eBook Gratuit

APPRENEZ phoenix-framework

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#phoenix-framework

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: phoenix-framework

It is an unofficial and free phoenix-framework ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official phoenix-framework.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec phoenix-framework

Remarques

Cette section fournit une vue d'ensemble de ce qu'est phoenix-framework et pourquoi un développeur peut vouloir l'utiliser.

Il convient également de mentionner tous les grands sujets au sein de Phoenix-Framework et de les relier aux sujets connexes. La documentation de phoenix-framework étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Versions

Version	Date de sortie
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	Date de sortie
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	2015-09-28
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	Date de sortie
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 est écrit en Elixir , et Elixir lui-même est basé sur le langage Erlang et exploite la machine virtuelle Erlang, connue pour exécuter des systèmes à faible latence, distribués et tolérants aux pannes. Les deux langues sont requises pour utiliser le framework Phoenix. Après l'étape suivante pour installer le framework phoenix:

1. Installez Elixir sur votre machine. Voir Installation Elixir et comment installer le guide Elixir .

2. Installez le gestionnaire de paquets **Hex** . Hex est un outil nécessaire pour faire fonctionner une application Phoenix et installer les dépendances supplémentaires dont nous pourrions avoir besoin tout au long du processus. À partir de votre fenêtre de contrôle de terminal ou de commande, tapez:

```
$ mix local.hex
```

Cette commande va installer ou mettre à jour Hex, si vous en avez déjà.

3. Installez Erlang sur votre machine. Sans Erlang, le code Elixir ne sera pas compilé car Elixir utilise la VM d'Erlang pour la compilation de code. Lorsque vous installerez Elixir, vous avez probablement installé Erlang également, mais si ce n'est pas le cas, suivez ces instructions sur le guide Elixir pour installer Erlang. Cependant, si vous utilisez un système basé sur Debian, vous devrez peut-être installer explicitement 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. Installez le framework Phoenix sur votre machine. Une fois que nous avons Elixir et Erlang, nous sommes prêts à installer l'archive Phoenix Mix. Une archive Mix est un fichier Zip contenant une application ainsi que ses fichiers BEAM compilés. Il est lié à une version spécifique de

l'application. L'archive est ce que nous allons utiliser pour générer une nouvelle application Phoenix de base à partir de laquelle nous pouvons construire. Voici la commande pour installer l'archive Phoenix:

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

Vous pouvez télécharger les paquets manuellement, si la commande ci-dessus ne fonctionne pas correctement pour vous. Téléchargez des packages vers vos archives de système de fichiers Phoenix et exécutez la commande suivante

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

5 Plug, Cowboy et Ecto sont des composants du framework Phoenix, ils seront installés automatiquement par mix, si vous laissez le mix installer ses dépendances, lorsque vous créerez d'abord des projets Phoenix. En outre, si vous ne permettez pas à Mix de télécharger ces composants, le mix vous indiquera comment procéder plus tard.

6. Installez Node.js (pas moins que la version 5.0.0) sur votre machine. Ceci est une dépendance **facultative** . Node.js est requis pour installer les dépendances brunch.io . Brunch.io est utilisé par Phoenix pour compiler des ressources statiques (javascript, css, etc.), par défaut.

Nous pouvons obtenir node.js de la page de téléchargement . Lors de la sélection d'un package à télécharger, il est important de noter que Phoenix requiert la version 5.0.0 ou supérieure.

Les utilisateurs de Mac OS X peuvent également installer node.js via homebrew .

Remarque: io.js, qui est une plate-forme compatible npm basée à l'origine sur Node.js, n'est pas connue pour fonctionner avec Phoenix.

Les utilisateurs Debian / Ubuntu peuvent voir une erreur qui ressemble à ceci:

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

Ceci est dû au fait que Debian a des binaires en conflit pour le noeud: voir la discussion sur la question SO suivante

Impossible d'installer les packages à l'aide du gestionnaire de packages de nœuds dans Ubuntu

Il existe deux options pour résoudre ce problème:

installer nodejs-legacy:

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

ou créer un lien symbolique

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

7 Installez la base de données (PostgreSQL) sur votre machine. Phoenix configure les applications pour l'utiliser par défaut, mais nous pouvons passer à MySQL en passant l' `--database mysql` lors de la création d'une nouvelle application. Le wiki PostgreSQL contient des guides d'installation pour différents systèmes.

