

# RAILS 3.1

From Idea to Deployment

# RUBY ON RAILS

---

**THIS IS PREPARED FOR BEGINNERS INTERESTED  
IN LEARNING RUBY ON RAILS DEVELOPMENT**

Katherine Rose Anne G. Pe (Katz)

Senior Web Application Developer

# THE IDEA

---

**“Simplicity boils down to two steps: Identify the essential. Eliminate the rest.” – Leo Babauta**

- The project: A basic blog application that allows us to add, edit and delete posts.
- You can actually host your blog for free on heroku and just buy a domain.

# **“BABY STEPS ARE NOT JUST FOR BABIES.”**

---

- Install Rails 3.1 on Windows/ Mac/ Linux
- Setup recipe for Rails Girls

# INSTALL POSTGRESQL

---

```
brew install postgresql
```

- Follow instructions (brew info postgresql)

```
initdb /usr/local/var/postgres
```

- PostgreSQL is the world's most advanced open source database

# TEXT EDITOR

---

- I prefer Textmate but you can use what you like. Textmate has bundles for Ruby on Rails, HAML and others.
- For Linux, GEdit is quite good and it's even free. For some time I have been using Gmate.

- HAML Textmate Bundle
- Ruby on Rails Textmate Bundle

# HAVING INSTALLATION WOES?

---

- Search on google.com or ask:
- Stackoverflow
- #rubyonrails on irc.freenode.net
- me (@katgironpe)

# DESIGN-DRIVEN DEVELOPMENT

---

- Consider how you want things to work, how you want things to look like and how you want to communicate.
- For those who attended Rails Girls (Singapore), there are a lot of resources on the workbook.

# USE A CSS FRAMEWORK

---

- I suggest starting with the design but since this is just a tutorial for you to learn Rails. Let's skip and use Twitter bootstrap.

- You might want to download Less Framework Grid (PSD)

# USE GIT

---

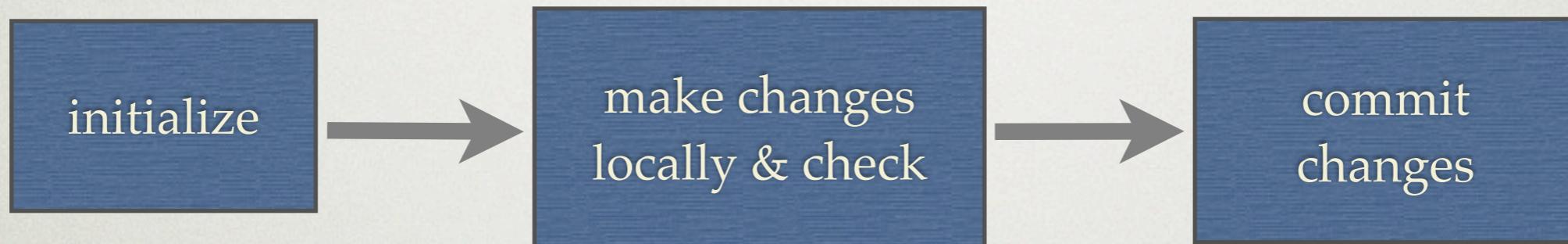
- Recommended reading:
- [Git for the Lazy](#)
- [Git Immersion](#)
- [Stop littering your .gitignore's](#)
- Create a [github.com](#) account (free)
- Create a [heroku account](#) (free)

You can download [GitX](#) for OS X.

# USE GIT

---

- Git is a version control system.
- The process is:



**git init**  
Update your .gitignore  
file as needed

**git remote add origin**  
user@server:~/gitpath

make changes  
locally & check

**git status**  
will display  
changes made

commit  
changes

**git commit -a -m "Message"**  
This will add files modified / created.  
**git push origin master**

# ESSENTIAL CONCEPTS

---

- Ruby - May take a few hours to read and understand
- REST - The wiki definition is complex. Please read about RESTful Rails.
- CRUD - The actions we perform — create, read, update, delete.
- MVC - Model View Controller

