on the Product maintenance page. Open the **Views\Product\ProductIndex.cshtml** file and add a closure named **pageController** immediately after the **'use strict';** statement, as shown in **Listing 1.** You only need to write a single method named setSearchValues() in this controller. This method's purpose is to gather any unique search field values on this page and pass them into

Listing 1: Add a pageController closure to help gather search values and pass them into the mainController

```
let pageController = (function () {
  function setSearchValues() {
    let searchValues =
      $("#SearchEntity_ProductName").val() +
      $("#SearchEntity_Category").val();

    mainController.setSearchValues(searchValues);
  }

  // Expose public functions from closure
  return {
    "setSearchValues": setSearchValues,
    "setSearchArea":
      mainController.setSearchArea,
    "isSearchFilledIn":
      mainController.isSearchFilledIn
  }
})();
```

the **mainController** using its setSearchValues() method. The return object on the **pageController**, exposes this setSearchValues() method, as well as mapping two others to call the methods in the **mainController** directly.

Now that you have the **pageController** written, add two lines of code in the $(document).ready() to call the setSearchValues() method, then the setSearchArea() method on the pageController closure.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();

  // Collapse search area or not?
  pageController.setSearchValues();
  pageController.setSearchArea();
});
```

Try It Out

Click on the **Admin > Products** menu to display the product page. Expand the "Search for Products" area and fill in the word "Auto" in the Product Name (or Partial) field. Click the Search button and now, after the search results are displayed, the "Search for Products" area should be opened so you can see the value you placed in there.

Add Search Functionality to Other Maintenance Pages

Let's now add this same functionality to the other maintenance pages that need it. Open the **Views\CustomerMaint\CustomerMaintIndex.cshtml** file and add a **pageController** closure within the <script> tag, as shown in the code below. Notice that the only thing different from the **pageController** closure you added to the Product Maintenance page is the code within the setSearchValues() method.

```
let pageController = (function () {
  function setSearchValues() {
    mainController.setSearchValues(
      $("#SearchEntity_LastName").val());
  }

  // Expose public functions from closure
  return {
    "setSearchValues": setSearchValues,
    "setSearchArea":
      mainController.setSearchArea,
    "isSearchFilledIn":
      mainController.isSearchFilledIn
  }
})();
```

Add two lines of code in the $(document).ready() to call the setSearchValues() method, then the setSearchArea() method on the pageController closure.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();

  // Collapse search area or not?
  pageController.setSearchValues();
  pageController.setSearchArea();
});
```

Open the **Views\PromoCode\PromoCodeIndex.cshtml** file and add a **pageController** closure within the <script> tag, as shown in the code below. Notice that the only thing different from the **pageController** closure you added to the Customer Maintenance page is the code within the setSearchValues() method.

```
let pageController = (function () {
  function setSearchValues() {
    mainController.setSearchValues(
      $("#SearchEntity_Code").val());
  }

  // Expose public functions from closure
  return {
    "setSearchValues": setSearchValues,
    "setSearchArea":
      mainController.setSearchArea,
    "isSearchFilledIn":
      mainController.isSearchFilledIn
  }
})();
```

Add two lines of code in the $(document).ready() to call the setSearchValues() method, then the setSearchArea() method on the pageController closure.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();

  // Collapse search area or not?
  pageController.setSearchValues();
  pageController.setSearchArea();
});
```

Open the **Views\VehicleType\VehicleTypeIndex.cshtml** file and add a **pageController** closure within the <script> tag, as shown in **Listing 2**. Notice that the only thing different

Listing 2: The setSearchValues() method can be simple, or a little more complicated, depending on the number and type of search fields you use

```
let pageController = (function () {
  function setSearchValues() {
    let searchValues =
      $("#SearchEntity_Make").val() +
      $("#SearchEntity_Model").val();

    let year = $("#SearchEntity_Year").val();

    // Year may contain <-- Select a Year -->
    // ignore that entry
    if (year && !year.startsWith("<--")) {
      searchValues += year;
    }

    mainController.setSearchValues(searchValues);
  }

  // Expose public functions from closure
  return {
    "setSearchValues": setSearchValues,
    "setSearchArea":
      mainController.setSearchArea,
    "isSearchFilledIn":
      mainController.isSearchFilledIn
  }
})();
```

Listing 3: Add a closure to control which search area is open on the shopping cart page

```
let pageController = (function () {
  // ***********************************
  // Private Variables
  // ***********************************
  let searchYearMakeModel = null;
  let searchNameCategory = null;

  // ***********************************
  // Private Functions
  // ***********************************
  function setSearchArea() {
    // Make collapsible regions mutually exclusive
    $("#yearMakeModel").on("show.bs.collapse", function () {
      $("#nameCategory").collapse("hide");
    });
    $("#nameCategory").on("show.bs.collapse", function () {
      $("#yearMakeModel").collapse("hide");
    });

    setYearMakeModel();

    setProductNameCategory();
  }

  function setYearMakeModel() {
  }

  function setProductNameCategory() {
  }

  // ***********************************
  // Public Functions
  // ***********************************
  return {
    "setSearchArea": setSearchArea
  }
})();
```

from the **pageController** closure you added to the Promotional Code Maintenance page is the code within the setSearchValues() method.

This setSearchValues() method is a little different from the previous ones because there's a drop-down list of years. In this method, you concatenate the two drop-down values together, but you should get the value of the selected year independently. The reason is if you haven't filled in any years yet, then a null is returned. You don't want to try to concatenate a null to string as this can have unpredictable results. In addition, if the year drop-down has been loaded with years, the first element is always **"<-- Select a Year -->"** and you need to ignore that element. The If statement checks to ensure that the years drop-down has a value and that it doesn't start with "<--". If it doesn't, the value is added to the **searchValues** variable, which is then passed to the setSearchValues() method in the **mainController**.

Add two lines of code in the $(document).ready() to call the setSearchValues() method, then the setSearchArea() method on the pageController closure.

```
$(document).ready(function () {
  // Setup the form submit
  mainController.formSubmit();

  // Collapse search area or not?
  pageController.setSearchValues();
  pageController.setSearchArea();
});
```

Try It Out

Run the application and try out each search area to make sure it's only open when a search value is filled in.

The Problem: Search Area on Shopping Cart Closes After Searching

If you remember from the previous article, you added code to the shopping cart page, so the two search areas are mutually exclusive on being open. If you run a search from either of the two search areas, notice that when the search returns, that search box remains open. As you haven't written any JavaScript code to control this, you can only assume (and rightly so) that this functionality is being controlled on the server.

The code for this functionality is contained on the Views\Shopping\Index.cshtml page, the ShoppingController, and the ShoppingViewModel classes. Two separate properties are needed to set the "collapse" class to one or the other of the two different search area elements. Although this isn't necessarily a bad thing, it does make the HTML markup a little more convoluted and a front-end UI developer might have trouble understanding what's going on. So, let's change this code to run from the client-side instead of from the server-side.

The Solution: Remove the Server-Side Code

Open the **ShoppingViewModel** class located in the **PaulsAutoParts.ViewModelLayer** project and remove the two properties that controls each of the search areas' collapsibility.

```
public string SearchYearMakeModelCollapse
  { get; set; }
public string SearchNameCategoryCollapse
  { get; set; }
```

Next, remove all references to these properties from the ShoppingController class. Open the **Controllers\Shopping\ShoppingController.cs** file and locate where these two variables are set in the Index(), SetYearMakeModel(), and SearchNameCategory() methods and delete those lines of code.

Finally, open the **Views\Shopping\Index.cshtml** file and locate the element where the **SearchYearMakeModelCollapse** property is used to set the class attribute.

```
<div id="yearMakeModel"
     class="@Model.SearchYearMakeModelCollapse">
```

Change the class attribute to **collapse**.

```
<div id="yearMakeModel" class="collapse">
```

Locate the element where the SearchNameCategoryCollapse property is used to set the class attribute.

```
<div id="nameCategory"
     class="@Model.SearchNameCategoryCollapse">
```

Change the class attribute to collapse.

```
<div id="nameCategory" class="collapse">
```

Add a closure at the end of the file immediately after the "use strict" statement, as shown in **Listing 3**.

The code used to make the two search areas on the page mutually exclusive has now been written in the setSearchArea() method. Modify the $(document).ready() function to delete

the code now in the setSearchArea() method, and replace it with the call to the pageController.setSearchArea() method as shown below.

```
$(document).ready(function () {
  // Determine if search area
  // should be collapsed or not
  pageController.setSearchArea();
});
```

Now go back to the **pageController** closure and fill in the code for the setYearMakeModel() to see if anything is filled into the Make or Model drop-downs.

```
function setYearMakeModel() {
  let value = "hide";

  // Have Make or Model been loaded?
  let searchValues =
    $("#SearchEntity_Make option").length +
    $("#SearchEntity_Model option").length;

  // Make and Model have been loaded
  // Thus, a year has been selected
  if (searchValues > 0) {
    value = "show";
  }
  $("#yearMakeModel").collapse(value);
}
```

In this method, add up the value of the lengths of the make and model drop-down lists. If this value is greater than zero, you know the Year drop-down has been loaded. If the Year drop-down has been changed to anything other than the first item in the list, the other two lists have been loaded. Remember that these are dependent lists, so once a valid year has been selected, the make and models have been loaded, and you need to display this search area. Add a new method named setProductNameCategory(), as shown in **Listing 4**, to see if anything is filled into the "Product Name / Category" search area.

In the setProductNameCategory() method, check to see if the Product Name search input value has been filled in and if so, set the value variable to **show** so that the collapsible area is displayed. If not, check to see if the Category drop-down has been loaded with data, and if so, check whether the current category selected is greater than the first element and set the value variable to **show**.

Try It Out

Run the application and fill in different search criteria in the two different search areas and make sure the appropriate search box stays open after searching.

The Problem: Print a Receipt but Hide Certain Elements

Run the sample MVC application, click on the **Shop** menu, and search for some products. Add one or two products to the cart. Click on the **n Items in Cart** menu and click on the Check Out button. Click on the Submit Payment button and you should be taken to the Your Receipt page, as shown in **Figure 2**. The user can just use the browser's print menu to print out the current page, but you might wish to hide some of the HTML elements so they aren't printed. In addition, I recommend that you add a "Print Receipt" button to make it easier for the user to perform the printing. When this button is clicked, call the **window.print()** function. You can use a Bootstrap CSS style, **d-print-none** to eliminate any element you want when entering the print mode for this page.

The Solution: Use Bootstrap Class to Hide Elements While Printing

Open the **Views\CheckOut\Receipt.cshtml** file and add the **d-print-none** class on the element that displays the CCAuth field.

```
<div class="col d-print-none">
  <p>CC Auth:
     @Model.PaymentInformation.CCAuth</p>
</div>
```

Also add the **d-print-none** class on the following element that displays the Response field.

```
<div class="col d-print-none">
  <p>CC Response: @Model.PaymentInformation.Response</p>
</div>
```

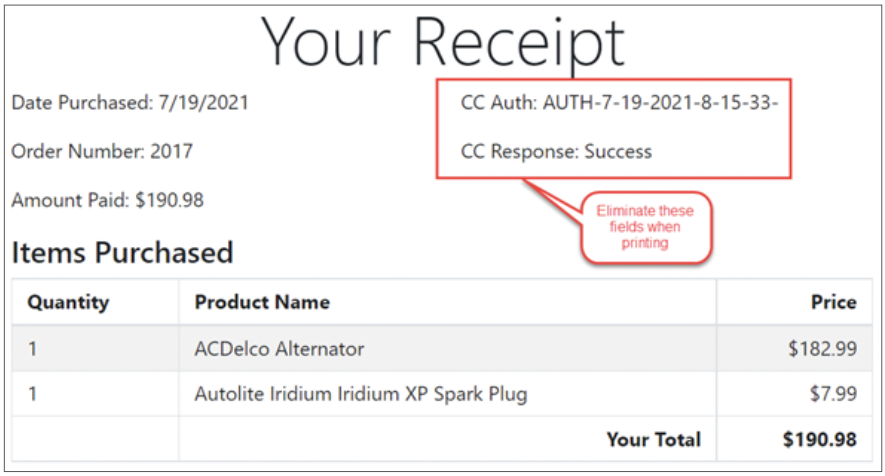


Figure 2: You might want to eliminate some text when printing the receipt

Listing 4: Check whether a product name and/or category has been filled in on the shopping page

```
function setProductNameCategory() {
  let value = "hide";

  // Check for a value in product name
  if ($("#SearchEntity_ProductName").val()) {
    value = "show";
  }
  else {
    // Has category drop-down been loaded?
    if ($("#SearchEntity_Category option").length > 0) {
      // Has a valid category been selected?
      if ($("#SearchEntity_Category").prop("selectedIndex") > 0) {
        value = "show";
      }
    }
  }

  $("#nameCategory").collapse(value);
}
```

Add a new <button> element at the bottom of the page so the user doesn't need to use the browser's print feature, which can be in different places on different browsers. In the **onclick** event of this button, call the **window.print()** function to bring up the browsers' print dialog.

```
<div class="row">
  <div class="col text-center">
    <button type="button"
            onclick="window.print();"
            class="d-print-none">
      Print Receipt
    </button>
  </div>
</div>
```

Try It Out

Run the application and add some items to the shopping cart. Then go through the payment and click on the Print Receipt button. Notice that all the HTML elements that have been marked with **d-print-none** don't show up in the print preview window.

The Problem: Avoid Post-Backs Just to Validate User Input

When you create your entity classes to match the tables in your database, most code generators (such as the Entity Framework generator) can only infer so much from the meta-data in the database. They can usually detect the data type, primary key, the maximum length of a string, whether the field is required or not, and a few other items. This meta-data is added as data annotation attributes to each property in the generated class. However, you probably know a little bit more about each column and can add more information such as a minimum length, a range of valid data, and whether a field holds an email address, phone number, or a credit card number.

The Solution: Add Data Annotations to Entity Classes

Go through each of the properties on each entity class used for data input and add or modify the appropriate data annotations. The data annotations on input properties are turned into unobtrusive jQuery validation on the client-side. This jQuery validation enforces business rules on the client-side and thus avoids a lot of post-backs just to perform business rule checking. Let's modify and also add some additional data annotations to some of the entity classes in the MVC application.

Open the **TableEntities\Customer.cs** file in the **PaulsAutoParts.EntityLayer** project. Modify the **FirstName** property's [StringLength] attribute to set the **MinimumLength** to 2.

```
[Display(Name = "First Name")]
[Required(ErrorMessage =
  "First Name must be filled in.")]
[StringLength(50, MinimumLength = 2,
  ErrorMessage = "First Name must be
    between {2} and {1} characters long.")]
public string FirstName { get; set; }
```

Modify the **LastName** property's [StringLength] attribute to set the **MinimumLength** to 2.

```
[Display(Name = "Last Name")]
[Required(ErrorMessage =
   "Last Name must be filled in.")]
[StringLength(75, MinimumLength = 2,
  ErrorMessage = "Last Name must be
    between {2} and {1} characters long.")]
public string LastName { get; set; }
```

Modify the **Email** property's [StringLength] attribute to set the **MinimumLength** to 3. Add the [EmailAddress] data annotation to the Email property.

```
[Display(Name = "Email Address")]
[Required(ErrorMessage =
  "Email Address must be filled in.")]
[StringLength(255, MinimumLength = 3,
  ErrorMessage = "Email Address must be
    between {2} and {1} characters long.")]
[EmailAddress]
public string EmailAddress { get; set; }
```

Modify the Phone property's [StringLength] attribute to set the MinimumLength to **3**. Add the [Phone] data annotation to the Phone property

```
[Phone]
public string Phone { get; set; }
```

Open the **TableEntities\Product.cs** file and modify the *ProductName* property's [StringLength] attribute to set the **MinimumLength** to 3.

```
[Display(Name = "Product Name")]
[Required(ErrorMessage =
  "Product Name must be filled in.")]
[StringLength(50, MinimumLength = 3,
  ErrorMessage = "Product Name must be
    between {2} and {1} characters long.")]
public string ProductName { get; set; }
```

Modify the Category property's [StringLength] attribute to set the MinimumLength to **3**.

```
[Display(Name = "Category")]
[Required(ErrorMessage =
  "Category must be filled in.")]
[StringLength(20, MinimumLength = 3,
  ErrorMessage = "Category must be
    between {2} and {1} characters long.")]
public string Category { get; set; }
```

Add the [Range] data annotation to the Price property to set the minimum, maximum, and error message. Adding the Range attribute tells jQuery validation to enforce the data entry by the user to be between the minimum and maximum ranges set in this attribute.

```
[Display(Name = "Price")]
[DataType(DataType.Currency)]
[Column(TypeName = "decimal(18,2)")]
[Range(0.01, 99999, ErrorMessage =
  "{0} Must Be Between {1} and {2}")]
public decimal Price { get; set; }
```

Open the **ShoppingEntities\CreditCard.cs** file and add the [CreditCard] data annotation to the **CardNumber** property.

```
[Required]
[StringLength(25)]
[Display(Name = "Credit Card Number")]
```

```
[CreditCard]
public string CardNumber { get; set; }
```

Modify the [StringLength] data annotation to the **SecurityCode** property to set the **MinimumLength** to **3** and set the appropriate error message.

```
[Required]
[StringLength(4, MinimumLength = 3,
  ErrorMessage = "{0} must be between
   {2} and {1} characters long.")]
[Display(Name = "Code on Back")]
public string SecurityCode { get; set; }
```

Add the [Range] data annotation to the **ExpMonth** property to set the **minimum**, **maximum**, and **errormessage** attributes.

```
[Display(Name = "Expiration Month")]
[Range(1, 12, ErrorMessage =
  "{0} Must be Between 1 and 12")]
public int ExpMonth { get; set; }
```

Open the **ShoppingEntities\ShoppingCartItem.cs** file and add the [Range] data annotation to the **Quantity** property to set the range between 1 and 99999 and the error message to display if the quantity is outside of this range.

```
[Display(Name = "Quantity")]
[Range(1, 999999, ErrorMessage =
  "{0} must be between {1} and {2}")]
public int Quantity { get; set; }
```

If you are using the [Phone] and/or [CreditCard] attributes you need to include the **additional-methods.min.js** file. Open the **Views\Shared_ValidationScriptsPartials.cshtml** file in the **PaulsAutoParts** project. Add the following line at the end of this file.

```
<script src="~/lib/jquery-validation/dist/
            additional-methods.min.js">
</script>
```

Try It Out

Run the application and click on **Admin > Products** menu. Click the Add button, then click the Save button and you should see the "Please Wait While Loading" message appear. But the form is invalid and you can just barely see the messages because the page has been grayed out. This is the problem you now need to solve in the next section of this article.

The Problem: "Please Wait" Message Displays Even If Input Isn't Valid

As you just saw, if you attempt to post-back the form but the data is invalid, the "please wait" message is still being displayed. To solve this problem, you need to check to see if the form is valid before displaying this message.

The Solution: Check If Form Is Valid Using jQuery

Because you centralized the form submission code into the site.js file, this fix is very easy to employ. Open the **wwwroot\js\site.js** file and locate the formSubmit() method and add an if() statement to check whether the form is valid. If it is, then display the "please wait" message; otherwise, don't do anything. Modify the formSubmit() method to look like the following code snippet.

```
function formSubmit() {
  $("form").submit(function () {
    if ($("form").valid()) {
      pleaseWait(this);
    }
  });
}
```

Try It Out

Run the application and click on **Admin > Products** menu. Click the Add button and immediately click the Save button. If you've done everything correctly, you should just see the error messages on the page and no "please wait" message is displayed. Go through some of the other pages, such as the customer maintenance and promotional code pages, and ensure they work as well.

The Problem: Don't Allow Numbers in Input Fields

Sometimes data annotations can't cover every type of business rule you wish to enforce. This is where the jQuery Validation library comes in handy. You can add custom rules using jQuery to handle situations where you don't want to have to post-back just to enforce basic rules, such as making sure that there are no numbers in a text string, that a certain date is greater than some other date, etc. Let's look at how to add jQuery Validation to ensure that no numbers are contained in the first name or last name of a customer.

The Solution: Add jQuery Validation Method

Open the **Views\CustomerMaint\CustomerMaintIndex.cshtml** file and add a new method in the **pageController** closure called addValidationRules(), as shown in **Listing 5**.

In the addValidationRules() method, call the $.validator.addMethod() on the jQuery validator object and pass in two parameters. The first parameter is a name, **nonumbers**, that uniquely identifies this custom rule. The second parameter is the function to run when enforcing the rule on the specific field(s) on the page. Make the validation rule name as descriptive as possible so you can tell what it does at a glance. This rule name is also referenced in the **rules** and **messages** objects in the jQuery validate() method.

The **nonumbers** function receives two parameters, the **value** typed in by the user, and the **element** object that caused this function to run. Pass the **element** to the optional() method, which checks to see if the HTML element is missing the required attribute and if the value is blank. If both these conditions are true, this function returns true immediately and the second part of the **or** condition isn't checked. If the **value** is filled in, a regular expression is used to determine whether any numeric values are contained in the value typed in by the user. If so, a false value is returned, and the error message specified in the **messages** object is returned.

In the **$("form").validate()** method, create an object that has two properties; **rules** and **messages**. The **rules** property is itself another object with property names that match up to HTML input elements with the **name** attribute set to the property name. Within each of these properties is another object with a series of properties to specify which rules to check. For example, you may have **required:true** as one of property/value pairs under the "SelectedEntity.FirstName" property. This tells jQuery that the HTML ele-

Getting the Sample Code

You can download the sample code for this article by visiting www.CODEMag.com under the issue and article, or by visiting www.pdsa.com/downloads. Select "Articles" from the Category drop-down. Then select "Enhance your MVC Applications using JavaScript and jQuery: Part 2" from the Item drop-down.

ment with the **name** attribute set to "SelectedEntity.FirstName" is a required field. In the code shown in **Listing 5,** you have the property/value pair **nonumbers:true,** which tells jQuery to invoke the function you wrote previously and pass in the value from the HTML element. Once the user tabs off the First Name field, the function executes and if it returns a true, the value input by the user is valid. If the function returns a false, the input value is invalid, and the message connected to the **nonumbers** property under the **messages** object is displayed. Make the addValidationRules() method public by adding it to the **return** object of the closure as shown in the code snippet below.

Listing 5: Build custom jQuery validation rules using addMethod() and validate() methods

```
function addValidationRules() {
  $.validator.addMethod("nonumbers",
    function (value, element) {
      // this.optional(element): returns true
      // if element is blank and NOT required
      return this.optional(element) ||
             /^([^0-9]*)$/.test(value);
  });

  $("form").validate({
    // The properties in the rules and messages
    // objects are based on the name= attribute
    // in the HTML, not the id attribute
    rules: {
      "SelectedEntity.FirstName": {
        nonumbers: true
      },
      "SelectedEntity.LastName": {
        nonumbers: true
      }
    },
    messages: {
      "SelectedEntity.FirstName": {
        nonumbers: "First Name may not
                    have any numbers in it"
      },
      "SelectedEntity.LastName": {
        nonumbers: "Last Name may not
                    have any numbers in it"
      }
    }
  });
}
```

Listing 6: Remember that the rules are tied to the name, and not the ID, attribute

```
function addValidationRules() {
  $.validator.addMethod("yearplusone",
   function (value, element) {
     // this.optional(element): returns true
     // if element is blank and NOT required
     return this.optional(element) ||
       value < (new Date().getFullYear() + 2);
  });

  $("form").validate({
    // The properties in the rules and messages
    // objects are based on the name= attribute
    // in the HTML, not the id attribute
    rules: {
      "SelectedEntity.Year": {
        yearplusone: true
      }
    },
    messages: {
      "SelectedEntity.Year": {
        yearplusone: "Year Must Be Less Than " +
          (new Date().getFullYear() + 2)
      }
    }
  });
}
```

```
return {
  "setSearchValues": setSearchValues,
  "setSearchArea": mainController.setSearchArea,
  "isSearchFilledIn":
    mainController.isSearchFilledIn,
  "addValidationRules": addValidationRules
}
```

Call the addValidationRules() method from within the $(document).ready() function at the bottom of the page.

```
$(document).ready(function () {
  // Add jQuery validation rules
  pageController.addValidationRules();

  // Setup the form submit
  mainController.formSubmit();

  // Collapse search area or not?
  pageController.setSearchValues();
  pageController.setSearchArea();
});
```

If the value input is invalid, jQuery displays the error message immediately after the HTML element by inserting a <label> element with the class attribute set to the error. Open the **wwwroot\css\site.css** and add a new CSS rule with the class name **error** so the error message displays as red on white text.

```
.error {
  color: red;
}
```

Try It Out

Run the application and select the **Admin > Customers/Orders** menu. Click the Add button and add a number to either the first name or last name fields and tab off the field. You should see the appropriate error message appear immediately after the field.

The Problem: Only Allow Years Less Than the Current Year Plus Two

When adding a new vehicle, the user is required to add a year, make, and model. When adding a year, they shouldn't be able to enter a year that's greater than one year from now. For instance, if this is the year 2021, they shouldn't be able to enter a year greater than 2022. There isn't a data annotation attribute for this (although you could create one). This task is easy to accomplish using the jQuery Validation library.

The Solution: Add jQuery Validation Rule

Open the **Views\VehicleType\VehicleTypeIndex.cshtml** file and add a new method named addValidationRules() to the **pageController** closure, as shown in **Listing 6**.

Just as in the previous addValidationRules() method, you start by calling the jQuery addMethod() on the validator object. The name of this validation rule is **yearplusone**. In the function for this rule, check to ensure that the **value** the user typed in is less than current year plus two. In the **messages** object, create the error message with the actual year that they must be less than. Doing this lets the user

know exactly the values they should be entering for this field. Make the addValidationRules() method public by adding it to the **return** object on the **pageController** closure, as shown in the following code snippet.

```
return {
  "setSearchValues": setSearchValues,
  "setSearchArea": mainController.setSearchArea,
  "isSearchFilledIn":
    mainController.isSearchFilledIn,
  "addValidationRules": addValidationRules
}
```

Call the addValidationRules() method from within the $(document).ready() function at the bottom of the page.

```
$(document).ready(function () {
  // Add jQuery validation rules
  pageController.addValidationRules();

  // Setup the form submit
  mainController.formSubmit();

  // Collapse search area or not?
  pageController.setSearchValues();
  pageController.setSearchArea();
});
```

Try It Out

Run the application, select the **Admin > Vehicle Types** menu, and click on the Add button. Set the Year field to the current year plus two and tab off the field. You should see the appropriate error message appear just after the input field.

The Problem: Validating Two Dependent Fields

Another custom rule you might need is to take the values from two different input fields and checking them to one another or combine them together to check against some other criteria. As an example, in the credit card entry screen, the user needs to select a month and year their credit card expires. These two fields are separate, but you need to compare the month and the year against the current month and year to make sure what they selected are greater than those.

The Solution: Add jQuery Validation Rules

Open the **Views\CheckOut\Index.cshtml** file and add a **pageController** closure immediately after the 'use strict' statement as shown in the code below.

```
let pageController = (function () {
  function isMonthYearValid(month, year) {
  }

  function addValidationRules() {
  }

  // Expose public functions from closure
  return {
    "addValidationRules": addValidationRules
  }
})();
```

Modify the isMonthYearValid() method in the **pageController** closure, as shown in the code below.

```
function isMonthYearValid(month, year) {
  let ret = true;
  let currentMonth = new Date().getMonth() + 1;

  // Only check month,
  // if the current year has been selected
  if (year == new Date().getFullYear()) {
    ret = month >= currentMonth;
  }

  return ret;
}
```

The isMonthYearValid() method accepts the **month** and the **year** from the respective drop-downs on the page. Only if the year passed in is the same as the current year will the month be checked. Otherwise, you assume the year is greater than the current year. When the Year drop-down is loaded, only the current year and greater years are added to the drop-down, thus you don't need to check for a year less than the current year. Modify the addValidationRules() method and add two addMethod() calls to define the custom methods for checking the month and the year, as shown in **Listing 7**.

The first validation rule created is named **checkexpmonth** and its function passes the **value** parameter (which is the month), and the year value from the drop-down to the isMonthYearValid() method. The return value from this function is returned from this validation function.

The second validation rule created is named **checkexpyear** and its function receives the **value** parameter, which is the year selected by the user from the Year drop-down. The month value is retrieved using jQuery and both these values are passed to the isMonthYearValid() method. The return value from this function is returned from this validation function.

Just below the two addMethod() calls, but still within the addValidationRules() method, add the code shown in **Listing 8** to set up the jQuery Form validation.

Depends Function

To get the Month and Year drop-downs to validate together, you need to add a **depends** property within the **checkexpmonth** property. The **depends** property is a function call that accepts the current **element**. You're not doing anything with the **element** argument, instead, you simply make a call to the specific element that's the Year drop-down. This means that both the month and the year rules are called, and both must return true for the validation to pass.

Listing 7: If you have multiple dependent fields, you need to call addMethod() for each one

```
function addValidationRules() {
  $.validator.addMethod("checkexpmonth",
    function (value, element) {
      let ret = isMonthYearValid(parseInt(value),
        $("#CustomerCreditCard_ExpYear").val());

      return ret;
  });

  $.validator.addMethod("checkexpyear",
    function (value, element) {
      let ret = isMonthYearValid(
        $("#CustomerCreditCard_ExpMonth").val(),
        parseInt(value));

     return ret;
  });
}
```

Listing 8: The Depends function is useful when two or more fields depend on one another

```
$("form").validate({
  rules: {
    "CustomerCreditCard.ExpMonth": {
      required: true,
      checkexpmonth: {
        depends: function (element) {
          // After checking the month,
          // always check the year too
          return $("form").validate()
            .element("#CustomerCreditCard_ExpYear");
        }
      }
    },
    "CustomerCreditCard.ExpYear": {
      required: true,
      checkexpyear: true
    }
  },
  messages: {
    "CustomerCreditCard.ExpMonth": {
      checkexpmonth: "Expiration Month/Year
        Must Be Greater Than or Equal to
        The Current Month/Year"
    },
    "CustomerCreditCard.ExpYear": {
      checkexpyear: "Expiration Month/Year
        Must Be Greater Than or Equal to
        The Current Month/Year"
    }
  },
  errorPlacement: function (error, element) {
    if (element.attr("name") ==
          "CustomerCreditCard.ExpMonth" ||
        element.attr("name") ==
          "CustomerCreditCard.ExpYear")
      error.insertAfter(
        "#CustomerCreditCard_ExpMonth");
    else
      error.insertAfter(element);
  }
});
```

ErrorPlacement Property

Besides the **rules** and **messages** properties, there's one additional property created in the jQuery validate() object called **errorPlacement**. The errorPlacement property is a function that's passed the error message to display and the element that caused the error. The default error placement is after the current element being validated. However, in this code I'm checking to see if the **name** attribute on the **element** passed into this function is either the Month or Year drop-downs. If it is, the error message for either of these is only going to be displayed after the Month drop-down. Because these two elements are validated together, the error message should be placed consistently after just one of them.

Add the <partial> element shown below to the </script> element at the end of the page to ensure that the jQuery validation script files are included on this page.

```
<partial name="_ValidationScriptsPartial" />
```

NOTE: If the validation checking doesn't work, ensure that you have the <partial> element after all your </script> tags. Add the code to call the addValidationRules() method inside the $(document).ready() function.

