

Level 3 - Section 1

New Records

Using *Elixir* to Output HTML Forms



From Listing to Creating

So far we have a page that lists all videos. Now we need a form page so we can **create new videos!**

- Elixir
- JavaScript
- Go

```
<form>  
</form>
```

Submits new video
and adds to list

Steps for Creating New Video Records

Let's write *Phoenix* code that allows users to create new video records. This includes:

1. Adding new routes to render a form and create a new record
2. Using form helpers to generate HTML forms.
3. Defining `:create` actions in the *Controller*

Requests for Creating a New Video

1 Render the form



Browser

GET /videos/new



Phoenix

`VideoController.new()`



<form>
</form>

2 Handle form submission



Browser

POST /videos

{ title: "PHP 101", }



Phoenix

`VideoController.create()`



- Elixir
- JavaScript
- Go
- **PHP 101**

Routes for a New Video

Using the router DSL, we use `get()` and `post()`, passing them the **path, controller and actions**.

lib/fire_starter_web/router.ex

```
defmodule FireStarterWeb.Router do
  ...
  scope "/", FireStarterWeb do
    ...
    get "/videos", VideoController, :index
    get "/videos/new", VideoController, :new
    post "/videos", VideoController, :create
  end
end
```


Path Helpers

Path helpers are functions **dynamically generated**, derived from each **controller** in the router.

lib/fire_starter_web/router.ex

```
defmodule FireStarterWeb.Router do
  ...
  scope "/", FireStarterWeb do
    ...
    get "/videos", VideoController, :index
    get "/videos/new", VideoController, :new
    post "/videos", VideoController, :create
  end
end
```

returns `"/videos"`, `"videos/new"`
and `"/videos"` respectively

connection made available
from the *Controller* action

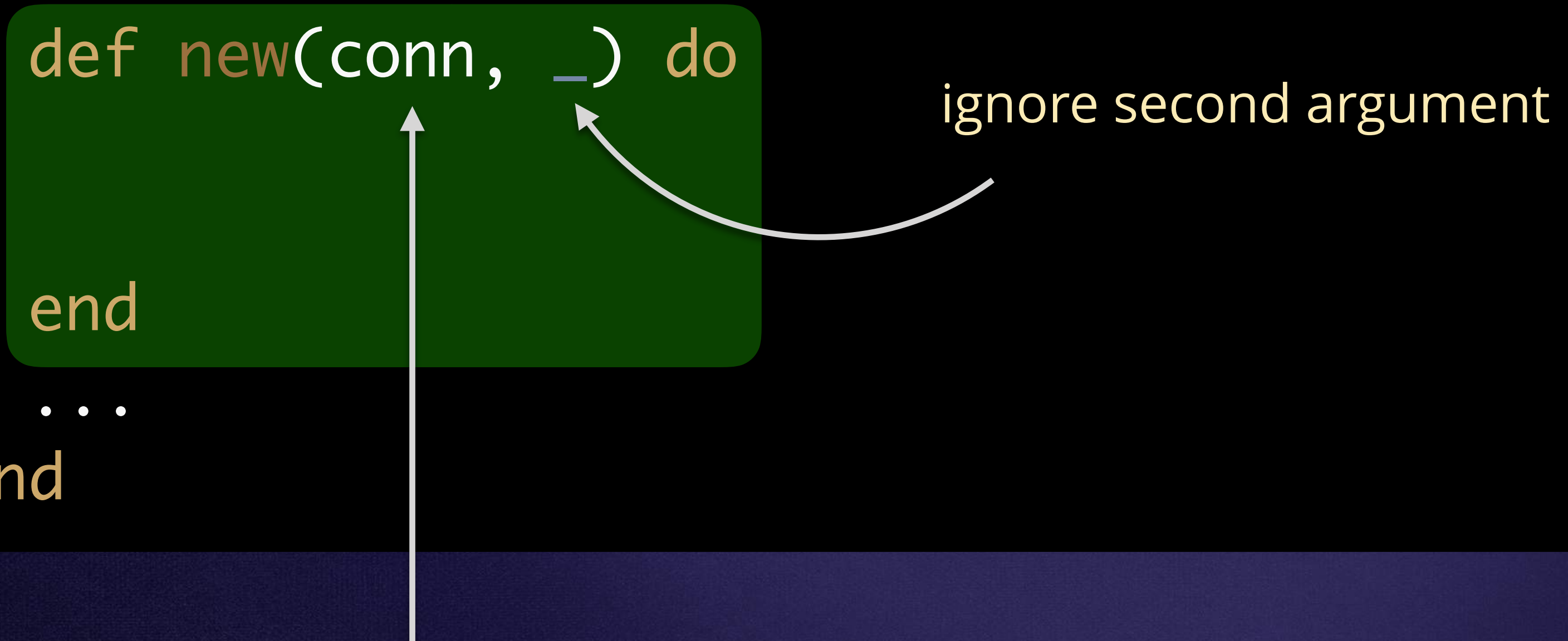
```
video_path(@conn, :index)
video_path(@conn, :new)
video_path(@conn, :create)
```


