Follow [Brij Kishore Pandey](https://www.linkedin.com/in/ACoAAAKDuMsBugjGZwz0pJy43LJ-6bVwc0gm9xQ) on Linkedin

**REST APIs: The foundation of modern web development**  
  
As a software engineer, I have always been passionate about sharing my knowledge and helping others to understand complex technical concepts. That's why I was so excited to create this small presentation on REST API.  
  
For those who may not know, a REST API (which stands for Representational State Transfer API) is a way for different software systems to communicate with each other through the internet.  
  
It's a crucial component of modern software development, and something that every aspiring developer should understand.  
  
Imagine you're trying to book a flight online. When you search for flights and enter your preferences, the website sends a request to the airline's computer system to find available flights that match your criteria. The airline's system then sends a response back to the website with the available flights, and you can choose one to book. This interaction between the website and the airline's system is made possible through a REST API.  
  
There are five main methods used in a REST API:  
  
GET - retrieves a specific resource or collection of resources  
POST - creates a new resource  
PUT - updates an existing resource  
DELETE - removes a specific resource  
PATCH - partially updates an existing resource  
  
The response from a REST API can either be a success or an error. A success response typically includes the requested information or a message confirming that the requested action was completed. An error response includes a message explaining why the request could not be completed.  
  
Overall, a REST API is a crucial tool for allowing different software systems to communicate with each other and exchange information seamlessly.  
  
If you're working with software systems or just want to understand how different websites and apps work together, it's important to understand the basics of REST APIs.  
  
I know firsthand how confusing and overwhelming it can be to learn about new technologies, especially when you're just starting out in the field. That's why I wanted to create this presentation - to provide a clear, concise, and easy-to-understand explanation of what a REST API is and how it works.  
  
I hope that this presentation will be helpful to those who are new to software development, or who are simply looking to learn more about APIs. And I plan to keep creating more such documents in the future, so that all the needy and new enthusiasts out there can benefit from my knowledge and experience.  
  
So if you're interested in learning more about APIs, or if you have any questions or feedback about this presentation, please don't hesitate to reach out. I'm always happy to help, and I'm committed to doing my part to make the world of software development more accessible and understandable to everyone.  
  
📌 If you like my posts, please follow  🇺🇸 🌐🌐 and hit the 🔔 on my profile to get a notification for all my new posts.

**REST API vs GraphQL**

Stay ahead of the curve in 2023 by learning GraphQL: the modern, flexible alternative to REST APIs.

As we gear up for another new year, I wanted to share some thoughts on a topic that I think will be crucial for developers in 2023: GraphQL.

If you're not already familiar with GraphQL, it's a query language for your API that allows clients to request exactly the data they need, and nothing more. This means that you can get all the data you need in a single request, rather than having to make multiple API calls to get all the data you need.

But why is this important? Well, for one thing, GraphQL can help improve the performance of your applications. With REST APIs, you often have to make multiple API calls to get all the data you need, which can be slow and resource-intensive.

With GraphQL, you can get all the data you need in a single request, which can greatly improve the performance of your application.

Another reason to learn GraphQL is that it can help improve the flexibility of your API. With REST APIs, you often have to design your API around the needs of your clients. This can be a problem if your clients have very different data needs, as you may have to design multiple endpoints to accommodate them all.

With GraphQL, clients can request exactly the data they need, which makes it much easier to design a flexible API that can accommodate a wide range of client needs.

So if you're a developer looking to stay on top of the latest trends in the field, learning GraphQL is definitely worth considering in 2023.

And if you're not familiar with the difference between REST APIs and GraphQL, the attached illustration should help demystify things for you.

**API Methods**

**Master 9 HTTP methods in just 9 minutes and stand out in 2023!**

🔸 As a developer, it's always important to stay up-to-date with the latest technologies and skills.

🔸 While it may seem like knowing just 4-5 REST API HTTP methods is enough, the reality is that there are actually 39 different verbs or methods available.

🌀 However, if you want to set yourself apart in the job market in 2023, it's crucial to master at least 9 of these methods.

👉 The GET Method allows you to retrieve data from a server.

👉 The POST Method allows you to send data to a server to create a resource.

👉 The PUT Method allows you to update a resource on a server.

