New Guideline: Implementing a Microservice in Golang
Tim Engbrocks authored 2 minutes ago

9ceb7b80

Code owners Assign users and groups as approvers for specific file changes. Learn more.

gu_implementing_a_microservice_in_golang.md 7.94 KiB

Implementing a Microservice in Golang

Using a PostgreSQL Database

To work with a PostgreSQL database the ORM (object-relational manager) GORM is used. Install the following dependencies in your Golang project:

```
go get -u gorm.io/gorm
go get -u gorm.io/driver/postgres
```

The full documentation of GORM can be found here: GORM Docs

Connecting to a PostgreSQL Database

To connect GORM to a PostgreSQL database, execute the code snippet below in your application. HOST, USER, PASSWORD, DB_NAME, and PORT need to be replaced by the values for your database. The `db` variable will later be used to perform operations on the database.

```
dsn := "host=HOST user=USER password=PASSWORD dbname=DB_NAME port=PORT sslmode=disable"
db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
```

Migrations

Migrations can be used to create a representation of your Golang structs in a relational database. Simply call the function `AutoMigrate` on the `db` variable that was created earlier with a reference to the struct that you want to use with your database. Golang structs can be annotated by following the type of a struct's field with backticks. Use the annotation `gorm:"primaryKey"` to mark a struct's field as the primary key of the respective database table.

```
type User struct {
  Name string `gorm:"primaryKey"`
```

```
db.AutoMigrate(&User{})
```

You can also link different structs together by creating foreign keys. Simply annotated the referenced struct with `gorm:"foreignKey:Number"` where Number is the primary key of the referenced struct. Note that the `CreditCard` struct has to be migrated first because the `User` struct needs the `CreditCard`'s table to already exist.

```
type User struct {
  Name string `gorm:"primaryKey"`
  CreditCard CreditCard `gorm:"foreignKey:Number"`
}

type CreditCard struct {
  Number   string `gorm:"primaryKey"`
  UserName string
}

db.AutoMigrate(&CreditCard{}, &User{})
```

CRUD Operations

The following are examples of performing simple CRUD (Create-Read-Update-Delete) operations with GORM.

To save an instance of a struct to your database, call `Create` on the `db` variable. The returned `result` variable contains information about the operation like `result.Error` which is either an error or nil depending on if the operation succeeded or not.

```
user := User{Name: "Max Mustermann"}
result := db.Create(&user)
result.Error // returns error or nil
```

To retrieve a single (or the first) instance of a struct, call `First` on the `db` variable with a reference to an empty instance of that struct. GORM will place the data that it found in the database into that instance.

```
var user User
result := db.First(&user)
```

To retrieve multiple instances of a struct, call `Find` on the `db` variable with a reference to an array of empty instances of that struct. GORM will place the data that it found in the database into that array. The returned `result` variable contains information about the operation like `result.RowsAffected` which represents the number of returned instances.

```
var users []User
result := db.Find(&users)
result.RowsAffected // returns count of records found
```

To conditionally retrieve instances of a struct, chain the `Where` function into your call. `Where` can contain any valid SQL WHERE-clause conditions. The `?` in the condition will be replaced by the second argument to `Where`.

```
var user User
result = db.Where("name = ?", "Max Mustermann").First(&user)
```

```
var user User
result := db.First(&user)
user.Name = "Max Müller"
result = db.Save(&user)
```

To delete an instance of a struct that has been saved in the database, first find the instance in the database with a call to `Where`. To this call you can simply chain a call to the function `Delete` which will delete the found instance. `Delete` also returns the deleted instance just like the read operations.

```
var user User
result = db.Where("name = ?", "Max Müller").Delete(&user)
```