Must Read: RESTful Rails

# ETC.

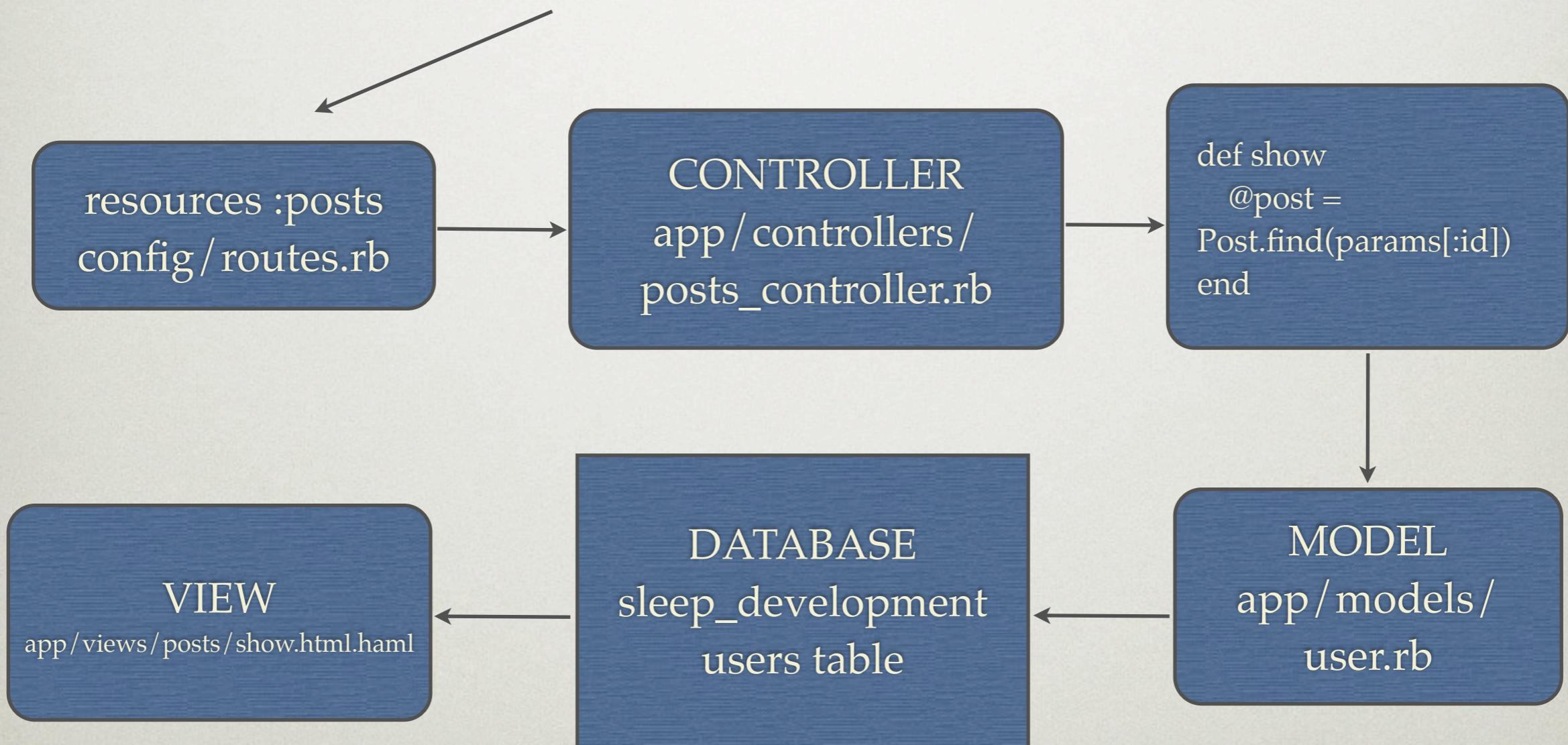
---

- KISS
- DRY
- SLEEP

# BIG PICTURE

GET is a request. In Rails, a SHOW action is always a GET request.

