CONTROLLER CONTROL

DESIGNING DOMAINS FOR WEB APPLICATIONS

FORMAT

- Move non-web stuff out of web
- Umbrellas

DEFAULT PHOENIX STRUCTURE

- Most of the code in web
- Some files in lib (endoint.ex, repo.ex)
- Split into controllers, models, views and templates

```
tree -d -I "node_modules|_build|deps|static" .

├── config
├── lib
│   └── oxo
├── priv
│   ├── gettext
│   └── repo
│       └── migrations
├── test
│   ├── channels
│   ├── controllers
│   ├── models
│   ├── oxo
│   ├── support
│   └── views
└── web
    ├── channels
    ├── controllers
    ├── models
    ├── templates
    │   ├── challenge
    │   ├── game
    │   ├── layout
    │   ├── page
    │   ├── session
    │   └── user
    └── views
```

WHAT IS A CONTROLLER?

- Glue code between model and view
- Should focus on web layer
- (Generally) last part of endpoint pipeline
- Interfaces with conns
- Friends with sockets and channels

A CONTROLLER IS NOT THE INTERFACE TO YOUR DATABASE

STOP USING REPO IN CONTROLLER

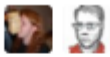
```
$ grep -ir "Repo" apps/oxo_web/web/
/controllers/game_controller.ex:    challenge = Repo.get(Challenge, id)
/controllers/game_controller.ex:        |> Repo.update()
/controllers/user_controller.ex:    case Repo.insert(changeset) do
/controllers/challenge_controller.ex:      |> Repo.all()
/controllers/challenge_controller.ex:      |> Repo.preload(:user)
/controllers/challenge_controller.ex:      |> Repo.insert!()
/web.ex:      alias Oxo.Repo
```

Sean Williamson
@SuperNullSet

Follow

@TheGazler Your suggestion to keep Repo out of controllers has been extraordinarily helpful! It led to a massive improvement in my app

