
On Fire
WITH
PHOENIX

Level 1 - Section 1

Sparks of Data

Basics of Phoenix and Working with a Database

What Is Phoenix ?

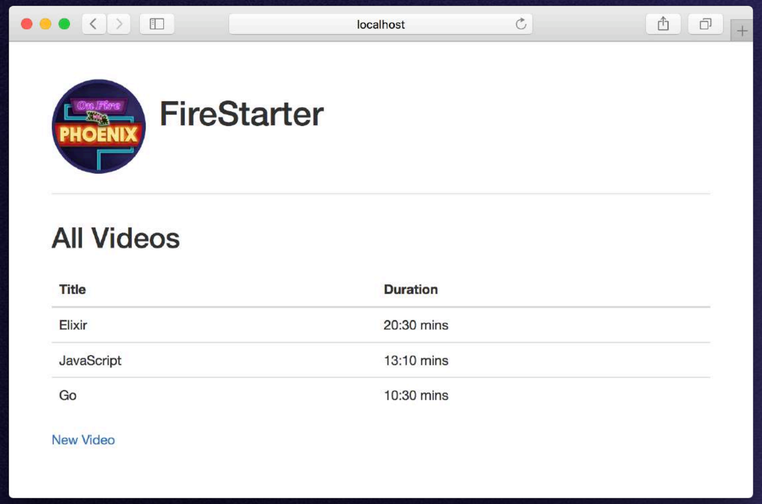
It’s a web development framework written in *Elixir*.

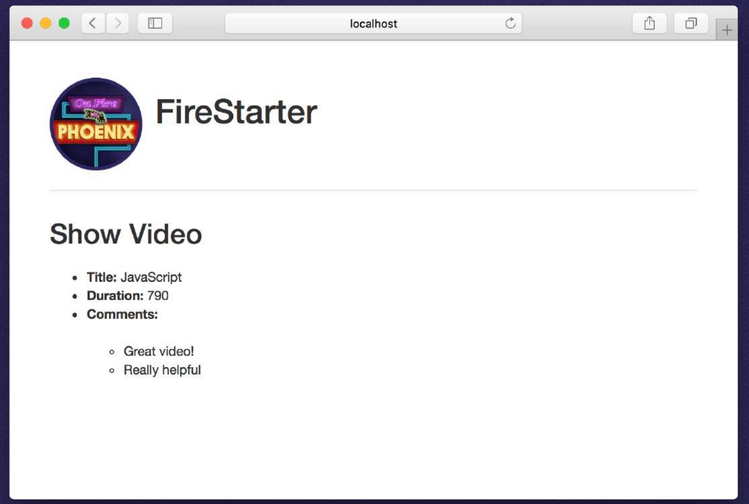
- **Model View Controller (MVC)** - Intuitive structure for code files.
- **Developer Productivity** - Leverages existing *Elixir* and *Erlang* conventions.
- **High Performance** - Runs on the blazing fast Erlang Virtual Machine.
- **Batteries Included** - Full stack, backend code, database access, JavaScript.

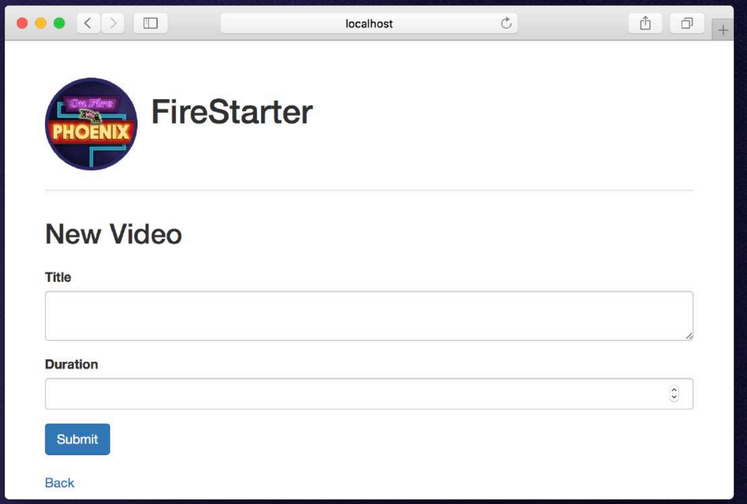
Before proceeding, make sure you know *Elixir* and *SQL*:

What We Will Learn in This Course

In this course, we'll write features for a *Phoenix* web app for viewing videos called ***FireStarter.***





