(Ruby on) Rails

Introduction to Rails

Introduction to Rails

- David Heinemeier Hansson （2003）

What is Rails?

- Rails is…
  - Written in Ruby
  - A web development framework
  - For development of web applications written in Ruby
- Benefits of Rails
  - Built-in functionality
  - Encourages good software development practices
  - Open source and lots of community support

Creating a New Rails App

- We simply call the Rails gem and give it the path to our new application
- Create your Rails application!

  rails new ***path-to-application***
- Example
  - rails new my_app
- This will spit out a bunch of files that compose your new Ruby web application

Starting Your Application

- Open your application folder (./my_app)
- Start Webrick, the built-in webserver

  rails server or **rails s**

- Working locally
  - Open up your favorite browser and view the app on localhost with port 3000

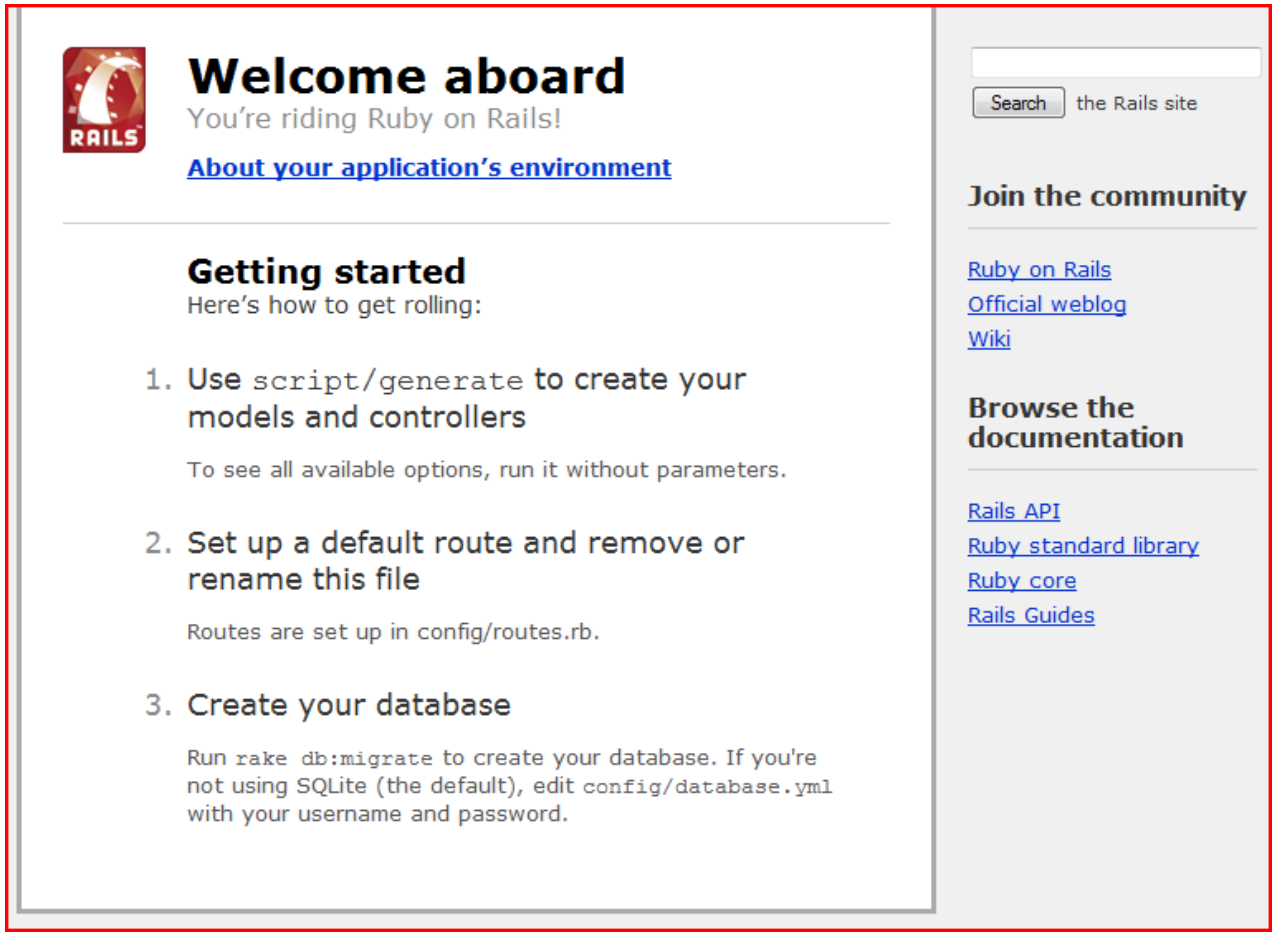
    http://localhost:3000/

Viewing Your Application

- A new application (Rails version 5.0.0)