👉 The PATCH Method allows you to update a resource on a server using partial data.

👉 The DELETE Method allows you to delete a resource on a server.

👉 The HEAD Method allows you to retrieve metadata from a server.

👉 The OPTIONS Method allows you to describe the communication options for a resource.

👉 The TRACE Method allows you to trace the communication path between a client and a server.

👉 The CONNECT Method allows you to establish a tunnel to a server.

✔️ Knowing these 9 methods can give you a significant advantage in the job market, as they are essential for building powerful and efficient REST APIs.

**API integration**

API integration made easy - Learn in just 5 minutes and stay ahead of the curve!

In today's digital age, understanding API integration has become increasingly important for businesses and developers alike.

But what exactly are APIs and why is API integration ?

APIs, or application programming interfaces, act as a bridge between different software systems and allow them to communicate with each other.

API integration refers to the process of connecting different software systems through APIs in order to share data and functionality.

Examples of API integration can be seen in many everyday tools and platforms.

Social media apps, for example, often use APIs to connect to other apps like Google Maps or Instagram to share information and functionality.

E-commerce platforms also use APIs to connect to payment gateways and shipping providers.

The API integration process typically involves designing and building the API, testing it, and then integrating it with other systems.

**What is an API?**

Merriam Webster defines API (Application Programming Interface) as “a set of rules that allows programmers to develop software for a particular operating system without having to be completely familiar with that operating system”. Said differently, you don’t need to know everything about the system you’re trying to interact with. The creators of the API will define everything for you, such as what data can you send them (requests) and what data you could get back (response).

Let me share an example using the Apple Maps “Ride” option. If you’re on your iOS device and you want to get a Lyft ride to the destination you’re searching for, you simply need to hit the “Ride” option. Instead of having to go out of the Apple Maps application and into Lyft, Lyft provides an API so Apple Maps can “call” and get the information it needs to show you ride options.

To break this down a bit more, when you click on the “Ride” option in Apple Maps, Apple has to send Lyft some information in a request so Lyft knows what information (response) to send back. This information might be, “where exactly are you” and “where do you want to go”. That request is sent to Lyft. Lyft reads it and sends back a response with the information requested. The information being requested by Apple Maps would probably be “what types of cars are available”, “how long until a car shows up at the starting point” and, “how much will it cost for each ride option”. Apple’s front end (User Interface) is set up to gather the response and show you the results in a usable manner.

Phone

lyft

Database

**Request**

**Response**

Cards available.

How much

what type

**API Testing Types and Their Purposes**

**Functionality:** The purpose of functional testing is ensuring you can send a request and get back the anticipated response using the requirements the API outlines. This includes negative and positive testing.

Ensure you’re covering all of the possible combinations of data. Don’t forget negative and positive testing!

Do not rely on UI testing for this. There could be bugs rooted in the unit level or backend that wouldn’t be visible via UI testing.

Error handling when data is outside of accepted parameters must be considered.

**Security:** The purpose of security testing is to make certain your communication with the API is secure and the person making the call is only allowed to do what they’ve been permitted to do.

Validate that your data is encrypted appropriately.

What type of authentication is required? Said another way, are you who you say you are?

What type of authorization should you have? In other words, are you allowed to do what you’re requesting to do?

Validate OAuth 2.0 workflows used to secure API calls:

Sending tokens, you can test the authorization and authentications from above.

**Performance:** The purpose of performance testing an API is to ensure it can handle user load and determine what happens when it reaches that load limit.

Increase the number of API calls and then monitor response times and throughput.

Monitor for memory leaks by performing an endurance test.

Stress the system out by loading it down with calls – how does it respond to failures and breakage?

**Reliability:** The purpose of reliability testing is to find possible interruptions. What happens if connectivity goes down in the middle of the test?

**Documentation:** The purpose of reviewing documentation is to validate that the documentation provides enough information to interact with the API. This is typically done as you’re testing the API. Do you have the information you need to successfully execute the testing? Would someone consuming this information know how to interact with the API?

**Integration Testing:** The purpose of integration testing is to verify success where multiple APIs are working together. Focus on call sequencing and ensure data is returned promptly and accurately.

**Types of API Protocols**

While there are multiple types of API protocols, I’ll focus on a couple of the most common ones for a quick overview.