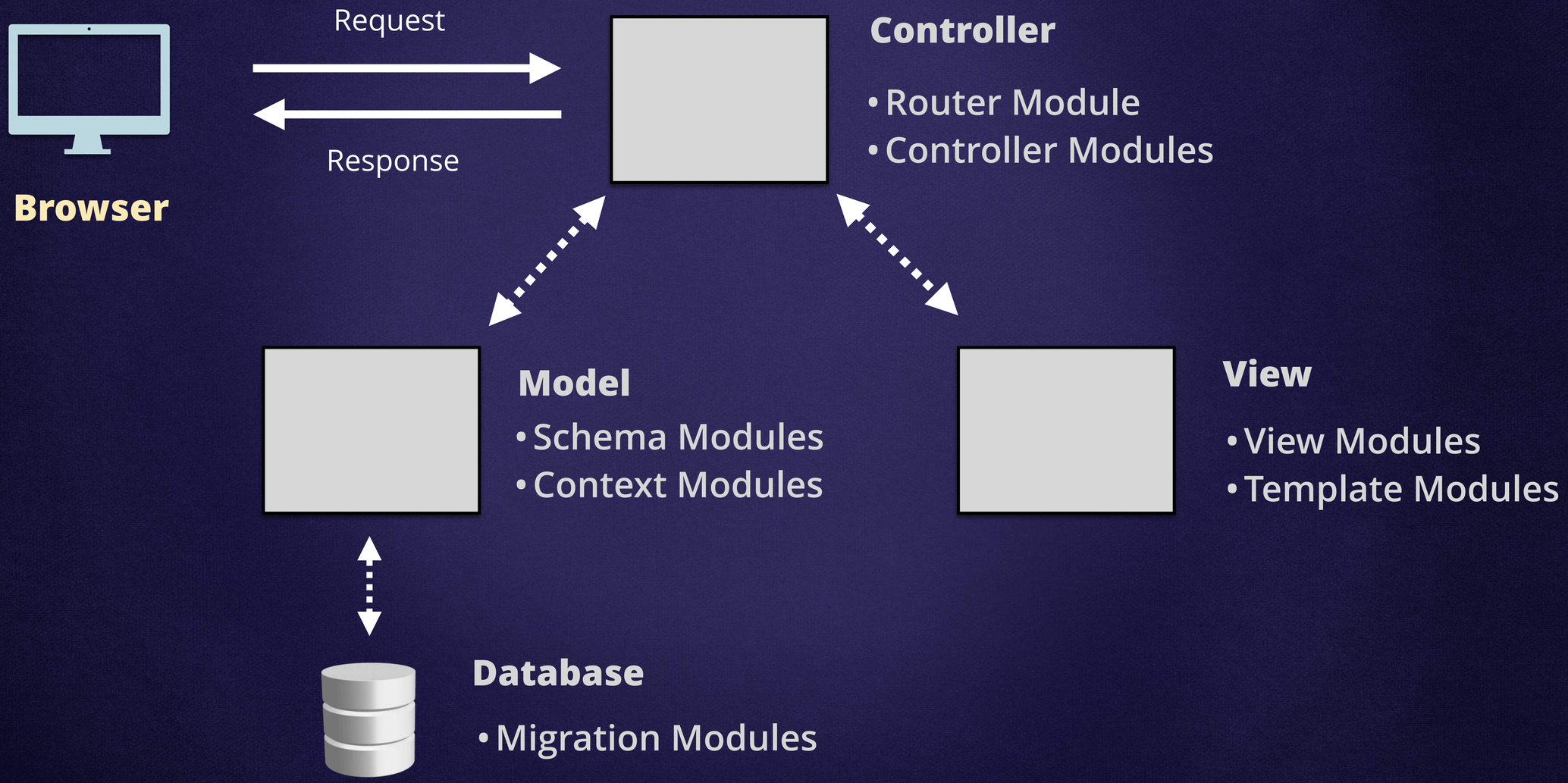


Some of the things we'll learn include:

- Using the **Ecto** library to work with a database.
- Creating new **HTTP routes.**
- Working with **forms.**
- **Validating** user input.

