

ChatGPT, Node.JS, ASP.NET Core, GraphQL, Microservices

CODE

JUL  
AUG  
2023

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

Cover: @midjourney + on sightdesign

# CODE

AI

## Understanding AI

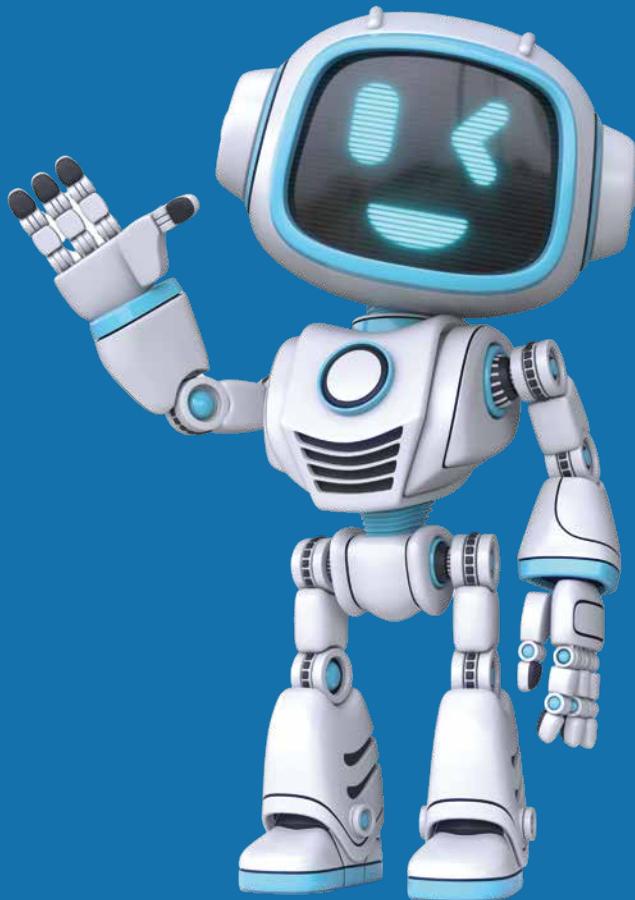
(This cover was designed by an AI)

Building  
Microservices  
with GraphQL

Building APIs  
with Node.JS  
and Express

Exploring  
the World  
of AI





ARE YOU WONDERING  
HOW ARTIFICIAL  
INTELLIGENCE CAN  
BENEFIT YOU TODAY?

## **EXECUTIVE BRIEFINGS**

Are you wondering how AI can help your business? Do you worry about privacy or regulatory issues stopping you from using AI to its fullest? We have the answers! Our Executive Briefings provide guidance and concrete advise that help decision makers move forward in this rapidly changing Age of Artificial Intelligence and Copilots!

**We will send an expert to your office to meet with you. You will receive:**

1. An overview presentation of the current state of Artificial Intelligence.
2. How to use AI in your business while ensuring privacy of your and your clients' information.
3. A sample application built on your own HR documents – allowing your employees to query those documents in English and cutting down the number of questions that you and your HR group have to answer.
4. A roadmap for future use of AI catered to what you do.

## **AI-SEARCHABLE KNOWLEDGEBASE AND DOCUMENTS**

A great first step into the world of Generative Artificial Intelligence, Large Language Models (LLMs), and GPT is to create an AI that provides your staff or clients access to your institutional knowledge, documentation, and data through an AI-searchable knowledgebase. We can help you implement a first system in a matter of days in a fashion that is secure and individualized to each user. Your data remains yours! Answers provided by the AI are grounded in your own information and is thus correct and applicable.

## **COPILOTS FOR YOUR OWN APPS**

**Applications without Copilots are now legacy!**

But fear not! We can help you build Copilot features into your applications in a secure and integrated fashion.

---

**CONTACT US TODAY FOR A FREE CONSULTATION AND DETAILS ABOUT OUR SERVICES.**

**[codemag.com/ai-services](http://codemag.com/ai-services)**

832-717-4445 ext. 9 • [info@codemag.com](mailto:info@codemag.com)

# Features

## 8 Building Web APIs Using Node.js and Express: Part 2

In this second article of his new series, Paul shows you how to move hard-coded values into a configuration file and then read those settings into an application. You'll also add middleware functions, write some errors, and retrieve data from a SQL Server table.

**Paul D. Sheriff**

## 19 OpenAI

If you've been living under a rock, you won't have heard about AI. Sahil has a look at the state of the technology, what it's already accomplished, and what it looks like is coming down the road.

**Sahil Malik**

## 26 An Introduction to OpenAI Services

Wei-Meng shows you how the inner workings of ChatGPT and Large Language Models (LLM) work with OpenAI.

**Wei-Meng Lee**

## 50 Building Microservices Architecture Using GraphQL and ASP.NET 7 Core

Joydip compares GraphQL with REST and gRPC and shows you how to take advantage of GraphQL when building microservices in ASP.NET Core7.

**Joydip Kanjilal**

# Columns

## 74 CODA: The Road Not Taken

Some decisions are minor and others affect everything. John looks at the decision-making process and how the pandemic has altered how we think about change.

**John V. Petersen**

# Departments

## 6 Editorial

## 25 Advertisers Index

## 73 Code Compilers

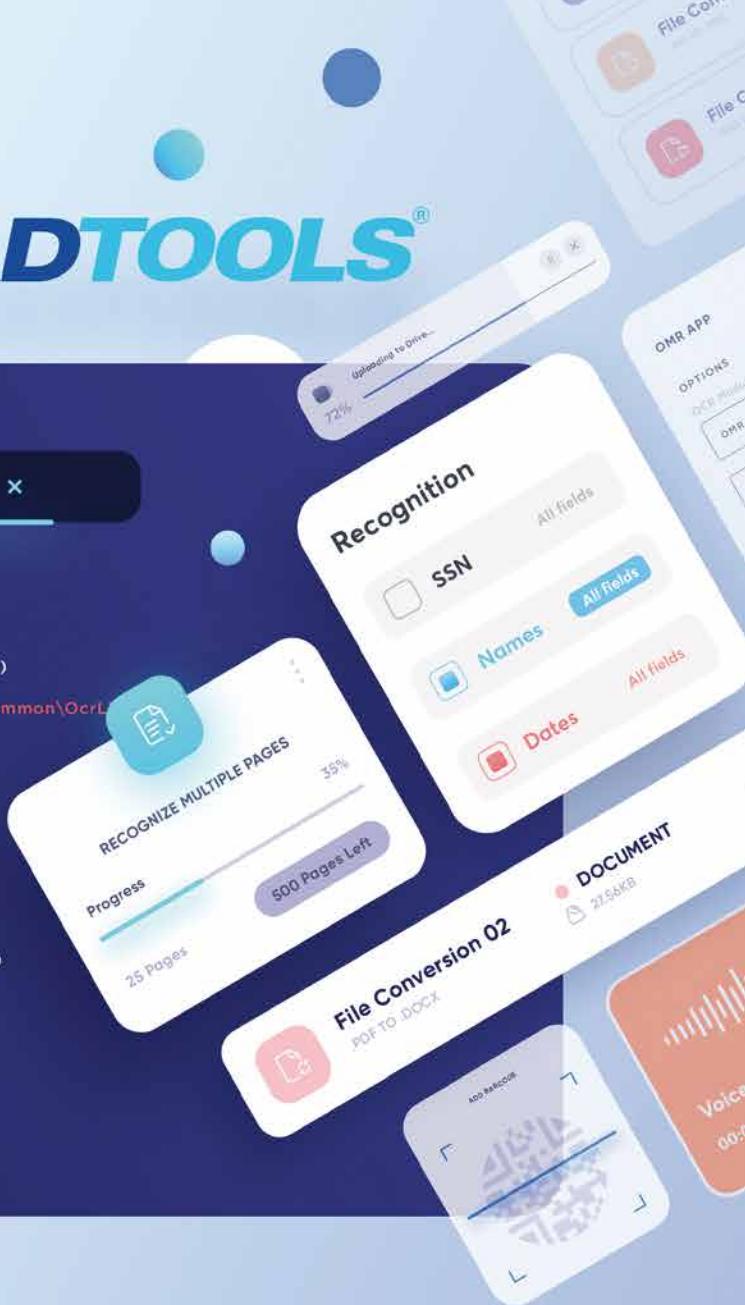
US subscriptions are US \$29.99 for one year. Subscriptions outside the US pay \$50.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards are accepted. Back issues are available. For subscription information, send e-mail to [subscriptions@codemag.com](mailto:subscriptions@codemag.com)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

Subscribe online at [www.codemag.com](http://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.

# BUILD BETTER APPS WITH **LEADTOOLS**<sup>®</sup>

```
viewer.js      dicom.net      PACS.cs      ocrdemo.py      X
1 def LoadandSaveOcrZones():
2     codecs = RasterCodecs()
3     ocrEngine = OcrEngineManager.CreateEngine(OcrEngineType.LEAD)
4     ocrEngine.Startup(codecs, None, None, r"C:\LEADTOOLS22\Bin\Common\OcrL")
5     zonesFile = r"PATH TO OCR ZONES FILE (OZF FILE)"
6     imageFile = r"PATH TO IMAGE TO CREATE OCR PAGE"
7     zonesOutFile = r"PATH TO SAVE OCR ZONES FILE (XML FILE)"
8     # Load the file as a RasterImage()
9     image = codecs.Load(imageFile, 1)
10    ocrPage = ocrEngine.CreatePage(image, OcrImageSharingMode(0))
11    print(ocrPage.Zones.Count, "Zones after (OcrPage creation.)")
12    ocrPage.LoadZones(zonesFile)
13    print(ocrPage.Zones.Count, "Zones after loading zones from file.")
14    for ocrZone in ocrPage.Zones:
```



Powered by patented artificial intelligence and machine-learning algorithms, LEADTOOLS SDKs help you create desktop, web, server, and mobile applications with the most advanced document, recognition, medical, and multimedia technologies, including:

OCR

DOCUMENT VIEWING & EDITING

BARCODE

PDF

IMAGE PROCESSING

DICOM

OFFICE FORMATS

FORMS RECOGNITION & PROCESSING

VIEWERS

+ MANY MORE

Get Started

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM





# Things Will Be Different

Evoking the theme of this issue, I started this editorial using ChatGPT. The prompt I fed it was: "Write an editorial in the style of Rod Paddock editor-in-chief of CODE Magazine. Discuss how containerized apps and microservices are not for every developer." The prompt I sent into ChatGPT generated a full editorial with an introduction that starts like this:

## **Containerized Apps and Microservices: Not a One-Size-Fits-All Solution**

*In the ever-evolving world of software development, containerized apps and microservices have garnered significant attention as modern solutions to streamline application deployment and scalability. The allure of these technologies is undeniable, but it is crucial to recognize that they are not the panacea for every development project. As Editor-in-Chief of CODE Magazine, I feel compelled to shed light on the potential pitfalls of blindly adopting containerization and microservices, urging developers to exercise caution and consider the specific needs of their projects before diving in."*

To be honest, the remainder of the editorial wasn't too bad. ChatGPT provided several good observations and created points of discussion written as though I'd written them myself. The only thing that was missing were references to Iron Maiden or Dungeons and Dragons. See what I did there?

The power of this tool is cool and scary at the same time. The world is changing and this new breakthrough in technology is a huge inflection point. The unbelievable thing about this is how rapidly it happened. The announcement of ChatGPT was November 30, 2022. In just over six months, the power of this technology has taken the world by storm.

I've been "noodling" on this editorial for a month. Numerous ideas came, only to be discarded for others as news and knowledge surrounding AI tech has evolved. It was just this last Saturday (June 10, 2023) that I came to realize how I wanted to talk about this new behemoth. All it took was a hangout day with my son Isaiah to make me realize how AI would change our lives in radical and unforeseen ways.

We started our trek at our awesome local bookstore. We gathered up various tomes on cooking, a book on the history of pop art stickers and a self-help book. What can I say, we have eclectic tastes. LOL. After raiding the bookstore, we headed across the street to our awesome local record store. Yes, Austin is just chock full of awesome.

As we proceeded to peruse the racks of CDs and albums, discussing the finer points of Drake's artistry, and commenting on various album covers, the discussion came around to purchasing some

CDs for his collection. Wait! What collection? In the year 2023, I'd have bet you that it's a rare thing for young people (I'll leave that definition up to you) to have CD collections. In the case of my children, that percentage would be 100%. Many people do have collections of music but they live in the cloud. This observation was a real eye opener. Things I take for granted are very different to today's generations.

As Isaiah and I continued our visit, I pondered just how different the world is for Isaiah and his sister. We then returned to the extensive racks of vinyl to see if we missed any interesting finds. This is when another observation hit me, the aisles of vinyl were...PACKED with fellow music aficionados. The vinyl section was mobbed with people searching for their own multicolored plaques of sonic joy. Keep in mind that vinyl records were all but dead just a generation ago. Now they are the rage. In a world full of digital annoyances, people have returned to the analog world of vinyl. What was old is new again. Things change. And sometimes they change by going backward.

A simple observation about young folk's lack of record collections leads to many other thoughts. What would be different with this tech? How will the semantic world of GPT merge with the syntactic worlds of software development? Who will this technology disrupt? What jobs will be created? The world of AI has a lot of potential for good and bad outcomes. This time, I believe we are more aware of the potential for negative outcomes and the discussions around them are happening daily.

As a developer, I think our world is going to be interesting. It's much easier to create software in 2023 than it has ever been. What this results in is the ability to create more interesting solutions to harder problems in less time. The perfect trifecta. Where AI will come in is the ability to build these even faster. Rather than searching for a code fragment or two on [www.stackoverflow.com](http://www.stackoverflow.com), you'll consult an AI bot that helps you gather potential solutions to your problems. The keyword in that sentence is **potential**. It will take a developer's reasoning and design skills to determine the applicability of the delivered code and whether or not to implement it in their code. AI has no judgement or reasoning—it's really a very effective mockingbird. It's up to you to make sure that what it's saying makes sense.

So, what does this all mean? In this author's humble opinion, AI is coming and soon it will be everywhere. AI in the future weeks, months, years, decades will be as big as the internet has been to us over the last three decades. Things will be different, and it's our duty to understand and use them carefully and considerately.

Rod Paddock  
**CODE**

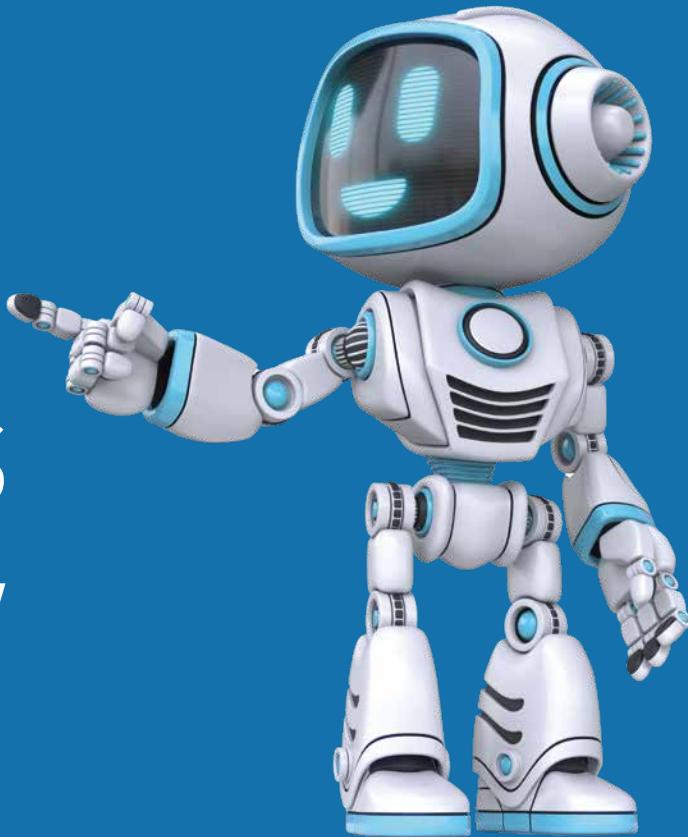
## Note from Melanie (editor of CODE Magazine)

I've already seen ChatGPT being used as a marketing tool and as an editorial tool, creating abstracts, looking for plagiarism, and helping with SEO. AI technology has been around as a writing platform to help with imposing style guide, grammar, and spelling rules for a few years (Writer.com, Acrolinx, Grammarly, etc.). I don't think the technology in its current form is headed toward eliminating the need for editors simply because the tool doesn't get it entirely right, especially when it comes to idioms and metaphorical expressions or analogies, and even for basic things like word or character count limitations. It's not even close. It's not ready yet to eliminate the need for writers either, but I can imagine that as a possibility coming soon.

I feel sad that people might not want to work with the words themselves, and extra sad that I won't get to work with truly brilliant writers much, as automation beats the deadlines better and cheaper. In light of the trend to look things up rather than collecting and memorizing information, I feel a little dread for the generation who grows up with readily available tools like this. If they don't learn things themselves, understanding the logic of why things work or what the probable outcome will be of some actions, how can they make good decisions? It's a problem much bigger than coding, but the apps we build will certainly be affected by the dumbing down of both creators and users. Don't get me wrong—I'm not against the tool. I can just imagine a world where people don't know how to use logic to help them get through life as we get increasingly dependent on computers to think for us.



# APPS WITHOUT COPILOTS ARE NOW LEGACY!



Microsoft has introduced Copilot-driven applications and announced that all Microsoft applications will follow this approach going forward. This aims to capitalize on the extreme productivity gains this new paradigm promises. The same applies to your own applications and systems.

We can help you build Copilots into your own applications.

**VISIT OUR COPILOT PAGE TO FIND OUT MORE ABOUT OUR SERVICES  
AND TO SEE DEMONSTRATIONS AND EXAMPLES!**

---

[codemag.com/copilot](http://codemag.com/copilot)

832-717-4445 ext. 9 • [info@codemag.com](mailto:info@codemag.com)

# Building Web APIs Using Node.js and Express: Part 2

In Part 1 of this article series (<https://codemag.com/Article/2305031/Building-Web-APIs-Using-Node.js-and-Express-Part-1>), you created a web server using Node.js, Express, and JavaScript. You built a series of API routes used to retrieve an array of product objects, search for a single object or a set of objects, and add, edit, and delete objects. In this article, you'll move some of the



**Paul D. Sheriff**

[www.pdsa.com](http://www.pdsa.com)  
psheriff@pdsa.com

Paul has been in the IT industry since 1984. In that time he has successfully assisted hundreds of companies 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 multiple courses in the [www.pluralsight.com](http://www.pluralsight.com) library (<https://bit.ly/3gvXgvj>) and on Udemy (<https://bit.ly/3WOK8kX>) on topics ranging from C#, LINQ, JavaScript, Angular, MVC, WPF, XML, jQuery and Bootstrap. Contact Paul at [psheriff@pdsa.com](mailto:psheriff@pdsa.com).



hard-coded values from the code into a configuration file and read those settings into your application. You're going to add middleware functions to handle exceptions. You're going to write different types of errors to different locations, such as to the console and to a file. Thus far, the set of product data coming from the API calls has been from a JSON file. In this article, you'll learn to retrieve the same data, but from a SQL Server table.

## Read Settings from a Configuration File

There are a few hard-coded strings in your Web API project from the last article that should be placed into a configuration file. There is an npm package called **config** that helps you read configuration data from a JSON file. If you were following along with the first article, open the **AdvWorksAPI** project you created in your favorite editor. If you weren't following along, download the project at <https://github.com/PaulDSheriff/Articles-CodeMagazine> and bring up that project in your favorite editor. Bring up a terminal window and install the **config** package into the AdvWorksAPI project using the following command.

```
npm install config
```

By default, the **config** package is expecting your project to have a **config** folder into which you place a JSON file. Create a **config** folder in your project and add a new file named **development.json**. You can use file names such as default, development, or production for your configuration file. For complete information on the **config** package, see the documentation at <https://www.npmjs.com/package/config>. Add the following JSON object into the **development.json** file.

```
{
  "host": "http://localhost",
  "port": 3000,
  "prefix": "/api"
}
```

Open the **index.js** file, locate the piece of code where you declare the **port** constant, and set it to 3000, as shown in the following code snippet.

```
// Specify the port to use for this server
const port = 3000;
```

Replace those two lines of code with the code shown in the following code snippet. This code loads the **config** module and retrieves the host, prefix, and port values from the configuration file using the **config.get()** method.

```
// Load the config module
const config = require('config');
// Get host server address
const host = config.get('host');
// Get prefix for all API calls
const prefix = config.get('prefix');
// Specify the port to use for this server
const port = config.get('port');
```

Now that you've eliminated the hard-coded port value, let's replace the other hard-coded string **/api**. Scroll down in the **index.js** file to locate that call and replace the string **/api** with the constant **prefix**, as shown in the following code snippet.

```
// Configure router so all routes
// are prefixed with /api
app.use(prefix, router);
```

The last value placed into the **development.json** file is the host name. This host name is only used in the message displayed within the **app.listen()** function. Replace the hard-coded value **http://localhost** within the **console.log()** with **`\${host}`**, as shown in the following code snippet.

```
let server = app.listen(port, function () {
  console.log(`AdvWorksAPI server is
running on ${host}:${port}.`);
});
```

## Try It Out

Save all the changes you have made to the project files. Start the **AdvWorksAPI** project using the command **npm start** in the terminal window. You should see the values from the configuration file appear in your terminal window. Open Postman or a browser and submit the GET query <http://localhost:3000/api/product> to ensure that the array of product data is returned from the route.

## Change the Name of the Configuration File

If you want to change which configuration file is read in, locate this line of code **const config = require('config')**, and **before** this line, add the following line of code that modifies the **NODE\_ENV** variable to the value **production**. This tells the **config** package to look for a file named **production.json** within the **config** folder.

```
// Set 'production' mode
process.env["NODE_ENV"] = "production";
```

Make a copy of the **development.json** file and change the name of the copied file to **production.json**. Modify all three values in the **production.json** file to some different values.

## Try It Out

Save all the changes you've made to the project files. Start the application and you should see the new values from the configuration file appear in your terminal window. After you've seen the new values, comment out the line of code you just added that changed the `NODE_ENV` value to `production`.

## Default Exception Handling

So far, you haven't handled any exceptions in your Web API calls. If you've been following along with the previous article, you most likely have encountered an exception or two when sending requests to your Web API. By default, Express returns an HTML error back to you. In case you haven't seen an exception yet, let's create one. Open the `repositories\product-file.js` file and modify the constant, as shown below, to use a `.json` file name that doesn't exist.

```
const DATA_FILE = './db/product-bad.json';
```

Save the changes made to all the files in your project and ensure that the project is still running. In Postman, submit the `GET` request `http://localhost:3000/api/product`. The response you receive from Express is an HTML body that looks like the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Error</title>
</head>
<body>
  <pre>Error: ENOENT: no such file or
  directory, open &#39;D:\Samples\
  AdvWorksAPI\db\product-bad.json&#39;;
  </pre>
</body>
</html>
```

## Add Exception Middleware

Express doesn't know if you're building a Web API application or a website application, so it defaults to returning HTML from unhandled exceptions. In every route you've created so far, you've seen the code `next(err)` in the error handling section. The function, `next()`, takes the error object and passes it to the middleware defined in Express that handles errors. This middleware takes that `err` object, extracts the error message, and wraps it within the HTML you saw above. Let's override this built-in Express exception middleware and eliminate the HTML being returned. Open the `index.js` file and add the following code just before the `app.listen()` function.

```
// Configure the last exception
// handler in the middleware chain
app.use(function (err, req, res, next) {
  res.status(500).send(err);
});
```

Express executes middleware in the order they're added in your `index.js`. Exception handling functions should always be the last middleware added to the chain prior to calling `app.listen()`. How do you know that the function shown

above is an exception handling piece of middleware? All exception handling functions have four parameters. The other middleware functions have only three parameters. In an exception handling function, the `err` parameter is defined first, followed by the other three standard parameters. The last exception handler listed before the `app.listen()` shouldn't have a call to `next(err)`. If you don't put a call to `next(err)` in your last exception handling function, the default Express exception handler never gets called.

The above final exception handler is very simple. If you are going to use your own handler, there are additional things you need to be aware of. Check out the Express documentation at <https://expressjs.com/en/guide/error-handling.html> for more information on best practices around exception handling.

## Try It Out

Save all the changes you made to the files in your project. Open Postman and submit the same request as last time. Remember, you still have the bad file name defined in your repository module. You should see that the raw `err` object (shown below) has been returned from your exception handling function. It's now expressed as a JSON object instead of HTML because the default exception handler in Express didn't get called.

```
{
  "errno": -4058,
  "code": "ENOENT",
  "syscall": "open",
  "path": "D:\\Samples\\AdvWorksAPI
    \\db\\product-bad.json"
}
```

## Build Your Own Error Object

If you were to set a breakpoint on the `res.status(500).send(err)` line, then debug the running Node application. You'll see that the `err` object contains a `stack` property that didn't show up in the error that was sent to Postman. This `stack` property might be useful in helping you debug that application. Sometimes the `message` and the `stack` properties are the same value, but you should grab both anyway. The `stack` property doesn't show up when running the application in production mode, nor would you want it to, as that could give away too much information about your server.

Let's add a new function to build an error object that contains the original error information and wraps it up into a JSON object. In addition to the error itself, add a few additional properties to this error object. Open the `index.js` file and add the function shown in the code below immediately after the `app.listen()` call.

```
// Create error object for logging
function buildError(err, req) {
  return {
    "status": 500,
    "statusText": 'Internal Server Error',
    "message": err.message,
    "stack" : err.stack ?? 'n/a',
    "originalError" : err,
    "requestInfo" : {
      "hostname": req.hostname ?? 'Unknown',
    }
  };
}
```

```

        "path": req.path ?? 'Unknown'
    }
};

}

```

Now, scroll back up in the index.js file and add the following code **before** the error handler you added previously.

```

// Configure a console exception logger
app.use(function(err, req, res, next) {
    // Build a better error object
    let errObject = buildError(err, req);
    // Log error information to the console
    console.log(errObject);
    // Call next error handler in the chain
    next(errObject);
});

```

This function calls the buildError() function to build the additional error information object. It then logs that error object to the console. You still call the *next(errObject)*, passing in this better error object. This JSON object gets passed to the final error handler you wrote previously.

**Listing 1:** The error returned shows a lot of information about the server.

```

{
  "status": 500,
  "statusText": "Internal Server Error",
  "message": "ENOENT: no such file or
  directory, open 'D:\\Samples\\AdvWorksAPI
  '\\db\\product-bad.json'",
  "stack": "Error: ENOENT: no such file or
  directory, open 'D:\\Samples\\AdvWorksAPI
  '\\db\\product-bad.json'",
  "originalError": {
    "errno": -4058,
    "code": "ENOENT",
    "syscall": "open",
    "path": "D:\\Samples\\AdvWorksAPI
  '\\db\\product-bad.json"
  },
  "requestInfo": {
    "hostname": "localhost",
    "path": "/api/product"
  }
}

```

**Listing 2:** Create an error handler module.

```

// Error Helper
let errorHelper = {};

// Build error object for internal logging
errorHelper.buildError = function (err, req) {
    return {
        "status": 500,
        "statusText": 'Internal Server Error',
        "message": err.message,
        "stack": err.stack ?? 'n/a',
        "originalError": err,
        "requestInfo": {
            "hostname": req.hostname ?? 'Unknown',
            "path": req.path ?? 'Unknown'
        }
    };
}

// Log error information to console
errorHelper.errorToConsole =
    function (err, req, res, next) {
        // Build error object for logging
        let errObject =

```

```

            errorHelper.buildError(err, req);
            // Log error information to the console
            console.error('Log Entry:
            ${JSON.stringify(errObject)}');
            console.error("*".repeat(80));
            // Pass error along to 'next' middleware
            next(err);
        }

        // Last function in error middleware chain.
        // Sends out the error object appropriate for
        // consumers of Web API calls.
        errorHelper.errorFinal =
            function (err, req, res, next) {
                res.status(500).send({
                    "status": 500,
                    "statusText": 'Internal Server Error',
                    "message": `An error has occurred,
                        please contact the system administrator.`
                });
            }
    }

module.exports = errorHelper;

```

## Try It Out

Save all the changes made to the files in your project. Open Postman and submit to the same request so you can generate the same error. You should now see the error message (**Listing 1**) appear in your VS Code terminal window and within the response body in Postman.

## Don't Expose Too Much Error Information

From a security standpoint, it's not a good thing to expose any information about your server to anyone. Thus, the **errObject** built in the buildError() method shouldn't be sent back to the caller. Instead, you should always send a generic message back to the caller that doesn't expose server information. You can log the **errObject** information to the console or into a file so you can look at it later. In the **index.js** file, modify the last exception handler in the chain to look like the following.

```

// Configure the last exception
// handler in the middleware chain
app.use(function (err, req, res, next) {
    res.status(500).send({
        "status": 500,
        "statusText": 'Internal Server Error',
        "message": `An error has occurred,
            please contact the system administrator.`
    });
});

```

## Try It Out

Save all the changes made to the files in your project. Open Postman and submit the same request to see the generic message appear. This is much better because now the caller doesn't see any path or file names on your server.

## Error Helper Module

As you would expect, it's a good idea to move these error handlers out of the index.js file and put them into their own module. Add a new folder named **helpers** to your project. Add a new file named **error.js** to the **helpers** folder and put the code from **Listing 2** into this new file.

### **Listing 3:** Create a log helper module to log errors to a file.

```
// Bring in node file system module
let fs = require('fs');

// Setup path/file name to store log info
const ERROR_FILE = './logs/error-log.txt';

// Log information to a file
let logToFileHelper = {};

// Write error information to log file
logToFileHelper.error =
  function (data, resolve, reject) {
    let toWrite =
      `${"*".repeat(80)}\nDate/Time: ${new Date().toJSON()}\nException Info: ${JSON.stringify(data)}\n${"*".repeat(80)}`;
    fs.appendFile(ERROR_FILE, toWrite,
      function (err) {
        if (err) {
          // ERROR: Invoke reject() callback
          reject(err);
        } else {
          // SUCCESS: Invoke resolve() callback
          resolve(true);
        }
      });
  };

module.exports = logToFileHelper;
```

In the **errorHelper** object, you create three functions: `buildError()`, `errorToConsole()`, and `errorFinal()`. The first function, `buildError()`, is the one you created previously at the bottom of the `index.js` file. The second function, `errorToConsole()` is like the other error handler you built that logs the error object to the console. The last function, `errorFinal()` is the last exception handler in the middleware chain because it doesn't call the `next()` function.

#### *Use the New Error Module*

Open the `index.js` file and add the code to load the new error module you created. Just after the line of code where you declare the port constant, add the following code.

```
// Load error helper module
const errorHelper = require('./helpers/error');
```

Scroll down in `index.js`, locate the previous two error handlers you added, and replace them with the following lines of code.

```
// Configure exception logger to console
app.use(errorHelper.errorToConsole);
// Configure final exception middleware
app.use(errorHelper.errorFinal);
```

Finally, **delete** the `buildError()` function you added at the end of the `index.js` file. Your `index.js` file is once again very clean and has less code in it, but still performs the same error handling.

#### *Try It Out*

Save all the changes made to the files in your project. Open Postman and submit the same query again. You should see the same error message as last time in the response body within Postman. Switch back to your editor and look at the terminal window and you should see the large error object created with the `buildError()` function.

## Log Exceptions to a File

Now that you can log your errors to the console, let's add a new middleware function to log the error object to a file. Create a new folder named `\logs` in your project. Create a new file in the `helpers` folder named `logToFile.js` and add the code shown in **Listing 3**.

The `logToFileHelper` object contains a single function named `error()` to which you pass some data to log. A string

### **Listing 4:** Add an additional middleware function to store error information to a file.

```
// Log error information to a file
errorHelper.errorToFile =
  function (err, req, res, next) {
    // Build error object for logging
    let errorObject =
      errorHelper.buildError(err, req);
    // Write error information to a file
    logToFileHelper.error(errorObject,
      function (data) {
        // SUCCESS: Information was written
        console.log(data);
      }, function (err) {
        // ERROR: Write error to console
        console.error(err);
      });
    // ERROR: pass error along to
    // the 'next' middleware
    next(err);
  }
```

is created with a series of asterisks, the current date/time, and the error object passed into this function. This string is written to a file named `error-log.txt` in the `logs` folder.

#### *Use Log Helper in Error Helper Module*

Open the `helpers\error.js` file and add the following code at the very top of the file.

```
// Load the log to file module
let logToFileHelper =
  require('../helpers/logToFile');
```

Add an additional function to the `errorHelper` object named `errorToFile()` as shown in **Listing 4**. The `err` object is passed to this middleware function and the large error object function is created by calling the `buildError()` function. This large error object is passed to the `error()` function of the `logToFileHelper` object and written to the log file. After a while, this file could get quite large, so it might be a better idea to use a logger package that truncates the file after so many days. There are several utilities you can use out there that will perform this functionality.

Open the `index.js` file and add the following lines of code in between the calls that configure the `errorToConsole` and `errorFinal` exception middleware:

```
// Configure exception logger to file
app.use(errorHelper.errorToFile);
```

### Try It Out

Save all the changes made to the files in your project. Open Postman and submit the same query again. You should see the same error message as last time in the response body within Postman. Switch back to your editor and look at the terminal window. You should see the large error object created in the buildError() function. Look in the **logs** folder and you should find the **error-log.txt** file. Open this file to see the error information. When you're done with testing your error handling, open the **product.js** file and change the file back to the correct file name, as shown in the following code snippet.

```
const DATA_FILE = './db/product.json';
```

## Create a SQL Server Database

Working with a flat file is fine for small sets of data that don't change, but for a production website, you should use a database. Let's learn how to connect your Web APIs to make a call to a SQL Server database. If you want to use Oracle, or a MySQL database, there are packages you can download for those database servers as well. The code I'm showing you for SQL Server should be similar even if using one of these other database servers. The first step is to install the **mssql** package into your Web API project by running the following command in your terminal window.