Viewing Your Application

- A new application (Rails version 4.0.0)



Navigating the Rails File System

- When we open our project (./my_app), we see a number of folders

- For now, we will only be interested in a few of them
  - The "app" folder, specifically "app\views"
  - The "config" folder

Rails files

app/	Contains the controllers, models, views and assets for your application. You'll focus on this folder for the remainder of this guide.
bin/	rails, rake, bundle,…
config/	Configure your application's runtime rules, routes, database, and more. This is covered in more detail in Configuring Rails Applications
db/	Contains your current database schema, as well as the database migrations.
lib/	Extended modules for your application.

Rails files

log/	Application log files.
public/	The only folder seen to the world as-is. Contains the static files and compiled assets.
test/	Unit tests, fixtures, and other test apparatus. These are covered in Testing Rails Applications
config.ru	Rack configuration for Rack based servers used to start the application.
Gemfile **Gemfile.lock**	These files allow you to specify what gem dependencies are needed for your Rails application.

The "config" Folder

- The "config" folder will be where we configure particular settings of our Rails application
- We will tell our application how to setup the URLs of our app in the "routes.rb" file
- Eventually, we will tell our app how to connect to a particular database in the "database.yml" file

Configuring your DB

- SQLite: built-in support, lightweight

```
default: &default
  adapter: sqlite3
  pool: 5
  timeout: 5000

development:
  <<: *default
  database: db/development.sqlite3
```

- Create an empty DB
  - **rake db:create**

Hello Rails!

- You need to create at minimum a controller and a view.
- You can do that in a single command. Enter this command in your terminal:

rails generate controller home index

erb

- Rails will create several files for you
- app/views/home/index.html.erb: is the template that will be used to display the results of the index action (method) in the home controller.
- Edit this file in your text editor and edit it to contain a single line of code:

```
<h1>Hello, Rails!</h1>
```