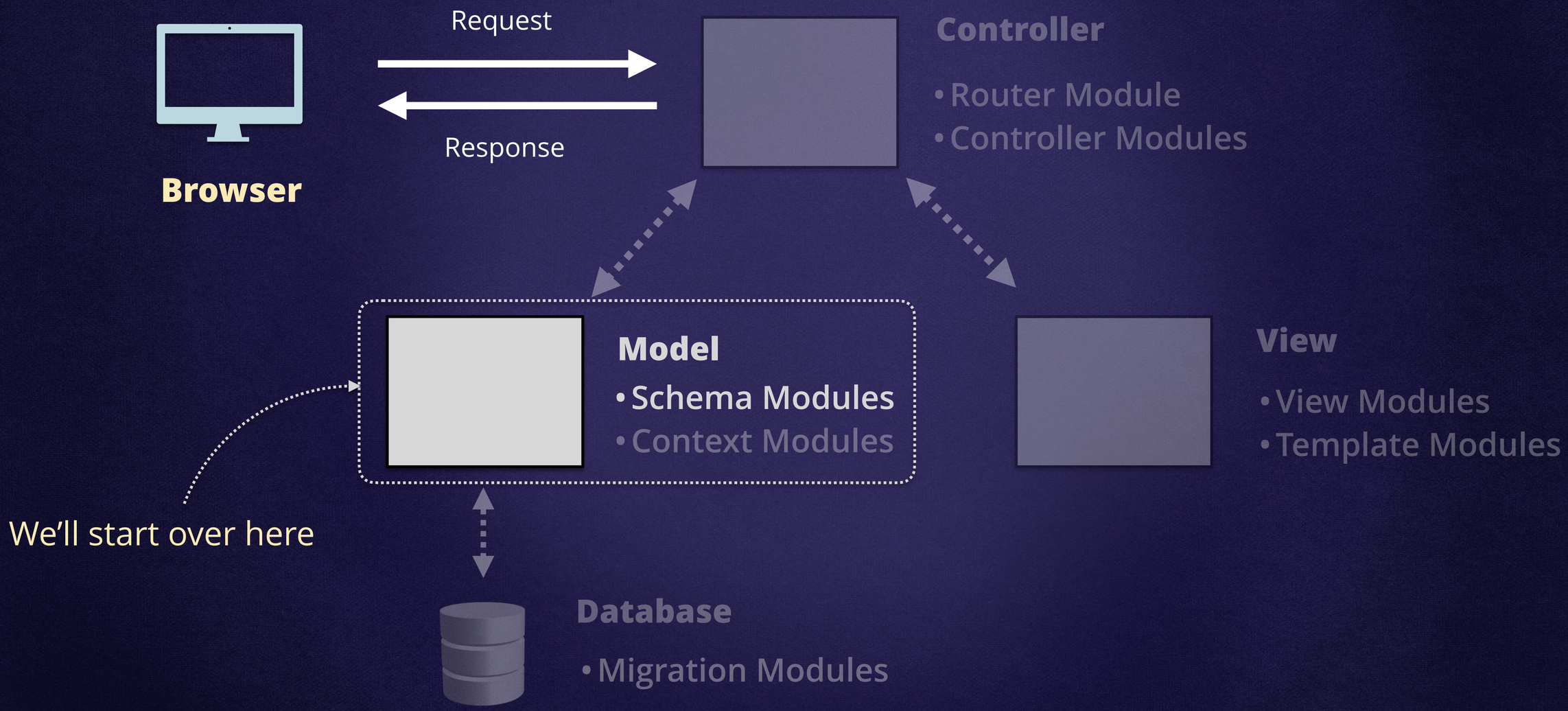
MVC in Phoenix

This is how **Model View Controller** (MVC) is represented in *Phoenix*.



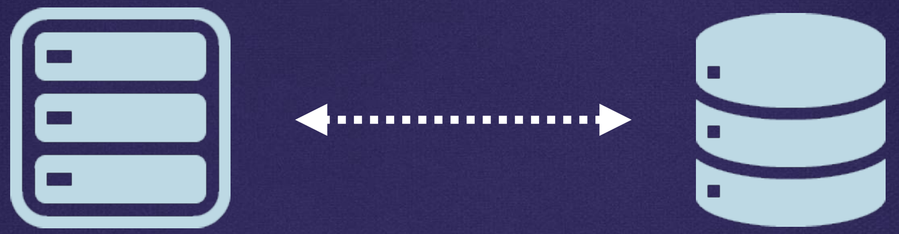
MVC in Phoenix

This is how **Model View Controller** (MVC) is represented in *Phoenix*.