Postgrex est une dépendance directe de Phoenix et sera utilisée pour créer des modèles. **Postgrex** sera automatiquement installé avec le reste des dépendances lorsque vous créerez et démarrerez un projet Phoenix.

8 inotify-tools (pour les utilisateurs linux) Il s'agit d'un observateur de système de fichiers Linux uniquement que Phoenix utilise pour le rechargement de code en direct. (Les utilisateurs de Mac OS X ou Windows peuvent l'ignorer en toute sécurité.)

Les utilisateurs de Linux doivent installer cette dépendance. Veuillez consulter le wiki inotify-tools pour obtenir des instructions d'installation spécifiques à la distribution.

Installation du squelette

Parfois, vous voulez une installation sans rien sauf la configuration minimale de Phoenix. La commande suivante vous donnera cela.

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

Remarque: Vous devez avoir installé Elixir, Erlang, Hex, Mix et l'archive Phoenix pour l'installation du squelette.

Création du projet Phoenix

Pour créer votre premier projet dans le framework Phoenix, **vous devez avoir** installé Elixir, Erlang, Hex et les archives Phoenix. PostgreSQL et node.js doivent également être installés pour créer une application par défaut.

Ouvrez le terminal ou l'invite de commande et accédez à l'emplacement sur votre système de fichiers où vous souhaitez **créer une application** . `phoenix.new` est la commande de mixage qui créera un nouveau projet pour vous. En supposant que le nom de notre application est `hello_phoenix_world` , tapez

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

Alternativement , nous pouvons exécuter mix phoenix.new à partir de n'importe quel répertoire afin d'amorcer notre application Phoenix. Phoenix acceptera un chemin absolu ou relatif pour le répertoire de notre nouveau projet

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

Sortie

```
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 générera la structure de répertoires de votre projet et créera tous les fichiers requis pour l'application. Mix vous demandera si vous souhaitez **installer d'autres dépendances requises** . Disons oui à ça.

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

Une fois les **dépendances installées** , la tâche vous demandera de changer dans notre répertoire de projet et de lancer l'application.

```
Move into your new project folder:

    $cd hello_phoenix_world
```

Vous devez maintenant configurer le nom d'utilisateur et le mot de passe postgres à moins que celui-ci ne soit déjà configuré avec le mot de passe postgres useranme et postgres par défaut. Modifiez votre fichier `config/dev.exs` et définissez le nom d'utilisateur et le mot de passe:

```
# 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.
```

Maintenant, permet d'ajouter hello world à l'application Phoenix. Ouvrez le fichier `web/templates/page/index.html.eex` et remplacez le contenu par le suivant et enregistrez le fichier:

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

Si vous n'avez pas quitté le serveur, le nouveau code sera automatiquement compilé et votre navigateur devrait maintenant afficher votre message "Hello World".

Vous pouvez maintenant créer une ressource CRUD .

Enfin, pour sortir du serveur, tapez `ctrl-c crtl-c` (appuyez `crtl-c` sur la `control key crtl-c` et la touche `c`).

Courir Elixir / Phoenix sur OSX

Elixir / Phoenix

Installez d'abord Homebrew :

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

Ensuite, le `brew install elixir` et installera Elixir et sa dépendance - Erlang.

Installez le mix avec `mix local.hex` .

Installez Phoenix selon les instructions:

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

Node.js

Vous pouvez installer et gérer vos versions Node.js avec NVM. Installez nvm avec:

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

Si `curl` n'est pas disponible, vous pouvez l'installer avec une `brew install curl` . Puis lancez:

```
nvm install node
```

pour télécharger et compiler et la dernière version de Node.js.

Base de données

Téléchargez Postgres.app et lancez-le. Lorsque vous créez votre projet Phoenix, dans votre fichier `config/dev.exs` , il vous suffit de fournir un nom à votre base de données - l'adaptateur utilisera les valeurs par défaut pour le reste:

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