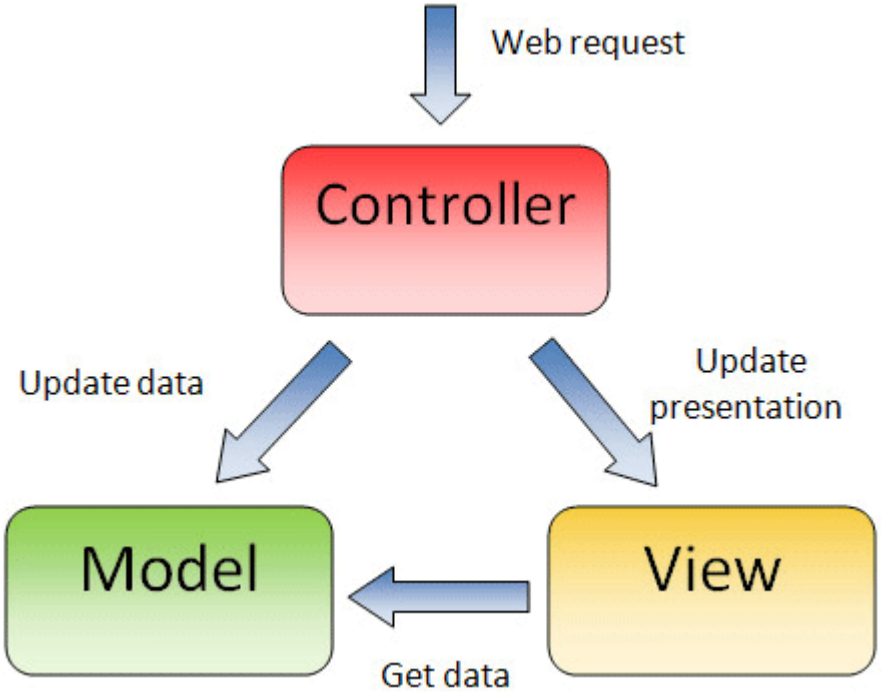
Setting your application homepage

- Open: config/routes.rb

```
#You can have the root of your site routed with "root"
# just remember to delete public/index.html.
# root :to => 'welcome#index‘

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

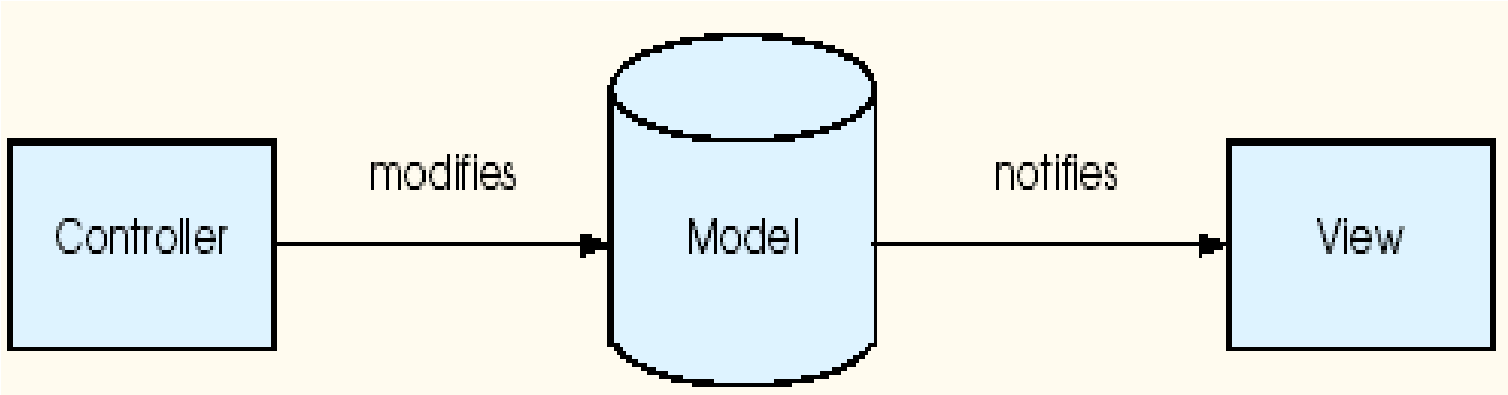
MVC on Rails

- Model-View-Controller

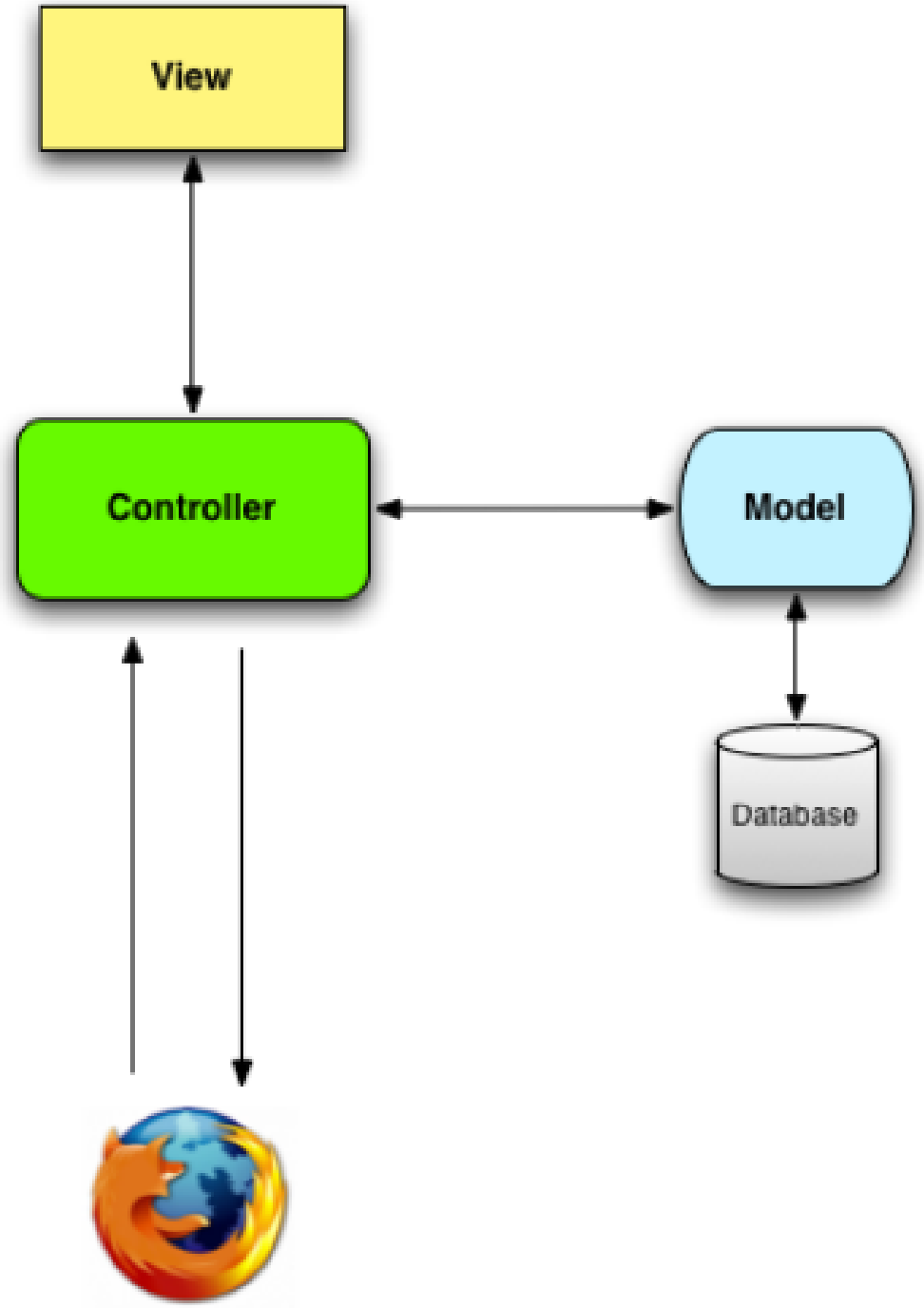


Model-View-Controller Architecture

The *model-view-controller* architecture (MVC) separates application data (contained in the *model*) from graphical presentation components (the *view*) and input-processing logic (the *controller*).



MVC on Rails



MVC

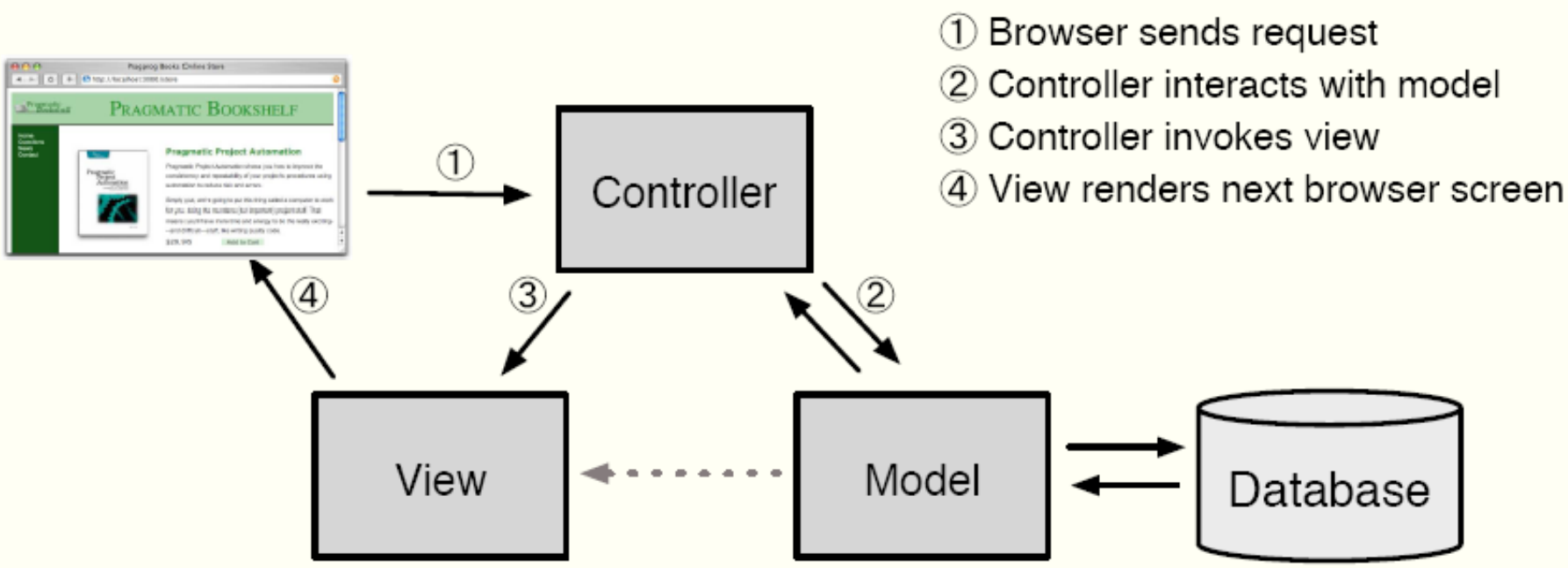


Figure 2.1: The Model-View-Controller Architecture

The "app" Folder

- The "app" folder deals with the actual code of our application.
- MVC (model-view-controller)
- It will hold all of our...
  - **M**: Objects ("models"),
  - **V**: .erb files ("views"), and…
  - **C**: code to work between the two ("controllers")

Scaffolding

- Rails ***scaffolding*** is a quick way to generate some of the major pieces of an application.