LIKES
2

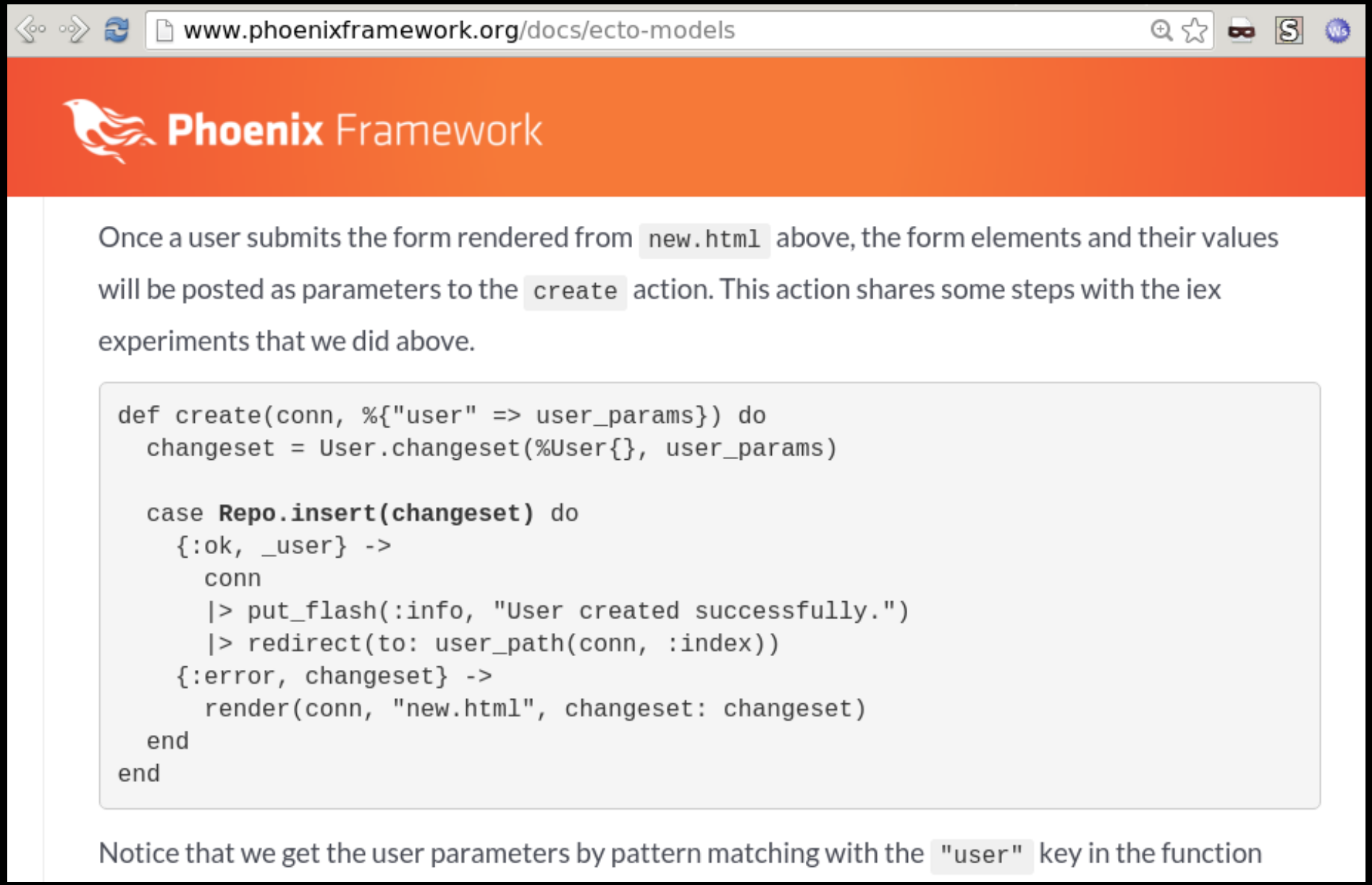


1:40 AM - 14 Sep 2016

2

Reply to @SuperNullSet

WHY IS REPO IN CONTROLLER?


www.phoenixframework.org/docs/ecto-models
Phoenix Framework
Once a user submits the form rendered from `new.html` above, the form elements and their values will be posted as parameters to the `create` action. This action shares some steps with the iex experiments that we did above.
def create(conn, %{"user" => user_params}) do
changeset = User.changeset(%User{}, user_params)
case Repo.insert(changeset) do
{:ok, _user} ->
conn
|> put_flash(:info, "User created successfully.")
|> redirect(to: user_path(conn, :index))
{:error, changeset} ->
render(conn, "new.html", changeset: changeset)
end
end
Notice that we get the user parameters by pattern matching with the `"user"` key in the function

```
  def create(conn, %{<%= inspect singular %> => <%= singular %>_params}) do
    changeset = <%= alias %>.changeset(%<%= alias %>{}, <%= singular %>_params)

    case Repo.insert(changeset) do
      {:ok, _<%= singular %>} ->
        conn
        |> put_flash(:info, "<%= human %> created successfully.")
        |> redirect(to: <%= singular %>_path(conn, :index))
      {:error, changeset} ->
        render(conn, "new.html", changeset: changeset)
    end
  end

  def show(conn, %{"id" => id}) do
    <%= singular %> = Repo.get!(<%= alias %>, id)
    render(conn, "show.html", <%= singular %>: <%= singular %>)
  end

  def edit(conn, %{"id" => id}) do
    <%= singular %> = Repo.get!(<%= alias %>, id)
    changeset = <%= alias %>.changeset(<%= singular %>)
    render(conn, "edit.html", <%= singular %>: <%= singular %>, changeset: changeset)
  end