```
npm install mssql
```

You're going to write code in the rest of this article to search, insert, update, and delete data in a SQL Server database. So I'm not going to be covering every feature of the **mssql** package. If you want more information, check out the documentation for this package at <https://www.npmjs.com/package/mssql>.

### Create a Database

To follow along with this article, create a database using SQL Server or SQL Server Express. Create a database named **AdvWorksProducts** using the appropriate tool for your database. Once the database is created, create a Product table within this database using the following T-SQL command:

```
CREATE TABLE [dbo].[Product](
    [ProductID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
```

```
[ProductNumber] [nvarchar](25) NOT NULL,
[Color] [nvarchar](15) NULL,
[StandardCost] [money] NOT NULL,
[ListPrice] [money] NOT NULL,
[ModifiedDate] [datetime] NOT NULL
) ON [PRIMARY];
```

If you download the samples for this article, you'll find a folder within the zip file called **Database**. In this folder, there are two SQL files. The **AdvWorksProducts-schema.sql** contains scripts to build the Product table and a stored procedure. The **AdvWorksProducts-data.sql** file contains insert statements to load sample data into the Product table. The **AdvWorksProducts.zip** file also contains the detached **AdvWorksProducts.mdf** and **AdvWorksProducts.ldf** files that you can attach to a SQL Server version 2019 or later. The **AdvWorksProducts-SqlExpress.zip** file contains the detached **AdvWorksProducts.mdf** and **AdvWorksProducts.ldf** files that you can attach to an instance of SQL Server Express. The last file is called **product.csv** and contains the product data as comma-separated values if you want to import the data into the Product table.

## Create a SQL Server Repository Module

In the repositories folder, you have a **product-file.js** file that allows you to interact with the **db\product.json** file. Create a new file named **product-sqlServer.js** in the **repositories** folder in which you are going to write the code to interact with the Product table in SQL Server. Into the new **product-sqlServer.js** file, place the code shown in **Listing 5**. In the code in **Listing 5**, you're first going to learn how to work with SQL Server. You'll then learn to make this code more reusable.

This code first loads the **mssql** module so you can use the features of the **mssql** package to submit queries to your SQL Server database. You create a **repo** object in which there's a function called **get()**. This function first creates a connection string. The connection string might have some things that you're not used to seeing if you've used SQL Server from a .NET application.

I added a comma after the server name and added the port number on which SQL Server typically listens for a connection: **Server=localhost,1433**. The **encrypt=false** key/value pair is typically needed if your web server and

**Listing 5:** Create a repository object to read from SQL Server

```
// Load SQL Server module
let sqlServer = require('mssql');

// Product repository using SQL Server
let repo = {};

// Retrieve set of products from SQL Server
repo.get = async function (resolve, reject) {
    // Create a connection string
    let cnn = `Server=localhost,1433;
        Database=AdvWorksProducts;
        user id=sa;
        password=P@ssw0rd;
        encrypt=false;
        TrustServerCertificate=True;
        Application Name=AdvWorksAPI;`;

    // The SQL statement to submit
    let sql = `SELECT productID, name,
        productNumber, color, standardCost,
        listPrice, modifiedDate FROM dbo.Product
        ORDER BY name, listPrice`;

    try {
        // Connect to SQL Server
        await sqlServer.connect(cnn);
        // Submit the SQL query
        const result = await
            sqlServer.query(sql);
        // SUCCESS: Invoke resolve() callback
        // Return first recordset
        resolve(result.recordset);
    } catch (err) {
        // ERROR: Invoke reject() callback
        reject(err);
    }
}

module.exports = repo;
```

SQL Server are both on your local computer. This key/value pair instructs the connection to either encrypt traffic or not between the web server and the SQL Server. The **TrustedServerCertificate=True** key/value pair ensures that the connection accepts any certificate for making encrypted calls. These key/value pairs may not be needed, but I wanted to include them here so you can troubleshoot things by changing the values of these additional key/value pairs.

Notice that the SQL statement uses camel casing for the field names. Even though the fields are upper case in SQL Server, I'm deliberately changing the first letter of each column to a lower-case because that's how most JSON objects are written. If you're using Oracle, or any case-sensitive database engine, you may need to alias your columns to return them with the first letter lower-case.

Within a try...catch block is where you attempt to make a connection to the SQL Server using the **await sqlServer.connect(cnn)** call. Once the connection is made, submit the SELECT statement using the **await sqServer.query(sql)** call. When the data is returned, the object looks like the following:

```
{
  "recordset": [{product object}, ...]
  "recordsets": [
    [{product object}, ...]
    [,other recordset], ...
  ]
  "rowsAffected": [15 [, n], ...]
}
```

The **recordset** property is set to the results of the first SQL statement in the query. If there are multiple SQL statements, the **recordsets** property contains each resulting recordset. The **rowsAffected** property contains an array of the number of rows affected by each SQL statement.

The last thing to do is to change all the routes in the **routes\product.js** file to use the new **product-sqlServer** module instead of the **product-file** module. Open the **routes\product.js** and change the line of code that loads the **product-file.js** to the **product-sqlServer.js** file.

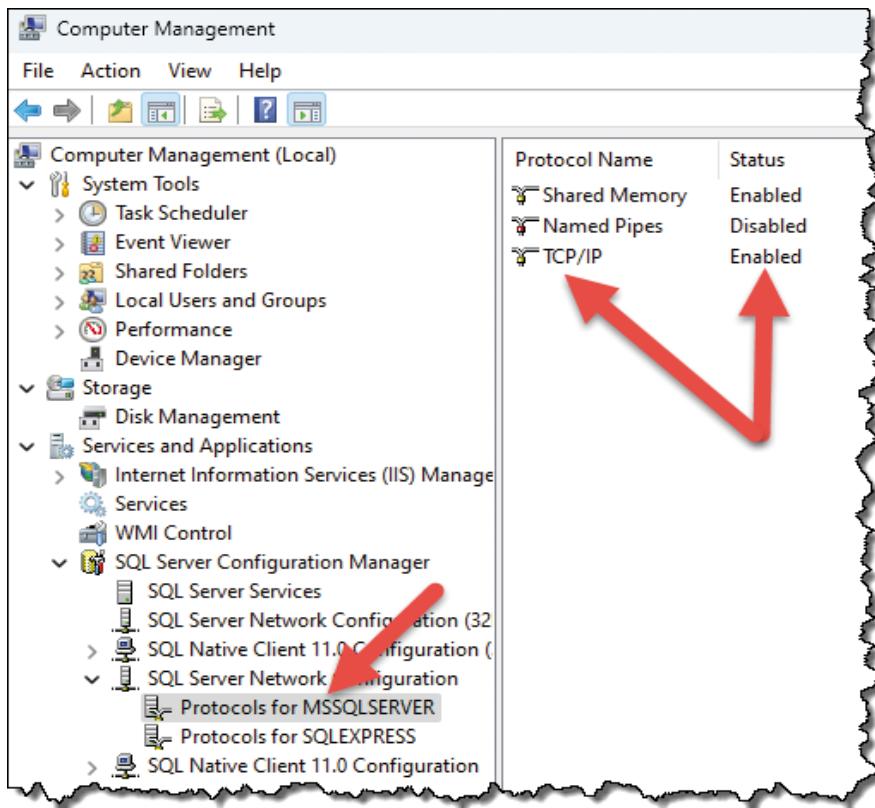
```
// Load product repository module
let repo = require(
  './repositories/product-sqlServer');
```

### Try It Out

Save all changes you made to the files in your project. Open Postman and submit the same **get all** query <http://localhost:3000/api/product>. If you put all the code in correctly, you should see the rows from the Product table appear in the response body.

### Did You Get an Error?

If you get an error message about not being able to make a connection, make sure TCP/IP is enabled for SQL Server on your computer. By default, it's usually turned off. Open the **Computer Management** app (**Figure 1**) on Windows and expand the SQL Server Configuration Manager node, then expand the Protocols for MSSQLSERVER (or Protocols for SQLEXPRESS if using SQLEXPRESS). Double-click on the **TCP/IP** protocol and in the dialog box, change the status to **Enabled**.



**Figure 1:** Turn on the TCP/IP protocol for SQL Server if you get errors when attempting to connect.

## Create a Reusable SQL Server Module

The code you just wrote in **Listing 5** is a simple example of how to make a call to SQL Server but it's not ideal. As you can imagine, you shouldn't duplicate this code for retrieving data from all your other tables. There are two major problems with this code: the connection string is hard-coded and the code within the try...catch block is repeated for each table.

Let's write a function that's more generic and can be used to retrieve data from any table such as customer, employee, or student tables. Create a SQL Server helper module with a generic function to submit any SQL Server query. In the **helpers** folder, create a new file named **sqlserver.js**. Add the code shown in **Listing 6** to this new file.

This code is similar to what you hard-coded in **Listing 5**, but retrieves the connection string from a configuration file and the SQL to submit is passed into the **submit()** function. There is also an optional parameter array that can be passed in. Parameters are built using the **request** object from the **mssql** module. A parameter object in the array has three properties: **name**, **type**, and **value**. The **name** is set to the name of the parameter included in the SQL statement. The **type** is the data type in SQL Server, and the **value** property is set to the value to pass to the server.

### Store the Connection String

To eliminate hard-coding the connection string, open the **development.json** file and add a **connectionString** property. Set the connection string to the one you used in your code earlier. Please note that you must put the connection string all on one line; I had to break the connec-

**Listing 6:** Create a helper module to submit queries to SQL Server.

```
// Load SQL Server Module
let sqlServer = require('mssql');

// Load the config module
const config = require('config');

// Get connection string from config file
let connectionString = config.get('ConnectionString');

// SQL Server Helper
let db = {};

/*
 * Submit SQL query with/without parameters
 * SELECT * FROM Product
 * [WHERE ProductID = @ProductID]
 * OPTIONAL: Pass array of parameter objects
 * { "name": 'ProductID',
 *   "type": sqlServer.Int,
 *   "value": 706 }
 */
db.submit = async function (sql, params, resolve, reject) {
  try {
    // Connect to SQL Server
    await sqlServer.connect(connectionString);
    // Create Request object
    let request = new sqlServer.Request();
    // Are there parameters to submit?
    if (params) {
      // Create parameters
      params.forEach(param => {
        request.input(param.name, param.type, param.value);
      });
    }
    // Submit SQL query
    const result = await request.query(sql);
    // SUCCESS: Invoke resolve() callback
    // Return recordset(s)
    resolve(result);
  } catch (err) {
    // ERROR: Invoke reject() callback
    reject(err);
  }
}
module.exports = db;
```

tion string across multiple lines for formatting purposes in this article.

```
{
  "host": "http://localhost",
  "port": 3000,
  "prefix": "/api",
  "ConnectionString": "Server=localhost;
  Database=AdvWorksProducts;
  user id=sa;password=P@ssw0rd;
  encrypt=false;
  TrustServerCertificate=True;
  Application Name=AdvWorksAPI;"}
```

#### Make the Product SQL Server Module Generic

Now that you have created a generic SQL Server module, you can significantly cut down the amount of code you wrote previously in the product-sqlServer.js file. Open the **product-sqlServer.js** file and replace the entire contents of this file with the code shown in **Listing 7**.

You still need to load the **mssql** package in this file because you're going to use it to build parameter data types when you write additional functions in this repository module. Load the **sqlserver.js** module you created so you can submit your SQL queries. The rest of the code should be self-explanatory.

#### Try It Out

Save all the changes you have made to the files in your project. Open Postman and submit the same query you've been using to ensure you're still retrieving all the product data from the SQL Server.

## Get a Single Product

Let's now add a function to the product repository module to retrieve a single row from the Product table. Open the **product-sqlServer.js** file and add a new function named **getById()**, as shown in **Listing 8**. In this function, create the SQL statement using a parameter **@ProductID**. This means that you need to create a parameter object in an array to pass to the **submit()** function. Set the **name** property to **ProductID** without the at (@) sign. The **type** property must match the type of the field in your SQL Server table. Finally, set the **value** property to the **id** passed in. Pass the SQL statement and the array of the single parameter object to the **submit()** function.

#### Try It Out

Save all change to the files you made in your project. Open Postman and submit a query to retrieve a single row of data, such as <http://localhost:3000/api/product/345>. Replace the number at the end of the query with a valid product ID from your table. You should see a single product object appear in the response body window. Try

**Listing 7:** Update product repository module to use the SQL Server module.

```
// Load SQL Server module
let sqlServer = require('mssql');

// Load SQL Server module
let db = require('../helpers/sqlserver');

// Product repository using SQL Server
let repo = {
  // Get all products from SQL Server
  get: function (resolve, reject) {
    // Create SQL statement to submit
    let sql = `SELECT productID, name,
    productNumber, color, standardCost,
    listPrice, modifiedDate
    FROM dbo.Product
    ORDER BY name, listPrice`;
```

```
// Submit SELECT query
db.submit(sql, null, function (data) {
  // SUCCESS: Invoke resolve() callback
  // Return data from first
  // recordset in the result set
  resolve(data.recordset);
}, function (err) {
  // ERROR: Invoke reject() callback
  reject(err);
});
}

module.exports = repo;
```

submitting an invalid product ID (<http://localhost:3000/api/product/1>) to ensure that you get a 404 response.

## Search for Products

Just like you searched for products in the JSON file, you should also create a search function to retrieve a set of products based on the criteria you pass in. Instead of putting a lot of SQL code within your JavaScript, I recommend creating a stored procedure in your database named **Product\_Search**, as shown in the following code snippet.

```
CREATE PROCEDURE dbo.Product_Search
    @Name nvarchar(50) null,
    @ListPrice money null
AS
BEGIN
    SELECT productID, name, productNumber,
        color, standardCost, listPrice,
        modifiedDate FROM dbo.Product
    WHERE (@Name IS NULL OR [Name]
        LIKE @Name + '%')
        AND (@ListPrice IS NULL OR
            ListPrice >= @ListPrice)
    ORDER BY [name], listPrice
END
```

This stored procedure allows you to pass a name (or partial name) and a list price. Both parameters can accept a null value. Each parameter is checked in the WHERE clause to see if it was passed in or not. If it was passed in, the value is used to filter on the data in the Product table. The **Name** parameter, if passed in, is used with a LIKE to allow you to pass in a value, such as **H**, and it finds all product names that start with the letter H. The **ListPrice** parameter, if passed in, is used to find all products where the list price is greater than or equal to the value passed in.

### Create the Search Function

Let's add a search() function to the product repository module. Open the **product-sqlServer.js** file and add the code shown in **Listing 9**. In this function, set the SQL statement to submit to the name of the stored procedure followed by the two parameters you're passing. Create an array of two parameter objects setting the values passed in via the **search** parameter into each of the parameter objects. Call the submit() function to submit the call to the stored procedure in your SQL Server.

### Try It Out

Save all the changes made to the files in your project and submit the following request via your browser or Postman.

```
http://localhost:3000/api/product/search?name=H
```

The above request should return a few records where the product name starts with the letter H. Submit the following request to look for both criteria.

```
http://localhost:3000/api/product/search?name=H
    &listPrice=1499
```

From this request, you should have a single product object returned. Next, submit the following request to check your error handling.

```
http://localhost:3000/api/product/search?name=z
```



# dtSearch®

## Instantly Search Terabytes

dtSearch's **document filters** support:

- popular file types
- emails with multilevel attachments
- a wide variety of databases
- web data

Over 25 search options including:

- efficient multithreaded search
- **easy multicolor hit-highlighting**
- forensics options like credit card search

Developers:

- SDKs for Windows, Linux, macOS
- Cross-platform APIs cover C++, Java and current .NET
- FAQs on faceted search, granular data classification, Azure, AWS and more

Visit [dtSearch.com](http://dtSearch.com) for

- hundreds of reviews and case studies
- fully-functional enterprise and developer evaluations

### The Smart Choice for Text Retrieval® since 1991

**dtSearch.com 1-800-IT-FINDS**

## Getting the Sample Code

You can download the sample code for this article by visiting [www.CODEMag.com](http://www.CODEMag.com) under the issue and article, or by visiting [www.pdsa.com/downloads](http://www.pdsa.com/downloads). Select "Articles" from the Category drop-down. Then select "Building Web APIs Using Node.js and Express – Part 2" from the Item drop-down.

From this request, you should get no records and thus a 404 status code and error object is returned from the query.

## Insert a Product

It's now time to start building the data modification methods in your product repository module. If you're submitting either an INSERT or UPDATE statement, you're going to have to create a set of parameter objects with each column in your Product table. If you're submitting an UPDATE statement, you also need to include the **ProductID** parameter for the WHERE clause. You don't need an ID for performing an insert because the Product table's primary key is set up as an IDENTITY and the ProductID will be

generated by SQL Server. Open the **repositories\productSqlServer.js** file and add the new function shown in **Listing 10** that's used to build the parameter array.

Next, add an insert() function (**Listing 11**) to create the INSERT statement with the list of columns and parameters. Call the createParams() function and pass in the product object passed to this insert() function. Pass a null value for the ID as you don't need an ID when performing an insert. Once you have the array of parameter objects, call the submit() function passing in the SQL statement and the parameter array. Typically, after submitting an INSERT, you want to get the row back that was inserted in SQL Server because you want to get back

**Listing 8:** Add a function to retrieve a single product object from SQL Server.

```
// Retrieve a single product
repo.getById =
  function (id, resolve, reject) {
    // Create SQL statement to submit
    let sql = `SELECT productID, name,
      productNumber, color, standardCost,
      listPrice, modifiedDate
    FROM dbo.Product
    WHERE ProductID = @ProductID`;

    // Create parameter array
    let params = [
      {
        "name": "ProductID",
        "type": sqlServer.Int,
        "value": id
      }
    ];
    // Submit SELECT query
    db.submit(sql, params, function (data) {
      // SUCCESS: Invoke resolve() callback
      if (data.recordset.length) {
        // Return product object from first
        // recordset in the result set
        resolve(data.recordset);
      } else {
        // Record not found
        resolve(undefined);
      }
    }, function (err) {
      // ERROR: Invoke reject() callback
      reject(err);
    });
  }
}
```

**Listing 9:** Add a function to search for one or many products in SQL Server.

```
// Search for one or many products
repo.search =
  function (search, resolve, reject) {
    if (search) {
      // Create SQL statement to submit
      let sql = `exec dbo.Product_Search
        @Name, @ListPrice`;

      // Create parameter array
      let params = [
        {
          "name": "Name",
          "type": sqlServer.NVarChar,
          "value": search.name
        },
        {
          "name": "ListPrice",
          "type": sqlServer.Decimal,
          "value": search.listPrice
        }
      ];
      // Submit SELECT query
      db.submit(sql, params, function (data) {
        // SUCCESS: Invoke resolve() callback
        // Return product data from first
        // recordset in the result set
        resolve(data.recordset);
      }, function (err) {
        // ERROR: Invoke reject() callback
        reject(err);
      });
    }
  }
}
```

**Listing 10:** Add a function to create an array of parameter objects.

```
// Create parameters for INSERT and UPDATE
repo.createParams = function (product, id) {
  // Create parameter array
  let params = [
    {
      "name": "Name",
      "type": sqlServer.NVarChar,
      "value": product.name
    },
    {
      "name": "ProductNumber",
      "type": sqlServer.NVarChar,
      "value": product.productNumber
    },
    {
      "name": "Color",
      "type": sqlServer.NVarChar,
      "value": product.color
    },
    {
      "name": "StandardCost",
      "type": sqlServer.Decimal,
      "value": product.standardCost
    },
    {
      "name": "ListPrice",
      "type": sqlServer.Decimal,
      "value": product.listPrice
    },
    {
      "name": "ModifiedDate",
      "type": sqlServer.DateTime,
      "value": new Date().toISOString()
    }
  ];
  if(id) {
    params.push({
      "name": "ProductID",
      "type": sqlServer.Int,
      "value": id
    });
  }
  return params;
}
```

**Listing 11:** Add a function to insert a product into SQL Server.

```
// Insert a product into SQL Server
repo.insert =
  function (newData, resolve, reject) {
    // Create SQL statement to submit
    let sql = `INSERT INTO dbo.Product
      (Name,ProductNumber,Color,
       StandardCost,ListPrice,ModifiedDate)
     VALUES
      (@Name,@ProductNumber,@Color,
       @StandardCost,@ListPrice,@ModifiedDate);
    SELECT productID, name, productNumber,
      color, standardCost, listPrice,
      modifiedDate FROM dbo.Product
     WHERE ProductID = SCOPE_IDENTITY();`;

    // Create parameter array
    let params = repo.createParams(newData, null);

    // Submit SELECT query
    db.submit(sql, params, function (data) {
      // SUCCESS: Invoke resolve() callback
      if (data.rowsAffected[0] == 1) {
        // Return product object from first
        // recordset in the result set
        resolve(data.recordset);
      }
      else {
        // Record not found
        resolve(undefined);
      }
    }, function (err) {
      // ERROR: Invoke reject() callback
      reject(err);
    });
  }
}
```

any generated field values, such as the ProductID, which is an IDENTITY property. Notice that in the `sql` variable you submit, there are two SQL statements: the first one is the INSERT statement and the second retrieves the newly inserted row by selecting immediately from the Product table using the `SCOPE_IDENTITY()` function to get the generated ProductID value.

### Try It Out

Save all the changes made to the files in your project. You now need to use Postman, or another tool, to be able to submit a POST with a JSON object within the body of the post back. Referring to **Figure 2**, follow the steps below to submit a new product object to be inserted into the Product table in SQL Server.

1. Select POST from the drop-down and fill in the URL to call your Web API.
2. Click on the Body tab.
3. Change the first drop-down to Raw.
4. Change the second drop-down to JSON.
5. In the body text box, add the following JSON object.

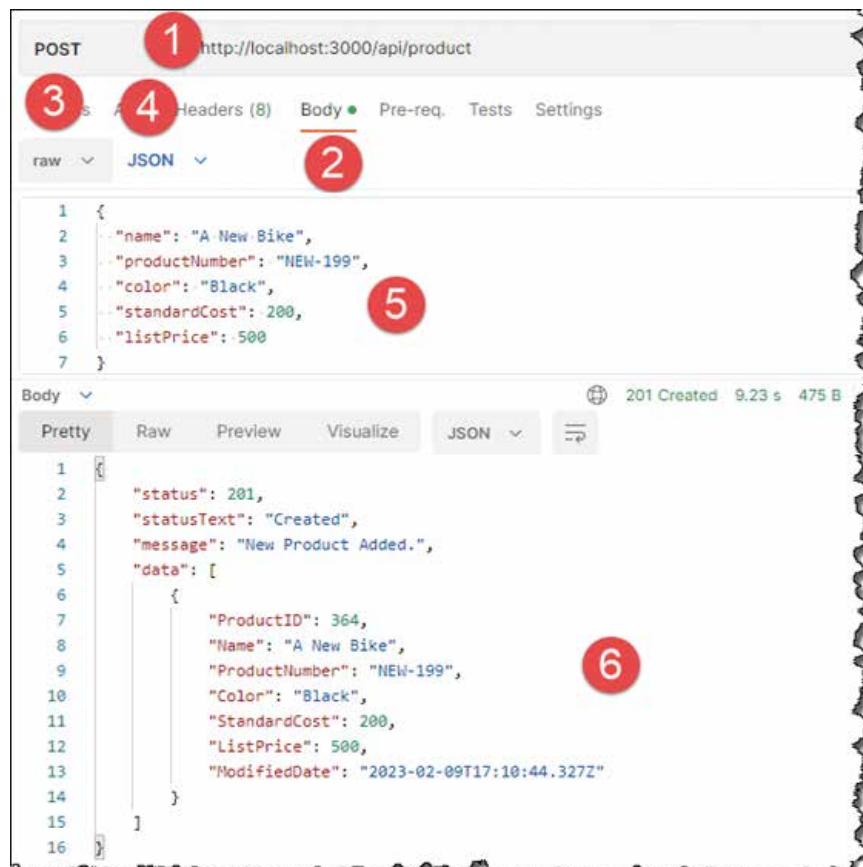
```
{
  "name": "A New Bike",
  "productNumber": "NEW-199",
  "color": "Black",
  "standardCost": 200,
  "listPrice": 500
}
```

6. Click the **Send** button to call the POST route you created. When the response returns, you should see the data in the lower window.

Open your SQL Server query tool and retrieve all rows in your Product table to see the new record you just inserted. Keep track of the new **ProductID** of your new product as you're going to need this for updating and deleting. Also take note of the **modifiedDate** property, as you're going to see that value change when you update this product in the next section of this article.

## Update a Product

The `update()` function is just like the `insert()` function in that you submit two SQL statements, the UPDATE and a SELECT, to retrieve the newly updated row. You call the `createParams()`



**Figure 2:** Use Postman to submit a POST to the Web API.

function to build the array of parameter objects, but you pass in the `id` of the product to update, so the extra parameter object is added to the array. The `update()` function does assume that you're submitting all of the columns in your table to update. Open the `repositories\product-sqlServer.js` file and add the `update()` function, as shown in **Listing 12**.

### Try It Out

Save all the changes made to the files in your project. Open Postman and submit a PUT with a JSON object within the body of the post back. All columns for your table should be included with this object. You'll want to replace the `productID` property with the value you recorded when you submitted the INSERT statement.

### **Listing 12:** Add a function to update a product into SQL Server.

```
// Update a product in SQL Server
repo.update = function (changedData, id, resolve, reject) {
  // Create SQL statement to submit
  let sql = `UPDATE dbo.Product
    SET Name = @Name
      ,ProductNumber = @ProductNumber
      ,Color = @Color
      ,StandardCost = @StandardCost
      ,ListPrice = @ListPrice
      ,ModifiedDate = @ModifiedDate
    WHERE ProductID = @ProductID;
  SELECT productID, name, productNumber,
    color, standardCost, listPrice,
    modifiedDate FROM dbo.Product
  WHERE ProductID = @ProductID;`;

  // Create parameter array
  let params = repo.createParams(changedData, id);

  // Submit SELECT query
  db.submit(sql, params, function (data) {
    // SUCCESS: Invoke resolve() callback
    if (data.rowsAffected[0] == 1) {
      // Return product object from first
      // recordset in the result set
      resolve(data.recordset);
    }
    else {
      // Record not found
      resolve(undefined);
    }
  }, function (err) {
    // ERROR: Invoke reject() callback
    reject(err);
  });
}
```

### **Listing 13:** Add a function to delete a product from SQL Server.

```
// Delete a product in SQL Server
repo.delete = function (id, resolve, reject) {
  // Create SQL statement to submit
  let sql = `DELETE FROM dbo.Product
    WHERE ProductID = @ProductID;`;

  // Create parameter array
  let params = [
    {
      "name": "ProductID",
      "type": sqlServer.Int,
      "value": id
    }
  ];

  // Submit SELECT query
  db.submit(sql, params, function (data) {
    // SUCCESS: Invoke resolve() callback
    if (data.rowsAffected[0] == 1) {
      // Return a numeric value
      resolve(1);
    }
    else {
      // Record not found
      resolve(undefined);
    }
  }, function (err) {
    // ERROR: Invoke reject() callback
    reject(err);
  });
}
```

```
{
  "productID": YOUR_VALUE_HERE,
  "name": "A CHANGED Bike",
  "productNumber": "NEW-199",
  "color": "Red",
  "standardCost": 300,
  "listPrice": 600
}
```

Modify the URL to add the product ID you wish to update at the end of the request, such as [http://localhost:3000/api/product/YOUR\\_VALUE\\_HERE](http://localhost:3000/api/product/YOUR_VALUE_HERE). Place the product ID returned from your insert at the end of the request. Click the **Send** button to submit the PUT request. You should see the updated product object returned in the response body of Postman. You should notice that the **modified-Date** property has a different date/time stamp than the one from the POST you submitted. If you want, open your SQL Server query tool and select all rows from your Product table to verify that the changes were made.

## Delete a Product

The last data modification functionality to add is to delete a product. Open the **repositories\product-sqlServer.js** file and add a **delete()** function, as shown in **Listing 13**. Pass in the primary key of the product to delete in the Product table and create the DELETE statement. The SQL should have a WHERE clause with a replaceable parameter for the ProductID field. Create a single parameter object in the params array that you pass to the **submit()** function. If the number of rows affected

comes back as a one (1), then you know the product was deleted successfully.

### Try It Out

Save all the changes made to the files in your project. Open Postman and submit the DELETE request [http://localhost:3000/api/product/YOUR\\_VALUE\\_HERE](http://localhost:3000/api/product/YOUR_VALUE_HERE) passing the product ID from your insert at the end of the URL line. Click the **Send** button and you should receive a **204 No Content** status code back from the request. There will be no content in the response body area, you should just see the 204 status code. If you want, open your SQL Server query tool, and select all rows from your Product table to see that the row has been deleted.

## Summary

In this article, you built upon the code from Part 1 of this article series. You added the ability to store hard-coded data into a configuration file and read that data from the file for use in your application. You took advantage of the middleware features of Node.js and Express to add exception handling that can log to both the console and to a file. Finally, you used the **mssql** package to connect to a SQL Server database. You then created functions to read, search, add, edit, and delete product data from a Product table. In the next article, you'll create an Express web server to build dynamic web pages. You're going to use these pages to make calls to the Web APIs you've created in these two articles to create a complete set of CRUD web pages.

Paul D. Sheriff  
**CODE**

# OpenAI

#We're living through interesting times. It's estimated that the human brain holds 2.5 petabytes of memory. Google Bard was trained in a dataset of around 1.56 petabytes. Yeah, we humans are still ahead, but consider that Bard took four months to learn all that, and Bard is far better than a human in making interconnections in so many ways. I use Bard as an example, but ChatGPT's

numbers are similar. Did you know that, since 2011, computer vision has been better than human vision in recognizing objects? Pair that with the ability of modern cameras and ultrasonic sensors to see in thick fog, across occlusions, in pouring down rain, and in ultraviolet and infrared spectrums. Can these technologies not one day help us to drive? After all, how do airplanes land in thick fog today?

It's not a surprise that almost every tech CEO can't seem to complete a sentence without the word "AI". In fact, at the latest Google keynote, the word AI was mentioned over a hundred times. I'm throwing all these numbers at you, but I just asked ChatGPT to tell me all this and I'm just repeating what it found.

I blogged about the societal impact and how we aren't ready for the sea change AI is about to bring into our lives (<https://winsmarts.com/thoughts-on-chatgpt-and-ai-a13d8a503638>), but I must admit, I approach it all with both caution and excitement.

In this article, I'll assume that you have no background in AI and explain where tech is from an OpenAI and Microsoft ecosystem perspective, and maybe even talk a bit about what the competitors are up to.

## A Brief History of OpenAI

Although it's tempting to dive into code, it's more important to understand the ecosystem and the path of how we got here. This is especially true for ground-breaking technologies, such as the printing press or AI. OpenAI isn't a technology. It's a research laboratory consisting of a nonprofit called OpenAI Incorporated and a for-profit subsidiary called OpenAI Limited Partnership.

It was started in 2015 by a number of powerful people and companies so the organization could freely collaborate and make its patents and research open to the public. The researchers worked with the best intent, gave up huge salaries at Silicon Valley tech firms, and focused on the mission. Over the years, they released and invented some amazing things.

As it happens with any such venture, once it becomes important, not only does it need more money, it also attracts opportunities to make money. So in 2019, OpenAI transitioned to "capped for profit," with the profit capped at 100 times any investment. This helped OpenAI attract investment and money attracts talent. Around the same time, Microsoft invested a billion dollars in OpenAI.

As time progressed, in 2020, OpenAI announced GPT-3, which stands for generative pre-trained transformer 3. Don't worry if it doesn't make a lot of sense yet, I'll explain all this momentarily. GPT-3 was something that, when given a prompt, generated human-like text. You

could converse with the computer in a number of natural languages.

In 2021, OpenAI introduced DALL-E, which was a deep learning model that could generate images from descriptions. You could describe anything, like a teddy bear riding a horse on a beach on Mars, and it generated an image for it.

In 2022, OpenAI released a free preview of ChatGPT 3.5, which received over a million sign ups in the first five days. People were absolutely amazed at the capabilities of this new chat-based interface.

In January 2023, Microsoft announced a \$10 billion investment in OpenAI, and then continued to integrate ChatGPT into the Bing search engine. Subsequently, it appeared in Edge, Microsoft 365, and other products, under the overall brand name of "Copilot." If you've tried GitHub Copilot, that's an example of a large language model that's been fine-tuned for code.

In March 2023, OpenAI released GPT-4.

The competition hasn't been sitting idle either. Google was clearly alarmed at this flurry of activity, and an internal document from Google (leaked to the public) was called "We have no moat and neither does OpenAI," <https://www.semianalysis.com/p/google-we-have-no-moat-and-neither>. Let's be honest, for the past 15 years, Google has been the gateway of information for all of us. This is an incredible power they've wielded; they can show us ads and charge money for them. Or they can suppress information they deem not worthy of our eyeballs. But with the advancements in the cloud and computing power, the availability of data, and many very intelligent minds at work, the moat has become about as relevant as castle walls once airplanes were invented.

Times have changed, and Google has risen to the challenge with their equivalent of ChatGPT called "Bard," and similar such products.

## A Crash Course in AI

When you read anything about AI, you're greeted by a number of terms. Generative AI? Machine learning? Deep learning? Reinforcement learning? What does all this mean? Let us understand some basics before moving forward.

First, **Artificial intelligence** (AI) is a branch of computer science that deals with the creation of intelligent agents, which are systems that can reason, learn, and act autonomously.

**Machine Learning** or ML is a subfield of AI. It gives computers the ability to learn without being explicitly programmed to do something. The thought is that if you show



**Sahil Malik**

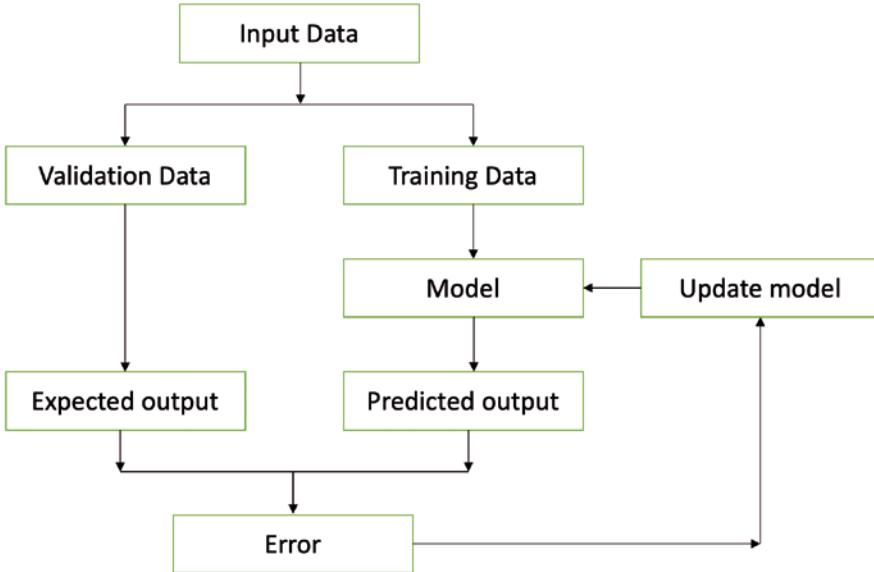
[www.winsmarts.com](http://www.winsmarts.com)  
@sahilmalik

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

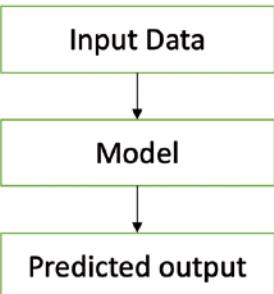
Sahil loves interacting with fellow geeks in real time. His talks and trainings are full of humor and practical nuggets.

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





**Figure 1:** Supervised learning



**Figure 2:** Unsupervised learning

the computer enough data, you have an algorithm, and based on that, the computer makes a prediction for data it's never seen before. The computer crunches a lot of data ahead of time, and creates a model. Based on that model, it makes predictions.

For example, I'm creating a model to identify a dog. If you've seen enough dogs, you know a dog is furry, has four legs, a tail, ears, usually acts like a stomach on legs. If I show you a Labrador, you know instinctively that it's a dog. If I show you a dachshund, you puzzle about how weird and long this dog is but you still know it's a dog. If I show you a Doberman and his tail is docked, you might pause, thinking that all dogs had tails. But you've seen enough dogs, and other animals, to know that this too, is a dog. Your model has seen a lot of data and it's a good model.

There are some common ML algorithms. Linear regression, for instance, is a simple algorithm that predicts a continuous value. A logistic regression can be used to predict a binary value and so forth.

ML can be split into two types: unsupervised ML models and supervised ML models. The main difference between the two is that the input data for supervised ML models is labelled. Let's take the example of dogs again. If I fed your model thousands of pictures, and clearly labelled what is a dog and what is not, the computer could use that input data to improve its understanding by validating it against the labels provided. In other words, in supervised learning, the computer learns from past learnings to predict future values. It takes a subset of the data and runs your algorithm against it to create a model. Then it uses the rest of the data to run against your model, and compares the predicted labels with the supplied labels. This gives you an error where your predictions didn't match the labels. Then you use that as input to tweak your model further until you minimize errors. This general flow can be seen in **Figure 1**.

A real-world example for **supervised** learning could perhaps be a bunch of biological markers from a person's

test results to predict whether or not a person has diabetes.

**Unsupervised** learning, on the other hand, is where the input data isn't labelled, and the computer is tasked with recognizing patterns or making sense of such unstructured data, as depicted in **Figure 2**. A good example of this could be clustering or detecting anomalies.

A real-world example for unsupervised learning could be to sift through login logs and detect anomalies. This helps you look for aberrations that you might not otherwise think of that hackers could exploit.

Another flavor of ML is RL or **reinforcement learning**. Here, the bot or agent you create makes predictions based on an algorithm or model. Call it trial and error. The agent is given feedback on what the right input should have been. In other words, there's a reward for the right output and a penalty for the wrong output. The agent is designed to continuously improve its model or algorithm to maximize rewards. A good example of this is training your dog, where successful execution of a command means a treat. Or, maybe the computer learns to play chess where the computer starts by being totally stupid, and as it loses, it learns not to make the same mistake again.

An even better example of reinforcement learning is self-driving, where the computer operates in shadow mode. It makes a prediction of what it might do under certain circumstances, but the human driver makes the actual moves. Those actual moves are compared constantly against the computer-predicted moves. If the computer's predictions match what the human would have done, it's considered a reward.

The real world is messy. What if the human was an awful driver? What if the computer could have driven safer than the human or had input the human can't see, such as radars, cognizance of six lanes around the car and four cars ahead and behind, data from other cars, that the human can't even see? Well, this is where you mix and match various approaches, along with the offline processing of data in the cloud.

Now you can see why Tesla ships autopilot hardware with every car. Whether or not you pay for autopilot, things still run in the background. The reinforcement data is uploaded into the cloud, and newer versions of autopilot software are constantly better than the last. This is mind-blowing stuff, really.

Speaking of the real world being messy, let's talk about **deep learning**, which is yet another important part of ML. Deep learning is a subset of machine learning that uses artificial neural networks to learn from data. Neurons are what make up the human nervous system. Artificial neural networks (ANNs) are inspired by our brains. Deep learning is made up of interconnected nodes called neurons. Each neuron receives input from other neurons and produces an output, and ANNs create layers upon layers of these neurons until the desired output is achieved.

These technologies are able to recognize complex patterns in data that would be hard for us to learn. And they do so using a combination of supervised and unsuper-

vised learning. When we talk of the interesting applications of AI, such as image recognition, natural language processing, speech recognition, self-driving cars, medical diagnosis, fraud detection, and so on, these are all examples of deep learning.

Finally, let's talk about **generative AI**, which seems to be all the rage these days. Generative is a subset of deep learning. Generative AI, as the name suggests, generates content. All the stuff OpenAI has made news about lately, DALL-E, ChatGPT, etc., are examples of Generative AI. When you compose an email in Gmail or Outlook, and Gmail or Outlook prompt you to complete an email as you type, that's an example of Generative AI. As you give a small prompt to a document based on some input and a larger final and polished document is created, that's an example of generative AI.

You know, that autocorrect on your touch screen phone? The one where you type and it predicts what the next

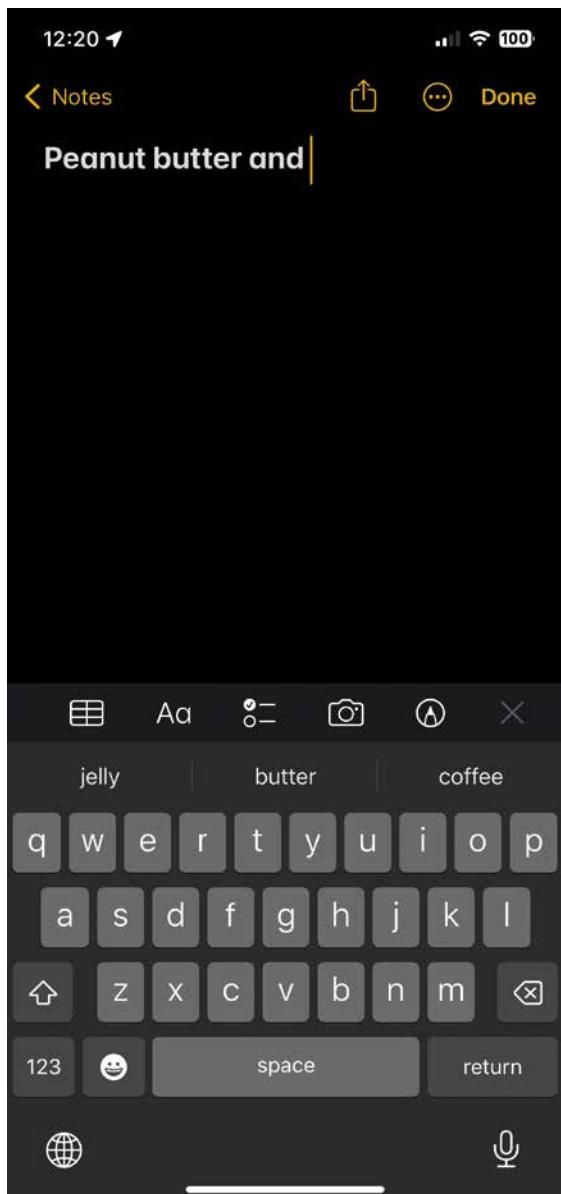


Figure 3: iPhone's predictive keyboard

word might be? That is also Generative AI in a rudimentary form.

Now we're getting somewhere. A great example of generative AI is large language models. Large language models, or LLMs for short, is an interesting field, which I feel will not only improve our touch-typing experience but also a lot more as well. Think about it. When you're typing "peanut butter and" and the automatically suggested word the phone shows is "jelly." How did your phone predict that the next word is "jelly"? A lot of analysis has been done on English text, and apparently peanut butter and jelly is a thing, although some of you also like to have peanut butter and butter, and peanut butter and coffee, as can be seen in **Figure 3**. The computer knows it and prompts you to complete it accordingly.

I retried this experiment on Samsung and Google keyboards and was disappointed to see behavior not much more advanced than iOS. What if I were typing "jelly" but instead typed **k** and **w** ("kwll"), which are characters right next to **j** and **e** on the keyboard? Why isn't the OS smart enough to guess that there's a high probability that I meant "jelly"? I distinctly remember Steve Jobs saying in one of Apple's keynotes that the keyboard is smart enough to judge such mistypes, but why don't I see it in action? Maybe it's too subtle.

## Large Language Models

Large language models are a specific type of generative AI. They're trained on massive amounts of text data, which allows them to learn statistical relationships between words and phrases, and they're then able to generate text that's similar to the text they were trained on. LLMs can then subsequently be used for interesting applications, such as text generation, where, given specific context, you can generate blog posts, compose emails, etc. Or they can be used for translating between languages. They can be used to summarize large articles. Or they can be used to build Q&A bots. Or, the latest craze, chatbots, like ChatGPT or Bard. These are all examples of LLMs.

Now, the topic of generating an LLM is far beyond the scope of this article, and frankly it can get quite academic. One day, I'd love to get into the guts of it, but for the moment, here's the best part. When you use OpenAI, the hard work is all already done for you. Let's see that next.

## OpenAI in Practice

So far, I've talked generally about the capabilities of AI and how we have reached where we are. As exciting as these concepts are, there are a few hurdles between us mere mortals and realizing the benefit of all this power for our applications. The hurdles are massive compute power, massive amounts of data, and well, huge brains on some really smart programmers.

All that stands between us and all this magic of AI is just some massive investment in compute, data, and huge brains. Lucky for us, OpenAI has done all of this. In fact, a subset of it is also exposed via Azure OpenAI. At the time of writing this article, Azure OpenAI is limited by invitation and private individuals, like me, can't use it. If you're part of a corporation, they will allow you to use it.

## SPONSORED SIDEBAR:

### Adding Copilots to Your Apps

The future is here now and you don't want to get left behind. Unlock the true potential of your software applications by adding Copilots. CODE Consulting can assess your applications and provide you with a roadmap for adding Copilot features and, optionally, assist you in adding them to your applications. Reach out to us today to get your application assessment scheduled.

[www.codemag.com/ai](http://www.codemag.com/ai)

The screenshot shows the OpenAI playground interface. At the top, there's a purple icon of a graduation cap and the text "Grammar correction". Below it are two buttons: "Transformation" and "Generation". On the right, a green button says "Open in Playground". The main area has a heading "Corrects sentences into standard English." Under "Prompt", there's a text input field containing "Correct this to standard English:" followed by the sentence "She no went to the market.". To the right, under "Settings", are the following parameters: Engine (text-davinci-003), Max tokens (60), Temperature (0), Top p (1.0), Frequency penalty (0.0), and Presence penalty (0.0). Under "Sample response", the AI-generated output is "She did not go to the market.". Below this, under "API request", is a code snippet in Python:

```
1 import os
2 import openai
3
4 openai.api_key = os.getenv("OPENAI_API_KEY")
5
6 response = openai.Completion.create(
7     model="text-davinci-003",
8     prompt="Correct this to standard English:\n\nShe no went",
9     temperature=0,
10    max_tokens=60,
11    top_p=1.0,
12    frequency_penalty=0.0,
13    presence_penalty=0.0
14 )
```

**Figure 4:** AI-based grammar police

But, as a private individual, I can only go to [www.openai.com](https://www.openai.com) and use the capabilities myself. On the other hand, Google has made some very amazing capabilities available in Google Cloud as well. I hope to write about them in a subsequent article. But for now, let's focus on OpenAI.

The first thing you need to do is visit [www.openai.com](https://www.openai.com) and click on the "Sign up" button on the page. Once you log in, you'll see three choices. Note that this is a highly evolving space so their website may have changed by now. The choices are ChatGPT, DALL-E, or OpenAPI. You can also use ChatGPT on Bing.com or use DALL-E on [bing.com/images/create](https://bing.com/images/create). You can also use Google's equivalent of ChatGPT at [bard.google.com](https://bard.google.com).

Of the three choices presented, when you click on **OpenAPI**, you'll see a few choices. You'll see a quick-start tutorial, some examples, and the ability to build ChatGPT plug-ins. I'll focus on the API portion.

At <https://platform.openai.com/examples>, you'll see a number of examples you can play with. Looking at the code, an example, sample input, and sample output is

free. To try things out for yourself, you're going to need a credit card.

On the top right-hand corner of the page, click on "upgrade," and upgrade to a paid account. Now, because I'm the cheapskate that I am, I went to [privacy.com](https://privacy.com) and set up a throw-away credit card number, and I also set up a usage limit that's pretty low. Hey, don't judge me.

Okay, now I'm ready to play! You can pick any sample code and build it easily. For example, if I click on the "Grammar Correction" example, it shows me how to build it, as shown in **Figure 4**.

In fact, you don't even need to write this code. You can just try it out in the playground. Very well done, OpenAI! As a sidenote, I noticed that OpenAI's log-in process uses Auth0, and their interactive documentation also feels very much like Auth0's documentation. Very well done OpenAI.

Let's build something fun and from scratch. Something that's not in the examples.

Ask question (or exit): What is Heisenberg's uncertainty principle?

...thinking...

Heisenberg's uncertainty principle states that the product of the uncertainties in position ( $\Delta x$ ) and momentum ( $\Delta p$ ) can never be smaller than a certain value, which is equal to Planck's constant ( $h$ ) divided by two ( $\Delta x \Delta p \geq h/2$ ). This tells us that if we know one of  $x$  or  $p$  very well, then we must be rather unsure of the other.

**Figure 5:** My AI exam cheat assistant

Let's imagine, through an anomaly in time and space, I find myself as a student of quantum mechanics and I need to learn everything this chapter tells me: [https://scholar.harvard.edu/files/david-morin/files/waves\\_quantum.pdf](https://scholar.harvard.edu/files/david-morin/files/waves_quantum.pdf) because tomorrow is exam day. I already feel like I'm time travelling back to college days. I seem to have spent the entire semester partying, and now, right before the exam, I'm just as unprepared and nervous as I used to be. But this time around, I have a secret weapon.

There are three main steps I need to do here.

**Note: To run this code example is going to cost money.  
Not a lot, but don't say I didn't warn you.**

First, I need to read up PDF as text and tokenize it. This text can be submitted to OpenAI APIs via the OpenAI SDK or as a simple REST call. I'll use the SDK.

The second thing to do is to create OpenAI embeddings. Embeddings are how OpenAI measures the relatedness of text strings. You can use it for various purposes, such as search, clustering, recommendations, anomaly detection, diversity measurement, and classification. It's essentially a vector of floating-point numbers, and the distance between two vectors measures their relatedness. Small distances mean high relatedness, and long distances mean low relatedness. Because this is a call to the API that's going to cost me money, I'll save the output locally.

The final step is to use this vector database and ask it questions in natural language. **Figure 5** shows what I am trying to achieve.

The first thing you need to do is go to OpenAI and generate an API key at <https://platform.openai.com/account/api-keys>. There are many ways to configure your project to use API keys, and I choose to create a config.ini in my Python project, which can be seen here:

```
[API_KEYS]
OPENAI_API_KEY = <key here>
```

Now in my Python code, I can choose to read the API key, as can be seen in **Listing 1**.

In my Python project, I'm going to take a dependency on a number of Python packages. These can be seen in my requirements.txt, as shown in **Listing 2**.

Go ahead and install these requirements in your project using the command below:

```
pip install -r requirements.txt
```

Next in my app.py file, I'll choose to add some imports. I'm going to use them at various steps, so let's get this out of the way. My imports can be seen in **Listing 3**.

The next step is to create embeddings. This is a two-step process. First, I need to read the PDF that I've saved locally and tokenize it. I'll use NLTK for tokenization and UnstructuredFileLoader from langchain.document\_loaders for this purpose. Then I'll use CharacterTextSplitter to break this apart into manageable chunks. The end result should be a variable called "text," which holds all the content I care about. Additionally, I don't want to do all this work multiple times. Once the embeddings are created, I want to save them and load them from the disk for the next execution. This first step of creating text chunks can be seen in **Listing 4**.

Once I have the text chunks, I can create embeddings. This step is going to cost me money, so I want to save the output locally. This can be seen in **Listing 5**.

#### **Listing 1: Get the OpenAPI key**

```
import os
import configparser

#
# Get the OpenAI API Key
config = configparser.ConfigParser()
config.read('config.ini')
os.environ["OPENAI_API_KEY"] =
    config.get('API_KEYS', 'OPENAI_API_KEY')
```

#### **Listing 2. My requirements.txt**

```
langchain
openai
tabulate
pdf2image
chromadb
tiktoken
```

#### **Listing 3: The imports my program depends on**

```
import nltk
import os
import configparser

from langchain.embeddings import OpenAIEMBEDDINGS
from langchain.vectorstores import Chroma
from langchain import OpenAI, PromptTemplate
from langchain.chains import RetrievalQA
from langchain.document_loaders import UnstructuredFileLoader
from langchain.text_splitter import CharacterTextSplitter
```

```
Ask question (or exit): What is the infinite square well?
```

```
...thinking...
```

```
The infinite square well is a quantum mechanical system in which a particle is confined to a region of space bounded by two infinitely high walls. The particle is free to move within this region, but is unable to escape. The energy levels of the particle are quantized, meaning that the particle can only have certain discrete energies. This system is often used to model the behavior of particles in a box, such as atoms in a solid.
```

Figure 6: Infinite square well

#### Listing 4: Create text chunks

```
vectorPath = 'vectordb'  
if not os.path.exists(vectorPath):  
    # Load PDF  
    nltk.download("punkt")  
    loader = UnstructuredFileLoader('waves_quantum.pdf')  
    documents = loader.load()  
  
    # Create chunks  
    text_splitter = CharacterTextSplitter(chunk_size=800,  
                                         chunk_overlap=0)  
    text = text_splitter.split_documents(documents)
```

Now that the embeddings are available as a local file, I can simply load them from the disk for subsequent runs and create a vectordb out of it. This can be seen in Listing 6.

Finally, I can start asking questions. This step also costs money, but there's no way around it. I've written a simple loop that allows the user to ask questions endlessly until the user types "exit." This can be seen in Listing 7.

That's it. Go ahead and run it and ask another difficult question. This time I asked, "What is an infinite square well?" The output can be seen in Figure 6. I already feel so smart.

#### Listing 5: Create embeddings

```
embeddings = OpenAIEmbeddings(  
    openai_api_key=os.environ["OPENAI_API_KEY"])  
Chroma.from_documents(text, embeddings,  
                      persist_directory=vectorPath)
```

This code example took me about 30 minutes to write. I ran it five or six times, and I want to share with you how much it cost me to do it. You can check your current usage at <https://platform.openai.com/account/usage>. I have shared my usage in Figure 7, and it looks like I spent six cents. I'll file an expense report.

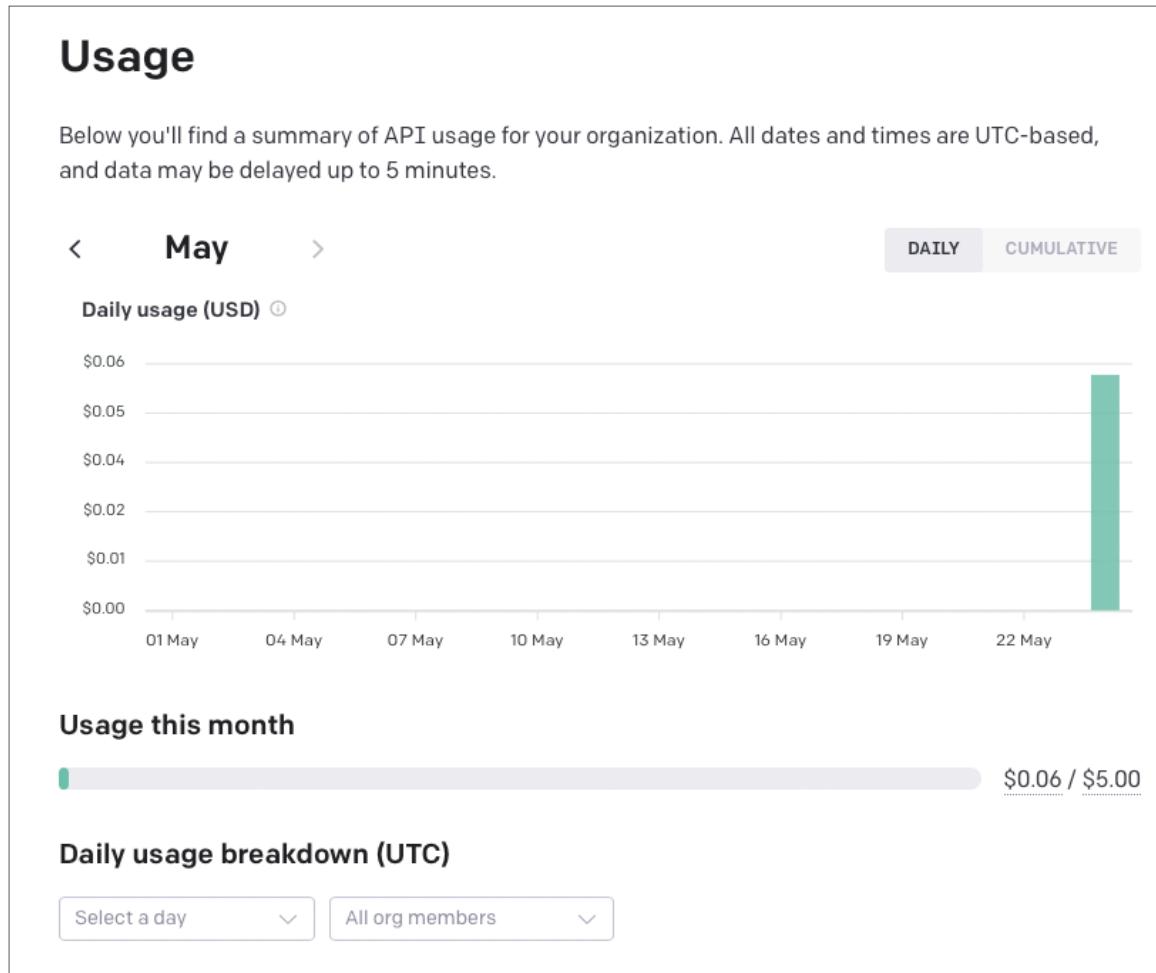


Figure 7: Cost to run this example

Jokes aside, there are a few things to keep in mind. These capabilities are impressive, but given the costs and limits on what I can consume, I wouldn't call this production-ready just yet. Also, there's a dependency on the cloud and personal identifying information (PII) concerns. Azure and Microsoft are working hard to fix all this, but they aren't quite there yet. Still, it's good to see what's possible and the pace of innovation here is quite furious. I'm sure, in almost no time, you'll be consuming OpenAI across your applications.

## Summary

AI feels like a ground-breaking change, like the invention of the cell phone, personal computer, internal combustion engine, aviation, printing press, fire, etc. Is this hyperbole? Honestly, I don't think so. The applications of AI are only beginning to be apparent. As we move forward, not only will the capabilities of AI improve by leaps and bounds, but we'll start applying our creativity to solve problems with AI as well.

Pushing the boundaries of AI may remain the domain of large companies with deep pockets, but there are many LLMs that can be trained at the cost of a few hundred dollars or locally on your modern laptop. They aren't as good, but maybe they're good enough.

There will be many to options pick from. I certainly picked the right career because now, many years in, I'm yet to be bored in this line of work. For the very first time, I'm a bit scared of what this means for our society and our future.

How will you differentiate a real image from a fake in the upcoming election? How will you differentiate a real voice in a phone call from a loved one, vs. a computer-generated voice? As humans get displaced by computers learning so much faster than people, what will humans who simply cannot learn at the pace of computers do? Will the tools of AI be used for a distributed equitable benefit for the society, or will it fuel multi-million-dollar salaries for Silicon Valley elites?

There will be interesting times ahead. Until next time, let's keep learning.

Sahil Malik  
**CODE**

### **Listing 6:** Load local vectordb

```
embeddings = OpenAIEmbeddings()
vectordb = Chroma(persist_directory=vectordbPath,
                  embedding_function=embeddings)
```

### **Listing 7:** Ask questions

