

---

# Using Web Frameworks

---

An introduction to Rails

---

# Lecture Goals

---

- Understand what frameworks are used for.
  - Know how to design an application with a web framework in mind.
  - Have a ***vocabulary*** to talk about different parts of web frameworks.
  - Understand some of the fundamental ***design patterns*** used by frameworks.
  - Know how to navigate a Rails project.
-

# Web Requests

---

When you type a URL into your browser, a ***client***, your computer issues a ***web request***.

A ***server*** on the other side handles processing your request and builds a ***response***.

---

# Web Requests

---



- ***HyperText Transfer Protocol*** (HTTP) is the language used to issue and respond to web requests.
-

# HyperText Transfer Protocol (HTTP)

---

HTTP contains **verbs** for distinguishing between different types of requests:

- GET - request a web page  
e.g. GET `http://www.reddit.com/r/todayilearned`
  - POST - send some resource data to a website  
e.g. make a new post on reddit
  - PUT - update a resource on a website  
e.g. make an edit to a post
  - DELETE - remove a resource from a website  
e.g. delete a post
-

# State and HTTP

---

Imagine having a conversation with someone who never remembers anything you've already said.

This communication is ***stateless***, *i.e.*, no context is remembered.

Need a way to remind people of what was spoken before.

---

# State and HTTP

---

HTTP is a stateless protocol. We use ***cookies***, ***URL variables***, as well as other methods to save state.

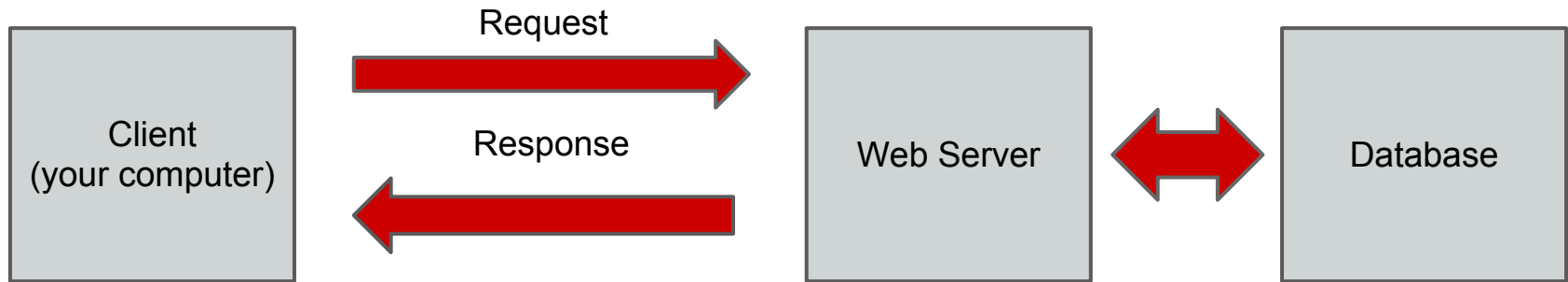
***State*** is any *stored* information that may change over time.

`http://www.google.com/search?q=reddit;`

---

# Databases

---



Application servers keep track of state (e.g. note accounts, forum posts, likes) by ***Creating, Reading, Updating and Deleting (CRUD)*** data stored as ***records*** in a ***database***.

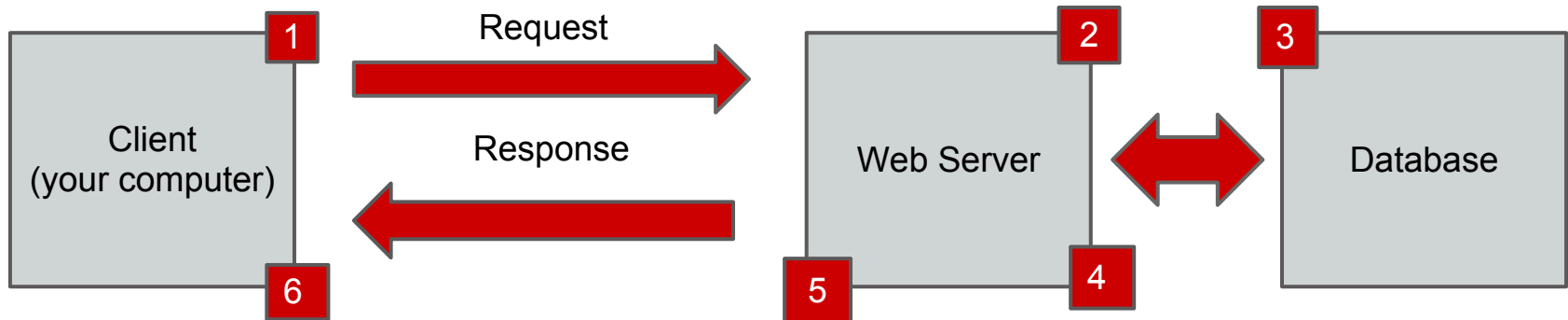
Stored information enables websites like Amazon.com to remember what you've previously bought.

---



# Generic Web Framework Pattern

---



notes issue web requests to a server, and that server in turn looks up and (maybe) changes data stored in the database.

The server then sends a response to the client, possibly displaying the new data.

---

# Responding to Web Requests

---

A ***Restful*** API (a design pattern) is a way to design your URLs to give web requests a standardized form.

