Chapter 18

RoR: Mi primera API

En este capitulo veremos con implementar un API en Ruby on Rails. API de sus siglas en ingles significa interfaz de programación de aplicaciones. Se trata de un conjunto de mecanismos que permiten la comunicación de dos aplicaciones que tipicamente estan en dos computadores diferentes. En programación web, normalmente se comunican dos aplicaciones: el navegador web (Chrome, Safari, Firefox, etc) y un servidor en la Nube. Podemos decir en palabras sencillas que el servidor es un programa ejecutandose en la nube que esta a la espera que los clientes (otras aplicaciones) se comuniquen con el y le hagan solicitudes para recuperar o guardar información.

En el curso implementaremos un servidor en Ruby on Rails. Para comunicarse con el se mandarán información en formato JSON mediante el protocolo HTTP. Por lo que este capitulo, primero describe el protocolo HTTP, luego da una pequeña introducción al formato JSON, finalmente, implementa una pequeño servidor que ofrece una interfaz para crear, recuperar y borrar estudiantes de base de datos.

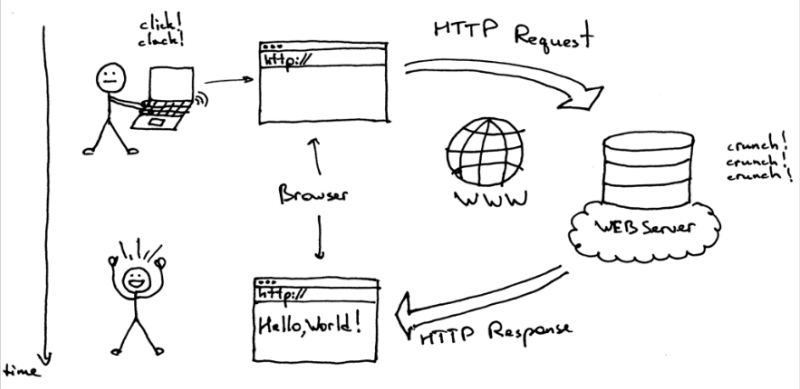
18.1 Protocolo de Transferencia de HiperTexto

HTTP significa Protocolo de Transferencia de HiperTexto, este protocolo es considerado una de las bases de la *web* a nivel mundial. Para entender HTTP, primero debemos enteder que es un protocolo. Un protocolo es un conjunto de reglas, en particular, los protocolos de red son estándares y politicas formales que definen politicas de intercambio de información entre dos computadores.

Supongamos que enviamos el texto siguiente información desde un computador a otro:

```
01101000011011110110110001100001
```

El computador que recibe el mensaje tiene problemas al interpretar la información ya que solo recibe un conjunto de bits. Este conjunto de bits podria un numero, una cadena, dos numeros, un password, existen un monton de posibilidades. Por este motivo para que el computador que recibe el mensaje pueda saber de que trata es necesario también enviar meta-data (datos sobre los datos). Sin embargo, estos meta-datos también tendrian el mismo problema ya que solo estariamos enviando una cade de bits mas grande. Por esta razón, es necesario definir protocolos que regulen como estructurar los datos y meta-datos que se envian entre dos computadores o procesos. En particular, el protocolo HTTTP propone separar el envio de información en dos: HTTP Request y HTTP Response.



HTTP Request.. La cual establece que si alguien requiere solicitar información debe enviar los siguientes datos. La primera linea debe contener 3 elementos:

- *HTTP method*: que en la practica es un verbo o nombre (en formato string) que describe la acción que se va ejecutar. Los mas utilizados son "GET" que indica que se el que envia la solicitud esta esperando una respuesta y "POST" que indica que el que envia la información esta enviando datos que espera que se "empujen" en el servidor.
- *Request target*: que usualmente es una URL donde se especifica cual es el objetivo a ejecutar.
- *HTTP version*. Donde indica la version del protocolo HTTP que esta utilizando para enviar la petición. Las versiones de protocolo HTTP tiene ligeras variaciones por lo cual es importante informar cual se esta utilizando. Normalmente, se utiliza la version 1.1.

