Time	1 hour
Learning Goals	• Learn to build a basic Sinatra server to serve HTTP requests • Learn how to limit access to an exposed API endpoint

In the previous lesson we covered the format of HTTP requests and the important parameters programmers need to specify when making them.

First, lets take a look at the first request we will be making.

```
require 'rest-client'


RestClient.get "localhost:4567/this_is_the_path"
```

After your experiences building websites with rails/sinatra, you are probably thinking

- I have to setup a server at localhost:4567
- I need to create a route to /this_is_the_path.

And you would be correct! However, the minimal code you need to do that is a lot less than you think.

Try creating a `server.rb` file with the following code:

```
require 'sinatra'


get '/this_is_the_path' do
 'Hello world!'
end
```

Now try running the server by running `ruby server.rb`, and execute the request code in a separate terminal. You should see the following:

Server:

```
== Sinatra (v1.4.8) has taken the stage on 4567 for development with backup from Puma
```

```
Puma starting in single mode...
* Version 3.7.0 (ruby 2.3.3-p222), codename: Snowy Sagebrush
* Min threads: 0, max threads: 16
* Environment: development
* Listening on tcp://localhost:4567
Use Ctrl-C to stop
127.0.0.1 - - [25/Apr/2017:18:31:02 +0800] "GET /this_is_the_path HTTP/1.1" 200 12 0.0065
```

Client:

```
irb(main):003:0* RestClient.get "localhost:4567/this_is_the_path"
=> "Hello world!"
```

Congratulations! You have made your first API endpoint and made a request to it!

More Complex API Interactions

Now that we have a basic endpoint up and running, lets try to implement a scenario you would want to handle in real life. In the process, we will learn about handling the Header and the Body from the request on the server side.

Preventing Unauthorized Access

When you're running a Sinatra server on your computer it is `localhost` only by default. That means only your computer can try making HTTP requests to `localhost:4567`.

However, if you deploy your app to Heroku, suddenly anyone can try making HTTP requests to it. That means hackers, bots, and any other interested parties can start bombarding your app with test requests to see if you have any exposed endpoints that they can make use of.

You still want authorized users to be able to use the server though. So how would you limit access to the server?

The most common way to do authorization is via passwords, which is basically a string that only the client and server should know. The flow usually looks something like this:

1. client sends password to server
2. server checks if password matches
3. if password matches, server now knows who is the `current_user` and if they are authorized
4. if password does not match anyone in the DB, server rejects the request and sends a response with `401 Forbidden`

So how would we implement this with HTTP requests/responses and Sinatra?

Remember the important parts of a HTTP request?

1. Method
2. Path
3. Headers
4. Body

We can't do much with Method and Path, so should we put the password in Headers or Body?

The answer is: Its up to the server creator! If the server creator wants to check headers, the requester has to use Headers, if the server creator wants to check body instead, you do that! Or they might even check both! Like this

Here is how you would check it from the headers

```
# server.rb
post "/this_path_checks_authorization" do
 p request.env["HTTP_PASSWORD"]
 password = request.env["HTTP_PASSWORD"]
 if password == "12345678"
   'Success'
 else
   'Failure'
 end
end
```

```
# client side
RestClient.post "localhost:4567/this_path_checks_authorization", "", {password: "12345678"}
```

And here is how you would check it from the body

```
# server.rb
post "/this_path_checks_authorization" do
  p params["password"]
  password = params["password"]
  if password == "12345678"
    'Success'
  else
    'Failure'
  end
end

# client side
RestClient.post "localhost:4567/this_path_checks_authorization", {password: "12345678"}
```