Level 4 - Section 1

Migrations & Associations

Creating a New Database Table

On Fire WITH PHOENIX

Showing *Video* with *Comments*

Title: Elixir

Comments:

- *Very Helpful!*
- Great video.

We want to show *comments* for each *video*...

...but first we need to create a database **table** for *comments*.

Reference to the **parent** video

```
comments

 id |        body              |    author    |  video_id
--+-----------------------+-------------+-------------
  1 | Very helpful!         |    Brooke    |      42
  2 | Great video           |    Sam       |      42
```

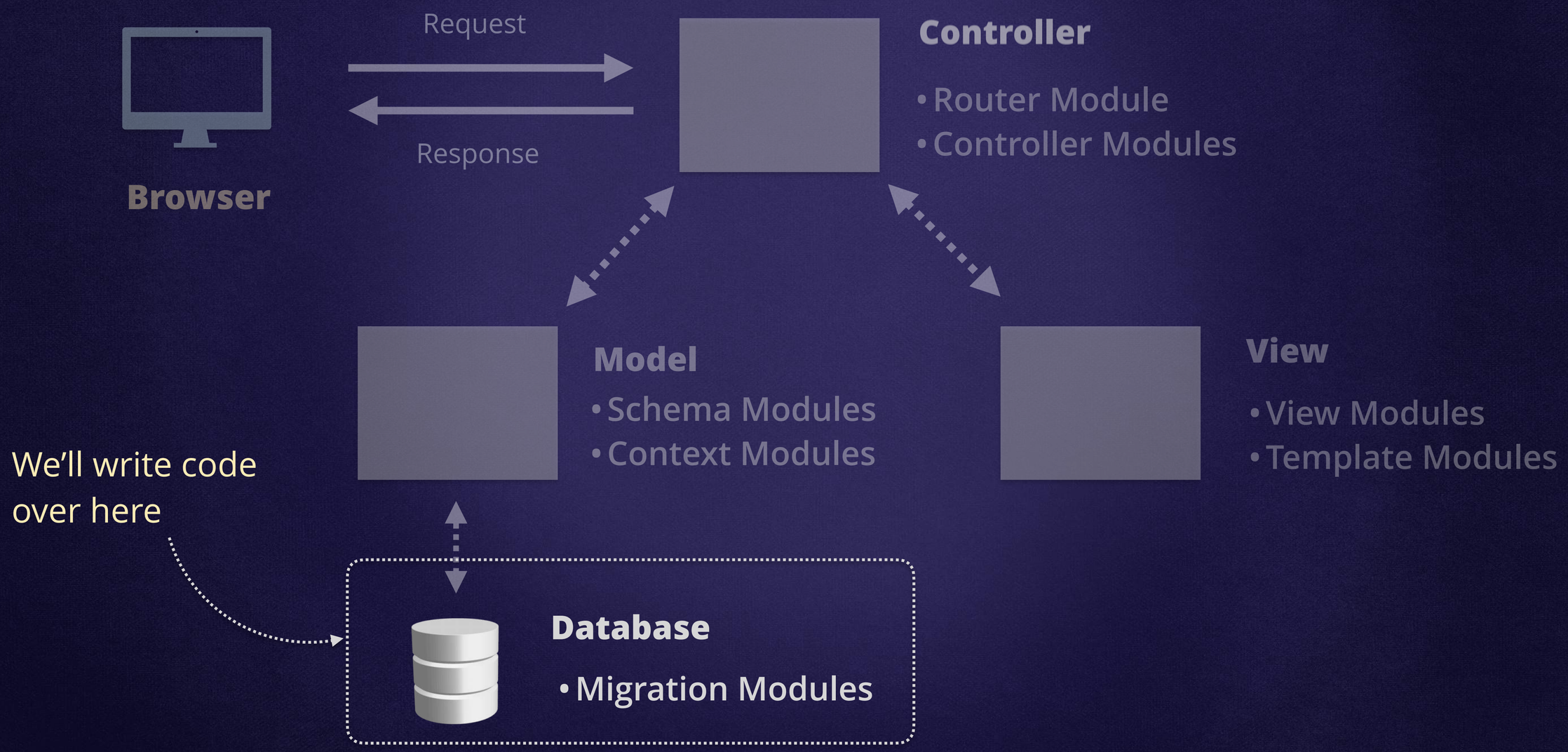
In This Level

We'll learn how to make changes to the database and how to create associations between *Schema Modules*. Things we'll learn include:

- Use **migrations** to create new tables
- Define **foreign key** fields and relationships
- Load **associated** records

Where Migrations Fit

In *Phoenix*, we use **migration files** in order to make changes to the database.



