iex cloud

Building With Financial Data

Apperate's Essential Resources

Building a real-time market data app

Financial data is messy; building a real-time market data app can be even messier. In this guide for API developers, Apperate has put together three key resources for building with financial data without the headache. We outline how to navigate sourcing for financial services apps, take you through everything you need to know for building a secure trading app that queries IEX Cloud Apperate for price data and other metrics for publicly traded stocks , and provide you with the key questions to ask your data provider before launching.

Table of contents:

A Strong Foundation: Building With Financial Data

In this piece, we'll explore the various types and sources for financial data with an eye toward softwareapplication and API developers. If you're building a financial services app, this piece will give you everything you need to start from a strong foundation.

Crunchbase lists tens of thousands of venture-funded financial services startups operating today, and in 2021 alone, these companies raised $130 billion.

While that number was down in 2022 as venture capitalists reigned in commitments across the board the sector has generally seen a massive uptick in the past decade.

> **"The major winners will be financial services companies that embrace technology." – Alexander Peh, PayPal and Braintree**

The boom in financial services businesses has coincided with a growth in data availability in all industries. Financial data is now more accessible than ever before, and that has contributed to a huge increase in demand and the number of use cases for it. Everyone from traditional financial advisors to news outlets to AI trading tools is using financial data to make better-informed decisions and improve investment outcomes.

In this piece, we'll explore the various types and sources for financial data with an eye toward software application and API developers. If you're building a financial services app, this piece will give you everything you need to start from a strong foundation.

What is Financial Data?

Financial data refers to data related to the financial performance of a company or industry. It typically includes figures like revenue, profit, expenses, and balance sheet items, but it can also include information about the company's stock price, shares outstanding, and trading volume.

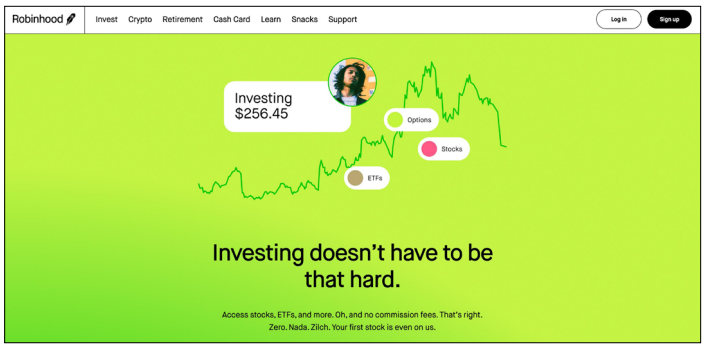
With thousands of publicly traded companies around the world, there is a lot of financial data out there, and as you'll see in this piece, collecting, organizing, and storing even a portion of it is no small feat.

Meanwhile, interest in non-traditional financial data is also growing. For example, some investors are now factoring environmental, social, and governance (ESG) data into their decisions. Investment firms also use social media sentiment, geographic data, and even internet of things (IoT) data to shape their trading strategies.

As machine learning models and more advanced AI techniques continue to develop, we'll see even more interesting use cases and types of data used in financial technology applications, but let's take a look at some of the most common use cases for financial data today.

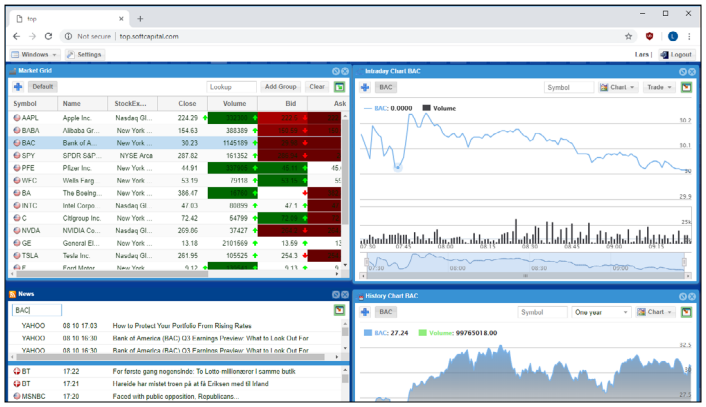
Use Cases for Financial Data

In the past few years, consumer interest in digital trading applications has skyrocketed. Popular mobile apps like Robinhood have made investing in stocks, cryptocurrencies, and other securities easier and faster than ever before, and all these tools rely heavily on fast and accurate financial data.



Robinhood uses financial data to inform retail investors

In addition to lightweight consumer apps like Robinhood, there are more advanced tools that offer expanded data and insights to investors. For example, SoftCapital's TOP trading terminal layers news and historical financial data into its browser-based trading platform.

