PRADITA
University

IF140303-Web Application Development

Session-10:
Saving Data in Phoenix

PRU/SPMI/FR-BM-18/0222

Alfa Yohannis

Saving Data in Phoenix

PRADITA University

- This session covers the process of saving user input to the database using Phoenix and Elixir.
- We will explore the workflow from user input to data persistence.
- We'll also see how to handle errors and enhance the user experience with CSS.

Understanding the Save Workflow

PRADITA University

- The process begins with user input parameters submitted via a form.
- The ‘changeset’ function validates this input.
- If validation succeeds, data is saved to the database.
- The user is then redirected to the topic list with a success message.
- If validation fails, the form is re-rendered with error messages.

Understanding Ecto

PRADITA University

- **Ecto** is a toolkit for data mapping and language integrated queries.
- It provides tools to define schemas, validate data, and interact with databases.
- The ‘changeset’ function in Ecto allows data validation and casting.
- **Repo** is an Ecto module responsible for persisting data to the database.

Updating the Create Function

PRADITA University

```
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       render conn, "new.html", changset: changeset
9       end
```

Updating the Create Function

PRADITA University

```
1       def create(conn, %{"topic" => topic}) do
2       changeset = Topic.changeset(%Topic{}, topic)
3       case Repo.insert(changset) do
4       {:ok, _topic} -> IO.inspect(post)
5       {:error, changeset} ->
6       render conn, "new.html", changeset: changeset
7       end
8       end
9       end
```

Explaining the Create Function

PRADITA University

- **new/2**: Initializes a new changeset for an empty 'Topic'.
- **create/2**: Handles form submission and attempts to save the new topic.
- The 'changeset' function validates the input data and prepares it for saving.
- **Repo.insert/2**: Attempts to insert the validated data into the database.
- If successful, the topic is saved and a success message is shown.
- If it fails, the form is re-rendered with error messages.

Error Handling with error_tag

PRADITA University

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

Understanding error_tag

PRADITA University

- ‘error_tag‘ helps in displaying error messages next to form fields.
- It checks if the ‘changeset‘ contains errors for a specific field.
- If an error exists, it generates an HTML element showing the error message.
- This improves user experience by providing immediate feedback.

Adding CSS to the Project

PRADITA University

- Add the following CSS to ‘web > static > css > app.css‘:

```
.help-block{
  color: red;
  text-transform: capitalize;
  position: absolute;
  top: 37px;
}

.form-group{
  position: relative;
  margin-bottom: 25px;
}
```

Impact of Adding CSS

PRADITA University

- Adding CSS to ‘app.css’ impacts the entire project.
- The ‘.help-block‘ class styles error messages, making them more noticeable.
- The ‘.form-group‘ class provides structure and spacing to form elements.
- These styles enhance the overall user interface and user experience.

Summary

PRADITA University

- We explored the workflow of saving data in Phoenix using Ecto.
- We updated the create function and handled errors with ‘error_tag’.
- Finally, we enhanced the user interface with custom CSS.

Updating the Router

PRADITA University

- The router is updated to change the root path to show a list of topics.
- This change breaks from the RESTful convention by routing "/" to the 'TopicController'.
- Remove unused 'PageController' components.

```
1       scope "/", Discuss do
2       pipe_through :browser
3       get "/", TopicController, :index
4       get "/topics/new", TopicController, :new
5       post "/topics", TopicController, :create
```

Explanation of the Router Update

PRADITA University

- The root path (”/”) now directs users to the ‘TopicController’’s ‘index‘ action.
- The ‘index‘ action will display a list of topics.
- Removed ‘PageController‘ since it’s no longer needed.
- The ‘scope‘ block specifies the routing paths for the Discuss application.

Adding the Index Function to TopicController

PRADITA University

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

alias Discuss.Topic

def index(conn, _params) do
topics = Repo.all(Topic)
render conn, "index.html", topics: topics
end
```

Explanation of the Index Function

PRADITA University

- The ‘index‘ function retrieves all topics from the database using ‘Repo.all/1‘.
- These topics are passed to the ”index.html” template for rendering.
- The ‘topics‘ list is available in the template as ‘@topics‘.

Creating the Index Template

PRADITA University

```
<h5>Topics</h5>

<ul class="collection">
<%= for topic <- @topics do %>
<li class="collection-item">
<%= topic.title %>
</li>
<% end %>
</ul>
```

Explanation of the Index Template

PRADITA University

- The template displays a list of topics.
- The ‘<ul>‘ element represents the unordered list, styled with the ‘collection‘ class.
- The ‘for‘ loop iterates over each topic and renders its title inside a ‘<li>‘ element.

Updating the Create Function

PRADITA University

```
1     def create(conn, %{"topic" => topic}) do
2     changeset = Topic.changeset(%Topic{}, topic)
3     case Repo.insert(changeset) do
4     {:ok, _topic} ->
5     conn
6     |> put_flash(:info, "Topic Created")
7     |> redirect(to: topic_path(conn, :index))
8     {:error, changeset} ->
9     render conn, "new.html", changeset: changeset
10    end
11    end
```

Explanation of the Updated Create Function

PRADITA University

- If ‘Repo.insert/1’ is successful, a success message is flashed.
- The user is then redirected to the ‘index’ action, displaying the list of topics.
- If the insert fails, the form is re-rendered with errors for correction.

Adding a Button to the Index Template

PRADITA University