```
$(document).ready(function () {
  // Add jQuery validation rules
  pageController.addValidationRules();

  // Setup the form submit
  mainController.formSubmit();
});
```

Try It Out

Run the application and click on the **Shop** menu. Add an item to the shopping cart and go to the checkout screen. Set the month to a value less than the current month and set the year to this year. You should see an error message appear once you tab off the drop-down.

Summary

In this article, you added additional jQuery to your pages to enhance the user experience and avoid post-backs. It's a good idea to centralize code that's used on multiple pages into a JavaScript file that's included on all pages and keep code that's only for that specific page within a closure on that page. It's also a good practice to reuse the same variable names, such as **pageController**, for each closure on your pages. Take advantage of jQuery validation to create client-side business rules, but you should still write the appropriate code on the server-side to also check those same rules. In the next article, you're going to learn to use Ajax to populate drop-downs, add and delete items from the shopping cart, use jQuery auto-complete, and calculate totals on the shopping cart page.

Paul D. Sheriff

Using Modern JavaScript in the Browser

JavaScript isn't the language it once was. In many ways it's the poster child for how standards bodies can improve products over time. For many people, JavaScript is a bad memory, but with the changes to JavaScript and the ecosystem around JavaScript, it's time to come back to this now ubiquitous language.

JavaScript in the Browser

Some great percentage of the Internet is running on JavaScript that has served us well for the past couple of decades. A typical example (using jQuery) looks something like this:

```
$(document).ready(function () {

  var $form = $("#listForm");

  var validator = new Pristine($form.get(0));

  $form.on("submit", function (e) {
    e.preventDefault();

    if (validator.validate()) {
      // Submit the form
      $form.trigger("reset");
      $("#listSignupResult").text("Sent...");
    } else {
      $("#listSignupResult")
        .text("Please correct errors...");
    }
  })

});
```

This worked well and continues to work. This style of JavaScript is certainly a product of the technology at the time. The early versions of JavaScript (e.g., ECMAScript) weren't trying to innovate in the language but work well across different browsers and operating systems. This style also predates many important software ideologies, like unit testing. This means that the style of coding didn't take some of the ideas into account. Much of this was driven by the fact that we thought we were writing small amounts of JavaScript, but over time, that grew and grew into more complex and harder to maintain codebases. Something had to change.

What Is "Modern JavaScript?"

When we talk about Modern JavaScript, we must dig a little deeper into what that means. Nearly a year after Sun Microsystems and Netscape announced the release of JavaScript (and support for it in the browser), they formalized the language by announcing a meeting to standardize the language.

This standardization allowed for versioning of the language and attempted to come to agreement about what features would be supported by all browsers. Of course, this only worked as much as the companies involved could cooperate and how much adoption there was for newer versions of browsers (I'm looking at you, Internet Explorer). This left the language stuck at ECMAScript 3 for almost two decades. Browser companies continued to add newer features—they just weren't agreed on. The big change was supposed to happen with version 4 of ECMAScript, but it was shelved and never released due to infighting on the ECMAScript board. When ECMAScript 5 (or ES5) was released, it ushered in the real beginning of what I call "Modern JavaScript."

Since then, the releases of the language have been on a pretty consistent release schedule and the browsers fairly consistently kept up with new features. This means, for the most part, that we can move on from the browser wars and work with JavaScript in a more mature way.

Enough history! On to the modern usage of JavaScript.

ECMAScript 6 and Beyond

After finally releasing ECMAScript 5, JavaScript standardization got complicated. The sands are moving faster than you might imagine in the specs because they're coming on a somewhat yearly basis. Because of that, we sometimes need to have a way to see whether a feature in modern JavaScript is supported by browsers. One useful tool in this is https://caniuse.com. For example, if I want to see if the **let** feature works, I can go to that site and put in that feature, as seen in **Figure 1**.

Although looking at the specific browsers is interesting, I'm mostly looking for the global usage (as shown in **Figure 2**). Once this number is over 95% or so, I know that I can use it natively in the browser.

Of course, the numbers might never be 100% covered. Browsers don't die; they slowly fade away. With this in mind, there are times when you'll need to transpile your JavaScript code for older browsers. Tools like Babel do this transpiling for support in a wide number of languages. They essentially write the JavaScript in a way that works in all browsers.

> Browsers don't die;
> they slowly fade away.

Whether you end up transpiling for older browsers or just use certain features because you know that your users are using modern browsers, there are a set of features that exemplify modern JavaScript including classes, arrow functions,

Shawn Wildermuth
shawn@wildermuth.com
wildermuth.com
twitter.com/shawnwildermuth

Shawn Wildermuth has been tinkering with computers and software since he got a Vic-20 back in the early '80s. As a Microsoft MVP since 2003, he's also involved with Microsoft as an ASP.NET Insider and ClientDev Insider. He's the author of over twenty Pluralsight courses, written eight books, an international conference speaker, and one of the Wilder Minds. You can reach him at his blog at http://wildermuth.com. He's also making his first, feature-length documentary about software developers today called "Hello World: The Film." You can see more about it at http://helloworldfilm.com.

transpilation and more. These features make JavaScript easier to write and easier to maintain. No longer is JavaScript a simple scripting language, but now it's a modern language because of these features and the tooling that allows you to transpile your larger projects into applications. Let's talk about the main features you should think about using in your own projects.

Var, Let, and Const

If you've written any JavaScript in the past twenty years, you're probably familiar with the simple **var** statement. It's the way to declare variables, but it's problematic. The issue is that var in the global scope creates a global object. Consider this code:

```
var x = 0; // it is global
if (this.x === 0) console.log("x is 0");
```

Using var creates a globally available object. That's why they introduced **let**. As an alternative, let defines a variable that is scoped:

```
let y = 1; // scoped
// this.y is undefined
if (this.y === 1) console.log("y is 1");
```

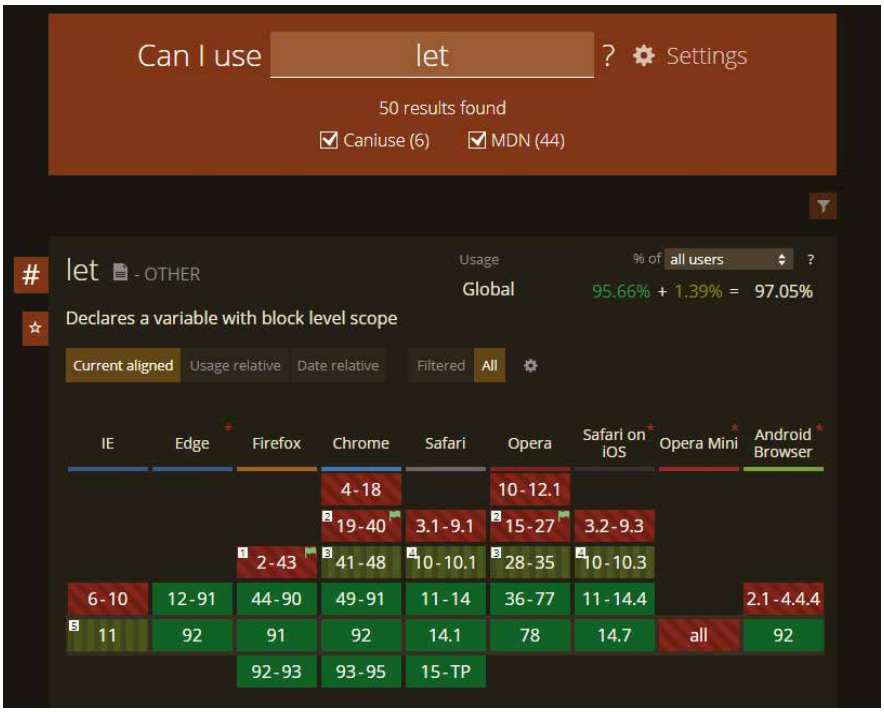


Figure 1: www.caniuse.com looks at let

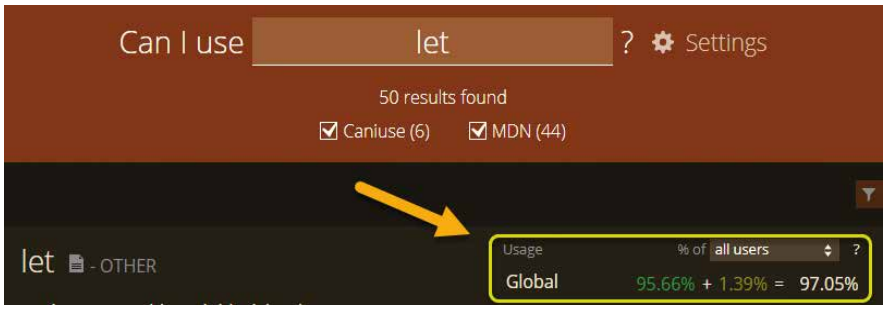


Figure 2: Global Usage

In this case, the **y** variable is scoped to the file, not into the global scope. Inside any real scope (e.g., function, block, etc.), your code is never leaked into other scopes accidently.

Along with **let**, a new way to define a variable is by using the **const** keyword:

```
const name = "Shawn";
name = 'Bob'; // error
```

In this example, you're specifying that the value can't be changed. In many cases, within a scope, you'll want to have as many of your variables use **const** as much as possible. This forces you to think about whether a variable is changed or not. In many cases, you aren't creating a variable to change, but just to hold a result. For example:

```
const someDiv = document.getElementById("someDiv");
```

You can see, if you've gotten some value, that you're never going to reassign this value but use it as starting point to some code. What's important here is that the **someDiv** isn't immutable but the variable is. So, you can set or change values on someDiv, but you can't reassign someDiv.

Unless you have a specific use for **var**, you should switch to using **let** and **const** for all variable construction in modern JavaScript.

Classes

Many of us came to JavaScript after working with more typical object-oriented languages (e.g., Java, C#, etc.). Modern JavaScript enables class structures that are similar (although not the same) as these other languages. The big difference is that classes shouldn't be considered the central object container like they are in Java and C#. Instead, continue using array and object syntax when a class isn't necessary. Now that I'm off my soapbox, let's see how classes work in JavaScript.

Classes are defined with class keyword:

```
class ListForm {
}
```

The names of classes are generally Pascal-cased because objects from classes need to use the **new** operator to create a new instance:

```
const form = new ListForm();
```

Classes can also have member variables:

```
class ListForm {
  form =
    document.getElementById("listForm");
  msg =
    document.getElementById("listSignupResult");
}
```

In addition, you can have methods (e.g., functions). Member methods are defined by just creating the shape of functions as members of the class (the function keyword isn't necessary):

```
class ListForm {
  form = document.getElementById("listForm");
```

```
  msg = document.getElementById("listSignupResult");

  validateForm() {
    this.form.reset();
    this.msg.innerText = «Sent...";
  }
}
```

Notice that to access members, you must use the **this** keyword. You can also parameterize construction by using a constructor:

```
class ListForm {

  constructor(form, msg) {
    this.form = form;
    this.msg = msg;
  }
...
```

Note that by defining the form and msg (in this example) using the **this** keyword, you're adding them as fields on the class. Therefore, the method in the class can still use them:

```
  validateForm() {
    this.form.reset();
    this.msg.innerText = «Sent...";
  }
```

In practice, you'll want to define the fields and initialize them to better document classes:

```
class ListForm {

  form = null;
  msg = null;

  constructor(form, msg) {
    this.form = form;
    this.msg = msg;
  }
...
```

In this form, the fields are defined as publicly available, but initializing them in the constructor. This is functionally equivalent to not defining them, but I think it makes the class much more obvious what are fields and what aren't fields.

Classes also support properties by defining functions that return or set properties:

```
  set theForm(value) {
    this.form = value;
  }

  get theMessage() {
    return this.msg.innerText;
  }
```

This allows you to have property accessors for reading and writing of fields on the class. If you're using accessors, you may want to hide the underlying fields. You can do this with private fields (although not supported in all newer browsers) by prefixing them with the **#**.

```
class ListForm {

  #form = null;
  #msg = null;

  constructor(form, msg) {
    this.#form = form;
    this.#msg = msg;
    this.validator = new Pristine(this.#form)
  }

  set theForm(value) {
    this.#form = value;
  }
```

Note that the # becomes part of the name of the variable, but this way, the fields aren't accessible outside the class.

I could write an entire article just on the class structure, but instead I'll just point out that the class supports other typical object-oriented features like:

- Inheritance (via **extends** and **super** keywords)
- Type information (via isInstanceOf keyword)
- Static fields and properties (via adding them to the type instead of the class)

Again, unless you need this type of object creation, information hiding, or polymorphism, I wouldn't replace every object or function in your code base with classes.

Arrow Functions

Before I can talk about what arrow functions are, let's talk about regular functions. Functions are defined by the **function** keyword:

```
function show(msg) {
  console.info(msg);
}
```

In this example, it's named, has an argument (e.g., data passed in), and has some body that contains some code to be executed. Pretty simple. Functions don't need to be named, but can be anonymous. For example, they're often used anonymously for callbacks or Promises:

```
form.addEventListener("submit", function(e) {
  console.info(e);
});
```

Here, we're using it to have the event listener execute the function once someone submits the form. The function structure is essentially the same (adding parameters and a body), but just not named (e.g., anonymous).

Functions have a secret though; they create a scope. That scope is important so that data inside functions doesn't leak into the parent (or global) scope:

```
form.addEventListener("submit", function(e) {
  let x = 0;
  console.info(e);
});

console.info(x); // fails, because x is not defined
```

So what does this have to do with Arrow Functions? Well, arrow functions are a way to more discreetly define functions.

The form is to just have the parameters and the code block separated by **=>** (which is why they're called arrow functions). These are similar to lambdas in other languages. For example:

```
form.addEventListener("submit", (e) => {
  console.info(e);
});
```

Arrow functions can omit the curly braces for a more concise format too:

```
form.addEventListener("submit",
                      (e) => console.info(e))
```

In some ways, the arrow function is to provide conciseness in the code. You can also assign the function to a variable (like you can do with anonymous functions):

```
const showInfo = (msg) => console.info(msg);

showInfo("Hello World");
```

There is one big difference between arrow functions and normal functions: Arrow functions do not create a scope. They're lighter than standard functions. You can think of it as arrow functions adopting their parent scope. But remember, anything created in the arrow function is valid outside the function:

```
form.addEventListener("submit",  (e) => {
  let x = 0;
  console.info(e);
});

console.info(x); // succeeds,
                 // x is now defined in this scope
```

I now use arrow functions in most cases when I need an anonymous function. This is especially true for callbacks or promises.

Destructuring

Another important feature that you'll want to get used is destructuring objects and arrays. The term comes from the idea of getting at parts of a structure by accessing individual components without needing to reference the original object. Let's start with object destructuring to explain what's going on. Let's assume that you have an object like so:

```
let user = {
  name: "Shawn",
  email: "shawn@aol.com",
  birthday: "1999-05-05"
};
```

Destructuring allows you to pull values out into their own variables. It does this by using curly braces to describe what properties come out. For example:

```
let { name, email, birthday } = user;

console.log(name);
```

This is equivalent to:

```
let name = user.name;
let email = user.email;
let birthday = user.birthday;
```

Although breaking out an object into individual properties might not be that interesting, it's really powerful when you don't need many of the properties:

```
let { name, email } = user;
```

It matches these by name, not position. You'll use these later when you look at Modular JavaScript.

Arrays are similar, but are destructed by position, not name (obviously):

```
let colors = ["blue", "red", "green"];

let [first] = colors;

console.log(first);
```

You're still creating a variable (called **first** in this example) but automatically putting the value from the ordinal position (e.g., **blue**).

Ultimately, destructuring is about flattening structures into simple values. Although you wouldn't use this in place of creating variables using property accessors or array indexes, it provides a new way to think about accessing parts of objects.

Spread Operator

In some ways, the spread operator represents the opposite of destructuring. The spread operator (**...**) allows you to take any properties or array elements and spread them into another object. For example:

```
let user = {
  name: "Shawn",
  email: "shawn@aol.com",
  birthday: "1999-05-05"
};

let authInfo = {
  token: "12345",
  ...user
};
// becomes { token, name, email, birthday }
```

Effectively, the spread operator tells the interpreter to destructure the user object and assign each of the properties onto the **authInfo** object. It works the same way with arrays too:

```
let primaryColors = [ "blue", "red", "green" ];

let colors = [
               ...primaryColors,
               "yellow",
               "orange",
               "purple"
             ];

// colors[0] = "blue"
```

Although you could do this by merging the arrays, this can be a powerful way to combine data using this simple syntax.

Array Methods

While we're talking a about arrays, let's talk about methods added to the Array type. I'm not going to cover all of them,

ECMAScript = JavaScript

Although ECMAScript is the official name of the language, I'll use JavaScript as the name of the language in this article. In day-to-day usage, almost no one uses ECMAScript as its name.

but for me, the real power is in some of the array manipulation that you've needed to use **underscore** or **lodash** to accomplish. They're now built-into the language.

First, let's see how **Array.find** works:

```
let primaryColors = [ "blue", "red", "green" ];

let red = primaryColors
            .find(val => val.length === 3);
```

The Find method takes a function (or Arrow function as you're seeing here) to find a single element. The first one that's matched by the function is returned. In this case, it returns **red** because it's the first element whose length is 3. The **val** variable is the value of each element of the array. This function is called for every member of the array until it returns a true. This is the same as this (more verbose) version:

```
let red = primaryColors.find(function (val) {
  return val.length === 3 ? true : false;
});
```

I'm showing this version to be really clear about what it's doing, but using an Arrow function makes this a lot shorter and clearer.

The next method is **filter**. It's similar to **find**, but instead returns all the matching elements. Here's how it works:

```
let colors = primaryColors
  .filter(val => val.length >= 4);
// [ 'blue', 'green' ]
```

A key difference here is that **filter** returns an array and **find** returns an element of that array.

Another interesting method is called **every**. This method returns whether all the members of the array satisfy some test. For example:

```
if (primaryColors
      .every(val => val.length > 0)) {
  console.log("No empty colors");
}
```

The inverse of **every** is **some** (it's not new, but lots of developers seem to have missed it):

```
if (primaryColors
      .some(val => val.length === 3)) {
  console.log("At least one is three chars long");
}
```

Again, all of these methods take a callback where you can do a small amount of computation or large. They really empower you as a developer to manipulate arrays without resorting to third-party libraries.

Modular JavaScript

Although most of the features I've talked about in Modern JavaScript have been new objects or syntax, Modular JavaScript is a big change in the way you think about JavaScript development. Before modular JavaScript, you wrote one or more JavaScript files, included them all on the page, and hoped that they were loaded early enough to make it all work:

```
<script src="jquery.min.js"></script>
<script src="bootstrap.min.js"></script>
<script src="pristine.js"></script>
<script src="index.js"></script>
```

By writing some of the code in index.js, you'd have to wait for the browser to load the first three scripts to be sure that the objects you needed in the global scope were available (e.g., **$**, **jQuery**, **Pristine**, etc.). The problem here was that we weren't clear about what the script did and didn't need. We'd just throw everything into the global scope and live with the bloat.

There were several approaches to solve this, but the two that are most prevalent are Node's CommonJS and JavaScript modules. Because the browsers are going to support the JavaScript import/export, let's see how that works.

Instead of depending on the global scope, in modular JavaScript, you just use **import** statements to bring in the libraries/code that you need. For example, let's say that you wanted to load a library to do:

```
import Pristine from "pristinejs";
```

This brings in an object called "Pristine" that you could use in the code. The code is self-documenting, as you know where all the objects are coming from, rather than depending on the global scope. You could then use the Pristine object like so:

```
const form = document
               .getElementById("listForm");
const validator = new Pristine(form);
```

The reverse of this is exporting from a script. For example,

```
export function validateForm() {
  ...
}
```

Then in another script, you could load this function to use it:

```
import { validateForm }
  from "./validateForm.js";

validateForm();
```

The form of import is to point at a relative JavaScript file that you're loading (called validateForm.js). In the early import, Pristine is an NPM package that you've loaded, so you're retrieving it by name.

You'll notice, in this import, that you're using the destructuring syntax to get the function called **validateForm**. That's necessary, as you could export multiple things out of a single file. If you change that export to default, it allows you to import it without a name (like you did for Pristine). For example:

```
export default function() {
  ...
}
```

In that case, you can call the function whatever you want as, when you import it, it just imports the default object (e.g., function, object, array, class, etc.):

```
import validateForm from «./validateForm.js";

validateForm();
```

By using imports and exports, you could see that a single import of some top-level object (like the main function of JavaScript), could walk through the code and know all the dependencies. In that way, modular JavaScript doesn't need to load every script, but know that it needs to load them as they walk the chain of files.

To make this work, you need to use the script tag's support for **modules**:

```
<script type="module" src="./main.js">
</script>
```

This tells the browser to load the main.js as a module (which can start the process) but then any **import** becomes a request to the server. In this way, the page doesn't need to know about each piece of code that's necessary. But it comes at the cost of higher network activity. This is great for development (see my article on Vite in the September/October 2021 issue of CODE Magazine), but can be problematic for production. In addition, script modules aren't well supported as of the writing of this article. To get around both of these issues, packaging your projects for the browser is a common way to handle this. Let's see how that works.

Packaging for the Browser

All of these features come together by packaging your projects for the browser. What do I mean by **packaging**? Essentially, leveraging the modular JavaScript to build a single (or usually a small number) of JavaScript files that are loaded by the browser. The essential idea here is to have a startup script (e.g., main.js) that loads and executes your JavaScript. What happens is shown in **Figure 3**.

Figure 3 shows how the packager is being asked to package **main.js**. It walks through the different imports to get a full list of required scripts and then packages them together in a resulting JavaScript file. This one file contains all the code necessary to run the code in **main.js**. Your production page just refers to the main.min.js (or whatever it's called) and it can run the project.

Although the packagers attempt to be concise for production builds, sometimes there's a lot of code in a project. Packagers allow you to do code splitting by implementing dynamic imports. Dynamic imports are simply a way of telling the packager to break up the project into multiple files that are loaded as necessary.

Other benefit of using packagers is that they often transpile your project down to ECMAScript 3 so that the code you're writing with the latest and greatest features of JavaScript will work with the maximum number of browsers. In this way, you can get the best of both worlds.

Lastly, packagers usually support something called "Tree Shaking" for production builds. Tree shaking is simply walking through the files that're imported, and only including the parts of any project that are necessary for your project. Consider my example, where you might use our **validateForm.js** file. If that project exported several different objects, but the code only uses one of them, the tree shaker just eliminates the code in validateForm.js that's not called. In this way, the size of the resulting built JavaScript should be significantly smaller when using generalized libraries (e.g., **lodash** or **momentjs**) where you might be calling one or two code paths in a larger project.

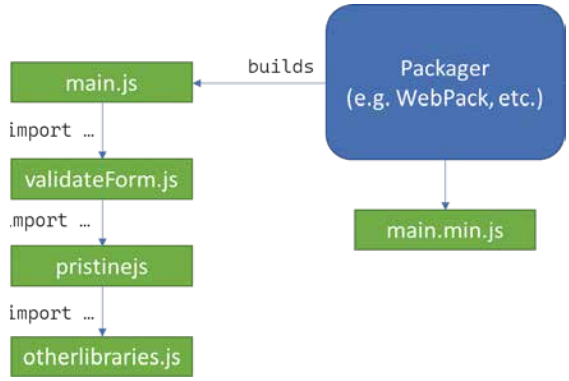


Figure 3: Packaging

Let's see how that works. In a simple project like this one, where you want to just perform some validation on a form, you could write the main.js like so:

```
// main.js
import validateForm from «./validateForm.js";

validateForm();
```

This then requires you to have written the **validateForm** like so:

```
import Pristine from "pristinejs";

export default function() {

  const form = document
    .getElementById("listForm");
  const msg = document
    .getElementById("listSignupResult");
  const validator = new Pristine(form);

  form.addEventListener("submit", (e) => {
    e.preventDefault();

    if (validator.validate()) {
      form.reset();
      msg.innerText = "Sent...";
    } else {
      msg.innerText = "Failed to validate";
    }
  });
}
```

Notice that you're using a library called **pristinejs**. In this case, you'd get this library from Node Package Manager (NPM). If you haven't used Node.js before, this may a little overwhelming, but let's stick to the pieces. If you don't have

```
[shawn] D:\..\ModernJS\Example>npx webpack ./main.js
asset main.js 6.4 KiB [compared for emit] [minimized] (name: main)
runtime modules 663 bytes 3 modules
orphan modules 460 bytes [orphan] 1 module
cacheable modules 16.1 KiB
  ./main.js + 1 modules 536 bytes [built] [code generated]
  ./node_modules/pristinejs/dist/pristine.js 15.5 KiB [built] [code generated]

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value.
Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/configuration/mode/

webpack 5.39.0 compiled with 1 warning in 464 ms
```

Figure 4: Webpack Output

Node.js installed, go to https://nodejs.org/ and install it first.

Next, you can just go to the project's directory and initialize a package file:

```
> npm init -y
```

This initializes a package.json file where you can install dependencies. You only need two. First, you can install the pristinejs package by calling:

```
> npm install pristinejs -save
```

You end up with a simple package.json with the pristinejs installed:

```
{
  "name": "Example",
  "version": "1.0.0",
  "description": "",
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    «pristinejs": "^0.1.9"
  }
}
```

In this example, you're going to use the most common packager: WebPack. There are others (e.g., Browserify, Rollup, etc.), but WebPack is pretty simple. You can just run it using the Node Package Executer (npx):

```
> npx webpack ./main.js
```

By default, this builds in a **./dist** folder and uses a production build (it gives you a warning about this), as seen in **Figure 4**.

Webpack supports changing a lot of this behavior, but in this smallest case, you can now use that compiled version of your project by simply pointing the HTML to use the single **main.js** in the **./dist** folder:

```
  <script src="./dist/main.js"></script>
```

Although this doesn't teach you all about packaging for the Web, you should have the basic idea of why you package for the Web at all.

Where Are We?

Whew! JavaScript has been through a number of small and large changes in the way we write modern JavaScript these days. If you've been using frameworks out there, like Angular, Vue, or React; you can see many of these changes reflected in how those projects work. My sincere hope is that you can adapt to these new tools in your JavaScript toolbox and see that, today, you can write large, powerful projects using JavaScript in the browser. By leveraging these techniques and features, you can write, test, and deploy more reliable and maintainable code.

Shawn Wildermuth

CODE

DAX with Dates: The Power Plays

Microsoft's Power BI works as the ultimate power tool for data analysis. Building a model lets you tap into many functionalities within Power BI-like visuals, Power Query, and DAX calculations. DAX is a language available in Power BI that enables you to build formulas and expressions for many kinds of calculations.

Helen Wall
linkedin.com/in/helenrmwall/
www.helendatadesign.com

Helen Wall is a power user of Microsoft Power BI, Excel, and Tableau. The primary driver behind working in these tools is finding the point where data analytics meets design principles, thus making data visualization platforms both an art and a science. She considers herself both a lifelong teacher and learner. She is a LinkedIn Learning instructor for Power BI courses that focus on all aspects of using the application, including data methods, dashboard design, and programming in DAX and M formula language. Her work background includes an array of industries, and in numerous functional groups, including actuarial, financial reporting, forecasting, IT, and management consulting. She has a double bachelor's degree from the University of Washington where she studied math and economics, and also was a Division I varsity rower. On a note about brushing with history, the real-life characters from the book The Boys in the Boat were also Husky rowers that came before her. She also has a master's degree in financial management from Durham University (in the United Kingdom).

DAX: Data Analysis Expressions

The DAX library gives you a litany of options for functions and operators to combine to build formulas and expressions. Within Power BI, you can leverage DAX (short for Data Analysis Expressions) formulas to create three different types of outputs.

- Tables
- Columns
- Measures

For the purposes of this project, you'll focus on creating DAX measures in Power BI Desktop specifically involving date calculations. DAX can look deceptively easy at first because the syntax looks simple, but the logic behind how it works can become a bit tricky. Modeling in DAX, however, become quite fun once you understand how DAX works.

Defining DAX Measures

DAX measures serve as portable formulas in Power BI models. Power BI uses a Vertipaq engine to compress and store the data, which means that DAX measures work more efficiently than DAX columns because they minimize model size by eliminating the need to add new calculated columns. Here are some helpful rules to keep in mind when working with DAX measure calculations.

- Calculations works against data source and not against the visual (such as a table) you're adding them to.
- Measures work independently of one another. This means removing a measure from a table won't impact the calculations of the other measures in the table.
- Filters apply first, then calculations. Unlike Excel, DAX doesn't work on individual cells within tables, but instead works on tables and columns. This means that you'll need to leverage a neat trick with harvesting parameters to reference the value of a cell equivalent in Power BI to create these calculations.

You can only create DAX measures in the Desktop version of Power BI. I typically create a separate table just for the DAX measures, which I think makes it easier to organize the measures in the model. You can do this by creating a new blank table and then adding DAX measures to this new table. You'll remove Column1 after you start adding new measures. Both the measures themselves and the entire Calculations table display calculator icons next to their names.

DAX versus Power Query

I'm not going to discuss Power Query much in this project, but you'll need to leverage it to get the data for these DAX calculations. The Power Query Editor easily lets you set up queries for many different types of data sources like the Federal Reserve of St. Louis Economic Data you'll leverage for this project: https://fred.stlouisfed.org/. If you want to learn more about Power Query, check out another article I wrote for CODE Magazine that focuses specifically on this ETL framework: https://www.codemag.com/Article/2008051/Power-Query-Excel%E2%80%99s-Hidden-Weapon.

You can easily configure this Web API data connection by creating a Blank Query in the Power Query Editor, then pasting the M code below into the Advanced Editor window. M is the language (similar to F#) that creates the ETL framework under the hood of Power Query. You're going to first need to request your own FRED API key (https://fred.stlouisfed.org/docs/api/fred/) to update the Source step below for the API query parameter api_key.

```
let
    Source = Xml.Tables(Web.Contents(
"https://api.stlouisfed.org/fred/series/observati
ons?series_id=MORTGAGE30US&api_key=api_key")),
    observation = Source{0}[observation],
    #"Changed Type" = Table.TransformColumnTypes(
observation,
{{"Attribute:realtime_start", type date},
     {"Attribute:realtime_end", type date},
     {"Attribute:date", type date},
{"Attribute:value", type number}}),
    #"Inserted Division" = Table.AddColumn(
#"Changed Type", "Division",
each [#"Attribute:value"] / 100, type number),
    #"Removed Columns" = Table.RemoveColumns(
#"Inserted Division",
{"Attribute:realtime_start",
"Attribute:realtime_end", "Attribute:value"}),
    #"Renamed Columns" = Table.RenameColumns(
#"Removed Columns",{{"Attribute:date", "Date"},
     {"Division", "Rate"}})
in
    #"Renamed Columns"
```

Once you get the query set up with this ETL framework, you'll then load the data into Power BI Desktop.

Creating Measures Using CALCULATE

The CALCULATE function is a key DAX function to know when you're creating measures. The CALCULATE function consists of two key components: the calculation part and the filter part. You'll want to think first about filtering the data table you're calculating the measure over, then perform the actual calculation over these filtered rows. However, within the CALCULATE function, the computation part goes before the filters part (separated by commas). The CALCULATE function doesn't require you populate the second filter's component of the formula to return a result, but you do need to populate the first component specifying the calculated aggregation to properly return results. In the measure formulas below, you're nesting the Rate field within the AVERAGE, MAX, or MIN DAX functions and nesting these aggregations within the CALCULATE function.

```
(1a) Average Rate = CALCULATE(AVERAGE(
'Mortgage Rates'[Rate]))
```

```
(1b) Max Rate = CALCULATE(MAX(
'Mortgage Rates'[Rate]))
```

```
(1c) Min Rate = CALCULATE(MIN(
'Mortgage Rates'[Rate]))
```

When creating DAX measures, you'll want to test that they work accurately. Because the DAX measures don't exist as actual columns within your model, you won't see them in the Data view of your model. You can, however, validate that these calculations work by adding them to a standard Power BI table visual, as you see on the left side of **Figure 1**.

Evaluation Context

You can define the evaluation context of the measures within a Power BI DAX model as the dimensions and filters of the visuals you're adding the measures to. In **Figure 1**, the dates in the table visual represent the pivot coordinates that you're evaluating at for each of the results the measure calculations return. Notice that the aggregated average Rate field in this same table returns the same result as the Average Rate measure, as well as the Max Rate and Min Rate measures. This occurs because of the context in which the DAX measure evaluates the expression. It's first filtering the data table by the dates in the first column of the summarized table visual. It doesn't matter whether you're calculating the average, maximum, or minimum in this table evaluation context because the data only contains one mortgage rate for each date, and thus the measures will result in the same result for each date regardless of the aggregation type.

To change the evaluation context, you can add or remove dimension fields to the rows in the table in **Figure 1**. You can remove the Date field from the table visual or you can create new card visuals for each measure to return the same results. You can see that the card visuals in the top right display the average, maximum, and minimum rate over the entire date in the data source instead of for each date. When the evaluation context for calculating the measure changes, so do the results of these calculations. These measures calculate the results without applying any filters because this visual doesn't have any pivot coordinates.

How Filters Work

You may notice that the mortgage rate data contains weekly date measurements collected over a fifty-year period. What if you wanted to take a closer look at a specific date range? You can add filters to Power BI in several different ways:

- By the pivot coordinate in which you're evaluating a DAX measure
- Adding filters directly to the Filters pane
- Configuring filtering between visuals like slicers that enable interactive user filtering
- You can apply filters in a DAX measure expression by adding filters as a parameter within a DAX expression

DAX Functions with Dates

Now let's focus attention on changing the filters applying to the measure calculations by changing the filter context directly by adding the filters to the second part of the CALCULATE function. Date filters work in two different ways:

- They can move each pivot coordinate date for evaluating the measure to a different date.
- They can expand filter context to include a wider date range for evaluating the measure over.

Filters can also either narrow down or expand the evaluation context for the measure results, depending on the fields that already feed into the calculated formula and the dimensions of the visual you're adding the measure to.

ALL

To calculate average, maximum, or minimum rate over all the mortgage rate dates that FRED measures, you can leverage the

Date	Average of Rate	(1a) Average Rate	(1b) Max Rate	(1c) Min Rate
4/2/1971	7.33%	7.33%	7.33%	7.33%
4/9/1971	7.31%	7.31%	7.31%	7.31%
4/16/1971	7.31%	7.31%	7.31%	7.31%
4/23/1971	7.31%	7.31%	7.31%	7.31%
4/30/1971	7.29%	7.29%	7.29%	7.29%
5/7/1971	7.38%	7.38%	7.38%	7.38%
5/14/1971	7.42%	7.42%	7.42%	7.42%
5/21/1971	7.44%	7.44%	7.44%	7.44%
5/28/1971	7.46%	7.46%	7.46%	7.46%
6/4/1971	7.52%	7.52%	7.52%	7.52%
6/11/1971	7.52%	7.52%	7.52%	7.52%
6/18/1971	7.54%	7.54%	7.54%	7.54%
6/25/1971	7.54%	7.54%	7.54%	7.54%
7/2/1971	7.54%	7.54%	7.54%	7.54%
7/9/1971	7.54%	7.54%	7.54%	7.54%
7/16/1971	7.60%	7.60%	7.60%	7.60%
7/23/1971	7.65%	7.65%	7.65%	7.65%
7/30/1971	7.69%	7.69%	7.69%	7.69%
8/6/1971	7.66%	7.66%	7.66%	7.66%
8/13/1971	7.73%	7.73%	7.73%	7.73%
8/20/1971	7.71%	7.71%	7.71%	7.71%
Total	**7.85%**	**7.85%**	**18.63%**	**2.65%**

Figure 1: CALCULATE function

ALL function to override the pivot coordinates for all the dates in the table. This enables the measures to return the same calculated result for each date in the table in **Figure 2**. In the second parameter of the CALCULATE function you'll place the ALL function around the Date fields and calculate the average, maximum, and minimum rate in the first part of the formula over the entire date range in Mortgage Rate data source.

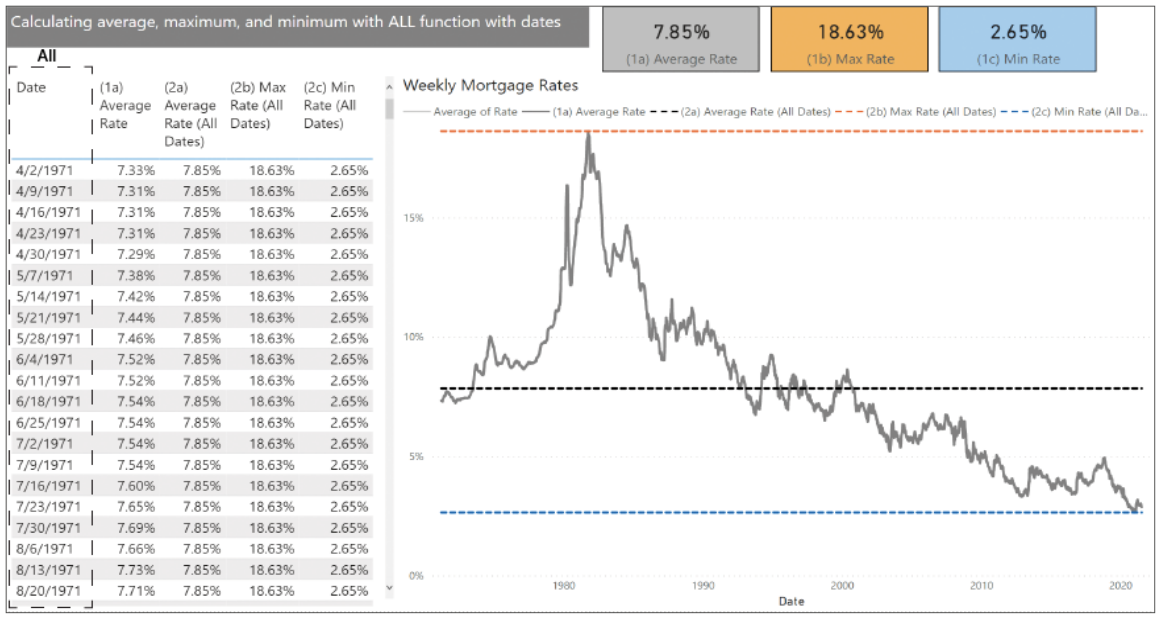
```
(2a) Average Rate (All Dates) =
CALCULATE(AVERAGE('Mortgage Rates'[Rate]),
ALL('Mortgage Rates'[Date]))
```