```
while True:
    user_input = input(f"\nAsk question (or exit): ")
    if user_input.lower() == 'exit':
        break
    else:
        prompt_template = """
            Context: {context} \n
            Question: {question} \n
            Answer:"""
        prompt = PromptTemplate(
            template=prompt_template,
            input_variables=["context", "question"]
        )
        chain_type_kwargs = {"prompt": prompt}
        retriever = vectordb.as_retriever(
            search_type="similarity",
            search_kwargs={"k": 2})
        qa_chain = RetrievalQA.from_chain_type(
            OpenAI(temperature=0, max_tokens=400),
            chain_type="stuff",
            retriever=retriever,
            chain_type_kwargs=chain_type_kwargs)
        print('...thinking...')
        response = qa_chain.run(user_input)
        print(response, "\n")
```

## ADVERTISERS INDEX

### Advertisers Index

CODE Consulting	<a href="http://www.codemag.com/ai-services">www.codemag.com/ai-services</a>	2
CODE Consulting	<a href="http://www.codemag.com/copilot">www.codemag.com/copilot</a>	7
CODE Consulting	<a href="http://www.codemag.com/Code">www.codemag.com/Code</a>	75
CODE Consulting	<a href="http://www.codemag.com/executivebriefing">www.codemag.com/executivebriefing</a>	76
DeveloperWeek	<a href="http://www.developerweek.com">www.developerweek.com</a>	53
DevIntersection	<a href="http://www.devintersection.com">www.devintersection.com</a>	29
dtSearch	<a href="http://www.dtSearch.com">www.dtSearch.com</a>	15
LEAD Technologies	<a href="http://www.leadtools.com">www.leadtools.com</a>	5
MS Power Platform Conf.	<a href="http://www.powerplatformconf.com">www.powerplatformconf.com</a>	37
UAV Expo	<a href="http://www.expouav.com">www.expouav.com</a>	69



**Advertising Sales:**  
Tammy Ferguson  
832-717-4445 ext 26  
tammy@codemag.com

This listing is provided as a courtesy to our readers and advertisers. The publisher assumes no responsibility for errors or omissions.

# An Introduction to OpenAI Services

Unless you've been living under a rock for the past couple of months, you've heard of the chatbot called ChatGPT. ChatGPT is an artificial intelligence chatbot developed by OpenAI and released in November 2022. Since then, it's taken the world by storm. Not a day passes without someone talking about it on social media, and numerous companies sprang up overnight to take



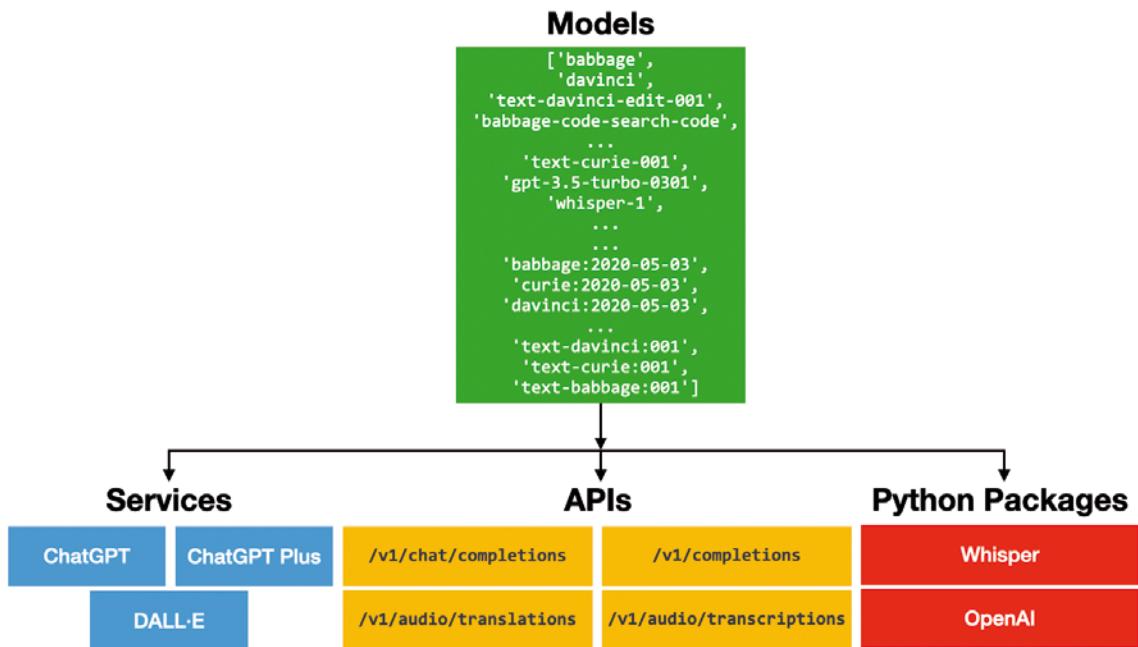
**Wei-Meng Lee**

weimenglee@learn2develop.net  
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 experiences 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.





**Figure 1:** OpenAI pre-trained models and the services it provides

advantage of the capabilities offered by ChatGPT. In this article, I'll talk about some of the powerful and exciting models offered by OpenAI. And of course, as this is a developer's magazine, my focus will be mostly on what developers can do with them. This is by no means an exhaustive view of what OpenAI offers (in fact I'm confident that things will have changed by the time this article is published), but I want to give you a head-start in getting started with artificial intelligence.

**OpenAI is an American artificial intelligence (AI) research laboratory consisting of the non-profit OpenAI Incorporated and its for-profit subsidiary corporation, OpenAI Limited Partnership.**

According to figures from research firm SemiAnalysis, OpenAI has spent an eye-watering \$700,000 a day to run the servers powering ChatGPT. This amount is expected to be much higher today as this figure was based on the older GPT-3 model. At the time of writing, the latest ChatGPT is powered by GPT-4, which is going to cost much more to run.

Microsoft (one of the main investors in OpenAI) is the exclusive provider of the computing power of OpenAI, which runs on the Microsoft Azure platform.

Another service that's similar to DALL E is Stable Diffusion, which is an open-source, machine learning model developed by Stability AI. Unlike DALL E, Stable Diffusion can be accessed through various websites. You can also access it locally through the command line.

## ChatGPT and Large Language Models

The GPT in ChatGPT gets its name from **Generative Pre-trained Transformer**. GPT is a family of language models known as **large language models**. Large language models (LLMs) are deep learning models that are trained using vast amounts of text data, such as books, articles, websites, and more. LLMs are able to process and generate human-like responses, and can perform sophisticated tasks, such as language translation, generating new content, generating code, debugging code, providing suggestions and recommendations, and much more. LLMs are often used in natural language processing (NLP) tasks, such as chatbots and virtual assistants, as they are able to hold intelligent conversations with human beings.

For this article, I won't be going into the details of how LLMs work. Rather, I want to focus on what OpenAI offers. In particular, how you can make use of OpenAI from the perspective as both a user and a developer. For the purpose of this article, **Figure 1** shows that behind all the services, APIs, and packages provided by OpenAI, it's essentially a set of pre-trained models. These models are what powers all the various services that users can use (such as ChatGPT). Some of the models are open-sourced (such as GPT-2) and some are not (such as GPT-4).

I will discuss how you can make use of OpenAI's services through:

- **Services:** This is for end-users who want to make use of AI to do their daily work.
- **APIs:** This is for developers who want to integrate OpenAI's services into their applications.
- **Python packages:** This is specifically for Python developers to use OpenAI's services.

## Using OpenAI Services

Now that you have a basic idea of the different parts in the OpenAI ecosystems, let's start with the services provided by OpenAI to the general public.

## ChatGPT

ChatGPT is the service that took the world by storm. Launched as a prototype November 30, 2022, it garnered a lot of attention for its detailed responses and ability to provide the answers to questions across many domains of knowledge. You can ask ChatGPT to:

- Write code for you
- Debug a block of code
- Provide advice

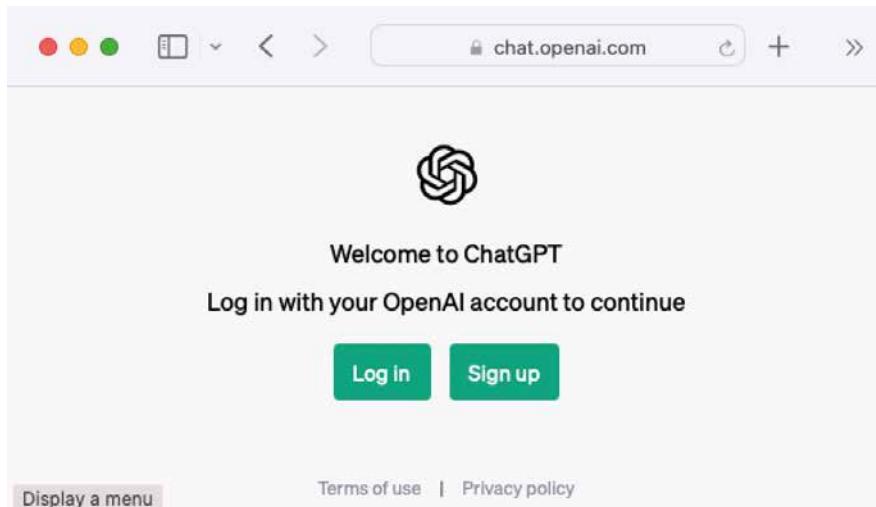


Figure 2: You need to sign up for ChatGPT before using it.

- Explain things
- Write a book summary
- Create a course outline and content

And much more!

To use ChatGPT, go to <https://chat.openai.com> (see Figure 2). You need to sign up using a Google or Microsoft account.

Despite ChatGPT's ability to engage in conversations and provide answers to questions, it's important to do your own fact checks because ChatGPT has been known to provide factually incorrect responses.

Once you've signed up, you can start asking ChatGPT some questions (see Figure 3).

For example, I can ask the following question (called a **prompt**): "Can you write me the regular expression to validate an email address?" ChatGPT responds with the regular expression (see Figure 4).

You can continue with the conversation by asking "Can you show me an example using Python?" ChatGPT remem-

A screenshot of the ChatGPT interface. The top navigation bar shows "chat.openai.com". On the left, there's a sidebar with a "New chat" button. The main area has a title "ChatGPT" and three sections: "Examples", "Capabilities", and "Limitations". Under "Examples", there are three items: "Explain quantum computing in simple terms", "Got any creative ideas for a 10 year old's birthday?", and "How do I make an HTTP request in Javascript?". Under "Capabilities", there are two items: "Remembers what user said earlier in the conversation" and "Allows user to provide follow-up corrections". Under "Limitations", there are two items: "May occasionally generate incorrect information" and "Limited knowledge of world and events after 2021". At the bottom, there's a message input field with "Send a message." and a note: "Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT Mar 23 Version". The footer includes "Upgrade to Plus", "Wei-Meng Lee", and "Display a menu".

Figure 3: You can ask ChatGPT anything!

# The Intersection of Technology

DEVintersection  
Conference

[DEVintersection.com](http://DEVintersection.com)

Microsoft Azure  
+AI Conference

[AzureAIConf.com](http://AzureAIConf.com)

Azure Data  
Conference

[AzureDataConf.com](http://AzureDataConf.com)

**DEC 5-7  
2023  
WORKSHOPS  
DEC 3, 4 & 8**

**WALT DISNEY  
WORLD  
SWAN &  
DOLPHIN  
RESORT**

Daily Keynotes by Microsoft Senior Leaders | 150+ Breakout Sessions | 18+ Full Day Workshops



**ARUN  
ULAGARATCHAGAN**

Corporate Vice President,  
Azure Data, Microsoft



**JESSICA  
HAWK**

Corporate Vice President,  
Data, AI & Mixed Reality,  
Microsoft



**ERIC  
BOYD**

Corporate Vice President, AI  
Platform, Microsoft



**SCOTT  
HANSELMAN**

Partner Program Manager,  
Microsoft



**SCOTT  
HUNTER**

Vice President, Director  
Program Management,  
Azure Developer Experience,  
Microsoft



[DEVintersection.com](http://DEVintersection.com)  
 @Devintersection

[AzureAIConf.com](http://AzureAIConf.com)  
 @AzureAIConf

[AzureDataConf.com](http://AzureDataConf.com)  
 @AzureDataConf

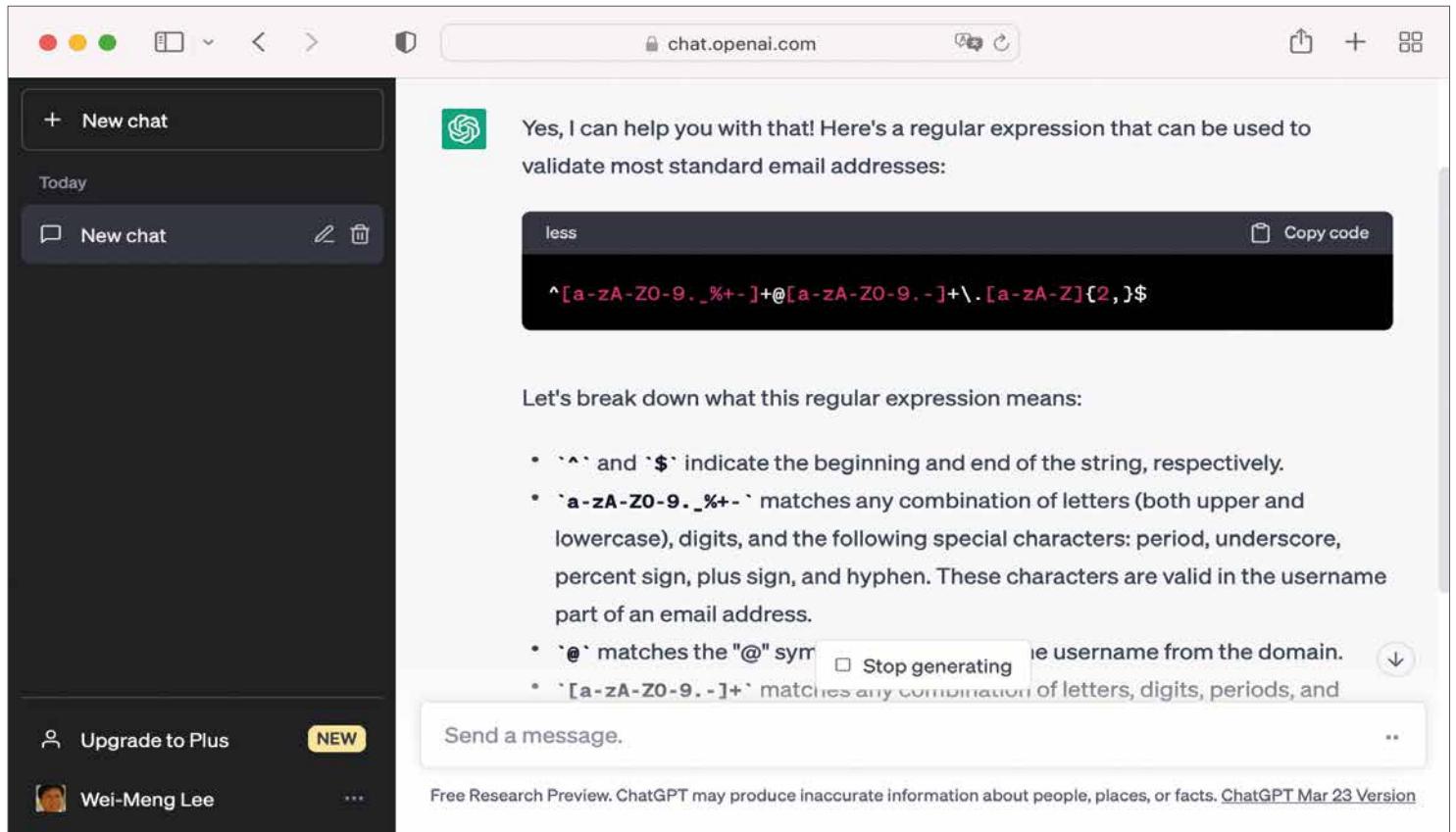


Figure 4: ChatGPT coming up with the regular expression to validate an email address

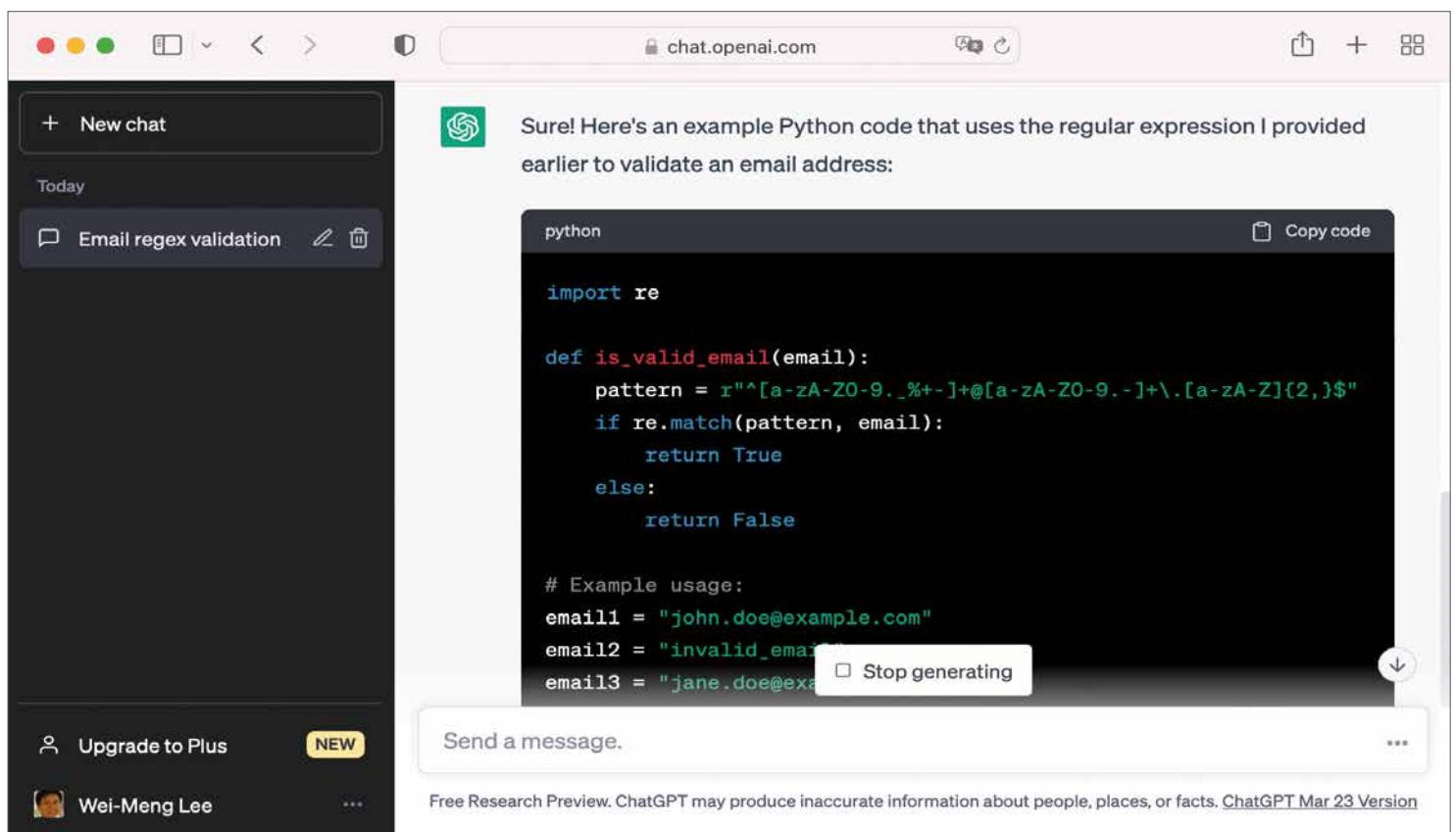


Figure 5: ChatGPT is able to remember your previous question and continue the conversation.

bers your previous question and provides you with sample code in Python (see **Figure 5**).

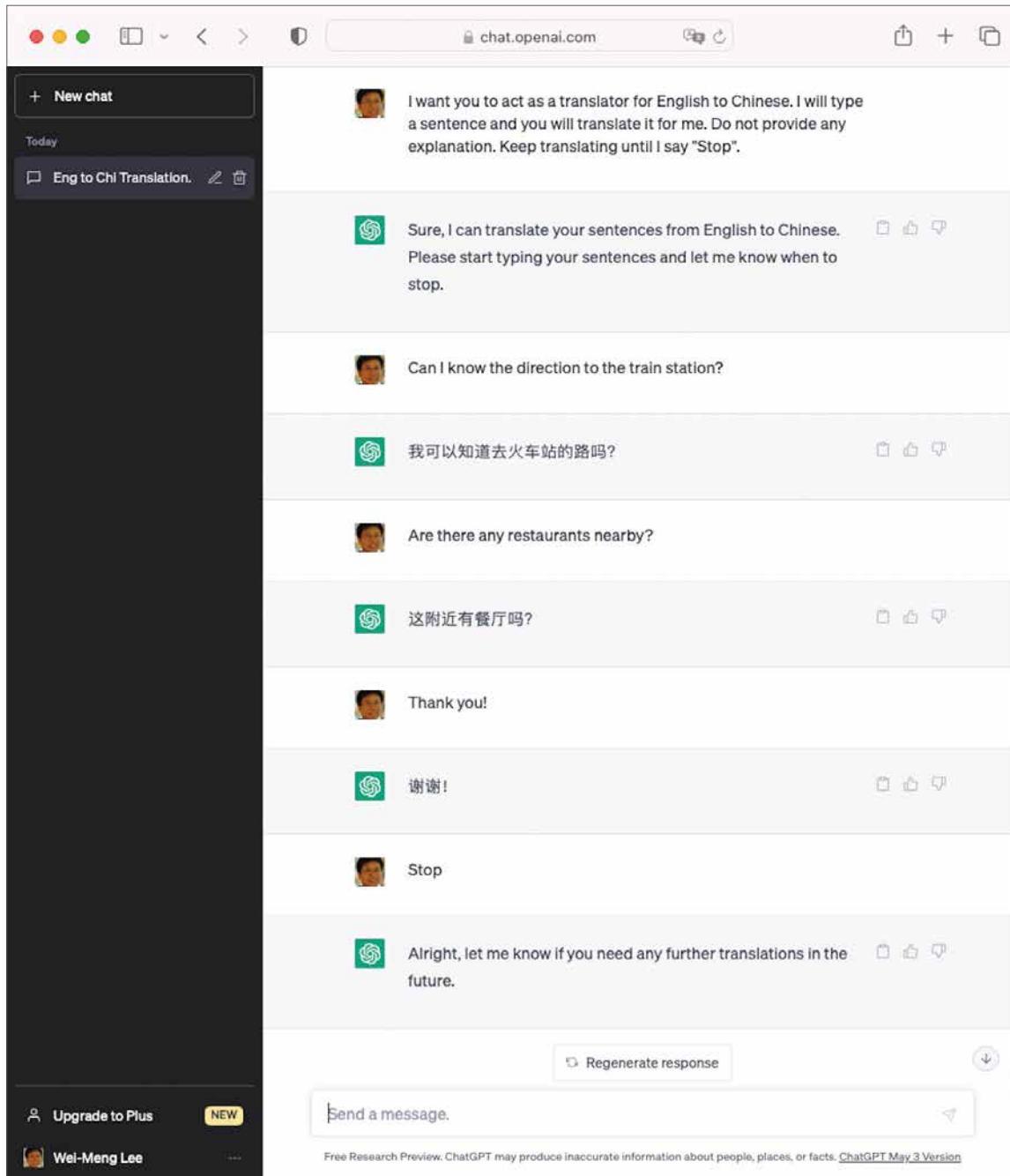
One awesome prompt is to ask ChatGPT to act as a translator. Let's say I want to translate English into Chinese. Instead of explicitly asking ChatGPT to translate each individual sentence I want to translate, I can simply use the following prompt:

"I want you to act as a translator for English to Chinese. I will type a sentence and you will translate it for me. Do not provide any explanation. Keep translating until I say 'Stop.'"

**Figure 6** shows the conversation I had with ChatGPT. For every sentence I typed in, ChatGPT straight-away translates it into Chinese until I say "Stop."

For more interesting prompts that you can use with ChatGPT, check out <https://github.com/f/awesome-chatgpt-prompts/>.

**At the time of this writing, ChatGPT Plus is available for a monthly fee of \$20. With ChatGPT Plus, you get faster responses, priority access to new features, and the service is always available even when the demand is high.**



**Figure 6:** Prompts are special instructions to make ChatGPT behave the way you want it to.

Another service that's similar to DALL-E is Stable Diffusion, which is an open-source, machine learning model developed by Stability AI. Unlike DALL-E, Stable Diffusion can be accessed through various websites. You can also access it locally through the command line.

### Using DALL-E 2 to Generate Images

DALL-E 2 is a neural network-based image generation system developed by OpenAI. Based on GPT, it's trained on massive amounts of images and text data. The training set consists of image-text pairs, where each image has an accompanying textual description of the image. The system then learns how to generate new images based on the data that it learned from the training set.

To try out DALL-E 2, go to <https://labs.openai.com> (see Figure 7).

To generate an image, give it a description (be as detailed as you can). Figure 8 shows the images that it generated for me when I asked DALL-E 2 to create a picture of me in front of a Mercedes.

## Using OpenAI APIs

Because this is a developer magazine, it won't do justice to you if I don't talk about interacting with OpenAI programmatically. So, for this section, I'll discuss how you can integrate OpenAI APIs into your own application.

OpenAI provides APIs for developers to access OpenAI's services. Using them, you can perform a number of tasks, such as:

- Listing the models available for use
- Creating chats
- Creating edits to prompts
- Generating images
- Creating variations of images
- Creating embeddings of a given input
- Creating transcription of audio

Check out <https://platform.openai.com/docs/api-reference/> for more details on OpenAI's API offerings.

### Accessing Through HTTP

In the following sections, I'll show you a few examples of how to use the OpenAI API using HTTP. To use the APIs, you need to first apply for an OpenAI API key. Head over to <https://platform.openai.com/account/api-keys> and click the **Create new secret key** button (see Figure 9).

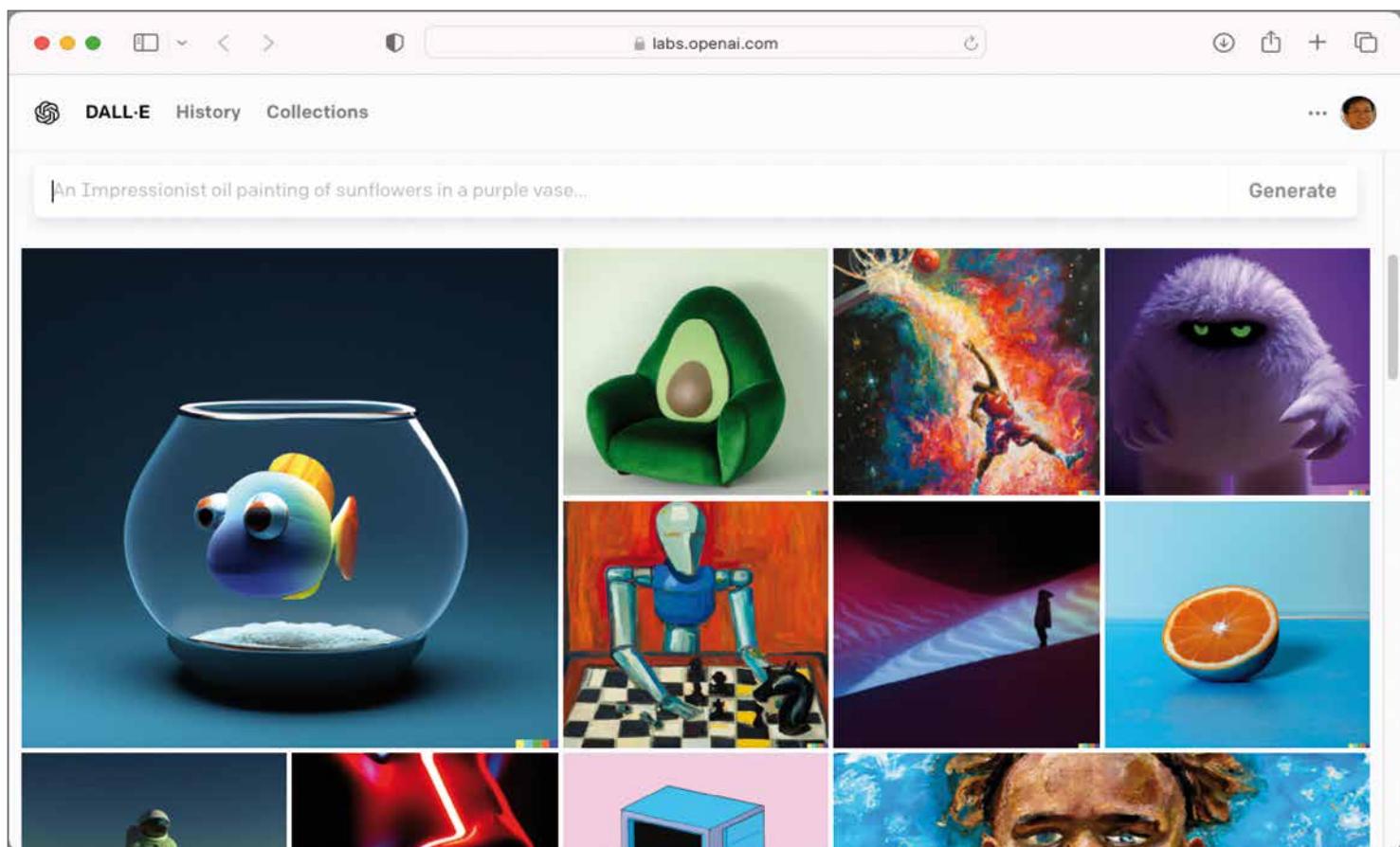
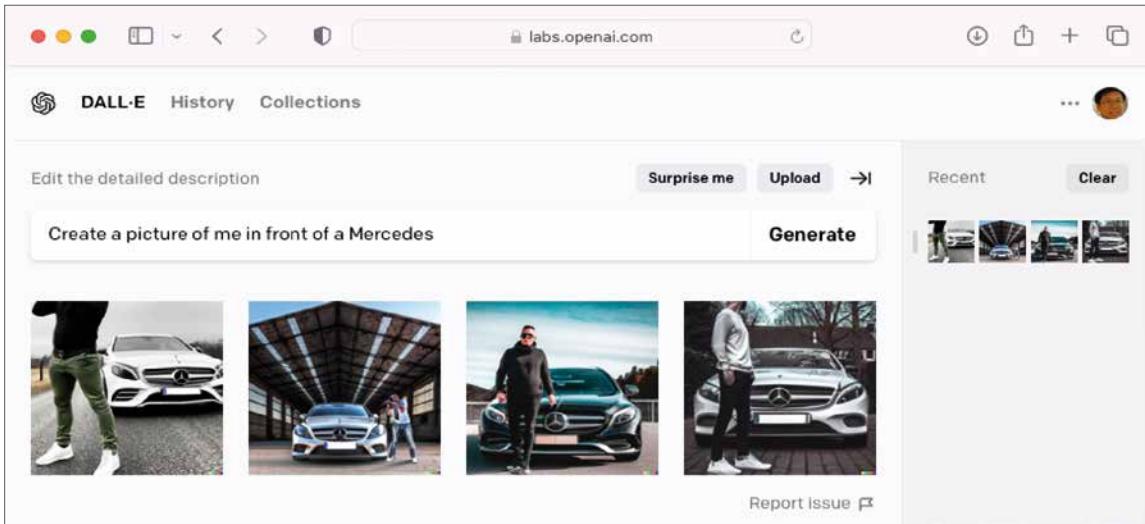


Figure 7: DALL-E 2 is able to generate new images based on your description.



**Figure 8:** The new image generated by DALL-E 2

Once the API key is obtained, it's useful to save it into an environment variable in your OS. For macOS, you can do so via the following command:

```
$ export OPENAI_API_KEY=xxxxxxxxxxxx
```

For Windows, the command is:

```
C:\>set OPENAI_API_KEY=xxxxxxxxxxxx
```

Replace xxxxxxxxxxxx with your actual OpenAI API key.

### Getting All the Models

The first thing you want to use the OpenAI API for is to get a list of models that it supports. You can do so using the following command in Terminal (macOS):

```
$ curl https://api.openai.com/v1/models \
-H "Authorization: Bearer $OPENAI_API_KEY"
```

Note that for Windows, the above command is (note that the \ character used for line continuation is not supported in Windows):

```
C:\>curl https://api.openai.com/v1/models \
-H "Authorization: Bearer %OPENAI_API_KEY%"
```

**Note:** Going forward I'll only show the command for macOS.

**Listing 1** shows the partial result returned by OpenAI.

jq is a lightweight and flexible command-line JSON processor. Go to <https://stedolan.github.io/jq/download/> for download and installation instructions.

### Listing 1: List of models returned by OpenAI

```
{
  "object": "list",
  "data": [
    {
      "id": "babbage",
      "object": "model",
      "created": 1649358449,
      "owned_by": "openai",
      "permission": [
        {
          "id": "modelperm-49FUp5v084tBB49tC4z8LPH5",
          "object": "model_permission",
          "created": 1669085501,
          "allow_create_engine": false,
          "allow_sampling": true,
          "allow_logprobs": true,
          "allow_search_indices": false,
          "allow_view": true,
          "allow_fine_tuning": false,
          "organization": "*",
          "group": null,
          "is_blocking": false
        }
      ],
      "root": "babbage",
      "parent": null
    },
    {
      "id": "davinci",
      "object": "model",
      "created": 1649359874,
      "owned_by": "openai",
      "permission": [
        {
          "id": "modelperm-U6ZwlyAd0LyMk4rcMdz33Yc3",
          "object": "model_permission",
          "created": 1669066355,
          "allow_create_engine": false,
          "allow_sampling": true,
          "allow_logprobs": true,
          "allow_search_indices": false,
          "allow_view": true,
          "allow_fine_tuning": false,
          "organization": "*",
          "group": null,
          "is_blocking": false
        }
      ],
      "root": "davinci",
      "parent": null
    },
    ...
  ]
}
```

The screenshot shows the OpenAI platform interface. On the left, there's a sidebar with 'ORGANIZATION' and 'USER' sections. Under 'USER', 'API Keys' is selected. The main content area has a title 'API keys'. It says 'Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.' Below this, it says 'Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically rotate any API key that we've found has leaked publicly.' A table lists one API key: 'SECRET KEY' (redacted), 'CREATED' (13 Feb 2023), and 'LAST USED' (13 Feb 2023). A red box highlights the 'Create new secret key' button. Below the table, under 'Default organization', it says 'If you belong to multiple organizations, this setting controls which organization is used by default when making requests with the API keys above.' A dropdown menu is set to 'Personal'. A note at the bottom says 'Note: You can also specify which organization to use for each API request. See [Authentication](#) to learn more.' A 'Display a menu' button is at the bottom left.

**Figure 9:** Creating OpenAI API keys

As the list is pretty long, I'll use the `jq` tool to extract the ID of each model:

```
$ curl https://api.openai.com/v1/models \
-H "Authorization: Bearer \$OPENAI_API_KEY" | jq '.data[].id'
```

The result of the above command now shows the ID of all the models available (see [Listing 2](#)).

### Chat

Of course, the most exciting features in OpenAI is its chat feature. The popular ChatGPT is powered by the **gpt-3.5-turbo** model, which you can access in the OpenAI API. The latest version of ChatGPT uses **GPT-4**, but at the time of writing, it's not publicly available yet.

Let's now use the **gpt-3.5-turbo** model to ask it to tell us a joke. To do that, use the following command:

```
$ curl \
https://api.openai.com/v1/chat/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer \$OPENAI_API_KEY" \
```

```
-d '{
  "model": "gpt-3.5-turbo",
  "messages": [{"role": "user",
    "content": "Tell me a joke!"}],
  "temperature": 0.7
}'
```

The result returned look like this (formatted for clarity):

```
{
  "id": "chatcmpl-7BuXX00gtQvg32UgU71D3jELqPgNm",
  "object": "chat.completion",
  "created": 1683073303,
  "model": "gpt-3.5-turbo-0301",
  "usage": {
    "prompt_tokens": 13,
    "completion_tokens": 14,
    "total_tokens": 27
  },
  "choices": [
    {
      "message": {
        "role": "assistant"
      }
    }
  ]
}
```

```

        "content": "Why don't scientists
trust atoms? \n\nBecause they make
up everything."
    },
    "finish_reason": "stop",
    "index": 0
]
}

```

The joke returned by the model is encapsulated in the **choices** key. In a later section of this article, I'll explain in more detail the meanings of **tokens** and **temperature**. Also, you can have a conversation with the model based on a previous conversation. I'll explain this in more detail when I talk about how to use this model in a Python application.

### Generating Images

OpenAI is also capable of generating images. The following command asks OpenAI to generate an image of the traditional Chinese lion dance:

```

$ curl \
https://api.openai.com/v1/
images/generations \
-H "Content-Type: application/json" \
-H "Authorization: Bearer \
$OPENAI_API_KEY" \
-d '{
  "prompt": "Lion dance",
  "n": 1,
  "size": "1024x1024"
}'

```

The result returned looks like this:

```
{
  "created": 1683075354,
  "data": [
    {
      "url": "https://oaidalleapiprodscus.blob.core.windows.net/private/org-t5oD1g1l8GcrGd2igJn9YYK9/user-5wXe19z9SoZ72ybzSoVjOs2M/img-BL1GLSqcFRuHEqlkpQIB3iM.png?st=2023-05-02T23%3A55%3A54Z&se=2023-05-03T01%3A55%3A54Z&sp=r&sv=2021-08-06&sr=b&rscd=inline&rsct=image/png&skoid=6aaadede-4fb3-4698-a8f6-684d7786b067&sktid=a48cca56-e6da-484e-a814-9c849652bc3&skt=2023-05-02T19%3A51%3A46Z&ske=2023-05-03T19%3A51%3A46Z&sks=b&skv=2021-08-06&sig=xpN8HlaJQI6/12yoJGTfJqtKjAsY4ByxbwlwL185P9M%3D"
    }
  ]
}
```

The value of the **url** key contains the image that was generated (see **Figure 10**).

## Using OpenAI Python Packages

OpenAI APIs are also available as Python packages. This is very useful if you want to integrate AI services into

your applications, particularly Python applications. For the following sections, I'll show you two examples of using OpenAI Python packages:

- Integrating ChatGPT into your Python application
- Using the Whisper API for audio transcription

### Using GTP-3 for Chatting

The first example that you'll see is how to integrate ChatGPT into your Python application. First, install the **openai** package using the **pip** command (for this section and the next, I'll be using Jupyter Notebook):

```
!pip install openai
```

The next step is to specify your OpenAI API key:

```
import openai
openai.api_key = "YOUR_API_KEY"
```

Using the **openai** package, you can list the models available:

```
models = openai.Model.list()
[model['id'] for model in models['data']]
```

**"gpt-3.5-turbo" is the latest model that is publicly available.**

You will see the list of models, similar to that in **Listing 2**.

Next, let's use the **"gpt-3.5-turbo"** for integrating ChatGPT into a Python application:

```
completion = openai.ChatCompletion.create(
model="gpt-3.5-turbo",
messages = [{"role": "user",
  "content": "What is Python?"}],
max_tokens = 1024,
```



**Figure 10:** The image of lion dance generated by OpenAI

**Listing 2:** The list of models supported by the OpenAI API

```
"babagge"
"davinci"
"text-davinci-edit-001"
"babagge-code-search-code"
"text-similarity-babagge-001"
"code-davinci-edit-001"
"text-davinci-001"
"ada"
"babagge-code-search-text"
"babagge-similarity"
"code-search-babagge-text-001"
"text-curie-001"
"code-search-babagge-code-001"
"text-ada-001"
"text-embedding-ada-002"
"text-similarity-ada-001"
"curie-instruct-beta"
"ada-code-search-code"
"ada-similarity"
"gpt-3.5-turbo-0301"
"code-search-ada-text-001"
"text-search-ada-query-001"
"babagge-search-document"
"gpt-3.5-turbo"
"ada-code-search-text"
"text-search-ada-doc-001"
"babagge-instruct-beta"
"text-similarity-curie-001"
"code-search-ada-code-001"
"ada-search-query"
"text-search-davinci-query-001"
"curie-search-query"
"babagge-search-query"
"babagge-search-document"
"ada-search-document"
"text-search-curie-query-001"
"whisper-1"
"text-search-babagge-doc-001"
"curie-search-document"
"text-search-curie-doc-001"
"babagge-search-query"
"text-babagge-001"
"text-search-davinci-doc-001"
"text-search-babagge-query-001"
"curie-similarity"
"curie"
"text-davinci-003"
"text-similarity-davinci-001"
"text-davinci-002"
"babagge-similarity"
"cushman:2020-05-03"
"ada:2020-05-03"
"babagge:2020-05-03"
"curie:2020-05-03"
"babagge:2020-05-03"
"if-davinci-v2"
"if-curie-v2"
"if-davinci:3.0.0"
"babagge-if:3.0.0"
"babagge-instruct-beta:2.0.0"
"text-ada:001"
"text-davinci:001"
"text-curie:001"
"text-babagge:001"
```

```
temperature = 0.8)

print(completion)
```

The `ChatCompletion.create()` function takes in the following arguments:

- **Model:** The model to use
- **Messages:** The message to send to the chat bot, which must be packaged as a list of dictionaries
- **max\_tokens:** The maximum number of tokens to generate in the completion. If you set this to a small number, the response returned may not be complete.
- **Temperature:** A value between 0 and 2. Lower value makes the output more deterministic. If you set it to a higher value like 0.8, the output is more likely to be different when you call the function multiple times.

Questions to be sent to ChatGPT must be enclosed in the list of dictionaries. For example, the above questions are sent to ChatGPT in the format shown in **Figure 11**.

For questions sent to ChatGPT, the dictionary must contain the **role** key set to user. The question to ask is set in the **content** key. The response from ChatGPT will look something like this:

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
```

Tokens can be thought of as pieces of words. Before the API processes your prompt, it's broken down into tokens. Approximately a 1500-word sentence is equivalent to about 2048 tokens. For more information on tokens, refer to: <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them>

```
"message": {
  "content": "Python is a high-level, interpreted programming language that is used for web development, data science, artificial intelligence, machine learning, software development, and many other applications. Python is known for its
```

```
[
  {
    "role": "user", "content": "your question"
  }
]
```

**Figure 11:** The format of the questions to send to ChatGPT

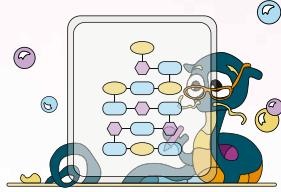
# JOIN US at the 2<sup>ND</sup> ANNUAL



## Microsoft Power Platform CONFERENCE

POWER BI   POWER AUTOMATE   POWER APPS   POWER VIRTUAL AGENTS   POWER PAGES

OCT 3-5, 2023 • MGM GRAND LAS VEGAS  
WORKSHOPS OCT 1, 2 & 6



**CHARLES LAMANNA**

*Corporate Vice President, Business Applications & Platform, Microsoft*



**HEATHER COOK**

*Principal PM Manager, Dynamics 365 & Power Platform Community Success, Microsoft*



**JULIE STRAUSS**

*General Manager, Power Platform Pro Developer & Admin Experiences, Microsoft*



**NIRAV SHAH**

*Vice President, Microsoft Dataverse, Microsoft*



**RYAN CUNNINGHAM**

*Vice President Microsoft Power Apps, Microsoft*



**SANGYA SINGH**

*Vice President of Product, Power Pages. Microsoft*



**STEPHEN SICILIANO**

*Vice President, Power Automate, Microsoft*



**KENDRA SPRINGER**

*Principal Design Lead for Power Virtual Agents and Conversational AI, Microsoft*

### ***Optional Full Day, Deep Dive Workshops***

#### **Sunday, October 1, 2023**

- Power Platform Administration and Governance
- Build a Modern App with Power Apps
- Learn How to Build PCF Controls to Give Low-Code App Makers Even More Power!
- Power Platform + List Formatting Integrations Master Class

#### **Monday, October 2, 2023**

- Power Platform Makeover: Designing Pretty, Performant and Intuitive Solutions
- Using Power Automate End-to-End: AI for Procure to Pay in SAP
- Dataverse Hands-On Workshop
- Unleash Your Inner App-BUILDER: Create a Fully-Functional Inventory Management App in One Day!
- Build an Immersive Customer Experience Portal Using Power Pages
- Everything You Wanted to Know About Power BI... But Were Afraid to Ask!

#### **Friday, October 6, 2023**

- Build Your Own Intelligent BOT Using the New Power Virtual Agents Authoring Canvas!
- Power Pages Pro Dev Workshop
- Mastering Model-Driven Power Apps: From Beginner to App Maker
- Power Automate Cloud Flows Hands-on Workshop
- Power Up Skills with Power BI
- The New and Improved! PVA Chatbot in a Day
- Day After Dashboard in a Day

**Register Today! [PowerPlatformConf.com](http://PowerPlatformConf.com) #MPPC23**



```

simplicity, readability, and ease of use,
making it a popular choice for beginners
and experts alike. It has a vast library
of modules and tools that can be used to
solve complex problems and automate tasks.
Python is open-source and free to use,
making it accessible to everyone.",
    "role": "assistant"
}
],
"created": 1683003427,
"id": "chatmpl-
7BcMVmIotmFcAFa4Wc9KyLB9Rp0gz",
"model": "gpt-3.5-turbo-0301",
"object": "chat.completion",
"usage": {
    "completion_tokens": 94,
    "prompt_tokens": 12,
    "total_tokens": 106
}
}

```

The result in JSON contains numerous information. In particular, the value of the **choices** key is an array, of



**Figure 12:** You need to append the previous response in order for ChatGPT to continue with the conversation.

which the first element contains the result that you want, stored in the **message** dictionary in the **content** key:

```

message = \
    completion.choices[0].message.content
print(message)

```

And here is the response returned by ChatGPT:

Python is a high-level, interpreted programming language that is used for web development, data science, artificial intelligence, machine learning, software development, and many other applications. Python is known for its simplicity, readability, and ease of use, making it a popular choice for beginners and experts alike. It has a vast library of modules and tools that can be used to solve complex problems and automate tasks. Python is open-source and free to use, making it accessible to everyone.

It's important to remember that ChatGPT doesn't remember your previous questions. So in order for you to have a meaningful conversation with it, you need to feed the conversation back to the API. Remember the list of dictionaries you need to pass to the API?

To feed the previous conversation back to ChatGPT, you first append the reply from ChatGPT to the list. Then, you append your follow-up question (see **Figure 12**).

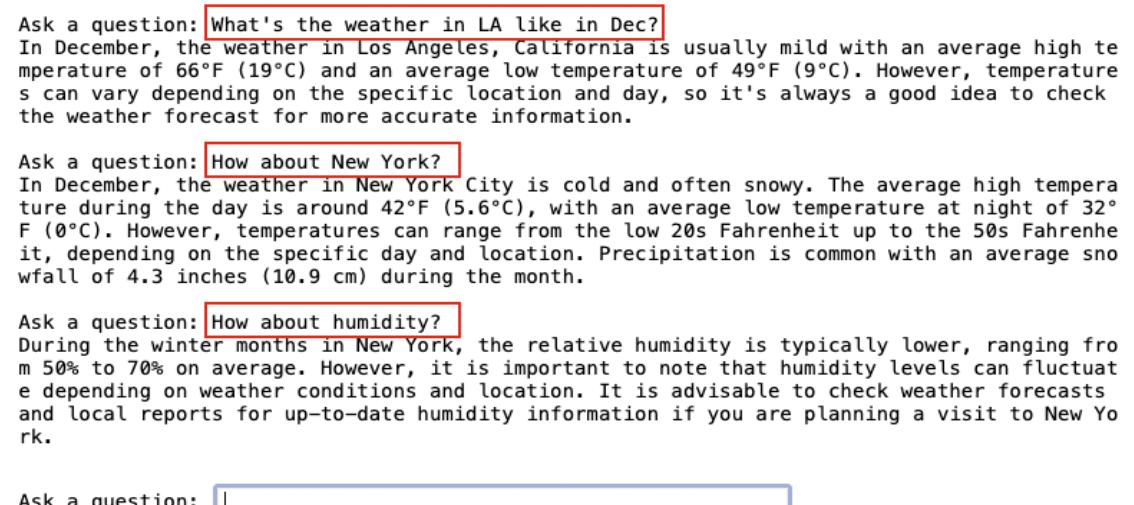
This way, ChatGPT is able to know the previous questions that you have asked and the responses it provided. Here's the updated code to allow the user to have a meaningful conversation with ChatGPT:

```

Messages = []

while True:
    prompt = input("\nAsk a question: ")

```



**Figure 13:** Having a conversation with ChatGPT using the Python application

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	<code>tiny.en</code>	<code>tiny</code>	~1 GB	~32x
base	74 M	<code>base.en</code>	<code>base</code>	~1 GB	~16x
small	244 M	<code>small.en</code>	<code>small</code>	~2 GB	~6x
medium	769 M	<code>medium.en</code>	<code>medium</code>	~5 GB	~2x
large	1550 M	N/A	<code>large</code>	~10 GB	1x

**Figure 14:** The various models for the Whisper API (source: <https://pypi.org/project/openai-whisper/>)

```
messages.append(
{
    'role':'user',
    'content':prompt
})

# creating a chat completion
completion = \
    openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages = messages)

# extract the response from GPT
response = \
    completion['choices'][0]\
        ['message']['content']
print(response)

# append the response from GPT
messages.append(
{
    'role':'assistant',
    'content':response
})
```

`whisper.available_models()`

There are five main models: tiny, base, small, medium, and large:

```
['tiny.en',
'tiny',
'base.en',
'base',
'small.en',
'small',
'medium.en',
'medium',
'large-v1',
'large-v2',
'large']
```

Model names ending with .en are English-only and tend to perform better.

**Figure 13** shows a conversation I had with ChatGPT. The rectangles in red are my questions.

### Whisper API

Whisper is a general-purpose speech recognition model offered by OpenAI. It's trained on a large dataset of diverse audio and is also a multitasking model that can perform multilingual speech recognition, speech translation, and language identification.

In this section, I'll show you how to use the Whisper API in Python to perform transcription of audio files (to text), and how to translate non-English results into English.