En las siguientes lineas se envian todos los "headers" necesarios. HTTP tiene una lista de headers posibles a enviar, uno podria ver un header como una variable que fue inicializada con un valor por defecto. Si uno quieren utilizar otro valor distinto al por defecto, este se debe especificar al enviar el HTTP Request. Los headers que utilizaremos durante el curso casi siempre seran los

mismos, por lo que no es necesario preocuparse mucho sobre cuantos tipos de headers existen, entre otras cosas.

Finalmente, el *body* (cuerpo) del mensaje va al final, debe existir un enter dentre los headers y el body.

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

HTTP Response. Del mismo modo la respuesta debe tener una estructura parecida.

- Una linea donde se especifica el estado de la solictud. Existe un conjunto de estados predefinidos en HTTP entre los mas comunes es el estado 200 OK, que significa que la solicitud fue ejecutada correctamente;
- la respuesta tambien contienen varios headers;
- una linea en blanco;
- finalmente, en ocacioens tambien cuenta con un body.

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

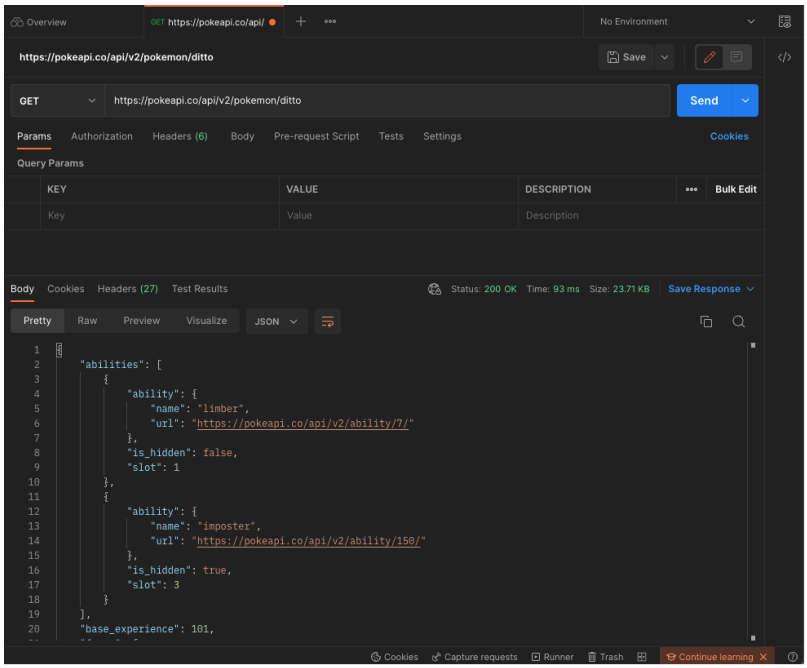
El mensaje anterior indica que el servidor realizo la petición de forma correcta.

18.2 Postman

En la actualizada existen varias API que utilizan el protoco HTTP, por ejemplo, pokeapi. Poke API es una base de datos que tiene información de pokemon. Existen varias herramientas o programas que permite enviar HTTP Requests y recibir HTTP Responses. En el curso utilizaremos Postman, un programa que proporciona una UI para armar una solicitud HTTP Request, enviarla y recibir respuestas HTTP. En esta sección utilizaremos Postman para solicitar información de pokemones a PokeAPI.

Postman es simple de utilizar, se debe utilizar las opciones del menu y buscar

la opción de “crear HTTP Request”. Existen varias formas, una de ellas es ir a “File/New…” y luegro presionar “HTTP Request”. Posteriormente, se debe llenar los datos de la HTTP Request en la UI proporcionada en postman. En nuestro ejemplo, solo necesitamos especificar que la operación es de tipo “GET”, poner la dirección URL la cual esta compuesta por el dominio de la applicación y el http target que en este caso es “pokemon/ditto”. No se agrego ningun header.



