Level 5 - Section 1

Using *Contexts*

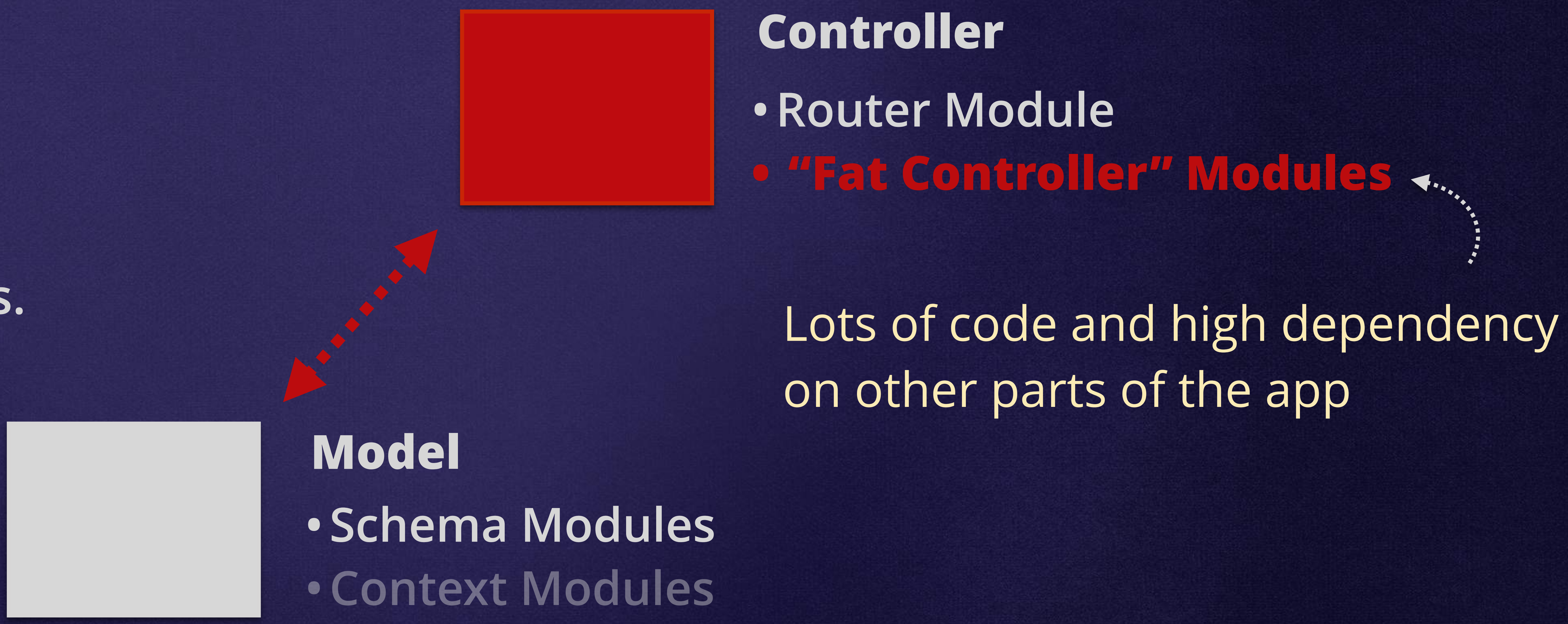
Reading Videos

Tight Coupling Leads to Bad Code

When parts of the app **know too much** about other parts, it's called **tight coupling.**

Examples of tight coupling in *Phoenix*:

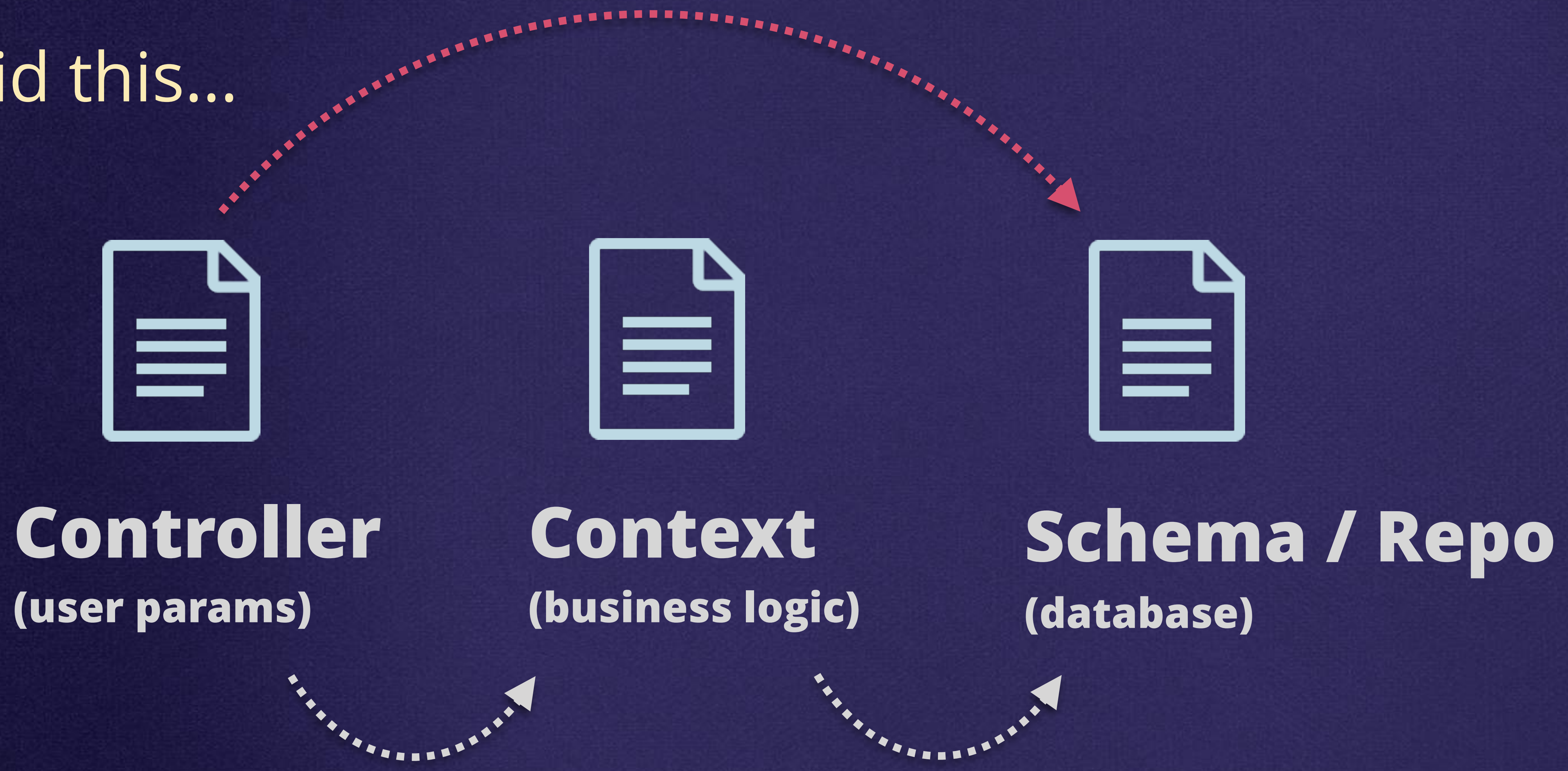
1. Too much code in *Controllers*, known as "Fat Controllers".
2. References to *Repo* and *Schema* functions from *Controller* actions.


Controller
• Router Module
• "Fat Controller" Modules
Lots of code and high dependency on other parts of the app
Model
• Schema Modules
• Context Modules