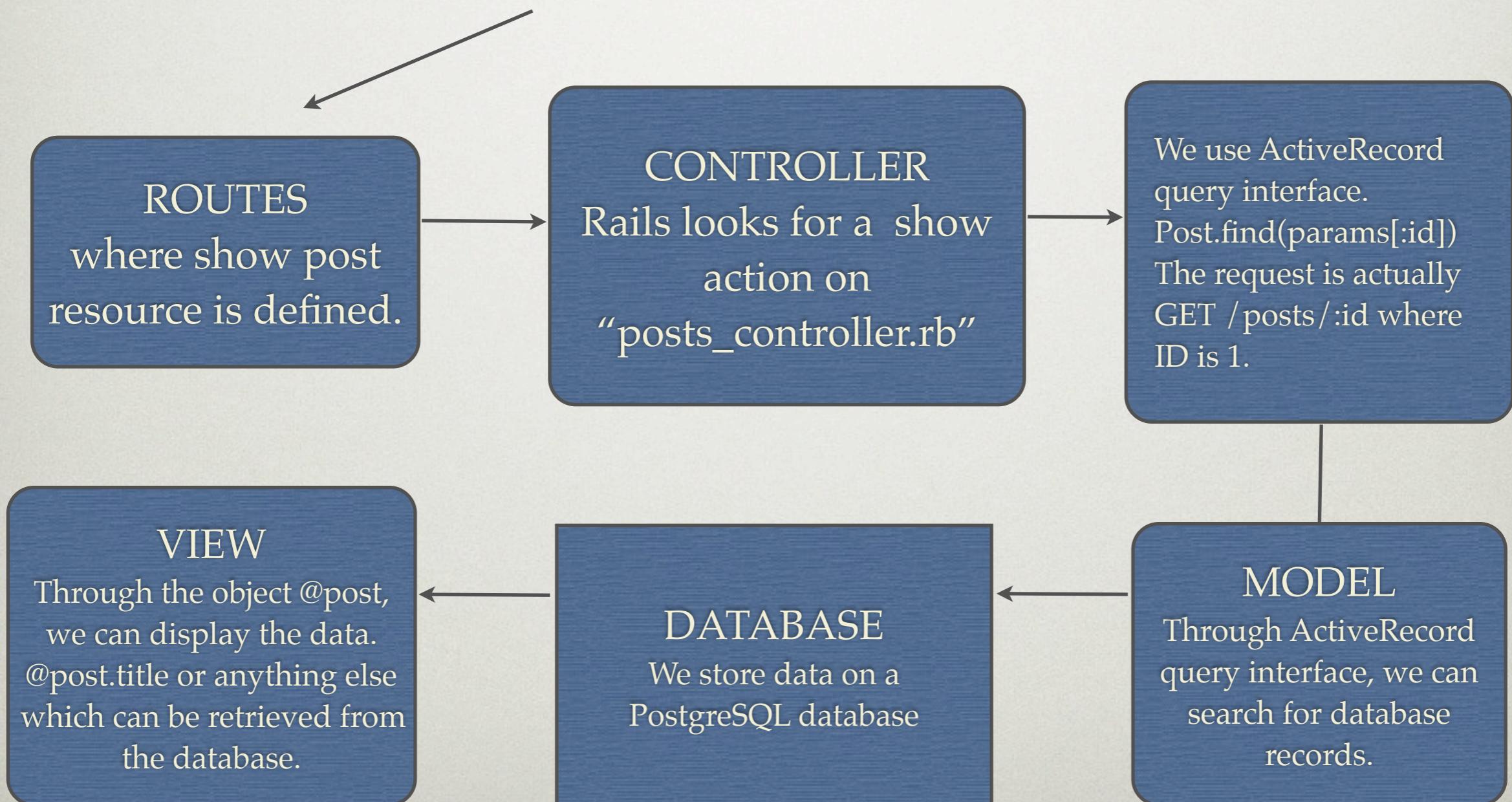
user requests for http://localhost:3000/posts/1



# BIG PICTURE

GET is a request. In Rails, a SHOW action is always a GET request.

user requests for http://localhost:3000/posts/1



# MORE ON REST

---

- There's a free Peepcode cheat sheet which may be a little outdated as it was written for Rails 2.x but it should help you understand REST for Rails.

# USE A RAILS TEMPLATE

---

- Templates make setting up a Rails application easier.
- This will install dependencies required for creating a Rails application:
- Rails Edge Template

Check out other [Rails 3.1 App Templates](#) or [build your own](#).

# USE A RAILS TEMPLATE

---

- rails new *appname* -m <https://raw.github.com/bridgeutopia/rails-edge-template/master/template.rb>

Answer yes for all except for the MySQL part. We will use PostgreSQL here and Twitter Bootstrap.

# UPDATE CONFIG FILE

---

- If you installed PostgreSQL on a Mac/Linux OS as suggested, your username should be the same as your Mac username. Your config/database.yml should look like:

```
development:  
  adapter: postgresql  
  encoding: unicode  
  database: sleep_development  
  pool: 5  
  username: katz  
  password:
```

database name will depend on the name of your app by default or it could be renamed to anything less confusing for you.