### Installing the Whisper Python Package

To install the Whisper API, type the following command in a new cell in your Jupyter Notebook:

```
!pip install -U openai-whisper
```

### Viewing Available Models and Languages

The Whisper API offers five main pre-trained models that you can use for your transcription. You can view the five models by using the `models()` function:

**Figure 14** shows the details of the various models, such as their number of trainable parameters, required memory, and relative execution speed.

### Creating a Model

Let's now load up a model that you want to use for the transcription:

```
import whisper
model = whisper.load_model("base")
```

When you create a model for the first time, the weights of the model are downloaded onto your computer. It will be saved in the `~/.cache/whisper` directory.

### Transcribing Audio and Translating Text

To transcribe audio, you can either load the audio file that's saved locally on your computer or supply a URL pointing to the audio file. The following example loads the audio file using an URL:

```
result = model.transcribe(
    'https://www.voiptroubleshooter.com/' +
```

	<b>id</b>	<b>seek</b>	<b>start</b>	<b>end</b>	<b>text</b>	<b>tokens</b>	<b>temperature</b>	<b>avg_logprob</b>	<b>compression_ratio</b>	<b>no_speech_prob</b>
<b>0</b>	0	0	0.00	6.52	The first can be slid on the smooth planks.	[50364, 440, 700, 393, 312, 1061, 327, 322, 26...	0.0	-0.292909	1.510526	0.024192
<b>1</b>	1	0	6.52	10.32	Glue the sheet to the dark blue background.	[50690, 49832, 264, 8193, 281, 264, 2877, 3344...	0.0	-0.292909	1.510526	0.024192
<b>2</b>	2	0	10.32	14.18	It's easy to tell the depth of a well.	[50880, 467, 311, 1858, 281, 980, 264, 7161, 2...	0.0	-0.292909	1.510526	0.024192
<b>3</b>	3	0	14.18	18.28	These days, the chicken leg is a rare dish.	[51073, 1981, 1708, 11, 264, 4662, 1676, 307, ...]	0.0	-0.292909	1.510526	0.024192
<b>4</b>	4	0	18.28	22.04	Rice is often served in round bowls.	[51278, 19386, 307, 2049, 7584, 294, 3098, 285...	0.0	-0.292909	1.510526	0.024192
<b>5</b>	5	0	22.04	25.92	The juice of lemons makes fine punch.	[51466, 440, 8544, 295, 47098, 1669, 2489, 813...	0.0	-0.292909	1.510526	0.024192
<b>6</b>	6	0	25.92	29.88	The box was thrown beside the part truck.	[51660, 440, 2424, 390, 11732, 15726, 264, 644...	0.0	-0.292909	1.510526	0.024192
<b>7</b>	7	2988	29.88	34.00	The house were fed chopped corn and garbage.	[50364, 440, 1782, 645, 4636, 16497, 9046, 293...	0.0	-0.341265	1.173077	0.000315
<b>8</b>	8	2988	34.00	37.48	Four hours of study work faced us.	[50570, 7451, 2496, 295, 2979, 589, 11446, 505...	0.0	-0.341265	1.173077	0.000315
<b>9</b>	9	2988	37.48	39.80	A large size of stockings is hard to sell.	[50744, 316, 2416, 2744, 295, 4127, 1109, 307,...]	0.0	-0.341265	1.173077	0.000315

**Figure 15:** Displaying the segments as a dataframe

```
'open_speech/american/' +
'OSR_us_000_0015_8k.wav')
result
```

If you see a warning like “*UserWarning: FP16 is not supported on CPU; using FP32 instead*”, that means your model isn’t able to make use of the GPU; instead it makes use of the CPU.

**FP16 (half-precision floating point) uses 16-bits for storing floating point numbers while FP32 (single precision floating point) uses 32-bits. Although FP32 allows higher precision and accuracy, it comes with a cost in terms of larger memory footprints. In general, for deep learning, FP16 is preferred over FP32 due to its faster computation times, and also due to the fact that tasks like image classification and object detection do not require a lot of precision.**

The result returned by the `transcribe()` function contains the transcription (text) and a list of all the transcription segments (segments):

```
{'text': " The first can be slid on the
smooth planks. Glue the sheet to the dark
blue background. It's easy to tell the
```

depth of a well. These days, the chicken leg is a rare dish. Rice is often served in round bowls. The juice of lemons makes fine punch. The box was thrown beside the part truck. The house were fed chopped corn and garbage. Four hours of study work faced us. A large size of stockings is hard to sell.”,

```
'segments': [{"id': 0,
'seek': 0,
'start': 0.0,
'end': 6.520000000000005,
'text': ' The first can be slid on the
smooth planks.',
'tokens': [50364,
440,
700,
393,
312,
1061,
327,
322,
264,
5508,
499,
14592,
13,
50690],
'temperature': 0.0,
'avg_logprob': -0.29290932700747535,
'compression_ratio': 1.5105263157894737,
'no_speech_prob': 0.024191662669181824},
{'id': 1,
'seek': 0,
'start': 6.520000000000005,
'end': 10.32,
'text': ' Glue the sheet to the dark
blue background.',
'tokens': [50690, 49832, 264, 8193, 281,
```

```

264, 2877, 3344, 3678, 13, 50880],
'temperature': 0.0,
'avg_logprob': -0.29290932700747535,
'compression_ratio': 1.5105263157894737,
'no_speech_prob': 0.024191662669181824},
...
],
'language': 'en'}

```

The detail of each segment contains information like start time, end time, text, and more. You can load the values of the segments key as a Pandas DataFrame for easier inspection (see **Figure 15**):

```

import pandas as pd
df = pd.DataFrame.from_dict(
    result['segments'])
df

```

If you're more interested in the transcription of the audio file, extract the value of the **text** key:

```
result["text"]
```

Here's the transcription of the audio:

"The first can be slid on the smooth planks. Glue the sheet to the dark blue background. It's easy to tell the depth of a well. These days, the chicken leg is a rare dish. Rice is often served in round bowls. The juice of lemons makes fine punch. The box was thrown beside the part truck. The house were fed chopped corn and garbage. Four hours of study work faced us. A large size of stockings is hard to sell."

As the transcription can take a while to perform, you can set the **verbose** parameter to **True** so that the transcription segments can be displayed as and when it is ready:

```

result = model.transcribe(
    'https://www.voiptroubleshooter.com/' +
    'open_speech/french/' +
    'OSR_fr_000_0043_8k.wav',
    verbose = True)
result

```

Here it the output when you set the **verbose** parameter to **True**:

```

Detecting language using up to the first 30
seconds. Use '--language' to specify the
language
Detected language: English
[00:00.000 --> 00:06.520] The first can be
slid on the smooth planks.
[00:06.520 --> 00:10.320] Glue the sheet to
the dark blue background.
[00:10.320 --> 00:14.180] It's easy to tell
the depth of a well.
[00:14.180 --> 00:18.280] These days, the
chicken leg is a rare dish.
[00:18.280 --> 00:22.040] Rice is often
served in round bowls.
[00:22.040 --> 00:25.920] The juice of
lemons makes fine punch.
[00:25.920 --> 00:29.880] The box was
thrown beside the part truck.

```

```

[00:29.880 --> 00:34.000] The house were
fed chopped corn and garbage.
[00:34.000 --> 00:37.480] Four hours of
study work faced us.
[00:37.480 --> 00:39.800] A large size of
stockings is hard to sell.

```

Not only can Whisper transcribe your audio, it can also translate the result into English. Let's transcribe an audio which is in French:

```

result = model.transcribe(
    'https://www.voiptroubleshooter.com/' +
    'open_speech/french/' +
    'OSR_fr_000_0043_8k.wav',
    verbose = True)
result

```

Here's the result in French:

```

Detecting language using up to the first
30 seconds. Use '--language' to specify
the language
Detected language: French
[00:00.000 --> 00:05.000] Néon Pâle, la
mémoire accorde.
[00:05.000 --> 00:09.000] Instinctivement,
il arrête à son cheval.
[00:09.000 --> 00:13.000] Je vous remercie
d'être là, mesur.
[00:13.000 --> 00:17.000] Le chef de ces
russes était un certain évon.
[00:17.000 --> 00:21.000] Le feu est signé
de civilisation.
[00:21.000 --> 00:25.000] Son visage était
sans expression.
[00:25.000 --> 00:30.000] Il y a-t-il une
raison majeure à cela ?
[00:30.000 --> 00:36.000] Gabriel va
épouser Simon, la petite boulangerie.
[00:36.000 --> 00:41.000] La vache noire
a eu un petit vêtre adorable.
[00:41.000 --> 00:46.000] C'est lui qui a
distribué les armes.
[00:46.000 --> 00:50.000] Ce dernier avait
un rendez-vous.
[00:51.000 --> 00:55.000] C'était peu,
mais c'était assez.
[00:55.000 --> 00:58.000] Mais le temps
lui a manqué.
[00:58.000 --> 01:03.000] Nous avons tous
notre idée sur la question.
[01:03.000 --> 01:08.000] Vous bénéficierez
des sièges très confortables.
[01:08.000 --> 01:14.000] Cette comédie ne
manque pas du mort.
[01:14.000 --> 01:19.000] La vallange
déferne sur les chalets.
[01:19.000 --> 01:24.000] Il jouit d'une
grande popularité.

```

If you want the transcription to be in English, you simply need to set the **task** parameter to **translate**:

```

result = model.transcribe(
    'https://www.voiptroubleshooter.com/' +

```

### **Listing 3:** The code to train ChatGPT using our own data

```
from gpt_index import SimpleDirectoryReader, \
    GPTListIndex, \
    GPTSimpleVectorIndex, \
    LLMPredictor, \
    PromptHelper

# the ChatOpenAI class allows you to use the OpenAI's
# models at https://platform.openai.com/docs/models
from langchain.chat_models import ChatOpenAI
import os

os.environ["OPENAI_API_KEY"] = "YOUR_OPENAI_API_KEY"

# index the documents in the specified path
def index_documents(folder):
    max_input_size = 4096
    num_outputs = 512
    max_chunk_overlap = 20
    chunk_size_limit = 600

    # helps us fill in the prompt, split the text, and
    # fill in context information according to
    # necessary token limitations
    prompt_helper = PromptHelper(max_input_size,
                                  num_outputs,
                                  max_chunk_overlap,
                                  chunk_size_limit=chunk_size_limit)

    # the LLMPredictor class is a wrapper around
    # an LLMChain from Langchain
    llm_predictor = LLMPredictor(
        llm = ChatOpenAI(temperature = 0.7,
                         model_name = "gpt-3.5-turbo",
                         max_tokens = num_outputs))

    # load the documents from the documents folder
    documents = \
        SimpleDirectoryReader(folder).load_data()

    # The GPTSimpleVectorIndex is a data structure
    # where nodes are keyed by embeddings, and
    # those embeddings are stored within a simple
    # dictionary. During index construction, the
    # document texts are chunked up, converted to
    # nodes with text; they are then encoded in
    # document embeddings stored within the dict.
    index = GPTSimpleVectorIndex(documents,
                                 llm_predictor = llm_predictor,
                                 prompt_helper = prompt_helper)
    index.save_to_disk('index.json')

index_documents("training documents")
```

### **SPONSORED SIDEBAR:**

#### AI Executive Briefing

Experience the game-changing impact of AI through CODE Consulting's Executive Briefing service. Uncover the immense potential and wide-ranging benefits of AI in every industry. Our briefing provides strategic guidance for seamless implementation, covering crucial aspects such as infrastructure, talent acquisition, and leadership. Discover how to effectively integrate AI and propel your organization into future success. Contact us today to schedule your executive briefing and embark on a journey of AI-powered growth. [www.codemag.com/ai](http://www.codemag.com/ai)

```
'open_speech/french/' +
'OSR_fr_000_0043_8k.wav',
verbose = True,
task = 'translate')
result
```

And here's the transcription in English:

```
Detecting language using up to the first
30 seconds. Use '--language' to specify
the language
Detected language: French
[00:00.000 --> 00:05.000] Do not let the
short memory be.
[00:05.000 --> 00:10.000] Instinctively,
instantly, it stops the sound.
[00:10.000 --> 00:14.000] I thank you for
being there, Mésure.
[00:14.000 --> 00:18.000] The chief of his
Russians was a certain Ivan.
[00:18.000 --> 00:22.000] The fire and
signs of civilization.
[00:22.000 --> 00:27.000] His face was his
expression.
[00:27.000 --> 00:31.000] Is there a major
reason for that?
[00:31.000 --> 00:36.000] Gabriel was
supposed to put Simon the little bull in
the air.
[00:36.000 --> 00:42.000] The black cow
had a little adorable.
[00:42.000 --> 00:46.000] It's him who distributed the weapons.
[00:46.000 --> 00:50.000] This last one
had a big one for you.
[00:50.000 --> 00:54.000] It was a little,
but it was enough.
[00:55.000 --> 00:58.000] But the time was
missing.
```

```
[00:58.000 --> 01:03.000] We all have our
ideas on the question.
[01:03.000 --> 01:08.000] You will benefit
very comfortable skies.
[01:08.000 --> 01:14.000] This comedy does
not lack humor.
[01:14.000 --> 01:18.000] The wave is
falling on the shoulders.
[01:18.000 --> 01:23.000] He is Jewish, a
great popularity.
```

#### **Training ChatGPT Using Your Custom Data**

Although it's fun to ask ChatGPT based on the data it's been trained on, what's more useful is if ChatGPT could answer specific questions based on your own training data. For example, say I have the PDFs of CODE Magazine for the past 20 years. These 20 years-worth of content provides a very useful database of coding knowledge. It would be fantastic if ChatGPT could learn from this set of magazine content and be able to answer questions that you throw at it. This is exactly what I'm going to do in this section! The following sections will show you how you can train ChatGPT using your own custom data.

#### **Installing the Prerequisites**

For this exercise, you need to install the following packages:

```
!pip install gpt_index==0.4.24
!pip install gradio
!pip install langchain==0.0.107
!pip install PyPDF2
```

The uses of these packages are as follows:

- **aIndex (gpt\_index)** is a project that provides a central interface to connect your LLMs with external data.
- **Gradio** is a Python package that displays UI for interacting with AI chatbot.

- **LangChain** is a framework for developing applications powered by language models.
- **PyPDF2** is a Python package for reading PDF files.

### Preparing the Training Data

For the data that I'll be using, I'll use three PDF documents of the most recent issues of CODE Magazine. Create a folder in the same folder as your Jupyter Notebook and name it as **training documents**. Drag and drop a few PDF documents into the folder (see **Figure 16**).

You can put as many files as you want in the **training documents** folder. Besides PDF documents, text files are also supported.

### Training ChatGPT

You're now ready to write the code to train ChatGPT using your own data. **Listing 3** shows the code with the comments inline.

After the training, the model is saved into the **index.json** file. When you run the code snippet, you'll see the following output:

```
INFO:root:> [build_index_from_documents]
Total LLM token usage: 0 tokens
INFO:root:> [build_index_from_documents]
Total embedding token usage: 218071 tokens
```

### Asking Questions

You can now ask questions pertaining to the content of the PDFs that you've used for training:

```
def Code_Mag_KB(input_text):
    index = \
        GPTSimpleVectorIndex.load_from_disk(
            'index.json')
    response = index.query(input_text,
                           response_mode = "compact")
    return response.response

Code_Mag_KB(
    'Summarize what a smart contract is?')
```

When you run the above code snippet, you get the following output:

```
INFO:root:> [query] Total LLM token usage:
  641 tokens
INFO:root:> [query] Total embedding token
usage: 9 tokens
```

And the response is also shown:

"\nA smart contract is a piece of code that is stored on a blockchain and is used to execute certain functions. It can write values onto the blockchain and transfer funds between accounts. Smart contracts are useful for verifying credentials and other data that needs to be stored securely and immutably."

### Wrapping ChatGPT Using a Web-Based UI

Now that ChatGPT is trained using your data, it would be more user-friendly if you could wrap it up using a web-based UI so that users can directly interact with ChatGPT. The following code snippet wraps ChatGPT using the **Gradio** package:

**Gradio** is a Python package that creates a web-based UI for interacting with your machine learning/deep learning models.

```
import gradio as gr

# display the UI
interface = gr.Interface(fn = Code_Mag_KB,
    inputs = gr.components.Textbox(
        lines = 5,
        label = 'Enter your question'),
    outputs = 'text',
    title = 'CODE Magazine Knowledge Base')

interface.launch(share=False)
```

When you run the above code snippet, you'll see the UI as shown in **Figure 17**.

Type in your question on the left side of the textbox and click **Submit**. The response from ChatGPT is shown in the right textbox (see **Figure 18**).

### Integrating ChatGPT with Jupyter Notebook

The last item that I want to discuss in this article is to demonstrate how you can integrate ChatGPT into your Jupyter Notebook. By integrating ChatGPT into your Jupyter Notebook, you can now get ChatGPT to:

- Add Docstrings for your Python functions.
- Provide an explanation for your code.
- Debug your code.
- Complete your code.
- Review your code.
- Ask questions.

As ChatGPT—Jupyter—AI Assistant is a Chrome extension, you need to run your Jupyter Notebook in Chrome.



**Figure 16:** Populating the training documents folder with three PDFs

The screenshot shows a user interface for a knowledge base. At the top, it says "CODE Magazine Knowledge Base". On the left, there's a text input field with the placeholder "Enter your question". To its right is a larger text area labeled "output". Below the input field are two buttons: "Clear" and "Submit". To the right of the "Submit" button is a "Flag" button.

**Figure 17:** The web UI shown by Gradio

The screenshot shows the same user interface as Figure 17, but with a question typed into the input field: "What is the language used for smart contract programming?". The output field now contains the response: "The language used for smart contract programming is Solidity.".

**Figure 18:** Type your questions and get the answer from ChatGPT

To use ChatGPT in your Jupyter Notebook, you need to install the ChatGPT—Jupyter—AI Assistant Chrome extension in your Chrome browser (<https://chrome.google.com/webstore/detail/chatgpt-jupyter-ai-assist/dlipnckjmmjjdpvcnokbdbkjadiejl/related>). Click **Add to Chrome** to add the extension (see **Figure 19**). In the pop-up that appears, click Add Extension.

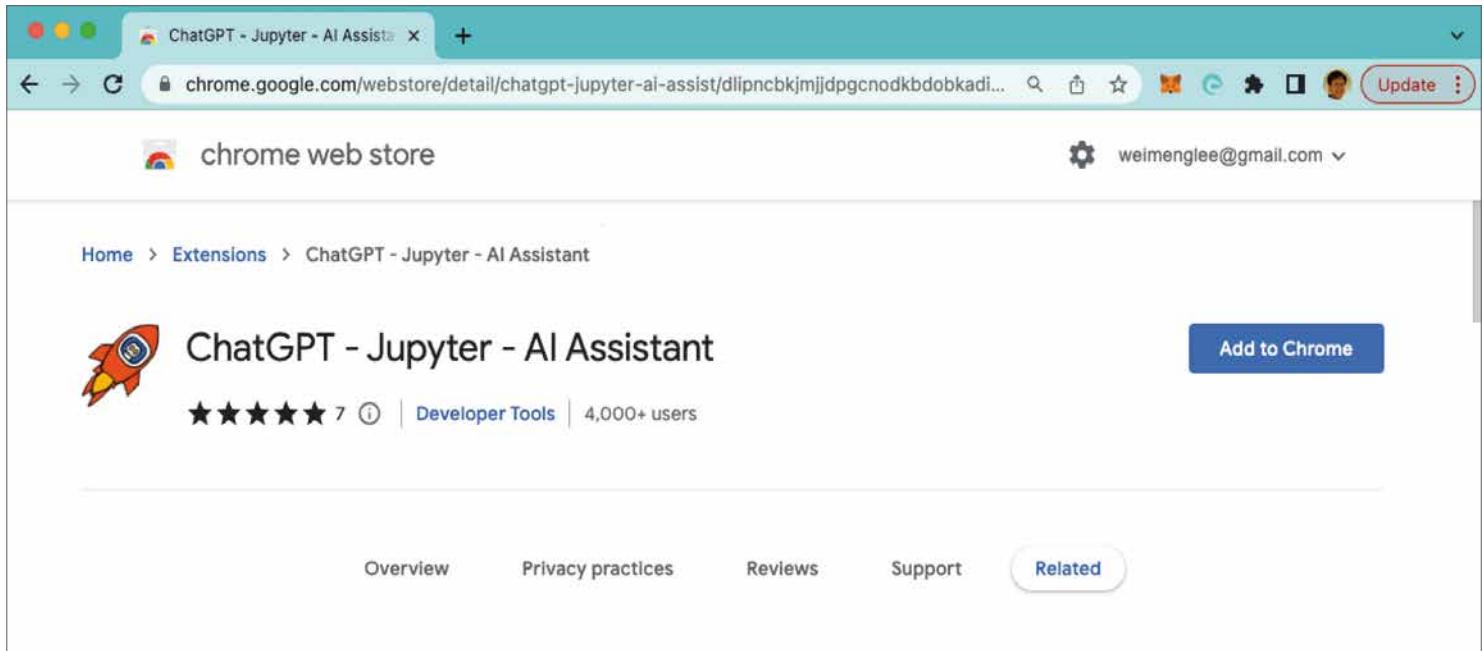
Once the extension is added, you need to configure it—you need to enter your OpenAI key (see **Figure 20**).

When you now run your Jupyter Notebook, you'll see the following buttons in the toolbar (see **Figure 21**). In the next few sections, I'll explain the use of each of the buttons.

#### *Format: Adding Docstrings to Your Python Function*

The first button is the **Format** button. Suppose I have the following code snippet in a cell:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
def calculate_vif(df, features):
    vif, tolerance = {}, {}
    # all the features that you want to
    # examine
    for feature in features:
        # extract all the other features
        # you will regress against
        X = [f for f in features if
              f != feature]
        X, y = df[X], df[feature]
        # extract r-squared from the fit
        r2 = LinearRegression().fit(X, y).score(X, y)
        vif[feature] = 1 / r2
        tolerance[feature] = 1 - r2
```



**Figure 19:** Adding the ChatGPT extension to Chrome

```
X, y).score(X, y)

# calculate tolerance
tolerance[feature] = 1 - r2

# calculate VIF
vif[feature] = 1/(tolerance[feature])

# return VIF DataFrame
return pd.DataFrame({'VIF': vif,
    'Tolerance': tolerance})
```

Clicking on the **Format** button makes ChatGPT add the Docstrings to the function as shown in **Figure 22**.

Here's what ChatGPT has added and modified (shown in bold):

```
import pandas as pd

from sklearn.linear_model import
LinearRegression

def calculate_vif(df: pd.DataFrame,
    features: list) -> pd.DataFrame:
    """
    Calculate Variance Inflation Factor
    (VIF) and Tolerance for each feature
    in a pandas DataFrame

    Parameters:
    df (pd.DataFrame): DataFrame containing
        data for all features features (list):
        List of strings of all column names in
        the DataFrame that needs to be
        calculated

    Returns:
    pandas.DataFrame: Returns DataFrame
        containing VIF and Tolerance for each
```

**Which AI Provider do you want to use?**

You might not see all your models until you authenticate with OpenAI. You can do so by using the extension once.

ChatGPT Webapp    **OpenAI API**

OpenAI official API: you will be charged per use!

gpt-3.5-turbo

Save And Activate As Current Provider

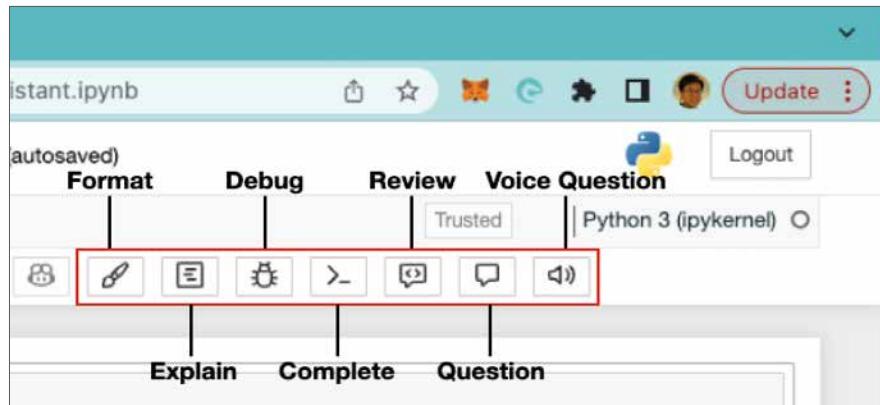
**What is your OpenAI API Key?**

API key: ..... You can find or create your API key here Warning: the voice features use the OpenAI Whisper API, which charges \$0.006 per minute.

Save API Key

Pro-tip: click here to set keyboard shortcuts --> ⌘ Shortcuts

**Figure 20:** Adding the OpenAI API key to the ChatGPT extension



**Figure 21:** The ChatGPT buttons in Jupyter Notebook

```

feature

"""
vif, tolerance = {}, {}

# Iterate over all the features that
need to be examined

for feature in features:
    # Extract all the other features
    # you will regress against
    X = [f for f in features if
         f != feature]
    X, y = df[X], df[feature]

    # Extract R^2 from the fit

```

The screenshot shows the Jupyter Notebook interface with the ChatGPT toolbar buttons highlighted by a red box. Below the toolbar, a code cell contains Python code for calculating VIF and Tolerance. A separate window titled 'ChatGPT - Improve Formatting (prompt)' shows the generated docstring for the function.

```

In [1]: import pandas as pd
       from sklearn.linear_model import LinearRegression

def calculate_vif(df, features):
    vif, tolerance = {}, {}

    # all the features that you want to examine
    for feature in features:
        # extract all the other features you will regress against
        X = [f for f in features if f != feature]
        X, y = df[X], df[feature]

        # extract r-squared from the fit
        r2 = LinearRegression().fit(X, y).score(X, y)

        # calculate tolerance
        tolerance[feature] = 1 - r2

    # calculate VIF

```

**ChatGPT - Improve Formatting (prompt)**

**Model:** OpenAI API - gpt-3.5-turbo @

```

import pandas as pd
from sklearn.linear_model import LinearRegression

def calculate_vif(df: pd.DataFrame, features: list) -> pd.DataFrame:
    """
    Calculate Variance Inflation Factor (VIF) and Tolerance for each feature in a pandas DataFrame

    Parameters:
    df (pd.DataFrame): DataFrame containing data for all features
    features (list): List of strings of all column names in the DataFrame that needs to be calculated

    Returns:
    pandas.DataFrame: Returns DataFrame containing VIF and Tolerance for each feature
    """

    vif, tolerance = {}, {}

```

**Figure 22:** ChatGPT adding the Docstrings to my function

```

r2 = LinearRegression().fit(
    X, y).score(X, y)

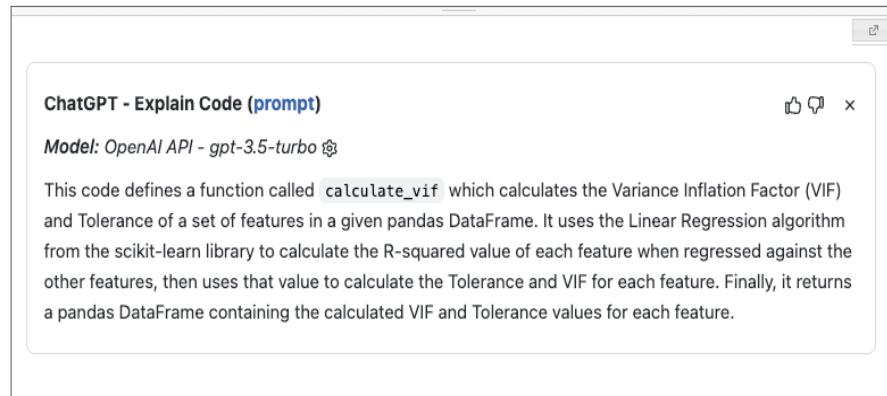
# Calculate tolerance
tolerance[feature] = 1 - r2

# Calculate VIF
vif[feature] = 1/(tolerance[feature])

# Return VIF DataFrame
return pd.DataFrame({'VIF': vif,
    'Tolerance': tolerance})

```

Not only does ChatGPT add the Docstrings for the function, it also modified my existing comments and the function signature:



**Figure 23:** ChatGPT attempts to explain what my code is doing.

**ChatGPT - Debug Code (prompt)**

**Model:** OpenAI API - gpt-3.5-turbo

The code you provided seems to be a function called `calculate_vif` that calculates Variance Inflation Factor (VIF) for a given set of features. However, there are a few issues in the code that could be causing your error:

- There is no dataframe `df` defined in the function, even though it is used to extract X and y values for the Linear Regression.
- The `features` input argument to the function is not used anywhere in the function.

To fix these issues, you can modify the code as follows:

```

import pandas as pd
from sklearn.linear_model import LinearRegression

def calculate_vif(df, features):
    vif, tolerance = {}, {}

    # all the features that you want to examine
    for feature in features:
        # extract all the other features you will regress against
        X = df[[f for f in features if f != feature]]
        y = df[feature]

        # extract r-squared from the fit
        r2 = LinearRegression().fit(X, y).score(X, y)

        # calculate tolerance
        tolerance[feature] = 1 - r2

        # calculate VIF
        vif[feature] = 1/tolerance[feature]

    # return VIF DataFrame
    return pd.DataFrame({'VIF': vif, 'Tolerance': tolerance})

```

In the modified code, we pass a dataframe `df` as an argument to the function, fix the extraction of X and y values for the Linear Regression, and use the `features` input argument to extract the relevant columns from the dataframe.

**Figure 24:** ChatGPT debugging my code

**ChatGPT - Complete Code (prompt)**

**Model:** OpenAI API - gpt-3.5-turbo

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif(df, features):

    X = df[features]
    vif = pd.DataFrame()
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return vif
```

I imported the `variance_inflation_factor` method from `statsmodels.stats.outliers_influence` module, which will be used to calculate the VIF scores. I then created a function `calculate_vif` with parameters `df` and `features`. In the body of the function, I assigned the dataframe columns containing the features to the variable `X`. I then created a new dataframe called `vif` with two columns 'Features' and 'VIF'. I calculated the VIF score using `variance_inflation_factor` function for each feature of `X`, and added it to the 'VIF' column. Finally, I returned the `vif` dataframe.

**Figure 25:** ChatGPT writes the code for you based on the function name.

**ChatGPT - Code Review (prompt)**

**Model:** OpenAI API - gpt-3.5-turbo

- It would be helpful to include some comments or docstrings to explain what the function does and how to use it.

```
# Recommend adding comments or docstrings to explain function purpose and usage
import pandas as pd
from sklearn.linear_model import LinearRegression

def calculate_vif(df,features):
    """
    Calculate the Variance Inflation Factor (VIF) and tolerance for a set of features.

    Parameters:
    -----
    df : pandas.DataFrame
        Dataframe containing the data
    features : list
```

**Figure 26:** ChatGPT offers ways to improve my code.

:: Your question or task: X

How to test for multicollinearity?

**Submit (ctrl+Enter)**

**Figure 27:** Asking questions to ChatGPT directly in Jupyter Notebook

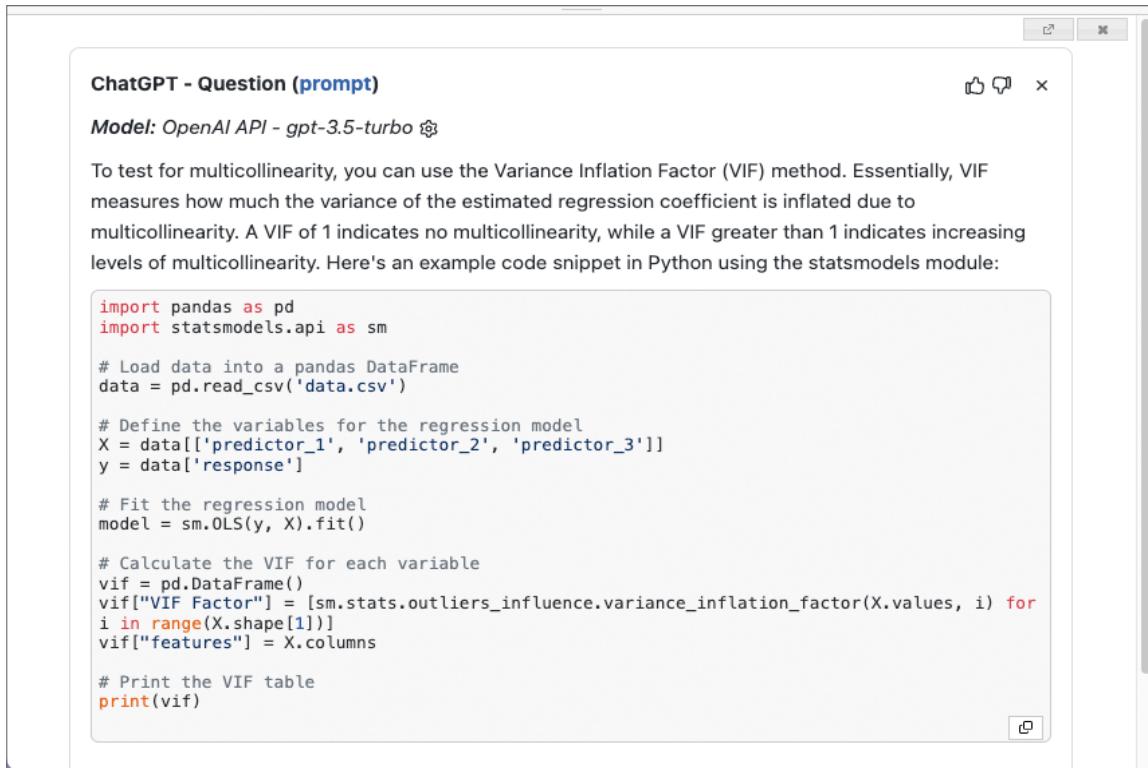
```
def calculate_vif(df: pd.DataFrame,
                  features: list) -> pd.DataFrame:
    """
```

**Explain: Providing an Explanation of What the Code Is Doing**

When you click on the **Explain** button, ChatGPT attempts to explain the code in the currently selected cell. For my function, it explains it very well (see **Figure 23**).

**Debug: Fixing the Bug in the Code**

Let's remove the `df` parameter from the `calculate_vif()` function in my sample code snippet:



**Figure 28:** The response from ChatGPT

```

def calculate_vif(features):
    vif, tolerance = {}, {}
    ...

```

When I now click on the **Debug** button, ChatGPT attempts to debug the code. It rightfully pointed out that there is no **df** defined in the function (see **Figure 24**). And it even offers a solution to fix the code.

#### Complete: Writing the Code for You

Suppose you want to write the VIF function but aren't sure how to do it. You could simply define the function like the following and then click on the **Complete** button:

```
def calculate_vif(df, features):
```

Based on the function name, ChatGPT attempts to write the code for you (see **Figure 25**)

Pretty amazing, isn't it?

#### Review: Performing a Code Review and Suggesting Improvements

With your function written, you could ask ChatGPT to do a code review of your code. When I click the **Review** button, ChatGPT shows me the suggestions, as shown in **Figure 26**.

Specifically, it suggested the following:

- It would be helpful to include some comments or docstrings to explain what the function does and how to use it.
- It may be useful to include some error handling in case the function is called with an empty or invalid features list.

- Consider making the function more flexible by allowing an optional argument for the regression model.
- The variable **X** is defined twice, in lines 18 and 23. It would be cleaner to extract it only once.
- The variable **X** is a list of feature names, but it would be more consistent to use it as a pandas DataFrame with corresponding column selection.

For each suggestion, it provides the modified code snippet.

#### Questions

Finally, in addition to going to <https://chat.openai.com> to ask a question, you can directly ask in Jupyter Notebook. Clicking the **Question** button displays the pop-up shown in **Figure 27**. Type your question and press Ctrl-Enter to submit the question.