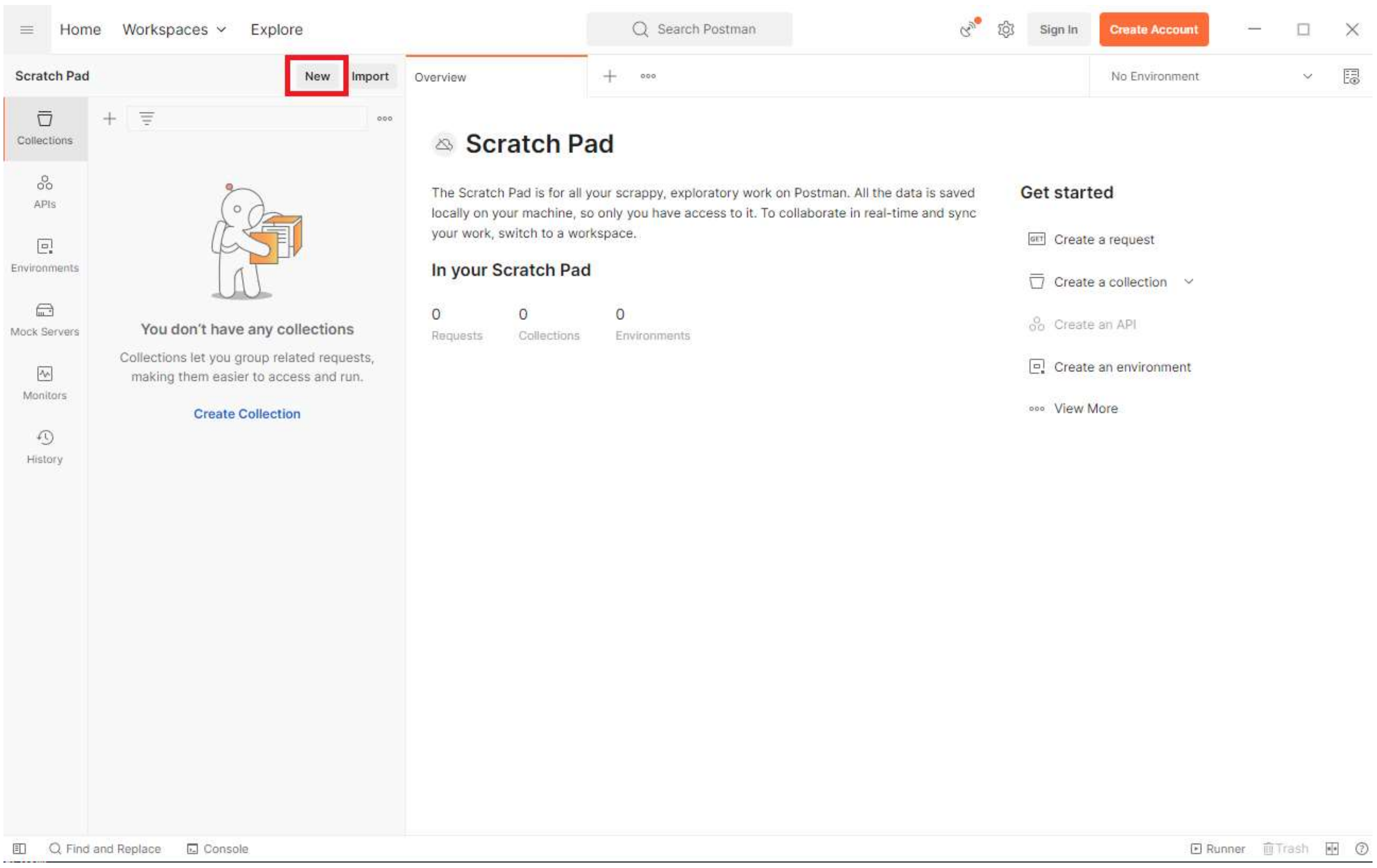
Sending Requests to a HTTP API

The tool Postman can be used to send requests to a HTTP API. To download and install Postman for your platform, follow this guide: [Installing Postman](). Note that an account is not needed to use the desktop software.

The full documentation of Postman can be found here: [Postman Docs]()

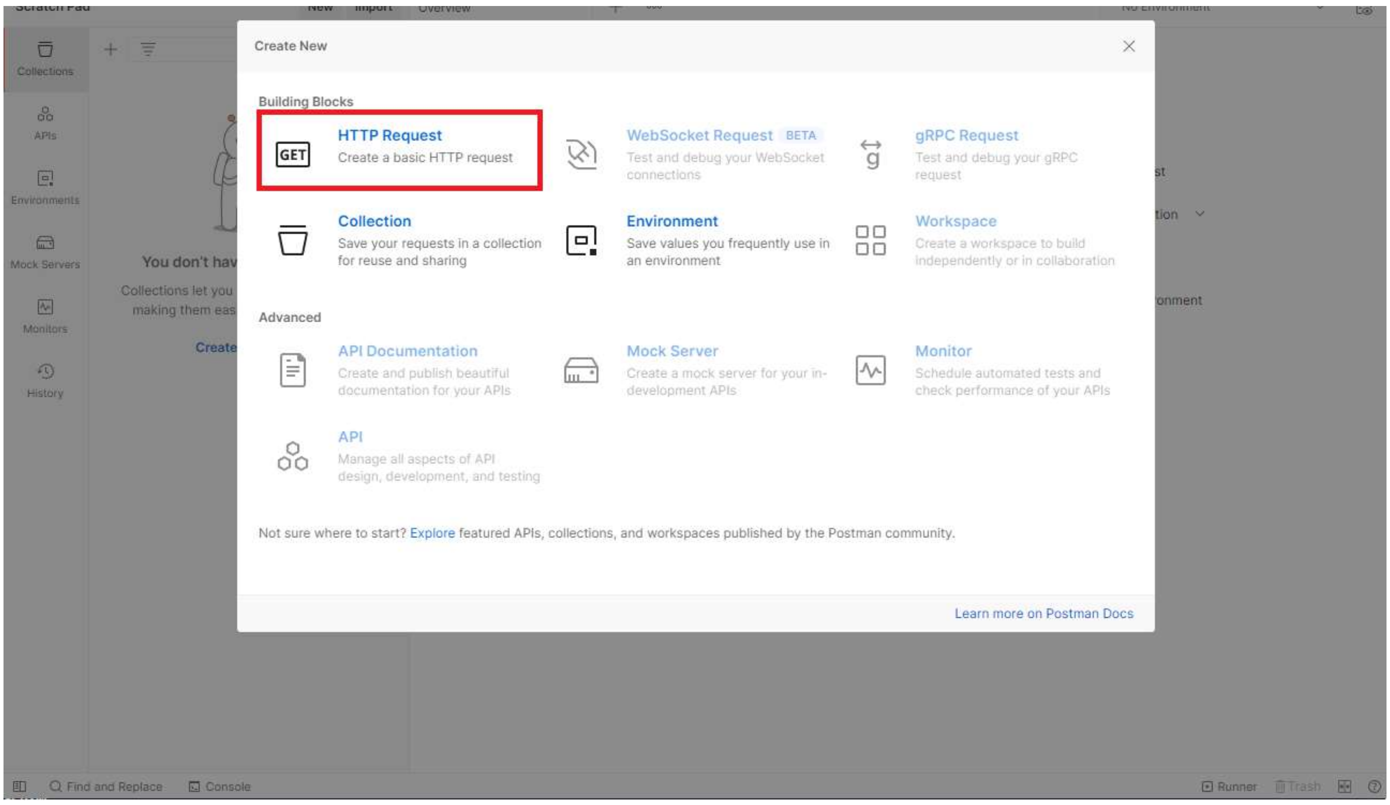
Creating a Request

To create a HTTP request with Postman, first click the New button which is highlighted in the image below.



The Scratch Pad is for all your scrappy, exploratory work on Postman. All the data is saved locally on your machine, so only you have access to it. To collaborate in real-time and sync your work, switch to a workspace.

