PRADITA University

IF140303-Web Application Development

Session-12:

WebSocket and PubSubs using Phoenix's LiveView

PRU/SPMI/FR-BM-18/0222

Alfa Yohannis

Introduction to Plugs in Elixir

PRADITA University

- A plug is a specification for composing web applications.
- Plugs are the building blocks of Elixir web applications like Phoenix.
- There are two types of plugs:
  - **Function Plug**: A simple plug, defined as a function.
  - **Module Plug**: A more complex plug, defined as a module.

Creating a Module Plug for Authentication

PRADITA University

- We will create a module plug to check if a user is logged in.
- This plug checks if a user ID is assigned in the connection object.
- If the user ID exists, it fetches the user from the database and assigns it to the connection object.
- The plug consists of two main functions:
  - `init/1`: Used for setup, called once.
  - `call/2`: Called with the connection and returns a connection.
- The `assign/3` function is used to store a value in the connection's assigns.

Creating the Plug Module

PRADITA University

```
1     defmodule Discuss.Plugs.SetUser do
2     import Plug.Conn
3     import Phoenix.Controller
4
5     alias Discuss.Repo
6     alias Discuss.User
7     alias Discuss.Router.Helpers
8
9     def init(\_params) do
10    end
```

Creating the Plug Module

PRADITA University

```
    def call(conn, \_params) do
    user\_id = get\_session(conn, :user\_id)
    cond do
    user = user\_id \&\& Repo.get(User, user\_id) ->
    assign(conn, :user, user)
    true ->
    assign(conn, :user, nil)
    end
    end
    end
```

- `init/1`: Placeholder function for initialization.
- `call/2`: Checks session for `user_id`, assigns user if found.
- `assign/3` stores user data in `conn.assigns` for later use.

Using the Plug in the Router

PRADITA University

- The created plug should be added to the router's pipeline.
- Modify `router.ex` to include the plug in the browser pipeline.

```
1       pipeline :browser do
2       plug Discuss.Plugs.SetUser
3       end
```

Adding a Login Button to the Header

PRADITA University

```
<body>
<nav class="light-blue">
<div class="nav-wrapper container">
<a href="/" class="brand-logo">Logo</a>
<ul class="right">
<\%= if @conn.assigns[:user] do \%>
<li>
<\%= link "Logout", to: session_path(@conn, :signout) \%>
</li>
```

Adding a Login Button to the Header

PRADITA University

- We add a login button to the application header.
- The button checks if a user is logged in and displays the appropriate option.
- If the user is logged in, a logout button is shown; otherwise, a login button appears.

Adding a Login Button to the Header

PRADITA University

```
<\% else \%>
<li>
<\%= link "Sign in with Github", to: session_path(@conn, :request, "github") \%>
</li>
<\% end \%>
</ul>
</div>
</nav>
```

- `@conn.assigns[:user]` is checked to see if a user is logged in.
- The `link/2` function creates the login/logout link.

Updating the Router for Signout

PRADITA University

- We add a signout route to the router.
- This allows users to log out of the application.

```
scope "/auth", Discuss do
pipe\_through :browser
get "/signout", AuthController, :signout
get "/:provider", AuthController, :request
get "/:provider/callback", AuthController, :request
end
```

Updating the AuthController for Signout

PRADITA University

- We add a `signout` function to the `AuthController`.
- This function clears the session and redirects the user to the home page.

```
...
def signout(conn, \_params) do
conn
|> configure\_session(drop: true)
|> redirect(to: topic\_path(conn, :index))
end
...
```

Adding a Logout Button to the Header

PRADITA University

- The logout button is dynamically displayed in the header.
- When clicked, it triggers the signout route and logs the user out.

```
<body>
<nav class="light-blue">
<div class="nav-wrapper container">
<a href="/" class="brand-logo">Logo</a>
<ul class="right">
<\%= if @conn.assigns[:user] do \%>
<li>
<\%= link "Logout", to: session_path(@conn, :signout) \%>
</li>
```

Adding a Logout Button to the Header

PRADITA University

```
<\% else \%>
<li>
<\%= link "Sign in with Github", to: session_path(@conn, :request, "github") \%>
</li>
<\% end \%>
</ul>
</div>
</nav>
```

- `link/2` creates a hyperlink to the logout route.
- The button is displayed based on the user's login status.

Adding Authorization: Restricting Actions Based on User Authentication

- We need to ensure only signed-in users can post, edit, or delete topics.
- To enforce this, we'll create a new plug in 'Web > controllers > plug > require_auth.ex'.

Creating the ‘RequireAuth‘ Plug

PRADITA University

```
defmodule Discuss.Plugs.RequireAuth do
use Plug.Conn
use Phoenix.Controller

alias Discuss.Router.Helpers

def init(_params) do
end

def call(conn, _params) do
if conn.assigns[:user] do
conn
```

Creating the ‘RequireAuth‘ Plug

PRADITA University

```
      else
      conn
      |> put_flash(:error, "You must be logged in.")
      |> redirect(to: Helpers.topic_path(conn, :index))
      |> halt()
      end
      end
      end
```

- `halt/0` stops the connection processing immediately.
- This ensures that unauthorized users are redirected before further actions.