What Are Migrations ?

Migrations are changes to the database structure expressed as *Elixir* code.

Elixir

Database

each and every database change
must be expressed as a migration

A Migration Module

Our migration modules are submodules of the FireStarter.Repo.Migrations **module.**

file creation date helps sort migrations

priv/repo/migrations/**20170523182010**_add_comments_table.exs

```
defmodule FireStarter.Repo.Migrations.AddCommentsTable do


end
```

The change() function

Inside the change() **function we write code that will be translated to SQL statements.**

priv/repo/migrations/20170523182010_add_comments_table.exs

```
defmodule FireStarter.Repo.Migrations.AddCommentsTable do

  def change do

  end
end
```

must be named `change`

code inside this function
will be translated to SQL

Creating a Table

In order to **create** a new table, we can use two functions: create() **and** table().

priv/repo/migrations/20170523182010_add_comments_table.exs

```
defmodule FireStarter.Repo.Migrations.AddCommentsTable do
  use Ecto.Migration

  def change do
    create table(:comments) do



    end
  end
end
```

both functions available
from this module

defaults to a **primary key**
of name `id` and type `serial` 👍

name of the table
to be created

Adding Columns

Also part of Ecto.Migration**, the** add() **function adds a new column to the table.**

column name and type are **required**

```
...
create table(:comments) do
  add :body, :text, null: false
  add :author, :text

  timestamps()
end
```

optional fields

similar to the one used on **Schema Modules**

the same as this

```
add :inserted_at, :naive_datetime
add :updated_at, :naive_datetime
```

Defining a Foreign Key

The references() function is used to define a foreign key to another database table.

```
...
create table(:comments) do
  add :body, :text, null: false
  add :author, :text
  add :video_id, references(:videos, on_delete: :delete_all)

  ...
end
```

deletes all *comment* records when its parent *video* record is deleted

a foreign key to the ***videos*** table is created

Creating a Database Index

The create() **function can be used alongside** index() **to create a database index.**

```
...
create table(:comments) do
  add :body, :text, null: false
  add :author, :text
  add :video_id, references(:videos, on_delete: :delete_all)

  ...
end

create index(:comments, [:video_id])
```

table name

column names **as a list**

Running a Migration

We use the *mix* task ecto.migrate to run migrations and issue changes to the database.

```
>>> mix ecto.migrate
```

```
12:20:24.602 [info]  == Running FireStarter.Repo.Migrations.AddCommentsTable.change/0 forward

12:20:24.602 [info]  create table comments

12:20:24.640 [info] == Migrated in 0.0s
```