```

The Pragmatic Programmers

Your Elixir Source

Programming Phoenix

Productive |> Reliable |> Fast

Chris McCord,
Bruce Tate,
and José Valim

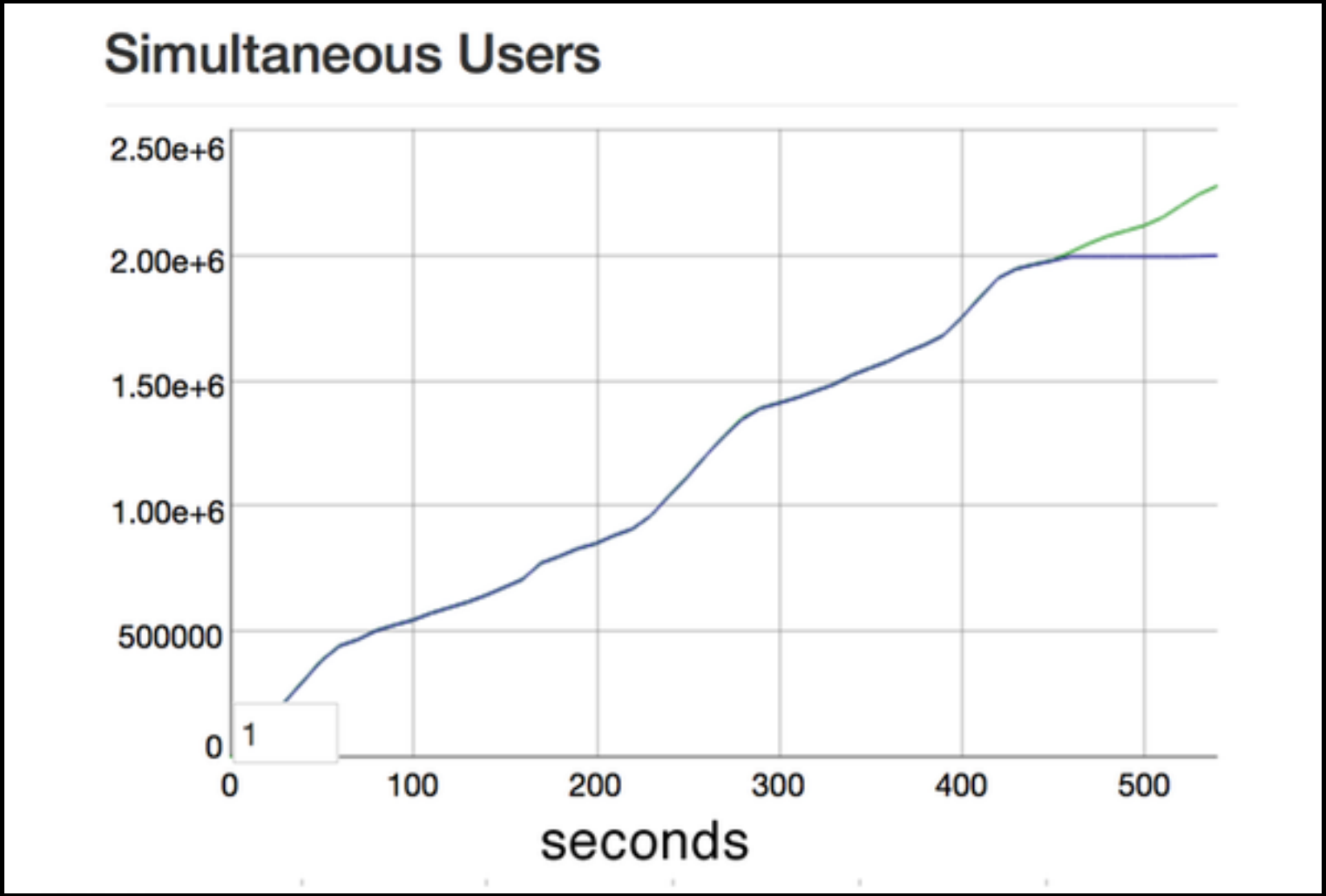
edited by Jacquelyn Carter

GARY POTTER

and the Order of the Phoenix

J.VALIM

elixir press


Simultaneous Users
2.50e+6
2.00e+6
1.50e+6
1.00e+6
500000
0
1
0
100
200
300
400
500
seconds

Should I use Ecto.Repo in Controller or Model for Elixir Phoenix?