The new action

Requests to “/videos/new” are routed to the new() function in the VideoController.

lib/fire_starter_web/controllers/video_controller.ex

```
defmodule FireStarterWeb.VideoController do
  ...
  def new(conn, _) do
    ...
  end
  ...
end
```



the *connection* is **always** the first argument to controller actions

ignore second argument

Creating a *changeset*

A *changeset* is a *Struct* representing changes made to the underlying *Schema*.*

lib/fire_starter_web/controllers/video_controller.ex

```
defmodule FireStarterWeb.VideoController do
```

```
...
```

```
import Ecto.Changeset
```

```
def new(conn, _) do
```

```
  changeset = change(%Video{})
```

```
  render conn, "new.html", changeset: changeset
```

```
end
```

```
...
```

```
end
```

empty changeset

available from here

will be made available
to the template

* Passing an empty changeset to the template helps us build our form fields.

Using form_for to Create Forms

The `form_for()` function takes a **changeset**, a **url path** and an **anonymous function**.

template file rendered from the
new action in the *VideoController*

lib/fire_starter_web/templates/video/**new.html.eex**

```
<%= form_for @changeset, video_path(@conn, :create), fn f -> %>
```

```
<h3>New Video</h3>
```

helper function
returns **"/videos"**

anonymous function

```
<% end %>
```


Using text_input to text inputs

The `text_input()` function creates an **HTML input** of type `text` and with the name given as the second argument.

argument to this function helps
generate names for the input fields...

lib/fire_starter_web/templates/video/new.html.eex

```
<%= form_for @changeset, video_path(@conn, :create), fn f -> %>
```

```
<h3>New Video</h3>
```

```
<p>Title: <%= text_input f, :title %></p>
```

...used as first argument to
form input helpers

```
<% end %>
```

populates **name** attribute on HTML
input with the value "video[title]"

The Rest of the Form Fields

The remaining form fields call the same `text_input()` with their respective *names*.

lib/fire_starter_web/templates/video/new.html.eex

```
<%= form_for @changeset, video_path(@conn, :create), fn f -> %>
  <h3>New Video</h3>
  <p>Title: <%= text_input f, :title %></p>
  <p>Url:<%= text_input f, :url %></p>
  <p>Duration: <%= text_input f, :duration %></p>
  <p><%= submit "Create" %></p>
<% end %>
```

generates submit button

The Form Markup

The `form_for()` function generates form markup with some added features, like:

- Protection against *Cross-Site Request Forgery* (CSRF) Attacks
- Attributes forcing browsers to use UTF-8 as the charset.
- Naming convention for input fields (`module[field]`)

```
<form accept-charset="UTF-8" action="/videos" method="post">
  <input name="_csrf_token" type="hidden" value="GRgSJipkFiJcKR....">
  <input name="_utf8" type="hidden" value="✓">
  <h3>New Video</h3>
  <p>Title: <input id="video_title" name="video[title]" type="text"></p>
  ...
</form>
```


The First Step Is Done!