```
(2b) Max Rate (All Dates) =
CALCULATE(MAX('Mortgage Rates'[Rate]),
ALL('Mortgage Rates'[Date]))
```

```
(2c) Min Rate (All Dates) =
CALCULATE(MIN('Mortgage Rates'[Rate]),
ALL('Mortgage Rates'[Date]))
```

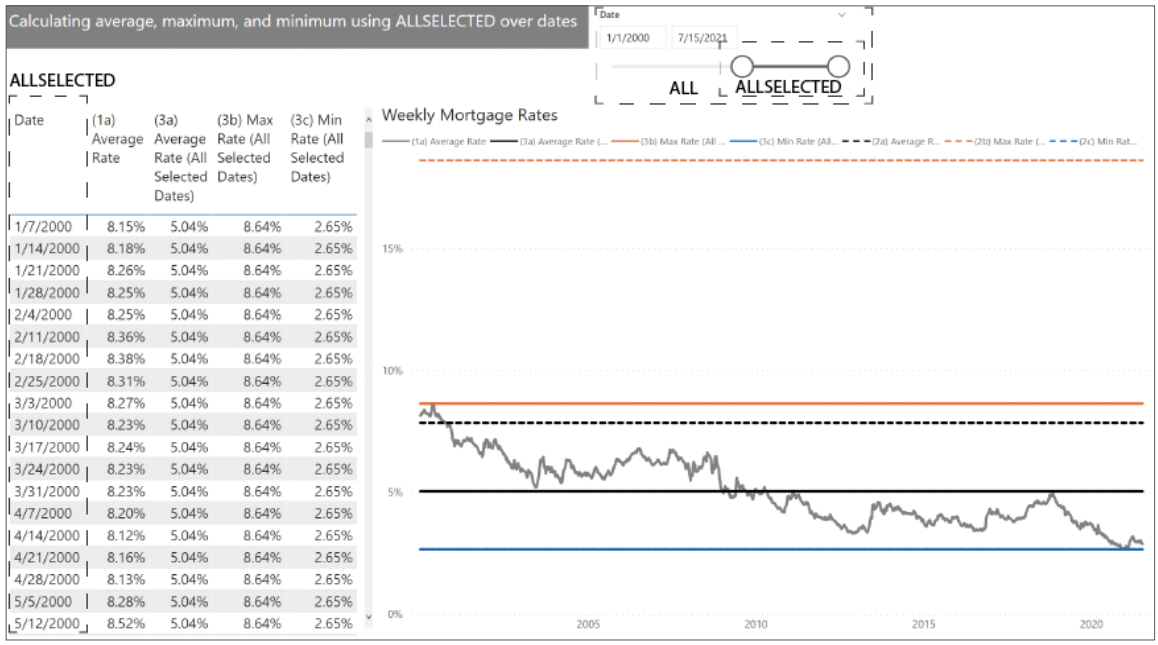
ALLSELECTED

What if you want to focus the calculations on a narrower date range than every week over the fifty-year period? If you add a slicer visual with the dates field, you'll give the end user the capability to select the date range over which to calculate the average, minimum, and maximum rates. Like the measures leveraging the ALL function, the ALLSELECTED function removes the date pivot coordinates, defining the evaluation context for the result the measures return. It doesn't, however, ignore all the filters because narrowing down the date range of the slicer visual does change the



Date	(1a) Average Rate	(2a) Average Rate (All Dates)	(2b) Max Rate (All Dates)	(2c) Min Rate (All Dates)
4/2/1971	7.33%	7.85%	18.63%	2.65%
4/9/1971	7.31%	7.85%	18.63%	2.65%
4/16/1971	7.31%	7.85%	18.63%	2.65%
4/23/1971	7.31%	7.85%	18.63%	2.65%
4/30/1971	7.29%	7.85%	18.63%	2.65%
5/7/1971	7.38%	7.85%	18.63%	2.65%
5/14/1971	7.42%	7.85%	18.63%	2.65%
5/21/1971	7.44%	7.85%	18.63%	2.65%
5/28/1971	7.46%	7.85%	18.63%	2.65%
6/4/1971	7.52%	7.85%	18.63%	2.65%
6/11/1971	7.52%	7.85%	18.63%	2.65%
6/18/1971	7.54%	7.85%	18.63%	2.65%
6/25/1971	7.54%	7.85%	18.63%	2.65%
7/2/1971	7.54%	7.85%	18.63%	2.65%
7/9/1971	7.54%	7.85%	18.63%	2.65%
7/16/1971	7.60%	7.85%	18.63%	2.65%
7/23/1971	7.65%	7.85%	18.63%	2.65%
7/30/1971	7.69%	7.85%	18.63%	2.65%
8/6/1971	7.66%	7.85%	18.63%	2.65%
8/13/1971	7.73%	7.85%	18.63%	2.65%
8/20/1971	7.71%	7.85%	18.63%	2.65%

Figure 2: ALL function



Date	(1a) Average Rate	(3a) Average Rate (All Selected Dates)	(3b) Max Rate (All Selected Dates)	(3c) Min Rate (All Selected Dates)
1/7/2000	8.15%	5.04%	8.64%	2.65%
1/14/2000	8.18%	5.04%	8.64%	2.65%
1/21/2000	8.26%	5.04%	8.64%	2.65%
1/28/2000	8.25%	5.04%	8.64%	2.65%
2/4/2000	8.25%	5.04%	8.64%	2.65%
2/11/2000	8.36%	5.04%	8.64%	2.65%
2/18/2000	8.38%	5.04%	8.64%	2.65%
2/25/2000	8.31%	5.04%	8.64%	2.65%
3/3/2000	8.27%	5.04%	8.64%	2.65%
3/10/2000	8.23%	5.04%	8.64%	2.65%
3/17/2000	8.24%	5.04%	8.64%	2.65%
3/24/2000	8.23%	5.04%	8.64%	2.65%
3/31/2000	8.23%	5.04%	8.64%	2.65%
4/7/2000	8.20%	5.04%	8.64%	2.65%
4/14/2000	8.12%	5.04%	8.64%	2.65%
4/21/2000	8.16%	5.04%	8.64%	2.65%
4/28/2000	8.13%	5.04%	8.64%	2.65%
5/5/2000	8.28%	5.04%	8.64%	2.65%
5/12/2000	8.52%	5.04%	8.64%	2.65%

Figure 3: ALLSELECTED function

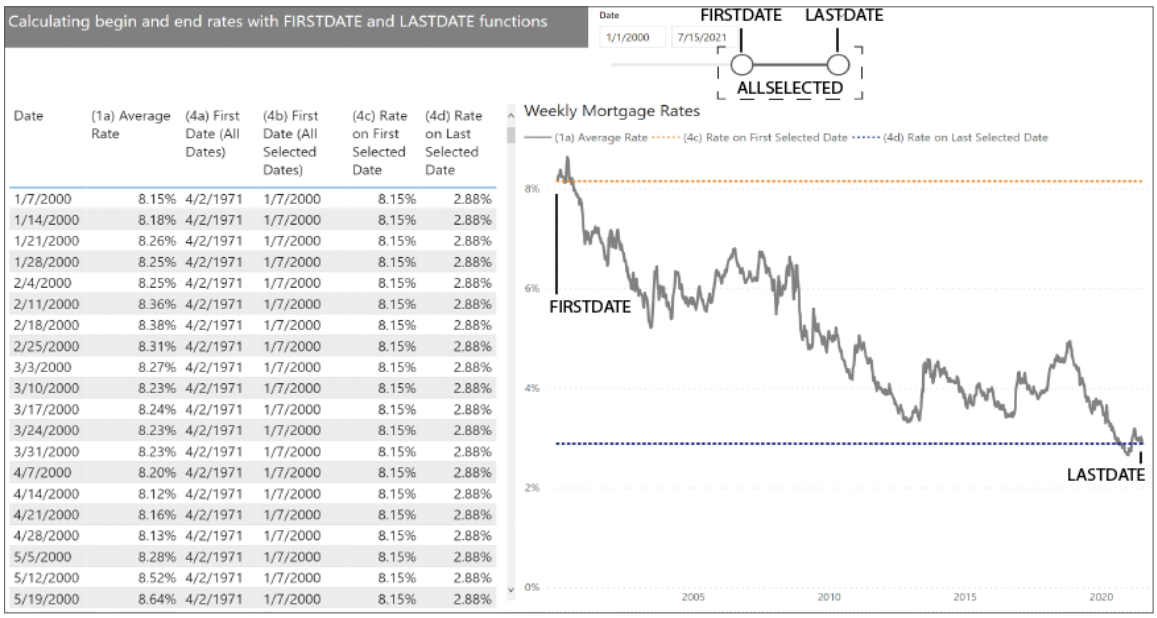


Figure 4: LASTDATE and FIRSTDATE functions

evaluation context for the measure, as you see in **Figure 3**. As you can see from the formulas below, you can swap out the ALL function with the ALLSELECTED function in the second parameter of the CALCULATE function.

```
(3a) Average Rate (All Selected Dates) =
CALCULATE(AVERAGE('Mortgage Rates'[Rate]),
ALLSELECTED('Mortgage Rates'[Date]))
```

```
(3b) Max Rate (All Selected Dates) =
CALCULATE(MAX('Mortgage Rates'[Rate]),
ALLSELECTED('Mortgage Rates'[Date]))
```

```
(3c) Min Rate (All Selected Dates) =
CALCULATE(MIN('Mortgage Rates'[Rate]),
ALLSELECTED('Mortgage Rates'[Date]))
```

When you narrow down the slicer's date range in **Figure 3**, the measures return the result over the selected date range instead of all the dates. You can see how the measures for the maximum and average rate over all the dates returns a much higher rate than the selected date range in the past twenty years because the interest rates in the early 1980s were higher than the rates today. You can see this by comparing the lines within the line charts in **Figure 3**, where the dashed lines indicate the measures calculated over the entire date range and the solid lines indicate the measures calculated over the selected date range.

FIRSTDATE and LASTDATE

Let's say you wanted to determine the mortgage rate on the first date or last date of data. You can use the FIRSTDATE and LASTDATE DAX functions in either the filtering or calculation component of the CALCULATE function. In the measure formulas below, you see how to calculate the first date over all the dates or just the elected date range and the results the measures return in **Figure 4**.

```
(4a) First Date (All Dates) =
CALCULATE(FIRSTDATE(
ALL('Mortgage Rates'[Date])))
```

```
(4b) First Date (All Selected Dates) =
CALCULATE(FIRSTDATE(
ALLSELECTED('Mortgage Rates'[Date])))
```

You can also use these calculated aggregations as filters within the CALCULATE expression by placing them into the second part of the CALCULATE function instead of the first part. In the formula below, you're filtering the data table by the selected date range, determining the first date within this narrower date range, and returning the maximum rate for this single date that the filters narrowed down the date range to, as you see in the table and line chart in **Figure 4**. You can leverage the MIN, AVERAGE, or even the SUM function as the aggregation for this measure. It returns the same result because the data only contains one rate for this filtered date.

```
(4c) Rate on First Selected Date =
CALCULATE(MAX('Mortgage Rates'[Rate]),
FIRSTDATE(ALLSELECTED('Mortgage Rates'[Date])))
```

You can also use the LASTDATE function in the filtering parameters to return the rate on the last selected date instead.

```
(4d) Rate on Last Selected Date =
CALCULATE(MAX('Mortgage Rates'[Rate]),
LASTDATE(ALLSELECTED('Mortgage Rates'[Date])))
```

For date, you can interchange the MIN and MAX functions specifically applied to dates with the FIRSTDATE and LASTDATE DAX functions respectively to return the same results.

ENDOFMONTH and STARTOFMONTH

The ENDOFMONTH and STARTOFMONTH functions work very much like the FIRSTDATE and LASTDATE functions as filter parameters in the CALCULATE function because you're moving each of the date pivot coordinate dates to a new date. Unlike the FIRSTDATE and LASTDATE functions however, the ENDOFMONTH and STARTOFMONTH functions only move the date pivot coordinates within each month instead of across the entire date range, as you see in **Figure 5**.

Think about the Filters First, Then the Calculation

Even though the actual calculation part of the CALCULATE function appears before the filter components, getting the filters to work can easily become the trickiest part of working with the DAX language. The DAX syntax looks easy at first, but this belies the challenges you may face with figuring out the logic behind how it works.

```
(5a) Rate Month End = CALCULATE([(1a) Average
Rate],ENDOFMONTH('Mortgage Rates'[Date]))
```

```
(5b) Rate Month Start = CALCULATE([(1a) Average
Rate],STARTOFMONTH('Mortgage Rates'[Date]))
```

Notice that these date filter functions reference the first or last date of data in each month, not necessarily the actual beginning or ending date of the month, unless those happen to coincide within the Mortgage Rate data source.

OPENINGBALANCEMONTH and CLOSINGBALANCEMONTH

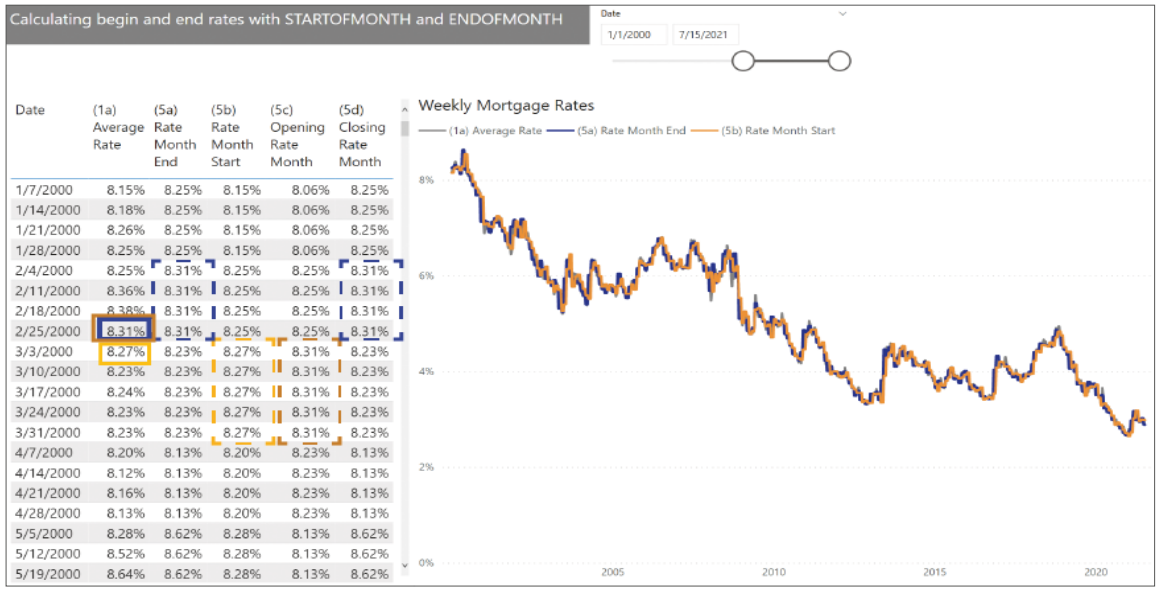
Let's explore the functions for the opening and closing monthly balances. These functions don't use the CALCULATE function directly, unless you place another measure in the first function parameter, which you see in the formulas below with the Average Rate measures. The second parameter contains the dates field you're trying to determine the opening or closing balance over. In **Figure 5**, you can see that these balance measures return the starting mortgage rate on the last date of the previous month, instead of the first date of the current month, like the Rate Month Start measure does.

```
(5c) Opening Rate Month = OPENINGBALANCEMONTH(
[(1a) Average Rate],'Mortgage Rates'[Date])
```

```
(5d) Closing Rate Month = CLOSINGBALANCEMONTH(
[(1a) Average Rate],'Mortgage Rates'[Date])
```

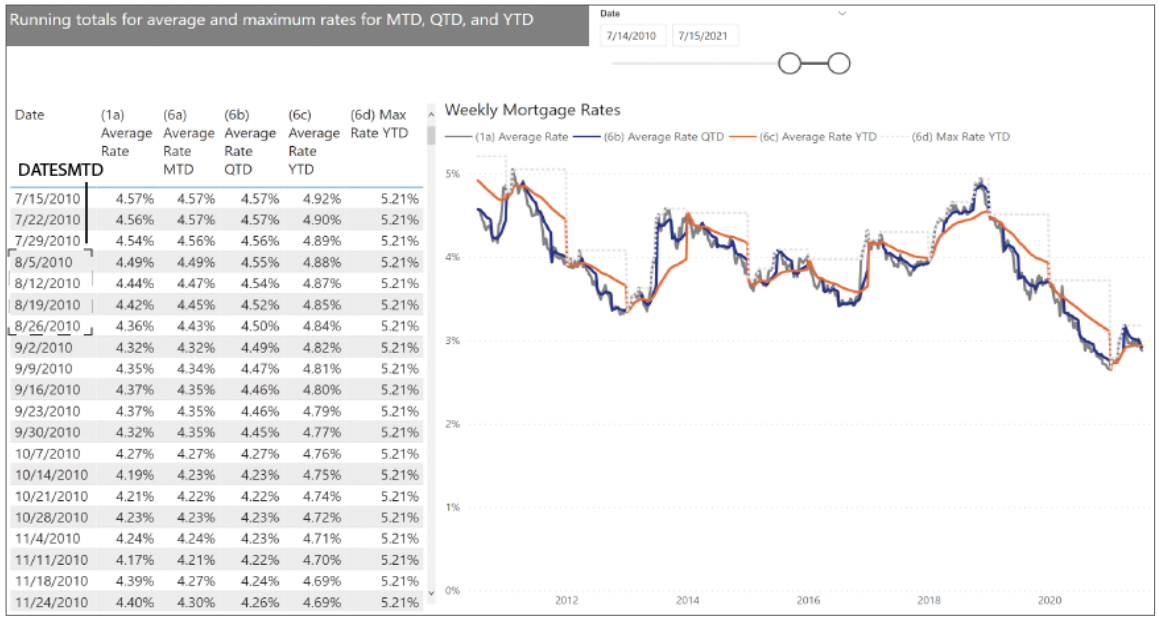
DATESMTD, DATESQTD, and DATESYTD

You can leverage the DATESMTD, DATESQTD, and DATESYTD DAX functions to create running totals for the maximum, minimum, and average rates. These filter functions expand the



Date	(1a) Average Rate	(5a) Rate Month End	(5b) Rate Month Start	(5c) Opening Rate Month	(5d) Closing Rate Month
1/7/2000	8.15%	8.25%	8.15%	8.06%	8.25%
1/14/2000	8.18%	8.25%	8.15%	8.06%	8.25%
1/21/2000	8.26%	8.25%	8.15%	8.06%	8.25%
1/28/2000	8.25%	8.25%	8.15%	8.06%	8.25%
2/4/2000	8.25%	8.31%	8.25%	8.25%	8.31%
2/11/2000	8.36%	8.31%	8.25%	8.25%	8.31%
2/18/2000	8.38%	8.31%	8.25%	8.25%	8.31%
2/25/2000	8.31%	8.31%	8.25%	8.25%	8.31%
3/3/2000	8.27%	8.23%	8.27%	8.31%	8.23%
3/10/2000	8.23%	8.23%	8.27%	8.31%	8.23%
3/17/2000	8.24%	8.23%	8.27%	8.31%	8.23%
3/24/2000	8.23%	8.23%	8.27%	8.31%	8.23%
3/31/2000	8.23%	8.23%	8.27%	8.31%	8.23%
4/7/2000	8.20%	8.13%	8.20%	8.23%	8.13%
4/14/2000	8.12%	8.13%	8.20%	8.23%	8.13%
4/21/2000	8.16%	8.13%	8.20%	8.23%	8.13%
4/28/2000	8.13%	8.13%	8.20%	8.23%	8.13%
5/5/2000	8.28%	8.62%	8.28%	8.13%	8.62%
5/12/2000	8.52%	8.62%	8.28%	8.13%	8.62%
5/19/2000	8.64%	8.62%	8.28%	8.13%	8.62%

Figure 5: STARTOFMONTH, ENDOFMONTH, OPENINGBALANCEMONTH, and CLOSINGBALANCEMONTH functions



Date DATESMTD	(1a) Average Rate	(6a) Average Rate MTD	(6b) Average Rate QTD	(6c) Average Rate YTD	(6d) Max Rate YTD
7/15/2010	4.57%	4.57%	4.57%	4.92%	5.21%
7/22/2010	4.56%	4.57%	4.57%	4.90%	5.21%
7/29/2010	4.54%	4.56%	4.56%	4.89%	5.21%
8/5/2010	4.49%	4.49%	4.55%	4.88%	5.21%
8/12/2010	4.44%	4.47%	4.54%	4.87%	5.21%
8/19/2010	4.42%	4.45%	4.52%	4.85%	5.21%
8/26/2010	4.36%	4.43%	4.50%	4.84%	5.21%
9/2/2010	4.32%	4.32%	4.49%	4.82%	5.21%
9/9/2010	4.35%	4.34%	4.47%	4.81%	5.21%
9/16/2010	4.37%	4.35%	4.46%	4.80%	5.21%
9/23/2010	4.37%	4.35%	4.46%	4.79%	5.21%
9/30/2010	4.32%	4.35%	4.45%	4.77%	5.21%
10/7/2010	4.27%	4.27%	4.27%	4.76%	5.21%
10/14/2010	4.19%	4.23%	4.23%	4.75%	5.21%
10/21/2010	4.21%	4.22%	4.22%	4.74%	5.21%
10/28/2010	4.23%	4.23%	4.23%	4.72%	5.21%
11/4/2010	4.24%	4.24%	4.23%	4.71%	5.21%
11/11/2010	4.17%	4.21%	4.22%	4.70%	5.21%
11/18/2010	4.39%	4.27%	4.24%	4.69%	5.21%
11/24/2010	4.40%	4.30%	4.26%	4.69%	5.21%

Figure 6: DATESMTD, DATESQTD, and DATESYTD functions

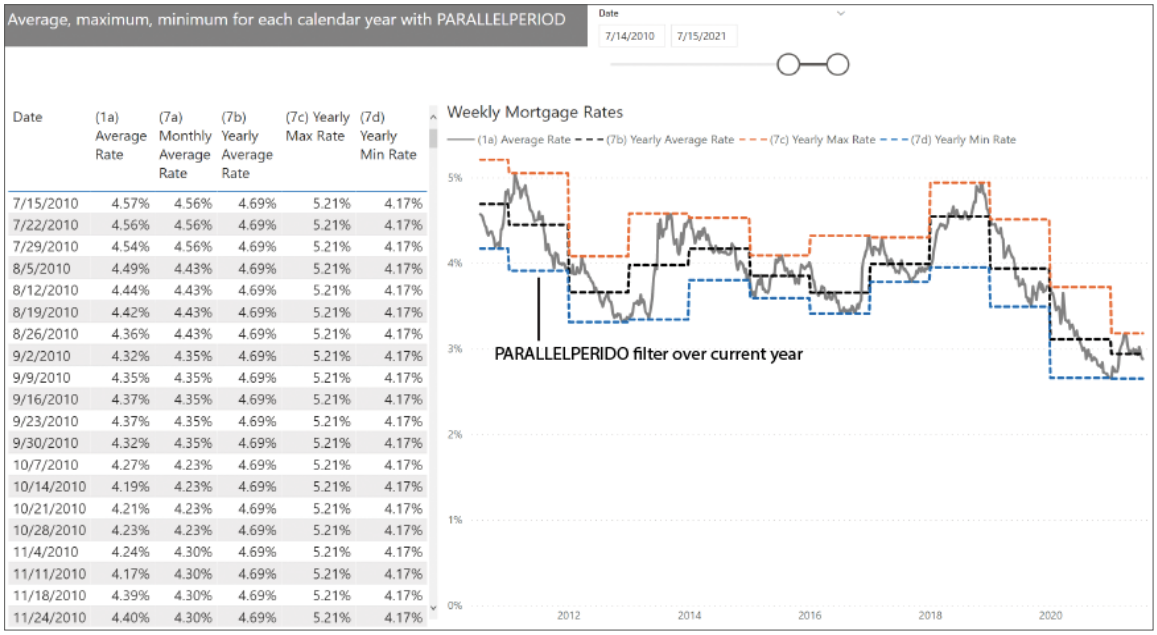


Figure 7: PARALLELPERIOD function

date range from each date pivot coordinate back to the start of the month, quarter, or year respectively. It then returns the aggregation calculation over this filtered date range.

```
(6a) Average Rate MTD =
CALCULATE([(1a) Average Rate],
DATESMTD('Mortgage Rates'[Date]))
```

```
(6b) Average Rate QTD =
CALCULATE([(1a) Average Rate],
DATESQTD('Mortgage Rates'[Date]))
```

```
(6c) Average Rate YTD =
CALCULATE(AVERAGE('Mortgage Rates'[Rate]),
DATESYTD('Mortgage Rates'[Date]))
```

```
(6d) Max Rate YTD =
CALCULATE(MAX('Mortgage Rates'[Rate]),
DATESYTD('Mortgage Rates'[Date]))
```

You can see in **Figure 6** how the running totals for the average and maximum numbers differ between the DATESMTD, DATESQTD, and DATESYTD DAX filter functions. The MAX Rate YTD measure returns the maximum year-to-date rate until the next date when the rate reaches a new maximum within this year so far, where it changes to this new maximum rate.

PARALLELPERIOD

You can also expand the date range filter through the PARALLELPERIOD DAX function. Unlike the running total DAX filter functions like DATESMTD, DATESQTD, and DATESYTD, which expand the date range back from the current date to the beginning of the specified period, the PARALLELPERIOD function expands the date range over the entire specified period. This means the measure returns the same result for each date within the period, as you see in **Figure 7**.

```
(7a) Monthly Average Rate =
CALCULATE([(1a) Average Rate],
PARALLELPERIOD('Mortgage Rates'[Date],0,MONTH))
```

```
(7b) Yearly Average Rate =
CALCULATE([(1a) Average Rate],
PARALLELPERIOD('Mortgage Rates'[Date],0,YEAR))
```

```
(7c) Yearly Max Rate =
CALCULATE(MAX('Mortgage Rates'[Rate]),
PARALLELPERIOD('Mortgage Rates'[Date],0,YEAR))
```

```
(7d) Yearly Min Rate =
CALCULATE(MIN('Mortgage Rates'[Rate]),
PARALLELPERIOD('Mortgage Rates'[Date],0,YEAR))
```

You can move the PARALLELPERIOD filter to a different specified time period by using another number instead of 0.

FILTER

The FILTER function lets you set up a filter over a table or field according to the condition criteria you specify. The FILTER function has two parts: the first part references the table or field you're filtering, and the second part configures the condition you're filtering this table for. Why can't you just set up the filter condition directly in the second part of the CALCULATE function here without using the FILTER function though? There are several other reasons why you'll need to employ the FILTER function for a measure to properly work and setting a condition equal to a measure is one of these scenarios.

Earlier, you calculated the minimum and maximum mortgage interest rate over the entire selected date range. But how can you determine on what date those maximum or minimum rates occur? You can create a measure calculation that determines the maximum or minimum rate over the entire date range, and then you'll match this rate to the date on which it occurs. In the formula below, you're leveraging the FILTER function to expand the filter to all the selected rows in the Mortgage Rate data table, then setting up a condition for the Rate field to equal the minimum mortgage rate over the filtered data table. Finally, you leverage the LASTDATE function to return the date at which this condition occurs.

```
(8a) Date of Min Rate (Incorrect) =
CALCULATE(LASTDATE('Mortgage Rates'[Date]),
FILTER(ALLSELECTED('Mortgage Rates'),
'Mortgage Rates'[Rate]
=[(3c) Min Rate (All Selected Dates)]))
```

However, you can see in **Figure 8** that this measure returns the last date of the entire data table instead of the date where the minimum rate occurs, so you'll need to explore other approaches for this calculation to properly work.

MAXX and MINX

Another way you can calculate a minimum rate over an entire selected date range is by leveraging the MINX function. Because this X function iterates over all the selected dates to return the minimum rate, it calculates the Average Rate measure for each of the pivot coordinates within this selected date range and returns the minimum rate across the entire date range. As you saw in **Figure 1**, adding the date pivot coordinates to the table means the average, maximum, and minimum measures return the same results, so you can use any of them within the MINX calculation.

```
(8b) Min Rate =
MINX(ALLSELECTED('Mortgage Rates'),
[(1a) Average Rate])
```

Now you can set this Min Rate measure equal to the mortgage rate field to determine when this minimum rate occurs. You can see that the formula takes the Min Rate calculation using the MINX function and places it into the filter conditions, but you can also reference the Min Rate measure itself. The FILTER function filters the selected data and then matches the rate to equal the results of the MINX measure calculation to ultimately return the date at which this condition occurs.

```
(8c) Date of Min Rate =
CALCULATE(LASTDATE('Mortgage Rates'[Date]),
FILTER(ALLSELECTED('Mortgage Rates'),
'Mortgage Rates'[Rate]
=MINX(ALLSELECTED('Mortgage Rates'),
[(1a) Average Rate])))
```

In **Figure 8**, you can see that this measure calculation returns the date you expect it to. You can do the same to calculate the maximum value over the same selected date range by utilizing the MAXX function instead of the MINX function.

```
(8d) Date of Max Rate =
CALCULATE(LASTDATE('Mortgage Rates'[Date]),
FILTER(ALLSELECTED('Mortgage Rates'),
'Mortgage Rates'[Rate]
=MAXX(ALLSELECTED('Mortgage Rates'),
[(1a) Average Rate])))
```

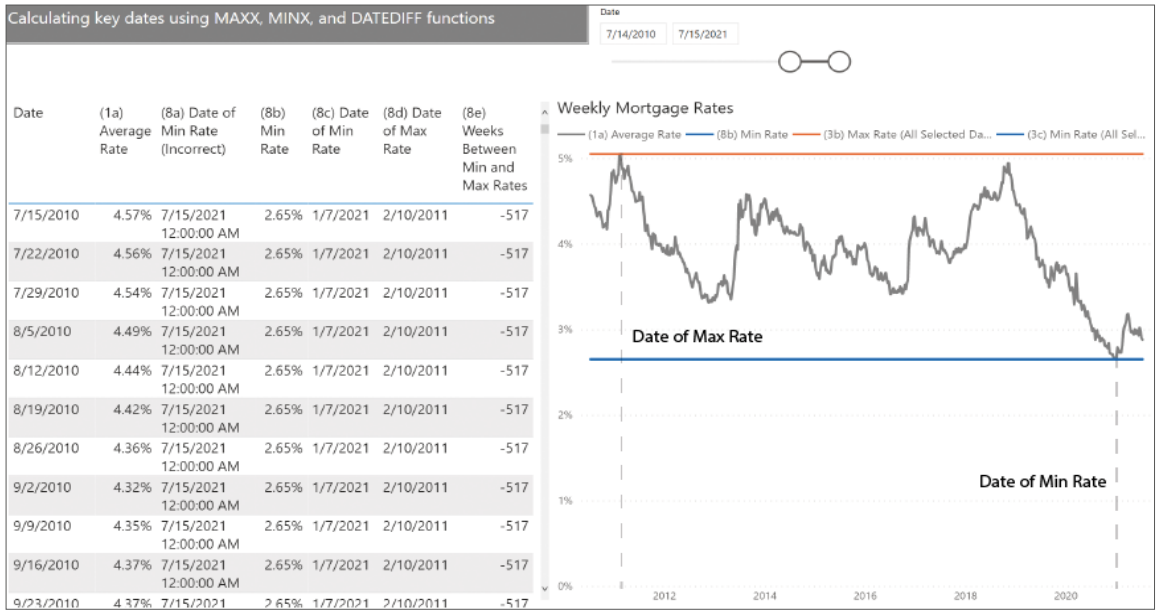
DATEDIFF

Now that you know when the maximum and minimum rates occur, how can you calculate the duration between these dates? You can set up a DAX measure to calculate this differently by leveraging DATEDIFF, which operates as a date calculation function instead of a filtering function. You can directly use the dates in the formula below because you already calculated the date when the minimum rate occurred and the date when the maximum rate occurred as measures of their own. You already determined the evaluation context for both these measures (**Figure 8**), and you can use them both directly to calculate this duration. Notice that the formula uses the period unit of WEEK to evaluate the duration but you can use other time periods like DAY as well.

```
(8e) Weeks Between Min and Max Rates =
DATEDIFF([(8c) Date of Min Rate],
[(8d) Date of Max Rate],WEEK)
```

Harvesting Parameters

DAX measures work on columns and tables instead of individual cells in a table (unlike Excel), so if you want to reference the date of each pivot coordinates in your mea-



Date	(1a) Average Rate	(8a) Date of Min Rate (Incorrect)	(8b) Min Rate	(8c) Date of Min Rate	(8d) Date of Max Rate	(8e) Weeks Between Min and Max Rates
7/15/2010	4.57%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
7/22/2010	4.56%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
7/29/2010	4.54%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
8/5/2010	4.49%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
8/12/2010	4.44%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
8/19/2010	4.42%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
8/26/2010	4.36%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
9/2/2010	4.32%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
9/9/2010	4.35%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
9/16/2010	4.37%	7/15/2021 12:00:00 AM	2.65%	1/7/2021	2/10/2011	-517
9/23/2010	4.37%	7/15/2021	2.65%	1/7/2021	2/10/2011	-517

Figure 8: MAXX, MINX, and FILTER functions

sure calculations directly, you'll need to set up a parameter harvesting measure that uses an aggregation function like LASTDATE or MAX to calculate each of the dates as its own measure value in this scenario.

```
(9a) Current Date =
CALCULATE(LASTDATE('Mortgage Rates'[Date]))
```

You can see in **Figure 9** that the results of the Current Date harvesting parameter exactly equal the date at each of the pivot coordinates. This calculated date isn't particularly helpful by itself, but it's quite helpful in other calculations.

PREVIOUSDAY

The PREVIOUSDAY function moves the date back a single day. If you have a contiguous field of dates in your Power BI model, you can set up a measure with the PREVIOUSDAY function to create a measure that works like a harvesting parameter, but for the previous day instead of the current day. However, as you can see from **Figure 9**, this measure returns blanks for this date range because the dates are weekly instead of daily, which means there's no yesterday's date for any pivot coordinate in the existing Date field

```
(9b) Previous Day =
CALCULATE(PREVIOUSDAY('Mortgage Rates'[Date]))
```

DATEADD

If you want to move the date pivot coordinates backward or forward by a specified number of days (or month, quarter, or years), you can leverage the DATEADD function to move each date pivot coordinate to another date by the specified number of periods in the filter formula. You can see below how to set up a measure calculating the previous week's Rate using the DATEADD filter function.

