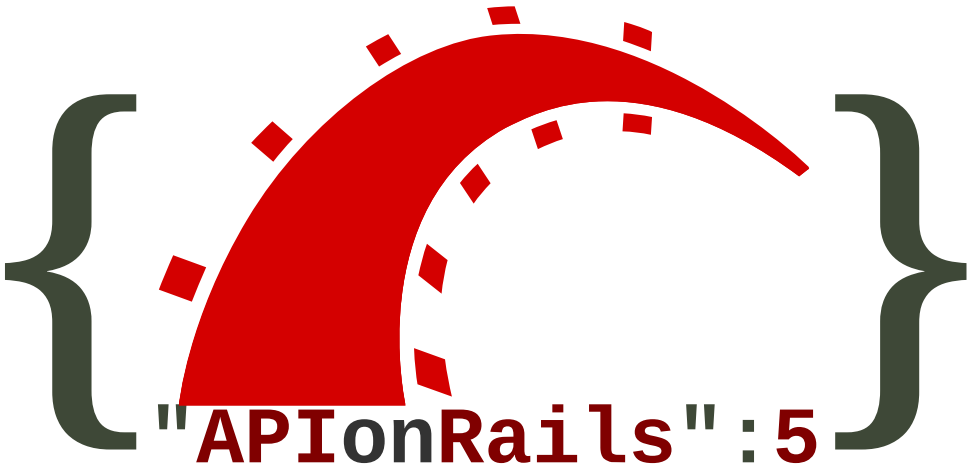
{"author": "Alexandre Rousseau"}



API on Rails 5

Alexandre Rousseau

Version 1.0, 2019-01-10

Table of Contents

Avant-propos

"API on Rails 5" est basé sur "APIs on Rails: Building REST APIs with Rails". Celui-ci fut initialement publié en 2014 par Abraham Kuri sous les licences MIT et Beerware.

L'œuvre originale étant non maintenue, j'ai voulu mettre à jour cet excellent ouvrage et contribuer à la communauté francophone en le traduisant moi-même. Cette mise à jour est aussi disponible dans la langue de Shakespeare [1: C'est-à-dire en anglais.].

A propos de l'auteur original

Abraham Kuri est un développeur Rails avec 5 ans d'expérience (sûrement plus maintenant). Son expérience inclut le travail en tant que *freelance* dans la construction de produits logiciels et plus récemment dans la collaboration au sein de la communauté open source. Diplômé en informatique d'ITESM, il a fondé deux sociétés au Mexique (Icalia Labs et Codeando Mexico).

De mon côté, je m'appelle Alexandre Rousseau et je suis un développeur Rails avec plus de 4 ans d'expérience (à l'heure où j'écris ces lignes). Je suis actuellement associé dans une entreprise (iSignif) ou je construis et maintiens un produit de type SAAS en utilisant Rails. Je contribue aussi à la communauté Ruby en produisant et maintenant quelques gemmes que vous pouvez consulter sur mon profil Rubygems.org. La plupart de mes projets sont sur Github donc n'hésitez pas à me suivre.

Tout le code source de ce livre est disponible au format Asciidoctor sur Github. Ainsi n'hésitez pas à forker le projet si vous voulez l'améliorer ou corriger une faute qui m'aurait échappée.

Droits d’auteur et licence

Cette traduction est disponible sous licence MIT. Tout le code source de ce livre est disponible au format Asciidoctor sur Github

Introduction

Bienvenue sur API on Rails 5, un tutoriel sous stéroïdes pour apprendre la meilleure façon de construire votre prochaine API avec Rails. Le but de ce livre est de vous fournir une méthodologie complète pour développer une API RESTful en suivant les meilleures pratiques existantes. Lorsque vous en aurez fini avec ce livre, vous serez en mesure de créer votre propre API et de l'intégrer à n'importe quel client comme un navigateur Web ou votre une application mobile. Le code généré est construit avec Ruby on Rails 5.2 qui est la version actuelle [2: pour plus d'informations à ce sujet, consultez rubyonrails.org]. La version la plus récente de APIs on Rails 5 se trouve sur Github. N'oubliez pas de mettre à jour votre version hors ligne si c'est le cas.