**Figure 28** shows the response provided by ChatGPT.

## Summary

In this article, I've discussed the various ways you can interact with ChatGPT. Take note that for the OpenAI API, first time users get a free \$5 worth of tokens to try out the services. These tokens expire after three months. Thereafter, you need to upgrade to a paid plan where you will be charged based on usage. This is important to note because for every call you make to the API, you will be charged. However, you can set a limit on your usage. Have fun with OpenAI!

Wei-Meng Lee  
**CODE**

# Building Microservices Architecture Using GraphQL and ASP.NET 7 Core

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 this article, I'll examine GraphQL, its features and benefits, and how it compares and contrasts with REST and gRPC. I'll then discuss how you can take



**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.



advantage of GraphQL to build microservices-based real-time applications in ASP.NET Core 7.

If you're to work with the code examples discussed in this article, you need the following installed in your system:

- Visual Studio 2022
- .NET 7.0
- ASP.NET 7.0 Runtime

If you don't already have Visual Studio 2022 installed in your computer, you can download it from here: <https://visualstudio.microsoft.com/downloads/>.

## What Is GraphQL?

GraphQL was created in 2012 by Facebook as a faster and more efficient way of querying data. Using GraphQL, you can query only the data you need from an API, enhancing efficiency and reducing unnecessary data transfers. REST APIs require a client to make multiple requests to retrieve data from different endpoints, which can be inefficient and time-consuming.

GraphQL can help you to specify only the data you might need in a single request. It's an efficient way of querying data compared to traditional methods such as REST or RPC. GraphQL provides a declarative approach to retrieving and manipulating data, making it easy for developers to work with complex data.

## Benefits and Downsides

GraphQL has various benefits over conventional REST APIs, such as:

- **Better efficiency:** By allowing clients to specify exactly what information they want, GraphQL boosts API productivity by cutting down on data transmission that isn't used.
- **Enhanced flexibility:** GraphQL's adaptability makes it simple to add new features to APIs without negatively affecting current users by letting them choose the specific data they desire to receive.
- **Improved developer experience:** Thanks to GraphQL's advanced type system, APIs are easier to learn and use. It also includes a powerful suite of tools for producing consistent results throughout development.
- **Better performance:** GraphQL can improve API performance by lowering the number of queries used to fetch data.
- **Platform agnostic:** GraphQL is platform agnostic to the data sources you use to read and write data.

Additionally, you can take advantage of platform agnostic tools to query data in GraphQL.

Despite these benefits, there are certain downsides to using GraphQL:

- **Complexity:** Setting up and managing GraphQL takes more time and resources than setting up and managing REST APIs.
- **Learning curve:** Developers may need time and money to learn GraphQL because they have to understand new ideas and syntax.
- **Caching:** Because GraphQL lets clients ask for only the information they need, caching might be harder than with REST APIs.
- **Over fetching/under fetching:** Although GraphQL can help stop REST APIs from over fetching (getting more data than needed), it can also make the problem of under fetching worse, which is when clients don't get all the data they want in one query.

## What Is a Schema in GraphQL?

In GraphQL, a schema describes the data your clients can request from a GraphQL API, including support for validation, parsing, automatic code generation, introspection, and type safety. You can specify the data you need (i.e., ID, firstName, lastName, etc.) in the GraphQL schema.

A schema is comprised of a collection of GraphQL types used to describe the structure of your data. A schema defines a contract between the client and the server and describes how the client can request the data from the server. A GraphQL schema typically contains two special types: query and mutation.

```
schema {
  query: Query
  mutation: Mutation
}
```

### Supported Types in a GraphQL Schema

Here are the supported types in a GraphQL schema

- Object type
- Scalar type
- Query type
- Mutation type

The following is an example of an Object type:

```
type Author {
```

GraphQL provides support for the following built-in scalar types:

- Int
- Float
- String
- Boolean
- ID

The code snippet below illustrates how a scalar type is defined in GraphQL:

```
type Author {  
  firstname: String!  
  lastname: String!  
}
```

### Queries and Mutations in GraphQL

A GraphQL query can be either of the following:

- A Read operation accomplished using Query to retrieve or fetch data
- A Write operation accomplished using mutation to create, update, or delete data

## What Are Resolvers?

Resolvers in GraphQL are analogous to query handlers that can generate responses to the queries and mutations executed by the server. The resolver is defined as part of the GraphQL schema and is responsible for fetching, processing, and transforming the data.

It should be noted that each field in a GraphQL schema has its resolver. Most importantly, you can use any supported language to build it. The GraphQL server takes advantage of resolver routines to retrieve data from a database or API efficiently.

## The GraphQL Architectural Components

GraphQL is made up of the following architectural elements:

- **Schema:** GraphQL employs a schema, which is defined using the GraphQL Schema Definition Language, to identify the sorts of data that may be requested and updated. Before queries and updates are done, this schema is used to verify them.
- **Queries and Mutations:** GraphQL supports two core operations: queries and mutations. The former obtains information from a server, whereas the latter can alter data on the server. Before the server executes API requests, they are verified for accuracy. When consumers submit API queries, they are validated for accuracy before being performed.
- **Resolvers:** A resolver is a component that governs how a query or mutation interacts with a specific data source by getting and transmitting data at the appropriate time.
- **Execution Engine:** An execution engine is in charge of processing queries and data changes against a schema. When the execution engine gets a query, it compares it to the schema and runs the necessary resolvers to generate the desired data.
- **Type System:** The extensive type system of GraphQL provides for the design of the API's data structure, guaranteeing that the API is legitimate and that queries are processed fast and without mistakes.
- **Client Libraries:** GraphQL client libraries are used to let clients and APIs communicate by delivering requests and updates, caching data, and addressing problems as they occur.
- **Tools and Frameworks:** GraphQL comes with a plethora of tools and frameworks, including code generators, testing frameworks, and middleware.

Figure 1 illustrates the GraphQL architectural components.

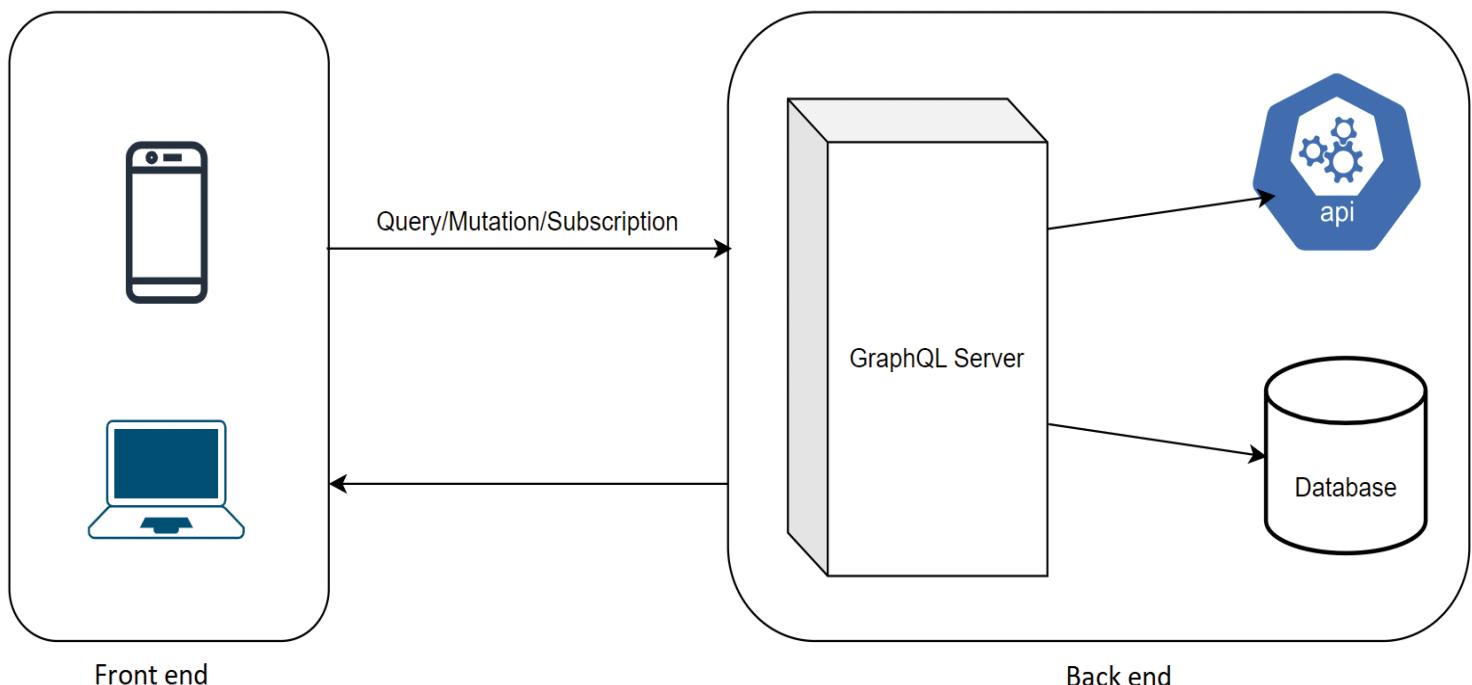


Figure 1: The GraphQL architectural components

## How Does GraphQL Work?

The server processes a GraphQL query sent by a client in three distinct phases: parsing, validation, and execution.

1. The initial processing of a GraphQL query involves parsing it into an abstract syntax tree (AST). The query is represented in the AST as a tree of nodes, which the server may then process. As part of the parsing process, the query's syntax is validated to make sure it conforms to GraphQL's syntax.
2. The server checks the query against the schema once it has been parsed into an AST. Schemas are used to define the various types of data and their relationships. By checking that the query satisfies the schema, the validation process guarantees that it's semantically correct and hence executable.
3. The server resolves each field by traversing the AST repeatedly while processing the query. Resolvers find and retrieve data from specified data sources (e.g., databases or REST APIs). The server generates the response while executing resolver functions.

After processing the query is complete, the server sends the response to the client or the consumer. It should be noted that the responses contain not only data but also details of errors (if any) that were encountered.

GraphQL also supports mutations, which are operations that can modify data on the server and are handled differently than queries and have their own syntax. You can consider mutations in GraphQL similar to POST, PUT, PATCH, and DELETE in REST.

Each GraphQL query is sent to the server through this same cycle of parsing, validation, and execution, as shown in **Figure 2**.

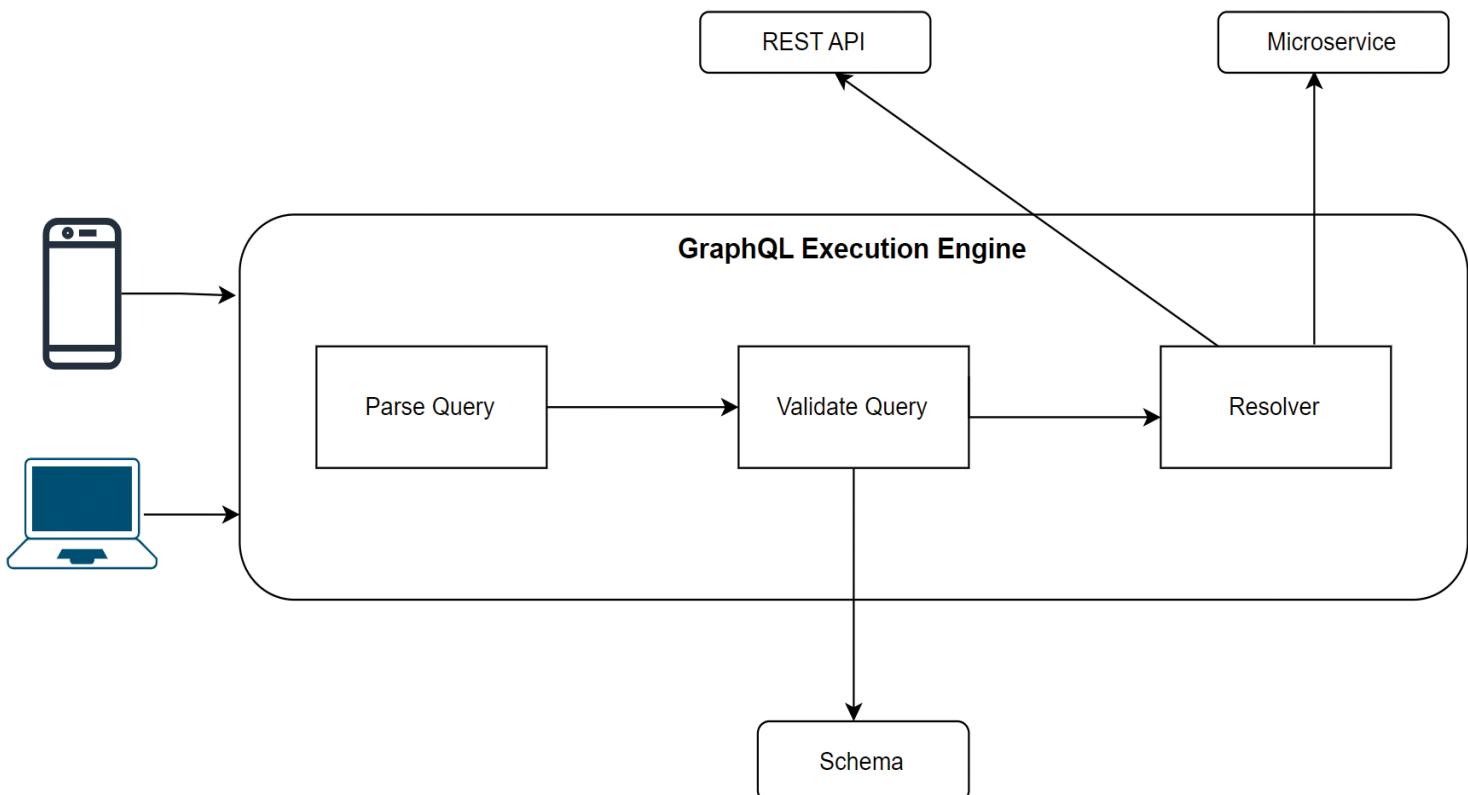
GraphQL also provides support for mutations, which are operations that alter data on the server. It should be noted that mutations are executed in a different way from queries and use a distinct syntax. When a mutation is executed, an updated copy of the data is included in the returned JSON object.

## The Problem of Under-Fetching and Over-Fetching

When working with APIs you might often come across challenges such as over-fetching and under-fetching. Over fetching is defined as a situation in which an API returns more information than is necessary for your application. For instance, a client might request Order ID and Order Date and receive Order ID, Order Date, and Product ID instead.

Under fetching occurs when an API doesn't provide all the data your application requests. A client may request Order ID and Order Date, but only receive Order ID. As a consequence, the client may experience reduced performance and improper or inefficient use of memory, CPU, and network resources.

Because GraphQL enables users to query for only the information they need, it helps in elegantly resolving these difficulties. If your API is adept at returning all the data your application needs, you'll not encounter any over-fetching or under-fetching issues.



**Figure 2:** The GraphQL execution engine

DEVELOPERWEEK

# CLOUDX

80+ Speakers

Aug 15-16 San Mateo, CA

Aug 22-23 Live Online

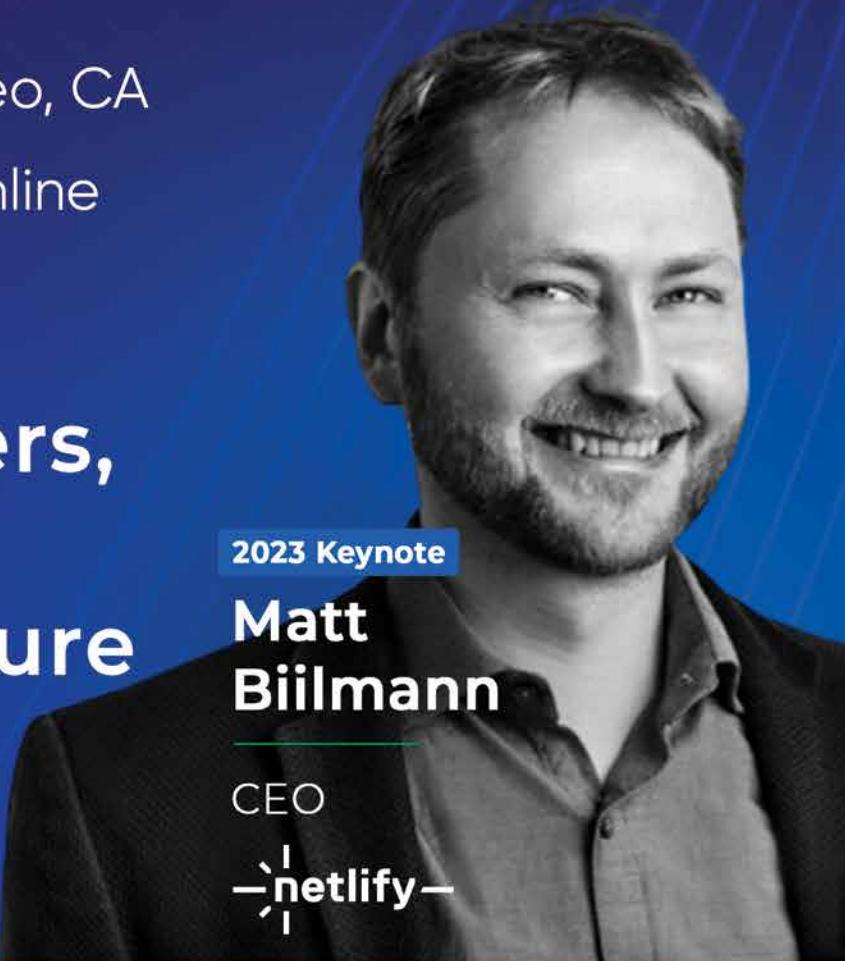
Join 3,500  
Cloud developers,  
architects,  
IT & infrastructure  
professionals  
& executives

2023 Keynote

Matt  
Biilmann

CEO

— netlify —



Use code **MP10B** for **\$100** off any conference. Register at [developerweek.com](https://developerweek.com)

## DEVELOPERWEEK GLOBAL SERIES

6 events. 5,000+ companies. 25,000+ annual attendees.

DEVELOPERWEEK  
EUROPE

April 2024

DEVELOPERWEEK

Feb 21-23, 2024

SF Bay Area

Feb 27-29, 2024

Virtual

DEVELOPERWEEK  
LATIN AMERICA

June 2024

DEVELOPERWEEK  
ENTERPRISE

November 15-16, 2023

DEVELOPERWEEK  
MANAGEMENT

May 2024

DEVELOPERWEEK  
CLOUDX

August 2024

## GraphQL vs. REST

GraphQL and REST are two different architectural approaches to building APIs, each with advantages and disadvantages. REST, or Representational State Transfer, is a style of software architecture that uses HTTP to communicate between clients and servers. The term REST refers to a type of architecture that emphasizes the usage of HTTP protocols for interaction between clients and servers.

Here are a few important distinctions between GraphQL and REST:

- A REST API takes advantage of "resources," or URLs, to represent data and HTTP methods like GET, POST, PUT, and DELETE to work with this data. Although REST APIs can be efficient, they can also suffer from over-fetching or under-fetching data, resulting in unnecessary data transfer.
- Although REST works only with HTTP protocol, there are no protocol constraints in GraphQL. In other words, GraphQL is agnostic of the transport layer—you can use it with any transport layer protocol.
- GraphQL can provide an elegant way to retrieve data from an API. Rather than using multiple requests and endpoints like REST, GraphQL APIs use a single endpoint and a GraphQL query language to retrieve only the needed data. This can lead to fewer data transfers and more efficient API performance.
- Another important distinction between GraphQL and REST is the existence of a schema. GraphQL features a sophisticated type system that specifies the structure of data sent by the API, which can guarantee that queries are valid and efficient. REST APIs lack a formal schema definition, which makes them more flexible but also more challenging to deal with.
- Caching is yet another important point to consider. REST APIs can use HTTP caching techniques to minimize the data sent between clients and servers. Native caching mechanisms, on the other hand, are not supported by GraphQL APIs.

## GraphQL vs. gRPC

GraphQL and gRPC are both modern approaches to building APIs, but they have different design goals and use cases. GraphQL is a query language that provides a flexible and efficient way to request and retrieve data from an API. It's designed to provide a single endpoint that can be used to retrieve all the data needed by a client in a single request.

GraphQL APIs are often used for web applications, mobile applications, and microservices architectures. On the contrary, gRPC is an open-source framework built by Google for creating remote procedure call APIs. It uses Protocol Buffers, a language- and platform-neutral serialization format, to define service interfaces and message types.

Here are some important distinctions between GraphQL and gRPC:

- **Query Language vs. Service Definition:** GraphQL offers a query language for retrieving data from the API, and gRPC provides a service definition for defining the methods and messages accessible on the API.

- **Data Transport:** GraphQL uses HTTP as its data transport, but gRPC employs HTTP/2, a binary protocol that offers a variety of performance and efficiency advantages.
- **Type System:** GraphQL has a comprehensive type system that specifies the structure of data delivered by the API. gRPC includes a robust type system as well, but it's built on Protocol Buffers and is intended to be language- and platform-agnostic.
- **Tooling:** Although GraphQL encompasses a rich ecosystem of libraries, tools, server frameworks, and code generators, gRPC is still in its infancy in comparison to GraphQL.

## The GraphQL Schema Definition Language (SDL)

The Schema Definition Language (SDL) represents the schema of a GraphQL API in a concise manner using a simple syntax and defines the following:

- The type of data you can query
- The relationships between those types
- The actions you can perform on them

The GraphQL SDL has a collection of built-in scalar types including String, Int, and Boolean, as well as the option to build new types. Fields representing data of other kinds may be used to create custom types.

- **Fields:** Fields are used to express data on a type that can be queried. Arguments to fields may be used to filter or change the data provided by the field.
- **Queries and Mutations:** Queries and mutations are used on GraphQL APIs to obtain and alter data. Queries are "read-only" operations used to retrieve data, while mutations are "write" operations used to change data.
- **Directives:** Directives are used to supplement a schema's metadata. They may be used to change the behavior of queries and modifications, as well as to provide extra documentation.

## Building GraphQL Applications: The Schema-First and Code-First Approaches

A GraphQL server requires two artifacts: a schema, which defines all types and fields; and resolvers, a collection of functions that are invoked to return data for those types. There are two approaches used to build GraphQL applications. These include the schema-first and code-first approaches. Although the former designs the schema up front, the latter uses code to build the schema programmatically.

### The Schema-First Approach

Here's an example of simple schema definition:

```
type Author
{
  id: Int!
  firstname: String!
  lastname: String!
  email: String!
```

```
  books: [Book!]!
}
```

In this example, the Author object has an ID, firstname, lastname, and email field, each with a specific scalar type (Int! and String!). The books field is an array of Book objects.

The following is an example of a typical query:

```
query{
  authors
  {
    id
    firstname
    lastname
  }
}
```

### The Code-First Approach

Using the code-first or resolver-first approach, you begin by coding the resolvers and then generate the schema from the code as a single source of truth as an artifact. Consequently, you still have a schema, but it's now generated automatically by a script instead of being created manually.

The Code-First approach is often used when you're using a typed programming language such as C# to build GraphQL applications. In this approach, you code the resolvers first and then, assuming code as the single source of truth, the schema is generated thereafter. I'll be using this approach throughout this article.

## Where Does GraphQL Fit in the Technology Stack?

GraphQL may thrive in a number of different places in the technological stack, depending on the use case and the requirements at hand. Clients may use GraphQL to request and retrieve data from an API in a flexible and efficient manner because it acts as a layer between them and the server. By acting as a mediator between the client and the server, GraphQL makes it possible for clients to make more elegant and efficient API requests.

GraphQL can fit in any of the following layers of the tech stack:

- **API Layer:** GraphQL may be used as an application programming interface (API) layer between a client and a server, facilitating centralized data retrieval and querying. Then the GraphQL server may offer the information in the format requested by mapping the queries to the appropriate data sources or services.
- **Server-side framework:** To create server-side APIs, GraphQL may be used as a server-side framework. Several modern languages have support for GraphQL server-side framework.
- **Client-side framework:** You can use GraphQL as a framework for creating applications on the client side that can access data using a GraphQL API.

GraphQL may function as a layer of abstraction for data between your app, database, and other sources. Here, the GraphQL server translates the client's requests into SQL queries against a database and provides the results.

## Error Handling in GraphQL

Contrary to errors in a REST-based application, you no longer have to bank on status codes and status text messages in GraphQL. It should be noted that GraphQL has a built-in error handling mechanism that allows servers to return informative error messages to clients in case of errors.

Typically, any GraphQL response should contain a data field and/or an errors field. In the event of an error, the standard error-handling mechanism in GraphQL returns JSON data that corresponds to the GraphQL operation, such as query, mutation, and subscription, together with an array of error objects with the message and location.

The error object in GraphQL gives specific information on any problems that may have arisen while an operation was being carried out. These problems might have been caused by a number of factors. In order to provide more in-depth information on an issue, a response may include an error object.

An error object in GraphQL is typically comprised of the following:

- **Message:** A message field is a required string with an error message that people can read.
- **Locations:** The locations field is an optional list of source locations that show where in the query the problem happened. There can be both line and column fields in a location field.
- **Path:** The path field contains the names of the fields in the GraphQL response where the error occurred.
- **Extensions:** This is an optional field that may give more details or metadata pertaining to the error, such as a custom error code or other error details.

Here is an error object with all these fields:

```
{
  "message": "The input is
not a valid
email address.",
  "locations": [
    {
      "line": 5,
      "column": 8
    }
  ],
  "path": [
    "createEmployee",
    "email"
  ],
  "extensions": {
    "code": "INPUT_ERROR",
    "details": {
      "email": "Please provide an
      email address in
      the correct format."
    }
  }
}
```

When this mutation named `createEmployee` is executed and a wrong email address or an invalid email address is specified, an error occurs and the appropriate error message is returned as part of the response. In the preceding code snippet, the `extensions` column is used to store a user-defined error code along with additional information related to the error. If you can provide specific error information, it will be easier to detect, debug, and resolve any errors that might occur when a GraphQL operation is in execution.

## Building a Simple Application Using GraphQL

It's time for writing some code. Let's now examine how to build a simple ASP.NET Core 7 Web API application using GraphQL.

### Create a New ASP.NET Core 7 Project in Visual Studio 2022

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 Core 7 Project in Visual Studio 2022:

1. Start the Visual Studio 2022 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 `GraphQLDemo` 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. 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. Because you won't be using minimal APIs in this example, remember to leave the **Use controllers (unchecked to use minimal APIs)** checkbox checked.
7. Click Create to complete the process.

Because the support for GraphQL isn't built in ASP.NET Core, you need to install it as a package from NuGet via NuGet Package Manager.

### Hot Chocolate

Hot Chocolate is an open-source, high-performance, versatile, and flexible framework for building GraphQL APIs using .NET Core. It enables developers to define GraphQL schema, connect to various data sources, and offer clients a reliable and sustainable API. Hot Chocolate fully supports GraphQL features like mutations, queries, and subscriptions. Hot Chocolate comes with all the tools and features you need to create a reliable, easy-to-maintain GraphQL API, regardless of whether it's for a single application or a complex system.

Hot Chocolate also supports advanced features like interfaces, unions, and enums, which is an advantage. The type system of Hot Chocolate allows defining GraphQL schema

using strongly typed .NET classes, which improves API reliability and maintainability while providing better support for tools like refactoring and IntelliSense.

Hot Chocolate provides an array of advanced tools for constructing robust GraphQL APIs. It incorporates capabilities such as caching and batching to optimize functionality and middleware to enable the personalization of GraphQL resolvers. Hot Chocolate enables you to define your GraphQL schema through a straightforward and user-friendly API and encompasses numerous resources that can assist you in testing and debugging your API. One of these is the GraphQL Playground, which allows you to engage within your API and logging and tracing tools to identify any problems during production.

In the application you're about to make, you'll take advantage of HotChocolate to generate GraphQL schemas. With HotChocolate, you can build an extra layer on top of your application layer that uses GraphQL. It's easy to set up and configure, and it eliminates the clutter of generating schemas.

### Install NuGet Package(s)

So far so good. The next step is to install the necessary NuGet Package(s). To install the required packages into your project, right-click on the solution, and then select Manage NuGet Packages for Solution.... Now search for the packages named `HotChocolate.AspNetCore` and `HotChocolate.AspNetCore.Playground` in the search box and install them one after the other. Alternatively, you can type the commands shown below at the NuGet Package Manager Command Prompt:

```
PM> Install-Package  
HotChocolate.AspNetCore  
PM> Install-Package  
HotChocolate.AspNetCore.Playground
```

You can also install these packages by executing the following commands at the Command Window:

```
dotnet add package  
HotChocolate.AspNetCore  
dotnet add package  
HotChocolate.AspNetCore.Playground
```

### Create the Supplier Class

Create a new class named `Order` in a file having the same name with a `.cs` extension and write the following code in there:

```
public class Supplier  
{  
    public int Id { get; set; }  
    public string FirstName { get; set; }  
    public string LastName { get; set; }  
    public string Address { get; set; }  
    public string Email { get; set; }  
    public string Phone { get; set; }  
}
```

The other entity classes aren't being shown here for brevity and also because this is a minimalistic implementation to illustrate how you can work with GraphQL in ASP.NET Core 7.

### Listing 1: The Supplier Repository class

```
public class SupplierRepository : ISupplierRepository
{
    private readonly List<Supplier> suppliers = new List<Supplier>
    {
        new Supplier
        {
            Id = 1,
            FirstName = "Steve",
            LastName = "Smith",
            Address = "USA",
            Phone = "1111111111",
            Email = "test@xyz.com"
        },
        new Supplier
        {
            Id = 2,
            FirstName = "Michael",
            LastName = "Foster",
            Address = "USA",
            Phone = "2222222222",
            Email = "abc@test.com"
        },
        new Supplier
    };
}
```

```

    {
        Id = 3,
        FirstName = "Anand",
        LastName = "Shah",
        Address = "India",
        Phone = "9999999999",
        Email = "xyz@abc.com"
    };
}

public async Task<List<Supplier>> GetSuppliers()
{
    return await Task.FromResult(suppliers);
}

public async Task<Supplier> GetSupplier(int Id)
{
    return await Task.FromResult(
        suppliers.FirstOrDefault(
            (x => x.Id == Id)));
}
```

### Create the ISupplierRepository Interface

Create a new .cs file named ISupplierRepository in your project and replace the default generated code with the following code snippet:

```
public interface ISupplierRepository
{
    public Task<List<Supplier>> GetSuppliers();
    public Task<Supplier> GetSupplier(int Id);
}
```

### Create the SupplierRepository Class

Now, create a new class named SupplierRepository in a file having the same name with a .cs extension. Next, write the following code in there:

```
public class SupplierRepository : ISupplierRepository
{
}
```

The SupplierRepository class illustrated in the code snippet below implements the methods of the ISupplierRepository interface:

```
public async Task<List<Supplier>> GetSuppliers()
{
    return await Task.FromResult(suppliers);
}

public async Task<Supplier> GetSupplier(int Id)
{
    return await Task.FromResult(
        suppliers.FirstOrDefault(
            (x => x.Id == Id)));
}
```

### Listing 2: The Supplier GraphQL Query class

```
using HotChocolate.Subscriptions;

namespace GraphQLDemo
{
    public class SupplierGraphQLQuery
    {
        public async Task<List<Supplier>> GetAllSuppliers([Service] ISupplierRepository supplierRepository, [Service] ITopicEventSender eventSender)
        {
            List<Supplier> suppliers = await supplierRepository.GetSuppliers();
            await eventSender.SendAsync("Returned a list of Suppliers", suppliers);
            return suppliers;
        }
    }
}
```

### Listing 3: The Supplier Type class

```
namespace GraphQLDemo
{
    public class SupplierType : ObjectType<Supplier>
    {
        protected override void Configure(IObjectTypeDescriptor<Supplier> descriptor)
        {
            Descriptor.Field(a => a.Id).Type<IdType>();
            descriptor.Field(a => a.FirstName).Type<StringType>();
            descriptor.Field(a => a.LastName).Type<StringType>();
        }
    }
}
```

#### **Listing 4:** The Supplier Subscription class

```
using HotChocolate.Execution;
using HotChocolate.Subscriptions;

namespace GraphQLDemo
{
    public class SupplierSubscription
    {
        [SubscribeAndResolve]
        public async ValueTask<ISourceStream<List<Supplier>>> OnSuppliersGet([Service] ITopicEventReceiver eventReceiver,
            CancellationToken cancellationToken)
        {
            return await eventReceiver.SubscribeAsync<List<Supplier>>(
                "Returned Suppliers",
                cancellationToken);
        }
    }
}
```

The complete source code of the SupplierRepository class is given in [Listing 1](#).

#### *Register the SupplierRepository instance*

The following code snippet illustrates how an instance of type ISupplierRepository is added as a scoped service to the IServiceCollection.

```
builder.Services.AddScoped<ISupplierRepository, SupplierRepository>();
```

#### *Create the GraphQL Query Class*

Create a new .cs file named SupplierGraphQLQuery in your project and replace the default generated code with the code given in [Listing 2](#).

#### *Create the GraphQL Object Type*

In GraphQL, Object Types are used to describe the type of data that are fetched using your API, and they're represented by creating a class that derives from the GraphQL.Types.ObjectGraphType class. Create a new file named SupplierType.cs in your project and replace the default code with the code given in [Listing 3](#).

#### *Create a GraphQL Subscription Type*

You should also create a subscription to enable your GraphQL server to notify all subscribed clients when an event occurs. Create a new class named SupplierSubscription and write the piece of code given in [Listing 4](#) in there.