Then choose HTTP request.



Create New

Building Blocks

HTTP Request Create a basic HTTP request

WebSocket Request BETA Test and debug your WebSocket connections

gRPC Request Test and debug your gRPC request

Collection Save your requests in a collection for reuse and sharing

Environment Save values you frequently use in an environment

Workspace Create a workspace to build independently or in collaboration

Advanced

API Documentation Create and publish beautiful documentation for your APIs

Mock Server Create a mock server for your in-development APIs

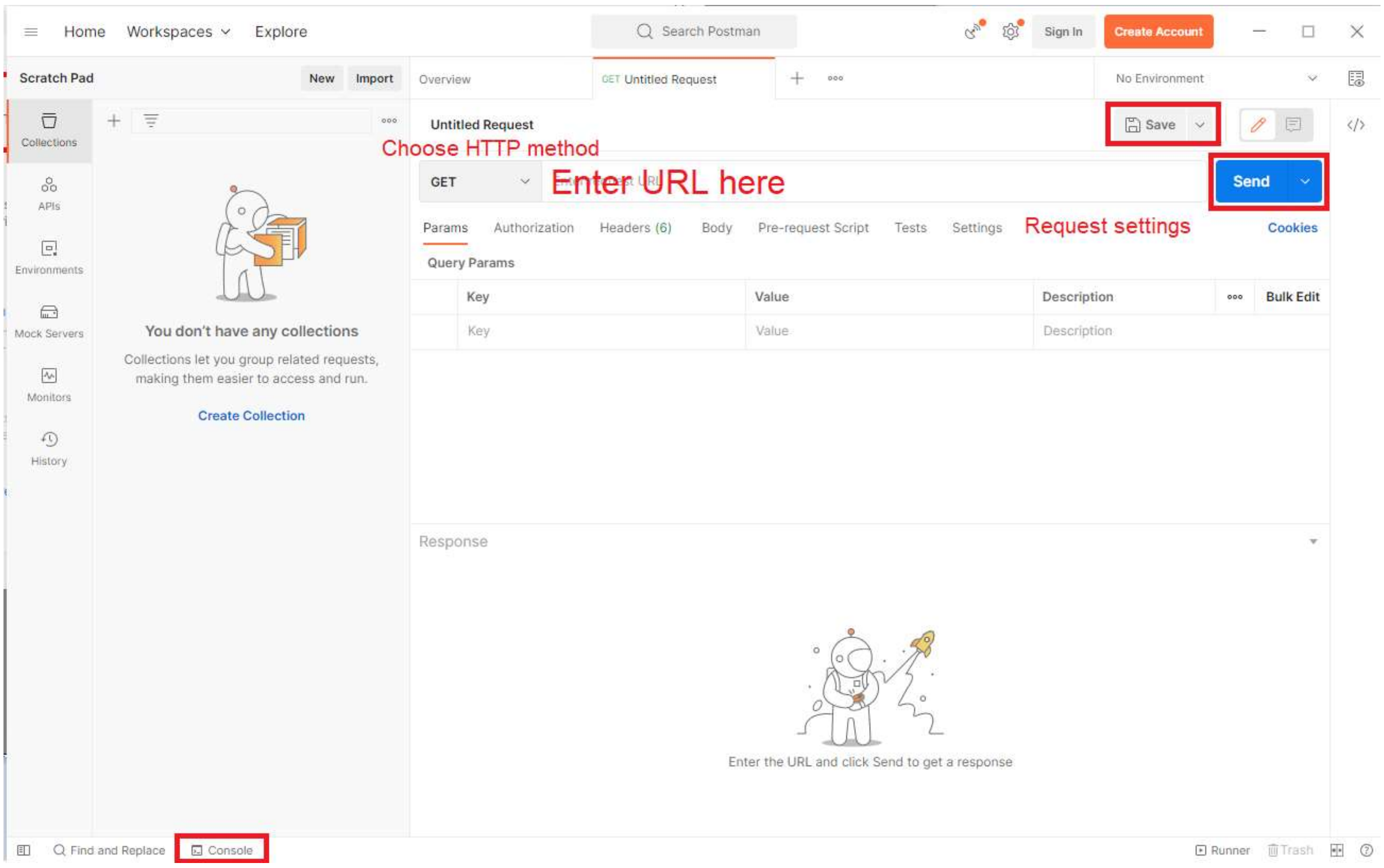
Monitor Schedule automated tests and check performance of your APIs

