

IF140303-Web Application Development

# Session-11: Applying GitHub OAuth Authentication

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

- 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

- 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

---

- 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

- Add the API key to config.exs:

```
1      config :ueberauth, Ueberauth,  
2        providers: [  
3          github: {Ueberauth.Strategy.Github, []}]  
4        config :ueberauth, Ueberauth.Strategy.Github.OAuth,  
5          client_id: "your_generated_client_id",  
6          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

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

```
1     scope "/", Discuss do
2       pipe_through :browser
3       resources "/", TopicController
4     end
5
6     scope "/auth", Discuss do
7       pipe_through :browser
8       get "[:provider]", AuthController, :request
9       get "[:provider/callback]", AuthController, :callback
10    end
```

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



# Generating Migration

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

## for Users Table

- In the migration file:

```
1      defmodule Discuss.Repo.Migrations.AddUsers do
2      use Ecto.Migration
3
4      def change do
5        create table(:users) do
6          add :email, :string
7          add :provider, :string
8          add :token, :string
9          timestamps()
10       end
11     end
12   end
```

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

Create a user model in web > models > user.ex:

```
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        timestamps()
9      end
10
11     def changeset(struct, params \\ %{}) do
12       struct
13       |> cast(params, [:email, :provider, :token])
14       |> validate_required([:email, :provider, :token])
15     end
16   end
```

This defines the structure and validations for the users table.

Update with controller ex to handle user authentication:



```
1  defmodule Discuss.AuthController do
2
3  use Discuss.Web, :controller
4
5  plug Ueberauth
6
7  alias Discuss.User
8
9  def callback(%{assigns: %{ueberauth_auth: auth}} =
10    conn, _params) do
11    user_params = %{token: auth.credentials.token,
12      email: auth.info.email,
13      provider: "github"}
14    changeset = User.changeset(%User{}, user_params)
15    insert_or_update_user(changeset)
16  end
17
18  defp insert_or_update_user(changeset) do
19    case Repo.get_by(User, email: changeset.changes.email)
20    do
21      nil -> Repo.insert(changeset)
```

# Short Explanation of Cookies

---

- 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.

Update auth\_controller.ex to include sign-in:

## Adding the Sign-In Functionality



```
1  defmodule Discuss.AuthController do
2
3    use Discuss.Web, :controller
4    plug Ueberauth
5
6
7    alias Discuss.User
8
9    def callback(%{assigns: %{ueberauth_auth: auth}} =
10      conn, _params) do
11      user_params = %{token: auth.credentials.token,
12        email: auth.info.email,
13        provider: "github"}
14      changeset = User.changeset(%User{}, user_params)
15      signin(conn, changeset)
16    end
17
18    defp signin(conn, changeset) do
19      case insert_or_update_user(changeset) do
20        {:ok, user} ->
21          conn
```