1 Answer

active | oldest | votes

13

You should keep your Repo calls inside your controller. If your logic is complicated then you should consider moving the logic out into its own service module

You should treat your model functions as pure (free from side effects) so they should only act on data. So for example you could have:

```
def alphabetical(query)
  order_by(query, [u], u.name)
end
```

But you should not have:

```
def alphabetical(query)
  order_by(query, [u], u.name)
  |> Repo.all
end
```

This is because queries are purely data, the call to `Repo.all` has side effects (going off to the database) so it belongs in your controller.

share edit delete flag

answered Dec 17 '15 at 17:30

Gazler

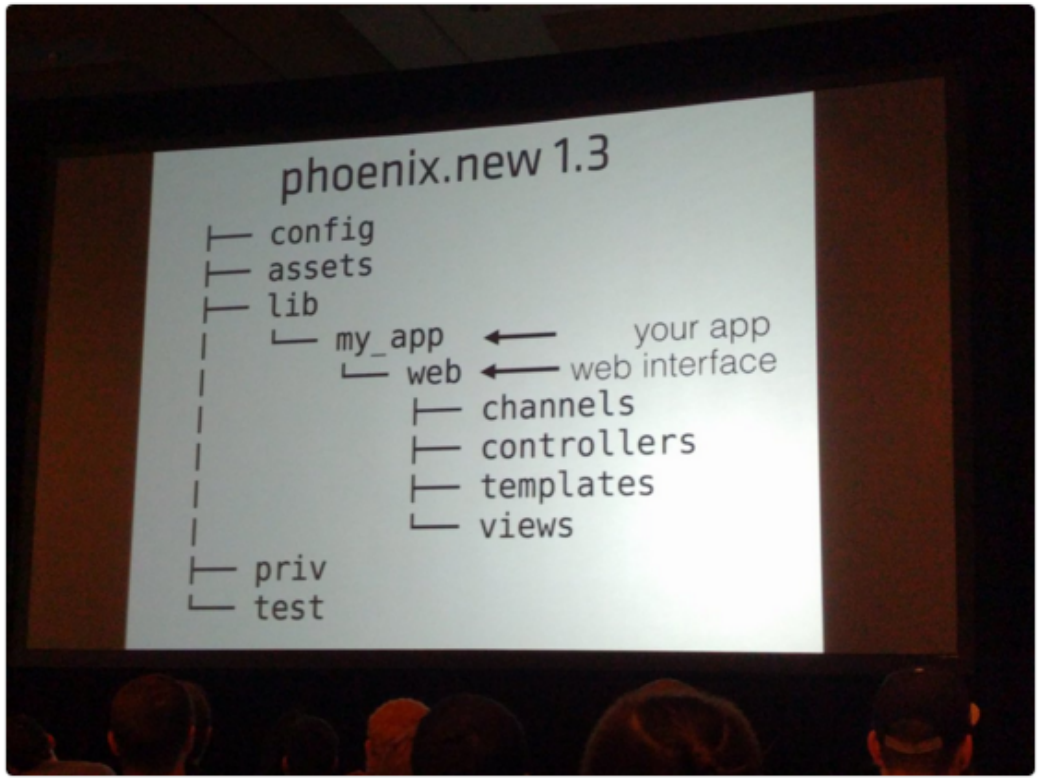
48.2k ● 8 ● 121 ● 144

WHY IS REPO IN CONTROLLER?
WE SAID IT SHOULD BE
BUT...
THINGS EVOLVE

Anton Domratchev
@antondom

Follow

New and improved Phoenix 1.3 directory structure. Keep your web in your app! #elixirconf

