LEARNING phoenix-framework

#phoenix-framework

Table of Contents

About

Chapter 1: Getting started with phoenix-framework

Remarks

This section provides an overview of what phoenix-framework is, and why a developer might want to use it.

It should also mention any large subjects within phoenix-framework, and link out to the related topics. Since the Documentation for phoenix-framework is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
0.1.1	2014-05-01
0.2.0	2014-05-01
0.2.1	2014-05-01
0.2.2	2014-06-05
0.2.3	2014-05-05
0.2.10	2014-05-22
0.2.11	2014-06-30
0.3.0	2014-07-01
0.3.1	2014-07-05
0.4.0	2014-08-31
0.4.1	2014-09-09
0.5.0	2014-10-14
0.6.0	2014-11-22
0.6.1	2014-11-30
0.6.2	2014-12-08
0.7.0	2014-12-10

Version	Release Date
0.7.1	2014-12-10
0.7.2	2014-12-11
0.8.0	2015-01-11
0.9.0	2015-02-12
0.10.0	2015-03-08
0.11.0	2015-04-08
0.12.0	2015-05-01
0.13.0	2015-11-15
0.13.1	2015-05-17
0.14.0	2015-06-30
0.15.0	2015-07-27
0.16.0	2015-08-06
0.16.1	2015-08-06
0.17.1	2015-08-27
1.0.0	2015-08-28
1.0.1	2015-09-03
1.0.2	2015-09-07
1.0.3	2015-09-29
1.0.4	2015-12-15
1.1.0	2015-09-16
1.1.1	2015-09-27
1.1.2	2016-01-09
1.1.3	2016-01-20
v1.2.0-rc.0	2016-04-29
v1.2.0-rc.1	2016-05-25

Version	Release Date
1.2.0	2016-06-23
1.2.2	2017-03-14
1.2.3	2017-03-15
1.2.4	2017-05-16
1.3.0-rc.1	2017-03-15
1.3.0-rc.2	2017-05-16

Examples

Installation

Phoenix framework is written in Elixir, and Elixir itself is based on Erlang language and leverages the Erlang VM, known for running low-latency, distributed and fault-tolerant systems. Both languages are required for using phoenix framework. Following following step to install phoenix framework:

1. Install Elixir on your machine. See Elixir Installation and how to install Elixir guide.

2. Install Hex package manager. Hex is a necessary tool to get a Phoenix app running and to install any extra dependencies we might need along the way. From your terminal or command control window, type:

```
$ mix local.hex
```

This command will install or update Hex, if you already have.

3. Install Erlang on your machine. Without Erlang, Elixir code will not compile because Elixir use Erlang's VM for code compilation. When you will install Elixir, you have probably installed Erlang too, but if it is not the case then follow these instruction on Elixir guide to install Erlang. However, If you are have Debian-based system, you may need to explicitly install Erlang.

```
$ wget https://packages.erlang-solutions.com/erlang-solutions_1.0_all.deb && sudo dpkg -i
erlang-solutions_1.0_all.deb
$ sudo apt-get update
$ sudo apt-get install esl-erlang
```

4. Install phoenix framework on your machine. Once we have Elixir and Erlang, we are ready to install the Phoenix Mix archive. A Mix archive is a Zip file which contains an application as well as its compiled BEAM files. It is tied to a specific version of the application. The archive is what we will use to generate a new, base Phoenix application which we can build from. Here's the command to install the Phoenix archive:

```
$ mix archive.install https://github.com/phoenixframework/archives/raw/master/phoenix_new.ez
```

Your can download packages manually, if above command doesn't work properly for you. Download packages to your file system Phoenix archives and run following command

```
mix archive.install /path/to/local/phoenix_new.ez
```

5 Plug, Cowboy, and Ecto are components of Phoenix framework, they will be installed automatically by mix, if you let mix install its dependencies, when you will first create Phoenix projects. Furthermore, if you don't allow mix to download these components then mix will tell you how how to do so later.

6. Install Node.js (not less then v5.0.0) on your machine. This is an **optional** dependency. Node.js is required to install brunch.io dependencies. Brunch.io is used by Phoenix for compiling static assets (javascript, css, etc), by default.

We can get node.js from the download page. When selecting a package to download, it's important to note that Phoenix requires version 5.0.0 or greater.

Mac OS X users can also install node.js via homebrew.