API Manage all aspects of API design, development, and testing

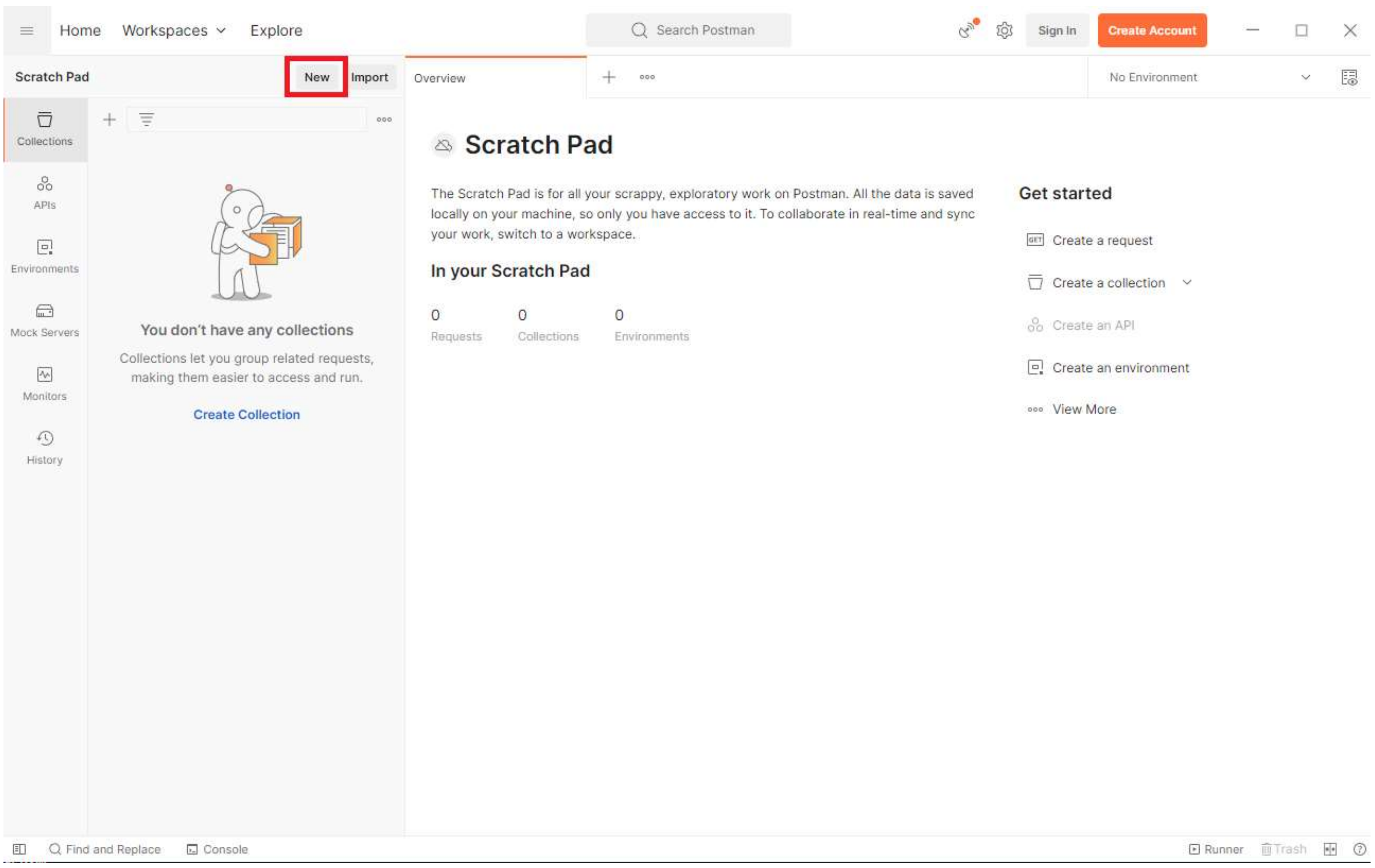
Not sure where to start? Explore featured APIs, collections, and workspaces published by the Postman community.

Learn more on Postman Docs

You can now enter the URL of the HTTP request that you would like to send. Next to the URL textfield you can also choose the HTTP method that should be used. Below that, you can set different options for the request like setting query parameters, headers, and the request body. Once you have set up your request, click the highlighted Save button so that your request will be persisted and you can reuse it later on. You can now send the request by clicking the highlighted Send button. The response to your request will be shown below the request settings. A more detailed log of the request can be found in the console which is located at the bottom of the page.



Collections can be used to group and persist requests. To create a collection, first click the new button.



Scratch Pad

The Scratch Pad is for all your scrappy, exploratory work on Postman. All the data is saved locally on your machine, so only you have access to it. To collaborate in real-time and sync your work, switch to a workspace.