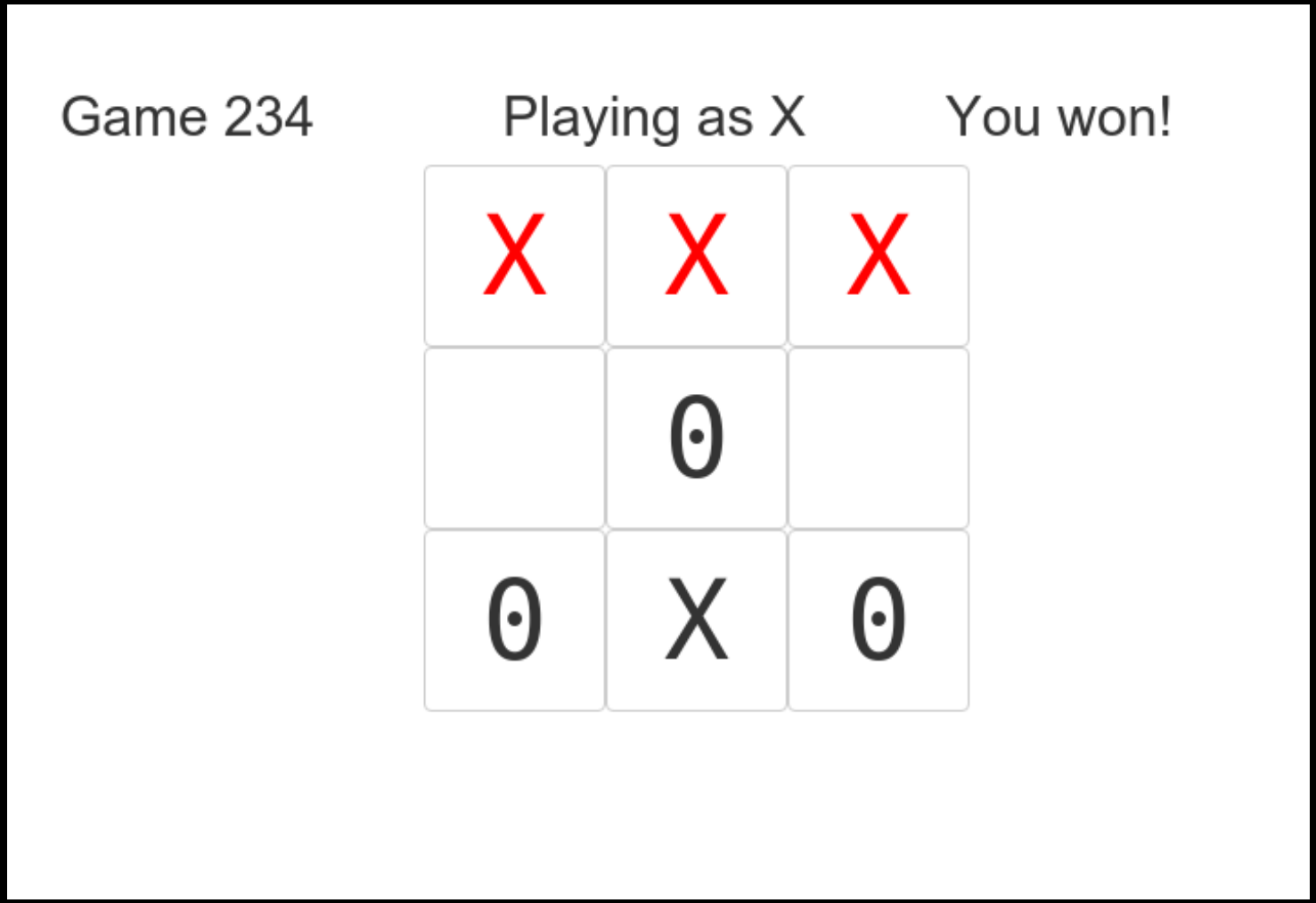


RETWEETS 3 LIKES 8

2:37 PM - 1 Sep 2016

3 8

WORLD OF GRAND THEFT POKÉMON GO 2.0



TODO: SHOW APPLICTION

HOW TO NOT REPO

- Create functions in "context"
- Functions should do everything non-web related
- Delete Repo alias in web.ex