Note: io.js, which is an npm compatible platform originally based on Node.js, is not known to work with Phoenix.

Debian/Ubuntu users might see an error that looks like this:

```
sh: 1: node: not found
npm WARN This failure might be due to the use of legacy binary "node"
```

This is due to Debian having conflicting binaries for node: see discussion on following SO question

Cannot install packages using node package manager in Ubuntu

There are two options to fix this problem, either:

install nodejs-legacy:

```
$ apt-get install nodejs-legacy
```

or create a symlink

```
$ ln -s /usr/bin/nodejs /usr/bin/node
```

7 Install Database (PostgreSQL) on your machine. Phoenix configures applications to use it by default, but we can switch to MySQL by passing the `--database mysql` flag when creating a new application. The PostgreSQL wiki has installation guides for a number of different systems.

Postgrex is a direct Phoenix dependency and it will be used to create models. **Postgrex** will be automatically installed along with the rest of dependencies when you will create and start Phoenix project.

8 inotify-tools (for linux users) This is a Linux-only filesystem watcher that Phoenix uses for live code reloading. (Mac OS X or Windows users can safely ignore it.)

Linux users need to install this dependency. Please consult the inotify-tools wiki for distribution-specific installation instructions.

Skeleton Installation

Sometimes you want an installation without anything except the bare minimum phoenix setup. The follow command will give you that.

```
mix phoenix.new web --no-brunch --no-ecto
```

Note: You must have installed Elixir, Erlang, Hex, Mix and the Phoenix archive for skeleton installation

Creating Phoenix project

For creating your first project in Phoenix framework at this point **you should have**, Elixir, Erlang, Hex, and the Phoenix archive installed. You should also have PostgreSQL and node.js installed to build a default application.

Open terminal or command prompt and go to location on your file system where you want to **create application**. `phoenix.new` is the mix command which will create new project for you. Assuming that the name of our application is `hello_phoenix_world`, then type

```
$ mix phoenix.new hello_phoenix_world
```

Alternately, We can run mix phoenix.new from any directory in order to bootstrap our Phoenix application. Phoenix will accept either an absolute or relative path for the directory of our new project

```
$ mix phoenix.new /Users/username/work/elixir-projects/hello_phoenix_world
```

Output

```
mix phoenix.new hello_phoenix_world
* creating hello_phoenix_world/config/config.exs
* creating hello_phoenix_world/config/dev.exs
* creating hello_phoenix_world/config/prod.exs
...
* creating hello_phoenix_world/web/views/layout_view.ex
* creating hello_phoenix_world/web/views/page_view.ex

Fetch and install dependencies? [Yn]
```

Phoenix will generate the directory structure for your project and it will create all the files required for application. Mix will ask you if you want it to **install other required dependencies**. Let's say yes to that.

```
Fetch and install dependencies? [Yn] Y
* running mix deps.get
* running npm install && node node_modules/brunch/bin/brunch build
```

Once **dependencies are installed**, the task will prompt you to change into our project directory and start application.

```
Move into your new project folder:

    $cd hello_phoenix_world
```

You now need to setup the postgres username and password unless its already setup with the default postgres useranme and postgres password. Edit your `config/dev.exs` file and set the username and password:

```
# config/dev.exs
config :hello_phoenix_world, HelloPhoenixWorld.Repo,
  adapter: Ecto.Adapters.Postgres,
  username: "postgres",
  password: "postgres",
  database: "hello_phoenix_world_dev",
  hostname: "localhost",
  pool_size: 10

Now, create the database with the ecto mix task:

    $ mix ecto.create

We have a working application! Run your Phoenix application:

    $ mix phoenix.server

You can also run your app inside IEx (Interactive Elixir) as:

    $ iex -S mix phoenix.server

Load `http://localhost:4000` into your browser and you will see the default landing page of
your application.
```

Now, lets add hello world to the Phoenix application. Open the `web/templates/page/index.html.eex` file and replace the contents with the following and save the file:

```
<h2>Hello World</h2>
```

If you have not quit the server, the new code will be automatically compiled and your browser should now display your "Hello World" message.

You can now create CRUD resource.

Finally, to exit out of the server, type `ctrl-c crtl-c` (press the `control key` and the `c` key together) twice in a row.

Running Elixir/Phoenix on OSX

Elixir / Phoenix

Install Homebrew first:

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Then running `brew install elixir` will install both Elixir and it's dependency - Erlang.