# NAMING CONVENTIONS

---

- In Rails, the naming convention is that a database table is in its **plural form** and a model should be in its **singular form**.
- Thus we have: `user.rb` and the class name `User`. Rails will expect that we have a table “`users`.” Controllers are also in plural form. E.g, `users_controller.rb`.

# RAILS MODELS

---

- Start with the models and the business logic.
- Essentially, we have an admin “user” who wants to login to create a post. And be able to update and delete it as well.

Important: [Learn Active Record Query Interface](#)

“FAT Models, Thin Controllers” is a best practice but I would go against FAT anything in your application. If you organize classes well enough, it won’t be so FAT. More on that later.

# ACTIVE RECORD

---

- Read about the Active Record query interface
- Simple examples:
- `Post.find(params[:id])`
- `Post.order('published_at DESC')` will sort based on publish date. The most recent ones will appear first.
- `Post.where(:published=>true).order('published_at DESC')` will only display published posts and order based on publish date.

# HAML, SASS & SCSS

---

- Instead of using ERB template system, I recommend HAML simply because it is more readable. For those who know HTML and CSS, you will find HAML quite easy to understand.

In ERB:

```
<div class='container'>
<%= yield %>
</div>
```

In HAML:

```
.container
  = yield
```

HAML

SASS

Textmate Bundle for HAML

# RUBY GEMS & BUNDLER

---

- What are ruby gems?
- They are ruby applications or libraries packaged (usually for open source distribution) via [rubygems.org](http://rubygems.org)
- What is bundler?
- Bundler is a gem which we use to manage a Ruby application's dependencies. The "Gemfile" contains all dependencies and we can just run "bundle" to install that.

We will use a lot of gems in this application.  
An important gem which will be used for authentication is devise.

# USER AUTHENTICATION

---

- Recommended reading: [Devise for Rails Authentication](#)
- The first file we will modify is the migration file which creates the fields required for authentication.
- By default, *trackable*, *confirmable* and *lockable* features of devise are not enabled. I do not need confirmable and lockable but I need [trackable](#) which allows me to see how many times a user signed in and also log the I.P address of the user.

Check The [Ruby Toolbox](#) for other options. Devise is just one of the gems we use for Authentication.

# DEVISE

---

- The rails template already installed devise if you answered “yes”. You can read the documentation and try to see whether there are any issues on github but I think it’s very stable.

What devise changes can we do for this app?

Create HAML views

Allow user to sign in using username or email

# MEET RAKE

---

- `rake -T` will show you your options. There's quite a lot but for now you only have to bother about two things: `rake db:create:all` and `rake db:migrate`
- `rake db:create:all` will create all databases for 3 different environments based on your `config/database.yml` file
- `rake db:migrate` will create the table “user” based on the migration file we just updated.

# MIGRATIONS

---

```
rake db:migrate
```

That will create the users table.

```
rails g migration add_username_to_users  
username:string
```

That creates a file which requires some update.

```
Rails 3.1 Update: Reversible migrations
```

# MIGRATIONS

---

- Adding username (string) cannot be reversed so we cannot use the change method. [Read this.](#)
- The change method is something new. Previously we are required to add down method with drop\_table :table\_name for each migration that required it.

# MIGRATIONS

---

```
def change
  add_column :users, :username, :string
  add_index :users, :username, :unique => true
end
```

Change method works for `create_table`. In this case, we need the “up” and “down” methods.

The `add_index` is for optimization.

```
def up
  add_column :users, :username, :string
  add_index :users, :username, :unique => true
end

def down
  remove_column :users, :username
end
```

# MIGRATIONS

---

- What happened? Open **Navicat Lite** and create a new PostgreSQL connection. If your username on your Mac/Linux is “katz,” change postgres to “katz”
- [View my PostgreSQL connection settings](#)
- After “rake db:migrate,” we should see a [users table](#) created.
- **Navicat** should later help you see records created as well.

# ADDING VALIDATIONS

---

- Validations are added on models files.
- On app/models/user.rb, we can add:

```
validates :username, :presence=>true, :uniqueness=>true
```

- We want to make sure username is present and unique every time we create or update a user. But even the validation will not work unless we specify that the field username is accessible via Ruby's **attr\_accessible**

```
attr_accessible :email, :password, :password_confirmation, :remember_me, :username
```

- **attr\_accessible** doesn't always have to exist for all of your models. It is added for security. To know more about it, [read this](#).