Success! 👍

Level 4 - Section 2

Migrations & Associations

Showing *Comments* for a *Video*

On Fire WITH PHOENIX

The *Comments* Table

With the new table in place, we can now start reading *comments.*

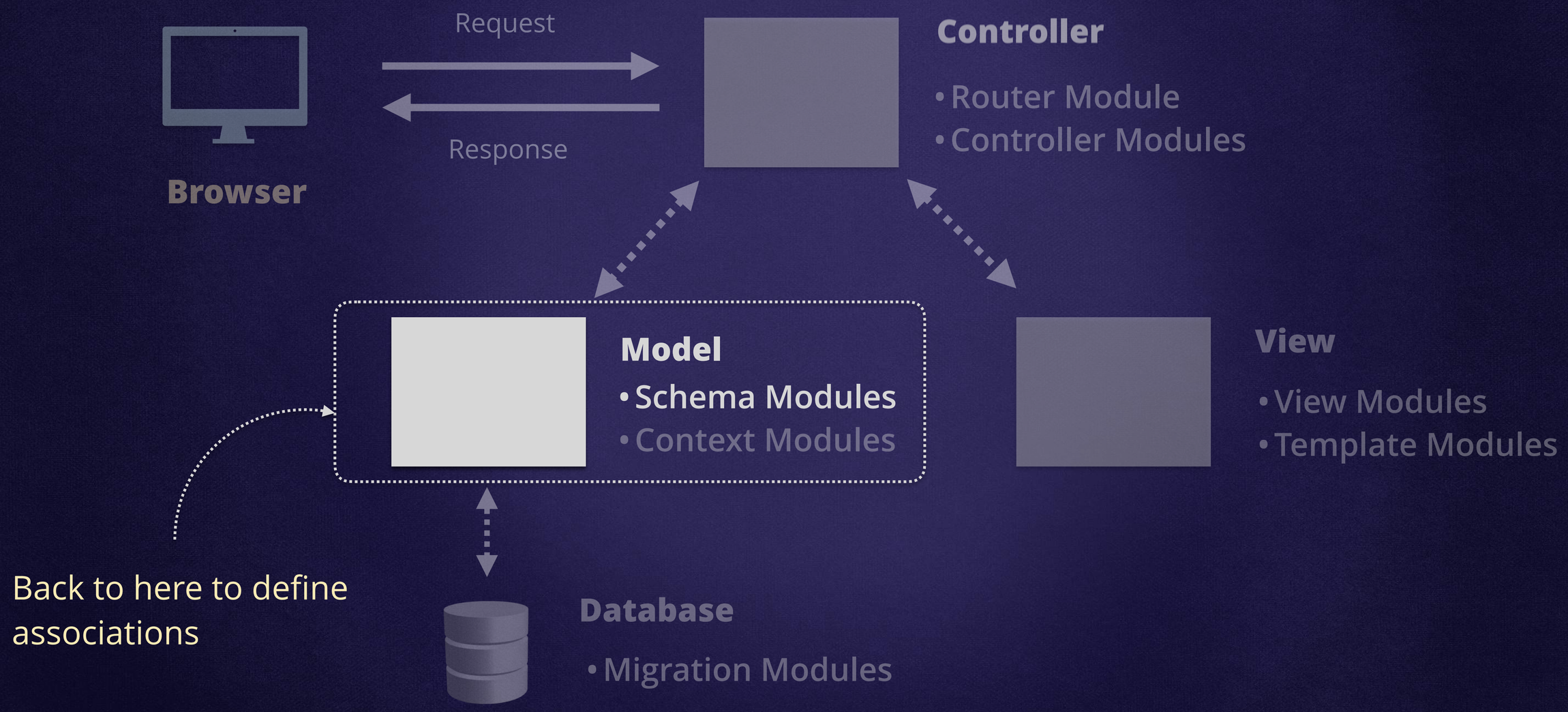
```
comments

 id |        body        |    author    | video_id
--+--------------------+--------------+----------
  1 | Very helpful!      |    Brooke    |    42
  2 | Great video.       |    Sam       |    42
```

```
videos
```

Let's learn how to write *Phoenix* code to read comments that **belong to Videos.**

Schemas Define Associations

We need to tell *Ecto* how our **Schema Modules** are associated with one another.



Adding a has_many association

The has_many function indicates a **one-to-many association** with another schema.

```
defmodule FireStarter.Video do
  use Ecto.Schema

  schema "videos" do
    field :title, :string
    field :url, :string
    field :duration, :integer

    has_many :comments, FireStarter.Comment

    timestamps()
  end
  ...
end
```

the associated module

the property name

"A video has many comments!"

Adding a belongs_to **Association**

The belongs_to **function indicates a one-to-one association between parent and child.**

```
defmodule FireStarter.Comment do
  use Ecto.Schema

  schema "comments" do
    field :body, :string
    field :author, :string

    belongs_to :video, FireStarter.Video

    timestamps()
  end
end
```

the associated module

the property name

"A comment belongs to a video!"