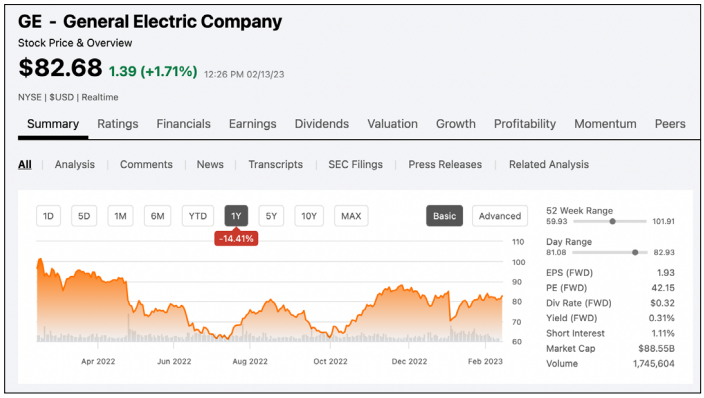


SoftCapital trading terminal

Similarly, many portfolio management applications rely on financial data to inform and guide users. For example, Simply Safe Dividends uses this data to help dividend investors monitor their cashflow and portfolio valuation over time.

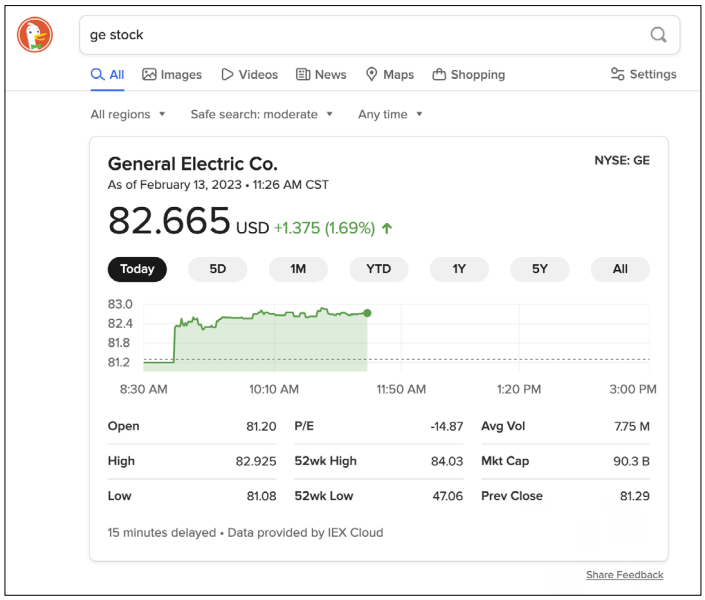
Simply Safe Dividends uses financial data to inform dividend investors

News sites and aggregators are often large consumers of financial data as well. Seeking Alpha, a popular stock market community and analysis tool leverages historical stock data and company financials to augment recommendations from users.



Seeking Alpha stock price data

Search engines like DuckDuckGo even integrate financial data into their search results.



DuckDuckGo integrates stock data into search results

Finally, algorithmic and AI-driven trading applications rely on fast and accurate financial data to make buy-sell decisions more quickly than human traders can.

IEX Cloud and FreeCodeCamp offer a free course on building an algorithmic trading platform in Python so if you'd like to learn more about the technical specifics of this field, that's a good place to start.

Building Software With Financial Data

While financial data is easier to get than ever before, and the number of possible applications for it continues to grow, there are some things to consider before you start building an application that relies on it. One of the first is where you'll get the data from, so let's take a look at how you can choose the right provider and the various types of financial data sources out there.

Choosing a Financial Data Provider

Depending on the type of application you're building, you may weigh some factors more than others, but most developers will look at the following:

Data Coverage and Quality

Your data provider should offer comprehensive coverage of the markets and instruments that are relevant to your business. Most providers have some limitations. For example, data feeds from a single exchange often don't include details on companies listed on other exchanges.

The data should also be accurate, up-to-date, and reliable. In cases where a slow response or inaccurate number can lead to millions of dollars in losses per minute, this is a make-or-break consideration. Additionally, data providers should handle and help you handle changes.

For example, what happens when a company changes its stock ticker (e.g., [`FB` to `META`)? It's important that your data provider uses best practices for symbology, financial data normalization and change management.

Integration Capabilities

In most cases, you'll be integrating your financial data with other applications or partner data, so it's important to consider whether the format and delivery mechanism will make this possible. I've seen situations where normalizing data from multiple sources proved to be a huge technical headache, so it's always easier to choose providers with pre-built integrations.