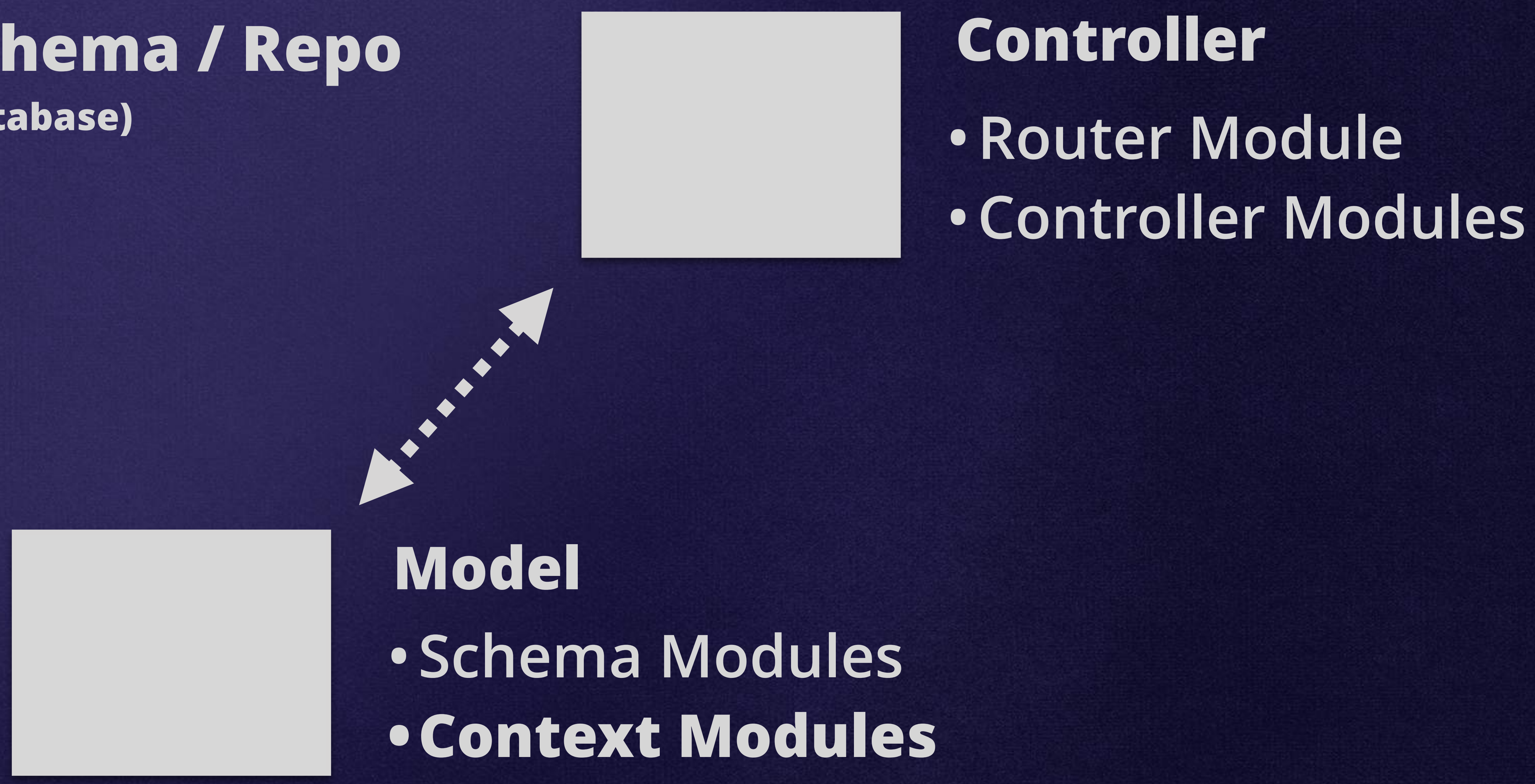
Controllers Should *Talk to Contexts*

Context modules allows us to **decouple and isolate** our code into manageable and independent parts.