```
(9c) Rate 7 Days Earlier =
CALCULATE(AVERAGE('Mortgage Rates'[Rate]),
DATEADD('Mortgage Rates'[Date],-7,DAY))
```

However, if you take a closer look at the **Figure 9** sheet of the attached *.PBIX file that includes all the visuals and DAX measures, you'll notice that holidays for example often skew the weekly numbers so the previous week's date occurs six or eight days before the current date for example. This means that the measure returns no results for this week because this DATEADD filter fixes the interval between weeks to exactly seven days.

DATESBETWEEN

One workaround for these infrequently inconsistent weekly intervals is by leveraging the DATESBETWEEN function. This filtering function lets you expand the date range between a start date and end date. In the formula below, you're expanding the dates for each of the pivot coordinates to a date range between 14 days and one day before the current date (which therefore excludes the current date from the date range). The LASTDATE calculation then returns the most recent of this expanded date range, which represents the previous week's date as you see in the table of **Figure 9**.

```
(9d) Previous Week Date =
CALCULATE(LASTDATE('Mortgage Rates'[Date]),
DATESBETWEEN('Mortgage Rates'[Date],
[(9a) Current Date]-14,[(9a) Current Date]-1))
```

Calculating previous week's rate with FILTER, DATEADD, and DATESBETWEEN

Date 7/14/2010 7/15/2021

Date	Harvesting pararameter (9a) Current Date	(9b) Previous Day	(1a) Average Rate	(9c) Rate 7 Days Earlier	(9d) Previous Week Date	(9e) Days Since Previous Week	(9f) Previous Week Rate	(9g) Rate Change Since Previous Date
7/15/2021	7/15/2021		2.88%	2.90%	7/8/2021	7	2.90%	-0.02%
7/8/2021	7/8/2021		2.90%	2.98%	7/1/2021	7	2.98%	-0.08%
7/1/2021	7/1/2021		2.98%	3.02%	6/24/2021	7	3.02%	-0.04%
6/24/2021	6/24/2021		3.02%	2.93%	6/17/2021	7	2.93%	0.09%
6/17/2021	6/17/2021		2.93%	2.96%	6/10/2021	7	2.96%	-0.03%
6/10/2021	6/10/2021		2.96%	2.99%	6/3/2021	7	2.99%	-0.03%
6/3/2021	6/3/2021		2.99%	2.95%	5/27/2021	7	2.95%	0.04%
5/27/2021	5/27/2021		2.95%	3.00%	5/20/2021	7	3.00%	-0.05%
5/20/2021	5/20/2021		3.00%	2.94%	5/13/2021	7	2.94%	0.06%
5/13/2021	5/13/2021		2.94%	2.96%	5/6/2021	7	2.96%	-0.02%
5/6/2021	5/6/2021		2.96%	2.98%	4/29/2021	7	2.98%	-0.02%
4/29/2021	4/29/2021		2.98%	2.97%	4/22/2021	7	2.97%	0.01%
4/22/2021	4/22/2021		2.97%	3.04%	4/15/2021	7	3.04%	-0.07%
4/15/2021	4/15/2021		3.04%	3.13%	4/8/2021	7	3.13%	-0.09%
4/8/2021	4/8/2021		3.13%	3.18%	4/1/2021	7	3.18%	-0.05%
4/1/2021	4/1/2021		3.18%	3.17%	3/25/2021	7	3.17%	0.01%
3/25/2021	3/25/2021		3.17%	3.09%	3/18/2021	7	3.09%	0.08%
3/18/2021	3/18/2021		3.09%	3.05%	3/11/2021	7	3.05%	0.04%
3/11/2021	3/11/2021		3.05%	3.02%	3/4/2021	7	3.02%	0.03%
3/4/2021	3/4/2021		3.02%	2.97%	2/25/2021	7	2.97%	0.05%
2/25/2021	2/25/2021		2.97%	2.81%	2/18/2021	7	2.81%	0.16%

Figure 9: DATEDIFF, PREVIOUSDAY, DATEADD, DATESBETWEEN, and FILTER functions

Once you determine the date for the previous week, you can then use it in other measure calculations, like determining the days between each week and the previous week using the DATEDIFF function.

```
(9e) Days Since Previous Week =
DATEDIFF([(9d) Previous Week Date],
[(9a) Current Date],DAY)
```

Now you can use this measure directly in another new measure referencing the Days Since Previous Week measure instead of a fixed seven days within the DATEADD filter function.

```
(9f) Previous Week Rate =
CALCULATE([(1a) Average Rate],
DATEADD('Mortgage Rates'[Date],
-[(9e) Days Since Previous Week],DAY))
```

You can now directly subtract (or add, multiply, or divide) one measure from another to determine the mortgage rate change rate from the previous week.

```
(9g) Rate Change Since Previous Date =
[(1a) Average Rate]-[(9f) Previous Week Rate]
```

Keep Practicing

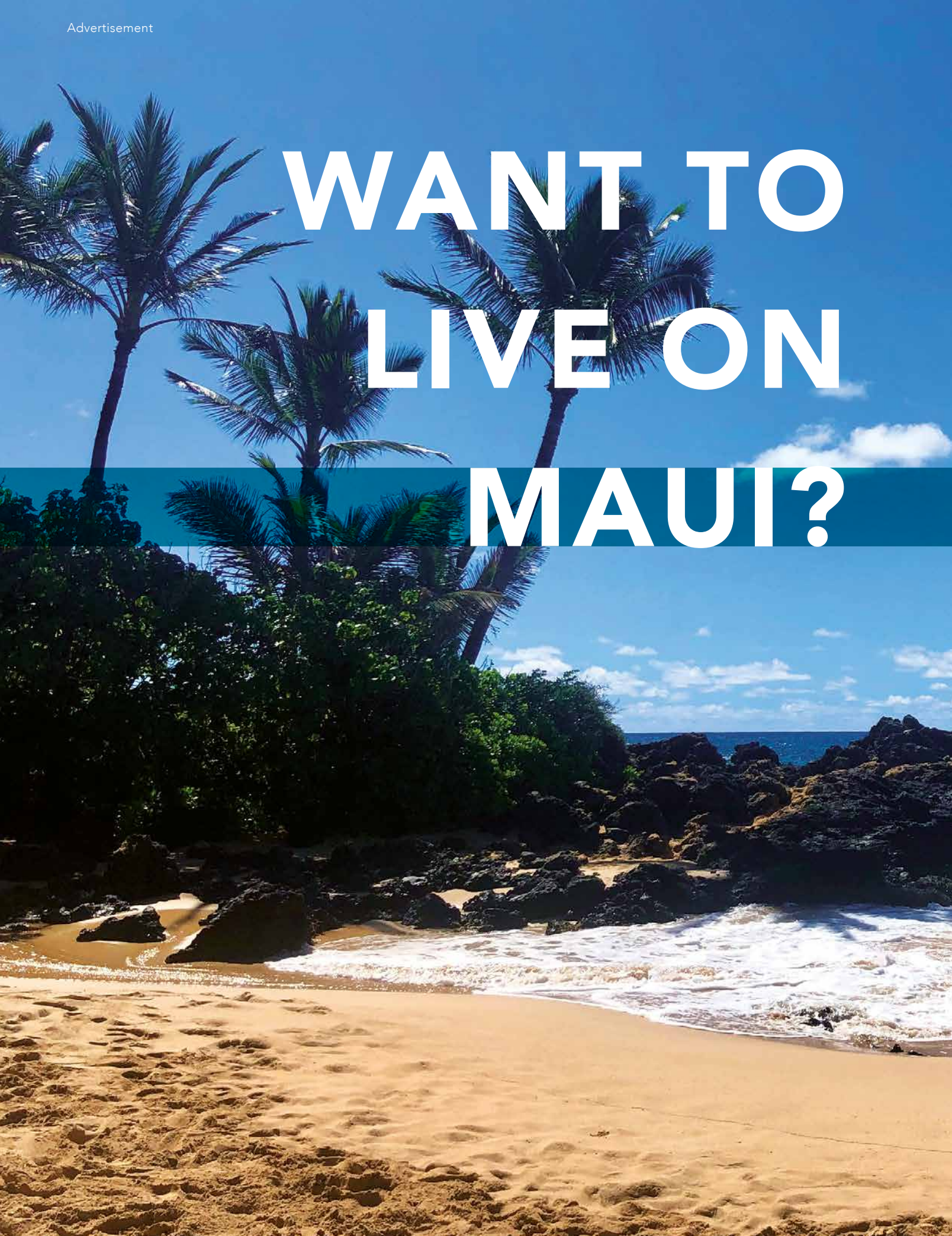
These measures represent just a few examples of how to leverage DAX modeling for dates within Power BI. As you can see, the DAX language enables you to create some powerful measures in a model like this. DAX measures take some time to become comfortable using, but testing them out on existing data offers one path forward. I encourage you to download the project file for this article and expand your own knowledge by trying out and testing DAX calculations on your own!

Helen Wall

CODE

Date DAX Filter Functions Change the Calculation Context in Two Ways

Filter functions specifically for dates in DAX can either move each of the date pivot coordinates you're evaluating the measure at to a new date, or it can expand the date range which you're calculating over.



IF YOU CAN WORK FROM HOME, WHY NOT MAKE PARADISE YOUR HOME?

The world has changed. Millions of people are working from home, and for many, that will continue way past the current crisis. Which begs the question: If you can work from home, then why not make your home in one of the world's premiere destinations and most desirable living areas?

The island of Maui in Hawai'i is not just a fun place to visit for a short vacation, but it is uniquely situated as a place to live. It offers great infrastructure and a wide range of things to do, not to mention a very high quality of life.

We have teamed up with *CODE Magazine* and Markus Egger to provide you information about living in Maui. Markus has been calling Maui his home for quite some time, so he can share his own experience of living in Maui and working from Maui in an industry that requires great infrastructure.

For more information, and a list of available homes, visit **www.Live-On-Maui.com**

Steve and Carol Olsen
Maui, Hawai'i

MAUI NO KA OI!

This loosely translates to "Maui is the best". As someone who has been calling Maui home for a while now, I can wholeheartedly confirm this. After having travelled the world, and after having lived in a variety of places, I find Maui to be truly unique.

Most people know Maui is a place to go for a week on vacation. And that is certainly great and very enjoyable. However, Maui is so much more! To me, Maui is the perfect mix that makes me feel like I am living on a tropical island yet being a developed place with great infrastructure and great quality of life. A lot of this is true for the Hawaiian Islands in general. But while Oahu (with its capital of Honolulu) is essentially a big city with a lot of people that always reminds me of Southern California, and while islands like Kauai or the Big Island of Hawai'i are a bit too "back to the roots" for me, Maui is just perfect. You can enjoy a great day at the beach or in nature, or you can go to a nice restaurant, the movies, or a concert. It's the quality of life provided by a modern place in the Western world, paired with a tropical island paradise.

Maui has many unique advantages. There is no hurricane season and no real rainy season. The weather is nice year-round, especially on the south-side of the island. There are no dangerous animals. Not even mosquitoes. How does 82-degree weather on a nice beach with a Mai Tai or Pina Colada sound? That's Maui for you!

As someone who works in the tech industry, good infrastructure is important to me. After all, I need to work as productively from Maui as I do when I am on the "mainland" (which is what we call the *continental US* here in Hawai'i). I have a 300Mbit internet connection going to my home, and it is inexpensive. My connection to other parts of the world is better and less expensive here than it is on most main-land locations. We have the same stores and supermarkets as everywhere else in the US. Schools are decent. The same is true for healthcare. Flight connections are great, and it is not at all difficult to travel from and to Maui.

I live on the south-side of the island in an area called "Kihei". Especially the southern parts of Kihei, known as "Wailea" and "Makena", are the areas I truly recommend to anyone (although there are other places worth considering also). I like this area for its great quality of housing, low crime, great weather, and the world's greatest beaches. I enjoy playing a round of golf or going to a great restaurant with friends. When I want to feel like I'm on vacation for an hour or two, I swing by one of the hotels for a snack at the pool bar. When I feel like exercising, I ride my bike along the ocean or go for a hike into the jungle or across lava fields.

As we have been going through the COVID-19 crisis, it has become more and more clear how great a location Maui is. For one, the warm weather and outdoor living have kept the COVID-19 numbers low, and the quality of life high. And while nobody wants to be hospitalized, it has been nice to know that we have better healthcare here than other tropical locations. (Essentially the same healthcare as anywhere else in the US.) While we also had to deal with a lockdown, and many restaurants and hotels have been closed, we have several places that are not just open, but since everything happens outdoors, many are perfectly safe to visit. I really can't think of a better place to weather this pandemic than Maui.

So yes: Maui No Ka Oi! To me, there isn't another place that even comes close. I have a long list of other places I enjoy visiting that are awesome too. Do I want to go to Bora Bora, Singapore, or many other great locations? Sure, I do! But what do you do in Bora Bora after two weeks? Maui on the other hand is a great place to set up a life and stay for good.

I would love to see you on Maui in the future. Maybe we can share one of those Mai Tais on the beach. I recommend talking to Carol Olsen, who has been helping me with all my real estate needs in South Maui. Moving to Maui has been the best decision of my life. I am sure you would enjoy it too!

Markus Egger
Publisher, *CODE Magazine*

MAUI PROPERTIES

Wailea Homes

Wailea Condos

Wailea Land

Kihei Homes

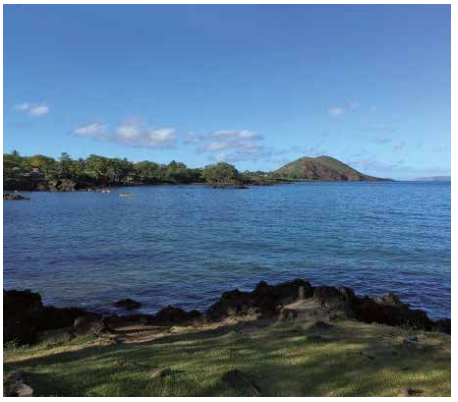
Kihei Condos

Kihei Land

Makena Home

Makena Condo

Makena Land



Building Dashboards Using Bokeh

Most data analysts and scientists using Python are familiar with plotting libraries such as matplotlib, Seaborn, and Plotly. These libraries are very useful for doing data exploration, as well as visualizing and generating graphics for reports. However, what if you want to generate all these charts and graphics and let your users view them on Web browsers? Also, it would be useful if the users can interact with your charts dynamically and drill down into the details they want to see. For this, you can use Bokeh. In this article, I'll walk you through the basics of Bokeh: how to install it, how to create basic charts, how to deploy them on Web servers, and more. So let the fun begin!

Wei-Meng Lee
weimenglee@learn2develop.net
http://www.learn2develop.net
@weimenglee

Wei-Meng Lee is a technologist and founder of Developer Learning Solutions (http://www.learn2develop.net), a technology company specializing in hands-on training on the latest technologies. Wei-Meng has many years of training experience and his training courses place special emphasis on the learning-by-doing approach. His hands-on approach to learning programming makes understanding the subject much easier than reading books, tutorials, and documentation. His name regularly appears in online and print publications such as DevX.com, MobiForge.com, and CODE Magazine.

What Is Bokeh

Bokeh is a Python library for creating interactive visualizations for Web browsers. Using Bokeh, you can create dashboards—a visual display of all your key data. What's more, Bokeh powers your dashboards on Web browsers using JavaScript, all without needing you to write any JavaScript code.

> Dashboards provide all your important information in a single page and are usually used for presenting information such as KPIs and sales results.

Installing Bokeh

For this article, I'll be using Anaconda for my Python installation. You can download Anaconda from https://www.anaconda.com/products/individual. Once Anaconda is installed, the next step is to install the Bokeh library.

To install the Bokeh library, simply use the **pip** command at the Anaconda Prompt/Terminal:

```
$ pip install bokeh
```

Creating Basic Glyphs

In Bokeh, a **plot** is a container that holds all the various objects (such as **renderers**, **glyphs**, or **annotations**) of a visualization.

Glyphs are the basic visual building blocks of Bokeh plots. The simplest way to get started is to create a simple chart using the various glyphs methods.

> In Bokeh, glyphs are the geometrical shapes (lines, circles, rectangles, etc.) in a chart.

In Jupyter Notebook, type the following code in a new cell:

```
from bokeh.plotting import figure, output_file,
                           show
import random

count = 10
x = range(count)
y = random.sample(range(0, 101), count)

p = figure()         # figure is a type of plot

# using various glyph methods to create scatter
# plots of different marker shapes
p.circle  (x, y, size=30,
           color='red',
           legend_label='circle')
p.line    (x, y, width=2,
           color='blue',
           legend_label='line')
p.triangle(x, y, size=10,
           color='gold',
           legend_label='triangle')

output_file('my_first_graph.html')  # name the
                                    # output
                                    # file
show(p)                             # show the
                                    # graph
```

The **figure()** function returns a **plot** object, which allows you to create various types of charts using the various glyphs methods. The **circle()**, **line()**, and **triangle()** glyph methods creates scatter plots with various marker shapes.

In the above, you also specified the filename of the page as **my_first_graph.html**, using the **output_file()** function;

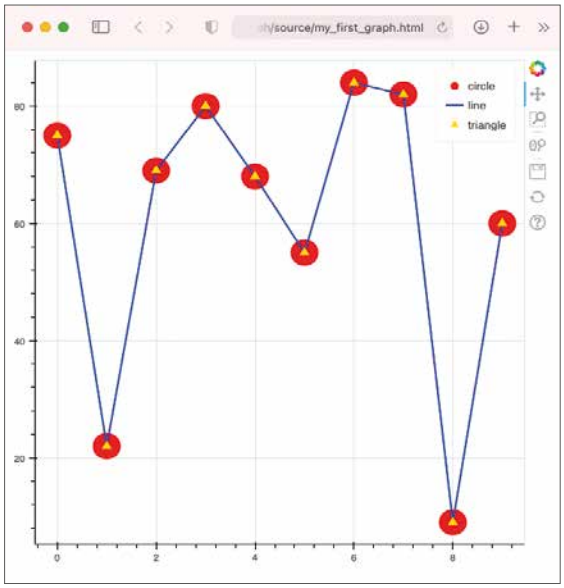


Figure 1: Basic plots created using Bokeh

if you don't specify the output filename, a random file will be generated every time you run the code (and hence a new tab is created on your browser). When you run the cell containing the above code, you'll see the scatter plot containing the various markers as shown in **Figure 1** in a new tab page on your Web browser.

Observe the toolbar displayed on the right side of the plot (see **Figure 2**).

The toolbar contains the following tools:

- **Bokeh:** Link to the Bokeh page
- **Pan:** Drag the chart to move it around
- **Box Zoom:** Use your mouse to select part of the plot to zoom in
- **Wheel Zoom:** Use the wheel on your mouse to zoom in and out of the plot
- **Save:** Save and download a copy of your chart in PNG format
- **Reset:** Restore the plot to its original state
- **Help:** Link to the page on Bokeh plot tools

The toolbar is customizable and in a later section, I'll show you how to hide/add tools in the toolbar.

Vertical Bars

Bar charts can be drawn easily in Bokeh using the **vbar()** method. The following example shows how to display a bar chart by supplying data through a **ColumnDataSource** object:

```
from bokeh.plotting import figure, output_file,
                    show
from bokeh.models import ColumnDataSource
import pandas as pd
import random

# create a Pandas dataframe
df = pd.DataFrame(dict(
    x = [1, 2, 3, 4, 5],
    y = random.sample(range(1,11), 5),
))

# create a ColumnDataSource obj using a
# dataframe
source = ColumnDataSource(data=df)

p = figure(plot_width=400,
           plot_height=400)

p.vbar(x = 'x',
       top = 'y',
       source = source,
       width = 0.5,
       bottom = 0,
       color = 'lightgreen')

output_file('vbar.html')
show(p)
```

In the previous example, you provided the data to be plotted using lists. Although this is perfectly fine, it's better to use a **ColumnDataSource** object as the supplier of data to your plots. In fact, when you pass data to your Bokeh plots using lists, Bokeh automatically creates a **ColumnDataSource** object behind the scenes. Using a **ColumnDataSource**, you can enable more advanced capabilities for your Bokeh plots, such as sharing data between plots, filtering data, etc.

> The ColumnDataSource object provides the data to the glyphs of your plot. It provides advanced capabilities for your Bokeh plots, such as sharing data between plots, filtering data, etc.

Figure 2: The items in the toolbar

In the above example, the **ColumnDataSource** object is passed in through the **source** parameter of the **vbar()** method:

```
p.vbar(x = 'x',                  # x-axis
       top = 'y',                # y-axis
       source = source,
       width = 0.5,              # width of bar
       bottom = 0,               # starts from
                                 # bottom
       color = 'lightgreen')
```

Printing the **source.data** property shows the following (formatted for clarity):

```
print(source.data)
# {
#   'index': array([0, 1, 2, 3, 4]),
#   'x': array([1, 2, 3, 4, 5]),
#   'y': array([ 7, 10,  5,  9,  2])
# }
```

The value "x" in the **vbar()** method refers to the value of the "x" key in the value of **source.data**, and "y" refers to the value of the "y" key. **Figure 3** shows the vertical bar chart generated by the **vbar()** method.

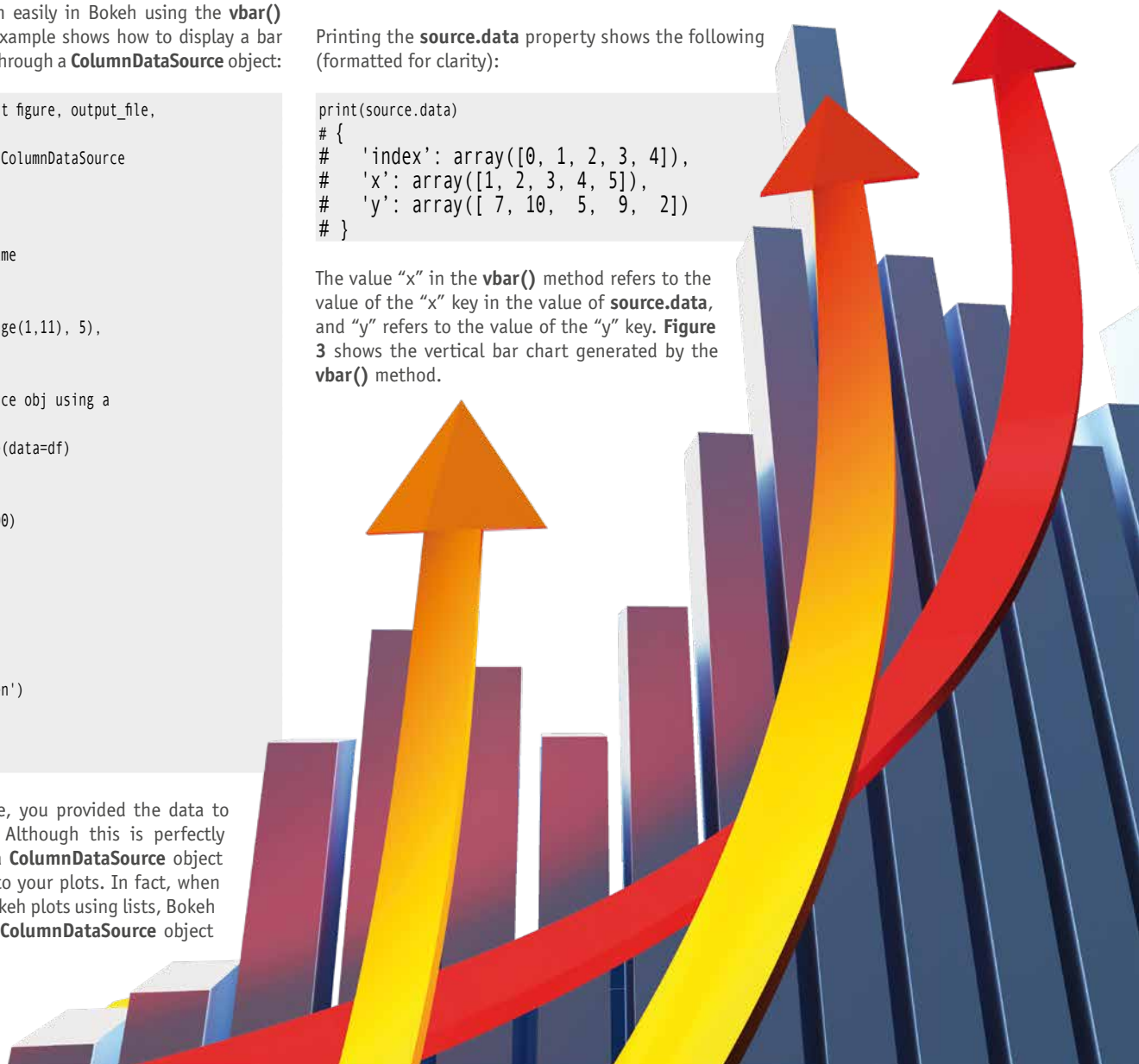


Figure 3: The plot created using the vbar() method

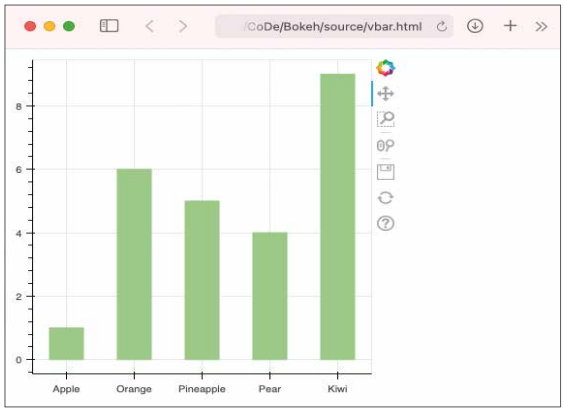


Figure 4: The updated vertical bar plot

What happens if the values for the x-axis are categorical values, such as a list of fruits. Consider the following:

```
df = pd.DataFrame(dict(
    fruits =
        ['Apple','Orange','Pineapple',
         'Pear','Kiwi'],
    sales = random.sample(range(1,11), 5),
))
```

In this case, in order to plot the chart correctly, you need to specify the **x_range** parameter in the **figure()** function and pass it the categorical values, like this:

```
from bokeh.plotting import figure, output_file,
                           show
from bokeh.models import ColumnDataSource
import pandas as pd
import random

df = pd.DataFrame(dict(
    fruits =
        ['Apple','Orange','Pineapple',
         'Pear','Kiwi'],
    sales = random.sample(range(1,11), 5),
))

source = ColumnDataSource(data=df)

p = figure(plot_width=400,
           plot_height=400,
           x_range=source.data['fruits'])

p.vbar(x = 'fruits',
       top = 'sales',
       source = source,
       width = 0.5,
       bottom = 0,
       color = 'lightgreen')

output_file('vbar.html')
show(p)
```

Figure 4 shows the updated vertical bar plot.

Displaying Legends and Colors

A chart needs a legend to be useful. Bokeh provides a collection of palettes for color mapping (https://docs.bokeh.org/en/latest/docs/reference/palettes.html). Consider the palette named Spectral (see **Figure 5**).

You can import the Spectral palette and examine its content:

```
from bokeh.palettes import Spectral
Spectral
# {3: ('#99d594', '#ffffbf', '#fc8d59'),
#  4: ('#2b83ba', '#abdda4', '#fdae61',
       '#d7191c'),
#  5: ('#2b83ba', '#abdda4', '#ffffbf',
       '#fdae61',
#      '#d7191c'),
#  ...
```

Each row in the palette contains a collection (tuples) of colors. To use the colors in the palette, use the **factor_cmap()** function to create a dictionary of colors, like the following example:

```
from bokeh.plotting import figure, output_file,
                           show
from bokeh.models import ColumnDataSource
import pandas as pd
import random

from bokeh.palettes import Spectral
from bokeh.transform import factor_cmap

df = pd.DataFrame(dict(
    fruits =
        ['Apple','Orange','Pineapple',
         'Pear','Kiwi'],
    sales = random.sample(range(1,11), 5),
))

source = ColumnDataSource(data=df)

p = figure(plot_width=400,
           plot_height=400,
           x_range=source.data['fruits'])

p.vbar(source = source,
       x = 'fruits',
       top = 'sales',
       width = 0.5,
       bottom = 0,
       color = 'lightgreen',
```

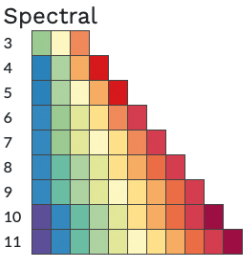


Figure 5: The various colors in the Spectral palette

Figure 6: Displaying the vertical bar plot using the Spectral palette

Figure 7: Plotting using the hbar() method

```
      legend_field = 'fruits',
      line_color = 'black',
      fill_color = factor_cmap('fruits',
          palette=Spectral[6],
          factors=source.data['fruits'])
     )

output_file('vbar.html')
show(p)
```

Figure 6 shows the vertical bar chart displayed in colors defined in the Spectral palette as well as with the legend.

Horizontal Bars

In addition to vertical bar chart, horizontal bar chart is also sometimes useful. The following code snippet shows how to display a horizontal bar chart using the **hbar()** method:

```
from bokeh.plotting import figure, output_file,
                           show
from bokeh.models import ColumnDataSource
import pandas as pd
import random

from bokeh.palettes import Spectral
from bokeh.transform import factor_cmap

df = pd.DataFrame(dict(
    fruits =
        ['Apple','Orange','Pineapple',
         'Pear','Kiwi'],
    sales = random.sample(range(1,11), 5),
))

source = ColumnDataSource(data=df)

p = figure(plot_width=400,
           plot_height=400,
           y_range=source.data['fruits'])

p.hbar(source = source,
       y = 'fruits',
       height = 0.5,
       left = 0,
       right = 'sales',
       color = 'lightgreen',
       legend_field = 'fruits',
       line_color = 'black',
       fill_color = factor_cmap('fruits',
           palette=Spectral[6],
           factors=source.data['fruits'])
      )

output_file('hbar.html')
show(p)
```

Figure 7 shows the output using the colors from the Spectral palette.

Stacked Bars

A Stacked bar chart is a variation of bar chart that allows you to compare numeric values between values of a categorical variable. Suppose you have a dataframe containing the following rows and columns:

```
    teams  males  females
0  Team A      9        8
1  Team B      3        4
2  Team C      4       10
3  Team D      1        3
4  Team E      6        1
```

It would be useful to use a stacked bar to show the number of males and females in each team. The following code snippet shows how to use the **vbar_stack()** method to display a stack bar showing the composition of genders in each team:

```
from bokeh.plotting import figure, output_file,
                           show
from bokeh.models import ColumnDataSource
import pandas as pd
import random

from bokeh.palettes import Category20

df = pd.DataFrame(dict(
    teams =
        ['Team A','Team B','Team C',
         'Team D','Team E'],
    males = random.sample(range(1,11), 5),
    females = random.sample(range(1,11), 5),
))

source = ColumnDataSource(data=df)
```

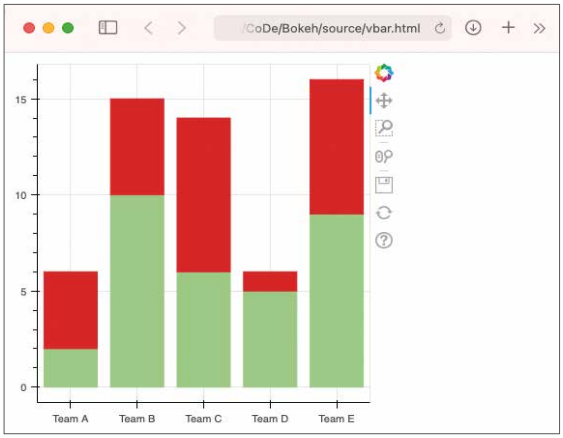


Figure 8: Plotting a stacked bar plot

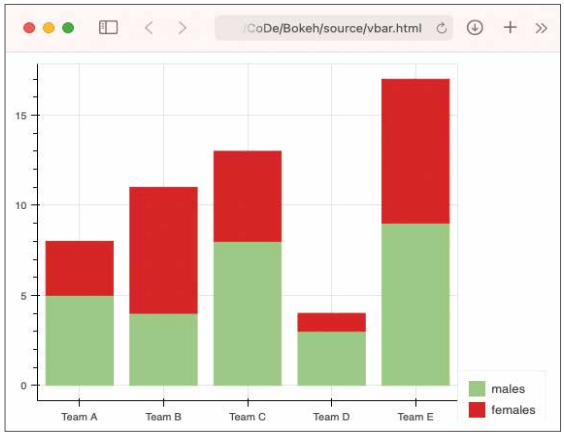


Figure 9: The stacked bar plot with the legend

```
p = figure(plot_width=400,
           plot_height=400,
           x_range=source.data['teams'])

v = p.vbar_stack(['males', 'females'],
                 source = source,
                 x = 'teams',
                 width=0.8,
                 color=Category20[8][5:7],
                )

output_file('vbar.html')
show(p)
```

Figure 8 shows the output for the stacked bars.

What the stacked bars lack is a legend to indicate what the red and green bars represent. So let's add a legend to the plot using the **Legend** class:

```
...

p = figure(plot_width=600,
           plot_height=400,
           x_range=source.data['teams'])

v = p.vbar_stack(['males', 'females'],
                 source = source,
                 x = 'teams',
                 width=0.8,
                 color=Category20[8][5:7],
                )

from bokeh.models import Legend
legend = Legend(items=[
    ("males",   [v[0]]),
    ("females", [v[1]]),
], location=(0, -30))

p.add_layout(legend, 'right')

output_file('vbar.html')
show(p)
```

You can specify the position of the legend and where to add it to the plot. **Figure 9** shows the stacked bars with the legend.

Pie Charts

A pie chart is another popular way to illustrate numerical proportion. In Bokeh, you can use the **wedge()** method of the **plot** object to display a pie chart. In the following code snippet, you have a dataframe containing a list of fruits and their sales:

```
      fruits  sales
0      Apple     10
1     Orange      8
2  Pineapple      5
3       Pear      4
4       Kiwi      7
5     Banana      2
6     Papaya      9
7     Durian      1
8      Guava      3
```

To display a pie chart showing the sales of the various fruits, you need to calculate the angle of each slice:

```
df['angles'] = \
    df['sales'] / df['sales'].sum() * \
    2*math.pi
```

In addition, you can also assign specific color to each slice:

```
df['colors'] = Category20c[len(df['fruits'])]
```

The following code snippet outputs the pie chart as shown in **Figure 10**:

```
from bokeh.plotting import figure, output_file,
                           show
from bokeh.models import ColumnDataSource
from bokeh.palettes import Category20c
from bokeh.transform import cumsum
import pandas as pd
import random
import math

df = pd.DataFrame(dict(
    fruits =
        ['Apple','Orange','Pineapple',
         'Pear','Kiwi','Banana',
         'Papaya','Durian','Guava'],
    sales = random.sample(range(1,11), 9),
))
```

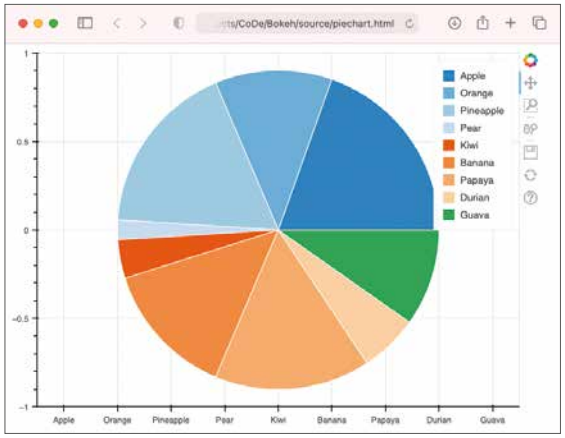


Figure 10: Plotting a pie plot using the wedge() method

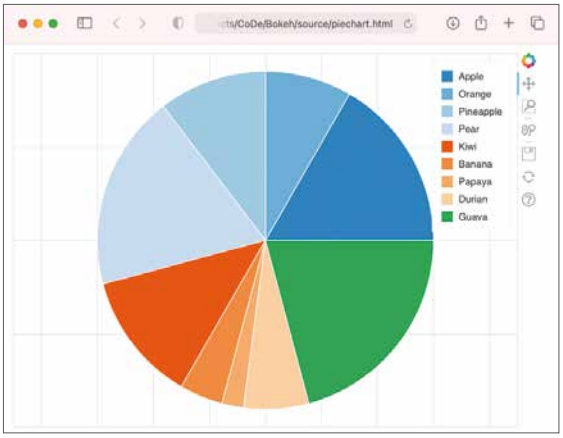


Figure 11: Turning off the axes for the pie plot

```
df['colors'] = Category20c[len(df['fruits'])]
df['angles'] = \
    df['sales'] / df['sales'].sum() * \
    2*math.pi

source = ColumnDataSource(data=df)

p = figure(plot_width=700,
           plot_height=500,
           x_range=source.data['fruits'])

p.wedge(x=len(df['fruits'])/2,
        y=0,
        radius=3.0,
        start_angle=cumsum('angles',
            include_zero=True),
        end_angle=cumsum('angles'),
        line_color="white",
        fill_color='colors',
        legend_field='fruits',
        source=source)

output_file('piechart.html')
show(p)
```

Notice that for the pie chart, it doesn't really make sense for the axes to be visible, and so you can turn them off using the **axis.visible** property:

```
p.wedge(x = len(df['fruits'])/2,
        ...)

p.axis.visible = False

output_file('piechart.html')
show(p)
```

Figure 11 shows the pie chart without the axes.

A pie chart without label on it may not be very useful, so let's now add text to each slice of the pie plot using the **LabelSet** class:

```
from bokeh.plotting import figure, output_file,
                           show
from bokeh.models import ColumnDataSource,
                         LabelSet
```

```
import pandas as pd
import random

from bokeh.palettes import Category20c

import math
from bokeh.transform import cumsum

df = pd.DataFrame(dict(
    fruits = ['Apple','Orange','Pineapple',
              'Pear','Kiwi','Banana',
              'Papaya','Durian','Guava'],
    sales = random.sample(range(1,11), 9),
))

df['colors'] = Category20c[len(df['fruits'])]
df['angles'] = \
    df['sales'] / df['sales'].sum() * \
    2*math.pi
df['label'] = (df['fruits'] + "-" +
               df['sales'].astype(
               str)).str.pad(30, side='left')

source = ColumnDataSource(data=df)

p = figure(plot_width=700,
           plot_height=500,
           x_range=source.data['fruits'])

p.wedge(x = len(df['fruits'])/2,
        y = 0,
        radius = 3.0,
        start_angle = cumsum('angles',
                     include_zero=True),
        end_angle = cumsum('angles'),
        line_color = 'white',
        fill_color = 'colors',
        legend_field = 'fruits',
        source = source)

p.axis.visible = False

labels = LabelSet(x = len(df['fruits'])/2,
                  y = 0,
                  text = 'label',
                  angle = cumsum('angles',
                          include_zero=True),
```

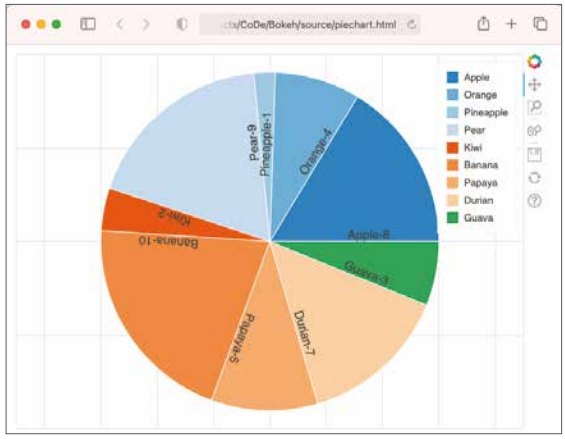


Figure 12: The pie plot with each slice clearly labeled

```
                source = source,
                render_mode = 'canvas')

p.add_layout(labels)

output_file('piechart.html')
show(p)
```

Figure 12 shows the much-improved pie plot with each slice labeled with the name and the sales.

Dashboard Layouts

Now that you have learned the basics of Bokeh plots, let's not forget why you want to use Bokeh in the first place. If you just want to plot some simple charts, you could jolly well use matplotlib or Seaborn. Using Bokeh, you could layout multiple charts in a single page.

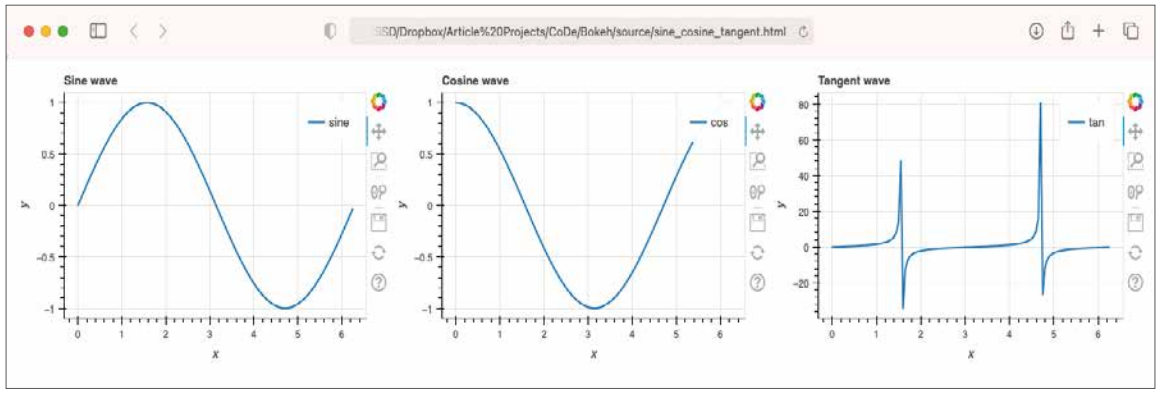


Figure 13: Laying out the plots in row format

Listing 1: Using the row layout

```
from bokeh.plotting import figure, output_file, show
from bokeh.models import ColumnDataSource
from bokeh.layouts import row, column, gridplot

import pandas as pd
import numpy as np
import math

x = np.arange(0, math.pi*2, 0.05)

df = pd.DataFrame(dict(
    x = x,
    sin = np.sin(x),
    cos = np.cos(x),
    tan = np.tan(x)
))

source = ColumnDataSource(data=df)

#-------------------------------
#          Sine Wave
#-------------------------------
p1 = figure(title = "Sine wave",
            x_axis_label = 'x',
            y_axis_label = 'y',
            plot_width=300,
            plot_height=300)
p1.line('x', 'sin',
        source=source,
        legend_label = "sine",
        line_width = 2)

#-------------------------------
#          Cosine Wave
#-------------------------------
p2 = figure(title = "Cosine wave",
            x_axis_label = 'x',
            y_axis_label = 'y',
            plot_width=300,
            plot_height=300)
p2.line('x', 'cos',
        source=source,
        legend_label = "cos",
        line_width = 2)

#-------------------------------
#          Tangent Wave
#-------------------------------
p3 = figure(title = "Tangent wave",
            x_axis_label = 'x',
            y_axis_label = 'y',
            plot_width=300,
            plot_height=300)
p3.line('x', 'tan',
        source=source,
        legend_label = "tan",
        line_width = 2)

#-------------------------------
#       Laying out in row
#-------------------------------
plot = row([p1, p2, p3],
           sizing_mode='stretch_both')

output_file("sine_cosine_tangent.html")
show(plot)
```

Bokeh provides several layout options:

- Row
- Column
- Gridplot

Listing 1 shows how you can use the **row** layout to display three charts. You make use of the **sizing_mode** parameter to configure how the charts are displayed when the display changes in size, as you can see in this snippet:

```
#-------------------------------
#       Laying out in row
#-------------------------------
plot = row([p1, p2, p3],
           sizing_mode='stretch_both')
```

Figure 13 shows the output of the three charts: Sine wave, Cosine wave, and Tangent wave.

You can also use the **column** layout:

```
#-------------------------------
#      Laying out in column
#-------------------------------
plot = column([p1, p2, p3],
              sizing_mode='stretch_both')
```

Figure 14 shows the charts displayed vertically in a column layout.

Finally, you can also display the charts in a **gridplot**:

```
#-------------------------------
#     Laying out in gridplot
#-------------------------------
plot = gridplot([p1, p2, p3],
                ncols=2,
                sizing_mode='stretch_both')
```

Based on the value of the **ncols** parameter, Bokeh automatically lays out your plot from left to right, top to bottom, as shown in **Figure 15**.

Observe that the Tangent chart is displayed on the left side on the second row. If you want to explicitly specify how the charts are displayed, you can leave out the **ncols** parameter and pass in your plots objects as lists:

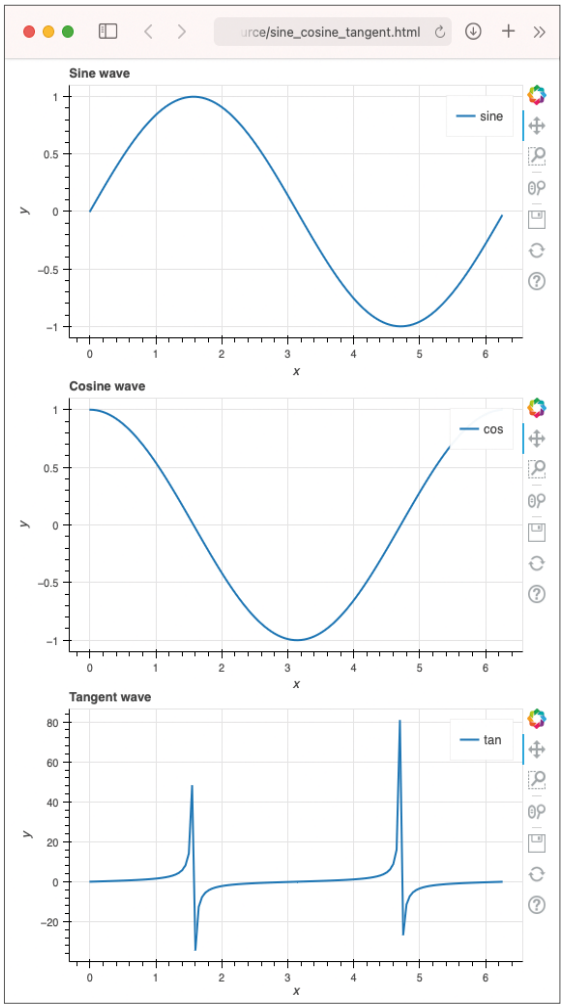


Figure 14: Laying out the plots in column format

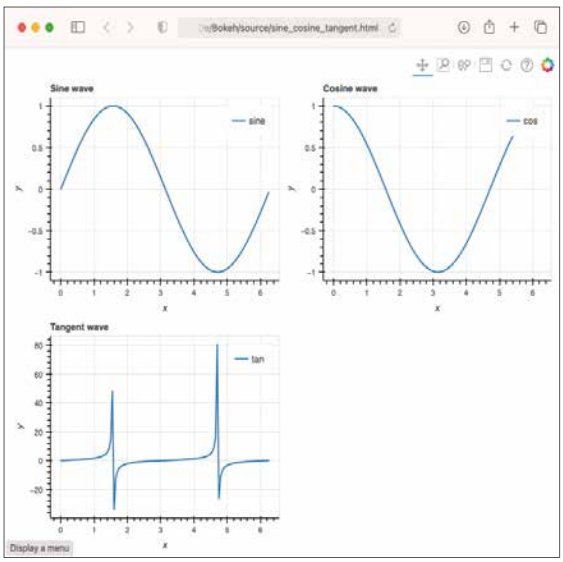


Figure 15: Laying out the plots in gridplot format

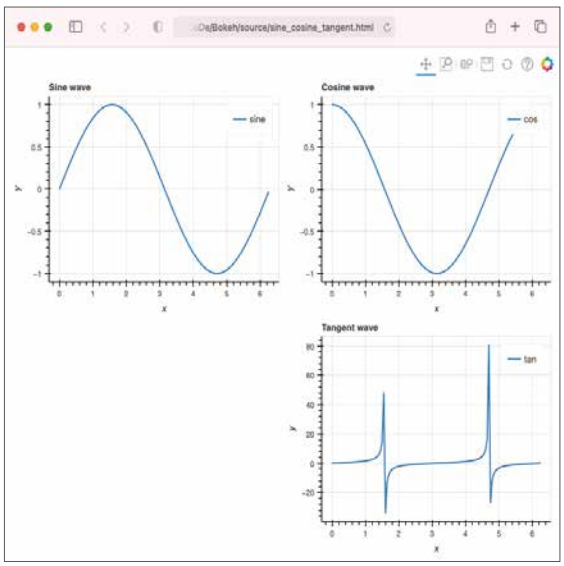


Figure 16: Controlling how each plot is shown in a gridplot

Listing 2: Saving the file as layouts.py to run it using the Bokeh server

```
from bokeh.plotting import figure, output_file, show
from bokeh.models import ColumnDataSource
from bokeh.layouts import row, column, gridplot

import pandas as pd
import numpy as np
import math

from bokeh.io import curdoc

x = np.arange(0, math.pi*2, 0.05)

df = pd.DataFrame(dict(
    x = x,
    sin = np.sin(x),
    cos = np.cos(x),
    tan = np.tan(x)
))

source = ColumnDataSource(data=df)

#-------------------------------
#           Sine Wave
#-------------------------------
p1 = figure(title = "Sine wave",
            x_axis_label = 'x',
            y_axis_label = 'y',
            plot_width=300,
            plot_height=300
           )
p1.line('x', 'sin',
        source=source,
        legend_label = "sine",
        line_width = 2)

#-------------------------------
#           Cosine Wave
#-------------------------------
p2 = figure(title = "Cosine wave",
            x_axis_label = 'x',
            y_axis_label = 'y',
            plot_width=300,
            plot_height=300
           )
p2.line('x', 'cos',
        source=source,
        legend_label = "cos",
        line_width = 2)

#-------------------------------
#           Tangent Wave
#-------------------------------
p3 = figure(title = "Tangent wave",
            x_axis_label = 'x',
            y_axis_label = 'y',
            plot_width=300,
            plot_height=300
           )
p3.line('x', 'tan',
        source=source,
        legend_label = "tan",
        line_width = 2)

#-------------------------------
#     Laying out in gridplot
#-------------------------------
plot = gridplot([p1, p2, p3],
                ncols=2,
                sizing_mode='stretch_both')

# output_file("sine_cosine_tangent.html")
# show(plot)

curdoc().add_root(plot)
curdoc().title = "Using the Bokeh Server"
```

```
plot = gridplot([[p1, p2], [None, p3]],
                sizing_mode='stretch_both')
```

Figure 16 shows the Tangent curve displayed on the right side of the second column.

Using the Bokeh Server

If you observe carefully, so far, all the charts generated by Bokeh run from the file system of your computer: file:///Volumes/SSD/Dropbox/Articles/CoDe/Bokeh/source/sine_cosine_tangent.html. Although this is useful if you're the only one viewing the charts, it's a problem if you need to publish the charts to a wider audience. This is where the **Bokeh server** comes in. The purpose of the Bokeh server is to make it easy for Python developers to create interactive Web applications that can connect front-end UI events to real, running Python code.

To convert an existing Bokeh to run using the Bokeh server, you just need to import the **curdoc()** function, and then add the plot object to the root of the current document:

```
from bokeh.io import curdoc

curdoc().add_root(plot)
curdoc().title = "Using the Bokeh Server"
```

Listing 2 shows the previous example being converted to run using the Bokeh server. To run the program using the Bokeh server, you need to save it as a Python file (**layouts.py**) and run it on the Anaconda Prompt/Terminal:

```
$ bokeh serve --show layouts.py
```

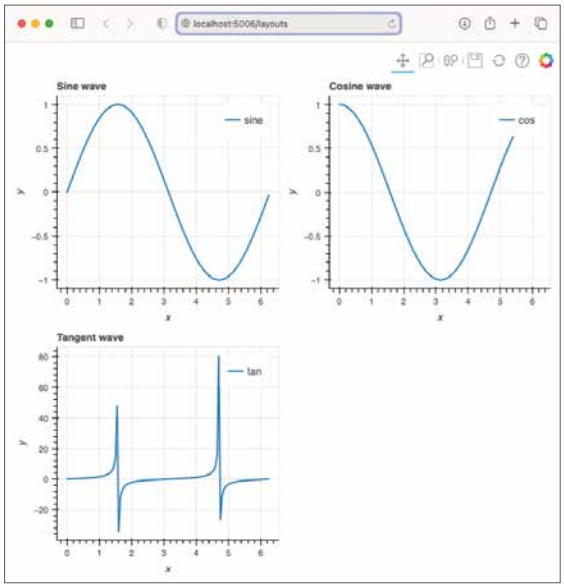


Figure 17: Hosting the dashboard on a Web server and viewing it on a Web browser

Figure 17 shows that the Bokeh server publishes the page using a Web server listening at port 5006.

Bokeh Widgets

You've learned that you can publish your charts using the Bokeh server so that your users can now view your charts easily through a Web browser. To create a front-end user interface for your users to interact with the charts, Bokeh

provides widgets—interactive controls that can be added to your Bokeh applications. Bokeh widgets include:

- Button
- CheckboxButtonGroup
- CheckboxGroup
- ColorPicker
- DataTable
- DatePicker
- DateRangeSlider
- Div
- Dropdown
- FileInput
- MultiChoice
- MultiSelect
- Paragraph
- PasswordInput
- PreText
- RadioButtonGroup
- RadioGroup
- RangeSlider
- Select
- Slider
- Spinner
- Tabs
- TextAreaInput
- TextInput
- Toggle

> Bokeh widgets are interactive controls that can be added to your Bokeh applications.

Due to space constraints, I'll only illustrate a few of the above widgets.

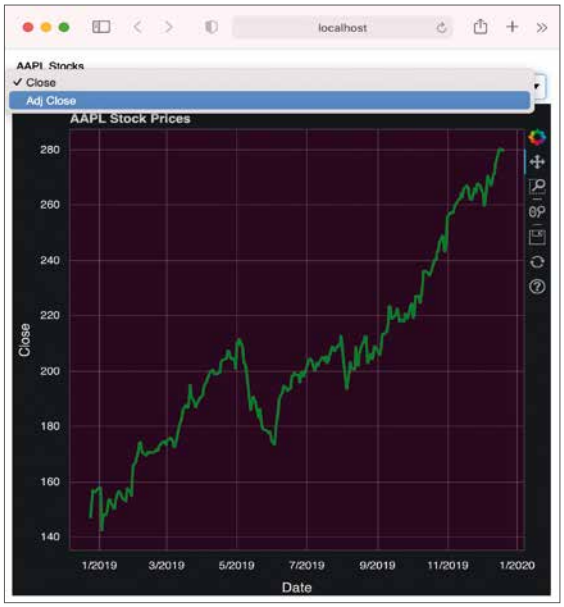


Figure 18: Using the Select Bokeh widget

Select Widget

The Select widget allows the user to make a single selection from a list of items. **Listing 3** shows a complete example of the use of the Select widget. In particular:

- The **on_change()** function allows you to set the event handler for the Select's **value** event.
- When the user selects an item from the **Select** widget, you make use of the event handler to update the **ColumnDataSource** object so that the chart can be updated dynamically.
- The event handler for the Select widget's value event takes in three parameters: **attr** (the property that triggered the event handler), **old** (the old value of the property), and **new** (the new value of the property).

Listing 3: Saving the file as SelectWidget.py to run it using the Bokeh server

```
import pandas as pd
from bokeh.plotting import figure
from bokeh.models import ColumnDataSource, Select
from bokeh.layouts import column
from bokeh.io import curdoc

# load the dataframe from a CSV file
df = pd.read_csv('AAPL.csv', parse_dates=['Date'])

# extract the columns
Date = df['Date']
Close = df['Close']
AdjClose = df['Adj Close']

# create the data source
source = ColumnDataSource(data=
{
    'x' : Date,
    'y' : Close
})

# called when the Select item changes in value
def update_plot(attr, old, new):
    if new == 'Close':
        # update the data source
        source.data['y'] = Close
        p.yaxis.axis_label = 'Close'
    else:
        source.data['y'] = AdjClose
        p.yaxis.axis_label = 'Adj Close'

# display a selection menu
menu = Select(options=[('Close','Close'),
                       ('Adj Close','Adj Close')],
              value='Close',
              title = 'AAPL Stocks')

# callback when the Select menu changes its value
menu.on_change('value', update_plot)

p = figure(x_axis_type='datetime') # display the x-axis as dates

p.line('x',
       'y',
       source=source,
       color='green',
       width=3)

p.title.text = 'AAPL Stock Prices'
p.xaxis.axis_label = 'Date'
p.yaxis.axis_label = 'Close'

curdoc().theme = 'night_sky'
curdoc().add_root(column(menu, p))
curdoc().title = "First Graph"
```

Figure 18 shows the output when the file is run in Anaconda Prompt/Terminal:

```
$ bokeh serve --show SelectWidget.py
```

Selecting an item in the Select widget updates the chart to display either the closing or adjusted closing price of AAPL.

Dropdown Widget

Another Bokeh widget that's like the Select widget is the Dropdown widget, a button that displays a drop-down list of mutually exclusive items when clicked.

The following code snippet shows how to display a Dropdown widget and wire it up with a callback function when it's clicked:

```
# display a Dropdown menu
menu = Dropdown(label="Select Company",
                menu=[
                    ('Apple','AAPL'),
                    ('Amazon','AMZN'),
                    ('Google','GOOG')])

# callback when the Dropdown menu is selected
menu.on_click(update_plot)
```

The callback function for the Dropdown function takes in a single argument:

```
# called when the Select item changes in value
def update_plot(event):
    print(event.item)   # the value of the item
                        # selected in the
                        # Dropdown widget
```

The **item** property allows you to know which item was selected in the Dropdown widget. **Listing 4** shows how you can load the content of different CSV files depending on which item has been selected in the widget.

Figure 19 shows the output when the file is run in Anaconda Prompt/Terminal:

```
$ bokeh serve --show Dropdown.py
```

Listing 4: Saving the file as Dropdown.py to run it using the Bokeh server

```
import pandas as pd
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource, Dropdown
from bokeh.layouts import column
from bokeh.io import curdoc

# load the dataframe
df = pd.read_csv('AAPL.csv', parse_dates=['Date'])

# create the data source
source = ColumnDataSource(df)

# called when the Select item changes in value
def update_plot(event):
    if event.item == 'AAPL':
        df = pd.read_csv('AAPL.csv', parse_dates=['Date'])
        p.title.text = 'Apple Stock Prices'
    elif event.item == 'AMZN':
        df = pd.read_csv('AMZN.csv', parse_dates=['Date'])
        p.title.text = 'Amazon Stock Prices'
    else:
        df = pd.read_csv('GOOG.csv', parse_dates=['Date'])
        p.title.text = 'Google Stock Prices'

    # update the date source
    source.data = df

# display a Dropdown menu
menu = Dropdown(label="Select Company",
                menu=[
                    ('Apple','AAPL'),
                    ('Amazon','AMZN'),
                    ('Google','GOOG')])

# callback when the Dropdown menu is selected
menu.on_click(update_plot)

p = figure(x_axis_type='datetime',
           plot_width=1500)
p.line("Date",
       "Close",
       source=source,
       color='green',
       width=3)

p.title.text = 'Apple Stock Prices'
p.xaxis.axis_label = 'Date'
p.yaxis.axis_label = 'Close'

curdoc().theme = 'night_sky'
curdoc().add_root(column(menu, p))
curdoc().title = "Stocks Chart"
```

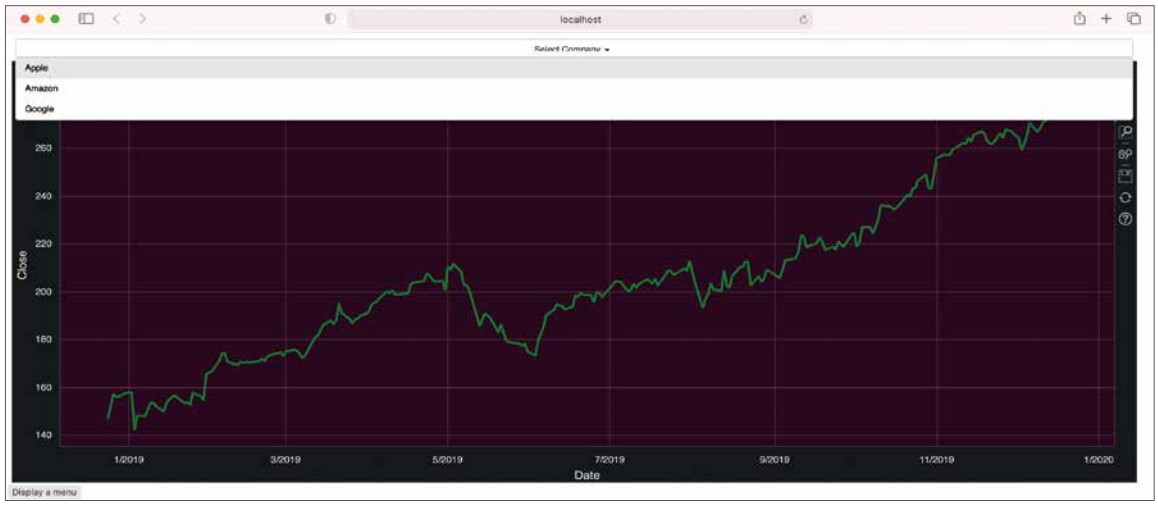


Figure 19: Using the Dropdown Bokeh widget

Listing 5: Saving the file as Tabs.py to run it using the Bokeh server

```
from bokeh.plotting import figure, output_file, show
from bokeh.models import Panel, Tabs
from bokeh.io import curdoc
from bokeh.layouts import column
from bokeh.models import Range1d

import numpy as np
import math

width = 500
height = 500

x = np.arange(0, math.pi*12, 0.05)

#-------------------------------------------------------
#                       Sine tab
#-------------------------------------------------------
p1 = figure(plot_width=width, plot_height=height)
p1.x_range = Range1d(0, math.pi*2)
p1.line(x, np.sin(x), line_width=2, line_color='yellow')
tab1 = Panel(child = p1, title = "Sine")

#-------------------------------------------------------
#                       Cosine tab
#-------------------------------------------------------
p2 = figure(plot_width=width, plot_height=height)
p2.x_range = Range1d(0, math.pi*2)
p2.line(x,np.cos(x), line_width=2, line_color='orange')
tab2 = Panel(child=p2, title = "Cos")

#-------------------------------------------------------
#                       Tangent tab
#-------------------------------------------------------
p3 = figure(plot_width=width, plot_height=height)
p3.x_range = Range1d(0, math.pi*2)
p3.y_range = Range1d(-5, 5)
p3.line(x,np.tan(x), line_width=2, line_color='green')
tab3 = Panel(child=p3, title = "Tan")

# Group all the panels into tabs
tabs = Tabs(tabs=[tab1,tab2,tab3])

curdoc().theme = 'contrast'
curdoc().add_root(column(tabs))
curdoc().title = "Trigonometry"
```

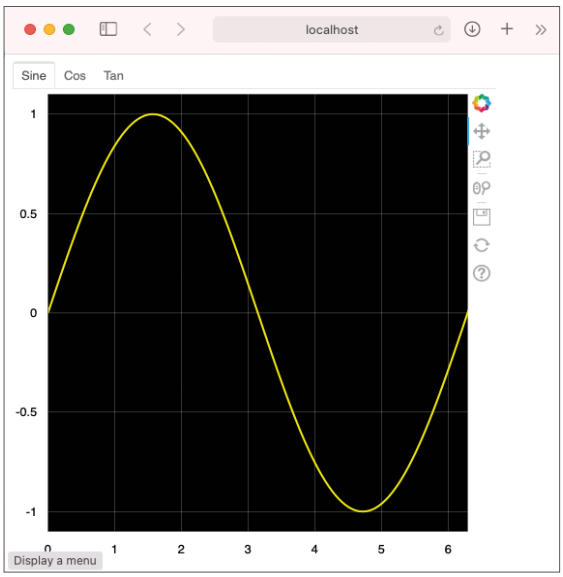


Figure 20: Using the Tabs widget to display the three trigonometric functions: sine, cosine, and tangent

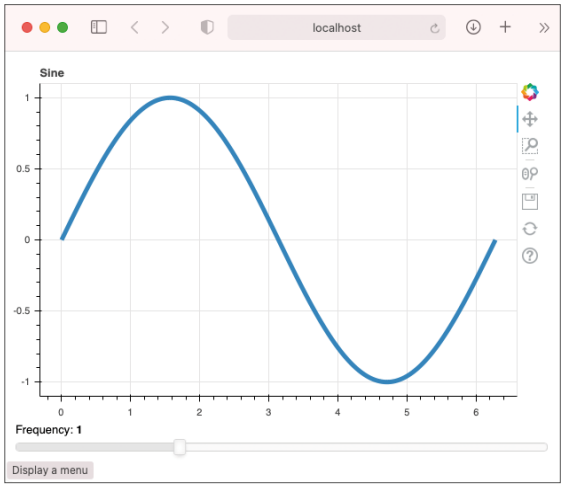


Figure 21: Plotting the sine wave with the user chosen frequencies

Tabs Widget

The Tabs widget allows you to display plots in different tabs, much like tabs in a Web browser. You first create **Panel** objects to act as the containers for your various plots. The Tabs widget then takes the various Panel objects and displays them in tabs.

Listing 5 shows an example of the Tabs widget containing three Panel objects, the first displaying the Sine wave, the second displaying the Cosine wave, and the third one displaying the Tangent wave.

Figure 20 shows the output when the file is run in Anaconda Prompt/Terminal:

```
$ bokeh serve --show Tabs.py
```

Slider Widget

The Slider widget allows users to select a range of floating values. To use this widget, simply set its start value, end value, current value, and the step size (the value to increment/decrement each time the slider moves).

Listing 6 shows an example of the Slider widget where users can select the frequencies of the sine wave to plot.

Figure 21 shows the output when the file is run in Anaconda Prompt/Terminal:

```
$ bokeh serve --show Slider.py
```

Displaying Tooltips on your Chart

Recall the earlier example where I displayed the stock prices of AAPL? It would be useful to be able to display the various information of AAPL's stock price when the user's mouse hovers over a particular data point on the chart. To do that in Bokeh, you can use the **HoverTool** class:

```
from bokeh.models.tools import HoverTool
hover = HoverTool(formatters={'@Date':'datetime'})
hover.tooltips=[
    ('Date',          '@Date{%Y-%m-%d}'),
    ('Opening',       '@Open{$0,0.00f}'),
    ('Highest',       '@High{$0,0.00f}'),
    ('Lowest',        '@Low{$0,0.00f}'),
```

```
    ('Closing',        '@Close{$0,0.00f}'),
    ('Adjusted Close', '@{Adj Close}{$0,0.00f}'),
    ('Volume',         '@Volume{0,0f}'),
]
p.add_tools(hover)
```

Figure 22 explains how the value set in the **tooltips** property helps to configure the tooltip that's displayed when the user hovers over a data point on the chart.

Listing 7 shows the Bokeh application with the code for the **HoverTool** class added.

Figure 23 shows the output when the file is run in Anaconda Prompt/Terminal.

Putting It Altogether

If you have been following along up to this point, you've learned quite a few tricks with Bokeh. So, let's now put everything that you've learned to good use and create a dashboard that users can interact with.

You'll make use of the dataframe that I have used earlier:

```
    teams  males  females
0  Team A      3        4
1  Team B      2       10
```

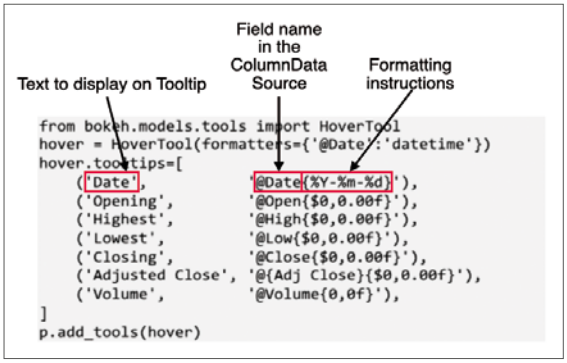


Figure 22: How to configure the tooltips property in the HoverTool class

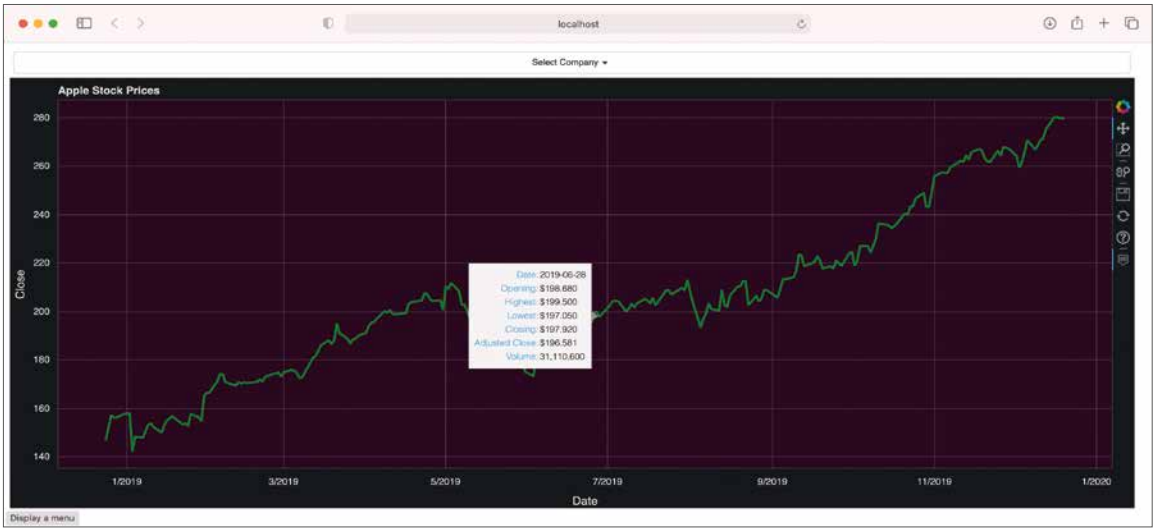


Figure 23: Displaying a tooltip when the user hovers over a chart

Listing 6: Saving the file as Slider.py to run it using the Bokeh server