Security and Compliance

In addition to providing accurate data, good providers will have robust security measures in place to ensure that sensitive financial data are protected. This will help you remain in compliance with relevant regulations and standards, plus it speaks to the provider's professionalism and accountability.

Support

Training, documentation, and technical support are all important factors. Some financial data providers are very hands-off, giving you a single data format and no option for support or customization while others may go as far as helping you implement their data into your application.

Pricing and Scalability

While you want the best data possible, you also need the budget to afford it. It's important to weigh the cost of each data provider with the services they offer, though. Saving a few hundred dollars per month at the cost of 20 extra engineering hours per week is not a good tradeoff. You also need to consider how well their pricing model scales, as switching from one data provider to another is exceedingly difficult.

Innovation and Development

Finally, look for financial data providers that are constantly innovating and developing new features. You want to ensure that your application will have access to the latest market data and analytics tools, so choose a provider who's actively maintaining and expanding its offerings.

Financial Data Sources

Now that you know what you're looking for in regard to financial data and how to evaluate providers, let's explore some of the options available. As with any high-level engineering decision, there's no single *best* option on this list, so you'll have to consider your team's existing resources and your application's requirements.

Stock Market Data Providers

There are a number of companies that specialize in providing stock market data, such as Refinitiv, S&P Global Market Intelligence, IEX Cloud, and FactSet. These providers offer a wide range of data, including historical data, real-time data, and analytics tools that will help you build robust financial applications and APIs more quickly, but the price of some of these options can vary dramatically. That said, IEX Cloud lets you start with low-cost pay-as-you-go plans with many data bundles to choose from.

Stock Exchanges

Many stock exchanges, such as the New York Stock Exchange (NYSE), IEX Exchange and the NASDAQ also provide market data through direct data feeds connected to their data centers. The downside is that each exchange can only provide data for the buy and sell orders on its own order book, so you may have to do a lot of work integrating data services to get a full view of the stock market. This leads again to the challenge of normalizing data formats, which is no small consideration.

Financial Websites and Portals

Websites like Yahoo Finance, Google Finance, and Bloomberg provide stock market data and other financial information that can be useful for new product development. While their datasets and tools may not be as comprehensive as purpose-built stock market data providers, they could be cheaper or easier to use if your business already subscribes to these services.

Government Agencies

Government agencies like the Securities and Exchange Commission (SEC) also provide market data, including financial reports and filings from public companies. This data is free, but it isn't real-time, and because of the organizational structure, it's harder to navigate than some of the other options here. That said, it could be a good way to augment data you already have for publicly traded companies.

Social Media and News Websites

While many social media platforms have reigned in the amount of data they offer freely in the wake of

the Cambridge Analytica scandal, you can still get real-time data on public posts from Twitter and many news aggregators. Financial applications often use this data to inform users about market trends, stock movements, or investor sentiment.

Building Software With Financial Data

The accuracy and quality of financial data available varies widely. It's worth noting that there are many unscrupulous and unregulated sources for financial data, but these are rarely worth the risk.

It may also be tempting to scrape data from other websites, but it's incredibly hard to ensure the accuracy and integrity of this data. You are also putting your company at legal risk as this kind of behavior typically violates terms of service and data ownership laws.

You should also be careful of how data is transferred from third parties. Unsecured HTTP requests or manual data entry can lead to data security and integrity issues. While having a formal data governance plan in place is a good idea and can help catch errors, it's not a replacement for using reputable data providers with the proper best practices in place.

Finally, don't get caught without a disaster mitigation and recovery plan in place. No matter how reliable your provider or your internal data practices, no technology is perfect.

Developing Your Own Financial Data API

Most modern applications wrap their data layer in a developer-friendly API, so likely, one of your first steps in building with financial data will be developing the API that internal and external stakeholders will use.

So, I'll finish up this piece with a brief outline of the challenges you have ahead. If you've already got a team in place, go through this list with them to ensure everyone feels comfortable with your capacity and the path forward.

Financial Industry Expertise

Building useful applications with financial data will require a deep understanding of the industry, as well as the technical expertise to develop the API itself. This often requires a multidisciplinary team of developers, financial experts, data scientists, and analysts.

Sourcing and Organizing Financial Data

Once your API is created, it must be sourced, or connected to various financial data providers. This typically involves negotiating access and establishing agreements with these providers. You'll also need to set up a data retrieval and differencing system to ensure it stays up-to-date and that formatting inconsistencies are resolved.