We should avoid this…

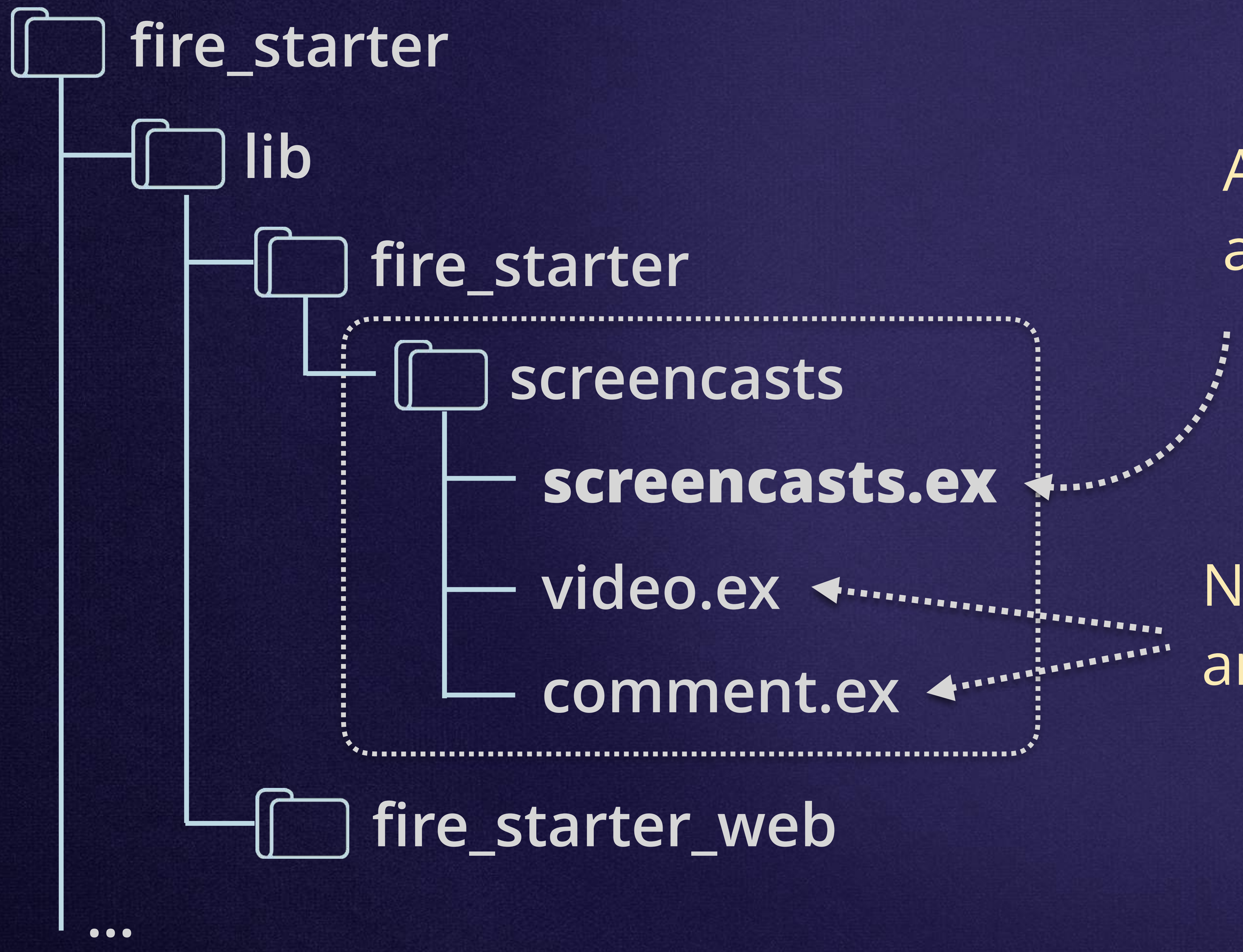


…and instead favor this



Moving *Video* **and** *Comment* **Inside** *Screencasts*

The *Screencasts* module will be the entry point for all video-related operations.



A *Context* is simply an **Elixir module**

Now part of the **screencasts** folder and *Screencasts* namespace

Tight Coupling When Listing Videos

Currently, any changes to reading videos from the database will **directly affect** this code.

lib/fire_starter_web/controllers/video_controller.ex

```
defmodule FireStarterWeb.VideoController do
  ...
  def index(conn, _) do
    videos = Repo.all(Video)
    render conn, "index.html", videos: videos
  end

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

Too much knowledge about other parts of the app...

...and references to *Repo.*

Moving Calls to *Repo* to the *Context*

Reading videos from the database is now **decoupled and isolated** from the *Controller*.

lib/fire_starter/**screencasts/screencasts.ex**