#### *Configure GraphQL Server in ASP.NET Core*

Once you've created the Query type to expose the data you need, configure GraphQL Server in the Program.cs file using the following code snippet:

```
builder.Services.
    AddGraphQLServer()
    .AddType<SupplierType>()
    .AddQueryType<SupplierGraphQLQuery>()
    .AddSubscriptionType<Subscription>()
    .AddInMemorySubscriptions();
```

You can then call the MapGraphQL method to register the middleware:

```
app.MapGraphQL();
```

#### **Listing 6:** The Program.cs file

```
using GraphQLDemo;
using HotChocolate.AspNetCore;
using HotChocolate.AspNetCore.Playground;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.

builder.Services.AddScoped<ISupplierRepository, SupplierRepository>();

builder.Services.AddGraphQLServer()
    .AddType<SupplierType>()
    .AddQueryType<Query>()
    .AddSubscriptionType<Subscription>()
```

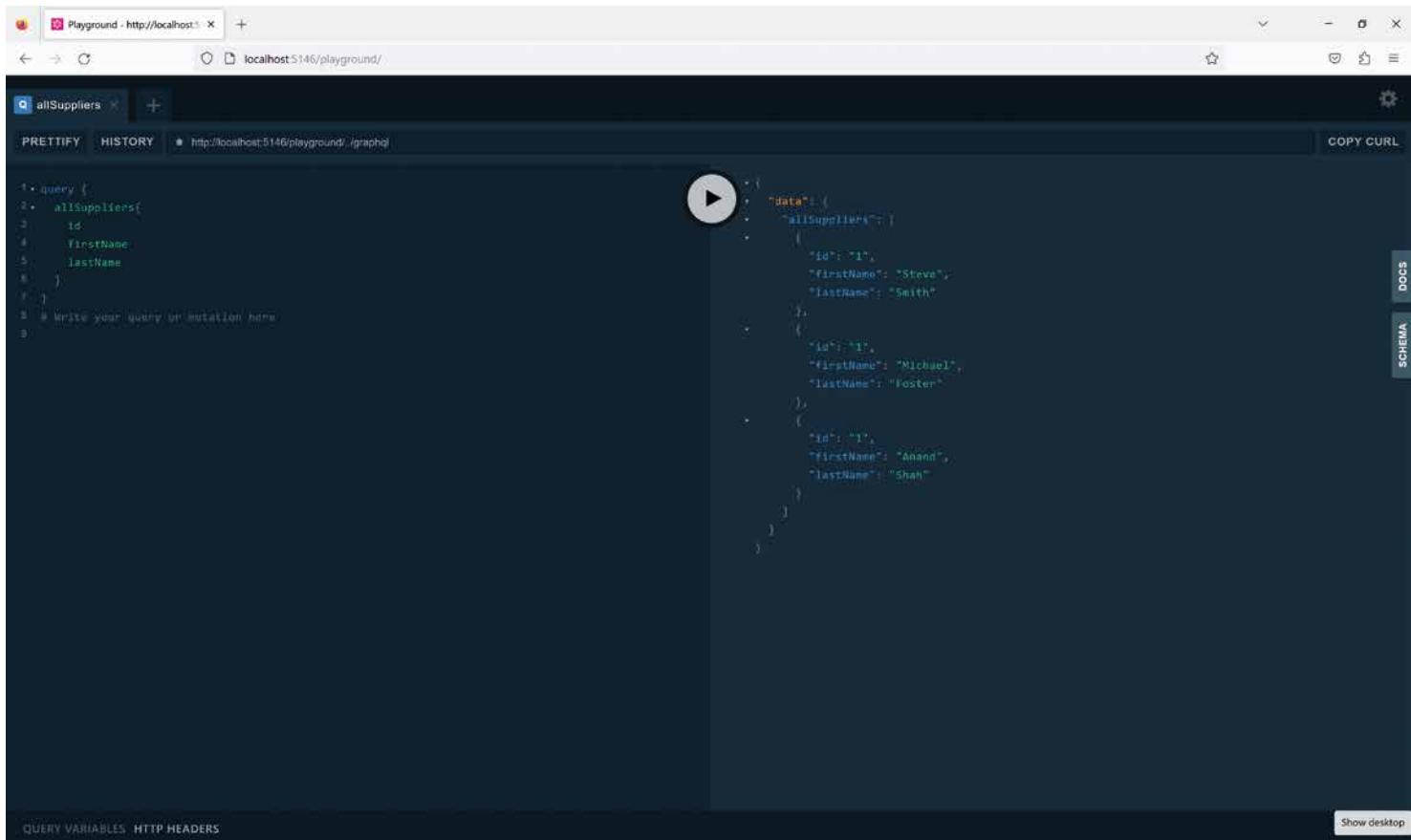
```
.AddInMemorySubscriptions();

builder.Services.AddControllers();
var app = builder.Build();

// Configure the HTTP request pipeline.

app.UseAuthorization();
app.MapControllers();
app.UsePlayground(new PlaygroundOptions
{
    QueryPath = "/graphql",
    Path = "/playground"
});

app.MapGraphQL();
app.Run();
```



**Figure 3:** GraphQL in action: displaying all supplier records

When you register this middleware, the GraphQL server is available at `/graphql` by default. You can also customize the endpoint where the GraphQL server will be hosted by specifying the following code in the `Program.cs` file:

```
app.MapGraphQL("/graphql/mycustomendpoint");
```

#### Create the SupplierController Class

Create a new API Controller in your project named `SupplierController` and write the code given in **Listing 5** in there. **Listing 6** shows the complete source of the `Program.cs` file.

## Execute the Application

When you run the following GraphQL query, the supplier data is displayed as shown in **Figure 3**.

```
query {
  allSuppliers{
    id
    firstName
    lastName
  }
}
```

To test the subscription, you can use the following GraphQL query:

```
subscription{
  onSuppliersGet
```

```
{
  id
  firstName
  lastName
}
```

Upon execution of the preceding query, the `OnSuppliersGet` event is triggered and the data is displayed in the web browser where you executed the subscription query, as shown in **Figure 4**.

Now that you know how to build applications using GraphQL in ASP.NET Core, you'll use this knowledge to create a microservices application that leverages GraphQL to query data. Before I delve deep into this implementation, let's learn the basic concepts related to microservice architecture.

## Introduction to Microservices Architecture

Microservice architecture is a conglomerate of independent, self-contained, interoperable services that can be deployed independently and use a wide range of technologies. A microservice architecture is designed for decentralization. Therefore, it's important to develop strategies that protect microservices-based systems from failures and make them fault-tolerant.

A microservices-based application is comprised of loosely connected services developed and deployed separately

The screenshot shows the GraphQL playground interface. In the top-left, there's a code editor window titled "untitled 1 - Banana Cake Pop" containing the following GraphQL query:

```

subscription {
  onSuppliersGet {
    id
    firstName
    lastName
  }
}

```

In the top-right, the URL is "localhost:5146/graphq/" and the status bar shows "localhost:5146" with a 200 status, 13 ms duration, and 179 B size.

Below the code editor, the "Operations" tab is selected. On the right, the "Response" tab is active, displaying the JSON response from the server:

```

1 {
  2   "data": {
  3     4     "allSuppliers": [
  5       6       {
  7         8         "id": "1",
  9         "firstName": "Steve",
 10         "lastName": "Smith"
 11       },
 12       13       {
 14         15         "id": "1",
 16         "firstName": "Michael",
 17         "lastName": "Foster"
 18       },
 19       20       {
 21         22         "id": "1",
 23         "firstName": "Anand",
 24         "lastName": "Shah"
 25       }
 26     ]
 27   }
 28 }

```

At the bottom, there are sections for "Variables" and "HTTP Headers". The "Variables" section has one entry: "subscription". The "HTTP Headers" section has one entry: "query".

**Figure 4:** The OnSuppliersGet event is triggered.

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

The key advantages of microservices architecture are:

- **Improved Scalability:** Scaling a microservice-based application requires scaling only certain components. This significantly optimizes resource utilization. In addition, microservices can be modified independently of each other, so that there is no interference with other services.
- **Reduced coupling:** Reduced coupling between the components of an application makes it easily adaptable for changes over time.
- **Better ROI:** Microservice 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:** The use of microservice architecture enables teams to work together on independent services and deploy them faster. Microservice architecture reduces development time because it's loosely coupled and autonomous.

- **Faster Releases:** With microservices, services are loosely coupled, so you don't have to rebuild your entire code base when you want to add or change a feature. The decentralized nature of developing, maintaining, and updating applications allows faster deployments and releases. You can treat each microservice as an independently deployable unit.

## Building a Microservices Application Using GraphQL

It's time to write some code. Let's now examine how to build a simple ASP.NET Core 7 Web API application using GraphQL. In this example, you'll examine how to build a simple order management application using GraphQL in ASP.NET Core.

### A Real-World Use Case

A typical Order Processing System is composed of several entities, such as Supplier, Order, Product, Customer, etc. To keep things simple, this application will display one or more order records.

Let's examine how to create an ASP.NET Core 7 project in Visual Studio 2022.

### Create a New ASP.NET Core 7 Project in Visual Studio 2022

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.

### Listing 7: The Model classes

```
public class Product
{
    public int Id
    { get; set; }
    public string Name
    { get; set; }
    = string.Empty;
    Public string Description
    { get; set; }
    = string.Empty;
    public decimal Price
    { get; set; }
}

public class Customer
{
    public int Id
    { get; set; }
    public string FirstName
    { get; set; }
    = string.Empty;
    public string LastName
    { get; set; }
    = string.Empty;
    public string Address
    { get; set; }
    = string.Empty;
    public string City
    { get; set; }
    = string.Empty;
    public string PostalCode
    { get; set; }
    = string.Empty;
    public string Country
    { get; set; }
    = string.Empty;
    public string Phone
    { get; set; }
    = string.Empty;
    public string EmailAddress
    { get; set; }
    = string.Empty;
}
```

```
    { get; set; }
    = string.Empty;
}

public class Order
{
    public int Id
    { get; set; }
    public string ShipAddress
    { get; set; }
    public string ShipCity
    { get; set; }
    public string ShipPostalCode
    { get; set; }
    public string ShipCountry
    { get; set; }
    public DateTime OrderDate
    { get; set; }
    public OrderDetails
    OrderDetails
    { get; set; }
}

public class OrderDetails
{
    public int Id
    { get; set; }
    public int OrderId
    { get; set; }
    public int ProductId
    { get; set; }
    public int CustomerId
    { get; set; }
    public decimal UnitPrice
    { get; set; }
    public int Quantity
    { get; set; }
    public decimal Amount
    { get; set; }
}
```

### Listing 8: The Repository interfaces

```
public interface IProductRepository
{
    public Task<List<Product>>
    GetProducts();
}

public interface ICustomerRepository
{
    public Task<List<Customer>>
    GetCustomers();
}

public interface IOrderRepository
{
    public Task<Order>
    GetOrder(int orderId);
    public Task<List<Order>>
    GetOrders();
}
```

To create a new ASP.NET Core 7 Project in Visual Studio 2022:

1. Start the Visual Studio 2022 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 GraphQLDemo 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. 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.

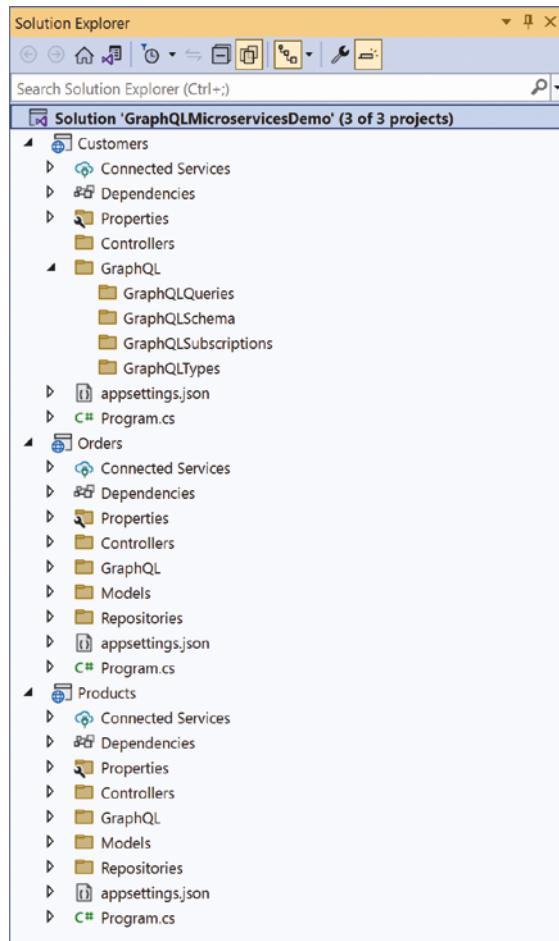
6. Because you’ll not be using minimal APIs in this example, remember to leave the **Use controllers (unchecked to use minimal APIs)** checkbox checked.
7. Click Create to complete the process.

Because the support for GraphQL isn’t built in ASP.NET Core, you need to install it as a package from NuGet via NuGet Package Manager.

In this application, you’ll take advantage of Hot Chocolate to generate GraphQL schemas. With Hot Chocolate, you can build an extra layer on top of your application layer that uses GraphQL. It’s easy to set up and configure, and it eliminates the clutter of generating schemas.

#### Install NuGet Package(s)

So far so good. The next step is to install the necessary NuGet Package(s). To install the required packages into your



**Figure 5:** The initial Solution structure

project, right-click on the solution and the select Manage NuGet Packages for Solution.... Now search for the packages named HotChocolate.AspNetCore, and HotChocolate.AspNetCore.Playground in the search box and install them one after the other. Alternatively, you can type the commands shown below at the NuGet Package Manager Command Prompt:

```
PM> Install-Package
HotChocolate.AspNetCore
PM> Install-Package
HotChocolate.AspNetCore.Playground
```

You can also install these packages by executing the following commands at the Command Window:

```
dotnet add package
HotChocolate.AspNetCore
dotnet add package
HotChocolate.AspNetCore.Playground
```

#### Create the Solution Structure

Create three WebAPI projects namely, Customers, Products, and Orders, in the solution you created earlier. You'll also create solution folders to organize the files in each of the projects. **Figure 5** shows how the initial solution structure would look.

In the sections that follow, you'll create classes and interfaces in each of these projects. In this example, you'll

consider how these projects correspond to the Products, Customers, and Orders microservices.

#### Create the Model Classes

Create the model classes as shown in **Listing 7** in the respective projects. You should create separate .cs files for each of these entity classes. This is a minimal implementation to illustrate how you can work with GraphQL in ASP.NET Core 7.

#### Create the Interfaces for your Repository Classes

Now create the interfaces for the repository classes, as shown in **Listing 8**.

#### Create the Repository Classes

Now create the repository classes, as shown in **Listing 9**, by implementing the interfaces you created in the preceding section.

#### Register the Repository Instances

Write the following code to add the repository instances as a scoped service to the IServiceCollection:

```
builder.Services.AddScoped
<IPrductRepository,
ProductRepository>();
builder.Services.AddScoped
<ICustomerRepository,
CustomerRepository>();
builder.Services.AddScoped
<IOrderRepository,
OrderRepository>();
```

#### Create the Controller Classes

Create the API Controller classes named ProductController, CustomerController, and OrderController and write the code shown in **Listing 10** in there.

For the sake of simplicity, you'll implement the Orders microservice in the following sections and skip the other two microservices, Products and Customers.

#### Create the OrderQuery Class

Create a new .cs file named OrderQuery in the Orders project and replace the default generated code with the code given in **Listing 11**.

#### Create the OrderType Class

In GraphQL, Object Types are used to describe the type of data that are fetched using your API and they're represented by creating a class that derives the GraphQL.Types.ObjectGraphType class. Create a new file named OrderType.cs in your project and replace the default code with the code given in **Listing 12**.

#### Create the OrderSubscription Class

You should also create a subscription to enable your GraphQL server to notify all subscribed clients when an event occurs. Create a new class named OrderSubscription and write the piece of code given in **Listing 13** in there.

#### Revisiting the Solution Structure

Now that you've created the three microservices, the solution structure looks like **Figure 6**.

Note that only the Orders microservice project has been expanded and is shown here for brevity.

### Listing 9: The Repository classes

```

public class ProductRepository :  
IProductRepository  
{  
    private readonly List<Product>  
products = new List<Product>();  
public ProductRepository()  
{  
    products.Add(new Product  
    {  
        Id = 1,  
        Name = "Samsung Mobile",  
        Description = "Android Mobile",  
        Price = 90000  
    });  
  
    products.Add(new Product  
    {  
        Id = 2,  
        Name = "HP Laptop",  
        Description = "i9 Laptop",  
        Price = 65000  
    });  
  
    products.Add(new Product  
    {  
        Id = 3,  
        Name = "Car",  
        Description = "Hyundai Xcent",  
        Price = 55000  
    });  
}  
public Task<List<Product>>  
GetProducts()  
{  
    return Task.FromResult(products);  
}  
}  
  
public class CustomerRepository :  
ICustomerRepository  
{  
    private readonly List<Customer>  
customers =  
new List<Customer>();  
  
public CustomerRepository()  
{  
    customers.Add(new Customer()  
    {  
        Id = 1,  
        FirstName = "Joydip",  
        LastName = "Kanjilal",  
        Address = "Banjara Hills",  
        City = "Hyderabad",  
        PostalCode = "500034",  
        Phone = "1234567890",  
        EmailAddress =  
        "joydipkanjilal@yahoo.com"  
    });  
  
    customers.Add(new Customer()  
    {  
        Id = 2,  
        FirstName = "Samuel",  
        LastName = "Jones",  
        Address = "Calgary, Canada",  
        Phone = "0987654321",  
        EmailAddress = "sjones@gmail.com"  
    });  
}  
public Task<List<Customer>> GetCustomers()  
{  
    return Task.FromResult(customers);  
}  
}  
  
public class OrderRepository :  
IOrderRepository  
{  
    private readonly  
List<Order> orders =  
new List<Order>  
{  
    new Order  
    {  
        Id = 1,  
        ShipAddress = "Banjara Hills",  
        ShipCity = "Hyderabad",  
        ShipPostalCode = "500034",  
        OrderDetails = new OrderDetails  
        {  
            Id = 1,  
            CustomerId = 1,  
            OrderId = 1,  
            ProductId = 1,  
            Quantity = 100,  
            UnitPrice = 50,  
            Amount = 5000.00M  
        },  
        OrderDate = DateTime.Now  
    },  
    new Order  
    {  
        Id = 1,  
        ShipAddress = "Banjara Hills",  
        ShipCity = "Hyderabad",  
        ShipPostalCode = "500034",  
        OrderDetails = new OrderDetails  
        {  
            Id = 1,  
            CustomerId = 1,  
            OrderId = 1,  
            ProductId = 1,  
            Quantity = 100,  
            UnitPrice = 50,  
            Amount = 5000.00M  
        },  
        OrderDate = DateTime.Now  
    },  
    new Order  
    {  
        Id = 1,  
        ShipAddress = "Banjara Hills",  
        ShipCity = "Hyderabad",  
        ShipPostalCode = "500034",  
        OrderDetails = new OrderDetails  
        {  
            Id = 1,  
            CustomerId = 1,  
            OrderId = 1,  
            ProductId = 1,  
            Quantity = 100,  
            UnitPrice = 50,  
            Amount = 5000.00M  
        },  
        OrderDate = DateTime.Now  
    }  
};  
public async  
Task<List<Order>> GetOrders()  
{  
    return await  
    Task.FromResult(orders);  
}  
public async Task<Order>  
GetOrder(int Id)  
{  
    return await Task.FromResult  
(orders.FirstOrDefault  
(x => x.Id == Id));  
}
}

```

### Listing 10: The Controller classes

```
[Route("api/[controller]")]
[ApiController]
public class ProductController : Controller
{
    private IProductRepository _productRepository;
    public ProductController(IProductRepository productRepository)
    {
        _productRepository = productRepository;
    }

    [HttpGet("GetProducts")]
    public async Task<List<Product>> GetProducts()
    {
        return await _productRepository.GetProducts();
    }
}

[Route("api/[controller]")]
[ApiController]
public class CustomersController : ControllerBase
{
    private readonly CustomerRepository _customerRepository;
    public CustomersController(CustomerRepository customerRepository)
    {
        _customerRepository = customerRepository;
    }

    [HttpGet("GetCustomers")]
    public async Task<List<Customer>> GetCustomers()
    {
        return await _customerRepository.GetCustomers();
    }
}

[Route("api/[controller]")]
[ApiController]
public class OrdersController : ControllerBase
{
    private IOrderRepository _orderRepository;
    public OrdersController(IOrderRepository orderRepository)
    {
        _orderRepository = orderRepository;
    }

    [HttpGet("GetOrders")]
    public async Task<List<Order>> GetOrders()
    {
        return await _orderRepository.GetOrders();
    }

    [HttpGet("{id}")]
    public async Task<Order> GetOrder(int id)
    {
        return await _orderRepository.GetOrder(id);
    }
}
```

### Listing 11: The OrderQuery class

```
public class OrderQuery
{
    public async Task<List<Order>> GetAllOrders([Service] IOrderRepository orderRepository, [Service] ITopicEventSender eventSender)
    {
        List<Order> orders = await orderRepository.GetOrders();
        await eventSender.SendAsync("Returned a list of Orders", orders);
        return orders;
    }
}
```

#### Configure GraphQL Server in ASP.NET Core

Once you've created the Query type to expose the data you need, configure GraphQL Server in the Program.cs file using the following code snippet:

```
builder.Services.AddGraphQLServer()
    .AddType<OrderType>()
    .AddQueryType<OrderQuery>()
    .AddSubscriptionType
```

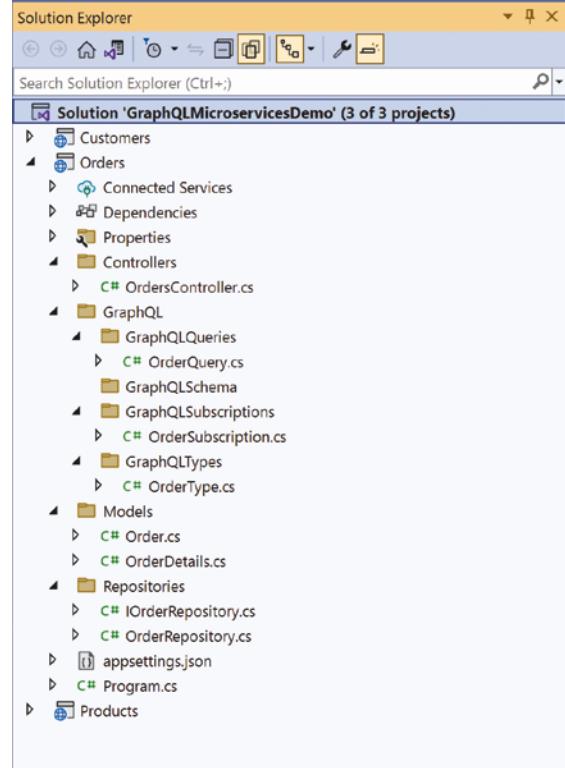


Figure 6: The final Solution structure

### Listing 12: The OrderType class

```
public class OrderType :  
    Object BaseType<Order>  
{  
    protected override void  
    Configure(IObjectTypeDescriptor  
<Order> descriptor)  
    {  
        descriptor.Field  
(a => a.Id).Type<IdType>().  
        Description("Id property.");  
        descriptor.Field  
(a => a.ShipAddress).  
        Type<StringType>().  
        Description  
        ("ShipAddress property.");  
        descriptor.Field(a =>  
  
a.ShipCity).Type<StringType>().  
        Description  
        ("ShipCity property.");  
        descriptor.Field  
(a => a.ShipCountry).  
        Type<StringType>().  
        Description  
        ("ShipCountry property.");  
        descriptor.Field(a =>  
a.ShipPostalCode).  
        Type<StringType>().  
        Description  
        ("ShipPostalCode property.");  
    }  
}
```

```
<OrderSubscription>()  
.AddInMemorySubscriptions();
```

You can then call the MapGraphQL method to register the middleware:

```
app.MapGraphQL();
```

When you register this middleware, the GraphQL server will be available at /graphql by default. You can also customize the endpoint where the GraphQL server will be hosted by specifying the following code in the Program.cs file:

```
app.MapGraphQL("/graphql/orderapiendpoint");
```

### Listing 13: Notify subscribed clients of an event

```
public class OrderSubscription  
{  
    [SubscribeAndResolve]  
    public async  
    ValueTask<ISourceStream  
<List<Order>>> OnOrdersGet([Service]  
    ITopicEventReceiver eventReceiver,  
    CancellationToken cancellationToken)  
    {  
        return await  
        eventReceiver.SubscribeAsync  
<List<Order>>  
        ("Returned Orders",  
        cancellationToken);  
    }  
}
```

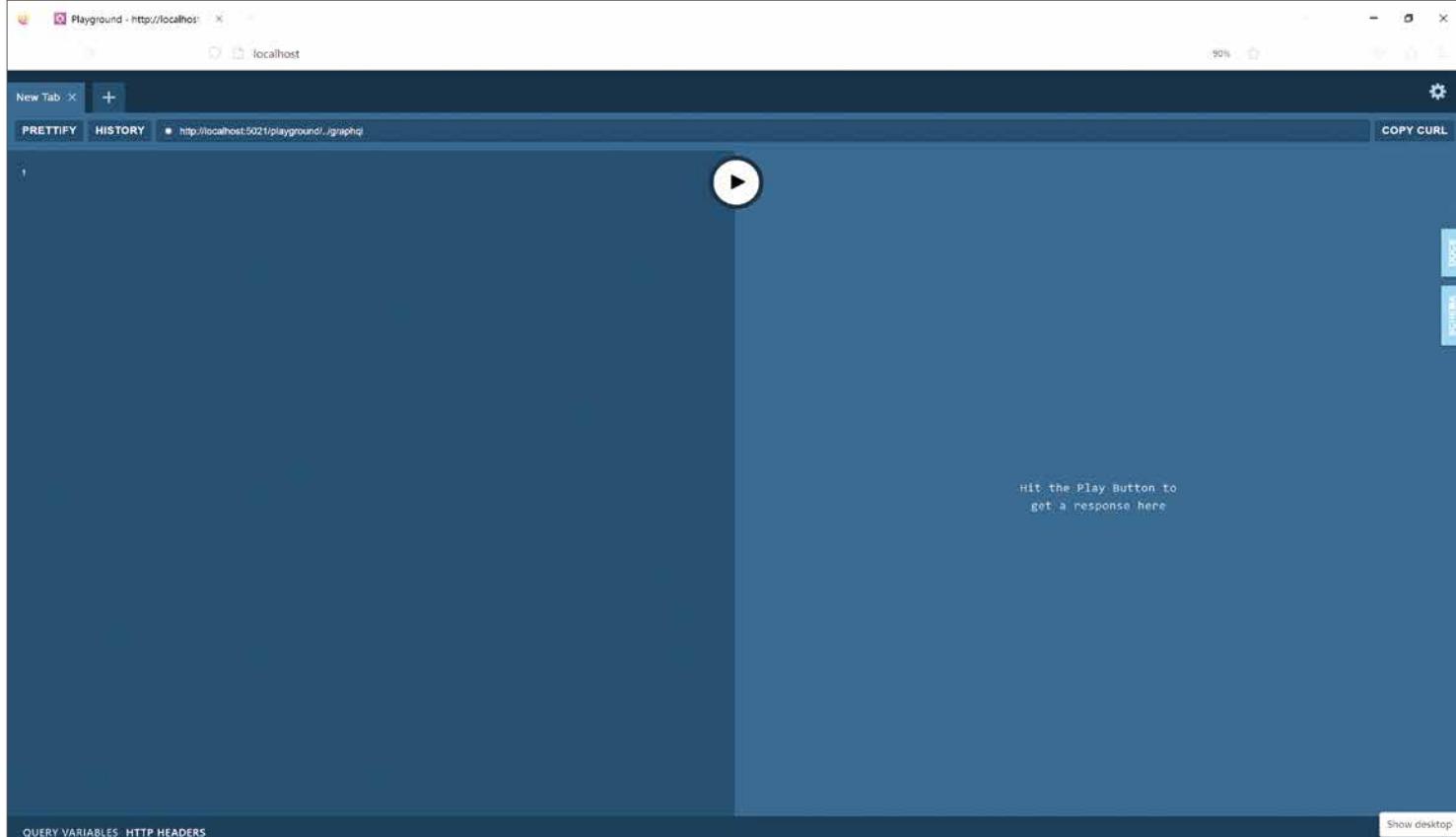


Figure 7: The Playground screen displayed in the web browser without any data

```

1 # Write your query or mutation here.
2 query {
3   allOrders{
4     id
5     shipCountry
6     shipAddress
7     shipPostalCode
8     shipCity
9     orderDetails
10    {
11      quantity
12      unitPrice
13      customerId
14    }
15  }
16 }

```

```

{
  "data": {
    "allOrders": [
      {
        "id": "1",
        "shipCountry": "India",
        "shipAddress": "Banjara Hills",
        "shipPostalCode": "500034",
        "shipCity": "Hyderabad",
        "orderDetails": {
          "quantity": 100,
          "unitPrice": 50,
          "customerId": 1
        }
      },
      {
        "id": "1",
        "shipCountry": "United States",
        "shipAddress": "Kooter Lane",
        "shipPostalCode": "28837",
        "shipCity": "Denver",
        "orderDetails": {
          "quantity": 100,
          "unitPrice": 50,
          "customerId": 2
        }
      },
      {
        "id": "1",
        "shipCountry": "India",
        "shipAddress": "Park Road",
        "shipPostalCode": "560024",
        "shipCity": "Bangalore",
        "orderDetails": {
          "quantity": 100,
          "unitPrice": 50,
          "customerId": 3
        }
      }
    ]
  }
}

```

**Figure 8:** The order data is displayed on execution of the query.

## Executing the Order Management Application Using HotChocolate

When you execute the application and invoke the `HttpGet` endpoint of the Orders microservice, the output looks like [Figure 7](#).

[Figure 8](#) shows the output when the query to retrieve order records is executed.

## Why Should You Use GraphQL in Microservices Architecture?

Microservices architecture splits applications into self-contained services that communicate through APIs. These services provide business operations and may be designed, deployed, and modified independently. Hence, services are simpler to manage and update, improving system agility, scalability, and resilience.

GraphQL's ability to retrieve and aggregate data from several services in a single request and its adaptability, speed, and flexibility make it a good fit for microservices architectures. GraphQL allows you to define the necessary data in a single request and then get this data from many services with a single query.

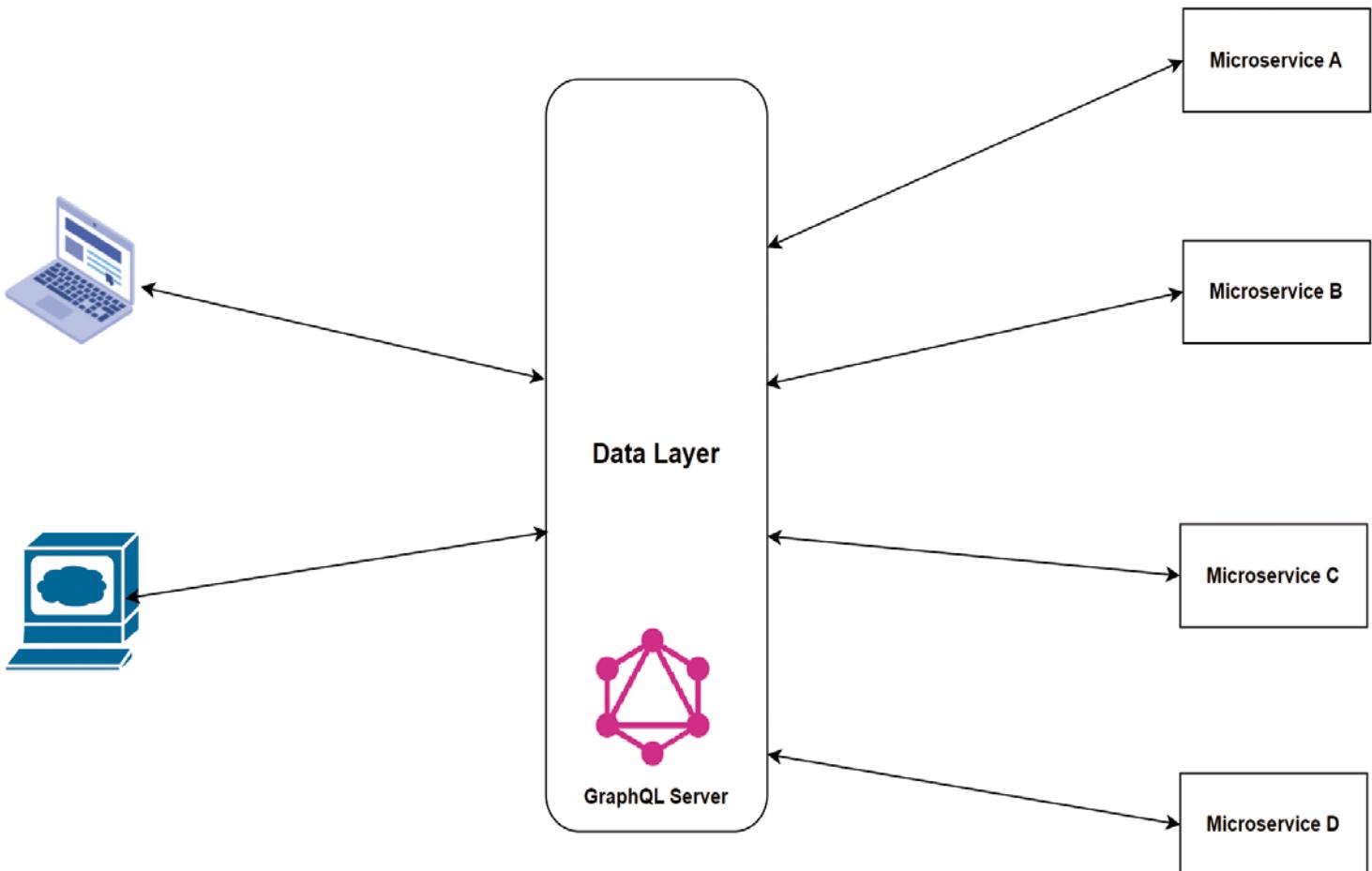
GraphQL offers additional benefits for microservices systems, including:

- The schema-first approach and robust type system of GraphQL ease the process of building and testing services independently, decreasing the likelihood of API incompatibility issues.
- Flexible and efficient, GraphQL enables easy updates and inclusion of new features and capabilities without altering the underlying data model.
- The flexibility to create and update APIs independently of back-end services allows cross-functional teams to operate efficiently.

## Improving Microservice Architecture with GraphQL API Gateways

GraphQL offers great flexibility in managing data, enabling a more scalable, modular, and maintainable way of creating APIs that combine data from retrieved from multiple downstream services. Here are the key reasons why GraphQL is a good choice for building API gateways:

- **Data aggregation:** API gateways aggregate data from multiple microservices and present it to clients as a single API. A GraphQL API endpoint can aggregate data from these services, so clients need not make repeated requests to different services.
- **Flexible data retrieval:** You can use GraphQL to define the precise data you want as well as its format,



**Figure 9:** Using GraphQL as a data layer

reducing bandwidth consumption and enhancing performance.

- **Effective data fetching:** With GraphQL, queries can be batch-processed and cached to improve performance and reduce the number of roundtrips to the server.
- **Strong typing:** By allowing for strong typing of API requests and responses, GraphQL's type system reduces the possibility of runtime errors and boosts developer efficiency.

## GraphQL as a Data Layer for Microservices

GraphQL can be used as the data layer for your microservices-based applications. In this scenario, GraphQL may be used as an API gateway to collect data from multiple downstream microservices and display it to users. Each microservice handles and saves its data in this architecture, with GraphQL as the API gateway. The server may batch and cache requests using DataLoader to increase performance.

A GraphQL schema describes the data managed by each microservice. The API gateway leverages these schemas to provide a unified, single-schema representation of the entire data graph. The API gateway responds to client requests by delegating the calls to the relevant downstream microservices and collecting the results as shown in **Figure 9**.

A consumer can fetch data with only one API request when GraphQL is used as an API gateway. The API gateway provides a uniform view of the data, abstracting away the complexities of the microservices. Each microservice can be built independently with this architecture because modifying the microservices does not necessitate altering the API gateway. **Figure 10** shows how you can use GraphQL to delegate calls to the downstream services.

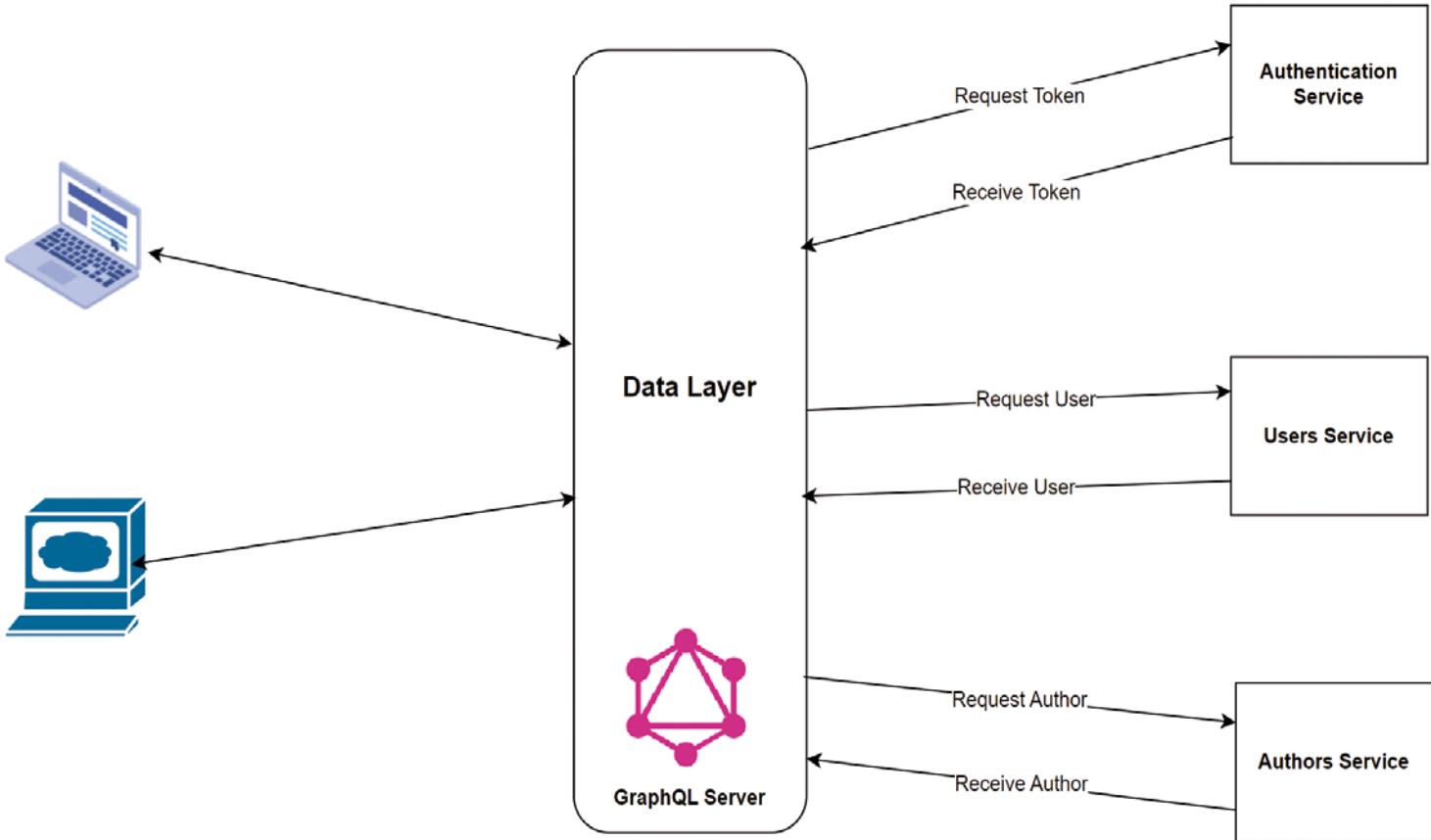
## Optimizing Data Fetching in GraphQL

This section examines the strategies that can be adopted for optimizing data retrieval in GraphQL.