Finalmente, se debe presionar el boton “Send” (en azul), en la parte inferior de postman se puede ver la respuesta (reponse) HTTP. Sepuede ver que el estado de la respuesta es 200 OK. En la parte central se ve el “body” de la respuesta que es una cadena que contiene datos en formato JSON:


```
{
    "abilities": [
        {
            "ability": {
                "name": "limber",
                "url": "https://pokeapi.co/api/v2/ability/7/"
            },
            "is_hidden": false,
            "slot": 1
        },
        {
            "ability": {
                "name": "imposter",
```

```
                "url": "https://pokeapi.co/api/v2/ability/150/"
            },
            "is_hidden": true,
            "slot": 3
        }
    ],
    "base_experience": 101,
    "forms": [
        {
            "name": "ditto",
            "url": "https://pokeapi.co/api/v2/pokemon-form/132/"
        }
    ],
    ...
}
```

En esta sección se vio como enviar HTTP-Request y recibir HTTP-Response al API de pokemones. En este ejemplo, se vio solo consumir uno de los servicios de pokeAPI, pokeAPI tiene varios servicios que permiten: listar los pokemones existentes, obtener detalles de un pokemon en particular, entre otros servicios que se pueden acceder a través de la web (web services).

18.3 Hello World API

En la sección anterior vimos como consumir un servicio web, en esta sección aprenderemos a crear un servicio web sencillo. Como primer ejemplo básico crearemos un servicio web en Ruby on Rails que devuelve "Hola Mundo" cada vez que un cliente haga una petición. Antes de empezar a realizar las instrucciones de este tutorial usted debe tener instalado Ruby y Ruby on Rails. Para verificar su instalación ejecute el siguiente comando en consola

```
>ruby --version
ruby 3.1.2p20
>rails --version
Rails 7.0.4.2
```

Si usted esta trabajando con versiones de rails o ruby superiores no deberia tener ningún problema.

Paso 1. Crear un proyecto ruby en modo API:

```
rails new hello --api
```

La linea anterior debe crear un folder llamado "hello" y dentro de el varias carpetas y archivos. A medida que avanzemos en el curso se ira detallando para que sirven cada uno de los archivos y folders generados. Una vez creado el proyecto debe ingresar dentro de la carpeta recién creada.

```
cd hello
```

Ya dentro de la caperta se debe crear un controlador, en este caso nombraremos a nuestro controllador "Hello". Un controlador en Rails es una clase cuyos objetos recibirán solicitudes (request) a través de sus métodos. Para crear el controlador debemos ejecutar el siguiente comando:

```
rails generate controller Hello
```

El comando anterior debe generarnos la siguiente clase en el folder "app/controllers/hello_controller.rb".

```
class HelloController < ApplicationController

end
```

Inicialmente el controlador no tiene ningún comportamiento asociado (ningún método), nosotros implementaremos un método llamado "sayHi" que devolverá la cadea 'Hello World' al que realize una solicitud a través de este método.

```
class HelloController < ApplicationController
   def sayHi
        render json:'Hello World!!'
  end
end
```

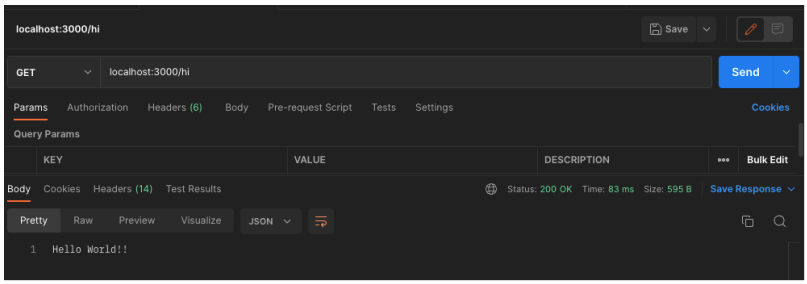
Las solicitudes en RubyOnRails se realizán mediante el protocolo HTTP. Las solicitudes HTTP viene compañadas de "target" en formato "string". El target indica que operación queremos ejecutar en el servidor. En nuestro ejemplo queremos que nuestra operación se pueda ejecutar utilizando el siguiente target: 'http://localhost:3000/hi'. Para lo cual es necesario indicar a ruby que el target '/hi' ejecutara el método "sayHi" de nuestro controllador llamado 'Hello' para mapear el target con el metodo a ejecutar se debe editar el archivo "config/routes.rb" como se ve a continuación:

```
Rails.application.routes.draw do
  get 'hi', to:'hello#sayHi'
end
```