1

Render the form



Browser

GET /videos/new



Phoenix

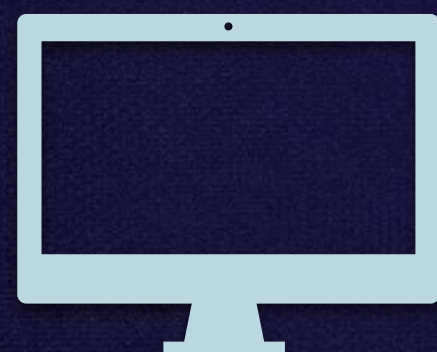
`VideoController.new()`



`<form>`
`</form>`

2

Handle form submission



Browser

POST /videos

{ title: "PHP 101", }



Phoenix

`VideoController.create()`



- Elixir
- JavaScript
- Go
- **PHP 101**

Level 3 - Section 2

New Records

Reading User Input and
Creating New Records



Handling Form Submission



1

Render the form



Browser

GET /videos/new



Phoenix

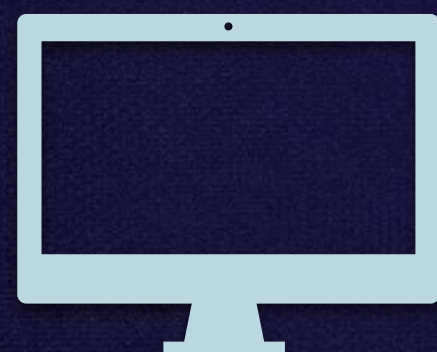
`VideoController.new()`



```
<form>
</form>
```

2

Handle form submission



Browser

POST /videos

{ title: "PHP 101", }



Phoenix

`VideoController.create()`



- Elixir
- JavaScript
- Go
- **PHP 101**

The create Action

The `create()` function takes two arguments: the **connection** and the **user form data**.

lib/fire_starter_web/controllers/video_controller.ex

```
defmodule FireStarterWeb.VideoController do
```

```
  ...  
  import Ecto.Changeset
```

using pattern matching to
read user data from a *Map*

```
  def create(conn, %{"video" => video_params}) do
```

```
  end
```

```
end
```

always the first argument to *Controller* actions!

Casting Form Data to Expected Types

The `cast()` function transforms user input data from String to their corresponding types and filters allowed fields.

lib/fire_starter_web/controllers/video_controller.ex

```
defmodule FireStarterWeb.VideoController do
  ...
  import Ecto.Changeset

  def create(conn, %{"video" => video_params}) do
    changeset = cast(%Video{}, video_params, [:title, :url, :duration])

  end
end
```

Given this data
(empty Video for now)...

...apply this data
sent by the user...

...but allow **ONLY** these
fields to be populated

Inserting Data

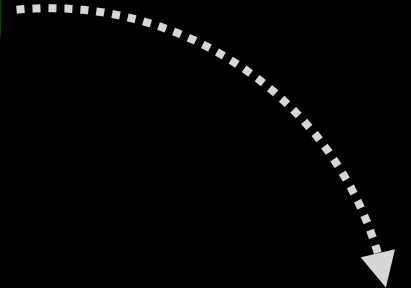
The `Repo.insert()` function takes a **changeset** as its single argument and translates it to an INSERT SQL statement.

lib/fire_starter_web/controllers/video_controller.ex

```
defmodule FireStarterWeb.VideoController do
  ...
  import Ecto.Changeset

  def create(conn, %{"video" => video_params}) do
    changeset = cast(%Video{}, video_params, [:title, :url, :duration])
    Repo.insert(changeset)

  end
end
```



SQL: INSERT INTO "videos" ...

Using case to Pattern Match

We can use the `case` statement to check for the return from `Repo.insert()` - responses are either `{ :ok, record }` or `{ :error, changeset }`.

```
defmodule FireStarterWeb.VideoController do
  ...
  import Ecto.Changeset

  def create(conn, %{"video" => video_params}) do
    changeset = cast(%Video{}, video_params, [:title, :url, :duration])
    case Repo.insert(changeset) do
      { :ok, _ } ->
      { :error, changeset } ->
    end
  end
end
```