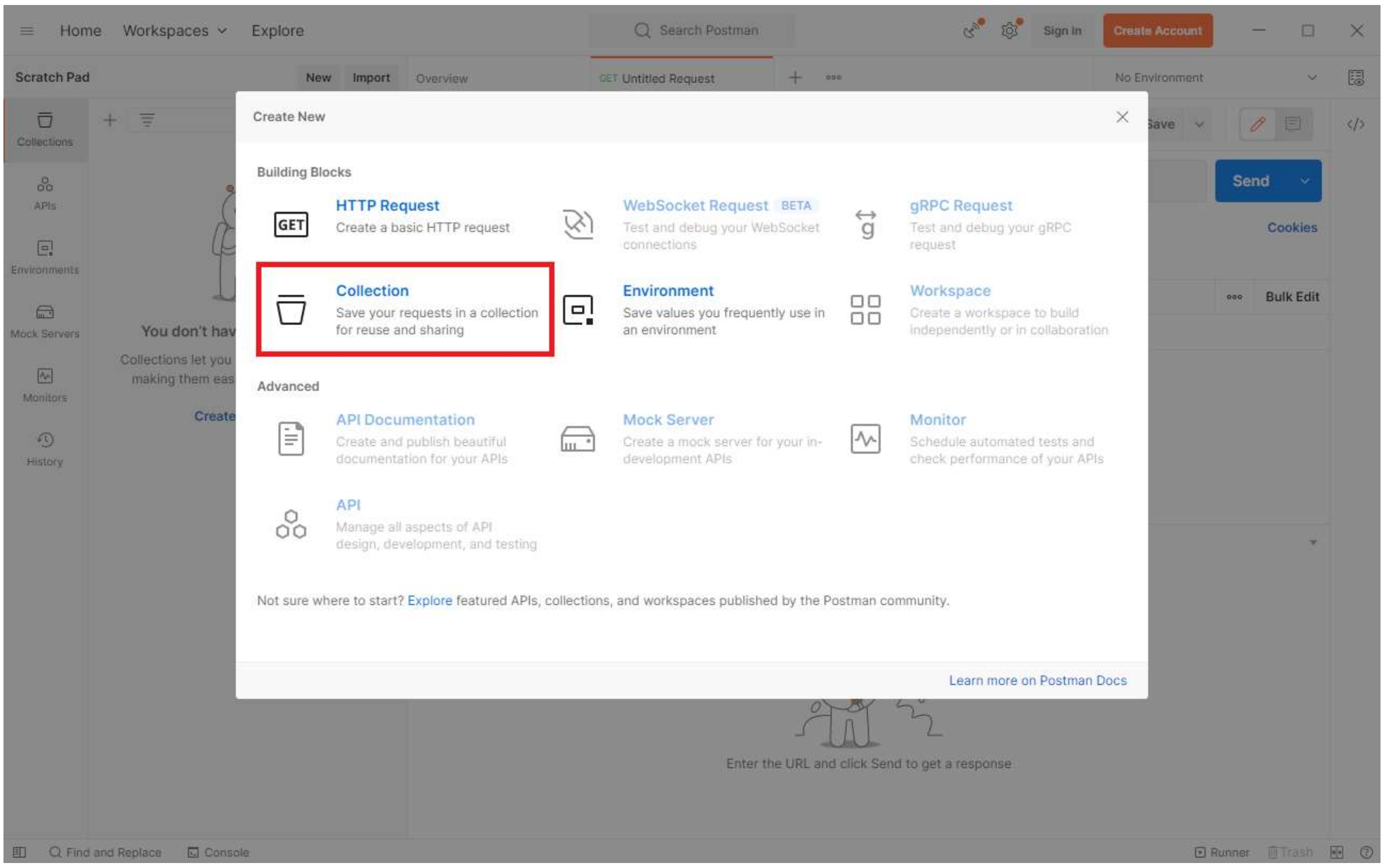
In your Scratch Pad

0 Requests 0 Collections 0 Environments

You don't have any collections

Collections let you group related requests, making them easier to access and run.

Then choose Collection.



HTTP Request — Create a basic HTTP request

WebSocket Request — Test and debug your WebSocket connections

gRPC Request — Test and debug your gRPC request

Collection — Save your requests in a collection for reuse and sharing

Environment — Save values you frequently use in an environment

Workspace — Create a workspace to build independently or in collaboration

API Documentation — Create and publish beautiful documentation for your APIs

Mock Server — Create a mock server for your in-development APIs

Monitor — Schedule automated tests and check performance of your APIs

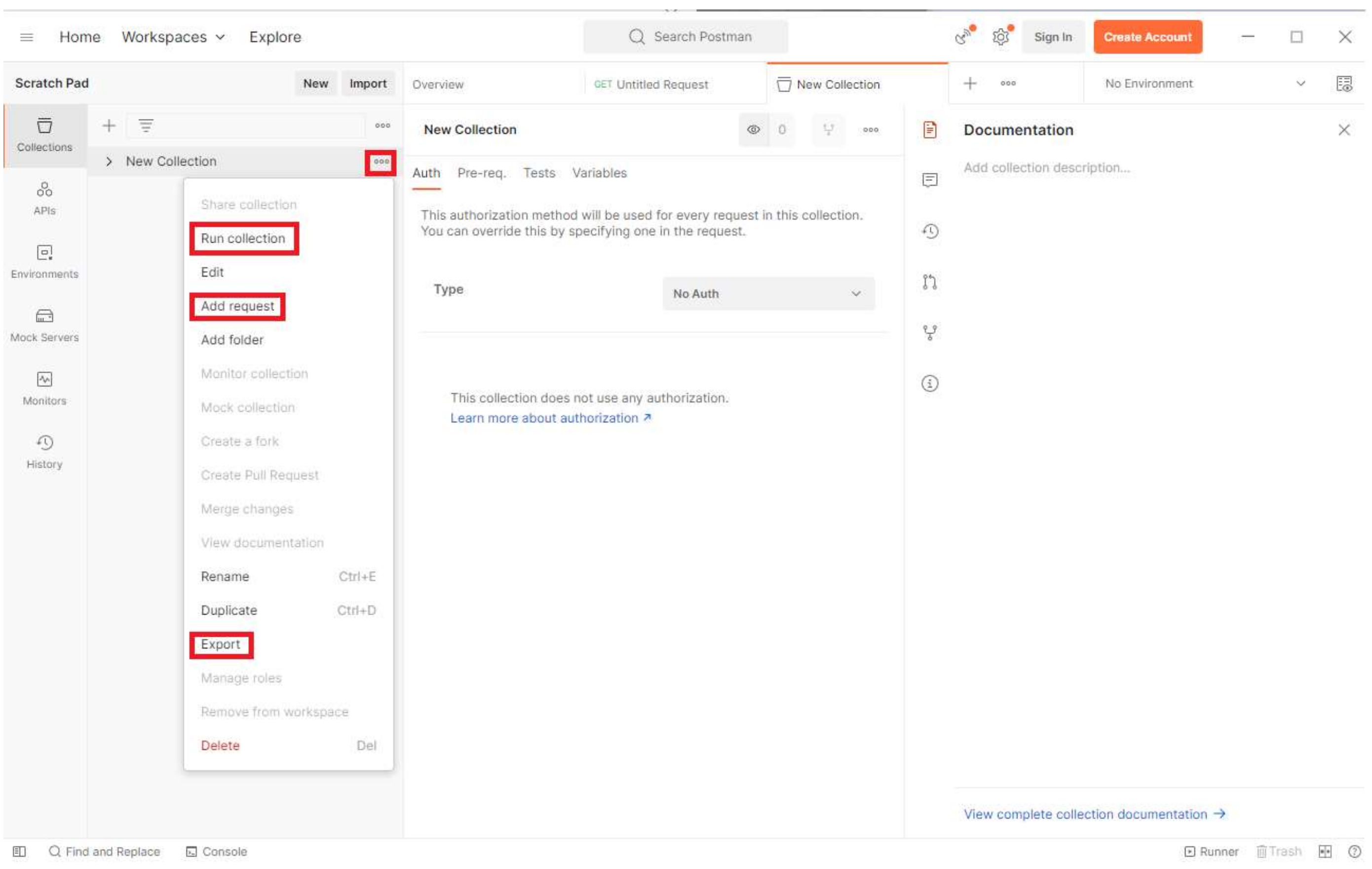
API — Manage all aspects of API design, development, and testing

Not sure where to start? Explore featured APIs, collections, and workspaces published by the Postman community.