Con lo anterior tenemos listo nuestro primer API, para probar su funcionamiento debemos ejecutar el siguiente comando en consola:

```
rails server
```

Para probar el API utilizaremos Postman, igual que en el ejemplo anterior solo necesario especificar el target y enviar la solicitud con el boton "Send". Como se puede ver en la parte inferior el servidor respondio con un "Hello World!"



18.4 API – Estudiantes

En esta sección crearemos un API un poco mas complejo. La idea es tener un servidor que permita listar estudiantes, y buscar un estudiante mediante un ID.

18.4.1 Creando el Modelo.

Primero, crearemos un clase que nos permita modelar (Model) los datos y comporamentiento de un estudiante (en particular los datos). Para crear esta clase utilizaremos un comando en rails.

```
rails generate model Student name score
```

Si todo funciono correctamente Rails creara los siguientes archivos:

```
invoke  active_record
create    db/migrate/20230312193956_create_students.rb
create    app/models/student.rb
invoke    test_unit
create      test/models/student_test.rb
create      test/fixtures/students.yml
```

La ventaja de crear la clase Student utilizando el comando anterior es que Rails ademas de crear la clase Student, generara archivos que me ayudaran a guardar estudiantes en la base de datos y hacer pruebas de unidad automaticas. Por ejemplo, en este momento ya es posible crear la base de datos para almacenar estudiantes utilizando el siguiente comando:

```
rails db:migrate
```

Si el comando funciono correctamente el siguiente mensaje deberia mostrarse:

```
== 20230312193956 CreateStudents: migrating ===================================
-- create_table(:students)
   -> 0.0007s
== 20230312193956 CreateStudents: migrated (0.0007s) ==========================
```

Note que la clase creada por Rails hereda de la clase ApplicationRecord

```
class Student < ApplicationRecord
end
```

Gracias a que hereda de esta clase (entre otras cosas) ya es posible crear objetos estudiantes, guardarlos y recuperarlos de la base de datos. Por ejemplo, las siguientes expresiones permitiran crear un objeto estudiante y guardarlo en la base de datos.

```
student = Student(name:"Juan Pablo", score:100)
result = student.save
```

En el ejemplo anterior, la variable result sera falso si es que Rails no pudo guardar al estudiante, por ejemplo, porque tal vez ya existe otro estudiante con el mismo ID en la base de datos o porque al estudiante le faltan algunos campos requeridos. A continuación se muestra otros métodos utilez que tiene la clase ApplicationRecord para (a) recuperar a un estudiante dado un ID, o para recuperar todos los estudiantes de la base de datos en una lista.

```
student = Student.find(1)
all_students = Student.all
```

La ultima instrucción devuelve una lista de objetos como se ve a continuación:

```
[#<Student:0x000000010a1c7478
  id: 1,
  name: "Juan Pablo",
  score: "100",
  created_at: Sun, 12 Mar 2023 19:53:07.784504000 UTC +00:00,
  updated_at: Sun, 12 Mar 2023 19:53:07.784504000 UTC +00:00>]
```

Note que cada estudiante dentro de la base de datos, aparte de name y score, tiene un ID, una fecha de creación y una fecha de actualización.