REGISTER USER

MOVE FROM CONTROLLER

```elixir
def create(conn, params) do
  {%{"email" => email, "password" => password}, player_p} =
    Map.split(params, ["email", "password"])
  case Accounts.register_player(email, password, player_p) do
    {:ok, player} -> redirect(...)
    {:error, changeset} -> render(...)
  end
end
```

```elixir
defmodule Oxo.Accounts do
  alias Oxo.{Player, Repo, Auth}

  def register_player(email, password, player_params) do
    Ecto.Multi.new()
    |> Ecto.Multi.run(:account, fn _ ->
      Auth.register(email, password)
    end)
    |> Ecto.Multi.run(:player, fn %{account: account} ->
      %Player{account_id: account.id}
      |> Player.changeset(player_params)
      |> Repo.insert()
    end)
    |> Repo.transaction()
    |> handle_new_registration()
  end
end
```

```elixir
defmodule Oxo.Accounts do
  alias Oxo.Mailer

  defp handle_new_registration({:ok, results}) do
    player = %{results.player | email: results.account.email}
    Mailer.send_registration_email(player)
    {:ok, player}
  end

  defp handle_new_registration({:error, _, changeset, _}) do
    {:error, changeset}
  end
end
```

REGISTER USER

- Accounts is the context
- Contains all database logic
- Leverages the auth context
- Handles email
- Transform response

MODULE BENEFITS

- Domain logic no longer tied to controller
- We can provider other interfaces to the function
  - Mix Task
  - Telnet
  - Email
- Can test the flow without the controller

LISTING GAME CHALLENGES

```
def index(conn, _params) do
  current_user = conn.assigns.current_user
-  player = Repo.preload(current_user, :player).player
-  challenges =
-    Challenge
-    |> Ecto.Query.where(open: true)
-    |> Ecto.Query.where([c], c.player_id != ^player.id)
-    |> Repo.all()
-    |> Repo.preload(:player)
+  challenges =
+    |> Accounts.get_player(current_user)
+    |> Game.list_open_challenges()
  render(conn, "index.html", challenges: challenges)
end
```

- **Doesn't matter where challenges are fetched from**

LISTING GAME CHALLENGES

```elixir
defmodule Oxo.Game do
  def get_player(%Auth.Account{} = account) do
    Repo.preload(account, :player).player
  end

  def list_open_challenges(%Player{} = player) do
    Challenge
    |> Ecto.Query.where(open: true)
    |> Ecto.Query.where([c], c.player_id != ^player.id)
    |> Repo.all()
    |> Repo.preload(:player)
  end
end
```

- Game is the context
- All database logic here

CREATE NEW CHALLENGE

```
def create(conn, _params) do
  current_player = conn.assigns.current_player
-  challenge =
-    current_player
-    |> Ecto.build_assoc(:challenges)
-    |> Challenge.changeset(%{})
-    |> Repo.insert!()
-
+  challenge = Game.issue_open_challenge!(current_player)
  redirect(conn, to: game_path(conn, :show, challenge))
end
```

- **This was an easy case because we use insert!**

CREATE NEW CHALLENGE

```elixir
defmodule Oxo.Game do
  def issue_open_challenge!(%Player{} = player) do
    player
    |> Ecto.build_assoc(:challenges)
    |> Challenge.changeset(%{})
    |> Repo.insert!()
  end
end
```

- Game is still the context
- Try to use descriptive verbs (issue instead of create)

CLOSE CHALLENGE

(GAMECONTROLLER.SHOW)

```
-    challenge = Repo.get(Challenge, id)
-    current_player_id = conn.assigns.current_player.id
-    case challenge do
-      %Challenge{player_id: ^current_player_id} ->
-        player_token = Phoenix.Token.sign(...)
-        render(...)
-      %Challenge{} ->
-        challenge
-        |> Challenge.changeset(%{open: false})
-        |> Repo.update()
+    case Game.close_challenge(id, current_player) do
+      {:ok, challenge} ->
         player_token = Phoenix.Token.sign(...)
         render(...)
       _ ->
        conn
        |> put_flash(:error, "Could not find challenge")
        |> redirect(to: challenge_path(conn, :index))
```