- If you want to create the models, views, and controllers for a new resource in a single operation, scaffolding is the tool for the job.

Blog Example

- Let's create a blog app with Rails
- Using Scaffold to create a Post resource that represents a single blog post:

rails generate scaffold
Post name:string title:string content:text

Ruby on Rails

Exploring the app

- Let's walk around the app a little...
  - Create a new post
  - Read (show) a post, or list all posts
  - Update a post's info
  - Destroy a post (!)

Ruby on Rails

What Scaffold generator created

db/migrate/**20100207214725_create_posts.rb**	Migration to create the posts table in your database (your name will include a different timestamp)
app/models/**post.rb**	The Post model
test/unit/post_test.rb	Unit testing harness for the posts model
test/fixtures/posts.yml	Sample posts for use in testing
config/**routes.rb**	Edited to include routing information for posts
app/controllers/**posts_controller.rb**	The Posts controller
app/views/posts/**index.html.erb**	A view to display an index of all posts
app/views/posts/**edit.html.erb**	A view to edit an existing post
app/views/posts/**show.html.erb**	A view to display a single post
app/views/posts/**new.html.erb**	A view to create a new post

What Scaffold generator created

app/views/posts/_form.html.erb	A partial to control the overall look and feel of the form used in edit and new views
test/functional/posts_controller_test.rb	Functional testing harness for the posts controller
app/helpers/posts_helper.rb	Helper functions to be used from the post views
test/unit/helpers/posts_helper_test.rb	Unit testing harness for the posts helper
app/assets/javascripts/posts.js.coffee	CoffeeScript for the posts controller
app/assets/stylesheets/posts.css.scss	Cascading style sheet for the posts controller
app/assets/stylesheets/scaffolds.css.scss	Cascading style sheet to make the scaffolded views look better

Database Migration

- Migrations are Ruby classes that are designed to make it simple to create and modify database tables.
- Use "rake"commands **rake db:migrate**
- Look in the db/migrate/**20160207214725_create_posts.rb** (yours will have a slightly different name)
  - It also creates two timestamp fields to allow Rails to track post creation and update times.

Add a link

- app/views/home/index.html.erb