ignoring this value for now

new changeset includes any validation errors

Inserting Data with Success

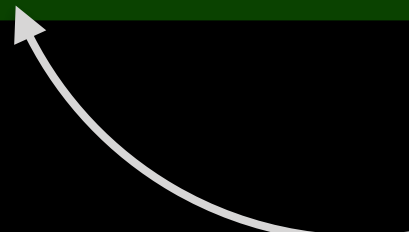
On successful responses, we use `put_flash()` to store a flash message for display and `redirect()` to issue a **301 HTTP response**, redirecting the request to the `:index` action.

```
...
def create(conn, %{"video" => video_params}) do
  changeset = cast(%Video{}, video_params, [:title, :url, :duration])
  case Repo.insert(changeset) do
    {:ok, _} ->
      conn
      |> put_flash(:info, "Video created successfully")
      |> redirect(to: video_path(conn, :index))
    {:error, changeset} ->
      ↑
      path helper function returns "/videos"
  end
end
```


Error When Inserting Data

When inserting a new record is **not** successful, then we `render()` the form again and pass the **new changeset** including the errors.

```
...
def create(conn, %{"video" => video_params}) do
  changeset = cast(%Video{}, video_params, [:title, :url, :duration])
  case Repo.insert(changeset) do
    {:ok, _} ->
      ...
    {:error, changeset} ->
      conn
      |> put_flash(:error, "Error creating video")
      |> render "new.html", changeset: changeset
  end
end
```



includes **errors** which prevented the insert from being successful

The Complete create Action

This is all the code for the create action in the *VideoController*.

```
...
def create(conn, %{"video" => video_params}) do
  changeset = cast(%Video{}, video_params, [:title, :url, :duration])
  case Repo.insert(changeset) do
    {:ok, _} ->
      conn
      |> put_flash(:info, "Video created successfully")
      |> redirect(to: video_path(conn, :index))
    {:error, changeset} ->
      conn
      |> put_flash(:error, "Error creating video")
      |> render "new.html", changeset: changeset
  end
end
```


It Works for Valid Data

When filling in the form with valid data, it works as expected.

```
<form>  
Title: PHP 101  
Url: example.com/php-101  
Duration: 100  
</form>
```

Video Created Successfully

- Elixir
- JavaScript
- Go
- **PHP 101**

Submission Errors Are Not Clear

If something goes wrong on the form submission, it's hard to tell where the error is.

```
<form>  
Title: PHP 101  
Url: example.com/php-101  
Duration: super quick  
</form>
```

Error Creating Video

```
<form>  
Title: PHP 101  
Url: example.com/php-101  
Duration: super quick  
</form>
```

The form looks **the same**.
Could use some help to **spot the error!**

Adding Error Helpers on Template

The `error_tag` function generates a tag for input errors, when they exist.

lib/fire_starter_web/templates/video/new.html.eex

```
...  
<p>Title: <%= text_input f, :title %>  
  <%= error_tag f, :title %></p>  
<p>url: <%= text_input f, :url %>  
  <%= error_tag f, :url %></p>  
<p>Duration: <%= text_input f, :duration %>  
  <%= error_tag f, :duration %></p>  
...
```

`is invalid`

The error tag for a field, when an error exists.

The Rendered Form Displaying Errors

Error Creating Video

<form>

Title: PHP 101

Url: example.com/php-101

Duration: super quick

is invalid

</form>

Now we know where to look!

Two Essential Steps to Create New Videos



1

Render the form



Browser

GET /videos/new



Phoenix

`VideoController.new()`

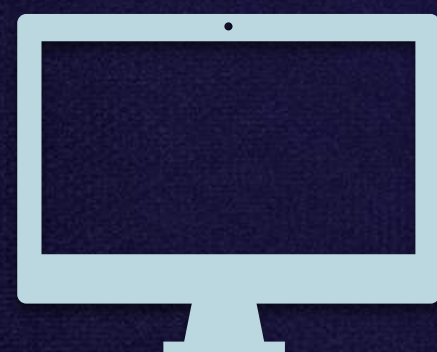


`<form>`
`</form>`



2

Handle form submission



Browser

POST /videos

{ title: "PHP 101", }



Phoenix

`VideoController.create()`



- Elixir
- JavaScript
- Go
- **PHP 101**