CLOSING A CHALLENGE

```elixir
def close_challenge(id, %Player{id: player_id} = player) do
  case Repo.get(Challenge, id) do
    %Challenge{player_id: ^player_id} = challenge ->
      {:ok, challenge}

    %Challenge{} = challenge ->
      update_challenge(challenge, %{open: false})

      {:ok, challenge}
    _ -> {:error, :not_found}
  end
end
```

CONVERTING CONTROLLERS SUMMARY

- Remove Repo alias from web.ex
- Hope this causes compilation errors
- grep -ir Repo controllers/
- Convert controller body to context module function
- Rename controllers/channels to be namespaced in web
- Do one controller at a time
- Commit at the end!

SO...MODELS

S/MODELS/SCHEMAS/G


IPHONE 6
IPHONE 7
STARECAT.COM

WHAT IS A MODEL?

- A representation of your domain (domain model)
- Nothing to do with the database tables
- Can leverage the database
- Can encapsualte many database transactions
- Can do other things too (email, etc.)

WHAT IS A SCHEMA?

- A representation of your table row
- Can reference other schemas (has_many, belongs_to)
- Can perform data integrity validations
  - Can't be blank
  - Must be unique (enforced by db index)
  - Not can't create on a Tuesday

CHANGING MODELS TO SCHEMAS

- Could just have rename "models" directory to "schemas"
- schemas/challenge.ex, schemas/player.ex

- Or nest schemas in context instead
- game/challange.ex, accounts/player.ex
- change ModelCase to DatabaseCase

```
tree -d -I "node_modules|_build|deps|static" .

├── config
├── lib
│   └── oxo
│       ├── accounts
│       ├── auth
│       └── game
├── priv
│   ├── gettext
│   └── repo
│       └── migrations
├── test
│   ├── channels
│   ├── controllers
│   ├── models
│   ├── oxo
│   ├── support
│   └── views
└── web
    ├── channels
    ├── controllers
    ├── plugs
    ├── templates
    │   ├── challenge
    │   ├── game
    │   ├── layout
    │   ├── page
    │   ├── session
    │   └── user
    └── views
```

MOVING WEB

```
git mv web/static/ assets/
git mv package.json assets/
mv node_modules/ assets/
echo "assets/node_modules" >> .gitignore
git mv web/ lib/oxo/web/
find lib/oxo/web/ -type f -exec sed -i \
  's/defmodule Oxo/defmodule Oxo.Web/g' {} +
```

```
tree -d -I "node_modules|_build|deps|static" .

├── assets
├── config
├── lib
│   └── oxo
│       ├── accounts
│       ├── auth
│       ├── game
│       └── web
│           ├── channels
│           ├── controllers
│           ├── plugs
│           ├── templates
│           │   ├── challenge
│           │   ├── game
│           │   ├── layout
│           │   ├── page
│           │   ├── session
│           │   └── user
│           └── views
├── priv
│   ├── gettext
│   └── repo
│       └── migrations
└── test
    ├── channels
    ├── controllers
    ├── models
    ├── oxo
    ├── support
    └── views
```

UMBRELLA APPLICATIONS

WHY USE AN UMBRELLA APPLICATION?

- Can run and develop each application on its own
- Can test each application on its own
- Makes contracts between applications explicit
- Releases can be built for a part of the umbrella
- Easy to extract as a dependency (e.g. hex, private repo)

WHY NOT USE AN UMBRELLA APPLICATION?

- More complicated?
- Can be harder to test the entire suite
  - Process name collisions
  - Applications started at the wrong time
  - Dependency conflicts harder to resolve (multiple overrides)
- Developing a new feature touches multiple applications

SHOULD YOU USE AN UMBRELLA APPLICATION?

IT DEPENDS!

THE GOOD NEWS

- You don't need to decide today!
- Moving code around in Elixir is easy
- Moving code between applications is quite easy
- Possible to convert from an umbrella to single application

LET'S BEGIN!

```
mix new oxo --umbrella
```

- Create a new umbrella application
- Even though we already have our application!