Listing Data

The first thing we'll learn in *Phoenix* is how to read data from a database table.



Phoenix

Database

Defaults to `PostgreSQL` but others are supported.

In this section we'll learn how to:

1. Map *Elixir* code to a database table.
2. List all records from a database table.

The Videos Table

Here's a database table called videos **with 4 columns (**id, title, url and duration**) and two records.**

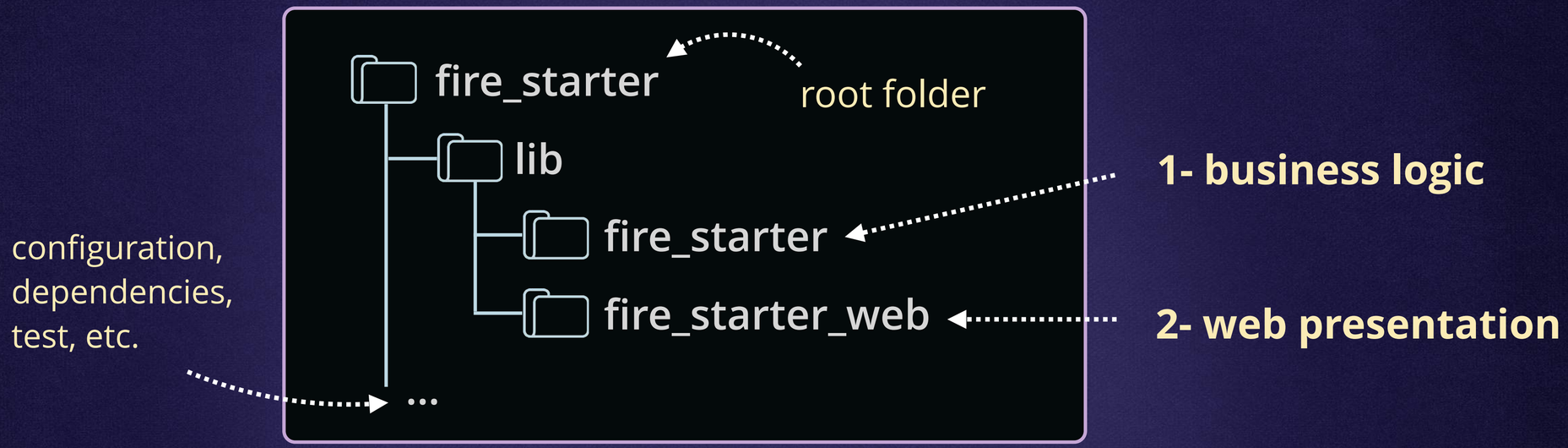
```
videos

 id |       title        |            url             | duration
----+--------------------+----------------------------+----------
  1 | Elixir             | example.com/elixir         |     1230
  2 | JavaScript         | example.com/javascript     |      790
```

text

integer

The Folder Structure

In order to promote **better design**, *Phoenix* organizes our source code in two main folders:



1) The `lib/<name_of_our_app>` folder

The place for **core business logic** of our application, i.e.:

a) *Calculating sales tax in a shopping cart*
b) *Max amount of users in a chat room*

2) The `lib/<name_of_our_app>_web`

The place for **web presentation** logic of our application, i.e.:

a) *Max number of records per page*
b) *Error messages on form submissions*

An *Ecto* Schema

Schema modules are responsible for **mapping** data sources (usually databases) to Elixir code.

lib/fire_starter/video.ex

```
defmodule FireStarter.Video do


end
```

The top-level module **must** be named after our app name, *FireStarter*.

The module name can be anything, but it's common to use the singular version of the **table name.**