L'intention de ce livre n'est pas seulement de vous apprendre à construire une API avec Rails mais plutôt de vous apprendre comment construire une API évolutive et maintenable avec Rails. C'est-à-dire améliorer vos connaissances actuelles avec Rails. Dans ce voyage, vous allez apprendrez à:

- Construire des réponses JSON
- Utiliser Git pour le contrôle de version
- Test de vos points finaux
- Optimiser et mettre en cache l'API

Je vous recommande fortement de suivre toutes les étapes de ce livre. Essayez de ne pas sauter des chapitres car je vais vous donner des conseils et des astuces pour vous améliorer tout au long du livre. Vous pouvez vous considérer comme le personnage principal d'un jeu vidéo qui obtient un niveau supérieur à chaque chapitre.

Dans ce premier chapitre, je vous expliquerai comment configurer votre environnement (au cas où vous ne l'auriez pas déjà). Nous allons ensuite créer une application appelée `market_place_api`. Je veillerai à vous enseigner les meilleures pratiques que j'ai pu apprendre au cours de mon expérience. Cela signifie qu'après avoir initialisé le projet, nous commencerons à utiliser **Git** .

Dans les prochains chapitres, nous allons construire l'application en suivant une méthode de travail simple que j'utilise quotidiennement. Nous développerons toute l'application en utilisant le **développement piloté par les tests** (TDD). Je vous expliquerai aussi l'intérêt d'utiliser une API pour votre prochain projet et de choisir un format de réponse adapté comme le JSON ou le XML. Plus loin, nous mettrons les mains dans le code et nous compléterons les bases de l'application en construisant toutes les routes nécessaires. Nous sécuriserons aussi l'accès à l'API en construisant une authentification par échange d'en-têtes HTTP. Enfin, dans le dernier chapitre, nous ajouterons quelques techniques d'optimisation pour améliorer la structure et les temps de réponse du serveur.

L'application finale sera une application de place de marché [3: Une application de type de marché permet à des vendeurs de mettre en place leur propore boutique en ligne.] où les utilisateurs seront en mesure de passer des commandes, télécharger des produits et plus encore. Il existe de nombreuses options pour créer une boutique en ligne comme Shopify, Spree ou Magento.

Tout au long de ce voyage (cela dépend de votre expertise), vous allez vous améliorer et être en mesure de mieux comprendre certaines des meilleures ressources Rails. J'ai aussi pris certaines des pratiques que j'ai trouvées sur Railscasts, CodeSchool et JSON API.

Conventions sur ce livre

Les conventions de ce livre sont basées sur celles du Tutoriel Ruby on Rails. Dans cette section, je vais en mentionner quelques-unes que vous ne connaissez peut-être pas.

Je vais utiliser de nombreux exemples en utilisant des ligne de commande. Je ne vais pas traiter avec Windows `cmd` (désolé les gars). Je vais baser tous les exemples en utilisant l'invite de ligne de commande de style Unix. Voici un exemple:

```
$ echo "A command-line command"
A command-line command
```

J'utiliserai quelques principes spécifiques à Ruby. C'est-à-dire:

- *"Éviter"* signifie que vous n'êtes pas censé le faire.
- *"Préférer"* indique que parmi les 2 options, la première est la plus appropriée.
- *"Utiliser"* signifie que vous êtes en mesure d'utiliser la ressource.

Si vous rencontrez une erreur quelconque lors de l'exécution d'une commande, je vous recommande d'utiliser votre moteur de recherche pour trouver votre solution. Malheureusement, je ne peux pas couvrir toutes les erreurs possibles. Si vous rencontrez des problèmes avec ce tutoriel, vous pouvez toujours m'envoyer un email.

Environnements de développement

Pour presque tous les développeurs, l'une des parties les plus douloureuses est de mettre en place un environnement de développement confortable. Si vous le faites correctement, les prochaines étapes devraient être un jeu d'enfant. Je vais vous guider dans cette étape afin de vous faciliter la tâche et de vous motiver.