**REST:** The most common API used today, it’s a web service API and stateless in nature. REST uses multiple standards such as HTTP, JSON, URL, and XML.

**SOAP:** One of the original protocols, it is used less often because of its strict standards. SOAP uses HTTP and XML standards.

**API Actions**

These are the various actions you can perform when API testing:

**GET:** This is the most common action and a way to simply tell the web service that you want to retrieve, or get, some information.

Use heuristics to ensure you’re not just checking sunny day scenarios

**DELETE:** This action will allow you to delete something.

Make certain to test that you can’t delete things you shouldn’t be able to. On the flip side, can you delete what you’re given authorization to?

**POST:** This action will tell the server that you want to create something new. For example, if you want to create a new product you could use this action. To do so, you’d have to input parameters so it knows what you want to create.

Ensure malicious users cannot intentionally cause harm OR unknowingly cause harm. This might include sending bad or missing parameter data like required fields.

**PUT:** This action will perform an action on something existing, like updating an existing product. Similar testing strategies might be employed between PUT and POST calls.

**Preparing to Test an API**

Before you start executing your testing, you’ll want to ensure you have outlined some testing parameters and fully understand the API:

* Does this API interact with other APIs?
* What User flows would trigger the API calls?
* Get the documentation for the API so the features and functions are understood.
* Outline positive and negative scenarios for input parameters.
* What are the API response requirements?
* What are the typical API loads? Work with your Product Manager to understand these metrics.
* Make certain you have a test environment in which to test the API.

**Testing an API**

Testing an API is as simple as submitting a request on behalf of an application (Client), using another application’s API (Server), and checking that it returns the expected response. While that seems straightforward, there can be a learning curve. The first, and greatest learning curve, is the tooling. To test an API, you’ll need to select an API testing tool to use. I will cover those options below. For now, let’s look at how to execute a non-automated, sunny-day scenario, using Postman.

Step 1: Enter the URL (using GET function) and define the parameters. In whatever tool you choose, you’ll need to tell it what API you want to test. In addition, you may need to define parameter values. If you’re looking for one to play with, here you go: https://jobs.github.com/positions.json?description=python&full\_time=true&location=sf. You can define the value for each parameter to play around with. Using that API, you can see I have three parameters I can define: description (value of “python”, full\_time (value of true), and location (value of sf). If I were testing multiple variations of these parameters, I would change the values to match my scenarios.

Step 2: Hit Send. You want to send your request, with specific parameters (the input), to the API.

Step 3: Review the response status. If you’re expecting no issues, you’d be looking for a response of 200.

Step 4: Review the response time. Determine what your performance limits are and ensure it returns within that threshold.

Step 5: Review the response body. Ensure the response body matches the expected results.

Step 6: Review the response headers. Ensure the response header matches the expected results.

**API Testing for Mobile**

API testing for mobile can be done similarly to web API testing with the appropriate testing tool such as Postman (see the section on tools).

**Best Practices and Random Advice**

Parameters to be used during testing should be called out in the test case.

Be careful if you’re using the Delete or Purge functions – those are one-time calls.

If you’re testing multiple APIs, consider the call sequencing.

Start with outlining all test scenarios before beginning; group the tests accordingly.

Make sure you look at similar APIs. Does this API follow modeling best practices? I’ve experienced an example where there were two different APIs that should call the same data. One for the current day and the other for previous time periods. The response content for each was considerably different – not good.

**Challenges of API Testing**

Documentation should be standard for any organization using APIs. However, that isn’t always the case. Without proper documentation, you can’t adequately test.

Being able to understand and write scenarios for all of the parameter combinations and call sequencing is tough. Especially if you get a large API with many parameter options intertwined with other APIs. In this case, I would use something like Equivalence Partitioning Testing to avoid over-testing and a Decision Table to ensure I hit all the combinations.

Learning an API testing tool can be challenging. Thankfully there are tons of YouTube videos of people eager to help. More information on that in the next section.

Exception and error handling are both challenging, as those things are rarely defined for you. Work with your dev team to understand what those exceptions or errors might be.

It’s intimidating. API testing is technical which causes some people to shy away from doing it. Start with basic testing. Use basic parameterization using simple GET calls. Work your way forward from there.