```
<h1>Hello, Rails!</h1>
<%= link_to "My Blog", posts_path %>
```

HTML :

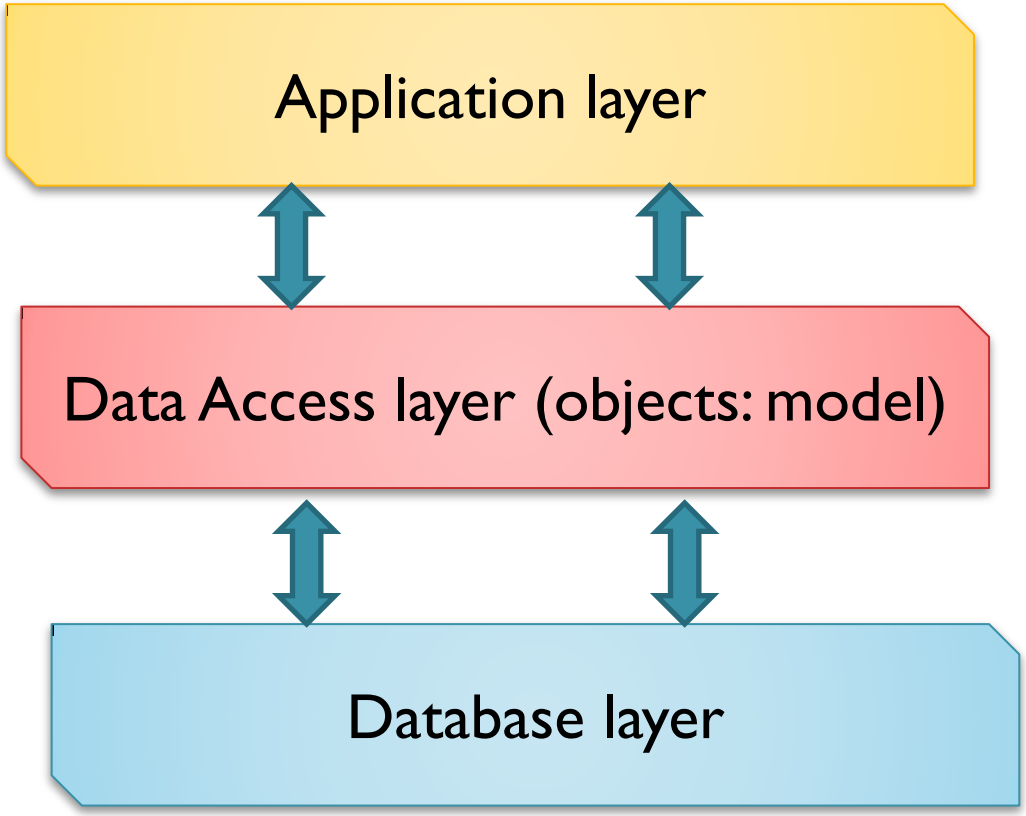
```
<body>
  <!--<h1>Home#index</h1>-->
<!--<p>Find me in app/views/home/index.html.erb</p>-->