Enter the URL and click Send to get a response

The most important features of a collection are:

- Run collection: Runs all requests inside of the collection in order.
- Add request: Add a new request to the collection.

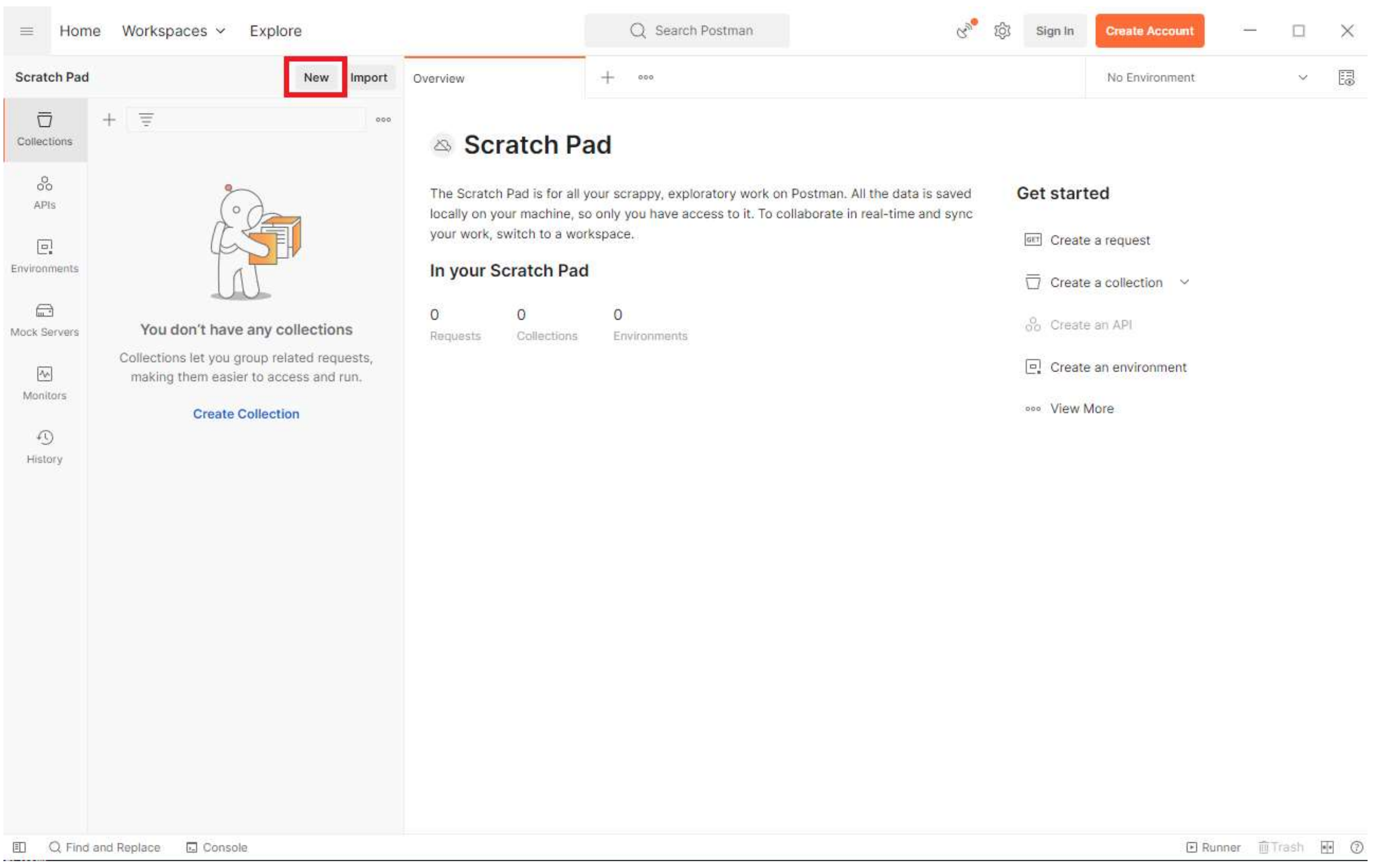


This authorization method will be used for every request in this collection. You can override this by specifying one in the request.