**API Testing Tools and How to Resources**

I was all hunkered down to outline all the testing tools I’ve used or heard good things about. And then, the Twitter Gods bestowed upon me a list of the top 10 API tools by Testim, which was just updated and re-published in early 2021. Instead of reinventing the wheel, I’ll link you to their article on it: <https://www.testim.io/blog/the-9-api-testing-tools-you-cant-live-without-in-2019-2/>

My personal tool of choice is Postman. It’s what I learned on and I’m comfortable it can handle everything I throw at it.

Additionally, I want to share some personal links to places that I’ve learned quite a bit from over time. I’m dollar-conscience so these are all FREE learning opportunities. You’ll find everything from learning about API testing to learning specific tools you might choose.

* **Restful Booker:** This is a website that hosts several APIs, with documentation, so you can play around and learn more. You can create, read, update and delete on this! <https://restful-booker.herokuapp.com/>
* **LinkedIn:** LinkedIn will give you a 30 day free trial to their learning platform. It would be worth signing up to take advantage of all the courses they have on API Testing. <https://www.linkedin.com/learning/search?keywords=API%20testing>
* If you’re going with Postman, you have to check out this Github repo. Danny Dainton captures “All Things Postman”. <https://github.com/DannyDainton/All-Things-Postman>.
* If you get stumped and need some real-time help, Ministry of Testing has a free slack world you can join and a dedicated #api-testing channel that stays pretty active. <https://www.ministryoftesting.com/slack_invite>

**Conclusion**

I truly hope this API testing guide has helped you get a better grip on this aspect of software testing. For so many, API testing remains an overwhelming and intimidating testing phrase. Use the resources I’ve provided you, pick out a tool that looks best for you, start practicing, and in no time you’ll be more confident in your API testing.

Let me know in the comments, are you starting to learn more about API testing? Any other tips and resources that you would add?

(source: <https://abstracta.us/blog/software-testing/api-testing-guide/>)

**API Scaling**

**How can you choose the features for your system?**

To answer this question, you need to know the future load on your future. You can do it by forecasting the possible load or analyzing customers’ requirements. For this project, we were lucky because the customer knew the approximate future load on the platform, so we could start designing a scaling scheme right away.

When a customer doesn’t know the estimated load, we assess the project and provide several scaling options depending on our experience with previous projects.

Here’s an example of our proposal for a system that processes 3, 30, and 300 requests per second. This comparison helps the customer choose between an affordable cloud environment and rich scalability options. We used the AWS Pricing Calculator to show the cost of each option.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **3 req/sec** | **30 req/sec** | **300 req/sec** |
| REST API | SQS  Lambda  DynamoDB | Application Load Balancer  EC2  DynamoDB | Application Load Balancer  EC2  DynamoDB |
| AWS price | $246 / month | $2,920 / month | $25,682 / month |
| Integration time (DevOps time) | 40 hours | 80 hours | 100 hours |

Our calculations include the option of a gradual load increase and changes to the system depending on the load. Also, the customer can choose the three requests per second option for a prototype and then add scaling and debugging to it.

**How can you choose instances for scaling?**

The correct choice of instances for a project helps you rent the right amount of cloud computing resources to work under a certain load. It saves your project from crashes and budget bloat. But the only way to know for sure which instances you need is to build a prototype and conduct load testing. That’s exactly what we did for our project:

* Wrote a simple application prototype with unchangeable logic.
* Launched the easiest and cheapest instance on AWS and loaded it until it failed.
* Saw how many queries this instance completed and which criteria made it fail.
* Launched a more powerful instance and tried to crash it with our prototype.

Usually, an instance crashes because of CPU or RAM overload, traffic limits, or third-party service limits. We used JMeter to find the maximum load on AWS instances until we found the most suitable ratio of instance parameters and time to crash.

Knowing the approximate minimum load on the project and the capacity of one machine, we could choose an optimal instance. For example, during development and testing of our customer’s system, up to 10 people worked with it simultaneously. Usually, you only need to keep one machine running for domain development.

Load testing helped us find out that for our customer’s system, the most suitable instance configuration was 4 CPUs, 8 GB RAM, unlimited traffic, and no third-party services limits. Such an instance can process 60 to 120 requests per minute, or 2 to 3 requests per second, before crashing.