# DEVISE VIEWS

---

- Most of us do not consume time trying to make authentication work. We may, however, spend some time altering devise defaults. For now we have two goals: allow user to log in using “username” and make sure the views are in HAML and not ERB. Those are sufficient for a basic Rails app that requires authentication. Sometimes logging in using “username” is not even required. I just feel it is better.

```
rails g devise:views
```

That generates view files on app/views/devise but let's use HAML for consistency.

```
for i in `find app/views/devise -name *.erb` ; do html2haml -e $i ${i%erb}haml ; rm $i ;
```

That generates converts everything devise folder to HAML and removes ERB files.

# UPDATING ROUTES

---

- Right now you can “rake routes” to see the links available for you. Notice it shows ‘/users/sign\_in’ and for me ‘/login’ is a more friendly link so let’s update **config/routes.rb** to make that work.

Below this line:

```
devise_for :users
```

Add:

```
devise_scope :user do
  get "/login" => "devise/sessions#new"
  get "/logout" => "devise/sessions#destroy"
  get "/register" => 'devise/registrations#new'
  get "/account" => 'devise/registrations#edit'
end
```

To create a new user, access <http://localhost:3000/register>

# USERNAME

---

- Add the username field for devise views. On app/views/devise/registrations/new.html.haml and edit.html.haml, add:

```
%div
  = f.label :username
  %br/
  = f.text_field :username
```

- Follow all of the steps on this page except for the migration part because we already have a username field on the database:

[How To: Allow users to sign\\_in using their username or email address](#)

# ADMIN/POSTS

---

```
rails g scaffold post user_id:integer title:string body:text slug:string published:boolean
```

- That creates a lot and pretty much everything needed to create a post. But we'll try to create a namespace so we can create posts via /admin/posts/new rather than /posts/new.

```
mkdir -p app/views/admin/posts && cp -R app/views/posts/* app/views/admin/posts
```

- That will copy view files to views/admin. But we need to update the form view (admin/views/posts/\_form.html.haml). Change this line:

```
= semantic_form_for @post do |f|
```

- to this:

```
= semantic_form_for [:admin, @post] do |f|
```

# ADMIN/POSTS

---

- Update the migration file. It should look like this:

```
def change
  create_table :posts do |t|
    t.integer :user_id
    t.string :title
    t.text :body
    t.string :slug
    t.boolean :published
    t.datetime :published_at

    t.timestamps
  end
  add_index :posts, :user_id
  add_index :posts, :slug, :unique=>true
end
```

- The slug field is for Friendly ID because we want friendly URL's.

# ADMIN/POSTS

---

- Update the model files. A user has many posts and a post belongs to a user. Let's update the two models and add validation.

app/models/post.rb should look like:

```
class Post < ActiveRecord::Base
  belongs_to :user
  validates :user_id, :presence=>true
  validates :title, :presence=>true
  validates :body, :presence=>true
end
```

and add this to app/models/user.rb:

```
has_many :posts
```

# ADMIN/POSTS

---

- We need friendly URL's so let's add the gem 'friendly\_id.' We've already added the slug field which is required by the gem.

app/models/post.rb should look like:

```
class Post < ActiveRecord::Base
  belongs_to :user
  validates :user_id, :presence=>true
  validates :title, :presence=>true
  validates :body, :presence=>true

  extend FriendlyId
  friendly_id :title, :use=> :slugged
end
```

# ADMIN/POSTS

---

- We need to copy the controller file to admin/controllers/admin.

```
mkdir -p app/controllers/admin/posts && cp -R app/controllers/posts_controller.rb app/controllers/admin/
```

- Update admin/posts\_controller.rb

```
class PostsController < ApplicationController
```

- should be:

```
class Admin::PostsController < ApplicationController
```

- For app/views/posts\_controller.rb, you can delete all methods except for index and show. The same is true for the views. Remove app/views/posts/new.html.haml, edit.html.haml and \_form.html.haml

# ADMIN/POSTS

---

- Update config/routes.rb and add:

```
namespace :admin do
  resources :posts
end
```

- That makes admin/posts/new, admin/posts/, admin/posts/:id, admin/post/edit accessible. But why can anyone access it? Because we haven't added the filter to make sure a user is logged in to access the admin pages. Let's update app/controllers/posts\_controller.rb and add this to top of all the methods:

```
before_filter :authenticate_user!
```