This collection does not use any authorization.
Learn more about authorization ↗

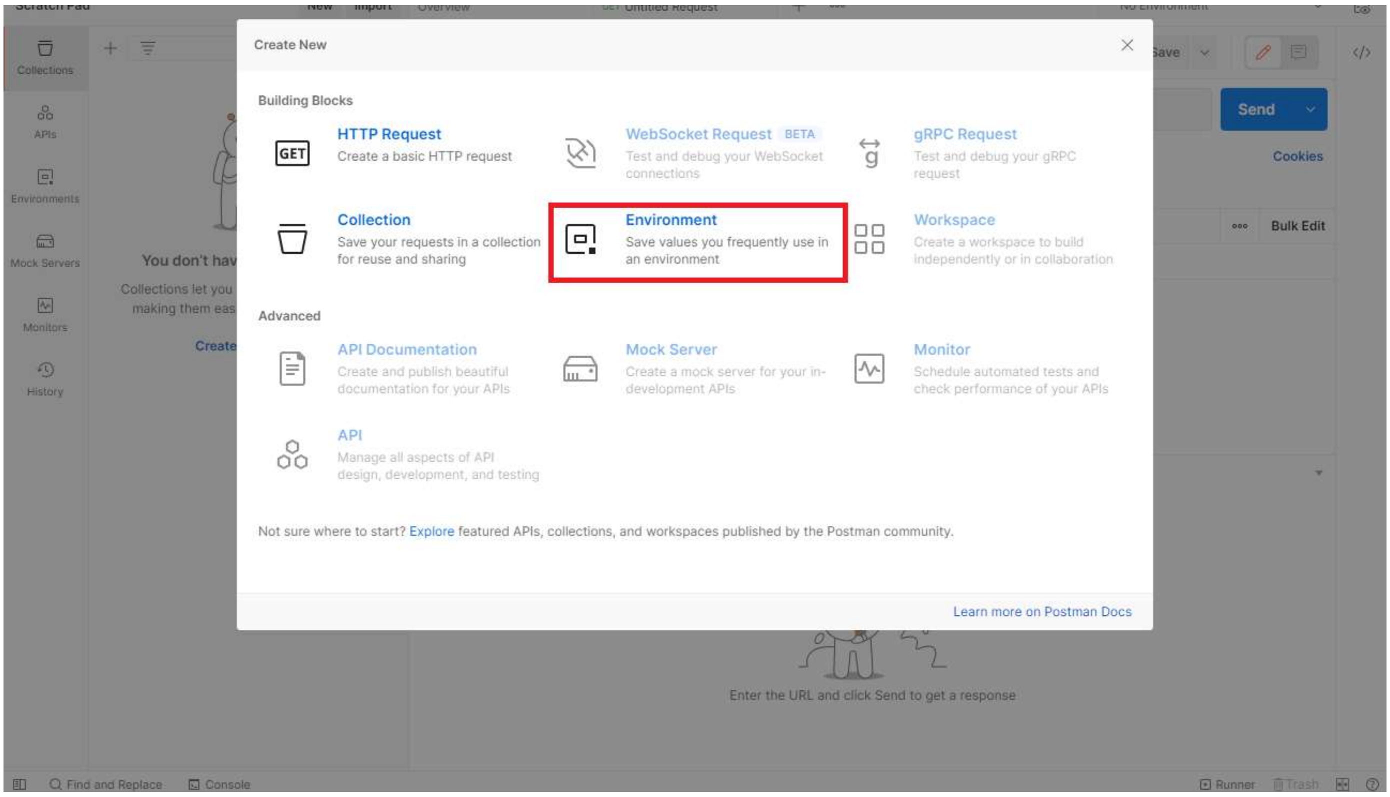
Using Environment Variables

Postman has the option to create environments which contain variables whose values can be reused across different requests. To use variables, you must first create a new environment by clicking the New button.



The Scratch Pad is for all your scrappy, exploratory work on Postman. All the data is saved locally on your machine, so only you have access to it. To collaborate in real-time and sync your work, switch to a workspace.

Then choose Environment.



Create New

Building Blocks

HTTP Request
Create a basic HTTP request

WebSocket Request BETA
Test and debug your WebSocket connections

gRPC Request
Test and debug your gRPC request

Collection
Save your requests in a collection for reuse and sharing

Environment
Save values you frequently use in an environment

Workspace
Create a workspace to build independently or in collaboration

Advanced

API Documentation
Create and publish beautiful documentation for your APIs

Mock Server
Create a mock server for your in-development APIs

Monitor
Schedule automated tests and check performance of your APIs

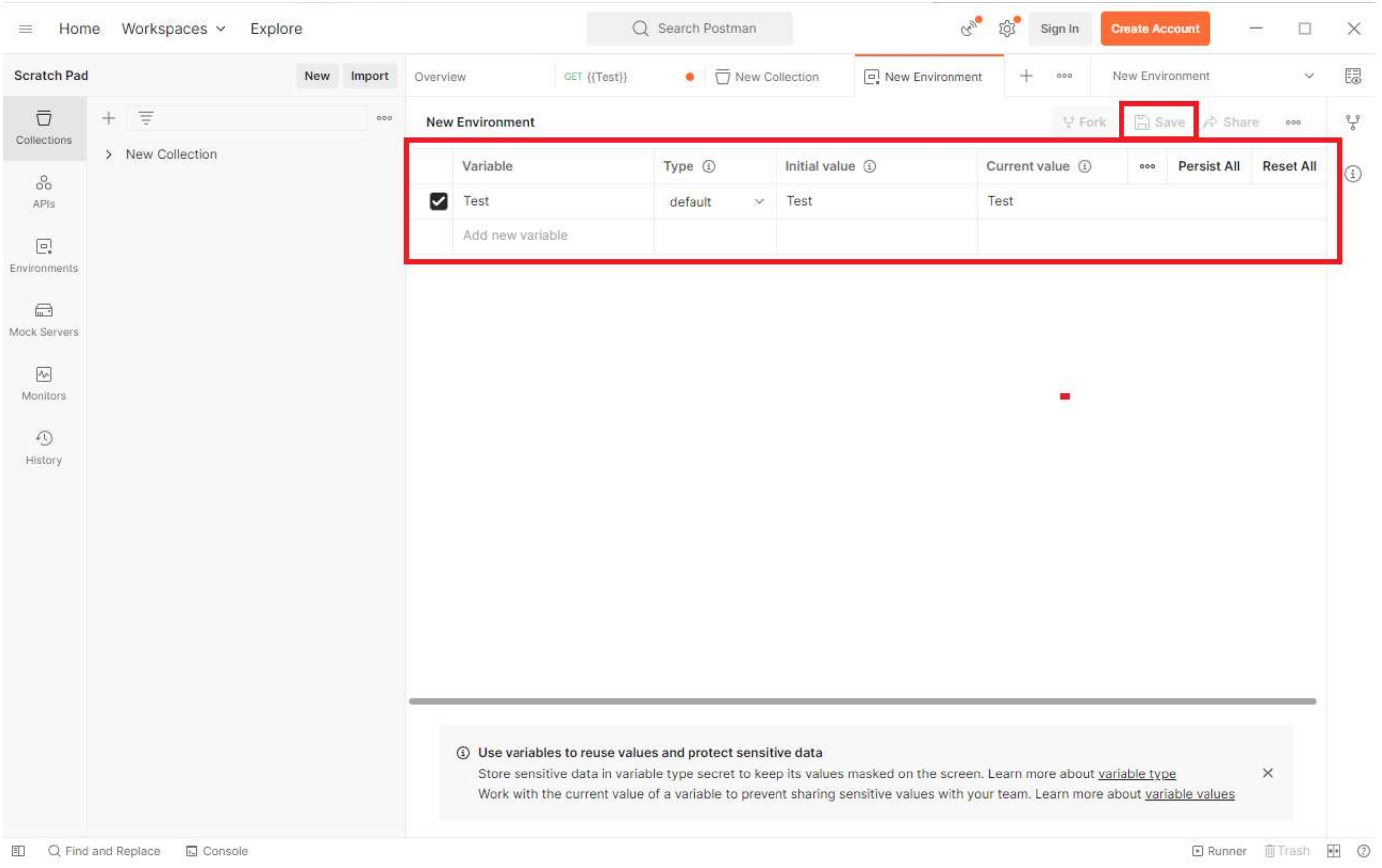
API
Manage all aspects of API design, development, and testing