After testing the prototype, we could predict how many simultaneous requests one instance could handle and how many instances the project would need. Based on this information, we offered our customer the optimal number of instances, determined their optimal configuration, and calculated their cost.

**How can you design and debug a cloud scaling system?**

Unfortunately, you may not understand why a new system crashes in production. But crashes may happen if development teams neglect load testing.

When we tell a client that hosting an application in a scalable cloud service will solve performance issues, we mean that we’ve already tested the system under a heavy load.

When you work on a new system, creating a prototype helps you plan and estimate the project, but it won’t solve all API performance issues. Connecting a prototype to the application load balancer is not enough, since problems start when you run several instances.

For our customer’s project, we managed to debug the following issues with scaling system performance:

Launching a new instance took five to seven minutes. During this time, the load could increase by twofold or threefold, crashing the instance upon launch. We managed to launch a new instance in three minutes by reducing the number of scripts started when an endpoint launches. We did it by creating a custom AMI image for AWS EC2 instance. The new AMI already has all libraries and we don’t need extra time to launch it.

The servers worked in the alarm zone for about one minute. After the alarm about it, we launch a new instance within two minutes.

Failure of one API request can affect other requests. Our system frequently used browsers for its work, which have their own performance limitations. When one request overloaded a browser, it couldn’t process subsequent requests. To fix this, we isolated each process and limited the load by creating internal requests and a request queue for each server.

The first step was to create a worker that would run our program independently so that in case of errors, we would lose only one request instead of all operations. This worker takes one task at a time from the queue after completing the previous one. In our case, the CPU load per worker was 7%.

Next, we increased the number of workers until the CPU was 80% loaded. This way, we got an isolated process with a predictable load and created a number of parallel tasks for the instance to process.

Load balancer evenly distributes the load between instances by default. Because of this, instances that launch first have a two to three times higher load than others, leading to a higher chance of crashing. We made the application load balancer send new requests to instances with the fewest active connections until all instances had an even load.

Our API keeps the connection active until it gets a response. Under a heavy load, it took our system three to five minutes to respond, and during this time the application load balancer ended the connection. We increased the maximum connection time for the API to three to four minutes. During this time, we can create a new instance if needed.

EC2

Load Balancer

Queue

Worker

API

EC2

EC2

EC2

Worker

Worker

Worker

**Scaling scheme for our system**

This approach allowed us to design an optimal way of scaling a high-performance API and our cloud system. API requests are distributed between several workers and AWS instances, giving them enough time to provide a response. Under peak load, we can create additional instances and avoid system crashes.

However, this method of scaling has several disadvantages you need to be aware of. The request queue inside the instance helps to manage API performance, but it shouldn’t accumulate more requests than the instance can process within three minutes. Since the API connection lasts for three to four minutes, the API won’t get responses to requests that take longer. That’s why we need to launch a new instance during this time.

Also, you need to keep track of the minimum number of requests for your system and add two or more instances if you anticipate a load spike. You can automate this by writing a script that controls the minimum number of interfaces.

**Conclusion**

Our experience implementing the scaling scheme we’ve described shows that there are several conditions you have to keep in mind to avoid performance issues:

Isolate processes using workers, which gives control the application over the performance of the instance.

Increase the lifetime of the API connection by the period necessary for the request from the queue to be executed in the worker.

If you use external services, remove restrictions on internet traffic.

Configure the load balancer to evenly distribute traffic between instances.

Taking into account future load and potential performance issues is an essential step in API development and the design of high-load cloud infrastructure. That’s why at Apriorit, we build a prototype of a client’s system and conduct load testing before we start development. It helps us figure out how to scale API performance in a high-load system, design the best possible scaling scheme for a particular project, and reduce the cost of cloud resources.

Leverage our expertise to make your cloud system work smoothly under any load!

(source: <https://www.apriorit.com/dev-blog/776-cloud-api-scaling>)

**Designing APIs with Postman**

**API development overview**

Postman supports API-first development with the API Builder. Use the API Builder to design your API in Postman. Your API definition can then act as the single source of truth for your API project.

You can connect various elements of your API development and testing process to your API definition, such as collections, documentation, tests, and monitors. You can also sync your API in Postman with a Git repository.