The schema() function

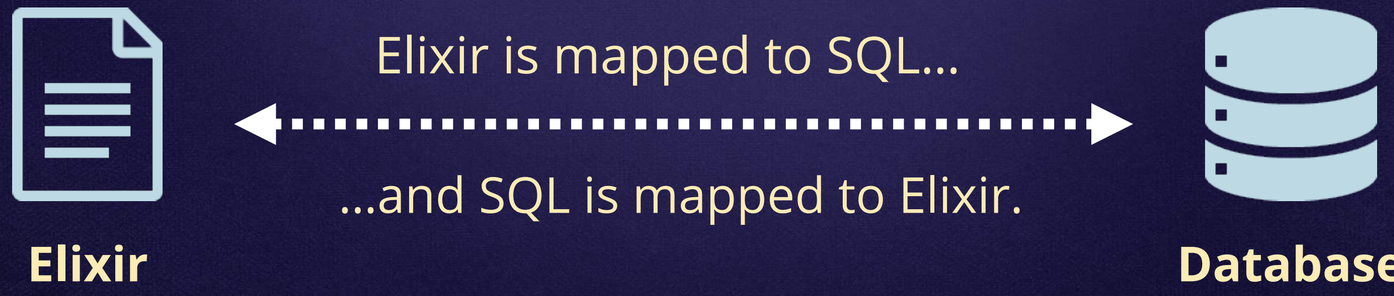
The schema() **function is available from** Ecto.Schema **and maps tables to *Elixir* modules.**

lib/fire_starter/video.ex

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

  schema "videos" do

  end
end
```

`use` allows us to call functions from `Ecto.Schema` as if they were part of `FireStarter.Video`

The `schema` function takes the name of the database table as its argument.



Mapping To Columns

The field() **function takes the name of a database column followed by an *Elixir* data type.**

lib/fire_starter/video.ex

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

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


  end
end
```

The `field` function takes the name of the column, followed by its data type

The `id` column is automatically inferred as the **primary key** for each table

id	title	url	duration
1	Elixir	example.com/elixir	1230
2	JavaScript	example.com/javascript	790

Tracking Creation and Last Update

The timestamps() **function maps to columns that help keep track of when a record was initially inserted and when it was last updated.**

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

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

    timestamps()
  end
end
```

same thing as...

```elixir
field :inserted_at, :naive_datetime
field :updated_at, :naive_datetime
```

Columns populated by *Ecto* when:

1. a record is **inserted**
2. a record is **updated**

```
 ... |     inserted_at      |     updated_at
-----+-----------------------+---------------------
 ... | 2017-05-25 20:27:19   | 2017-05-25 20:27:19
 ... | 2017-05-25 20:27:19   | 2017-05-29 18:13:05
```

automatically populated

Fetching All Records

All communication with the database is done through the FireStarter.Repo **module.**

`FireStarter.Repo`

wrapper around the data store

Database

The `all()` **function takes a** *Schema* **as argument and returns all records in the corresponding table.**

```
FireStarter.Repo.all(FireStarter.Video)
```

SQL: `SELECT * FROM "videos"`

Using Alias

The alias **directive lets us refer to** FireStarter.Repo **and** FireStarter.Video **as** Repo **and** Video**.**

```
alias FireStarter.Repo
alias FireStarter.Video
```

creates new alias named after
the last part of the module name

The code that used to be long...

```
FireStarter.Repo.all(FireStarter.Video)
```

same result

...can now be written like this!

```
Repo.all(Video)
```

Level 1 - Section 2

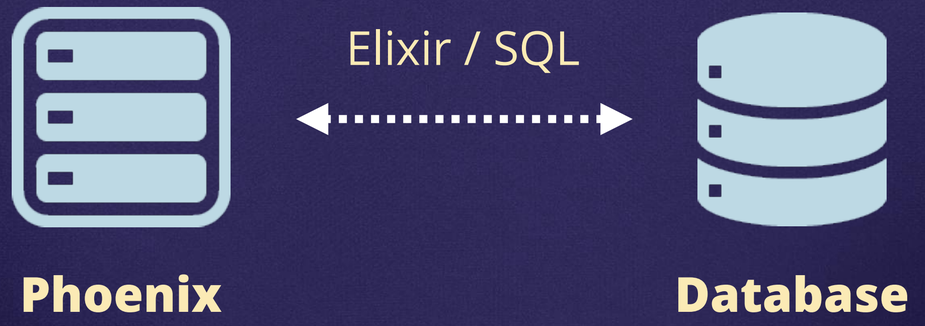
Sparks of Data

Reading Data With Conditions

More Ways to Read Data

We know how to fetch all videos from the database. In this section, we'll learn how to:

- Fetch a single video by its id
- Filter videos based on a condition.



Fetch a Video by id

The get() function takes a **Schema** and an **integer** as argument, and returns a single record.

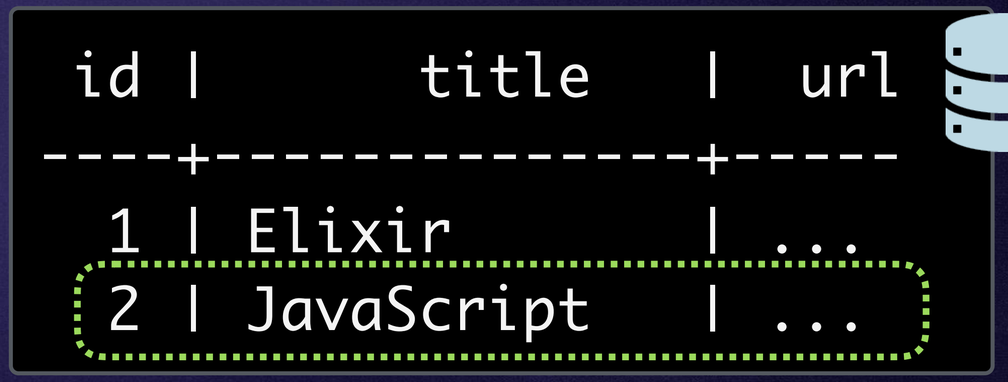
```
Repo.get(Video, 2)
```