```
defmodule FireStarter.Screencasts do


  def list_videos do
    Repo.all(Video)
  end

  def get_video(id) do
    Repo.get(Video, id) |> Repo.preload(:comments)
  end
end
```

A module part of the ***FireStarter*** namespace

Move code here from `VideoController`

Moving Aliases from *Controller* to *Context*

The necessary calls to *alias* must also be moved to *Screencasts* module.

lib/fire_starter/**screencasts/screencasts.ex**

```elixir
defmodule FireStarter.Screencasts do
  alias FireStarter.Repo
  alias FireStarter.Screencasts.Video

  def list_videos do
    Repo.all(Video)
  end

  def get_video(id) do
    Repo.get(Video, id) |> Repo.preload(:comments)
  end
end
```

Needed for shorter references to *Repo* and *Video*

Calling a *Context* from the *Controller*

The code for reading videos is now shorter and **decoupled** from *Repo* and *Schema.*

lib/fire_starter_web/controllers/video_controller.ex

```
defmodule FireStarterWeb.VideoController do
  use FireStarterWeb, :controller
  alias FireStarter.Screencasts
  def index(conn, _) do
    videos = Screencasts.list_videos()
    render conn, "index.html", videos: videos
  end

  def show(conn, %{"id" => id}) do
    video = Screencasts.get_video(id)
    render conn, "show.html", video: video
  end
end
```

Level 5 - Section 2

Using *Contexts*

Creating Videos

Tight Coupling for New Forms

The *Controller* is **tightly coupled** to the *Ecto* library for generating new video forms.

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
```

Controller needs to know about *Ecto*...

...in order to create a changeset

Moving changeset code to *Schema*

We'll slightly change the code for a **changeset** and move it inside the *Video* ***Schema.***

part of the *Screencasts* namespace

lib/fire_starter_web/controllers/video_controller.ex

```
defmodule FireStarter.Screencasts.Video do
  ...

  def changeset(%Video{} = video, attrs) do
    video
    |> cast(attrs, [:title, :duration])
  end
end
```

By using *cast* instead of *change,* we can later re-use the `changeset` function when creating a new *Video*

Moving alias **and** import

We need these two lines: one to invoke cast() **and the other one to reference** %Video{}.

```
defmodule FireStarter.Screencasts.Video do

  import Ecto.Changeset
  alias FireStarter.Screencasts.Video

  def changeset(%Video{} = video, attrs) do
    video
    |> cast(attrs, [:title, :duration])
  end
end
```

Creating a changeset from the *Context*

The *change_video* function from *Screencasts* is now in charge of **creating a changeset.**

```
defmodule FireStarter.Screencasts do

  ...

  def change_video(%Video{} = video) do
    Video.changeset(video, %{})
  end
end
```

This function will be called from the *VideoController*

Controller calls *Context* for changeset

The *Controller* now calls a function from *Screencasts* in order to create a changeset.

```
defmodule FireStarterWeb.VideoController do
  ...

  def new(conn, _) do
    changeset = Screencasts.change_video(%Video{})
    render conn, "new.html", changeset: changeset
  end

end
```

It's fine to use the *Schema* as argument here

No longer relies on *Ecto* for creating a changeset!

Tight Coupling for Creating New Videos

The *VideoController* is **tightly coupled** with *Ecto* and *Repo* for creating new videos.

```
defmodule FireStarterWeb.VideoController do
  ...
  import Ecto.Changeset
  alias FireStarter.Repo

  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
```

!

Controller needs to know about *Ecto* and *Repo*.

Too many details about creating video are exposed.

Moving Creation Code to *Context*

The new Screencasts.create_video function **encapsulates the logic** for creating a new video.

```
defmodule FireStarter.Screencasts do

  ...

  def create_video(attrs \\ %{}) do
    %Video{}
    |> Video.changeset(attrs)
    |> Repo.insert()
  end
end
```

User submitted attributes

Uses the new function we've created in *Video*.

It's ok to call *Repo* from the *Context*

Using *Context* to Create New Videos

The code for creating videos is now shorter and **decoupled** from *Repo* and *Schema*.

```elixir
defmodule FireStarterWeb.VideoController do
  ...

  def create(conn, %{"video" => video_params}) do
    case Screencasts.create_video(video_params) do
      {:ok, _} -> ...
      {:error, changeset} -> ...
    end
  end
end
```