PRADITA University

IF140303-Web Application Development

Session-09: Creating Forms and Routing in Phoenix

PRU/SPMI/FR-BM-18/0222

Alfa Yohannis

Introduction to Phoenix Models

- In Phoenix, models represent data and are linked to a PostgreSQL database.
- Models validate data before it is saved to the database.
- To create a model, use the `use Discuss.Web, :model` macro, which includes functions necessary for model creation.

Model Schema and Changeset Implementation

- To build a functional model, two main components are essential:
  1. Schema: Defines how the model relates to the database.
  2. Changeset: Manages data validation and transformations.
- These components ensure that the model accurately represents and validates the data.

Example: Defining a Schema

```elixir
defmodule Discuss.Topic do
use Discuss.Web, :model

schema "topics" do
field :title, :string
end
end
```

- `Discuss.Topic`: Defines the model module.
- `use Discuss.Web, :model`: Adds necessary functionalities to the model.
- `schema "topics"`: Maps the model to the ”topics” table in the database.
- `field :title, :string`: Defines a string field named `title`.

Introducing Changeset for Validation

- A changeset is used to validate and transform data before saving it.
- For instance, in our example, users must enter a `title` for the discussion topic.
- The changeset ensures that the `title` field is present and meets validation rules.

Validation in OOP vs Elixir

- In OOP:
  1. Create a class and instantiate an object.
  2. Update the object’s values.
  3. Validate the object’s state.
  4. If valid, save to the database.
- In Elixir:
  1. No classes; data is passed through multiple functions.
  2. Values are transformed and validated functionally.
  3. Default values are set using
     in function parameters.

Example: Implementing Changeset in Elixir

```
defmodule Discuss.Topic do
use Discuss.Web, :model

schema "topics" do
field :title, :string
end
```

Example: Implementing Changeset in Elixir

```elixir
def changeset(struct, params \\ %{}) do
struct
|> cast(params, [:title])
|> validate_required([:title])
end
end
```

- changeset/2:
  1. Takes a struct and parameters.
  2. Uses `cast` to extract the relevant fields.
  3. Validates that the `title` field is present using `validate_required`.

Summary

PRADITA University

- Phoenix models are integral for linking to databases and validating data.
- The model schema defines the structure of the model and its database relations.
- Changesets provide a functional way to validate and transform data.

IEx and Understanding Changeset

- `Discuss.Web, :model` automatically creates our struct.
- `iex>` allows interactive exploration of changesets.

Exploring Changesets with IEx

To simulate how the program works, we will execute some iex command

```
iex> struct = %Discuss.Topic{}
iex> params = %{title: "Great JS"}
iex> Discuss.Topic.changeset(struct, params)
%Discuss.Topic{
  __meta__: #Ecto.Schema.Metadata<:built, "topics">,
  id: nil,
  title: "Great JS"
}
```

Exploring Changesets with IEx

```
iex> Discuss.Topic.changeset(struct, %{})
{:error, changeset} =
%Discuss.Topic{
  __meta__: #Ecto.Schema.Metadata<:built, "topics">,
  id: nil,
  title: nil
}
```

- Here we create a Discuss.Topic struct and params to input the title
- The first command successfully validates and casts the title.
- The second command fails due to missing required fields.

Linking Model to TopicController

- `alias Discuss.Topic` allows shorthand for referencing the model.
- `changeset = Topic.changeset(%Topic{}, %{})` prepares a blank form for a new topic.

```
defmodule Discuss.TopicController do
use Discuss.Web, :controller

alias Discuss.Topic

def new(conn, _params) do
changeset = Topic.changeset(%Topic{}, %{})
...
```

Creating TopicView and Template

- The `TopicView` helps render the view layer.
- The `new.html.eex` file generates the HTML for the new topic form.

```
defmodule Discuss.TopicView do
use Discuss.Web, :view
end
```

- Now we will write the new.html.eex with:
  `<h1>New Test Form</h1>`
- when we access it `localhost:4000/topics/new` it will give error as render function has not been implemented

Adding Render to TopicController

- The `render/3` function is necessary to display the form.
- The `changeset` is passed to the view for rendering.

```elixir
def new(conn, _params) do
changeset = Topic.changeset(%Topic{}, %{})
render conn, "new.html", changeset: changeset
end
```

Elixir Helpers for HTML Forms

- `form_for/4`: Builds an HTML form using an Ecto Changeset. It takes a changeset, the form action, and a function to define form elements.
- `text_input/4`: Creates a text input field within a form. It requires the form builder, field name, and optional attributes like placeholder and class.
- <%= %>: Inserts Elixir code into an EEx template to generate HTML output.

Using Elixir Helpers in HTML Forms

```
<%= form_for @changeset, topic_path(@conn, :create), fn f ->
    %>
<div class="form-group">
<%= text_input f, :title, placeholder: "Title", class: "form
   -control" %>
</div>
<%= submit "Save Topic", class: "btn btn-primary" %>
<% end %>
```

- Generates a form and inputs, enhancing readability and maintainability.

Updating Router for Post Handling

- **Problem:** Missing post handler for topic creation.
- **Solution:** Update the router.ex file.
- **Router Configuration:**
  - `scope "/", Discuss do`: Defines a scope for routes under the root path.
  - `pipe_through :browser`: Applies the ‘browser‘ pipeline to all routes in the scope.
  - `get "/", PageController, :index`: Maps the root path to the ‘index‘ action of ‘PageController‘.
  - `get "/topics/new", TopicController, :new`: Maps the ‘/topics/new‘ path to the ‘new‘ action of ‘TopicController‘.
  - `post "/topics", TopicController, :create`: Maps the ‘/topics‘ path to the ‘create‘ action of ‘TopicController‘ for handling form submissions.

Updating TopicController with Create Action

- `create/2` handles form submissions and extracts parameters.
- Pattern matching extracts `topic` from the submitted params.

```
    def create(conn, %{"topic" => topic}) do
    end
```

Exploring Phoenix Routes with IEx

- Use `iex> mix phoenix.routes` to list all routes.
- Provides a clear overview of available routes and their corresponding actions.

```
iex> mix phoenix.routes
Helper             Path                         Controller
page_path          GET    /                     PageController :
    index
topic_path         GET    /topics/new           TopicController :
    new
topic_path         POST   /topics               TopicController :
    create
```

Summary

- We explored Phoenix models, schemas, and changesets.
- We linked models to controllers and created views and templates.
- We examined routing, form creation, and handling form submissions.
- Understanding these concepts is crucial for building functional web applications with Phoenix.