Securing the Data

In addition to creating and seeing your API, you have to ensure that access to it is secure and reliable while maintaining ease of use. API keys and OAuth is a good starting point, but you should also consider roles and permissions. If you're building an API designed for external consumption, you also need to consider potential abuse, so features like rate limiting and usage monitoring are especially important.

Designing, developing, sourcing, and securing your financial APIs is a big undertaking, but it's obviously essential for building a robust finance application. Fortunately, you don't have to build it yourself from the ground up. Tools like IEX Cloud's Apperate offer a real-time streaming data platform specifically for finance applications, greatly decreasing your time to market and the engineering resources required in this endeavor.

Conclusion

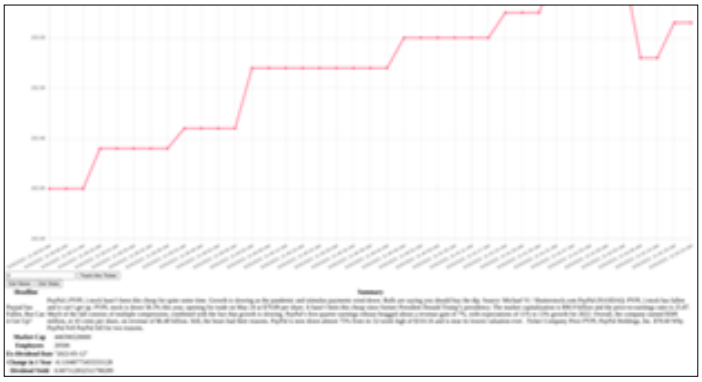
As financial data becomes more readily available and interest in creating new financial technology grows, the financial data space is likely going to get even more exciting. In this piece, you've learned about the foundations of building applications and APIs with financial data, some of the data providers available, and best practices for selecting one.

One of the easiest ways to start building on top of financial data is by leveraging a reputable, high-quality financial market data provider. IEX Cloud and its streaming data platform Apperate can help you get the data and tooling you need to build faster, more secure financial applications. So, sign up for a free trial to get started today.

How To Build a Stock Trading App

Combine data from IEX Cloud Apperate with IBKR's brokerage API to build a trading app.

Overview



This article shows you how to create a web application that queries IEX Cloud Apperate (iexcloud.io) for information on publicly traded stocks such as price data, company financial metrics and recent news articles about the stock. In addition, Interactive Brokers, a popular brokerage platform, is used to allow users to enter and execute stock orders directly from the web page. By the end of this article, you'll learn how to create rich web applications for your own use case that combine both data from Apperate and execute trades using a brokerage service, using the details in this article as a roadmap for your own implementation.

Target Audience

The target audience for this article is developers with moderate experience building web applications and integrating APIs. All code used to implement the example in this article is available on github.

Technology Stack

A web framework is dedicated software running on a server that accepts inbound HTTP requests, directs the request to the appropriate end point, and sends the response back to the requestor. There are many widely-used web frameworks such as Flask and Django. Django was ultimately chosen for this article.

Django

Django is an open-source web framework that uses the model-template-views architecture strategy to respond to web requests. The urls.py file maps incoming web requests to the views.py file which can load a specific HTML file in the templates folder. Consult the Django website for information on how to install and configure Django.

Chart.js

A key component of this project was a price chart that would constantly update the latest price data from Apperate at a regular time interval. Chart.js was identified as an easy-to-use solution. Consult the Chart.js docs for more information about this product.

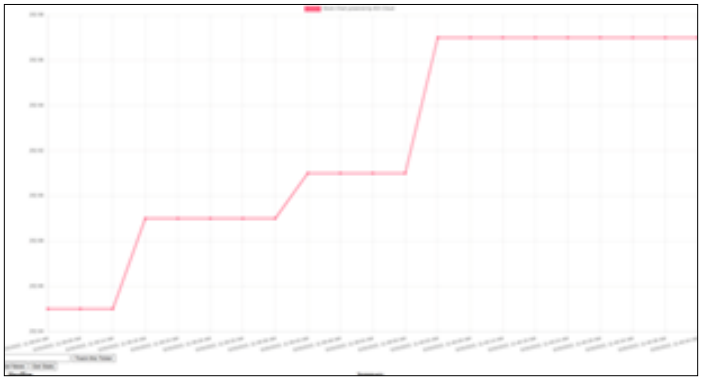
HTML

HTML was used for the front end of this project. The user interface was chosen to be functional for the purposes of the demonstration. You can make many enhancements to the UI in your implementation, for instance using Cascading Style Sheets (CSS) and other solutions. The user interface designed for this article was chosen to be simple on purpose.