### Caching in GraphQL

Every application that makes repeated queries to a server may benefit significantly from caching. Caching may improve GraphQL application performance and responsiveness by decreasing the number of round trips to the server. In GraphQL a client submits a request to the server, typically using the HTTP POST method. Neither the client nor the server caches these queries; thus, a complete round trip must be made for the query to get the desired results. Here's where caching helps.

There are several strategies to implement caching when working with GraphQL:



**Figure 10:** Using GraphQL as a data layer, delegating calls to downstream services

#### SPONSORED SIDEBAR:

Ready to Modernize a Legacy App?

Need advice about migrating yesterday's legacy applications to today's modern platforms? Take advantage of CODE Consulting's years of experience and contact us today to schedule a free consulting call to discuss your options. No strings. No commitment. Nothing to buy. For more information, visit [www.codemag.com/consulting](http://www.codemag.com/consulting) or email us at [info@codemag.com](mailto:info@codemag.com).

- **Client-side:** Caching GraphQL query results locally on the client computer eliminates the need to requery the server whenever the same information is needed. You can take advantage of libraries like Apollo Client or Relay, which provide support for in-built caching.
- **Server-side:** Caching GraphQL query results on the server side allows subsequent requests to be provided directly from the cache rather than requiring a new query to be processed.
- **CDN caching:** CDN caching allows GraphQL queries to be cached at the network's edge. In CDNs, answers may be cached according to the query parameters, allowing faster caching and serving future requests with the same parameters. Ensure that you use the appropriate caching strategy when using GraphQL to store data in a cache.

Although server-side caching is well-suited for applications with a lot of data and traffic, client-side caching works well for applications that require more user interactivity.

#### Batching in GraphQL

With batching, GraphQL clients can send a bunch of queries in a single request, reducing network latency and boosting performance. GraphQL servers execute the resolver functions separately when GraphQL queries are executed and return the results in a single response. You package several queries together to make a single request when you use

batching. Before returning the results in a single response, the server runs all resolver functions in a single batch.

In order to batch, GraphQL clients have to send one request with a list of queries or mutations. The server processes each query or mutation and sends a single response with a list of results. GraphQL clients, including Apollo, allow you to batch multiple queries together. Batching is a great option when you want to combine several queries.

#### Using a Data Loader

By batching requests and caching data, data loaders may help your GraphQL APIs perform better, especially when several fields in a query need data from the same data store or when you need to receive data from many searches for the same data. DataLoader may batch queries and cache data to lessen the network cost and latency of getting data from a remote source.

You can easily integrate a data loader with various data sources, such as REST APIs, databases, etc. Additionally, it's compatible with well-known GraphQL libraries such as Apollo Server and GraphQL.js and works with any GraphQL implementation that supports middleware.

The data loader caches the data as soon as the data is fetched from a data store. It examines the cache to determine whether the requested data exists before forwarding it to the resolution function. If the data loader doesn't al-

# COMMERCIAL UAV EXPO

SEP. 5-7, 2023

CAESARS FORUM / LAS VEGAS

**VERTICAL FOCUS.**  
**GLOBAL REACH.**



**Register at**  
[expouav.com](http://expouav.com)

Use code **SAVE100** for  
\$100 off a conference  
pass or a **FREE** exhibit  
hall pass



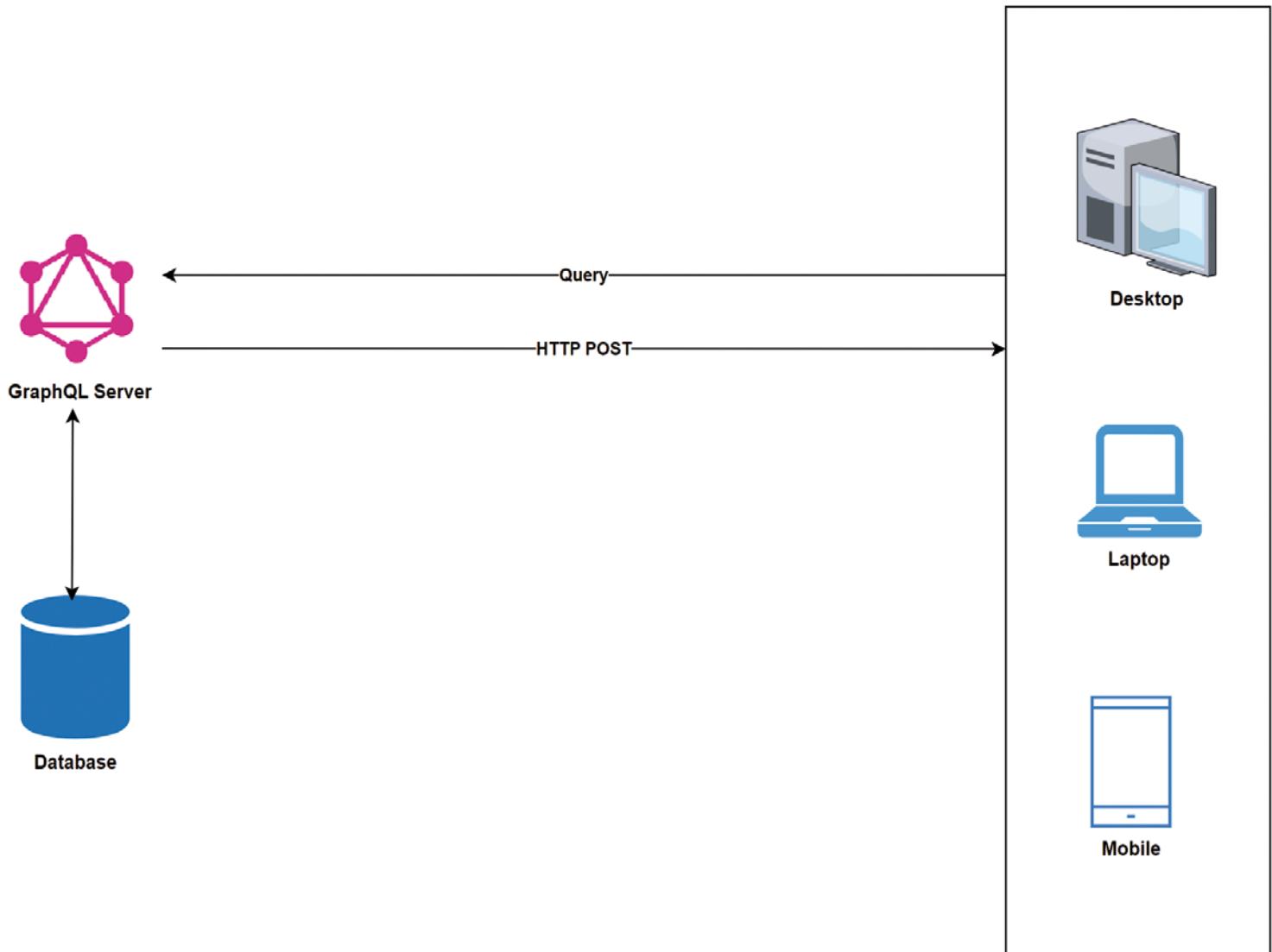
THE COMMERCIAL UAV EVENT FOR:



Presented by: COMMERCIAL UAV NEWS

Official Association Partner: COMMERCIAL DRONE ALLIANCE

Produced by Diversified Communications



**Figure 11:** GraphQL Server with connected database

ready have it in the cache, it adds the request to a batch and waits for all requests of the batch to be fulfilled. Once all the requests in the batch have been collected, the data loader executes a single query to retrieve data for all requests in the batch. The retrieved data is sent to each resolver function and any new data is cached.

## GraphQL Architectural Patterns

When using GraphQL, you can take advantage of several architectural patterns, depending on your requirements. These patterns are more like guiding stones that provide you with the needed flexibility when working with data using GraphQL.

### GraphQL Server with a Connected Database

This approach integrates GraphQL with the underlying database through a single server. Clients communicate with GraphQL servers by sending queries. The server responds to client queries by reading them and fetching requested data from the underlying database. Upon resolving the query, the GraphQL server returns the requested data to the client as shown in **Figure 11**.

### GraphQL Layer Integrated with Existing Systems

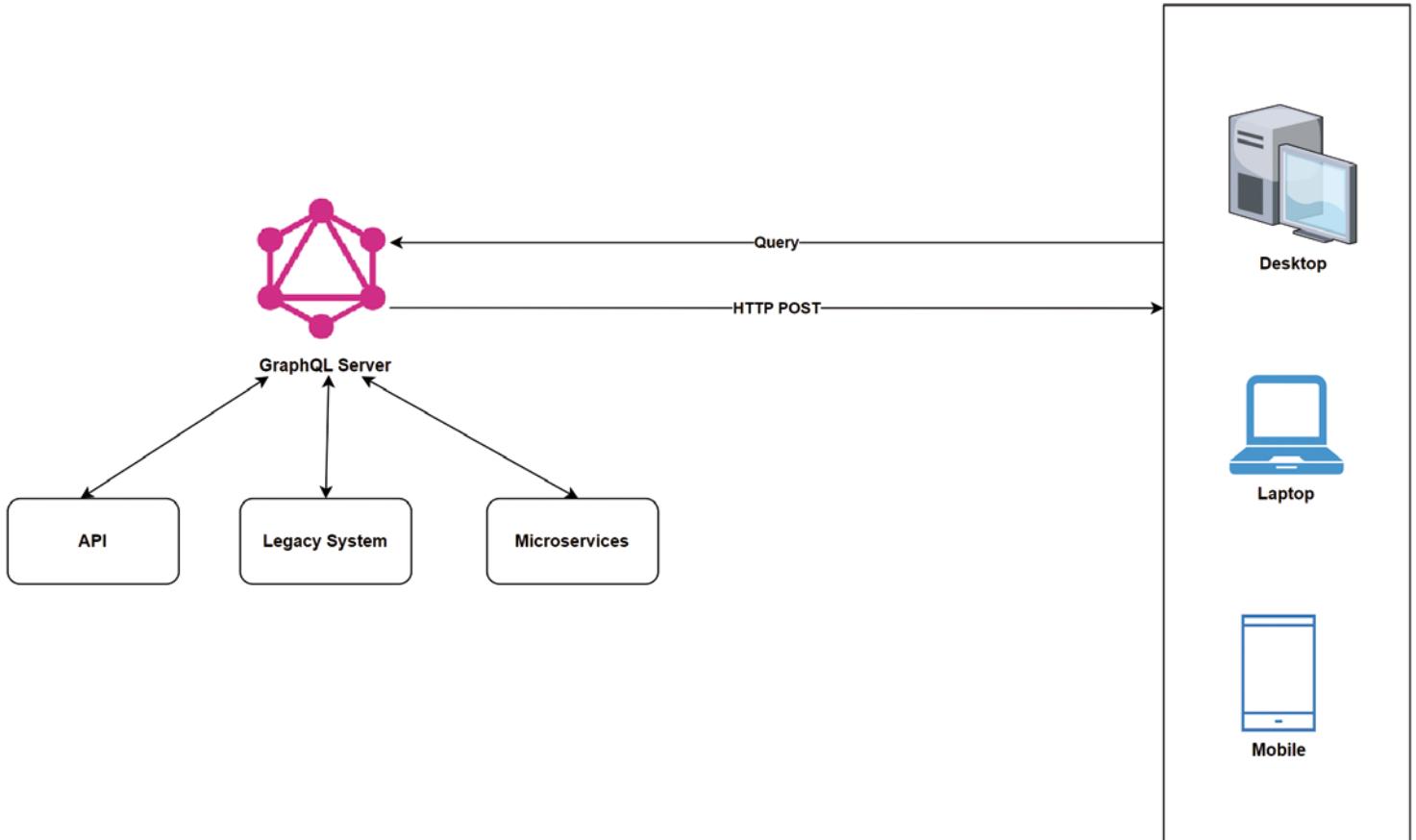
In this architectural pattern, the GraphQL server hides the complexity of data fetching logic, providing clients with a unified interface to access data from various disparate systems. It's well suited for projects with legacy infrastructure and multiple APIs, as illustrated in **Figure 12**.

### Hybrid Approach with Connected Database and Integrated System

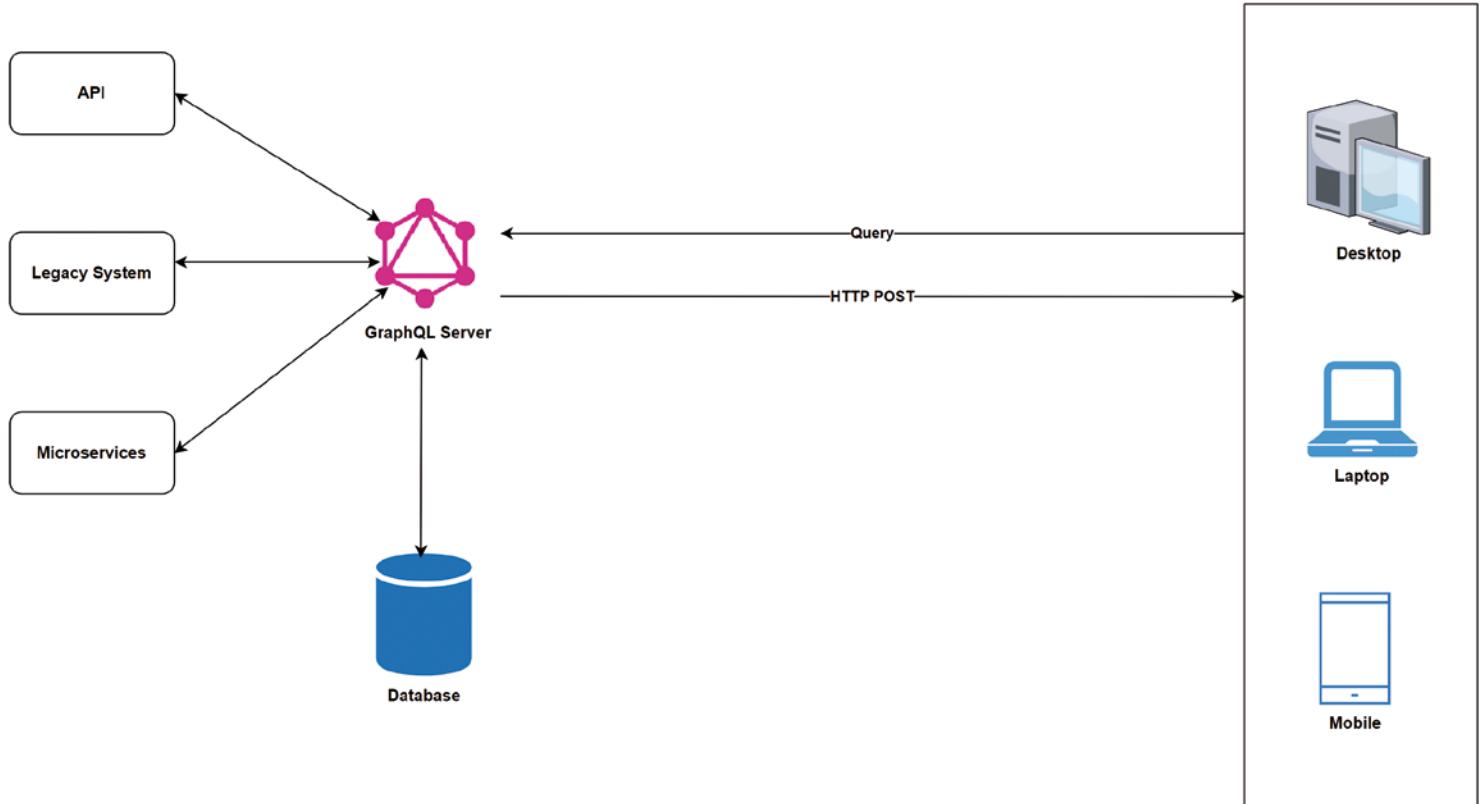
This architectural pattern combines the two previous approaches where the GraphQL server resolves the query received from a client and then fetches data from the connected database or the integrated APIs. In this approach, the GraphQL server has its connected database, and it can also be connected to external systems while providing the clients with a unified API, as shown in **Figure 13**.

## Securing GraphQL APIs

There are several ways to secure GraphQL APIs. I'll examine some of the key strategies you can adopt to secure your GraphQL API in this section.



**Figure 12:** GraphQL Layer integrated with other systems



**Figure 13:** A hybrid approach with GraphQL, connected database, and an integrated system

## Using Rate Limits

The words "rate limiting" refer to a strategy that restricts the number of requests a client may make to a sever in a particular time frame. Rate limitation may be used to prevent your GraphQL APIs from being overloaded. It can help thwart denial-of-service attacks and brute-force attacks by restricting the number of requests made to your GraphQL APIs in a certain time period.

## Using Timeouts

For several reasons, when you're working with applications that leverage APIs, the downstream services might need to be fixed, i.e., they might be non-responsive. Possible causes can be because the services are processing data, they're taking quite a lot of time to read from the database, etc. Whatever the reason, you shouldn't allow your GraphQL API to hang for too long while the clients wait for a response. To prevent this, you can leverage timeouts for your slow or non-responsive services.

## Using Authentication and Authorization

Authentication aims to verify that a person is who they claim to be, i.e., that the credentials they use for logging into the system are genuine and can be trusted. You can choose strategies, such as JWT tokens, OAuth tokens, or API keys to implement authentication. Authorization refers to the process by which the application determines what resources are accessible to an authenticated user and what information they can access within the application. You can use several techniques to implement authorization, such as attribute-based or role-based access control systems.

## Input Validation

Another way to protect APIs is by verifying that the input data, that's the data entered by the user, is valid and satisfies specific requirements. To secure your API, you must validate the input data, such as the user's entered data. This helps you protect your API from injection attacks that include cross-site scripting (XSS) and SQL Injection.

## Best Practices in GraphQL

Here are some recommended best practices for GraphQL:

- **Define a clear schema.** The schema serves as the basis of your GraphQL API, defining the sorts of data that may be queried. Ensure that your schema is well-defined, understandable, and well-documented.
- **Use efficient resolvers.** Resolvers are functions that get data from a GraphQL query for a specified field. Ascertain that your resolvers are efficient and optimized in order to reduce the amount of database queries and network requests required to complete a request.

## • Use pagination.

While browsing through large amounts of data, it's recommended to make use of pagination in order to limit the quantity of data returned over the wire, thus improving performance while lowering server load.

## • Use caching.

Caching is a proven technique that may help you improve performance by reducing the number of database and network calls needed to fulfil a request. You should choose a caching solution commensurate with your use case and the application's data.

## • Use nested types to reduce network calls.

You should design your GraphQL schema in such a way that related data are grouped in nested types.

## • Avoid N+1 queries.

N+1 queries occur when a GraphQL query results in several database searches for the same data. To prevent this issue, use DataLoader or other batch-loading approaches.

## • Use input types for mutations.

Use input types to define mutation input parameters and simplify the structure of your GraphQL documents. This might make your schema more understandable and consistent.

## • Use directives.

You may use directives to add information to your schema and change the behavior of resolvers. To add validation, authorization, or other functionality to your API, use directives.

## • Handle errors gracefully.

GraphQL APIs might report errors in a number of situations, such as when a query is incorrect or a database query fails. Ensure that your API handles problems gracefully and offers useful error messages.

## • Use a testing tool.

To test your API and explore your schema, use GraphQL or GraphQL Playground. This can help you in executing the queries and mutations and also identifying problems in your API.

## Conclusion

GraphQL offers a comprehensive framework for building extendable, efficient, and user-friendly APIs. You can use GraphQL to describe the data you need, and the API will provide it in the most appropriate manner. This capability allows APIs to be easily changed or upgraded without disrupting existing deployments.

In the next and concluding part of this series of articles on GraphQL, I'll examine how GraphQL can be used to build scalable, high performance, flexible microservices architectures.

Joydip Kanjilal  


(Continued from 74)

that I should choose the one that appears to be reliably more interesting, more meaningful, and more engaging.

Although a little chaos may be "interesting," maybe that road is too well worn. How about some peace and tranquility? That's when I decided to peel back the onion layers in search of a new road. To get out of my rut, I've decided to return to the road not taken in quite some time. It's the road of basics and fundamentals. Not matter how big or small a project is, if the fundamentals aren't sound, it's akin to building on a sand foundation. Lately, like many others, I've been reassessing what Agile is, what it means to me, what the associated frameworks are, how we go about practicing it, evangelizing it, implementing it, etc. In these pages, at one time or another, I've quoted another famous Dane named Niels Bohr. One of his fundamentals was that no matter how complex the topic, the topic should be explainable in plain simple language. With that background in mind, I began to assess what was the most primary, basic thing I could focus on in the Agile/DevOps space. Soon thereafter, I recalled a three-word phrase that every DevOps practitioner knows: *People, Process, and Tools*.

The answer was staring me in the face. It's the people. How important are people? Labor is the essential element that capital requires to enable production and produce value. Each of us, in this technical space, does so in the realm of the real world, not the world as we would wish it to be. The lens through which we each view the world has bearing on whether we perceive a problem domain as being clear, complex, complicated, chaotic, or completely unmoored as a state of confusion. Indeed, one person's obvious may be another person's confusion! As team members and fellow humans, we should take stock of those perceptions and the differences between perceptions. There may be nothing else that goes more toward team dynamics than how these various perceptions are acknowledged and reconciled. This model discussed here on how we perceive a problem domain is the basic thesis of David Snowden's Cynefin Framework [https://en.wikipedia.org/wiki/Cynefin\\_framework](https://en.wikipedia.org/wiki/Cynefin_framework).

Presuming that a firm wishes to adopt a more "people-centric" approach, the question is how to enable that effort. One major issue to address, especially given the past 3.5 years, is the notion of psychological safety at work. Are people encouraged to speak their minds? Is it safe to admit mistakes? Are risks encouraged, without fear of negative consequences? One of the foremost experts in this field is Professor Amy Edmondson of Harvard University. In November/December 2016, Ted Neward also wrote in CODE Magazine about the necessity of

psychological safety in the context of building strong teams.

Referencing again the basic phrase *People, Process, and Tools*, assuming that we can move the organization sufficiently along and because we live in the real world, pragmatism must take hold. There must be some process and tooling, even if rudimentary, that's compatible with personnel and required use-cases that can enable strong teams.

Referencing again the Agile space and Scrum in particular, one general task that's rife with inaccuracies is estimation. Too often, we're called upon to provide a single, static number. Also, estimations tend to only be from one perspective: effort. What about value? How about costs or complexity? To add to all of that, there's the remote work dynamic, with teams distributed across many time zones and continents. How are teams supposed to collaborate in such cases?

Fortunately, I found a tool and an associated process methodology, this time from some Danes you haven't heard of: Agile Lean House: <https://www.agileleanhouse.com>. One of their tools, Cool Estimation (<https://estimation.agileleanapps.com/>) addresses the issues listed above with traditional agile estimation. It's also this small shop that has managed to put together, in a quite original and people-centric manner, how the Cynefin Framework (and other decision frameworks) and Professor Edmondson's philosophy may apply to a particular agile implementation. If any of that only sounds moderately interesting, checkout their Agemba tool that, among other things, can illustrate **dependencies** across multiple epics/stories and multiple teams <https://agemba.agileleanapps.com/>. Dependencies among things, whether large, like organizations, or small, like application modules, are important to understand, as those things are often a prime source of technical debt. It's important to get transparency and visibility on these matters, not only for benefit of a project, but for the people as well, and thereby, the entire organization.

In that spirit of human- and people-centeredness, Agile Lean House is doing important work for university students at Ukraine's Kiev National University by providing these war-displaced folks training for the day when this war is over. The work sets about to rebuild a country's infrastructure. I learned that their spirit is still largely intact when I participated in one of these two-day sessions. Although I've trained many students before, this was a new road for me. By simply going back to basics, perhaps just pausing, stepping back, and assessing, it's amazing what you can find!

Examine the road you're on. Always ask yourself why you're contemplating a change in the first place. This time, I **chose** a different road and it

made all the difference. And if you can't choose, then just pick the other road, and move on. No matter where you're going, you'll get there.

In the face of other people's disagreement, I like to cite the Dude from the Big Lebowski as an appropriate response: "Yeah, well, you know, that's just, like, your opinion, man." Maybe the real lesson isn't in how we make decisions, but whether there is a decision to be made in the first place? Sometimes, we just need to get off the dime to get out of the rut. And if you're wondering how to do that, ping me and let's talk. I only ask that you pay it forward someday. Sometimes, we just need to ask for help along the way. And to that, there needs to be someone on the other end to pick up the call. Every good thing that ever happened began with a conversation and comity among different people. Let's constitute ourselves and our service to others toward that worthwhile and most interesting end. I look forward to that road. Hygge!

At no time was any form of AI considered or used in this article's preparation and writing.

John V. Petersen  
**CODE**



Jul/Aug 2023  
Volume 24 Issue 4

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  
Joydip Kanjilal  
Sahil Malik  
Paul D. Sheriff

Wei-Meng Lee  
John Petersen

Technical Reviewers  
Markus Egger  
Rod Paddock

Production  
Friedl Raffeiner Grafik Studio  
[www.frigraf.it](http://www.frigraf.it)

Graphic Layout  
Friedl Raffeiner Grafik Studio in collaboration with onsight ([www.onsightdesign.info](http://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](mailto:tammy@codemag.com)

Circulation & Distribution  
General Circulation: EPS Software Corp.  
Newsstand: American News Company (ANC)

Subscriptions  
Subscription Manager  
Colleen Cade  
[ccade@codemag.com](mailto: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. Back issues are available. For subscription information, e-mail [subscriptions@codemag.com](mailto:subscriptions@codemag.com).

Subscribe online at  
[www.codemag.com](http://www.codemag.com)

CODE Developer Magazine  
6605 Cypresswood Drive, Ste 425, Spring, Texas 77379  
Phone: 832-717-4445



# CODA: The Road Not Taken

The inspiration for this month's column is Robert Frost's 1915 same-named poem. The poem's text may be found here: <https://www.poetryfoundation.org/poems/44272/the-road-not-taken>, as well as a reading guide here: <https://www.poetryfoundation.org/articles/89511/robert-frost-the-road-not-taken>.

Already, you may be thinking that on the title alone, this month's column is serious, but the poem was anything but. As the associated guide explains, Frost liked to say that "I'm never more serious than when joking." That statement can be taken multiple ways. Generally, Frost's statement requires us to confront the fact that some ways may conflict with each other, and both can still be true. We might choose only one way despite *wishing* we could easily avail ourselves of both choices, even for only a brief time. It's the stuff that makes decision-making so difficult. Let's dig into it.

Fundamentally, what we do, how we do it, and the way we do it begins with making decisions, whether big or small. The past 3.5 years since the COVID pandemic began has been what can be reasonably summed-up as "living in interesting times." In Chinese philosophy, peace and tranquility, from an English idiomatic point of view, are "not interesting." The implication is that chaos is interesting. And therein lies the irony and what some have attributed to an ancient Chinese curse. And as it turns out, Frost's poem was meant as an ironic dig at a friend. Ironic or not and as misunderstood as it may be, the poem has always had a special meaning for me. What something means in the eye of the beholder is part of what art is about. It's a matter of perception, which is determined by what we value. For many, including myself, the last 3.5 years have had a big knock-on effect to what we value. And what we value goes to how we view the recent past.

For some, what's transpired, whether it's the pandemic itself, our response to it, or some combination has been clear and simple to understand. At the same time, others may have quite different perceptions ranging from complicated, to complex, to chaotic, and even confusion. From my perspective, all these domains could have applied at one time or another. Much has gone on that is of interest, grabs our attention, and engages us, often to the point of fatigue. What makes something "Interesting" is unique to each person because our perceptions are influenced by our cumulative experiences to that point.

Everyone has biases. It's just a fact of life. We should endeavor to understand them because

our individual perceptions and biases can not only influence the decisions we make, but how we view the context within which those decisions need to be made. After what may have been a challenging decision, how many times have you lamented "I should have chosen another option?" As someone who has undergone much self-examination during this same 3.5 years, such rumination is a waste of time. It is a waste of time because things just learned weren't previously known.

And yet we quite often beat ourselves up for not having been clairvoyant in the past. We need to ask different questions. Perhaps the best question is how can we improve the conditions for future decisions? Improving conditions for the future is the primary objective of any plan. The uncertainty of the future coupled with uncertain present information and how we each perceive a problem domain makes our work today more challenging than ever. It can be exceptionally challenging when there isn't the ability to make any sort of decision, great or small, no matter the significance, and no matter how favorable the conditions are to make what many might call a "slam dunk" easy decision to make. That's what Frost's poem was mockingly about: His friend struggles with decision-making.

We live in interesting and complicated times where we're bombarded with too much information and we need to make a quick decision, or we're confronted with a dearth of information and we need to make a quick decision. Either way, we're often forced to make quick decisions regardless of how available and reliable meaningful information is. Which road to choose? How do we go about that process?

The Danish philosopher and father of Existentialism Søren Kierkegaard taught us in his 1843 work "Either/Or" that picking and choosing are not necessarily the same thing. In one sense, I may decide to pick 12 tomatoes from a basket. In that process of determining which 12, I'll choose the ones that I find to be optimal..... if I have time. On the other hand, if I'm rushed, whether due to lack of planning or because the new task is a bona-fide emergent matter, I may just have to pick the best 12 I can under the

circumstances. The result may be satisfactory despite the sub-optimal or unknown task context. Or the result may be a total disaster.

Not choosing is choosing. Making significant decisions to choose among alternatives involves some level of conflict because we're forced to confront the risk of making decisions based on imperfect and incomplete information. It's just a fact of life because we can't possibly know everything about everything at any time. This is the practical reality of things. The reality of things often requires us to pay close attention to the ethics of the choice, rather than the aesthetics of how it makes things look. For example, "All the cool kids are writing in React now, so we'll do that!"

And how does the angst of having to make a choice get dealt with? Often by not making the affirmative choice and, instead, waiting for things to take their course. This is what complacency looks and sounds like. The root of that complacency is often borne of fear. You may be familiar with the phrase "Nobody ever got fired for buying IBM stock." That may be. But I'm quite sure that at least one person got fired who did buy IBM's products. Breaking it all down, it's unavoidable that decisions must be made, each with its own amount of risk and uncertainty.

Eventually, the path we're on requires choosing from two divergent paths, thus requiring us to change the path we're on. The question of how do we do that by ourselves and as part of a team has been nagging at me for quite some time. It's my practice area of Agile and DevOps that has me confronting these questions with renewed scrutiny over the past several years. It seems that we live in an ecosystem of this tool or that, this process or that, etc. these days. In retrospect, process and tool debates are nothing new. Perhaps for me, it's because of the road I kept taking, which was the first big realization for me. That road for me was well-worn and well-known. If I were going to stay on that road, that was on me. If I were going to take the other road, that was on me too. As either way it's on me, it seems obvious

*(Continued on page 72)*



CUSTOM SOFTWARE DEVELOPMENT

STAFFING

TRAINING/MENTORING

SECURITY

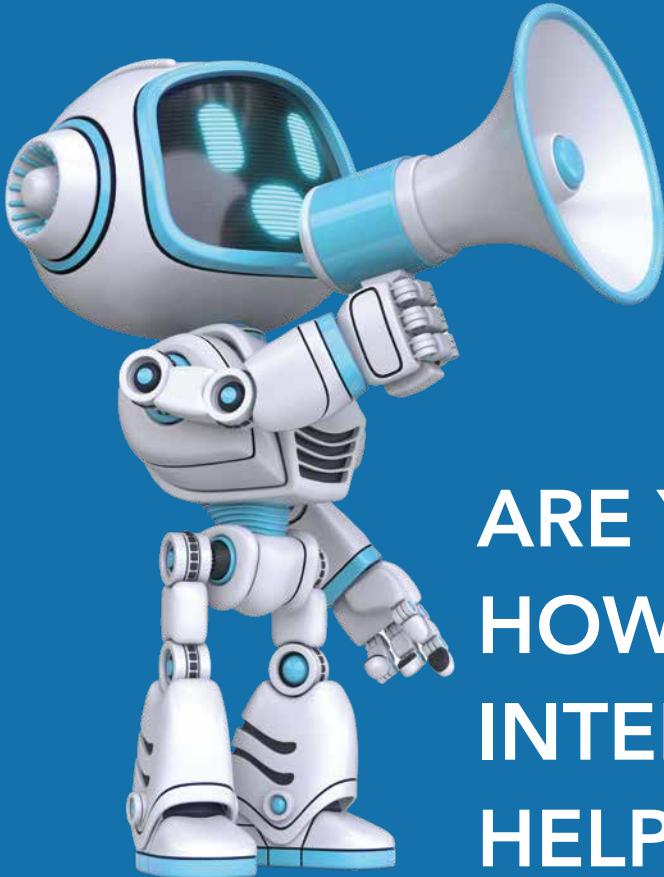
MORE THAN JUST  
A MAGAZINE!

Does your development team lack skills or time to complete all your business-critical software projects? CODE Consulting has top-tier developers available with in-depth experience in .NET, web development, desktop development (WPF), Blazor, Azure, mobile apps, IoT and more.

Contact us today for a complimentary one hour tech consultation. No strings. No commitment. Just CODE.

[codemag.com/code](http://codemag.com/code)

832-717-4445 ext. 9 • [info@codemag.com](mailto:info@codemag.com)



# ARE YOU WONDERING HOW ARTIFICIAL INTELLIGENCE CAN HELP YOUR BUSINESS?

Do you worry about privacy or regulatory issues stopping you from using AI to its fullest?

**We have the answers!**

We will send an expert to your office to meet with you. You will receive:

1. An overview presentation of the current state of Artificial Intelligence.
2. How to use AI in your business while ensuring privacy of your and your clients' information.
3. A sample application built on your own HR documents – allowing your employees to query those documents in English and cutting down the number of questions that you and your HR group have to answer.
4. A roadmap for future use of AI catered to what you do.

**CONTACT US TODAY FOR A FREE CONSULTATION AND DETAILS ABOUT OUR SERVICES.**

[codemag.com/executivebriefing](http://codemag.com/executivebriefing)  
832-717-4445 ext. 9 • info@codemag.com