PRADITA University

IF140303-Web Application Development

Session-11:

Session and Authentication using Plugs in Phoenix Framework

PRU/SPMI/FR-BM-18/0222

Alfa Yohannis

Introduction to GitHub OAuth Authentication

- We'll implement OAuth authentication using GitHub.
- OAuth flow:
  1. User clicks "Login with GitHub" on the web page.
  2. Redirect to GitHub.com for authorization.
  3. After authorization, GitHub redirects back with an authorization code.
  4. Use the authorization code to retrieve user details.
  5. Create a new user record and log the user in.

Ueberauth Helper
Using the Library for OAuth

PRADITA University

- We'll use the `Ueberauth` library to handle the OAuth process.
- Create a new controller to manage OAuth flows.
- When the user clicks "Login with GitHub":
  1. The controller triggers `Ueberauth` to start the OAuth flow.
  2. Redirect to GitHub for authentication.
  3. Handle GitHub's response with a callback function in the controller.

Installing Dependencies

PRADITA University

- Open `mix.exs` to add dependencies:

```
1        defp deps do
2        [{..., :ueberauth, "~> 0.3"},
3        {:ueberauth_github, "~> 0.4"}]
4        end
5        def application do
6        [extra_applications: [...,:ueberauth, :
             ueberauth_github]]
7        end
```

- Run `mix deps.get` to install dependencies.

Generating GitHub API Key

PRADITA University

- Go to GitHub to generate an API key for OAuth.
  1. Navigate to Profile > Settings > Developer Settings.
  2. Click on `OAuth applications` and register a new application.
  3. Fill out the form, including the Authorization Callback URL.
  4. Generate the Client ID and Client Secret.

Including the API Key in Configuration Files

PRADITA University

- Add the API key to `config.exs`:

```
        config :ueberauth, Ueberauth,
        providers: [
        github: {Ueberauth.Strategy.Github, []}]
        config :ueberauth, Ueberauth.Strategy.Github.OAuth,
        client_id: "your_generated_client_id",
        client_secret: "your_generated_secret_id"
```

- This configuration links Ueberauth with GitHub OAuth.
- It's recommended to hide the API key if uploading to a public repository.

Creating the Authentication Controller

- Create a new controller at `web > controllers > auth_controller.ex`:

```
1        defmodule Discuss.AuthController do
2        use Discuss.Web, :controller
3        plug Ueberauth
4        def callback(conn, params) do
5        IO.inspect(conn.assigns)
6        IO.inspect(params)
7        end
8        end
```

- `conn.assigns` returns user data (email, username, etc.).
- `params` returns the authorization code and provider.

Updating the Router

PRADITA University

- Open `router.ex` and update it to include the auth routes:

```
        scope "/", Discuss do
        pipe_through :browser
        resources "/", TopicController
        end

        scope "/auth", Discuss do
        pipe_through :browser
        get "/:provider", AuthController, :request
        get "/:provider/callback", AuthController, :callback
        end
```

- `pipe_through` specifies the pipeline to use.
- The `scope "/auth"` block defines routes for OAuth-related actions.

Generating Migration for Users Table

PRADITA University

- Generate a migration file with `mix ecto.gen.migration add_users`
- In the migration file:

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

def change do
create table(:users) do
add :email, :string
add :provider, :string
add :token, :string
timestamps()
end
end
end
```

- This creates a `users` table with `email`, `provider`, `token`, and timestamps.

Creating the User Model

- Create user model in `web > models > user.ex`:

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

schema "users" do
field :email, :string
field :provider, :string
field :token, :string
timestamps()
end

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

 This defines the structure and validations for the `users` table.

Updating the Auth Controller

- Update auth_controller.ex to handle user authentication:

PRADITA

```
defmodule Discuss.AuthController do
use Discuss.Web, :controller
plug Ueberauth

alias Discuss.User

def callback(%{assigns: %{ueberauth_auth: auth}} =
    conn, _params) do
user_params = %{token: auth.credentials.token,
  email: auth.info.email,
  provider: "github"}
changeset = User.changeset(%User{}, user_params)
insert_or_update_user(changeset)
end

defp insert_or_update_user(changeset) do
case Repo.get_by(User, email: changeset.changes.email)
    do
nil -> Repo.insert(changeset)
```

Short Explanation of Cookies

PRADITA University

- Cookies store user data on the client side, enabling session persistence.
- In our flow:
  1. After successful OAuth, the user's ID is stored in a cookie.
  2. On subsequent requests, the server checks the cookie to retrieve the user ID.
  3. If the user exists, they're automatically signed in.

Adding the Sign-In Functionality

- Update auth_controller.ex to include sign-in:

```
defmodule Discuss.AuthController do
use Discuss.Web, :controller
plug Ueberauth

alias Discuss.User

def callback(%{assigns: %{ueberauth_auth: auth}} =
    conn, _params) do
user_params = %{token: auth.credentials.token,
  email: auth.info.email,
  provider: "github"}
changeset = User.changeset(%User{}, user_params)
signin(conn, changeset)
end

defp signin(conn, changeset) do
case insert_or_update_user(changeset) do
{:ok, user} ->
conn
```