Éditeurs de texte et Terminal

Les environnements de développement diffèrent d'un ordinateur à l'autre. Ce n'est pas le cas avec les éditeurs de texte. Je pense que pour le développement avec Rails, un IDE est beaucoup trop lourd. Cependant certains pensent que c'est la meilleure façon de travailler. Si c'est votre cas, je vous recommande d'essayer RadRails ou RubyMine. Tous deux sont bien maintenus et possèdent de nombreuses intégrations par défaut. Maintenant, pour ceux qui, comme moi, préfèrent des outils simples, je peux vous dire qu'il existe beaucoup d'outils disponibles que vous pourrez personnaliser via des plugins et plus.

- **Éditeur de texte**: J'utilise personnellement Vim comme éditeur. Au cas où vous n'êtes pas un fan de Vim, il y a beaucoup d'autres solutions comme Sublime Text qui est facile à prendre en main et surtout multi-plateforme . Il est fortement inspiré par TextMate. Une troisième option est d'utiliser un éditeur de texte plus récent comme Atom de Github. C'est un éditeur de texte prometteur fait en JavaScript. Il est facile à personnaliser pour répondre à vos besoins. N'importe lequel des éditeurs que je viens de vous présenter fera le travail. Choisissez donc celui où vous êtes le plus à l'aise.
- **Terminal**: Je ne suis pas un fan de l'application Terminal par défaut sous Mac OS. Je recommande iTerm2, qui est un remplacement de terminal pour Mac OS. Si vous êtes sous Linux, vous avez probablement déjà un bon terminal.

Navigateur web

Quand il s'agit de navigateurs, je conseillerai directement Firefox. Mais d'autres développeurs utilisent Chrome ou même Safari. N'importe lequel d'entre eux vous aidera à construire l'application que vous voulez. Ils proposent tous un bon inspecteur pour le DOM, un analyseur de réseau et de nombreuses autres fonctionnalités que vous connaissez peut-être déjà.

Gestionnaire de paquets

- **Mac OS**: Il existe de nombreuses options pour gérer la façon dont vous installez les paquets sur votre Mac, comme Mac Ports ou Homebrew. Les deux sont de bonnes options, mais je choisirais la dernière. J'ai rencontré moins de problèmes lors de l'installation de logiciels avec Homebrew. Pour installer `brew` il suffit d'exécuter la commande ci-dessous:

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

- **Linux**: Vous êtes déjà prêts! Peu importe si vous utilisez `apt`, `pacman`, `yum` tant que vous vous sentez à l'aise et que vous savez comment installer des paquets.

Git

Nous utiliserons beaucoup Git et vous devriez aussi l'utiliser (non seulement pour ce tutoriel mais aussi pour tous vos projets). Pour l'installer, c'est très facile:

- sous Mac OS: `$ brew install git`
- sous Linux: `$ sudo apt-get install git`

Ruby

Il existe de nombreuses façons d'installer et de gérer Ruby. Vous devriez probablement déjà avoir une version installée sur votre système. Pour connaître votre version, tapez simplement:

```
$ ruby -v
```

Rails 5.2 nécessite l'installation de la version 2.2.2 ou supérieure. Pour l'installer, je vous recommande d'utiliser Ruby Version Manager (RVM) ou rbenv. Ces outils vous permettront d'installer plusieurs versions de `ruby`. Dans ce tutoriel, nous allons utiliser RVM mais peu importe laquelle de ces deux options que vous utiliserez.

Pour installer RVM, rendez vous sur https://rvm.io/ et installez la clé GPG [4: La clé GPG vous permet de vérifier l'identité de l'auteur des sources que vous téléchargez.]. Une fois cela fait:

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-keys
409B6B1796C275462A1703113804BB82D39DC0E3 7D2BAF1CF37B13E2069D6956105BD0E739499BDB
$ \curl -sSL https://get.rvm.io | bash
```

Ensuite, vous pouvez installer la dernière version de Ruby:

```
$ rvm install 2.5
```

Si tout s'est bien passé, il est temps d'installer le reste des dépendances que nous allons utiliser.

Gemmes, Rails et bibliothèques manquantes

Tout d'abord, nous mettons à jour les Gemmes sur l'ensemble du système:

```
$ gem update --system
```

Dans la plupart des cas, si vous êtes sous Mac OS, vous devriez installer des bibliothèques supplémentaires:

```
$ brew install libtool libxslt libksba openssl
```

Nous installons ensuite les gemmes nécessaires et ignorons la documentation pour chaque gemme:

```
$ printf 'gem: --no-document' >> ~/.gemrc
$ gem install bundler
$ gem install foreman
$ gem install rails -v 5.2
```

Vérifiez que tout fonctionne bien:

```
$ rails -v 5.2
5.2.0
```

Bases de données

Je vous recommande fortement d'installer Postgresql pour gérer vos bases de données. Mais ici, pour plus de simplicité, nous allons utiliser SQlite. Si vous utilisez Mac OS vous n'avez pas de bibliothèques supplémentaires à installer. Si vous êtes sous Linux, ne vous inquiétez pas, je vous guide:

```
$ sudo apt-get install libxslt-dev libxml2-dev libsqlite3-dev
```

ou

```
$ sudo yum install libxslt-devel libxml2-devel libsqlite3-devel
```

Initialisation du projet

Vous devez sans doute déjà savoir comment initialiser une application Rails. Si ce n'est pas le cas, jetez un coup d'œil à cette section.

Sachez que nous utiliserons Rspec comme suite de test. Assurez-vous donc d'inclure l'option `--skip-test` lors de la création de l'application et l'option `--api`.

NOTE	L'option `--api` est apparue lors de la version 5 de Rails. Elle permet de limiter les librairies et *Middleware* inclus dans l'application. Cela permet aussi d'éviter de générer les vues HTML lors de l'utilisation des générateurs de Rails.

La commande est donc la suivante:

```
$ mkdir ~/workspace
$ cd ~/workspace
$ rails new market_place_api --skip-test --api
```

Comme vous pouvez le deviner, les commandes ci-dessus généreront les éléments indispensables à votre application Rails. La prochaine étape est d'ajouter quelques gemmes que nous utiliserons pour construire l'API.

Installer Pow ou Prax

Vous pouvez vous demander:

> Pourquoi diable voudrais-je installer ce type de paquet?

La réponse est simple: parce que nous allons travailler avec des sous-domaines. Pow et Prax vont nous aider a les créer très facilement.

Installer Pow

Pow ne fonctionne que sous Mac OS. Mais ne vous inquiétez pas: il existe une alternative qui imite les fonctionnalités sous Linux. Pour l'installer, tapez simplement:

```
$ curl get.pow.cx | sh
```

Et c'est tout ce que vous avez à faire. Il suffit d'établir un lien symbolique avec l'application pour configurer l'application Rack. D'abord vous allez dans le répertoire `~/.pow`:

```
$ cd ~/.pow
```

Ensuite, vous pouvez créer le lien symbolique

```
$ ln -s ~/workspace/market_place_api
```

N'oubliez pas de changer le répertoire utilisateur pour celui qui correspond au vôtre. Vous pouvez maintenant accéder à l'application via http://market_place_api.dev/. Votre application devrait être en cours d'exécution.

Installer Prax

Pour les utilisateurs de Linux uniquement, Prax distribue des paquets déjà compilés pour les distributions Debian / Ubuntu. Il suffit donc de télécharger le paquet `.deb` et de l'installer avec `dpkg`.