Interactive brokers

Interactive brokers (IBKR) is an online brokerage with a robust API that allows developers to enter orders programmatically. IBKR has two products, Trader Workstation and IB Gateway, that can be configured to receive and execute commands from another locally run program. In this example, TWS was chosen to process buy orders created from user input on our web page running locally. For instructions on how to download and install TWS please visit their documentation here.

Building the Price Chart



To build a price chart into a dashboard, start with pulling your data from IEX Cloud Apperate.

Apperate is IEX Cloud's streaming financial data platform. It includes financial data readily available via API (also known as Core Data), as well as a suite of database capabilities that help you build and scale your application backend. For the purposes of this tutorial, you'll use Apperate's Core Data functionality. (IEX EDITOR'S NOTE: with the new data bundles introduced as of Feb. 2023 core financial data is offered for purchase through different data bundles. One of our Equities market bundles would be appropriate to use for this tutorial. Users can You can learn more about IEX Cloud Financial Data Bundles here.)
The data on Apperate is updated in real-time by API standards, although for actually executing trades many trading apps get the data from a market data vendor with cross-connect because there is still the latency of retrieving data via API.

There are front-end and back-end components for building a price chart. For simplicity, one HTML file was created for this project that handled displaying all the features including chart, company statistics, news, and user inputs like text fields and buttons. Chart.js provides the API to render datasets and labels.

Theurls.py file maps the incoming requests to the specific file and function that will be called with a simple interface: **path('', views.index_entry,name='index').**

This path entry means the landing page is mapped to index_entry where the render function is called on the chart.html file in the templates folder.

Insidethe chart.html, notice "myChart" is defined in the canvas id section. HTML will call this javascript variable later in the html file. MyChart is defined as a new Chart object with a constant config variable. The config is how Chart.js sets the type and style of the chart. See Chart.js documentation for further information on available style options. The data field is the most important config field to us. If a list of points is defined in the data list, Chart.js will immediately plot them. However, for our implementation we want a dynamically updating chart every minute as new price data becomes available in IEX Cloud Apperate, so I left the data list empty to start.

A poll function was written to enable the chart to dynamically update. This function is triggered when the “Track this Ticker” button is clicked on the page. It submits an AJAX query (https://en.wikipedia.org/wiki/Ajax_(programming)) to the /ChartData endpoint on the Django backend, passing the ticker from the text field as an input parameter. The update_price function inviews.py performs a Get request to an IEX Cloud Apperate endpoint:

```
def get_price(ticker):
    url = "https://cloud.iexapis.com/stable/stock/" + ticker + "/price?token="
    price = req.get(url)
    return price


@never_cache
def update_price(request):
    ticker = request.GET.get('ticker','')
    price = get_price(ticker)
    return HttpResponse(price)
```

https://cloud.iexapis.com/stable/stock/[ticker]/price?token=[Your token]

```
function poll() {
  $.ajax({
                url: "/ChartData",
                success: function(data) {

                  // get current eastern time
                  var today = new Date();
                  var newLabel = today.toLocaleString('en-US', { timeZone: 'America/New_York' });

                  // add new label and data to chart and update
                  myChart.data.labels.push(newLabel);
      myChart.data.datasets.forEach((dataset) => {
                dataset.data.push(data);
                  });
                  myChart.update();
    },
                data: {ticker: document.getElementById('ticker').value},
                complete: setTimeout(function() {poll()}, 5000),
    timeout: 2000
  })
);
```

Once IEX returns the latest price data, update_price returns it to the front end. The poll function in the html file will create a new label which is today’s date in eastern time zone format and push the price and label into the chart dataset. Then calling chart.update() will force Chart.js to render the latest data to the page.

Displaying Company Data

A key component of our sample fintech application is a dashboard that displays company data and statistics such as market cap, employee headcount, ex-dividend date and the dividend yield. This information is accessible from Apperate with simple HTTP requests. (IEX EDITOR’S NOTE: Access to company data is now available for purchase through one of our Company Data bundles.)

Dashboard

A “Get Stats” button will be added underneath the price chart that displays company statistics from Apperate in a table format. The backend server is responsible for submitting all statistics requests to the individual Apperate endpoints and the front end is responsible for parsing the return package and placing the individual values in rows in the HTML table.

Front End Elements

A “Get Stats” button contains an action to call a JavaScript function. The stats function submits an AJAX query to the /stats endpoint on the server, passing it the ticker symbol from the HTML text field. The reason for usingAJAX queries is to allow the rest of the page to run independently from the stats table. The price chart will continuously update while the stats table is awaiting a response from the server.

