PRADITA University

IF140303-Web Application Development

Session-06:
Avatar Generator (part 2 of 2)

PRU/SPMI/FR-BM-18/0222

Alfa Yohannis

Introduction to Structs in Elixir

PRADITA University

- 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

PRADITA University

```
defmodule Avatar.Image do
defstruct hash: nil, color: nil, grid: nil, pixel_map: nil
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

PRADITA University

```
iex> %Avatar.Image{}
%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

PRADITA University

- 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

```elixir
def generate(input) do
input
|> compute_hash
|> select_color
|> create_grid
|> remove_odd_cells
|> generate_pixel_map
|> render_image
|> store_image(input)
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

PRADITA University

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

PRADITA University

```elixir
def create_grid(%Avatar.Image{hash: hash} = image) do
grid =
hash
|> Enum.chunk_every(3)
|> Enum.map(&reflect_row/1)
|> List.flatten
|> Enum.with_index

%Avatar.Image{image | grid: grid}
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

PRADITA University

```elixir
def remove_odd_cells(%Avatar.Image{grid: grid} = image) do
grid = Enum.filter grid, fn({code, _index}) ->
rem(code, 2) == 0
end

%Avatar.Image{image | grid: grid}
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.

Step 4 - Generating the Pixel Map

PRADITA

```elixir
def generate_pixel_map(%Avatar.Image{grid: grid} = image) do
pixel_map = Enum.map grid, fn({_code, index}) ->
x = rem(index, 5) * 50
y = div(index, 5) * 50

top_left = {x, y}
bottom_right = {x + 50, y + 50}

{top_left, bottom_right}
end

%Avatar.Image{image | pixel_map: pixel_map}
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

PRADITA University

```
def render_image(%Avatar.Image{color: color, pixel_map:
    pixel_map}) do
image = :egd.create(250, 250)
fill = :egd.color(color)

Enum.each pixel_map, fn({start, stop}) ->
:egd.filledRectangle(image, start, stop, fill)
end

:egd.render(image)
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

PRADITA University

```
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.