```
$ cd /tmp
$ wget https://github.com/ysbaddaden/prax.cr/releases/download/v0.8.0/prax_0.8.0-
1_amd64.deb
$ sudo dpkg -i prax_0.8.0-1_amd64.deb
```

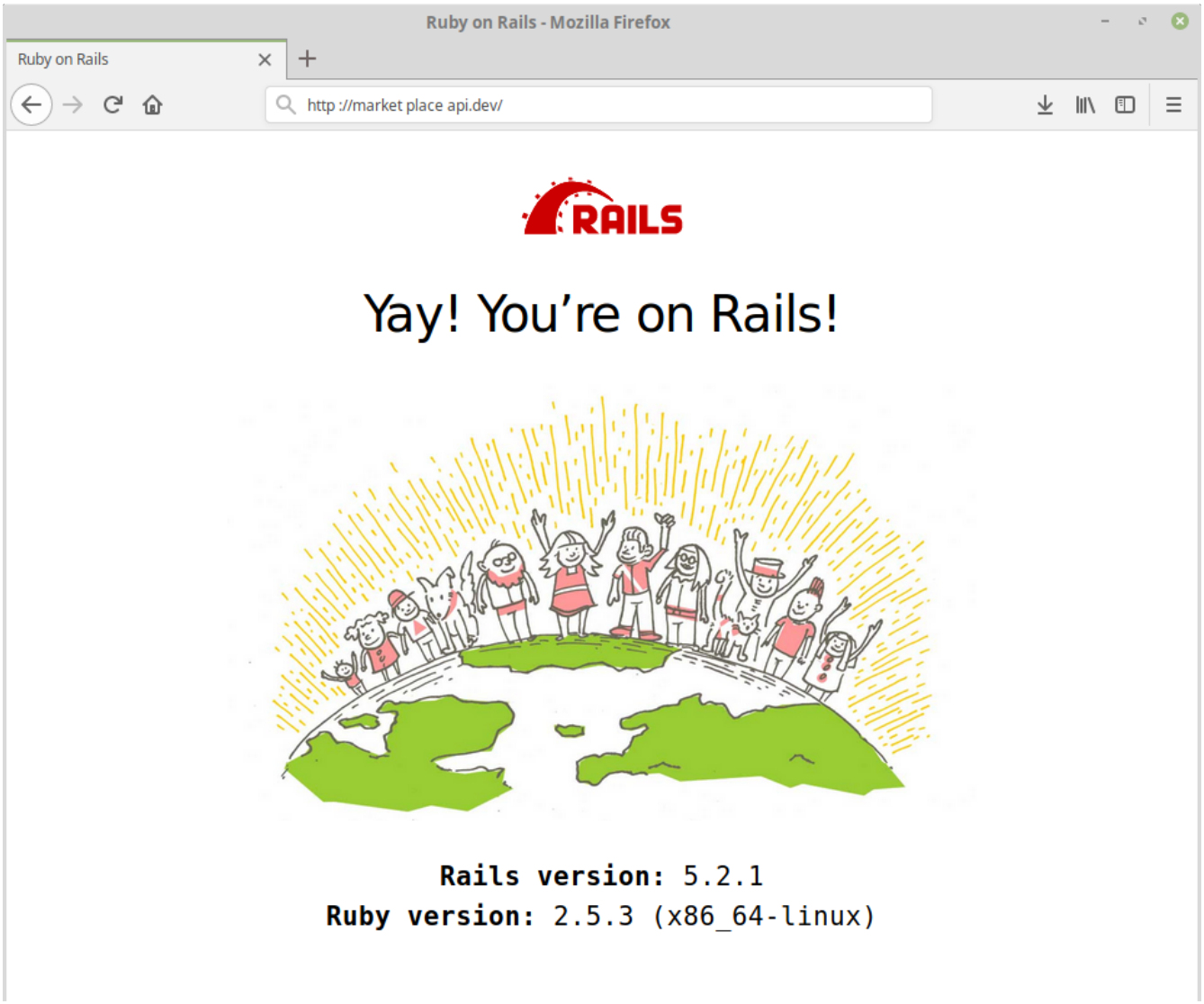
Ensuite, il ne nous reste plus qu'à lier les applications:

```
$ cd ~/workspace/market_place_api
$ prax link
```

Si vous voulez démarrer le Prax automatiquement, ajoutez cette ligne au fichier `.profile`:

```
prax start
```

Lors de l'utilisation de Prax vous devez spécifier le port de l'URL, dans ce cas-ci: http://market_place_api.dev:3000. Vous devriez ainsi voir l'application en marche comme le montre l'image suivante.



Une fois l'application Rails créée, l'étape suivante consiste à ajouter une gemme simple (mais très puissante) pour sérialiser les ressources que nous allons exposer avec l'API. La gemme s'appelle `active_model_serializers`. C'est un excellent choix pour la construction de ce type d'application car la librairie est bien maintenue et la documentation est incroyable.

Votre `Gemfile` devrait donc ressembler à ceci après avoir ajouté la gemme `active_model_serializers`:

Gemfile

```
source 'https://rubygems.org'
git_source(:github) { |repo| "https://github.com/#{repo}.git" }

ruby '2.5.3'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '~> 5.2.0'
# Use sqlite3 as the database for Active Record
gem 'sqlite3'
# Use Puma as the app server
gem 'puma', '~> 3.11'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'

