# How do you I create a simple web server with Elixir using Cowboy and Plug?

*Written by Seth Corker on Benevolent Bytes*

You'll often hear talk about Phoenix a lot in the Elixir ecosystem but there are a few other options for web servers. We're going to take a look at what underpins Phoenix and see how to create a simple web server in Elixir. We'll use `plug_cowboy` which combines the Erlang web server Cowboy and a nice adapter Plug.

## What we're making

We're going to make what can barely be called a web server, it's simply going to respond with "Hello World" over HTTP/1.1. It's a humble start but from here, it's easy to add routing, serve static assets, make your own API and more.

If you'd like to see the code right away - take a look at the example repo here.

## Creating a simple server

We'll create a new project, add the only dependency we need - `plug_cowboy`, then do a little bit of setup.

### Create a new project

Let's start with a new app: `mix new simple_plug_rest --sup`, this will create a barebones Elixir project.

Now, install`plug_cowboy` in `mix.exs` like so:

```
def deps do
  [
    {:plug_cowboy, "~> 2.0"}
  ]
end
```

*You can update the dependencies afterwards by running `mix deps.get`.*

### Create a simple Plug

A plug is a specification for our web server, let's make a simple plug based on that simply replies with "Hello World".

```
defmodule SimplePlugRest do
  @moduledoc """
  A Plug that always responds with a string
  """
  import Plug.Conn

  def init(options) do
    options
  end

  @doc """
  Simple route that returns a string
  """
  @spec call(Plug.Conn.t(), any) :: Plug.Conn.t()
  def call(conn, _opts) do
    conn
    |> put_resp_content_type("text/plain")
    |> send_resp(200, "Hello World")
  end
end
```

### Setup Application:

The final piece of the puzzle is to start the Cowboy server as a supervised process and tell `plug_cowboy` which plug to use.

```elixir
defmodule SimplePlugRest.Application do
  @moduledoc false

  use Application

  @impl true
  def start(_type, _args) do
    children = [
      # Start `SimplePlugRest` and listen on port 3000
      {Plug.Cowboy, scheme: :http, plug: SimplePlugRest, options: [port: 3000]}
    ]

    # See https://hexdocs.pm/elixir/Supervisor.html
    # for other strategies and supported options
    opts = [strategy: :one_for_one, name: SimplePlugRest.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

Start your web server up by running `mix run --no-halt` then visit `localhost:3000` in your browser! It's not pretty, but it works.

## What about JSON?

One of the most common response types nowadays is JSON. If you'd like, you can leverage the jason package to encode a map in Elixir.

To get started, add `{:jason, "~> 1.1"}` to your `mix.exs` deps.

Then use `Jason.encode!` to encode the data. If you were to build upon this, a simple helper function could do make creating a json payload easier.

In the meantime, the `call` method will look something like this:

```elixir
def call(conn, _opts) do
  body = Jason.encode!(%{hello: "world"})

  conn
  |> put_resp_content_type("application/json")
  |> send_resp(200, body)
end
```

That's it. You now have the building blocks for creating a web server in Elixir with some easy to use libraries.

## Resources

- Plug documentation
- Jason documentation - for JSON encoding and decoding