Create the web service application

Let's build the application and run it on your development machine. We will adjust the Arduino sketch to connect to this instance of the application while we test it. Once we are satisfied that everything is working well, we'll deploy to the cloud and update the sketch to use the cloud instance.

Here is the Ruby code, all in a single file named web.rb (this code is available on Github).

```
require 'sinatra'
```

Use the Sinatra DSL.

```
configure do
    require 'redis'
    configure(:production) do
        uri = URI.parse(ENV["REDISCLOUD_URL"])
        $redis = Redis.new(  :host => uri.host,
                             :port => uri.port,
                    :password => uri.password)
    end
    configure(:development){ $redis = Redis.new }
    set :server, :puma
end
```

The Sinatra configuration block. First, imports the **redis** gem so that your server can store and retrieve data. When the server runs on Heroku, it is set to run in production mode. In this case, it will make use of the Redis server offered on the Heroku platform (more about this later).
If your server is running on the development machine, use the Redis default settings.
For both environments, the application web server is **Puma**.

```
get '/' do
    erb :index
end
```

Define the root ("**/**") route. The code between "do" and "end" will execute. "**get**" refers to the HTTP verb "**GET**".

Run the code in **index.erb**, inside the **views** directory.

```
post '/post_message' do
    require "json"
    $redis.set( params[:element_1],
       { "message" => params[:element_2],
         "buzzer" => params[:element_3_1]}.to_json)
    "Thank you, message posted."
end
```

Define the **POST** route for **/post_message**. The code that follows take the data from a web form and stores it in Redis.

Data is stored in **JSON** format, so require it in this block.

This string is returned to the browser and shown to the user.

Form parameters are stored in the **params** array. With **$redis.set** we set a key-value pair with key being the value of **params[:element_1]**, value being a hash made of (1) key "**message**" and the value of **param[:element_2]** and (2) key "**buzzer**" and the value of **params[:element_3_1]**. In Redis, objects must be serialized, so we do this by calling the **to_json** method. This converts the hash into a JSON-formatted string.

```
get '/get_message/:dmd_id' do
        require "json"
        response = JSON.parse(
              $redis.get(params[:dmd_id])
                                )
        if response["buzzer"]
               "1" + response["message"] + "\n"
        else
               "0" + response["message"] + "\n"
        end
end
```

Define the GET request route for "**/get_message**". This route accepts an argument, **":dmd_id"**, which is accessible via the **params** array.

Retrieve the value stored against key in **params[:dmd_id]** in Redis. This returns the serialised JSON representation that was created in the **post_message** method. We remake the actual hash object by calling the **JSON.parse** method and applying it to the string that Redis returns.

If the hash return from the Redis store contains the **buzzer** key, then start the response string with a "**1**", followed with the message, otherwise start with a "**0**". The Arduino will evaluate this first character to determine if it should activate the buzzer.