

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

```
1  defmodule Discuss.Topic do
2    use Discuss.Web, :model
3
4    schema "topics" do
5      field :title, :string
6    end
7  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

```
1  defmodule Discuss.Topic do
2    use Discuss.Web, :model
3
4    schema "topics" do
5      field :title, :string
6    end
```

Example: Implementing Changeset in Elixir

```
1  def changeset(struct, params \\ %{}) do
2    struct
3    |> cast(params, [:title])
4    |> validate_required([:title])
5  end
6  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

- 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

```
1      iex> struct = %Discuss.Topic{}
2      iex> params = %{title: "Great JS"}
3      iex> Discuss.Topic.changeset(struct, params)
4      %Discuss.Topic{
5          __meta__: #Ecto.Schema.Metadata<:built, "topics">,
6          id: nil,
7          title: "Great JS"
8      }
```

Exploring Changesets with IEx

```
1 iex> Discuss.Topic.changeset(struct, %{})  
2 {:error, changeset} =  
3 %Discuss.Topic{  
4   __meta__: #Ecto.Schema.Metadata<:built, "topics">,  
5   id: nil,  
6   title: nil  
7 }
```

- 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.

```
1  defmodule Discuss.TopicController do
2    use Discuss.Web, :controller
3
4    alias Discuss.Topic
5
6    def new(conn, _params) do
7      changeset = Topic.changeset(%Topic{}, %{})
8      ...
```

Creating TopicView and Template

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

```
1 defmodule Discuss.TopicView do
2   use Discuss.Web, :view
3 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.

```
1  def new(conn, _params) do
2    changeset = Topic.changeset(%Topic{}, %{})
3    render conn, "new.html", changeset: changeset
4  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

```
1      <%= form_for @changeset, topic_path(@conn, :create), fn f ->
2          %>
3      <div class="form-group">
4      <%= text_input f, :title, placeholder: "Title", class: "form
5          -control" %>
6      </div>
7      <%= submit "Save Topic", class: "btn btn-primary" %>
8      <% 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.

```
1  def create(conn, %{"topic" => topic}) do
2  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.

```
1      iex> mix phoenix.routes
2      Helper          Path          Controller
3      page_path      GET    /          PageController :
         index
4      topic_path     GET    /topics/new TopicController :
         new
5      topic_path     POST   /topics   TopicController :
```

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.