```
%FireStarter.Video{id: 2, title: "JavaScript",
duration: 790, url: "example.com/javascript"}
```

Returns a video *Struct*

SQL:

```
SELECT * FROM "videos"
WHERE id = 2
```



A Schema Is a *Struct*

Structs are data types built on top of *Maps* and provide **compile-time checks.**

A *Map*...

...sees `tilte` *(not* `title`*)* and assigns it a value anyway

```
video = %{tilte: "Elixir"}
video.title
```

```
** (KeyError) key :title not found
```

A *Struct*...

...checks that `tilte` was **NOT** defined for `Video` and immediately **raises error.**

```
%Video{tilte: "Elixir"}
```

```
** (KeyError) key :tilte not found in:
%FireStarter.Video{id: nil, duration: nil,
inserted_at: nil, title: nil, updated_at: nil,
url: nil}
```

The keys allowed are the ones defined using the `field()` function in the `Video` **Schema Module.**

get() **vs.** get!()

The get() **function returns** nil **when no record exists; the** get!() **version raises an error.**

When no record is found...

```
Repo.get(Video, 3)
```

➔ `nil` ...no error is raised

```
 id |       title    |  url
----+----------------+-----
  1 | Elixir         | ...
  2 | JavaScript     | ...
```

When no record is found...

```
Repo.get!(Video, 3)
```

➔ `** (Ecto.NoResultsError)` ...an error is raised

Filtering Videos based on a Condition

We want all videos where duration is **LESS THAN** 800 seconds.

```
 id |       title      | url  | duration
----+------------------+------+----------
  1 | Elixir           | ...  |     1230
  2 | JavaScript       | ...  |      790
  3 | Go               | ...  |      630
```

SQL: `SELECT * FROM "videos" WHERE duration < 800`

Using Ecto.Query

The Ecto.Query **module provides a Domain Specific Language** (DSL) for querying data.

```elixir
defmodule FireStarter.Video do
  use Ecto.Schema
  import Ecto.Query

  schema "videos" do
    ...
  end

  def short_duration do
    from v in __MODULE__, where: v.duration < 800
  end
end
```

references enclosing module

Generated SQL

SQL: `SELECT * FROM "videos" WHERE duration < 800`

Running a Query

The all() **function can also take an** Ecto.Query **and it will return a filtered list of records.**

```
defmodule FireStarter.Video do
  ...
  def short_duration do
    from v in __MODULE__, where: v.duration < 800
  end
end
```

1. **builds** a Query...

```
Video.short_duration |> Repo.all
```

2. ...**executes** the Query.

```
[%FireStarter.Video{id: 2,
  duration: 790, title: "JavaScript", ...},
 %FireStarter.Video{id: 3,
  duration: 630, title: "Go", ...}]
```

Returns a *list* of video *Structs*

alias **vs.** use **vs.** import

alias **helps setup aliases for modules so we can refer to them using shorter names.**

```
alias FireStarter.Repo
alias FireStarter.Video
```

import **allows easy access to functions from other modules without using the fully-qualified name.**

```
import Ecto.Query

def short_duration do
  from v in __MODULE__,
       where: v.duration < 800
end
```

Imported from here

use **is similar to** import**, but gives module authors more control over what is imported and allows for "injecting" code** (metaprogramming)

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

  schema "videos" do
    ...
  end
end
```