```
import numpy as np
from bokeh.io import curdoc
from bokeh.layouts import column
from bokeh.models import ColumnDataSource
from bokeh.models.widgets import Slider
from bokeh.plotting import figure

N = 200
x = np.linspace(0, 2*np.pi, N)
y = np.sin(x)

source = ColumnDataSource(data = dict(
    x = x,
    y = y)
)

p = figure(plot_height = 400, plot_width = 600, title = "Sine")
p.line('x',
       'y',
       source = source,
       line_width = 5,
       line_alpha = 0.9)

freq = Slider(
    title = "Frequency",
    value = 1.0,
    start = 0.1,
    end = 3.0,
    step = 0.1,
    )

freq.width = 600

def update_data(attrname, old, new):
    y = np.sin(new * x)    # new is same as freq.value
    # update the source
    source.data['y'] = y

freq.on_change('value', update_data)

curdoc().add_root(column(p, freq, width = 600))
curdoc().title = "Sliders"
```

Listing 7: Saving the file as Tooltips.py to run it using the Bokeh server

```
import pandas as pd
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource, Dropdown
from bokeh.layouts import column
from bokeh.io import curdoc

# load the dataframe
df = pd.read_csv('AAPL.csv', parse_dates=['Date'])

# create the data source
source = ColumnDataSource(df)

# called when the Select item changes in value
def update_plot(event):
    if event.item == 'AAPL':
        df = pd.read_csv('AAPL.csv', parse_dates=['Date'])
        p.title.text = 'Apple Stock Prices'
    elif event.item == 'AMZN':
        df = pd.read_csv('AMZN.csv', parse_dates=['Date'])
        p.title.text = 'Amazon Stock Prices'
    else:
        df = pd.read_csv('GOOG.csv', parse_dates=['Date'])
        p.title.text = 'Google Stock Prices'

    # update the date source
    source.data = df

# display a Dropdown menu
menu = Dropdown(label="Select Company",
                menu=[
                    ('Apple','AAPL'),
                    ('Amazon','AMZN'),
                    ('Google','GOOG')])

# callback when the Dropdown menu is selected
menu.on_click(update_plot)

p = figure(x_axis_type='datetime',
           plot_width=1500)
p.line("Date",
       "Close",
       source=source,
       color='green',
       width=3)

p.title.text = 'Apple Stock Prices'
p.xaxis.axis_label = 'Date'
p.yaxis.axis_label = 'Close'

from bokeh.models.tools import HoverTool
hover = HoverTool(formatters={'@Date':'datetime'})
hover.tooltips=[
    ('Date',           '@Date{%Y-%m-%d}'),
    ('Opening',        '@Open{$0,0.00f}'),        # format as $
    ('Highest',        '@High{$0,0.00f}'),        # format as $
    ('Lowest',         '@Low{$0,0.00f}'),         # format as $
    ('Closing',        '@Close{$0,0.00f}'),       # format as $
    ('Adjusted Close', '@{Adj Close}{$0,0.00f}'), # format as $
    ('Volume',         '@Volume{0,0f}'),          # format as
                                                  # number
]
p.add_tools(hover)

curdoc().theme = 'night_sky'
curdoc().add_root(column(menu, p))
curdoc().title = "Stocks Chart"
```

```
2  Team C      9        6
3  Team D      7        3
4  Team E      6        9
```

Note that the numbers for each gender in each team is randomly generated, so the chart you'll see later may not correspond to the numbers you see here. Here's what you'll build for your dashboard:

- A dashboard containing three charts:
  - A stacked bar chart showing the number of males and females for each team
  - A pie chart showing the total number of members in each team
  - A pie chart showing the proportion of males and females in each selected team(s)
- When a bar or slice is selected, the other charts update accordingly.

Creating the ColumnDataSource objects

For this application, you'll create two **ColumnDataSource** objects:

- The first one for use by the stacked bar chart and the first pie chart
- The second one for use by the second pie chart

The data for the first **ColumnDataSource** object have the following values (the values in the **angle** column are calculated based on the total number of males and females in each team):

```
    teams  males  females    angles   colors
0  Team A      3        4  0.745463  #0868ac
1  Team B      2       10  1.277936  #43a2ca
2  Team C      9        6  1.597420  #7bccc4
3  Team D      7        3  1.064947  #bae4bc
4  Team E      6        9  1.597420  #f0f9e8
```

The data for the second **ColumnDataSource** object have the following values:

```
         count    angles   colors
males       27  2.875356  #98df8a
females     32  3.407829  #d62728
```

The values for this object are updated dynamically whenever the user changes his selection on the bar or pie chart.

Allowing Items to Be Selected in Your Chart

To allow users to select bars or slices in a bar or pie chart, you need to include the **tap** tool in the toolbar (see **Figure 24**). The Tools parameter in the **figure()** function allows you to select which tools to display in the toolbar:

```
p1 = figure(plot_width=400,
            plot_height=400,
            x_range=source.data['teams'],
            tools=
                'tap,pan,wheel_zoom,box_zoom,reset')
```

The Tap tool is, by default, not shown in the toolbar.

Synchronizing Charts

To update all of the affected charts when a selection is made on one, you need to handle the **indices** event of the **ColumnDataSource** object:

```
#---set the event handler for ColumnDataSource---
source.selected.on_change('indices',
    source_index_change)
```

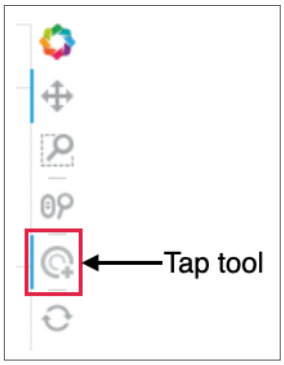


Figure 24: Locating the Tap tool in the toolbar

The event handler for the indices event has three parameters – **attrname** (name of the attribute changed), **old** (old value for the attribute), and **new** (the new value of the attribute). Here, you'll make use of **new** to know which item(s) has been selected in the bar or pie chart. Based on the bars or slice(s) selected, you can now recalculate the data for the second ColumnDataSource object:

```
#------event handler for ColumnDataSource-------
def source_index_change(attrname, old, new):
    # new is a list containing the index of all
    # selected items (e.g. bars in a bar
    # chart, etc)
    # new is same as source.selected.indices

    # if user clicks outside the bar
    if new == []:
        new = range(len(df)) # set to all
                             # selected

    print('Selected column(s):', new)
    print('Total Males:',
        df.iloc[new]['males'].sum())
    print('Total Females:',
           df.iloc[new]['females'].sum())

    # count the males and females for
    # the selected team(s)
```

Listing 8: Saving the file as Dashboard.py to run it using the Bokeh server

```
from bokeh.plotting import figure, output_file, show
from bokeh.models import ColumnDataSource, Legend
from bokeh.palettes import Category20, Spectral, GnBu
from bokeh.io import curdoc
from bokeh.layouts import row
from bokeh.transform import cumsum

import pandas as pd
import random
import math

df = pd.DataFrame(dict(
    teams =
        ['Team A','Team B','Team C','Team D','Team E'],
    males = random.sample(range(1,11), 5),
    females = random.sample(range(1,11), 5),
))

#----used for displaying a pie chart---
df['angles'] = (df['males'] +  df['females']) / (df['males'] +
                df['females']).sum() * 2*math.pi
df['colors'] = GnBu[len(df['teams'])]

source = ColumnDataSource(data=df)

#---create another source to store count of males and females---
males = df['males'].sum()
females = df['females'].sum()
df2 = pd.DataFrame(
    [males,females],
    columns=['count'],
    index =['males','females'])
df2['angles'] = df2['count'] / df2['count'].sum() * 2*math.pi
df2['colors'] = Category20[8][5:7]

source2 = ColumnDataSource(data=df2)

#-----------------------------------------------------
#                     Bar chart
#-----------------------------------------------------
p1 = figure(plot_width=400,
            plot_height=400,
            x_range=source.data['teams'],
            tools='tap,pan,wheel_zoom,box_zoom,reset')
v = p1.vbar_stack(['males', 'females'],
            source = source,
            x = 'teams',
            width=0.8,
            color=Category20[8][5:7])
legend = Legend(items=[
    ("males",   [v[0]]),
    ("females", [v[1]]),
], location=(0, 0))
p1.add_layout(legend)
p1.title.text = "Gender distribution in each team"

#-----------------------------------------------------
#                     Pie chart 1
#-----------------------------------------------------
p2 = figure(plot_width=400,
            plot_height=400,
            tools='tap,pan,wheel_zoom,box_zoom,reset')
p2.wedge(x=0,
         y=1,
         radius=0.7,
         start_angle=cumsum('angles', include_zero=True),
         end_angle=cumsum('angles'),
         line_color="white",
         fill_color='colors',
         legend_field='teams',
         source=source)
p2.axis.visible = False
p2.title.text = "Members in each team"

#-----------------------------------------------------
#                     Pie chart 2
#-----------------------------------------------------
p3 = figure(plot_width=400,
            plot_height=400)
p3.wedge(x=0,
         y=1,
         radius=0.7,
         start_angle=cumsum('angles', include_zero=True),
         end_angle=cumsum('angles'),
         line_color="white",
         fill_color='colors',
         legend_field='index',
         source=source2)
p3.axis.visible = False
p3.title.text = "Gender distribution for all teams"

#------event handler for ColumnDataSource-------
def source_index_change(attrname, old, new):
    # new is a list containing the index of all
    # selected items (e.g. bars in a bar chart, etc)
    # new is same as source.selected.indices

    # if user clicks outside the bar
    if new == []:
        new = range(len(df))
    print('Selected column(s):', new)
    print('Total Males:', df.iloc[new]['males'].sum())
    print('Total Females:',df.iloc[new]['females'].sum())

    # count the males and females for the selected team(s)
    df2.loc['males','count'] =  df.iloc[new]['males'].sum()
    df2.loc['females','count'] =  df.iloc[new]['females'].sum()
    source2.data['angles'] =
        df2['count'] / df2['count'].sum() * 2*math.pi
    p3.title.text = \
        f"Gender distribution for {[df.iloc[t,0] for t in new]}"

#------set the event handler for ColumnDataSource-------
source.selected.on_change('indices',source_index_change)

curdoc().add_root(row(p1,p2,p3, width = 1200))
curdoc().title = "Interactive Graph"
```

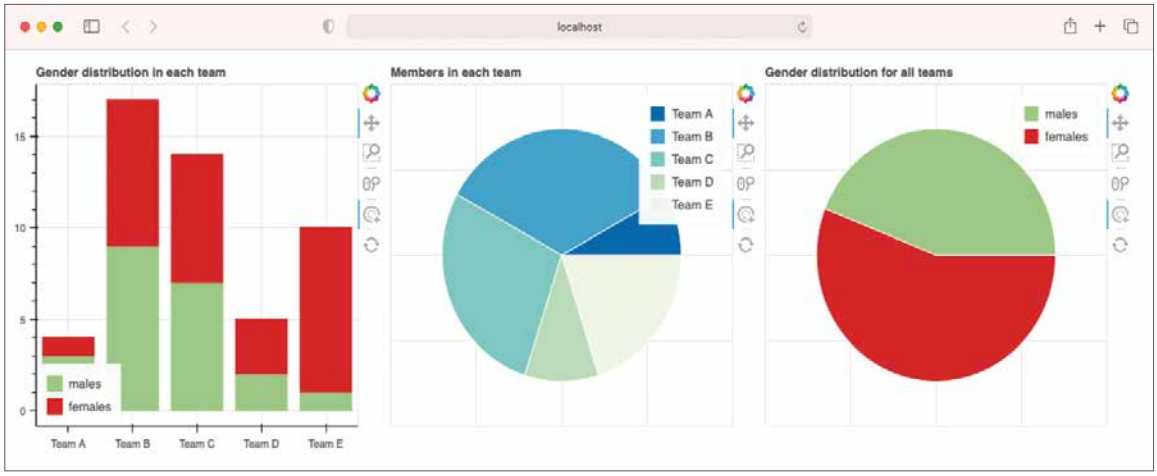


Figure 25: The dashboard containing three plots

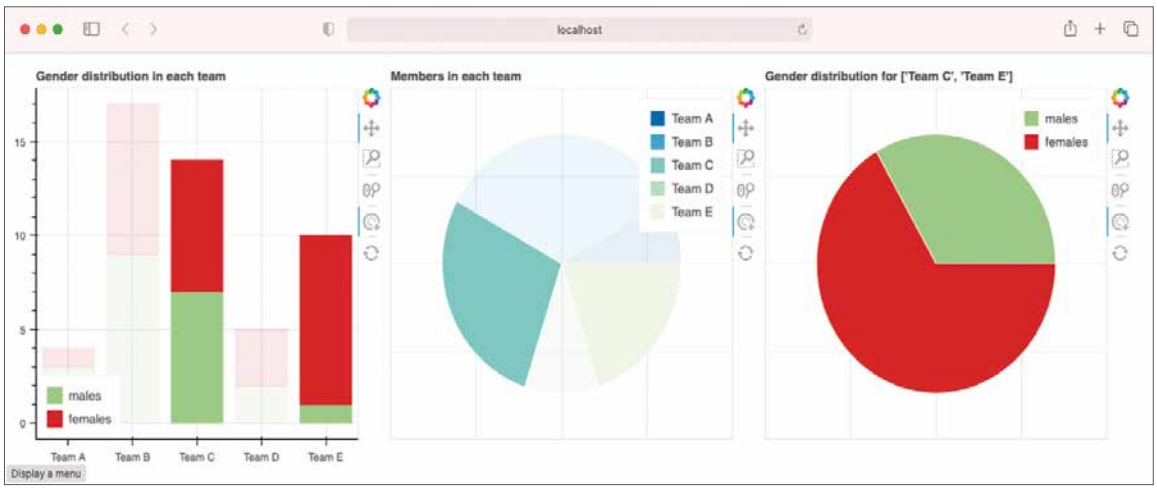


Figure 26: Selecting any bar or slice will update the third plot

```
df2.loc['males','count'] = \
    df.iloc[new]['males'].sum()
df2.loc['females','count'] = \
    df.iloc[new]['females'].sum()
source2.data['angles'] = \
    df2['count'] / df2['count'].sum() * \
    2*math.pi
p3.title.text = \
    f"Gender distribution for {
        [df.iloc[t,0] for t in new]}"
```

The full code listing for this application is shown in **Listing 8**.

Figure 25 shows the dashboard showing the stacked bar chart and the two pie charts when the file is run in Anaconda Prompt/Terminal:

```
$ bokeh serve --show Dashboard.py
```

You can click on any of the bars in the stacked bar chart as well as any of the slices in the pie chart (press the Shift key to select more than one item) and the second pie chart will now display the distribution of the gender in the select bars or slices (see **Figure 26**).

Summary

In this article, I've taken you on a whirlwind tour of Bokeh. I've discussed how to plot different plots using the various glyphs methods, as well as how to group them using the various layout classes. You also learned how to deploy your Bokeh plots using the Bokeh server, and how to use the Bokeh widgets to facilitate user interactions. Finally, I end this article by creating an interactive dashboard where interacting with one plot automatically updates the others. I hope you have fun with your newfound knowledge!

Wei-Meng Lee

Beginner's Guide to Deploying PHP Laravel on the Google Cloud Platform

Today, I introduce you to a new series of articles, starting from this issue, on deploying a PHP Laravel application on Google Cloud Platform (GCP). GCP is rich in services and documentation. They've done a great job documenting their services in a variety of programming languages including, and not limited to, Go, PHP, Java, Node.js, and others.

Bilal Haidar
bhaidar@gmail.com
http://blog.bilalhaidar.com
@bhaidar

Bilal Haidar is an accomplished author, Microsoft MVP of 10 years, ASP.NET Insider, and has been writing for CODE Magazine since 2007.

With 15 years of extensive experience in Web development, Bilal is an expert in providing enterprise Web solutions.

He works at Consolidated Contractors Company in Athens, Greece as a full-stack senior developer.

Bilal offers technical consultancy for a variety of technologies including Nest JS, Angular, Vue JS, JavaScript and TypeScript.

If the GCP documentation is so good, why am I writing this series, you may ask. The GCP team gives us access to the How-To's (*https://cloud.google.com/docs*), like using the GCP services in detail for different programming languages. They cover all the basics well, including scenarios on using the services in different situations. However, there's almost no focus on custom or specific frameworks like PHP Laravel. I found little documentation or articles online on deploying a PHP Laravel application on GCP. Hosting a PHP Laravel on GCP was nightmarish for me. Not only hosting but automating and optimizing the deployment process in itself was near impossible! That's why I'm writing this—so you don't have the same experience.

In this series of articles, I'll be dissecting every single detail that you need to deploy your PHP Laravel application on GCP. I'll assume that you have no prior experience with GCP and will guide you accordingly on all aspects of automating your deployment of the PHP Laravel application on GCP.

This first article covers the following topics:

- Creating a GCP account
- Creating a GCP first project
- Creating a PHP Laravel 8 application using Docker containers
- Hosting the application's source code on GitHub
- Connecting the GitHub repository to GCP
- Automating the deployment process from GitHub to GCP, using GCP Cloud Build service

Create a GCP Account

A GCP account connects directly to a Gmail account. To start, get yourself a Gmail account and then visit the GCP home page at *https://cloud.google.com/*. GCP offers its clients and potential clients a free trial and tier account. You can read more about the details here: *https://cloud.google.com/free*.

Sign in to GCP by visiting the URL (*https://cloud.google.com/*) and clicking the **Sign-in** button. You need to provide a valid Gmail account to sign in.

> You need to have a Gmail account to gain access to Google Cloud Platform.

Right after signing in, click the **Console** button located at the top-right side of the screen. **Figure 1** shows the GCP home page when you first sign in.

You start by:

1. Selecting the **Country** you reside in.
2. Accepting the **Terms of Service** by checking the checkbox.
3. Opting in or out of receiving **Email Updates** on GCP services and the newsletter.
4. Clicking on the **AGREE AND CONTINUE** button.

You can skip the next step or select the purpose you are using GCP. For now, I skipped it.

Let's activate this free account by clicking the Activate button located at the top-right side of the screen. The process for registering a free tier account takes you on a series of steps to get the account up and running.

What you need to do is:

1. Fill-in and verify your GCP account information
2. Fill-in and verify GCP payment information
3. Fill out a post-registration survey.

When you complete the registration, GCP automatically creates your first project and names it **My First Project**. You can continue using this project, disregard it, or create another project, as you see fit. In addition, GCP sets up your billing account automatically.

Create a PHP Laravel 8 Application

Laravel is a mature PHP framework for building Web applications (*https://laravel.com/*). It offers a foundational starting point, with extensive documentation and integrations with third-party services to give you most of the features you need to kick off your next Web application.

From the documentation: "Laravel strives to provide an amazing developer experience while providing powerful features such as thorough dependency injection, an expressive database abstraction layer, queues and scheduled jobs, unit and integration testing, and more."

Now, I'll guide you through creating your first Laravel project locally on your computer. The intention is neither to explain the rich features of Laravel nor to teach you how to develop and use them. If you're interested in that, I suggest you follow the documentation, as they offer everything a newbie needs to start using the framework.

Depending on your local environment, whether you are using a macOS, Windows, or Linux, you can choose the right way to install a fresh copy of the Laravel project. Check the documentation (*https://laravel.com/doc/8.x/installation*) for more information.

The Laravel team recommends using a service named Laravel Sail (*https://laravel.com/docs/8.x/sail*), which is a built-in solution for running your Laravel project using Docker (*https://www.docker.com/*). That's the best option when you have a team of developers or are planning on expanding the team to include more developers.

> Google App Engine Flexible supports PHP with a maximum version of 7.3.

Before you install Laravel locally, make sure you have the following two requirements:

- Docker, installed and running on your local computer
- Composer (https://getcomposer.org/) installed

Step 1: Install Laravel Using Composer

Start by running this command:

```
composer \
  create-project \
  laravel/laravel="8.5.23" \
  gcp-app2
```

I've named my new project as **gcp-app** and explicitly installed Laravel v8.5.23. As of the time of writing, that's the maximum Laravel version you can deploy on the GAE Flexible environment. This environment allows a maximum version of PHP 7.3. Higher versions of Laravel make use of third-party libraries that require PHP 8.0.

Step 2: Download and Install Laravel Sail

Right after installing the Laravel project locally, you need to download and install the Laravel Sail package. This package adds all the necessary boilerplate tools to allow running your Laravel application in Docker containers. Run the following command to download the package:

```
composer require laravel/sail --dev
```

Figure 2 shows running this command in action.

Composer downloads and configures the package locally. The next step is to install the Sail package. **Figure 3** shows the installation process.

Run the following command:

```
php artisan sail:install
```

The command installs the package and prompts you to select one or more services to be included within the Sail installation and later run them as Docker containers.

Here, I just selected **mysql**. You can install multiple services by separating them with commas (e.g., 0, 3, 6).

Before running the application, make sure to open the **/composer.json** file and fix the Laravel Framework package to version 8.40 as follows:

```
"laravel/framework": "8.40"
```

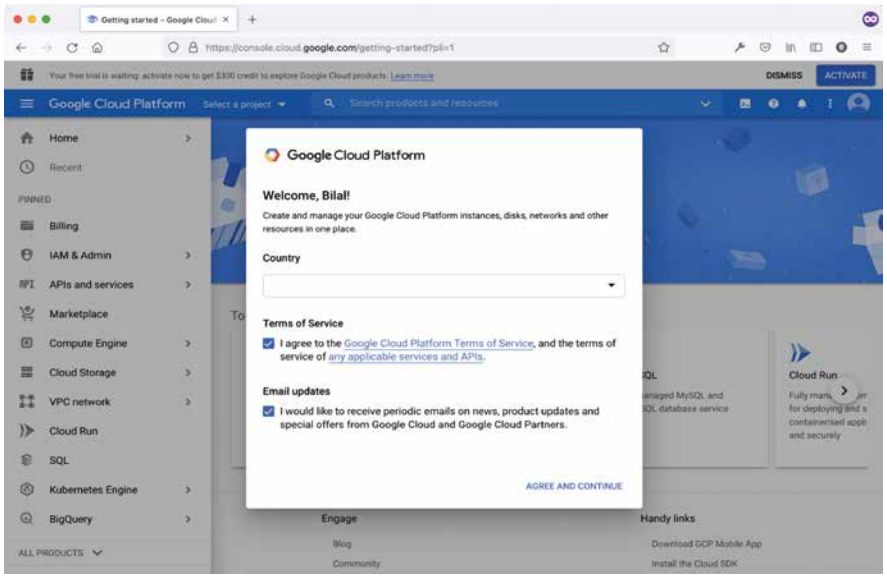


Figure 1: Signing-in to the GCP for the first time

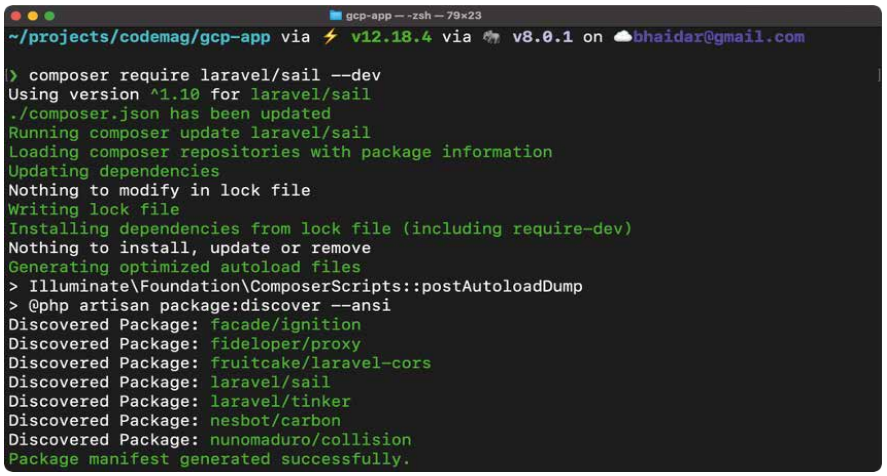


Figure 2: Download the Laravel Sail package

> Make sure the laravel/framework package you select doesn't have any third-party package dependencies using a higher version of PHP 7.3.

Step 3: Run the Application Locally

The final step is to run the application and view it in the browser. You need to use the Sail package to run the application as follows:

```
./vendor/bin/sail up --d
```

This command starts up all Docker containers (MySQL, Nginx, PHP, etc.) and makes the application ready to receive requests.

It takes a few minutes, depending on your Internet speed, to download all the Docker images, start them, and then serve your application. The sail command runs all the Docker containers in the background silently using the **--d** command option.

Let's run the default migrations that ships with a brand-new Laravel project:

```
sail artisan migrate:fresh
```

This command runs all the migrations present locally and creates the database tables accordingly.

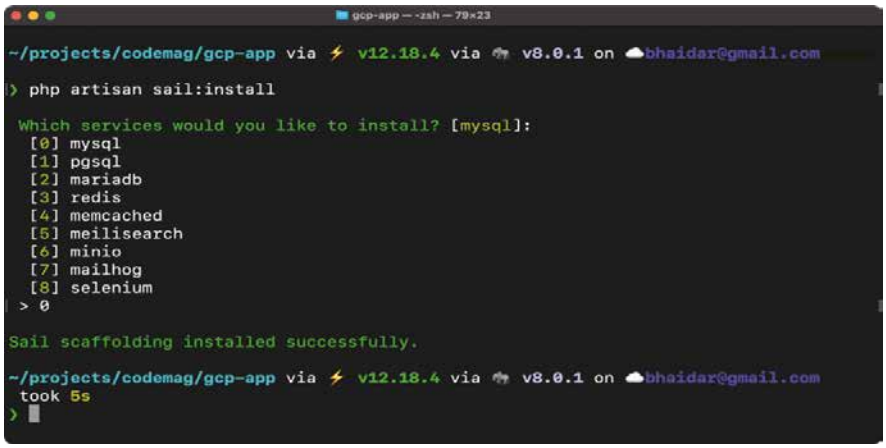


Figure 3: Laravel Sail installation

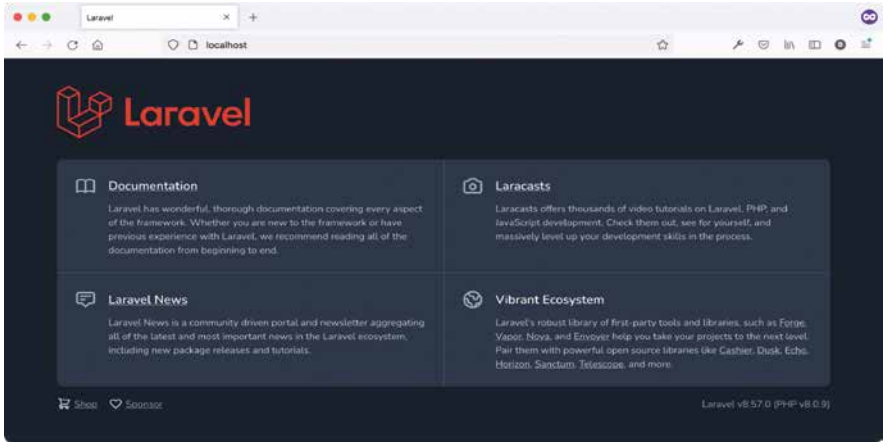


Figure 4: Laravel project running locally

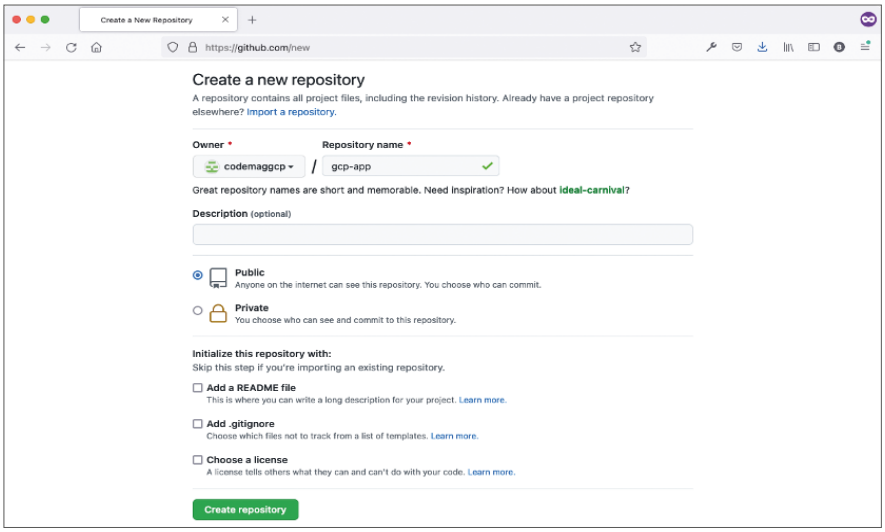


Figure 5: Create a new GitHub repository to host your Laravel project.

Now that you have your Laravel project ready, let's run it in the browser by visiting the URL *http://localhost/*.

Figure 4 shows the Laravel project up and running locally inside the browser.

Hosting the Application on GitHub

Let's start by creating a new GitHub repository named **gcp-app**. **Figure 5** shows the creation screen for this new repository.

Right after creating the repository, run the following commands to push your local Laravel to GitHub:

```
echo "# gcp-app" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/codemaggcp/gcp-app.git
git push -u origin main
```

Switch back to GitHub and confirm that you have successfully pushed all the files and folders. **Figure 6** shows the GitHub repository populated with your Laravel project.

That's it for now! In the coming section, you'll be connecting this GitHub repository to the GCP project you created, to automate the deployment of the Laravel project onto the GCP environment.

Create a GCP App Engine Application

The GCP App Engine (GAE) (*https://cloud.google.com/appengine/*) is a fully managed, serverless platform for developing and hosting a wide range of Web applications across multiple technologies. It has tight integration with the rest of the services offered by GCP, like Cloud Storage, and others. It takes the burden of managing Virtual Machines off your shoulders and handles all the nifty small details on your behalf.

GCP offers two flavors of GAE: Standard and Flexible environment. In brief, the standard environment (*https://cloud.google.com/appengine/docs/standard*) has limitations that make it more suitable to host stateless Web applications that respond to HTTP requests quickly. The flexible environment (*https://cloud.google.com/appengine/docs/flexible*) runs your applications in Docker containers on Google Compute Engine virtual machines (VMs) (*https://cloud.google.com/compute/docs/instances*). It offers more flexibility and freedom to host any kind of Web application on Docker containers.

Let's start by navigating to the App Engine section on Google Cloud Console by selecting it from the left-side menu. **Figure 7** shows where to locate the GAE on the main GCP menu.

> You can always pin menu items inside the GCP left-side menu for ease of access

Click the **CREATE APPLICATION** button. Follow the steps to create your first GAE application.

Listing 1: app.yaml file

```
runtime: php
env: flex

runtime_config:
  document_root: public

# Ensure we skip ".env", which is only for local development
skip_files:
  - .env

automatic_scaling:
  min_num_instances: 1
  max_num_instances: 1

resources:
  cpu: 2
  memory_gb: 4
  disk_size_gb: 100

env_variables:
  # Put production environment variables here.
  APP_DEBUG: true
  APP_ENV: production
  APP_KEY: YOUR_APP_KEY
  APP_NAME: "GCP App - CODE"