**High-level steps**

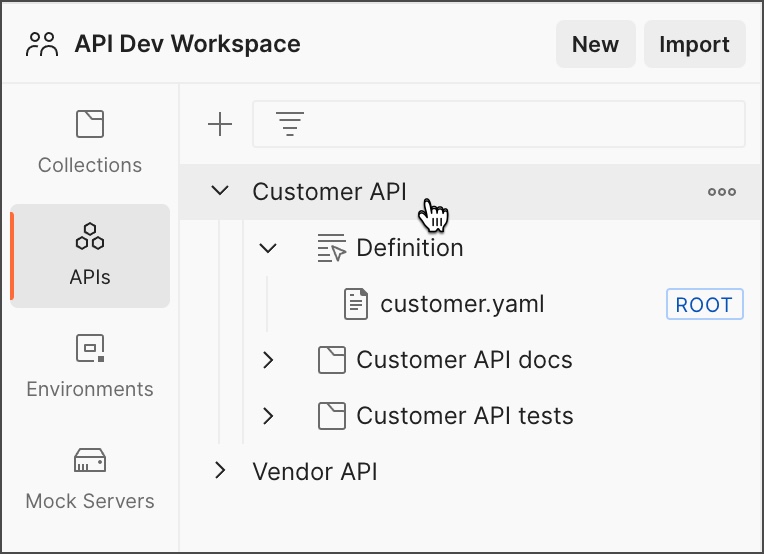
The following are the high-level steps involved when designing and developing APIs in Postman:

* Creating an API - Start a new API in Postman. (You can also import an API.)
* Using API version control - Sync your changes between Postman and a remote Git repository. When you're ready to share your changes with consumers, publish a version.
* Developing an API - Build out your API by editing your API definition and adding collections. You can also generate server-side code from your OpenAPI 3 specification and validate your API definition and linked elements.
* Testing an API - Add tests to your API, and use CI integration to build and automate testing.
* Deploying an API - Connect your API deployment service to have greater visibility within Postman.
* Observing an API - Add monitors to observe your API.
* Managing and sharing APIs - Share your API with other team members, add comments, view the changelog, and watch for changes.
* Mocking an API - Set up a mock server to simulate your API's endpoints for testing or development purposes.
* Using API reports - Use reports to view statistics on how APIs are being used, developed, and improved by your team.

**Navigating the API Builder**

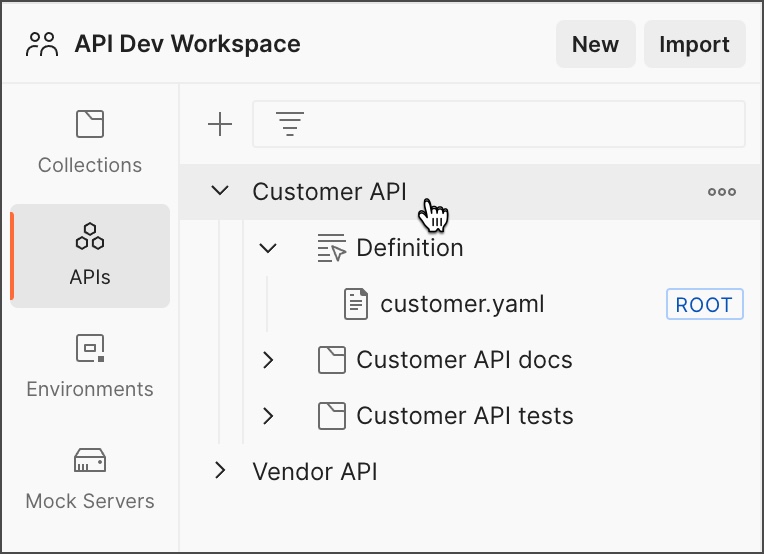
To view and work with APIs, select APIs in the sidebar. You can open and edit any existing APIs from here, or create or import new APIs. Use the arrows to expand an API in the sidebar and view the API's definition files and linked elements.