The success function of the AJAX query handles processing the returned data. For simplicity, we can return all the statistics data in a JSON object. This is straightforward to parse by calling JSON. parse(data) and then the individual field values are placed in the stats Cells for the HTML table.

```
function stats(){
    $.ajax({
        url: "/Stats",
        success: function(data) {

            parsed = JSON.parse(data);

            // Add some text to the new cells:
            statsCell1.innerHTML = parsed["marketCap"];
            statsCell2.innerHTML = parsed["employees"];
            statsCell3.innerHTML = parsed["exDivDate"];
            statsCell4.innerHTML = parsed["yrChange"]
            statsCell5.innerHTML = parsed["divYield"];
        },

        data: {ticker: document.getElementById('ticker').value},
        })
}
```

Back End Elements

The Django urls.py file maps the stats endpoint to the get_company_stats function in the views.py file. The request enters the function with the ticker symbol captured from the front-end page. The ticker symbol is concatenated with each specific URL string used to make Get requests to Apperate as well as your personal access token. For example, to get the employee headcount for a specific company the request will look like:

“https://cloud.iexapis.com/stable/stock/” +ticker + “/stats/employees?[Personal Token]”

The return from IEX is placed in a dictionary object and serialized to JSON before returning in a HTTP Response.

Displaying News

News headlines are another piece of data that users may like in addition to company statistics. This was captured the same way. On the backend, a news() function was added to views.py that makes a request to:

https://cloud.iexapis.com/stable/stock/[ticker]/news/last/1?token=[PersonalToken].

This request requires a ticker and the last number of news headlines you would like. (IEX EDITOR'S NOTE: Company News access is available for purchase through the Company News data bundle.)

For simplicity, I request only the last headline. This is returned to the web page as a JSON object.

On the front end, the news return object needs to be cleaned to remove stray characters like single apostrophes and replace them with double quotes so that JSON parsing will work correctly. After parsing, the headline and summary can be placed into cells in the HTML table.

Submitting Buy Orders to Interactive Brokers

The final feature added to the web application is a button that allows the user to submit a share purchase order to Interactive Brokers (IBKR), a popular online brokerage firm. The buy button submits an AJAX call with the ticker symbol and number of shares from the text field to the /Buy URL on the Django backend. This URL is mapped to the buy_stock function in views.py. All code for communicating with IBKR is written in a separate file called ibkr.py.

Establishing a Connection to IBKR TWS

Interactive Brokers offers a convenient API that allows developers to submit buy/sell orders for financial instruments among other capabilities. After installing the IBKR TWS application, run the GUI and select "Allow connections from localhost only" and enter 7497 (port number for paper trading) in the Socket port number field in the Global Configurations menu.

The TWS API has two classes, EWrapper and EClient that are used and overridden as required by the developer. As per IBKR TWS documentation, create a TestClient and TestWrapper class that inherits from the respective TWSclasses. Then create a third class, TestApp, that initializes the other two class objects.

After successful initialization, a connect request to IBKR's servers can be attempted via the connect call passing the localhost IP address and the paper trading port number. This call will passthrough the TWS application that is running in the background. A successful connection or refusal message will appear in the TWS application log.

```
class TestApp(TestWrapper, TestClient):
        def __init__(self):
                TestWrapper.__init__(self)
                TestClient.__init__(self, wrapper=self)
    # 4002 for gateway, 7497 for TWS, 127.0.0.1 is localhost
                self.connect("127.0.0.1", 7497, clientId=0)
```

Submitting a Buy order to IBKR TWS

A successful connect request will result in a callback from IBKR. The callback will be intercepted by the next Valid Id function. Override this function in the Test App class and store the next valid order id provided by IBKR on a class member variable. When you submit an order, IBKR will check the next valid order number in the request matches what is stored on their server. A mismatch results in a rejected order.

To facilitate placing a buy order, create a buy function in the Test App class that calls the place Order function in Test Client. Test Client will override the place Order function on TWS' EClient class, first creating an order and contract object and then calling the place order function on EClient, the parent class, via the super call.

```
def nextValidId(self, orderId: int):
                self.nextValidOrderId = orderId
                print("NextValidId:", orderId)


def buy(self, ticker, numShares):
    self.placeOrder(self.nextValidOrderId, ticker, numShares)
    self.nextValidOrderId += 1
```

Several parameters must be specified in the order object such as the order action, type and total quantity of shares which is passed in from the text field on our web application. The contract object requires the ticker symbol, security type and exchange that lists the ticker. In our application, these parameters are hard coded for simplicity, but a more flexible implementation might allow the user to set these parameters dynamically on the page.