Install mix with `mix local.hex`.

Install Phoenix as per instructions:

```
mix archive.install https://github.com/phoenixframework/archives/raw/master/phoenix_new.ez
```

Node.js

You can install and manage your Node.js versions with NVM. Install nvm with:

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.4/install.sh | bash
```

If `curl` is not available, you can install it with `brew install curl`. Then run:

```
nvm install node
```

to download and compile and latest version of Node.js.

Database

Download Postgres.app and run it. When you create your Phoenix project, in your `config/dev.exs` file, you just need to supply a name for your database - the adapter will use default values for the rest:

```
config :myphoenixapp, MyPhoenixApp.Repo,
  adapter: Ecto.Adapters.Postgres,
  database: "myphoenixapp_dev",
  hostname: "localhost",
  pool_size: 10
```

Generating resources for a model

To generate schema, view, controller, migration file for the repository, default CRUD templates and test files for a model (like a scaffolding in Rails) one can use `phoenix.gen.html` mix task like this:

```
mix phoenix.gen.html Book books title note:text pages:integer author_id:references:authors
```

Where `Book` is the module name, `books` is plural form used for schema, followed by resource fields: `title` (string by default), `note` (text field), `pages` (integer), `author_id` which creates a `belongs_to` association with the Author model.

Read Getting started with phoenix-framework online: https://riptutorial.com/phoenix-framework/topic/4996/getting-started-with-phoenix-framework

Chapter 2: Ecto models usage in phoenix

Introduction

How to generate, edit and use ecto models in the phoenix frameworks.

Examples

Generate User model from command line

To generate `json` user model with `username`, `password_hash`, `email_id`, `created_at`, `updated_at`, type

```
mix phoenix.gen.json User users username:string email_id:string password_hash:string
timestamps()
```

Migrations of ecto model

When you run `mix phoenix.gen.html` or `mix phoenix.gen.json` from command line, migrations are created in `priv -> repo -> migrations` in your project folder.

To run migrations type `mix ecto.migrate`.

To generate migrations for your project `mix ecto.gen migrations <model_name>`

To generate migrations for a different repository than default one run `mix ecto.gen migrations <model_name> -r <repo_name>`

Read Ecto models usage in phoenix online: https://riptutorial.com/phoenix-framework/topic/10890/ecto-models-usage-in-phoenix

Chapter 3: Generate project documentation

Examples

Rationale

The correct invocation of helper modules and functions can be intimidating because

- these are generated dynamically (e.g., when creating a new project or adding a new `resource`)
- they are not documented explicitly (e.g., `MyApp.ErrorHelpers.error_tag`)
- the documentation does not cover all examples (e.g., `MyApp.Router.Helpers.*_path` in `Phoenix.Router`).

Although the created helpers are scattered all over your project but their location follows a solid logic. You can get used to them pretty quick and fortunately, when you generate a project with Phoenix, the code is shipped with documentation via Elixir's `@doc` and `@moduledoc` module attributes.

These docs are not limited to helpers only but you can also

- see your project broken down by submodules/functions/macros
- add your own documentation
- look up any functions that were generated under the namespace of your project (e.g., `MyApp.Repo` contains callback function implementations from `Ecto.Repo`)

Generating the docs

To generate documentation from your source code, add `ex_doc` as dependency into your `mix.exs` file:

```
# config/mix.exs

def deps do
  [{:ex_doc, "~> 0.11", only: :dev}]
end
```

> You can use Markdown within Elixir `@doc` and `@moduledoc` attributes.

Then, run `mix deps.get` to fetch and compile the new modules and generate the project documentation with `mix docs`. An example output is the official Elixir Docs.

To serve them immediately use `mix docs --output priv/static/doc` and navigate to `my_app_url_or_ip/doc/index.html`.

Additional reading:

- `ex_doc`
- Version requirement operators (`Elixir.Version`)

The bulk of this guide is referenced from Elixir Recipes.

Read Generate project documentation online: https://riptutorial.com/phoenix-framework/topic/5868/generate-project-documentation

Credits

S. No	Chapters	Contributors
1	Getting started with phoenix-framework	Community, helcim, penguin, Steve Pallen, SURAJ KUMAR, Svilen
2	Ecto models usage in phoenix	Faizan Ali
3	Generate project documentation	toraritte