**API Builder sidebar**



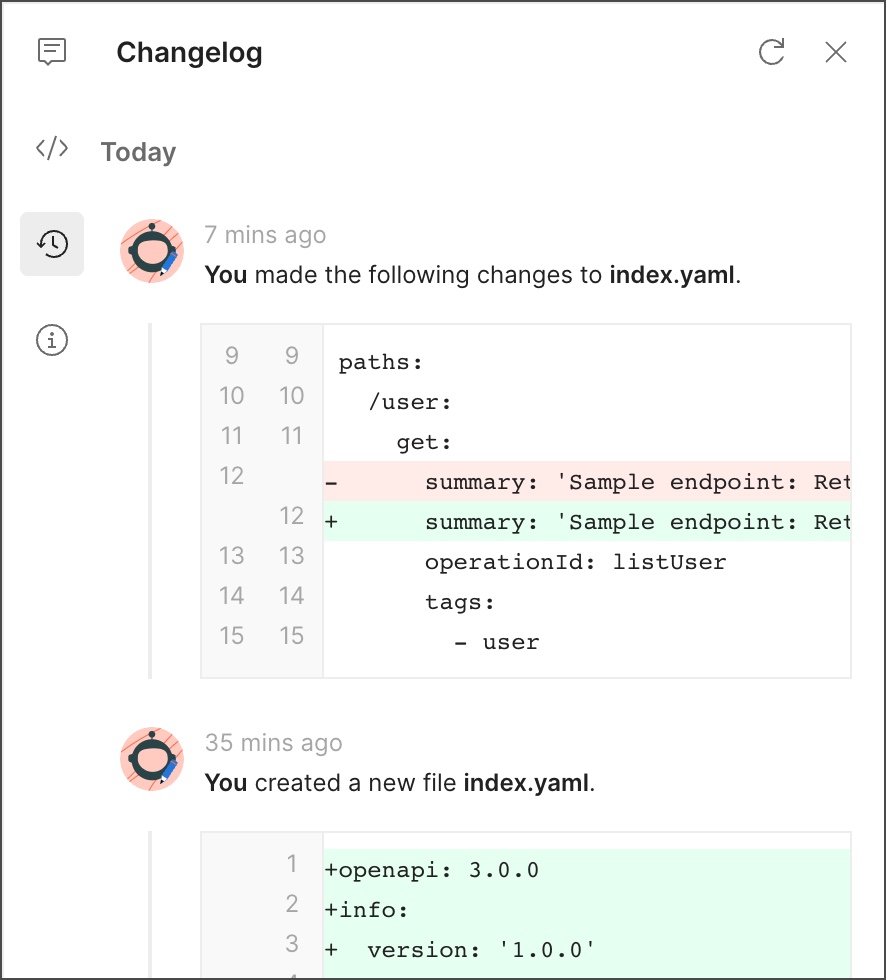
When you select an API in the sidebar, the workbench displays an overview of the API. The overview displays a description of the API, linked collections, and published versions. The overview also provides quick access to tasks such as connecting to a repository, adding tests, monitoring the API, and viewing deployments.

**API Builder overview**



The right sidebar offers more options for working with your API. You can view comments, generate server code from your API definition, and view the API's changelog.

**API Builder changelog**



For API's connected to the Git repository, the changelog is replaced by the Source Control pane, which you can use to sync your changes with a remote Git repository.

Reference Links

 1. API and REST API Fundamentals - <https://lnkd.in/e8eMet_k>  
      
 2. API Methods - <https://lnkd.in/ey9v7-hU>  
      
 3. API Terminologies - <https://lnkd.in/eRsPMzpd>  
      
 4. API Authentication - <https://lnkd.in/eNPfpAdE>  
      
 5. API Status Codes - <https://lnkd.in/egXizUrS>  
      
 6. REST API vs GraphQL - <https://lnkd.in/eZHREdgC>  
      
 7. API Integration - <https://lnkd.in/eDASPP5m>  
      
 8. API Testing - <https://lnkd.in/emgmWJqH>  
      
 9. API with Python - <https://lnkd.in/eM23ah2y>  
      
10. API Scaling - <https://lnkd.in/e3mZSvmn>  
      
11. Designing and Developing Robust APIs - <https://lnkd.in/eBXzbFyg>  
      
12. Designing APIs with Postman- <https://lnkd.in/ezue3d4B>  
      
13. Testing APIs with Postman - <https://lnkd.in/eCPnGTGi>