

IF140303-Web Application Development

Session-06: Structs and Avatar Generator

PRU/SPMI/FR-BM-18/0222

Alfa Yohannis



Introduction to Structs in Elixir

- Structs in Elixir are special maps with a defined set of keys and default values.
- Unlike regular maps, structs enforce the presence of specific keys, providing more structure and clarity in your code.
- Example: Creating a struct in the `Avatar . Image` module.

Defining a Struct

```
1  defmodule Avatar.Image do
2    defstruct hash: nil, color: nil, grid: nil, pixel_map: nil
3  end
```

- The `Avatar.Image` struct has predefined keys: `hash`, `color`, `grid`, and `pixel_map`.
- This ensures that only these keys can be used, enhancing data consistency.

Using Structs in IEx

```
1 iex> %Avatar.Image{}  
2 %Avatar.Image{color: nil, grid: nil, hash: nil, pixel_map:  
  nil}
```

- When initialized, the struct fields are set to their default values.
- Attempting to add keys not defined in the struct will result in an error, ensuring only the expected fields are used.

Overview of the Avatar Generator

- The AvatarGenerator module generates a unique avatar based on an input string.
- This lesson will walk through the key steps in the avatar generation process, focusing on the code after hashing.

Main Function: Generating the Avatar

```
1  def generate(input) do
2    input
3    |> compute_hash
4    |> select_color
5    |> create_grid
6    |> remove_odd_cells
7    |> generate_pixel_map
8    |> render_image
9    |> store_image(input)
10  end
```

- The generate/1 function is the main entry point, taking an input string and transforming it into an avatar.
- The process is broken down into distinct steps, chained together using the pipe operator (|>).

Step 1: Selecting a Color

```
1   def select_color(%Avatar.Image{hash: [r, g, b | _tail]} =  
    image) do  
2   %Avatar.Image{image | color: {r, g, b}}  
3   end
```

- Extracts the first three elements from the hash to form an RGB color.
- Updates the `Avatar.Image` struct with this color.

Step 2: Creating a Grid

```
1  def create_grid(%Avatar.Image{hash: hash} = image) do
2    grid =
3      hash
4      |> Enum.chunk_every(3)
5      |> Enum.map(&reflect_row/1)
6      |> List.flatten
7      |> Enum.with_index
8
9    %Avatar.Image{image | grid: grid}
10   end
```

- Converts the hash into a grid format by chunking the list and reflecting rows to ensure symmetry.
- The grid is then flattened and indexed, creating a structured layout for the avatar.

Step 3: Removing Odd Cells

```
1 def remove_odd_cells(%Avatar.Image{grid: grid} = image) do
2   grid = Enum.filter grid, fn({code, _index}) ->
3     rem(code, 2) == 0
4   end
5
6   %Avatar.Image{image | grid: grid}
7 end
```

- Filters out cells in the grid where the code is odd, keeping only even values.
- This step simplifies the grid, focusing on symmetrical, even-numbered cells for the final avatar.

```
1  def generate_pixel_map(%Avatar.Image{grid: grid} = image) do
2  pixel_map = Enum.map grid, fn({_code, index}) ->
3  x = rem(index, 5) * 50
4  y = div(index, 5) * 50
5
6  top_left = {x, y}
7  bottom_right = {x + 50, y + 50}
8
9  {top_left, bottom_right}
10 end
11
12 %Avatar.Image{image | pixel_map: pixel_map}
13 end
```

- Translates the grid into a pixel map by calculating the coordinates for each cell.
- Each cell is mapped to a rectangular region on the avatar image, forming the final visual representation.

Step 5: Rendering the Image

```
1  def render_image(%Avatar.Image{color: color, pixel_map:  
    pixel_map}) do  
2  image = :egd.create(250, 250)  
3  fill = :egd.color(color)  
4  
5  Enum.each pixel_map, fn({start, stop}) ->  
6  :egd.filledRectangle(image, start, stop, fill)  
7  end  
8  
9  :egd.render(image)  
10 end
```

- Creates a new image canvas and fills the pixel map regions with the selected color.
- The image is then rendered, producing the visual output that represents the avatar.

Step 6: Storing the Image

```
1  def store_image(image, input) do
2    File.write("#{input}_avatar.png", image)
3  end
```

- The final image is saved as a PNG file with a name based on the input string.
- This concludes the avatar generation process, producing a unique, personalized avatar for each input.