<h1>Hello, Rails!</h1>
<a href="/posts">My Blog</a>

</body>
```

The Model

- Check app/models/**post.rb**
- The Post class inherits from ActiveRecord
- Active Record supplies a great deal of functionality to your Rails models for free, including
  - basic database CRUD (Create, Read, Update, Destroy) operations,
  - data validation,
  - sophisticated search support

Architecture Design Principle

- Manipulate objects rather than database



Mapping between objects and tables

Basic Concepts: Active Record

- Relational Databases: organize large amounts of data
  - Issue: no easy mapping to object-oriented design
- Active Record Pattern
  - A *design pattern* first described by Martin Fowler
  - Object = table row / attributes stored as columns
  - Table = collection of objects
  - OOD = database (collection of tables)
  - Rails: link classes through id fields in tables

Mapping database to objects

- Operations on the class and object
  - post = Post.new
  - attr-accessor: post.name ; post.name = ‘Mike’
  - post.save
  - post.updated_at
  - foo = Post.create(name: "Foo", title: "nil")
  - foo.destroy
  - Post.find(1); Post.find_by(title: "nil")
  - Post.all; Post.first

Add Validation

- Edit the model:

```
class Post < ActiveRecord::Base

  validates :name,  :presence => true
  validates :title, :presence => true,
                    :length => { :minimum => 5 }

end
```

Ruby on Rails

Test validation

```
$ rails console

>> p = Post.new(:content => "A new post")
=> #<Post id: nil, name: nil, title: nil,
     content: "A new post", created_at: nil,
     updated_at: nil>
>>p.save
>> p.errors.full_messages
=> ["Name can't be blank", "Title can't be blank",
"Title is too short (minimum is 5) "]
```

Routes

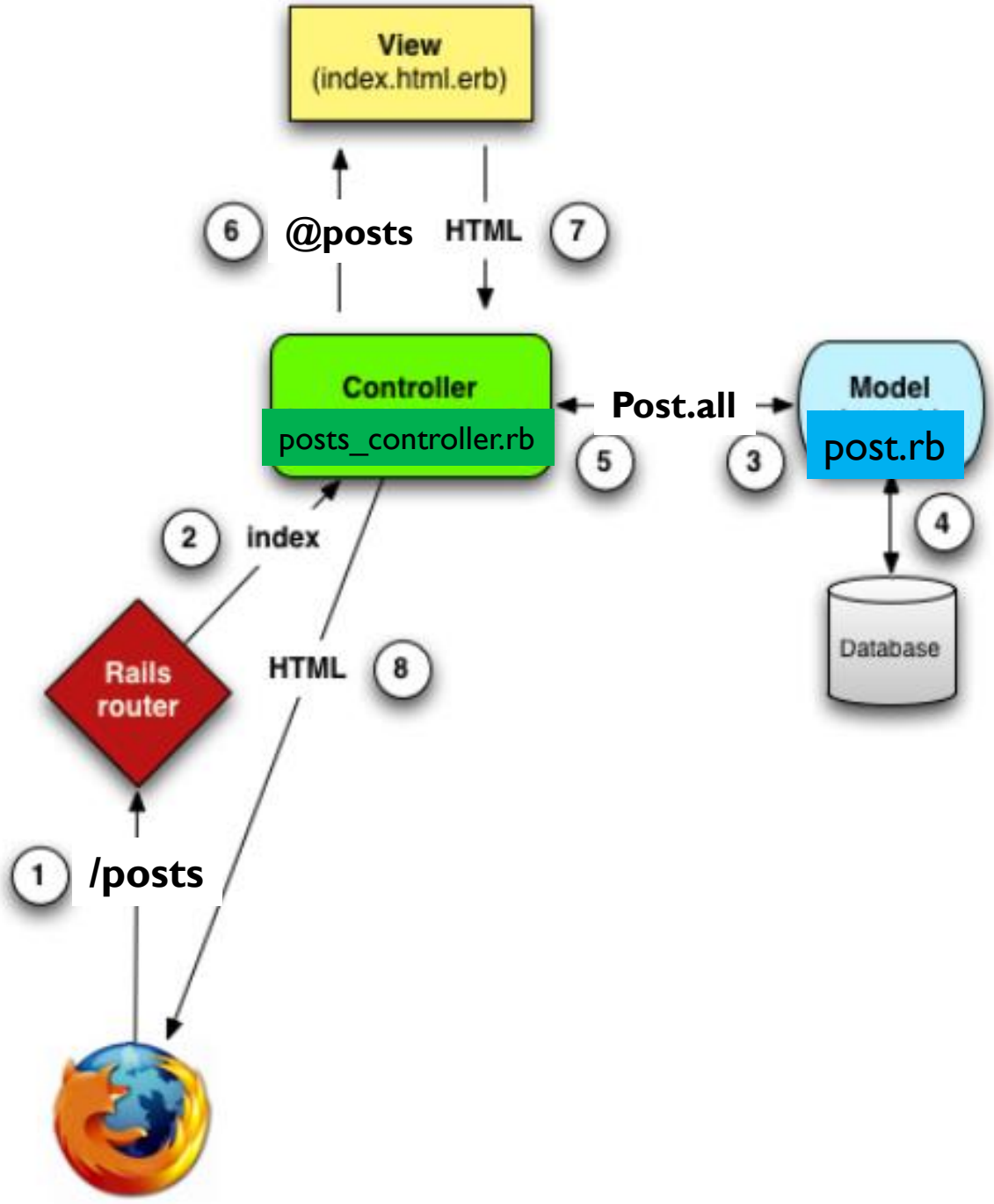
- resources :classes