The *Comments* and *Videos* Tables

This is the data that currently resides in our tables.

```
comments

 id |       body        |    author    | video_id
--+-------------------+-------------+----------
  1 | Very helpful!     |    Brooke    |    42
  2 | Great video.      |    Sam       |    42
```

```
videos

id |       title      |          url           | duration
---+-----------------+-----------------------+----------
 42 | Elixir          | example.com/elixir    |     1230
 43 | JavaScript      | example.com/js        |      790
```

Reading *Comments* for a *Video*

Using Repo.get() **does NOT automatically load associations.**

```
video = Repo.get(Video, 42)
video.comments
```

Gets *video*, but **NOT** its *comments.*

```
#Ecto.Association.NotLoaded
<association :comments is not loaded>
```

Preloading Associations

The Repo.preload() **function returns a struct with its associations preloaded.**

```
video = Repo.get(Video, 42) |> Repo.preload(:comments)
video.comments
```

```
[%FireStarter.Comment{..., video_id: 42},
 %FireStarter.Comment{..., video_id: 42}]
```

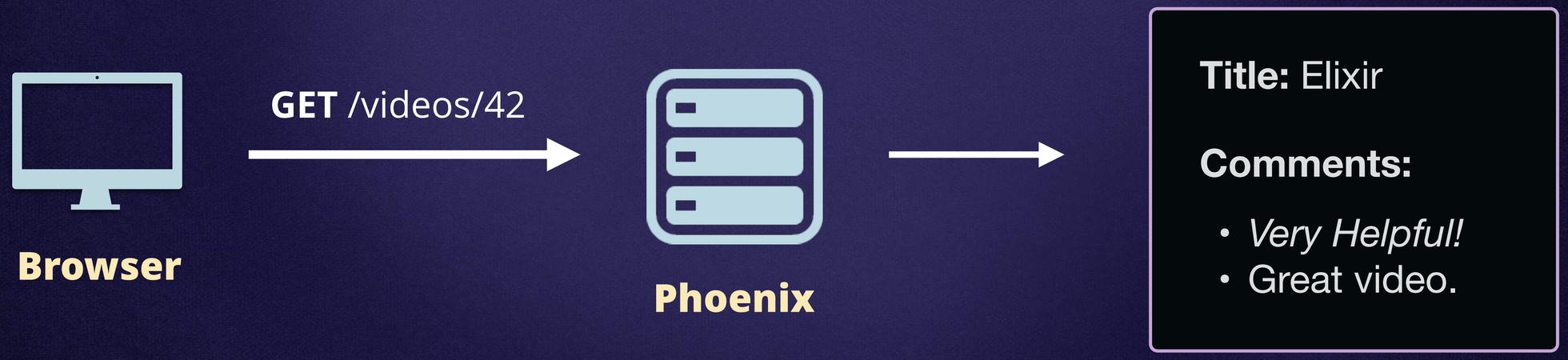
Gets *video* **AND** *comments*

all comments belong to same video

Building the *Video* page

To finish building the *Video* page we need three things:

1. Add a **new route** for the video page.
2. Fetch the video and preload comments.
3. Render the HTML with the video title and list of comments



The Route for a *Video*

The new route will match GET requests to "/videos/" followed by a value.

(spoiler alert: the value is the **id** for the video)

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
    get "/videos/:id", VideoController, :show
  end
end
```

calls the `show()` function on `VideoController`

The *VideoController* :show **Action**

On the show() function, we use **pattern matching*** to read the **id** from the path.

lib/fire_starter_web/controllers/video_controller.ex

```
defmodule FireStarterWeb.VideoController do
  ...

  def show(conn, %{"id" => id}) do
    video = Repo.get(Video, id) |> Repo.preload(:comments)
    render conn, "show.html", video: video
  end
end
```

the value from `/videos/:id`

* Using pattern matching to read values passed by the router is a widely used practice in *Phoenix*

The *Video* show Template

On the *show* template, we can read the *comments* property from *@video*

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

```
<h2><%= @video.title %></h2>
<ul>
  <%= for comment <- @video.comments do %>
    <li><%= comment.body %></li>
  <% end %>
</ul>
```

using *list comprehension* to loop through comments

The *Video* page

The video show page is now complete!