Not sure where to start? Explore featured APIs, collections, and workspaces published by the Postman community.

Learn more on Postman Docs

Enter the URL and click Send to get a response

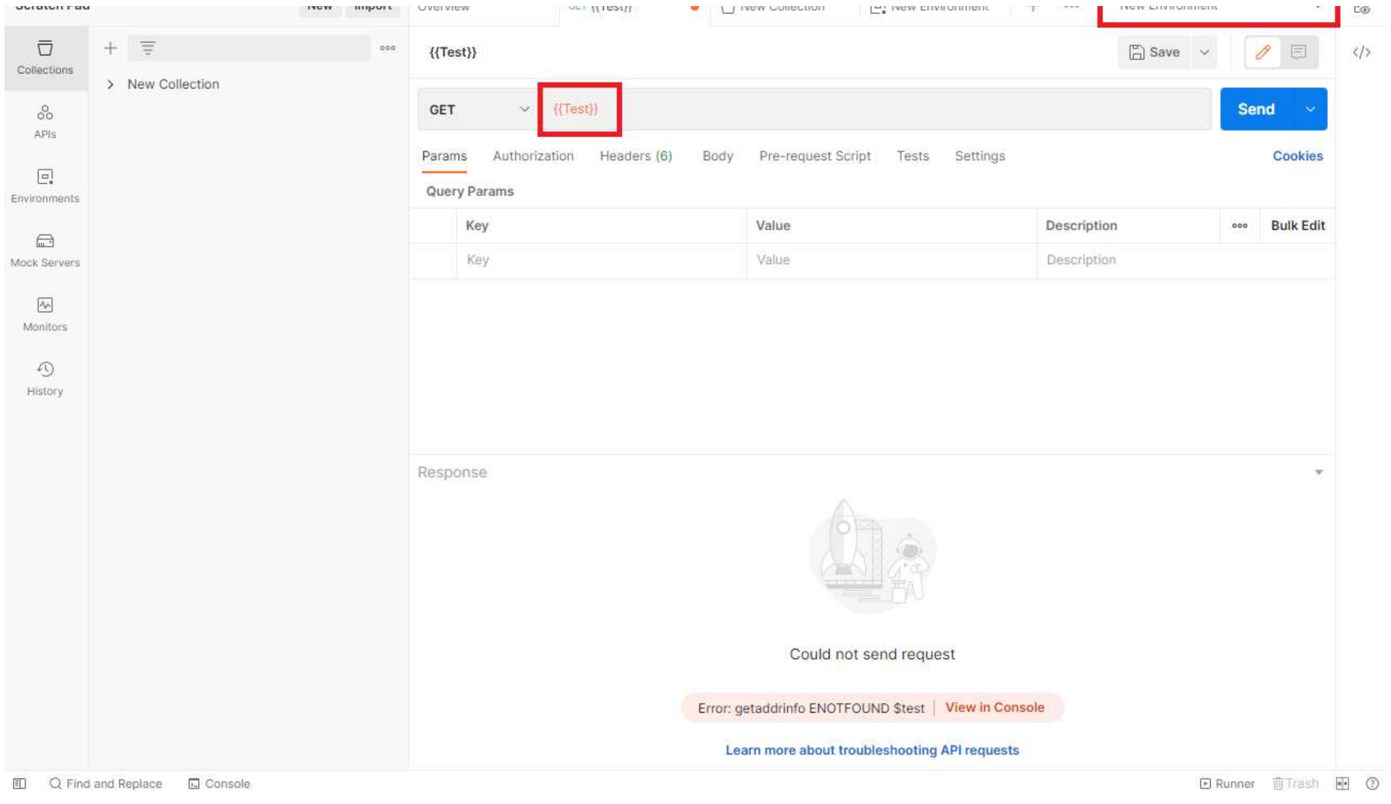
Here you can now create variables by giving them a name and entering their value. Once you have created your variables, click the Save button to persist your environment so that it can be used.



	Variable	Type	Initial value	Current value
✓	Test	default	Test	Test
	Add new variable			

Use variables to reuse values and protect sensitive data
Store sensitive data in variable type secret to keep its values masked on the screen. Learn more about variable type
Work with the current value of a variable to prevent sharing sensitive values with your team. Learn more about variable values

To use a variable, for example, in a request's URL, you must first select the environment that you created in the top right corner. Then you can enter the name of the variable into the request's URL field surrounded by double curly-braces as shown in the image below.



Could not send request

Error: getaddrinfo ENOTFOUND $test | View in Console

Learn more about troubleshooting API requests

References

[1]: https://gorm.io/docs/ GORM

[2]: https://learning.postman.com/docs/introduction/overview/ Postman