```
Rails.application.routes.draw do

  resources :posts

  get 'home/index'

  root :to => "home#index"

  # For details on the DSL available
guides.rubyonrails.org/routing.html
end
```

HTTP request	URL	Controller	Action (method)	Process
GET	/posts	Post	index	Page to show all posts
GET	/posts/1	Post	show	Page to show the post whose ID is 1
GET	/posts/new	Post	new	Page to create a new post
GET	/posts/1/edit	Post	edit	Page to edit the post whose ID is 1
POST	/posts	Post	create	To create a new post
PATCH	/posts/1	Post	update	To update ID=1 post
DELETE	/posts/1	Post	destroy	To delete ID=1 post

Methods for URLs

Standard Methods	HTTP request	URL	Action (method)
posts_path	GET	/posts	index
post_path(:id)	GET	/posts/:id	show
new_post_path	GET	/posts/new	new
edit_post_path(:id)	GET	/posts/:id/edit	edit
	POST	/posts	create
	PATCH	/posts/:id	update
	DELETE	/posts/:id	destroy


View
(index.html.erb)
6
@posts
HTML
7
Controller
posts_controller.rb
Post.all
Model
post.rb
5
3
4
2
index
Database
Rails
router
HTML
8
1
/posts

Steps to show './posts' in MVC

- 1. Http Request: http://localhost:3000/posts
- 2. Route file: routes.rb receive request:
  - Request: Get, Controller: Post, Action: Index
- 3. Controller: posts_controller.rb
  - Method Index: @posts = Post.all
- 4. Model: post.rb
  - Post.all retrieve information in sqlite database
- 5. View: index.html.erb
  - <% @posts.each do |post| %> show all posts
  - Layout file: app/views/layouts/application.html.erb

Display an individual post

- http://localhost:3000/posts/1
- Where is the code that shows post?
  - Http Request: Get /posts/1
  - Controller: posts_controller, Action: Show
    - before_action :set_post, only: [:show, :edit, :update, :destroy]
    - def set_post : @post = Post.find(params[:id])
  - Model: @post
  - View: show.html.erb

Web page to edit posts

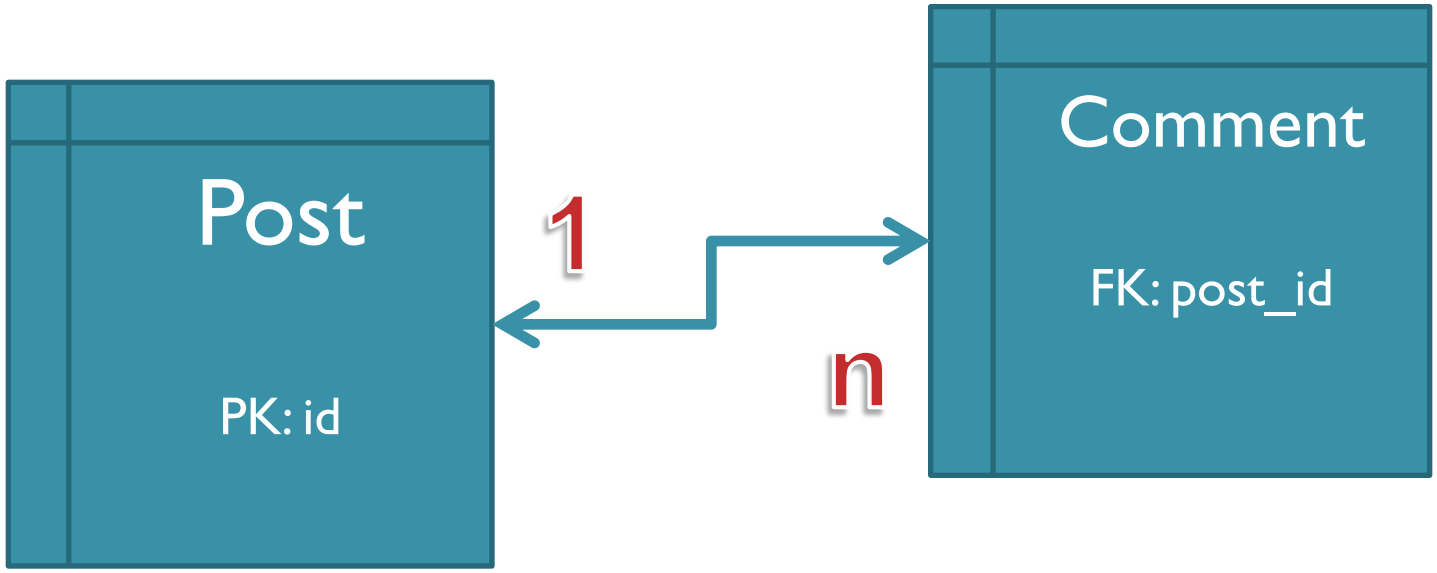
- http://localhost:3000/posts/1/edit
- Where is the code?
  - Http Request: Get /posts/1/edit
  - Controller: posts_controller, Action: edit
  - View: edit.html.erb
    - <%= render 'form', post: @post %>
    - form view: _form.html.erb
- To Update
  - HTML: <form class="edit_post" … action="/posts/1" …method="post">
  - Http Request: post /posts/1

Creating new posts

- http://localhost:3000/posts/new
- Where is the code?
  - Http Request: Get /posts/new
  - Controller:
  - View: new.html.erb
    - <%= render 'form', post: @post %>
    - form view: _form.html.erb
- To Create
  - HTML: <form … action="/posts" … method="post">
  - Http Request: post /posts

Add comment page

- Models



Add comment page (cont'd)

- Generate controller and model
  - *rails generate controller comment show*
  - *rails generate model Comment content:string post_id:integer*
  - *rake db:migrate*

- Establish tables' relations in Models
  - Class Post : ***has_many :comments***
  - Class Comment: ***belongs_to :post***

Add comment page (cont'd)

- Route
  - *match '/comment', to: 'comment#show', via: 'get'*
  - *post 'comment/create'*

- Link to comment page
  - Post index.html.erb:
  - *<td><%= link_to 'Comments', comment_path(postid: post.id) %></td>*

Comments List

- Comment Controller
  - *def show*
  - *@post = Post.find(params[:postid])*
  - *@comments = @post.comments unless @post.nil?*
  - *end*
- Comment View
  - *<ul id="comments_list">*
  - *<% @comments.each do |com| -%>*
  - * <%= com.id -%> | <%= com.content -%> | <%= com.created_at -%> *
  - *<% end -%>*
  - *
*

Add Comment

- Comment View
  - *<%= form_for :newcomment, url: {action: "create"} do |f| %>*
  - *…*
  - *<% end %>*
- Comment Controller
  - *def create*
  - *@newcomment = Comment.new(comment_params)*
  - *@newcomment.save*
  - *redirect_to comment_path(:postid => @newcomment.post_id)*
  - *end*

Thank You