```
  pool_size: 10
```

Générer des ressources pour un modèle

Pour générer un schéma, une vue, un contrôleur, un fichier de migration pour le référentiel, des modèles CRUD par défaut et des fichiers de test pour un modèle (comme un échafaudage dans Rails), vous pouvez utiliser la tâche de mixage `phoenix.gen.html` :

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

Où `Book` est le nom du module, `books` est la forme plurielle utilisée pour le schéma, suivie des champs de ressources: `title` (chaîne par défaut), `note` (champ de texte), `pages` (entier), `author_id` qui crée une association `belongs_to` avec le modèle Author.

Lire Démarrer avec phoenix-framework en ligne: https://riptutorial.com/fr/phoenix-framework/topic/4996/demarrer-avec-phoenix-framework

Chapitre 2: Générer la documentation du projet

Examples

Raisonnement

L'invocation correcte des modules d'aide et des fonctions peut être intimidante, car

- ceux-ci sont générés dynamiquement (par exemple, lors de la création d'un nouveau projet ou de l'ajout d'une nouvelle `resource`)
- ils ne sont pas documentés explicitement (par exemple, `MyApp.ErrorHelpers.error_tag`)
- la documentation ne couvre pas tous les exemples (par exemple, `MyApp.Router.Helpers.*_path` dans `Phoenix.Router`).

Bien que les assistants créés soient disséminés dans tout votre projet, leur emplacement suit une logique solide. Vous pouvez vous y habituer assez rapidement et heureusement, lorsque vous générez un projet avec Phoenix, le code est fourni avec la documentation via les attributs de module `@doc` et `@moduledoc` Elixir.

Ces documents ne sont pas limités aux assistants uniquement mais vous pouvez également

- voir votre projet décomposé par sous-modules / fonctions / macros
- ajoutez votre propre documentation
- rechercher les fonctions générées sous l'espace de noms de votre projet (par exemple, `MyApp.Repo` contient des implémentations de fonctions de rappel de `Ecto.Repo`)

Générer les documents

Pour générer la documentation à partir de votre code source, ajoutez `ex_doc` tant que dépendance dans votre fichier `mix.exs` :

```
# config/mix.exs

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

> Vous pouvez utiliser Markdown dans les `@doc` Elixir `@doc` et `@moduledoc` .

Ensuite, lancez `mix deps.get` pour `mix deps.get` et compiler les nouveaux modules et générer la documentation du projet avec `mix docs` . Un exemple de sortie est le document officiel Elixir .

Pour les servir immédiatement, utilisez `mix docs --output priv/static/doc` et accédez à `my_app_url_or_ip/doc/index.html` .

Lecture supplémentaire:

- `ex_doc`
- Opérateurs d'exigence de version (`Elixir.Version`)

L'essentiel de ce guide est référencé dans Elixir Recipes .

Lire Générer la documentation du projet en ligne: https://riptutorial.com/fr/phoenix-framework/topic/5868/generer-la-documentation-du-projet

Chapitre 3: Utilisation des modèles Ecto dans Phoenix

Introduction

Comment générer, éditer et utiliser des modèles ecto dans les frameworks phoenix.

Examples

Générer un modèle d'utilisateur à partir de la ligne de commande

Pour générer un modèle d'utilisateur `json` avec le `username` d' `username` , `password_hash` , `email_id` , `created_at` , `updated_at` , tapez

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

Migrations du modèle ecto

Lorsque vous exécutez `mix phoenix.gen.html` ou que vous `mix phoenix.gen.json` partir de la ligne de commande, les migrations sont créées dans `priv -> repo -> migrations` dans le dossier de votre projet.

Pour exécuter les migrations, tapez `mix ecto.migrate` .

Pour générer des migrations pour votre projet, `mix ecto.gen migrations <model_name>`

Pour générer des migrations pour un référentiel différent de celui utilisé par défaut, exécutez un `mix ecto.gen migrations <model_name> -r <repo_name>`

Lire Utilisation des modèles Ecto dans Phoenix en ligne: https://riptutorial.com/fr/phoenix-framework/topic/10890/utilisation-des-modeles-ecto-dans-phoenix

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec phoenix-framework	Community, helcim, penguin, Steve Pallen, SURAJ KUMAR, Svilen
2	Générer la documentation du projet	toraritte
3	Utilisation des modèles Ecto dans Phoenix	Faizan Ali