18.4.2 Creando un Controlador

Ahora procederemos a crear un controlador que tenga métodos (operaciones) que permitan manipular a los estudiantes dentro de la base de datos. En este ejemplo, haremos dos operaciones: (a) buscar estudiante por ID, y (b) listar todos los estudiantes. Para crear el controlador, al igual que el ejemplo anterior, tenemos que ejecutar la siguiente instrucción:

```
rails generate controller Student
```

18.4.3 Listando Estudiantes

Posteriormente, agregaremos una operación al controllador llamada “index” que sera encargada de listar a todos los estudiantes:

```
class StudentController < ApplicationController
    def index
        @all_students = Student.all
        render json:@all_students
    end
end
```

Finalmente, mapeamos el "target" al método del controlador respectivo.

```
Rails.application.routes.draw do
  get 'index', to:'student#index'
end
```

En el código anterior estamos mapeando la ruta "localhost:3000/index" al controllador 'student' (StudentController) y al método 'index'. Para probar recuerde ejecutar:

```
rails server
```

Si accedemos al servicio web utilizando postman, la salida deberia ser la siguiente:

```
[
    {
        "id": 1,
        "name": "Juan Pablo",
        "score": "100",
        "created_at": "2023-03-12T19:53:07.784Z",
        "updated_at": "2023-03-12T19:53:07.784Z"
    }
]
```

18.4.4 Mostrando un estudiante

En este ejemplo, veremos como se puede enviar la información al servidor a traves de la URL. Por ejemplo, queremos que cuando el cliente escriba "localhost:3000/estudiante/1", muestre la información del estudiante con el ID igual a 1.

Primero debemos agregar la operación en el controlador:

```
class StudentController < ApplicationController
    # GET /student/:id
    def show
        @student = Student.find(params[:id])
        render json: @student
    end
end
```

Posteriormente, mapeamos la ruta a la operación:

```
Rails.application.routes.draw do
  get '/index', to:'student#index'
  get '/student/:id', to:'student#show'
end
```

Note que a diferencia del anterior ejemplo, esta vez mandamos un argumento a la operación como parte de la URL. En la ruta el argumento se define como ":id", y para recuperar este argumento dentro del controlador utilizamos la siguiente expresion "params[:id]".

18.4.5 Creando un estudiante

Para enviar información mas compleja a traves de un HTTP Request utilizaremos el *body* y el formato *JSON*. Por ejemplo, podemos enviar la información de un nuevo estudiante en formato JSON de la siguiente forma:

```
{
  "student": {
    "name": "Pedro",
    "score": 51
  }
}
```

Note que en el caso de crear un estudiante, nosotros no queremos recuperar información (GET) sino introducir información en el servidor. Por lo que en este caso el HTTP Request deberia ser de tipo POST. A continuación podrá encontrar la operación que permite registrar un nuevo estudiante.

```
class StudentController < ApplicationController
    # POST /student
    def create
        @student = Student.new(student_params)
        if @student.save
          render json: @student
        else
          render json: @student.errors, status: :unprocessable_entity
        end
    end

    def student_params
        params.require(:student).permit(:name, :score)
    end
end
```

El método student_params permite recuperar la información del body del request y construir un hash con los datos del estudiante. Este hash es enviado al momento de crear al estudiante. En caso que el estudiante logre guardarse en la

base de datos el método save devolvera “verdad”, “falso” en caso contrarío. En caso que no sea posible guardar al estudiante devolveremos un HTTP Response, con status “unprocessable_entity” y enviaremos los errors que guarder Rails en la variable “errors”.

El ultimo paso es mapear la ruta del URL con la operación:

```
Rails.application.routes.draw do
  get '/index', to:'student#index'
  get '/student/:id', to:'student#show'
  post '/student', to:'student#create'
end
```

Usted ya puede probar esta operación utilizando Postman, no olvide enviar el request utilizando el método POST y copiando el JSON del estudiante nuevo en el body. Si todo sale correctamente el HTTP response debería tener el siguiente body:

```
{
    "id": 2,
    "name": "Pedro",
    "score": "51",
    "created_at": "2023-03-12T23:39:13.687Z",
    "updated_at": "2023-03-12T23:39:13.687Z"
}
```

18.5 Ejercicios

Agregue la operación filter al controlador Student, el método filter debe recibir un numero como argumento, y retornar en formato JSON los datos de los estudiantes que tienen una nota igual al argumento del request (params[:score]). Crea la request en Postman para demostrar que el método filter funciona correctamente.