  LOG_LEVEL: debug
```

GCP, like other cloud platforms, defines a location where the VMs hosting and handling your Web applications are. In GCP terms, this is known as Region and Zone. **Figure 8** shows the region selection page.

You're free to choose the region you like. In my case, I selected the **europe-central2** region. Then click the **NEXT** button. You're done! On the next page, you can select the **Language** and **Environment** field. In case you want to deploy the Laravel project manually, go ahead and click the **DOWNLOAD the CLOUD SDK** on this page.

Follow the steps on-screen to finish the deployment. I won't be deploying the project manually; instead, I'll be using another service offered by GCP, named Cloud Build, that automates the deployment process every time you push new code to your GitHub branch.

The GAE follows an advanced and quite complex structure to host your Web applications. It defines the concept of a Service. You may have one or more services serving your Web application. Services can easily communicate with each other.

A Service can contain one or more Versions. A Web application can have multiple versions deployed inside the same Service. You may read more about GAE Services and Versions here (*https://cloud.google.com/appengine/docs/flexible/php/configuration-files*).

Let's add the **app.yaml** file into the root folder of the Laravel project. This file instructs the GAE on how to deploy the application, what language it uses, what resources it needs, and many other options we'll discover together along the way.

Create a new app.yaml file and paste the text shown in **Listing 1**.

Let's explore the configuration settings contained in **Listing 1**. The **runtime** variable specifies the language that the project uses. In this case, Laravel makes use of the PHP programming language. The **env** variable specifies whether this project uses GAE Standard or Flexible environment.

The **runtime_config** variable allows you to configure Nginx, PHP runtime, Supervisord, and other runtime environments. In this case, I'm setting the value of **document_root** to public. The document_root represents the DOCUMENT_ROOT for Nginx and PHP.

The **skip_files** variable defines which file or file types are to be ignored when deploying this project to GAE virtual machines.

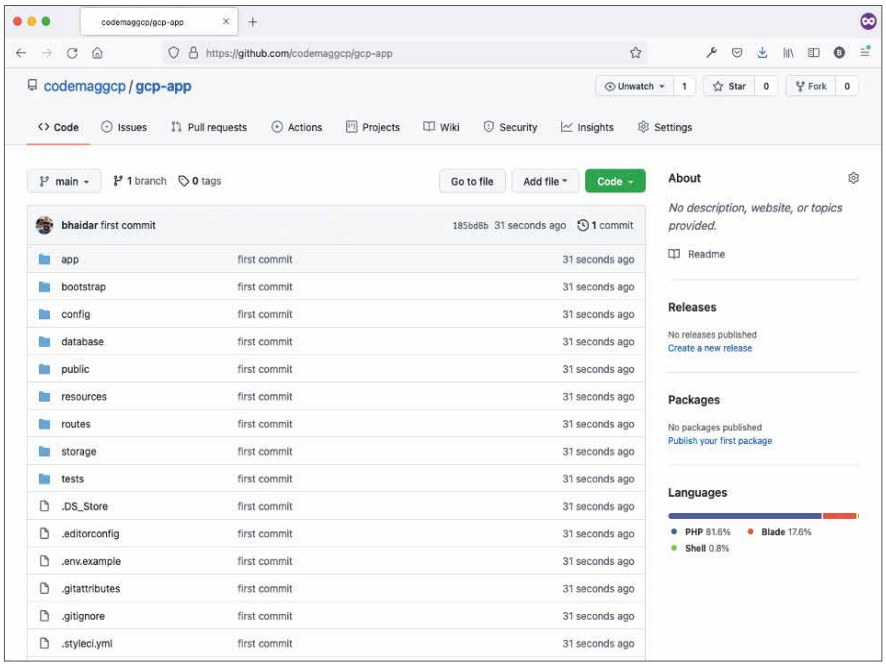


Figure 6: Laravel project hosted on GitHub repository

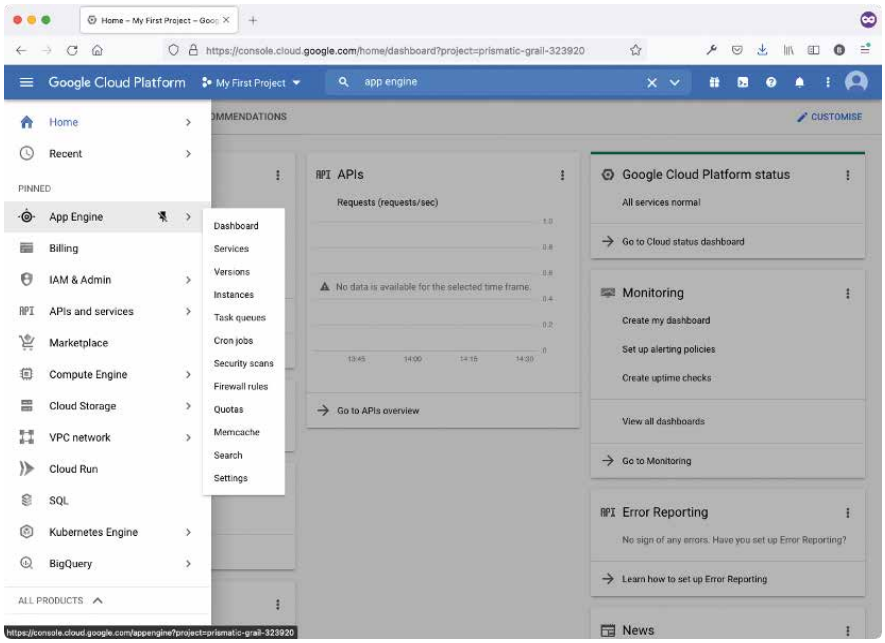


Figure 7: GCP App Engine

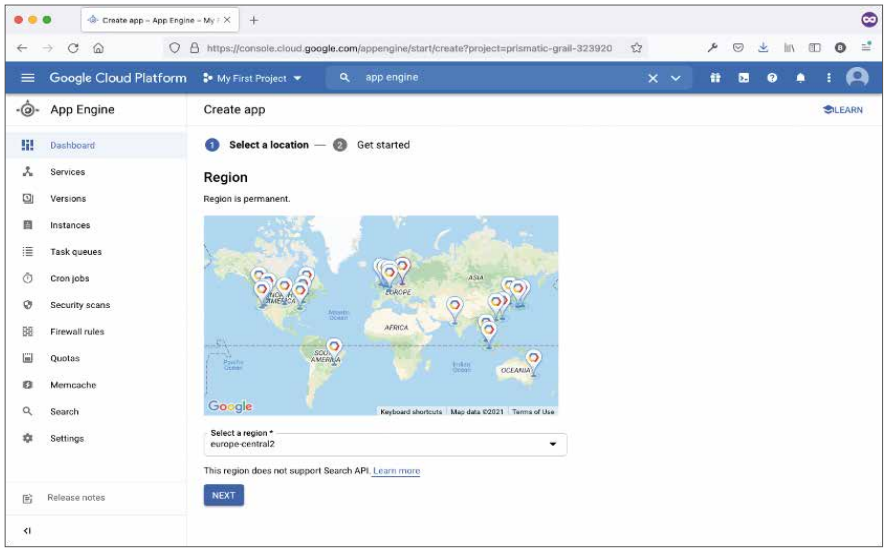


Figure 8: GCP App Engine: Region selection

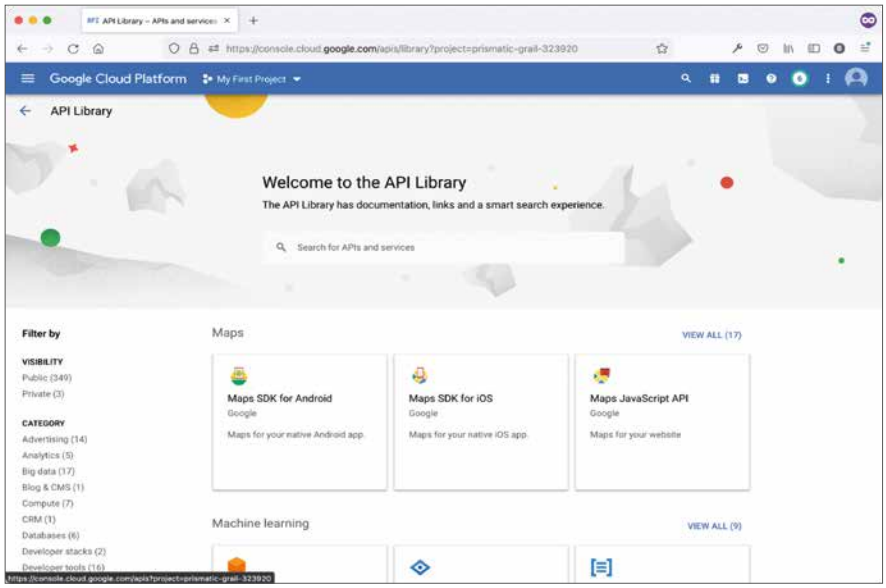


Figure 9: GCP API Library page

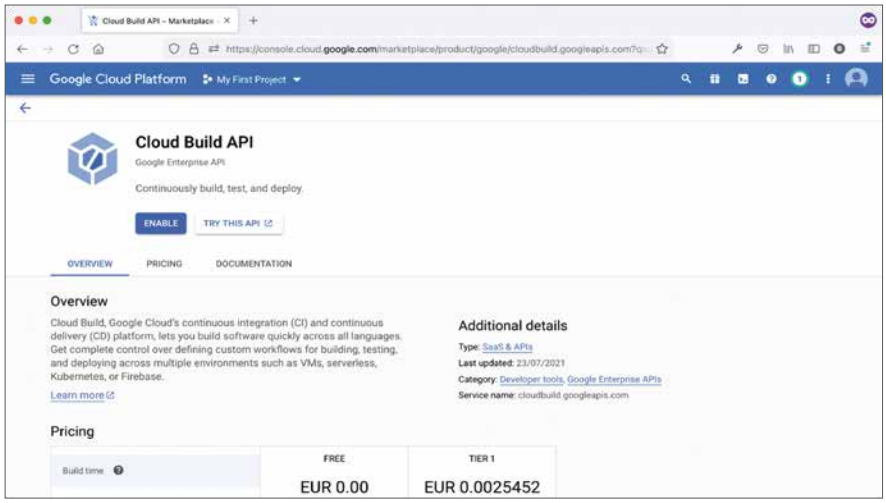


Figure 10: Cloud Build API page

You can use the **automatic_scaling** variable to specify the autoscaling requirements. In this case, you're configuring the GAE to use a single virtual machine instance at any time.

The **resources** variable allows you to specify the hardware capabilities of the instance(s) that GAE uses to host your Web application. In this case, you're specifying the virtual machine instance to have two CPUs, 4 GB of RAM, and a disk of size 100 GB.

The **env_variables** represent the **.env** file for the project when it runs in the GAE cloud.

The app.yaml is much more involved and allows you to specify more settings. You can review them all here: *https://cloud.google.com/appengine/docs/flexible/php/reference/app-yaml*.

Now let's replace the **YOUR_APP_KEY** with an actual Laravel application key. Run the following command:

```
php artisan key:generate --show
```

Grab the key that's displayed on the screen and paste it instead of YOUR_APP_KEY inside the app.yaml file. Then, make sure you append the following script under the **scripts** section of the composer.json file:

```
"scripts": {
        ...
        "post-install-cmd": [
            "chmod -R 755 bootstrap\/cache",
            "php artisan cache:clear"
        ]
},
```

This script runs after a successful deployment to set the proper permissions on the **bootstrap/cache** folder and clear the cache of the Laravel project.

For now, commit the changes you've made to the source code (without pushing them to GitHub).

Now that you've successfully configured the GAE, you can proceed to the next step and make use of the GCP Cloud Build to deploy your project.

Create a GCP Cloud Build Workflow

Cloud Build is a service that executes your builds on GCP's infrastructure. When it runs, it searches for a build config file inside your application folder. This file contains build steps to instruct Cloud Build on how to build the application and what to do with the artifact generated as a result of the build process.

Every build step consists of a Docker image to use and a command to run inside this Docker image. This means that every step of the Cloud Build workflow executes inside its own Docker container. The Docker container can run fetch dependencies, unit tests, integration tests, static analysis tools, and it can generate artifacts for deployment.

Cloud Build is part of Google's CI/CD Platform (*https://cloud.google.com/docs/ci-cd*) that allows you to build, deploy, and automate the entire process.

With Cloud Build, you can:

- Build your app (run tests, etc.).
- Deploy your app (on GAE, Cloud Run, and other services).
- Automate the entire process by using Cloud Build Triggers.

By the end of this next section, you'll build a complete Cloud Build Workflow to automate the build and deployments of the Laravel project at hand.

Step 1: Enable GCP APIs and Libraries

First things first: You need to enable a few APIs and Services on GCP. Locate the following APIs, and enable them in this order:

- App Engine Admin API
- Cloud Build API
- Compute Engine API
- Cloud Deployment Manager V2 API

To enable any API or Library on the GCP, on the left-side menu, locate and click the **APIs and Services** menu item. Then, once you're on the APIs and Services page, click the **Library** menu item. **Figure 9** shows the API Library page.

Search for a library, click on it, and enable it. **Figure 10** shows the Cloud Build API page.

Click the **ENABLE** button to enable this API.

Step 2: Give Permissions to the Cloud Build Service Account

GCP has the concept of a **service account**. A service account is a special kind of account used by an application or a virtual machine instance, not a person.

For example, applications use service accounts to make authorized API calls. In another example, a Compute Engine VM can run as a service account, and it can be given permissions to access the resources it needs. This way, the service account is the identity of the service, and the service account's permissions controls which resources the service can access.

You can read more about GCP Service Accounts here: *https://cloud.google.com/iam/docs/service-accounts*.

When you create a GAE application, GCP creates a default GAE service account behind the scenes. When you enable the Cloud Build API, once again, it creates a Cloud Build service account.

To allow the Cloud Build API to build your application and deploy it to the GAE, you need to give its service account all the necessary permissions to do its job properly.

Locate and access the Identity and Access Management & Administration (IAM & Admin) section on the left-side menu. This section allows you to manage all identity and access management to your project(s) on GCP.

Figure 11 shows all the required roles and permissions that I've assigned to the Cloud Build service account. Look for the member that ends with **@cloudbuild.gserviceaccount.com** to locate the Cloud Build service account.

You can click the **pencil** icon to edit the list of roles assigned to the Cloud Build service account or any other member.

Those are all the permissions needed so far to enable the Cloud Build service to build your application and deploy it to GAE.

Step 3: Add a cloudbuild.yaml Build Config File

To deploy an app on GAE, you can either do it manually by using the Cloud SDK commands (*https://cloud.google.com/sdk*) or automate the entire process using Cloud Build.

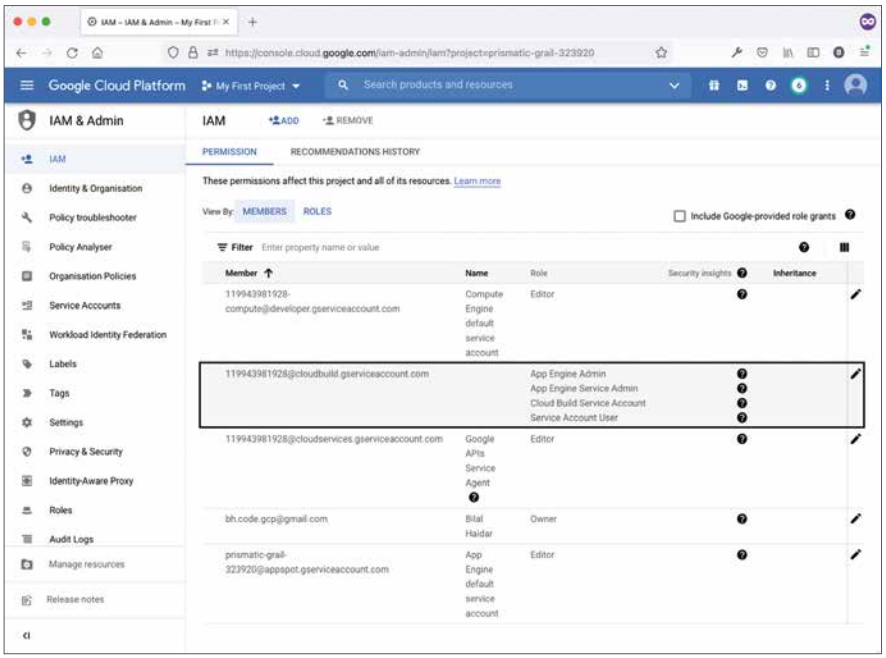


Figure 11: Cloud Build service account roles and permissions

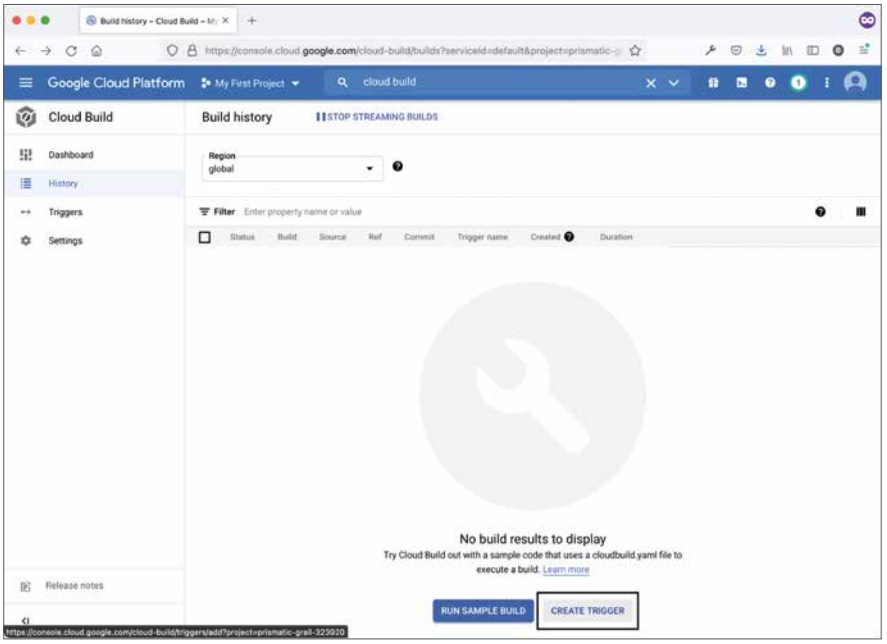


Figure 12: Cloud Build Triggers

Listing 2: ci/cloudbuild.yaml file

```
steps:
  - name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
    entrypoint: 'bash'
    args:
      - '-c'
      - |
        gcloud config set app/cloud_build_timeout 3600 \
        && gcloud app deploy -q --promote -v=$BUILD_ID \
        --project=$PROJECT_ID

timeout: '3600s'
```

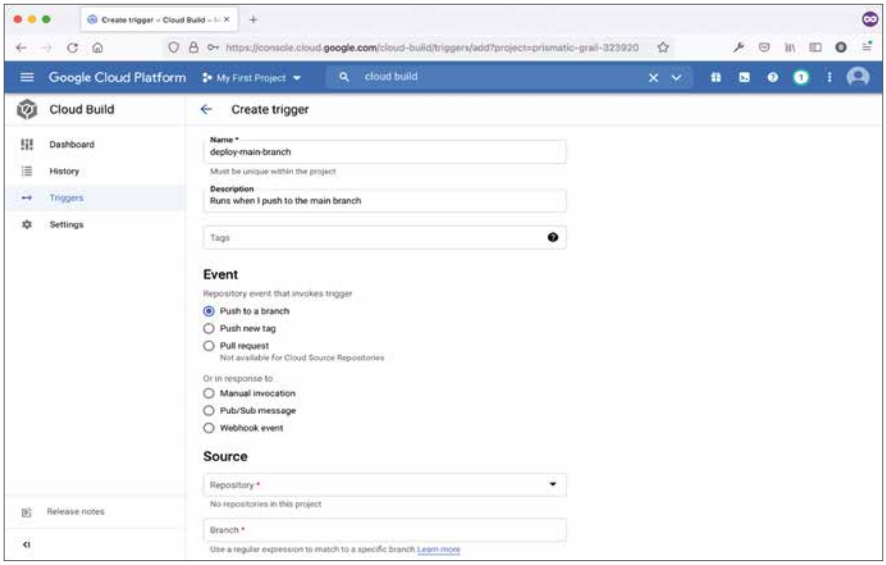


Figure 13: Create the Trigger page.

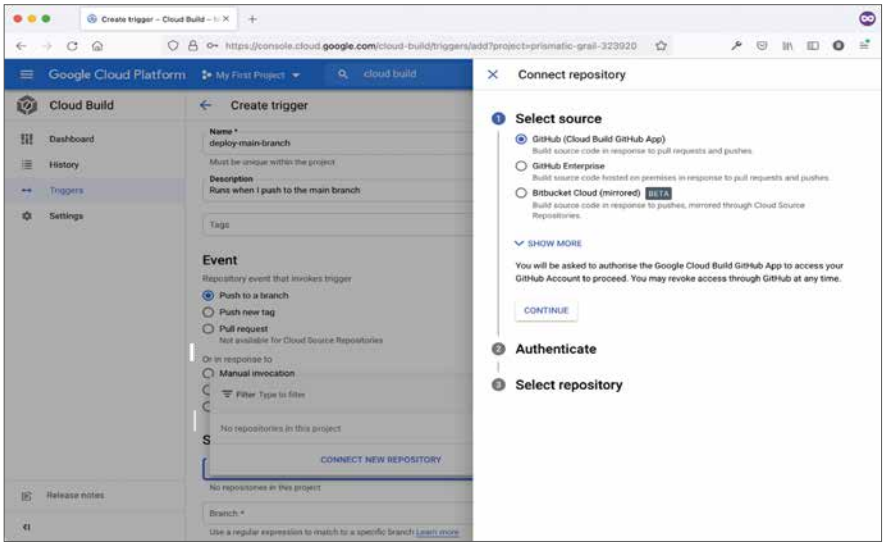


Figure 14: The connect repository window

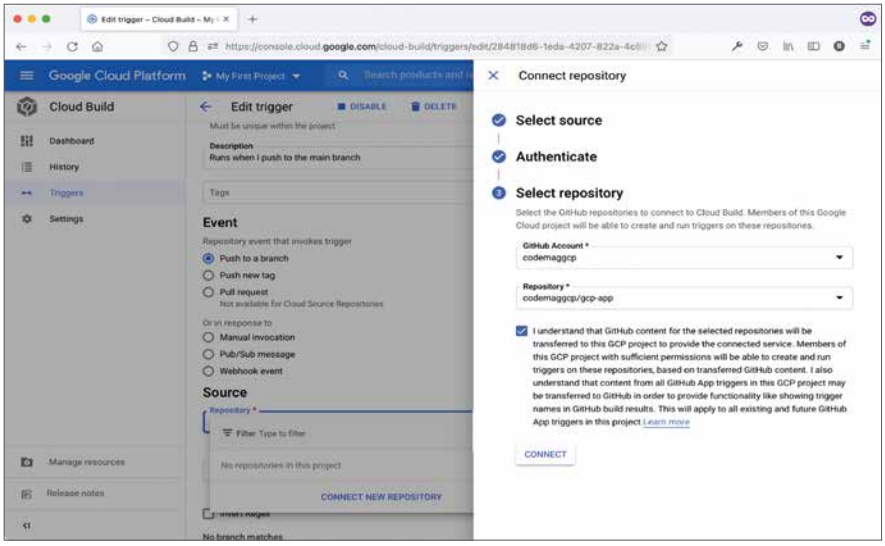


Figure 15: Trigger repository

As mentioned, Cloud Build requires having a build config file. Let's start by creating the **ci/cloudbuild.yaml** file. Paste the content in **Listing 2** inside the cloudbuild.yaml file.

The cloudbuild.yaml file has a single build step. It runs two Google Cloud SDK commands inside a Docker container that's based on the Cloud SDK Docker image named **gcr.io/google.com/cloudsdktool/cloud-sdk**.

The two Cloud SDK commands are straightforward:

- **gcloud config set app/cloud_build_timeout 3600:** The timeout, in seconds, to wait for Docker builds to complete during deployments.
- **gcloud app deploy -q --promote -v=$BUILD_ID --project=$PROJECT_ID:** This command deploys the application to the GAE using the following options:
  - **-q:** Disables all interactive prompts when running gcloud commands. If the command required an input, defaults will be used or an error will be raised
  - **--promote:** Allows the deployed version to receive all traffic.
  - **-v:** The version of the app that will be created or replaced by this deployment. In this case, I'm using a built-in variable, the Cloud Build ID, to serve as a version name.
  - **--project:** The GCP project ID to use for this invocation.

Now that the cloudbuild.yaml file is ready, let's move on to automate running the Cloud Build and deploying the Laravel project.

Step 4: Create a Cloud Build Trigger

Navigate to the Cloud Build section on the left-side menu. Locate and click **Triggers**. **Figure 12** shows the Triggers page.

Click the **CREATE TRIGGER** button located in the bottom center of the page. **Figure 13** shows the first part of the trigger creation page.

Give this new trigger a name and description (optional).

The **Event** defines the event that, when it happens, causes this trigger to run. There are a variety of options to pick from. In this case, I'll be using the **Push to a branch** event. This means that every time I push a GitHub branch, this trigger executes.

The next step is to select the Git repository and branch. Click inside the **Repository** field and then click the **CONNECT NEW REPOSITORY** button. **Figure 14** shows the pop-up window to connect to the repository.

You can connect to a variety of source controls. I'll keep the default option and use GitHub. Click the **CONTINUE** button.

GCP wants to take your permissions and authenticate you to GitHub so that it can connect to the GitHub repository on your behalf. Authenticate GitHub and give the permissions needed.

Next, select the GitHub repository. If you haven't connected your GitHub account before with GCP, you need to locate and click the **INSTALL GOOGLE CLOUD BUILD** button on one

or all of your GitHub repositories. Once done, you can select your GitHub account and repository to use in this Cloud Build trigger. **Figure 15** shows the repository that I've selected.

Now, click the **CONNECT** button. Go back to the trigger creation page and input the following expression for the branch name: **^main$**. By completing these steps, you're instructing GCP to execute this trigger every time you push code into the main branch.

Finally, you want to specify the file path of the cloudbuild.yaml file. **Figure 16** shows where to specify the file path.

In summary, you've created a Cloud Build trigger that runs when you push code to the application GitHub repository main branch that eventually runs and executes the cloudbuild.yaml file. Return to the Cloud Build Triggers page and make sure that you've created the trigger successfully. **Figure 17** shows the new trigger that you just created.

Step 5: Run Cloud Build Trigger

The Cloud Build Trigger is the glue that connects three major activities in the life cycle of a project:

- Developing and pushing code to source control (GitHub in this case)
- Building the application on GCP infrastructure
- Deploying the application on GAE

Thanks to the Cloud Build Trigger, when you push your code to GitHub, the trigger runs and executes the Cloud Build Workflow. In turn, it builds the Laravel project and deploys it on GAE.

To deploy your application, you've got two options:

- Click RUN on the trigger itself.
- Push code to GitHub on the main branch of the application's repository.

Whichever option you choose, **Figure 18** shows a successful execution of the trigger.

Navigate back to the App Engine section under Services and click the **default** service. The Laravel application deployed opens in a new instance of the browser.

Conclusion

In this article, I laid down the foundation for the coming articles in this new series. Starting from the very basics of deploying a Laravel project to GCP is a steppingstone in understanding the rest of the features and requirements of deploying applications to GCP.

Next time, I'll dive into database topics like creating and using a GCP Cloud SQL and connecting to a Cloud SQL from within a Laravel application. Also, you'll see how to run Laravel migrations as part of the Cloud Build Workflow, and much more.

Stay tuned!

Bilal Haidar

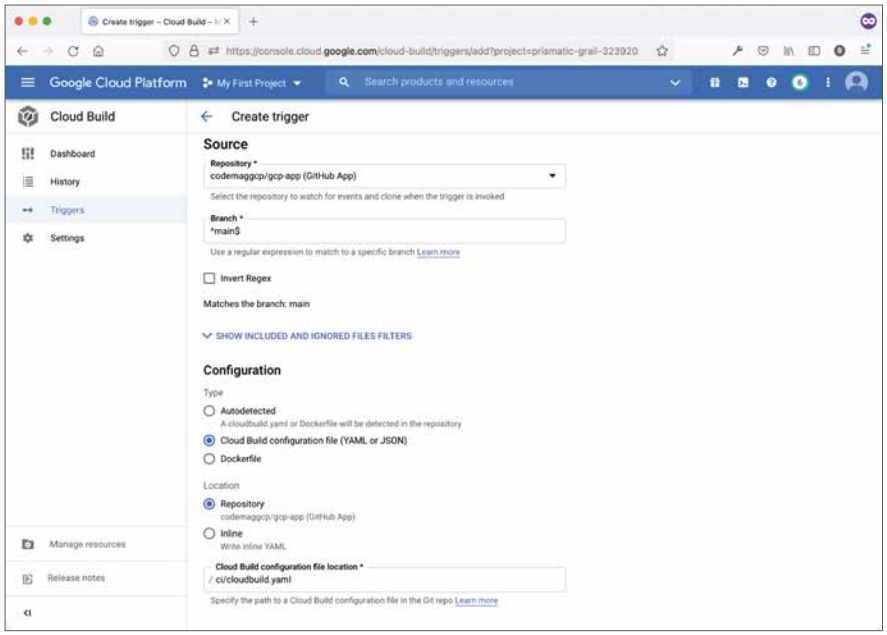


Figure 16: Specify the cloudbuild.yaml file path.

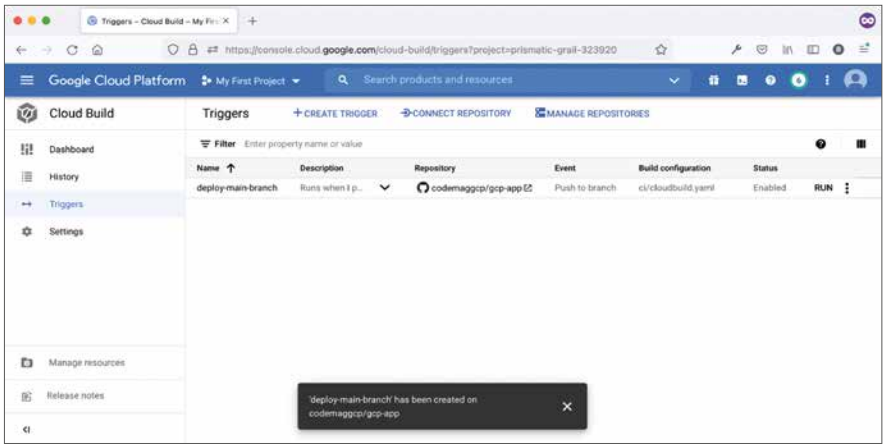


Figure 17: The new trigger is created.

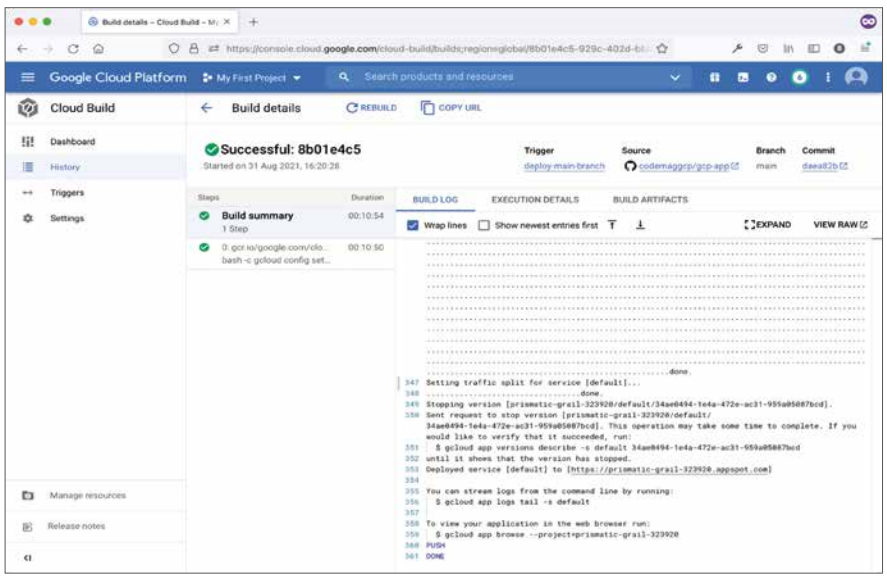


Figure 18: A successful run of a cloud build trigger

Designing Microservices Architecture for Failure

Microservices architecture is a structural approach that organizes an application as a collection of small independent services modeled around a business domain that can communicate with one another, if need be. In a typical distributed environment, dealing with unexpected errors or failures is challenging. When building microservices, you should consider resilience to failures.

Joydip Kanjilal
joydipkanjilal@yahoo.com

Joydip Kanjilal is an MVP (2007-2012), software architect, author, and speaker with more than 20 years of experience. He has more than 16 years of experience in Microsoft .NET and its related technologies. Joydip has authored eight books, more than 500 articles, and has reviewed more than a dozen books.

Hence, your design should support resilience and high availability. The resiliency of an application is defined as its capability to extricate from failures. Resilient microservices are adept at withstanding faults and unexpected failures.

This article talks about how you can design your microservices for failure, i.e., make them resilient and fault-tolerant.

Prerequisites

If you're to work with the code examples discussed in this article, you should have the following installed in your system:

- Visual Studio 2022 Preview (an earlier version will also work but Visual Studio 2022 is preferred)
- .NET 6.0 Preview
- ASP.NET 6.0 Runtime Preview

You can download Visual Studio 2022 from here: https://visualstudio.microsoft.com/downloads/.

The Problem with Monoliths

A monolithic application creates problems in maintaining the code base, implementing continuous deployment, and even adding new features when needed. In an application based on monolithic architecture, a single fault can bring down the entire application.

Moreover, unlike a microservice, a monolith can scale in a single dimension only. If you're working on an application with dispersed teams, it makes more sense to use microservices architecture instead of monolithic architecture.

> Distributed teams flourish with microservices because they can work independently, yet connected, with the larger whole of the application.

Distributed teams often find it helpful to use microservices architecture because it allows them to work independently. Microservices architecture offers an incredible amount of modularity and flexibility. Moreover, several teams can work on a single microservice built using homogenous or heterogeneous technologies. Microservices architecture is more cost-effective and can help you to scale the application seamlessly.

Introduction to Microservices Architecture

Microservices architecture is an architectural style used to create flexible, extendable, and independently deployable services built using a collection of homogenous or heterogeneous technologies. In microservices, you have a single application comprised of a suite of small, independently deployable services, each of which executes and communicates with the others via lightweight communication mechanisms. Given the distributed nature of a microservices-based architecture, you should develop scalability strategies that protect microservices-based systems against outages and make them fault-tolerant.

A microservices-based application comprises loosely connected services developed and deployed separately and can run on heterogeneous platforms. It's an architectural style used to build applications by splitting business components into small autonomous services. The Microservices Architecture pattern imposes a degree of modularity that's very difficult to accomplish with a monolithic codebase.

Advantages of Microservices Architecture

Improved Scalability: To scale a microservices-based application, you'd need to scale only specific components—this optimizes resource usage to a considerable extent. The decoupled nature of the microservices enables them to be modified independent of other services, eliminating the possibility of interfering with the operation of other services.

Reduced coupling: Reduced coupling between the components of an application makes it adaptable for changes over time easily.

Better ROI: Microservices architecture facilitates the separation of responsibilities that's essential for developing highly scaled applications. It allows developers to work independently on individual services without interfering with the work of other developers.

Faster Deployment and Reduced Development Time: Different teams can collaborate on independent services when you use microservices, allowing you to deploy more rapidly. Usage of microservices architecture cuts down on the development time because of the loosely coupled and autonomous nature of microservices.

Faster Releases: You don't have to rebuild your whole codebase if you want to add or change a feature because microservices operate with loosely connected services. The decentralized nature of developing, maintaining, and updating applications allow for faster deployments and releases. Each microservice can be built, tested, and deployed

individually. So, you can treat each microservice as an independently deployable unit.

The Need for Resilient Microservices

In today's ever-changing world of IT, legacy architectures are not sufficient for the needs of the business. Modern day applications should be able to run 24x7 with an uptime as close as possible to 100%, and they must be resilient, scalable, and cloud native.

The capability of an application to recover from failures is referred to as resiliency. In a typical distributed environment, dealing with unexpected errors or failures is challenging. Despite all the benefits provided by microservices, there are new challenges, because the services must communicate with one another or communicate with a database over a network.

Unlike a microservices application, in a monolithic application, one error might bring down the entire application. Because a microservices application comprises several small independently deployable units, failure of one part of the application doesn't bring down the entire application.

When building microservices, consider resilience to failures. Your design should support resilience and high availability. This article presents a discussion on the strategies that can be adopted for building resilient microservices so that your applications can be fault-tolerant and can handle failures gracefully.

A microservices application is not resilient to failures by default so you must design your application with failure in mind. A microservices-based application should be architected in such a way that even if a service is down (for whatever reason, such as network failure, bug in the application, network outage, or system failure), the application should continue to function and be able to recover gracefully from the failure with minimal effort.

Patterns of Microservices Architecture

There are several patterns you can follow to build resilient applications using microservices architecture.

Retry

In a typical microservice-based application, the services might often have several dependencies, such as back-end services and components or databases. There might be network outages, due to which a service call might fail. One possible solution to this problem is using the Retry pattern.

The Retry mechanism can be used for intermittent or instantaneous failures and is typically used to retry a failed operation after a specified duration for a specified number of times, both of which are usually configurable. The objective of this pattern is to ensure that failed services are invoked one or more times after a specific time interval until the expected response from the service is received.

Note that you should avoid chaining retries and only retry transient failures. You should also log appropriately so that you can examine these logs later and then determine the cause of failures. You should give your service the time needed to recover to avoid cascading failures. In doing so, you can save on network resources while, at the same time, allow the failed service to recover.

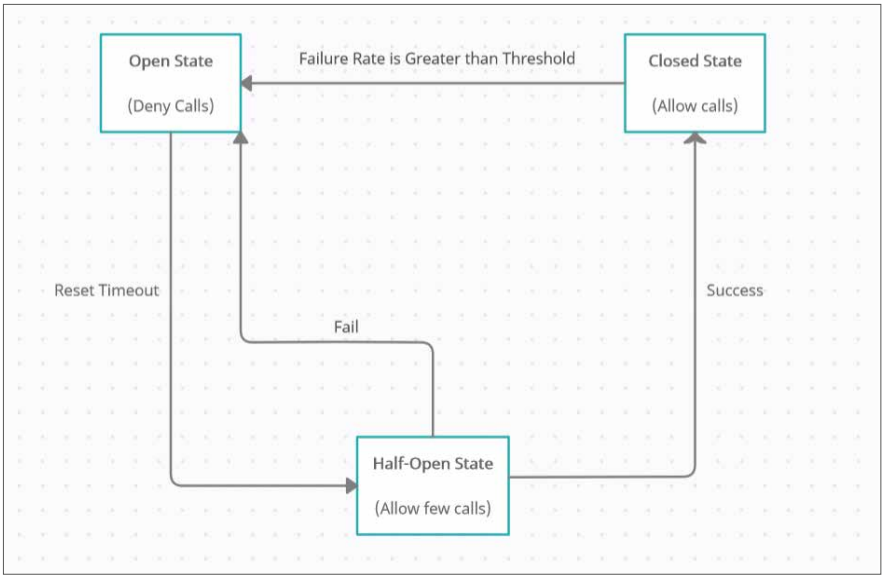


Figure 1: Illustrating the circuit breaker pattern

Circuit Breakers

When working in microservices-based applications, you might invoke a failed service several times until you get the desired result. However, invoking the service in this way might result in more damage and cascading failures. The Circuit Breaker pattern can be used as a solution to this problem—you can take advantage of this pattern to shield your application against non-transient faults.

Your microservices-based application may suffer from faults for several reasons. Such faults can broadly be categorized into the following:

- **Transient**: These faults are often intermittent, and they may cause the application to be unavailable for a brief period, usually a few seconds.
- **Non-transient**: These faults can bring down the application for minutes or even hours

A circuit breaker is a state machine that begins in a closed state and enables requests to pass through it. When a fault is determined, the circuit breaker switches to an open state, which prevents the processing of requests for a predetermined duration. The circuit breaker switches to a half-open condition once that time has elapsed, and the initial request is considered a test request. If this request is successful, the circuit is closed, and normal functioning restored. If the request is unsuccessful, the circuit is reset to an open state. It stays there for a predetermined amount of time before being reset to a half-open state once again. See **Figure 1** for an illustration.

Essentially, the circuit breaker is a component that sits in between the caller of the service (i.e., the service consumer) and the service endpoint. Here's how the circuit breaker pattern works:

- When the services are communicating under normal circumstances, the caller invokes the circuit breaker, which, in turn, delegates the call to the service endpoint. Note that currently, the circuit breaker is in a closed state.
- When the services are communicating in a faulty state, i.e., when there is an invocation failure, the circuit breaker continues in the closed state and maintains a

count of the number of times a service call has failed. When this count reaches or exceeds a pre-defined threshold, the circuit breaker opens the circuit.

- When the circuit breaker is in an open state, the circuit breaker itself sends back the response rather than delegating the service call request to the actual service. Typically, this response consists of an error code and an error message. After the circuit reset timeout has elapsed, the service is re-invoked to determine whether the connectivity has been restored.
- The circuit breaker can also be in a half-open state. In this state, the caller makes another request to the circuit breaker. This call is then delegated to the service endpoint and the response from the service is passed all the way to the service caller.
- When the state becomes stable, i.e., when the service can be called again successfully, the circuit breaker goes back to the closed state. Calls to the service endpoint from the service caller are deemed appropriate and the response from such calls are returned to the service caller.

The circuit breaker can be used in conjunction with the retry pattern to reduce or eliminate resource wastage and prevent cascading failures. You should design the circuit breaker in such a way that it's able to examine the service failures and then change the strategies as appropriate. The circuit breaker should also be thread-safe and asynchronous to make it high performant and scalable.

Bulkheads

This is yet another pattern ensuring that faults in one component of the application don't affect the other components. The bulkhead achieves this by partitioning the application into granular, independent components. These components usually have business logic that's isolated from the rest of the application. In the event of a service call failure, the component or components that are affected don't bring the entire application to a standstill.

The Bulkhead pattern is analogous to the bulkheads used in ships. A bulkhead is a wall that separates different cargo sections of a cargo ship, ensuring that any fire or flood in one section is limited to that section only and doesn't affect the other sections of the ship.

The Bulkhead design isolates the elements into pools such that the failure of one pool wouldn't bring the entire application down. As a result of this isolation, the application's components will continue to function even if one of them has failed. To implement the Bulkhead pattern, you can take advantage of the single responsibility, asynchronous communication, and fail-fast patterns.

Other Considerations

You can use rate limiters to limit the number of calls to your service within a specified time interval. This ensures that the service isn't overloaded with service calls—typically a load balancer is used to increase scalability if needed.

Load balancing is another option for You; you can take advantage of load balancing to distribute the load as appropriate among several services and, in the event of a service failure, invoke an alternate service.

Implementing the Circuit Breaker Pattern in ASP.NET Core 6

This is a simple Order Management application with two microservices: Order and Product. I'll build two applications: one that throws an exception at runtime (the Product microservice) randomly and another that leverages the circuit breaker pattern (the Order microservice) to implement resilience.

Note that I'll be using the terms "Order microservice" and "OrderApi" interchangeably—they refer to the same project. Likewise, I'll use the terms "Product microservice" and "ProductApi" interchangeably—both point to the same project.

An order can have several products, so you should be able to know the product details, such as product code, product name, price, etc. The Order microservice calls the GetProducts method of the Product microservice to get a list of the products to prepare OrderDetails and send it back to the client. When an exception is thrown in the Product microservice, the GetOrderDetails method of the Order microservice doesn't return any data. Rather, detailed exception trace information will be displayed, as shown in **Figure 2**.

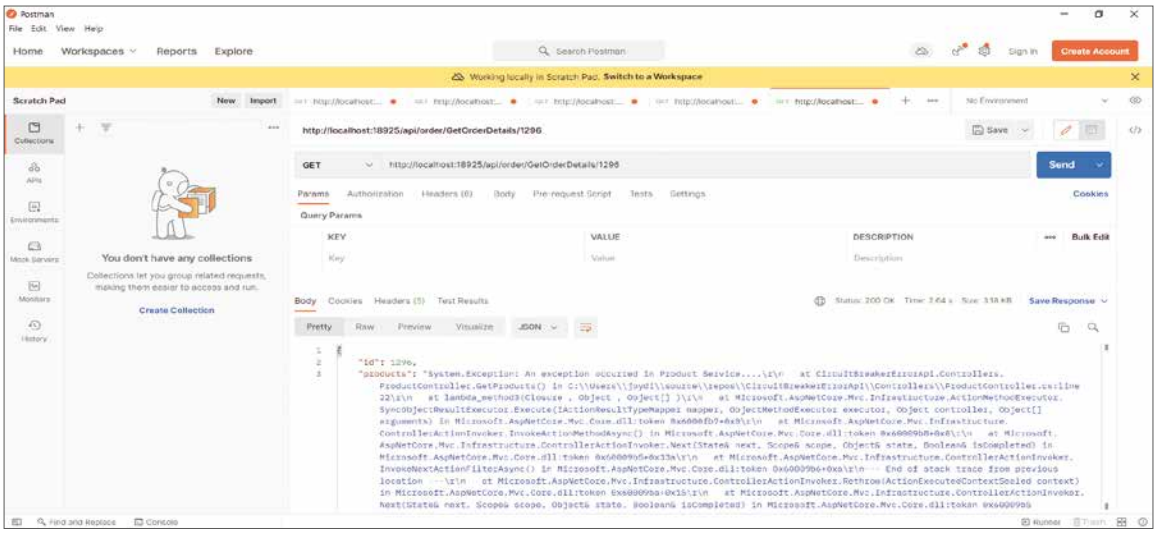


Figure 2: Detailed exception trace of the exception thrown in Product Microservice is displayed

Listing 1