RENAME UMBRELLA APPLICATION

```
defmodule Oxo.Mixfile do
  use Mix.Project
  #...
end
```

- There is already an Oxo.Mixfile in the Phoenix application
- Rename to Oxo.Platform.Mixfile
- or something equally as generic

SET UP GIT FOR UMBRELLA

```
git init
git add .
git commit -m "initial commit"
```

- Git is important for the next step
- Good idea to commit before making changes
- Prevent a massive initial commit

ADD EXISTING APPLICATION TO UMBRELLA

- Could just `cp` the directory
  - Loses git history
  - No longer possible to bisect, blame, etc.
  - Introduces a massive import commit
- There is a better way!

INTRODUCTING GIT SUBTREE

- Imports code into directory (unlike submodules)
- Maintains entire history
- Can subtree merge more than once

SUBTREE ADD THE APPLICATION

```
git remote add oxo_remote git@github.com:Gazler/oxo.git
git fetch oxo_remote
git subtree add --prefix apps/oxo_web oxo_remote/master
```

- Entire history is preserved! (With original SHAs)
- Some changes required since we use oxo_web
- Application could have been called `oxo` instead

UPDATE APP CONFIG

```
def project do
  [#...
  build_path: "../../_build",
  config_path: "../../config/config.exs",
  deps_path: "../../deps",
  lockfile: "../../mix.lock",
  #...
  ]
end
```

- Share build, config and lock files
- git mv apps/oxo_web/mix.lock mix.lock

CHECK EVERYTHING IS WORKING

- Run mix deps.get && mix test
- Hope everything works!
- Run mix phoenix.server for good measure
- Don't forget to cd `apps/oxo_web/assets && npm install`!

SPLITTING OUT OUR APPLICATION

- Web stays, everything else goes
- Remove concerns of other applications
  - Using dependencies of other apps directly (Comeonin, etc.)
  - Config for other applications
- Remember to move the tests too

MAKE A NEW `OXO` APPLICATION

- mix new oxo --sup
- Add dependency in oxo_web
- Don't forget to add it to applications list

```
defp deps do
  [{:oxo, in_umbrella: true},
   {:phoenix, "~> 1.2.0"},
   {:phoenix_pubsub, "~> 1.0.0"},
   #...
   {:cowboy, "~> 1.0"}]
end
```

MOVE DATABASE TO CORE APPLICATION

- Move database config from config/*.exs
- Update ecto_repos in config/config.exs
- Add Ecto and other dependencies (Comeonin, etc.)
- Add repo in app supervisor

MOVE TESTS OVER

- A commit with failing tests is not a good commit
- Will need DatabaseCase
- Update elixirc_paths in mix.exs
- Update test_helper.exs

```
tree -d -I "node_modules|_build|deps|static" .

├── apps
│   ├── oxo
│   │   ├── config
│   │   ├── lib
│   │   │   └── oxo
│   │   │       ├── accounts
│   │   │       ├── auth
│   │   │       └── game
│   │   ├── priv
│   │   │   └── repo
│   │   │       └── migrations
│   │   └── test
│   │       ├── oxo
│   │       └── support
│   └── oxo_web
│       ├── assets
│       ├── config
│       ├── lib
│       │   └── oxo
│       │       └── web
│       │           ├── channels
│       │           ├── controllers
│       │           ├── plugs
│       │           ├── templates
│       │           └── views
│       ├── priv
│       │   └── gettext
│       └── test
│           ├── channels
│           ├── controllers
```

MIGRATING TO UMBRELLA SUMMARY

- Not too difficult to migrate to an umbrella application
- Entire git history can be maintained
- grep -ir Repo apps/oxo_web should return no results

GOING FORWARD

- There can be more than 2 applications
- Try to think in applications
- Split out common functions into own applications
  - Authentication
  - Email/Notifications
  - Game logic

Gazler @TheGazler

voicelayer

https://github.com/Gazler/oxo