# ADMIN/POSTS

---

- We need to know who created the post at least so let's update the create action.

```
@post = Post.new(params[:post])  
should now be  
@post = Post.new(params[:post].merge(:user => current_user))
```

- Finally to get the links working on admin/posts. Simply replace new\_post\_path to new\_admin\_post\_path and edit\_post\_path(post) to edit\_admin\_post\_path(post) on the views. And update admin/controllers/posts\_controller.rb redirect paths:

```
redirect_to @post  
should now be  
redirect_to admin_post_path(@post)
```

# INDEX/ROOT

---

- Make things prettier. Do what you want to with the interface. We can display blog posts on index / root.

```
rm public/index.html
```

- Update config/routes.rb and add below all the other entries:

```
root :to => "posts#index"
```

# DEALING WITH TIME

---

- Why do we have the published\_at field? It makes sense especially if we want to publish a post on a future date. But published\_at is optional and sometimes we just want it to be the same as created\_at. We now need to create an instance method that will allow us to display published\_at if it is available or display created\_at if not. Add this on post.rb

```
def posted_on
  published_at.present? ? published_at : created_at
end
```

- That is shorthand for:

```
if published_at.present?
  published_at
else
  created_at
end
```

# DEALING WITH TIME

---

- We might want to format time and remove the hours and minutes. I like that and prefer that.  
So I added config/initializers/time\_formats.rb

```
Time::DATE_FORMATS[:no_time] = "%B %d, %Y"
```

- On the views file (posts/index.html.haml), we can use that format.

```
.post-details= "Posted on #{post.posted_on.to_s(:no_time)}"
```

# COMMENTS

---

- During Rails Girls, I noticed that many had interest in learning how to add a feedback form and most of you got it working but I have to be honest that I'd rather not code for comments. I use **Disqus**. For those interested in having that feature for any app and don't want to use Disqus, try [acts\\_as\\_commentable](#) gem. Don't scaffold a comments feature (`rails g scaffold comment user_id:integer body:text`), all you have to do is follow the instructions on the documentation and make changes to `comments_controller.rb`. Polymorphism is another concept you should learn.

```
@comment = Comment.new(params[:comment])
```

```
@commentable = params[:comment]
[:commentable_type].constantize.find(params[:commentable_id])
params[:comment][:commentable_id] = @commentable.id
```

# DEPLOYING ON HEROKU

---

- Make sure you have a public key (id\_rsa.pub) on .ssh folder.

```
ls ~/.ssh
```

- If you don't have that, try:

```
ssh-keygen -t rsa -C "your_email@youremail.com"
```

- Register on heroku and install heroku gem (gem install heroku).
- And do:

```
heroku create --stack cedar
```

- Log in to heroku and rename the app created.

# DEPLOYING ON HEROKU

---

```
git remote add heroku git@heroku.com:sleepy.git  
git status  
git add -A  
git commit -m "Message"  
git push heroku
```

Deploying with Git on Heroku

# DEPLOYING ON HEROKU

---

```
RAILS_ENV=production rake assets:precompile  
git commit -a -m 'all that'  
git push heroku master  
heroku run rake --trace db:migrate
```

Update your config/database.yml only if the last command returns an error.

```
heroku run console  
ENV['DATABASE_URL']  
postgres://username:password@hostname/database  
heroku run rake --trace db:migrate
```

[PostgreSQL on Heroku](#)

# SLEEPY APP

---

- Access <http://sleepy.herokuapp.com>

```
https://github.com/bridgeutopia/sleep
```

```
git clone git://github.com/bridgeutopia/sleep.git
```

Credits : Rico Sta. Cruz for his gist on global .gitignore and [filler text generator](#).

Recommended Reading: [Free Rails 3.0 tutorial by Bryan Bibat](#)

# OH-VER

---

- Overwhelming? There's still a lot to learn. I admit I have a lot to learn despite being an old, senior developer.
- If you are a serious type who wants to learn how to get things done right, learn about **Test Driven Development**. Based on experience, when apps become too complex and there are no tests written, even with a QA tester, it's difficult to see what's wrong. The last thing you want to hear is a user complaining.

RSpec

# THANK YOU

I've had much fun during my short stay as a traveler in Singapore. Thanks for the invitation and everything imparted.

#geekcamp Singapore

Rails Girls 

Jason Ong of JasoMedia