The place order request will pass through the TWS application running in the background and any successful execution or failure messages will appear in the TWS message log. TWS will also add the purchased securities to your portfolio. Once successfully executed, the buy function must increment the next valid order id by 1 to prepare for the next buy request.

The last piece that must be implemented to allow the web application to operate independently from the IBKR client is placing the client on a separate thread via Python's threading library. This is done by creating an exec function and calling app.run() inside this function to start the connection process inside the mainline of the script.

```
import threading

def exec():
    app.run()

app = TestApp()
t1 = threading.Thread(target=exec)
t1.start()
```

```
class TestClient(EClient):
    def __init__(self, wrapper):
        EClient.__init__(self, wrapper)

    def placeOrder(self, orderId, ticker, numShares):

        # create order
        order = Order()
        order.action = "BUY"
        order.orderType = "MKT"
        order.totalQuantity = numShares

        # create contract
        contract = Contract()
        contract.symbol = ticker
        contract.secType = "STK"
        contract.exchange = "NYSE"

        super(TestClient, self).placeOrder(orderId, contract, order)
        print("placed order id: ", orderId)
```

The Python threading library will call the exec function once the thread is started. Without multithreading, the web application will block while our TWS client maintains a connection to IBKR's servers.

Summary

In this tutorial, you learned how to build a web app that:

1. Pulls financial data from IEX Cloud Apperate and displays it within a dashboard, including stock prices, company data, and other essential stats. (IEX EDITOR'S NOTE: Learn about IEX Cloud Financial Data bundles in our docs.)

2. Integrates a brokerage service API to execute trades.

Keep in mind that if you wanted to bring in data from multiple sources into your app (not just IEX Cloud), the database capabilities help you do that with speed and ease. Learn more about Apperate here.

12 Questions to Ask Your Financial Data Provider Before You Launch Your App

Creating a fintech app or website that uses financial data? Before jumping in, you need to know what you're getting into with your financial data provider. We've put together a list of 12 questions to help you navigate potential challenges and identify key success factors.

If you're creating a fintech app or website—whether it's for investment analysis, portfolio management, or financial research—you need reliable and accurate financial data. Collecting and organizing financial data yourself is time-consuming and expensive though, so you'll likely choose to source your data from a financial data provider. These are companies that collect and aggregate data from financial reports, market prices, and broker and dealer desks and provide it to different financial institutions and investors.

However, relying on a financial data provider doesn't abscond you of responsibility. You're still responsible for ensuring that the data on your app or website is accurate and timely. You should ensure that your data provider offers the data you need and that their data can integrate with your app by asking them some questions.

The questions you ask should help you do two things: First, they should answer everything you need to know about the data itself—the types of data offered, how its accuracy is ensured, and so forth. Second, they should help you get to know more about the services of the financial data provider, such as their technical support, customization options, and security measures.

This article covers twelve questions you can ask your financial data provider to determine whether they're the vendor you want to rely on when you launch your app or website. Remember that the relevance of these questions will depend on your business needs. You know your business's priorities and constraints best. Use the questions in this list as your springboard, and adapt them as you see fit.

Yes/No Questions

First, let's cover four simple yes/no questions that could help you quickly flag potential issues with some providers.

Can They Provide Sample Data to Test the Integration with Your App before Launch?

It's important to know whether your data vendor is willing to provide sample data to test the integration with their data before you launch. Detecting issues in the data once your app is launched is disastrous. You need to test it way before that. To do so, you'll need a sample of their data.

The sample you need doesn't have to be enough to build your entire app, but it should be large enough so that you can ensure the provider's data can integrate with your app.

Are There Any Limitations or Restrictions on the Use of Their Data, such as Geographic or Regulatory Constraints?

The use of some data is often restricted to certain countries or areas. If the data you get from the financial data provider comes with any geographic or regulatory constraints, you need to know about i t up front. You'll need to weigh whether it still meets your needs, or it might affect how you build your app.

Are There Any Additional Costs or Fees Associated with Their Data, such as Usage or API Fees?

You need to know the cost of using a provider's data. Obtaining data from financial data providers can get expensive. There could also be hidden costs, such as usage or API fees. Before you choose your vendor, you need to be sure you know what *all* the costs related to their service are.

Can They Provide References or Case Studies of Clients Who Successfully Integrated and Used Their Financial Data?

Any data provider could market their services as the best one out there. It's your responsibility to make sure the data provider you choose has actually helped other clients with similar needs as yours. One of the easiest ways to check this is to ask for references and case studies. You can use these to verify

Open-Ended Questions

Next, let's consider eight open-ended questions you could ask that are more nuanced. Make sure you get more than a standard sales reply, and weigh up the answers you get against best practice and the minimum requirements for your business needs.