# Api gems
gem 'active_model_serializers'
# ...
```

NOTE	Notez que j'enlève les gemmes `jbuilder` et `turbolinks` et `coffee-rails` car nous n'allons pas les utiliser.

C'est une bonne pratique aussi d'inclure la version Ruby utilisée sur l'ensemble du projet, ce qui empêche les dépendances de casser si le code est partagé entre différents développeurs, que ce soit pour un projet privé ou public.

Il est également important que vous mettiez à jour le `Gemfile` pour regrouper les différentes gemmes dans l'environnement correct:

Gemfile

```
# ...
group :development do
  gem 'sqlite3'
end
# ...
```

Ceci, comme vous vous en souvenez peut-être, empêchera l'installation ou l'utilisation de Sqlite lorsque vous déployez votre application chez un fournisseur de serveurs comme Heroku [5: Heroku facilite le déploiement de votre application en installant les dépendances sur un serveur en analysant votre *Gemfile*].

Une fois cette configuration effectuée, il est temps d'exécuter la commande d'installation du paquet pour intégrer les dépendances correspondantes:

```
$ bundle install
```

Une fois que la commande a terminé son exécution, il est temps de commencer à **versionner le projet** avec Git.

Contrôle de version

Rappelez-vous que Git vous aide à suivre et à maintenir l'historique de votre code. Gardez à l'esprit que le code source de l'application est publié sur Github. Vous pouvez suivre le projet sur Github

À ce stade, je suppose que vous avez déjà configuré Git et que vous êtes prêt à l'utiliser pour suivre le projet. Si ce n'est pas votre cas, initialisez simplement les paramètres basiques suivants:

```
$ git config --global user.name "Type in your name"
$ git config --global user.email "Type in your email"
$ git config --global core.editor "vim"
```

Il est donc temps d'initier le projet avec Git. N'oubliez pas de naviguer dans le répertoire racine de l'application `market_place_api`:

```
$ git init
Initialized empty Git repository in ~/workspace/market_place_api/.git/
```

L'étape suivante est d'ignorer certains fichiers que nous ne voulons pas suivre. Votre fichier `.gitignore` devrait ressembler à celui montré ci-dessous:

.gitignore

```
# Ignore bundler config.
/.bundle

# Ignore the default SQLite database.
/db/*.sqlite3
/db/*.sqlite3-journal

# Ignore all logfiles and tempfiles.
/log/*
/tmp/*
!/log/.keep
!/tmp/.keep

# Ignore uploaded files in development
/storage/*

/node_modules
/yarn-error.log

/public/assets
.byebug_history

# Ignore master key for decrypting credentials and more.
/config/master.key
```

Après avoir modifié le fichier `.gitignore`, il suffit d'ajouter les fichiers et de valider les modifications. Les commandes nécessaires sont indiquées ci-dessous:

```
$ git add .
$ git commit -m "Initial commit"
```

TIP	J'ai appris que commencer un message par un verbe au présent décrit ce que fait le commit et non ce qu'il a fait. De cette façon il est plus facile de lire et de comprendre l'historique du projet (ou du moins pour moi). Je vais suivre cette pratique jusqu'à la fin du tutoriel.

Enfin, et c'est une étape optionnelle, nous déployons le projet sur **Github** (je ne vais pas l'expliquer ici) et poussons notre code vers le serveur distant. On commence donc par ajouter un serveur distant:

```
$ git remote add origin git@github.com:madeindjs/market_place_api.git
```

Ensuite on pousse le code:

```
$ git push -u origin master
```

Au fur et à mesure que nous avançons dans le tutoriel, j'utiliserai les pratiques que j'utilise quotidiennement. Cela inclut le travail avec les branches, le rebasage, le squash et bien d'autres. Vous n'avez pas à vous inquiéter si vous ne connaissez pas tous ces termes, je les expliquerai le temps venu.

Conclusion

Cela a été un chapitre assez long. Si vous êtes arrivés ici, permettez-moi de vous féliciter. Les choses vont s'améliorer à partir de ce point. Commençons à mettre les mains dans le code!