```
1       <div class="fixed-action-btn">
2       <%= link to: topic_path(@conn, :new), class: "btn-floating
           btn-large waves-effect waves-light red" do %>
3       <i class="material-icons">add</i>
4       <% end %>
5       </div>
```

Explanation of the Button Addition

PRADITA University

- The button is created using the ‘link/2‘ helper, directing users to the ‘new‘ action.
- ‘topic_path(@conn, :new)‘ generates the appropriate URL for the ‘new‘ topic form.
- The button uses Materialize CSS for styling, with a floating effect and an add icon.

Adding Material Icons

PRADITA University

```
1       <link rel="stylesheet" href="https://fonts.googleapis.com/
           icon?family=Material+Icons">
```

Explanation of Material Icons

PRADITA University

- The ‘<link>‘ tag includes Material Icons from Google Fonts into the project.
- This allows the use of icons like the ”add” icon in buttons or other UI elements.

Adding Edit and Update Routes

PRADITA University

```
1       scope "/", Discuss do
2       pipe_through :browser
3
4       get "/", TopicController, :index
5       get "/topics/new", TopicController, :new
6       post "/topics", TopicController, :create
7       get "/topics/:id/edit", TopicController, :edit
8       end
```

Explanation of the Edit Route

PRADITA University

- The ‘edit‘ route allows users to load a form to edit an existing topic.
- The ‘:id‘ parameter in the path corresponds to the topic being edited.
- The ‘edit‘ action will retrieve the topic and display it for editing.

Adding the Edit Function to TopicController

PRADITA University

```
    def edit(conn, %{"id" => topic_id}) do
    topic = Repo.get(Topic, topic_id)
    changeset = Topic.changeset(topic)

    render conn, "edit.html", changeset: changeset, topic: topic
    end
```

Explanation of the Edit Function

PRADITA University

- The ‘edit‘ function retrieves the topic to be edited using ‘Repo.get/2‘.
- A ‘changeset‘ is created for the topic, allowing for validation and form rendering.
- The ‘edit.html‘ template is rendered with the ‘changeset‘ and ‘topic‘ data.

Adding `edit.html.eex` to `templates > topic`

PRADITA University

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

Adding `edit.html.eex` to `templates > topic`

PRADITA University

- `form_for @changeset`: Generates an HTML form bound to the `update` action.
- `topic_path(@conn, :update, @topic)`: The URL for submitting form data to `update`.
- `text_input f, :title`: Creates an input field for `title`, with validation.
- `error_tag f, :title`: Displays error message if `title` is invalid.
- `submit "Save Topic"`: Submit button to save changes.

Updating Router to Include put "/topics/:id"

PRADITA University

```
scope "/", Discuss do
pipe_through :browser
get "/", PageController, :index
get "/topics/new", TopicController, :new
post "/topics", TopicController, :create
get "/topics/:id/edit", TopicController, :edit
put "/topics/:id", TopicController, :update
end
```

Updating Router to Include `put "/topics/:id"`

PRADITA University

- `put "/topics/:id"`: Defines the route for updating an existing topic.
- The route maps to the `update` action in the `TopicController`.
- `:id` is a placeholder for the topic's ID.

Updating `TopicController` with `update/2` Function

PRADITA University

```
def update(conn, %"id" => topic_id, "topic" => topic) do
old_topic = Repo.get(Topic, topic_id)
changeset = Topic.changeset(old_topic, topic)
case Repo.update(changset) do
:ok, _topic ->
conn
|> put_flash(:info, "Topic Updated")
|> redirect(to: topic_path(conn, :index))
```

Updating `TopicController` with `update/2` Function

PRADITA University

- ```
  :error, changeset ->
  render conn, "edit.html", changeset: changeset, topic: old_topic
  end
  end
  ```
- Retrieves the existing topic from the database using `Repo.get`.
- Creates a changeset to validate and apply updates.
- If update succeeds, flashes success message and redirects to `index`.
- If update fails, re-renders the edit page with the errors.
  ```

Linking `index.html.eex` to Edit Pages

PRADITA University

```
 <h5>Topics</h5>
<ul class = "collection">
<%= for topic <- @topics do %>
<li class="collection-item">
<%= topic.title %>
<div class="right">
<%= link "Edit", to: topic_path(@conn, :edit, topic) %>
</div>
</li>
<% end %>
</ul>
```

Linking `index.html.eex` to Edit Pages

PRADITA University

- Lists all topics with an "Edit" link next to each one.
- `link "Edit"` generates a link to the edit page for each topic.
- The link is tied to the specific topic's ID.

Updating Router with Resource Handler

PRADITA University

```
 scope "/", Discuss do
resources "/", TopicController
end
```

- Simplifies routing by using a `resources` macro.
- Automatically generates RESTful routes for `TopicController`.
- Eliminates the need to manually define individual routes.

Adding `delete/2` Function to `TopicController`

PRADITA University

```
 def delete(conn, %"id" => topic_id) do Repo.get!(Topic, topic_id) |>
Repo.delete! conn |> put_flash(:info, "Topic Deleted") |>
redirect(to: topic_path(conn, :index)) end
```

- Retrieves the topic to delete using `Repo.get!`.
- Deletes the topic from the database using `Repo.delete!`.
- Flashes a success message and redirects to the `index` page.

Linking `index.html.eex` to Delete Link

```
<h5>Topics</h5>
<ul class = "collection">
<%= for topic <- @topics do %>
<li class="collection-item">
<%= topic.title %>
<div class="right">
<%= link "Edit", to: topic_path(@conn, :edit, topic) %>
<%= link "Delete", to: topic_path(@conn, :delete , topic), method:
:delete %>
</div> </li> <% end %>
</ul>
```

- Adds a "Delete" link next to each topic.
- The `method: :delete` ensures the correct HTTP method is used.
- The link is tied to the specific topic's ID.