Updating ‘TopicController‘ with ‘RequireAuth‘ Plug

PRADITA University

- We will update the ‘TopicController‘ to restrict specific actions to signed-in users.
- The plug is applied conditionally using ‘when‘ with the ‘action‘ keyword.

Updated ‘TopicController‘ with ‘RequireAuth‘ Plug

PRADITA University

```
1  defmodule Discuss.TopicController do
2  use Discuss.Web, :controller
3
4  alias Discuss.Topic
5
6  plug Discuss.Plugs.RequireAuth when action in [:new, :create
       , :edit, :update, :delete]
7  ...
```

- Only the specified actions (new, create, edit, update, delete) will trigger the ‘RequireAuth‘ plug.
- This ensures non-logged-in users cannot access these actions.

Associating Users with Topics

PRADITA University

- We'll associate users with topics using a foreign key relationship.
- This involves adding a 'user_id' column to the 'topics' table.
- Migration will alter the existing table, leaving the 'user_id' empty for previously created topics.

Migration: Adding ‘user_id‘ to Topics

PRADITA University

```
defmodule Discuss.Repo.Migrations.AddUserIdToTopics do
use Ecto.Migration

def change do
alter table(:topics) do
add :user_id, references(:users)
end
end
end
```

- `references(:users)` establishes the foreign key relationship.
- Migrate the database with ‘mix ecto.migrate‘.

Updating Models for Association

PRADITA University

- Now, we need to associate the ‘User‘ model with the ‘Topic‘ model.
- This is done by adding the appropriate associations in both models.

User Model: Adding ‘has_many‘ Association

PRADITA University

```
1   defmodule Discuss.User do
2   use Discuss.Web, :model
3
4   schema "users" do
5   field :email, :string
6   field :provider, :string
7   field :token, :string
8   has_many :topics, Discuss.Topic
9   timestamps()
10  end
```

User Model: Adding ‘has_many‘ Association

PRADITA University

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

- `has_many :topics` sets up the one-to-many relationship.
- A user can have multiple topics.

Topic Model: Adding ‘belongs_to‘ Association

PRADITA University

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

schema "topics" do
field :title, :string
belongs_to :user, Discuss.User
end
```

Topic Model: Adding ‘belongs_to‘ Association

PRADITA University

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

- `belongs_to :user` establishes the reverse relationship.
- Each topic is associated with a specific user.

Updating ‘TopicController‘ to Associate Topics with Users

PRADITA University

- We need to update the ‘create‘ function to associate a new topic with the current user.
- This involves using ‘build_assoc/2‘.

Updating the ‘create‘ Function

PRADITA University

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

Updating the ‘create‘ Function

PRADITA University

```
    {:error, changeset} ->
    render conn, "new.html", changeset: changeset
    end
    end
```

- `build_assoc/2` automatically associates the new topic with the current user.
- This ensures that the ‘user_id‘ field in the ‘topics‘ table is populated.

Restricting Edit/Delete Buttons Based on Own

- We will update the ‘index.html.eex‘ template to ensure only the topic owner can see edit/delete buttons.
- This is done using an ‘if‘ statement.

Updating ‘index.html.eex‘ Template

PRADITA University

```
1       <ul class = "collection">
2       <%= for topic <- @topics do %>
3       <li class="collection-item">
4       <%= topic.title %>
5
6       <%= if @conn.assigns.user.id == topic.user_id do %>
7       <div class="right">
8       <%= link "Edit", to: topic_path(@conn, :edit, topic) %>
9       <%= link "Delete", to: topic_path(@conn, :delete , topic),
            method: :delete %>
10      </div>
11      <% end %>
```

Updating ‘index.html.eex‘ Template

PRADITA University

```
        </li>
        <%end%>
        </ul>
```

- The ‘if‘ statement ensures that only the topic owner sees the edit/delete options.
- This provides a basic level of authorization on the front-end.

Adding a Plug to Refuse Unauthorized Actions

PRADITA University

- Even though unauthorized users cannot see the buttons, they might still access actions via URL.
- We'll create a plug to refuse unauthorized actions.

Adding a Plug to Refuse Unauthorized Actions

PRADITA University

```
1       def check_topic_owner(conn, _params) do
2       %{params: %{"id" => topic_id}} = conn
3       if Repo.get(Topic, topic_id).user_id == conn.assigns.user.id
            do
4       conn
5       else
```

Adding a Plug to Refuse Unauthorized Actions

PRADITA University

```
    conn
    |> put_flash(:error, "You cannot edit that")
    |> redirect(to: topic_path(conn, :index))
    |> halt()
    end
    end
```

- We compare the ‘user_id‘ of the topic with the ‘id‘ of the logged-in user.
- If they don’t match, we refuse the action.

Conclusion

PRADITA University

- We implemented authorization to restrict actions based on user authentication.
- Users are now associated with their topics.
- These steps ensure better security and user management within our Discuss application.