What Types of Financial Data Do They Offer?

The type of data you need will depend on your use case. Not all vendors provide all types of data, so you need to ensure you get all the data that you need to build and launch your app.

For example, you can get some market data via the Yahoo Finance API. If you want economic data, you can get it from the IMF. If you need only one type of data, these sources could work for you. However, if you're using various data sources, each will require a different process of data preparation. Using one source gives you all the financial data you need in one place. IEX Cloud's comprehensive set of financial data, for example, includes equities market data, forex/currency data, company data, economic data, and so on.

How Often Is Their Data Updated?

Another question, which is somewhat related to the previous one, is how often the provider's data is updated. Since data changes constantly in the financial industry, it's crucial that your app should reflect that. Using data that's not current could result in wrong or useless results.

You must ensure that your financial data provider's data updates in a timely manner. How often this is will depend on the nature of the dataitself though—stock data must be updated in seconds while some economic data may not need updating for weeks.

How Is Their Data Sourced and Validated to Ensure Reliability?

The reliability of financial data—in other words, its completeness, accuracy, and consistency—is one of its most important characteristics. Fintech apps built with inaccurate data are essentially useless, and so is the data itself.

Ascertaining what a data provider's process is for sourcing and validating their data can help you detect any flaws that could lead to inaccurate and unreliable data. At the least, a financial data provider should ensure they're using the right external data sources and that they're regularly auditing the data. For example, IEX performs trillions of calculations daily to curate its data and make it readily available for you.

Which Security Measures Protect Their Data and Prevent Unauthorized Access?

To avoid security risks, you don't want someone to get unauthorized access to the data you use. You need to make sure your financial data provider has the necessary processes and measures in place to ensure their data is protected from unauthorized access.

For example, some important security measures that IEX Cloud provides are keeping audit logs for one to thirty days (depending on the package), data encryption at rest, data encryption in transit, data permissioning by domain, as well as usage monitoring.

What Technical Support or Resources Do They Provide to Help with Integrating Data into Your App?

Regardless of how good your team is, issues such as delays in the data delivery and security risks can crop up when you integrate data into your app. If this happens, you'll want to be able to get support from your financial data provider.

For example, IEX Cloud provides detailed documentation that contains everything you need to know to use its data. All IEX Cloud plans also come with customer support.

Which Customization Options Are Available to Tailor Their Data to Your Specific Needs?

Every business has different data needs, so no data provider will have standard packages to cater for every possible use case. If you have a need that a vendor doesn't cater for, it's worth inquiring whether they have customization options that will let you tailor their data to your needs. For example, IEX Cloud provides custom functions for data management in all of their plans.

Which Reports or Analytics Lets You Monitor Data Usage and Performance?

You'll want to be able to check the usage and performance of the data you use to know whether it meets the requirements of your app. Monitoring data performance will allow you to identify and resolve any performance failures early on. Tracking data usage and performance can also give you insights on how to optimize usage and improve performance.

You want to know which reports and analytics your provider offers to determine whether it lets you adequately monitor your data usage and performance.

What Is the Latency of Their Data?

Last but certainly not the least, you need to ask about latency. Latency refers to the time it takes the financial data vendor to provide the data from its source to your app. Low data latency means your app always has up-to-date information.

Data latency is extremely important in finance. For example, a fintech app through which you can trade stocks must have real-time data about current market prices. Data that's even mere seconds late will cause your users to make suboptimal decisions about buying or selling a stock. In this context, data latency is measured as the time delay between the point when a request for data is made to the point the data is received by the user. In time-sensitive fields such as fintech, the latency of data shouldn't exceed even ten milliseconds.

Conclusion

This article offered you some guidance on what to ask your financial data provider before you launch your app with their data.

As this list of questions suggests, handling real-time data comes with its challenges. Using a reputable provider of financial market data such as IEX Cloud can help you tackle these difficulties. It's even better when combined with Apperate, a cloud-based platform for managing real-time data.

IEX not only provides you with data of exceptional quality; it also offers multiple ways to connect that data into your system, shape it into products, and deliver it to your desired destination. This makes IEX fit for several different purposes, including portfolio management, risk management, market data visualization, real-time analysis, and investment analysis.

Apperate, which is integrated with IEX CloudData Bundles, allows you to build event-driven applications with serverless event processing. It provides a purpose-built real-time data store to powerthose applications and an API gateway. You can sign up for a free trial of Apperate to explore what it can offer you.

> **To request more information about how Apperate helps you build with financial data and get your app to market faster, sign up for a free trial here. https://iexcloud.io/cloud-login#/register**