```
namespace ProductApi;
public class ProductService: IProductService
{
    public List<Product> GetProducts()
    {
        var random = new Random().Next(1, 25);

        if (random > 10)
        {
            throw new Exception
                ("An exception " +
            "occurred in Product Service....");
        }

        List<Product> products =
            new List<Product>()
        {
            new Product { Id = 1, Code = "P0001",
                Name = "Lenovo Laptop",
                Price = 1200.00 },
            new Product { Id = 2, Code = "P0002",
                Name = "DELL Laptop",
                Price = 1300.00 },
            new Product { Id = 3, Code = "P0003",
                Name = "Asus Laptop",
                Price = 1100.00 },
            new Product { Id = 4, Code = "P0004",
                Name = "HP Laptop",
                Price = 1400.00 }
        };

        return products;
    }
}
```

Listing 2

```
public void ConfigureServices(IServiceCollection
            services)
{
   services.AddScoped<IProductService,
       ProductService>();
   services.AddControllers();
}
```

I'll now build both these applications in the sections that follow.

Create a New ASP.NET 6 Project in Visual Studio 2022

Let's start building the Product microservice first. You can create a project in Visual Studio 2022 in several ways. When you launch Visual Studio 2022, you'll see the Start window. You can choose "Continue without code" to launch the main screen of the Visual Studio 2022 IDE.

To create a new ASP.NET 6 project in Visual Studio 2022:

1. Start the Visual Studio 2022 Preview IDE.
2. In the "Create a new project" window, select "ASP.NET Core Web API" and click Next to move on.
3. Specify the project name as ProductApi and the path where it should be created in the "Configure your new project" window.
4. If you want the solution file and project to be created in the same directory, you can optionally check the "Place solution and project in the same directory" checkbox. Click Next to move on.
5. In the next screen, specify the target framework and authentication type as well. Ensure that the "Configure for HTTPS," "Enable Docker Support," and the "Enable OpenAPI support" checkboxes are unchecked because you won't use any of these in this example.
6. Click Create to complete the process.

Follow the same steps outlined above to create another ASP.NET Core 6 Web API project. Name this project OrderApi. Note that you can also choose any meaningful name for both these projects. You now have two ASP.NET Core 6 Web API projects: the ProjectApi and the OrderApi.

Let's now create the classes and interfaces for both the two microservices.

Build the Product Microservice

Let's start building the Product microservice first.

Create the Model

Create a new file at the root of the ProductApi project named Product.cs and write the following code in there:

```
namespace ProductApi;

public class Product
{
    public int Id { get; set; }
    public string Code { get; set; }
    public string Name { get; set; }
    public double Price { get; set; }
}
```

Create the Product Service

The ProductService class creates a list of products and populates it with some data in the constructor. It contains a method called GetProducts that returns the list of products. The IProductService interface contains the declaration of the GetProducts method. The following code snippet shows the IProductService interface.

```
namespace ProductApi;
public interface IProductService
{
    public List<Product> GetProducts();
}
```

The ProductService class implements the IProductService interface. **Listing 1** illustrates the ProductService class.

Note the use of Random class in the GetProducts method of the ProductService class. This method generates a random number between 1 and 25 and checks whether its value is more than 10. If the random number has a value more than 10, an exception is thrown.

Register the ProductService class

You should register the ProductService class in the ConfigureServices method so that you can use dependency injection to access its instance in the ProductController class as shown in **Listing 2**.

Create the ProductController Class

The ProductController takes advantage of ProductService to return a list of products as shown in **Listing 3**.

Listing 3

```
using Microsoft.AspNetCore.Mvc;
using ProductApi;

namespace CircuitBreakerErrorApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductController : ControllerBase
    {
        private readonly IProductService
            _productService;
        public ProductController(IProductService
            productService)
        {
            _productService = productService;
        }
        [HttpGet("GetProducts")]
        public ActionResult<List<Product>>
        GetProducts()
        {
            return _productService.GetProducts();
        }
    }
}
```

Build the Order Microservice

So far so good. Let's now create the classes and interfaces of the Order microservice.

Create the Model

The Order class represents the only model class in the OrderApi project. The following snippet illustrates the Order class.

```
namespace OrderApi
{
    public class Order
    {
        public int Id { get; set; }
        public string Products { get; set; }
    }
}
```

Note that the Products property in the Order class would hold the list of products returned by the Product microservice. This property is of type string for simplicity.

Install NuGet Packages

In this article, I'll take advantage of Polly, a mature library available as a NuGet package used for implementing the retry and Circuit Breaker patterns. Polly is a .NET and .NET Core compliant library that can be used to build resilient applications. You can take advantage of Polly to define policies that dictate what should happen when an error or exception occurs.

A NuGet package is represented as a file that has a .nupkg extension and comprises compiled code (also called DLLs), other related files, and a manifest that provides information related to the package, such as version number, etc.

Let's now install the necessary NuGet Package(s) in the OrderApi project. To install the required packages into your

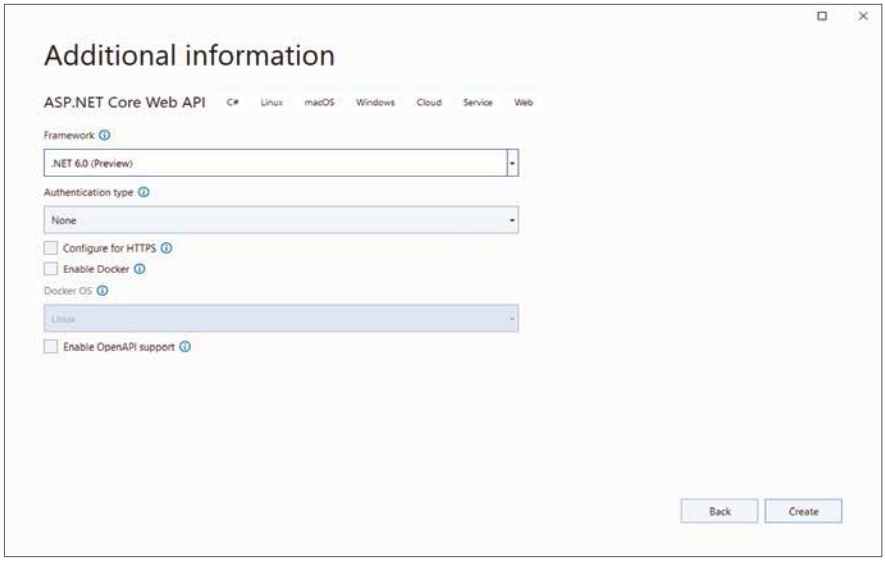


Figure 3: Specify additional information while creating the project

Listing 4

```
public void ConfigureServices(IServiceCollection
   services)
{
   var productApiEndpointAddress = new
               Uri("http://localhost:49510");
    services.AddHttpClient("ProductApi", c =>
    {
        c.BaseAddress =
        productApiEndpointAddress;
    })
    .AddTransientHttpErrorPolicy(p => p.
    CircuitBreakerAsync(2,
    TimeSpan.FromMinutes(1)));
    services.AddControllers();
}
```

Listing 5

```
using Microsoft.AspNetCore.Mvc;
using System.Net.Http;
using System.Threading.Tasks;

namespace OrderApi.Controllers
{
  [Route("api/[controller]")]
  [ApiController]

  public class OrderController : ControllerBase
  {
      private readonly IHttpClientFactory
              _httpClientFactory;
      public OrderController(IHttpClientFactory
            httpClientFactory)
        {
            this._httpClientFactory =
            httpClientFactory;
        }

      [HttpGet("GetOrderDetails/{orderId}")]
      public async Task<ActionResult<Order>>
            GetOrderDetails(int orderId)
      {
         var client = _httpClientFactory.
         CreateClient("ProductApi");
         var response = await client.
         GetAsync("api/product/getproducts");
         var products = await response.
         Content.ReadAsStringAsync();
         return new Order() {Id = orderId,
         Products = products };
      }
  }
}
```

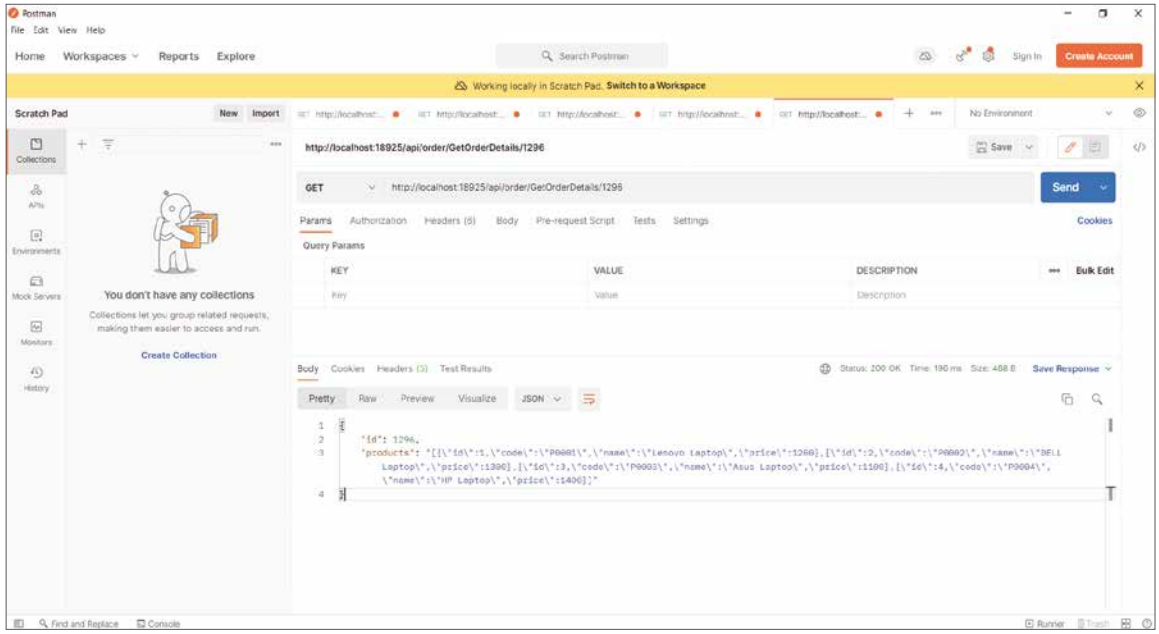


Figure 4: Displaying Order Details for a specific order

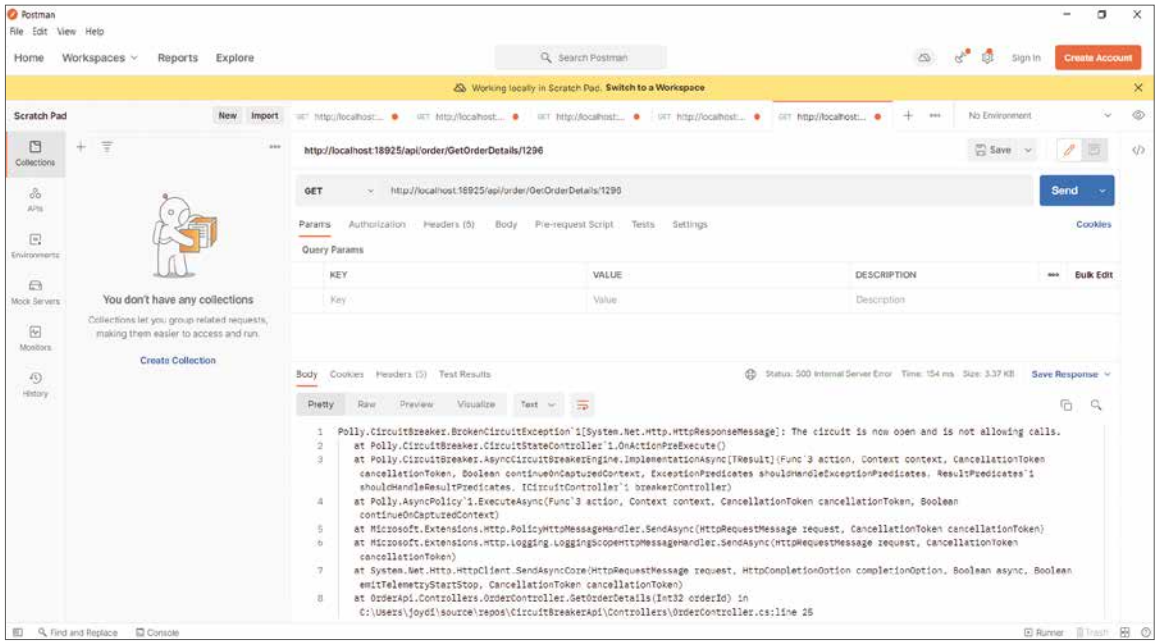


Figure 5: The circuit is now open and all subsequent calls are disallowed.

project, execute the following commands at the NuGet Package Manager Console.

```
dotnet add package Microsoft.Extensions.
Http.Polly
dotnet add package Microsoft.Extensions.
Configuration
dotnet add package Microsoft.Extensions.
DependencyInjection
```

Configure Polly

Listing 4 illustrates how you can configure Polly and specify Circuit-Breaker Policy in the ConfigureServices method of the Order microservice.

Create the OrderController Class

Create a new controller named OrderController in the Controllers solution folder the OrderApi project and replace the default code with the code illustrated in **Listing 5**.

Execute the Application

Run both the applications and then hit the HTTP Get endpoint GetOrderDetails of the Order microservice, as shown in **Figure 4**.

You can see the data this time. Now, send the request multiple times and you encounter the circuit breaker error. After two consecutive failures (exception thrown by the Product

microservice), the circuit opens, and a circuit breaker exception is thrown with the error message shown in **Figure 5**.

Summary

Microservice architecture has been an emerging trend in software architecture over the past few years and holds many promises: faster time to market, better scalability, and loosely coupled components. However, this conglomeration of several small autonomous services modeled around a business domain comes at a cost—it complicates service to service communication, handling data consistency, and increases the attack surface manifold.

A microservice-based application must be reliable. Your microservices should fail fast and open the circuit. You should design your microservices so that you can revert to a previous version if needed. Your microservices should not only be fault-tolerant but should be able to handle failures gracefully, such as using the circuit breaker pattern to manage resilience in microservices-based applications.

Joydip Kanjilal

(Continued from 74)

Assuming that you have no codified and accepted rules and are starting from scratch, one question that needs to be answered is to ask, "for a given set of required response data, what's the required request message data?" That question begets several other questions, not the least of which is authentication. Does the function operate in an acceptable manner and what's the best evidence to answer that question?

The best evidence for answering the acceptability question isn't the documentation. Documentation may be the best evidence for how the function is specified to work. But how the function actually works under the circumstances relative to your requirements, that's the gap that must be closed.

It turns out that documentation, in this context, is irrelevant. The best acceptability evidence is a set of test fixtures that exercise the function within some range of load. The fixture not only demonstrates performance, but the code also explains how to consume the function, whether as an embedded resource or as a call to an external endpoint (API). If you're looking for a better way to incorporate technical documentation into your development, well-written tests, whether they be unit, performance, load, or integration, are your best source, as they not only illustrate what's to be done, but how it's done, and the end result is always success or failure.

In my experience, the best evidence of whether a function works is its track record of operation under production conditions. In other words, is the API reliable under a given load? These API metrics are important because it's how the team consuming the API can isolate its performance data from the API. It's also important for the cloud engineers when they're assessing how to configure the platform in which the component under development will be hosted. And that gets to why projects benefit from legislation (rules-making) and administration.

Today, development is more complicated than ever. Whether it's distributed teams, the ever-changing OSS used, the cloud platforms themselves, or the legal and regulatory environment, there's an ever increasing need to explain how things work, how bad things happened, and why a proposed remediation is likely to have a benefit. The only way those questions can be answered is if there's the requisite data that was generated as a direct by-product of a process that was, in turn, subject to established and enforced rules.

Let's assume that for test purposes, the function works. What happens when you put it into production? Production is the most important test of all. That's where instrumentation by way of logging and telemetry come into play. But the only way those things see the light of day is if instrumentation facilities are in scope as a first-class feature and the only way to mandate that such features exist is by rule. But rules can't enforce themselves. That's where the people on a development team must hold a common sentiment around why these things are important. Leadership must also share that exact same common sentiment.

The best evidence of a mature and well-functioning team is one that can directly demonstrate A): that it enforces codified and adopted rules and procedures, and B): objectively verifiable high-quality software is delivered in a timely manner to production, subject to those rules and procedures. To the extent that development teams and the businesses collaborate and cooperate where they are each independently reading from the same playbook and are guided by and believe in the same over-arching principles, the software under development is afforded a meaningful opportunity at high quality and success.

That's the question the business ultimately must answer: whether it wants high-quality software. If the business does, it will afford teams the time necessary to cultivate the best evidence to answer its questions and it won't turn a blind eye to rules and procedures. The result? Better, high-quality software deliveries enabled by rules that kept the teams on the right path.

John V. Petersen

CODE

Nov/Dec 2021
Volume 22 Issue 6

Group Publisher
Markus Egger

Associate Publisher
Rick Strahl

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Contributing Editor
John V. Petersen

Content Editor
Melanie Spiller

Editorial Contributors
Otto Dobretsberger
Jim Duffy
Jeff Etter
Mike Yeager

Writers In This Issue
Bilal Haidar
Wei-Meng Lee
John V. Petersen
Helen Wall
Joydip Kanjilal
Sahil Malik
Paul D. Sheriff
Shawn Wildermuth

Technical Reviewers
Markus Egger
Rod Paddock

Production
Friedl Raffeiner Grafik Studio
www.frigraf.it

Graphic Layout
Friedl Raffeiner Grafik Studio in collaboration with onsight (www.onsightdesign.info)

Printing
Fry Communications, Inc.
800 West Church Rd.
Mechanicsburg, PA 17055

Advertising Sales
Tammy Ferguson
832-717-4445 ext 26
tammy@codemag.com

Circulation & Distribution
General Circulation: EPS Software Corp.
Newsstand: American New Company (ANC) Media Solutions

Subscriptions
Subscription Manager
Colleen Cade
ccade@codemag.com

US subscriptions are US $29.99 for one year. Subscriptions outside the US are US $50.99. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards accepted. Bill me option is available only for US subscriptions. Back issues are available. For subscription information, e-mail subscriptions@codemag.com.

Subscribe online at
www.codemag.com

CODE Developer Magazine
6605 Cypresswood Drive, Ste 425, Spring, Texas 77379
Phone: 832-717-4445

CODA: On Rules and Procedures

If we compare a software project to a tunnel construction project, each has two ends and construction begins from both ends with the ultimate objective being that they meet each other at a pre-defined location. For software, there's the business side and there's the delivery side. Each side must have a clear and common understanding of what is to be delivered. That's the project. But what governs the project? The rules we codify, adopt, and enforce.

Rules are a means by which an activity is constrained through one or more requirements. Procedures are the means by which the benefits conferred by rules are realized. Procedures are themselves a form of rule and are subject to rules.

In the legal realm, there exists the Best Evidence Rule for documents sought to be admitted at trial. The rule is simple: The "Original" is deemed to be the best evidence of what the document purports to contain. The rule is codified in the Federal Rules of Evidence (FRE), Rule 1002—Requirement of the Original. That rule states: "An original writing, recording, or photograph is required in order to prove its content unless these rules or a federal statute provides otherwise." In plain English, the rule presumes using an original, not a copy. In the event that the original isn't available, the burden falls to the party seeking to admit the copy to successfully argue why they should be permitted to do so. There are other rules that govern what's required to make such an argument and the rule quoted above acknowledges that it's subject to other rules.

What if the original isn't available and only a copy is available? Is it a true and correct original copy? Or is it something else? Rules 1003-1007 address the various scenarios when the original isn't available. There are other rules, such as the Federal Rules of Criminal Procedure and Civil Procedure to which the FRE applies. From a systems perspective, the Federal Rules of Criminal and Civil Procedure each have a dependency on the Federal Rules of Evidence. This is in the context of a criminal or civil trial where evidence is sought to be admitted. What's required to seek admittance? How do we decide what's admissible? If admissible, how do we admit it? This is what a judge is tasked with deciding. And to make those decisions, very simplified, a judge applies the facts to rules and makes a ruling. For the party that lost the ruling, the rules and procedures specify how to preserve issues for appeal and keep the trial moving forward. If the trial can't move forward, it just stop.

A useful, overarching metaphor is that rules should be the sharp edges that keep our work on the correct path. When a rule is violated, it should cause some level of discomfort. For if there are no consequences, what good are rules in the first place? Any software project is subject to constraints, whether they by imposed by the business, engineering, or through governmental regulation. In that regard, software projects are no different in concept than any construction project, legal case, business plan, marketing program, etc. At a certain level, the things humans produce, which includes process and procedures, are alike to the extent that all are subject to constraints, they are undertaken to produce value, and when things don't work, the consequences are noticeable, and perhaps quite uncomfortable. Rules are your compass.

Despite the utility that rules provide, people often see rules and procedures as cumbersome things that just get in the way. On the other hand, people want to know what, if any, rules exist to inform how an action should be undertaken. Whether it's a trial, organizational administration, or a software project, there are many details at play, initiated by different people, and all of which must be compatible with one another. Mature teams employ the discipline and rigor to develop operating rules and principles based on the best evidence that provides direct empirical data, and that allows teams to objectively eliminate or rank alternatives. Mature and competent leadership recognizes that teams must be afforded the time to develop such rules, which requires a lot of evidence gathering.

The gap between what the desired state is, based on rule and actual state, those are crucial data points that don't come for free. It all takes time and thus, incurs a cost. It only makes sense if there's an associated increase in quality. Whether it's a legal or a development team, the thing being acted upon is an undertaking. That's what a software project essentially is, an undertaking. Although the subject matter is different, at an abstract level, all undertakings and projects are the same in that they start, have some period of activity, and they finish. For there to be any objective quantitative or qualitative post-software delivery analysis, there must be some measurable basis to do so. Rules specify how that requirement is to be enforced.

Rules are important, but you may not have anything formalized. When you recognize that, if you're afforded the time to start codifying your rules, start that task immediately. An excellent place to start is with your team's Definition of Done.

Before delving into a specific example, I want to address the importance of shared context. For the rules discussed thus far, the shared context is the U.S. Constitution. Without a shared context, we're each left to make our own assessment. In a group that needs to collaborate and cooperate in order to achieve some end, our respective assessments need to reflect our individual points of view, but also give way to some common assessment that's based on some operating rule or procedure. The shared context in this case isn't something as broad as "Technology." That's too broad of an expanse and can only be addressed in an abstract way.

The broad context isn't your specific project either. Your project is made up of several, perhaps many, sub-components, that are all subject to their own rules. And often, there may be conflict among those rules. Perhaps you need to write a gateway for an external API. That's a manageable context for this examination. What are the rules for interfacing with an external API? There are your own development standards perhaps. There are also legal and regulatory requirements (think HIPAA, PCI, SOC, etc.). From project to project and within a project, the goal should be to standardize your decision-making. You might find that relevant facts are entirely dissimilar—even so, the basis on how to decide how to proceed should be consistent. Otherwise, how can you measure a team's performance, especially if they're required to re-invent the wheel each time decisions must be made? Going back to the trial metaphor, from trial to trial, the facts will be different. But the conceptual framework that's independent of any one case is what's used to determine whether a specific document is admissible evidence. In other words, rules become the distilled melting pot of past, specific instances. That's true with the law and it's certainly true in software development.

(Continued on page 73)

The intersection of geospatial + the built world

GEO WEEK

AEC NEXT

INTERNATIONAL LIDAR MAPPING FORUM

SPAR3D

FEBRUARY 6-8, 2022
DENVER, CO - USA
geo-week.com

Accomplish a year's worth of geospatial business in just one week by attending Geo Week 2022

Imagine a single powerhouse event that champions the coming together of geospatial technologies and the built environment. Where professionals from a range of disciplines network and gain insight into the increasing confluence of their worlds. Where cutting-edge technology offers new possibilities, improved efficiencies, and better outcomes. And where education opens the door to the future just ahead.

AEC Next Technology Expo & Conference, International Lidar Mapping Forum, and SPAR 3D Expo & Conference, along with partner events ASPRS Annual Conference and USIBD Annual Symposium, are coming together in 2022 to form Geo Week. Each event features its own unique conference programming and combines in a single exhibit hall and inclusive networking activities. Welcome to Geo Week!

Use code SAVE100 for $100 off a conference pass or a free exhibit hall pass.

Learn more!
geo-week.com

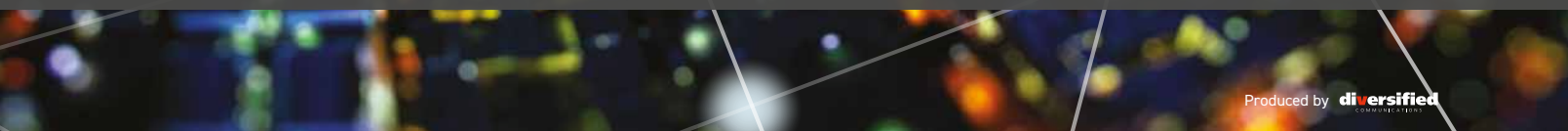
INDUSTRIES SERVED

- Architecture, Engineering & Construction
- Asset & Facility Management
- Disaster & Emergency Response
- Earth Observation & Satellite Applications
- Energy & Utilities
- Infrastructure & Transportation
- Land & Natural Resource Management
- Mining & Aggregates
- Surveying & Mapping
- Urban Planning/ Smart Cities

PARTNER EVENTS

asprs THE IMAGING & GEOSPATIAL INFORMATION SOCIETY

USIBD U.S. Institute of BUILDING DOCUMENTATION



Produced by diversified

NEVRON OPEN VISION (NOV)

The future of .NET UI development is here

NEVRON

GPU Accelerated OpenGL Rendering

NOV lets you build applications for Windows, Mac and Blazor using a single codebase.

The suite features industry-leading UI components including Chart, Diagram, Gauge, Grid, Scheduler, Rich Text Editor, Ribbon, and others.

Nevron provides advanced solutions for:

Microsoft ASP.net | Blazor | SharePoint | Microsoft SQl Server | WPF | WinForms | Xamarin

Learn more at: **www.nevron.com** today