Restful APIs create a mapping between HTTP Verbs, ***Universal Resource Identifiers (URIs)***, and ***actions*** taken by the web server to satisfy requests.

API = ***Application Programmer's Interface***

---

# Restful URLs (API)

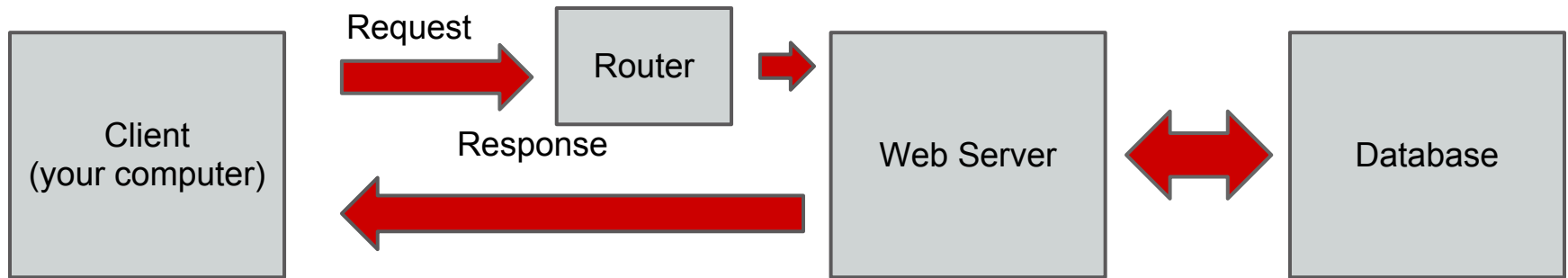
---

Example:

| HTTP Verb | URI      | Action  | Purpose                      |
|-----------|----------|---------|------------------------------|
| GET       | /notes   | index   | List all notes on website    |
| GET       | /notes/1 | show    | Display details about note 1 |
| POST      | /notes   | create  | Create a new note            |
| PUT       | /notes/1 | update  | Update note 1                |
| DELETE    | /notes/1 | destroy | Delete note 1                |

# Responding to Web Requests

---



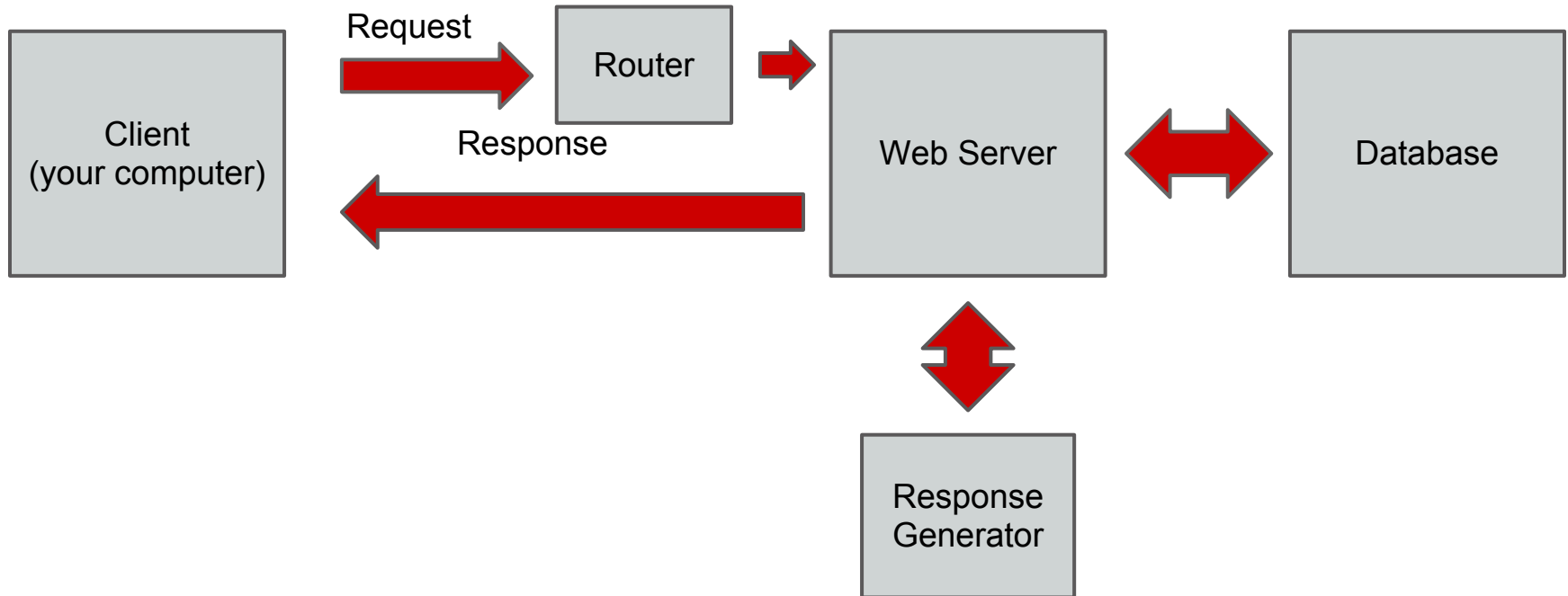
Web requests are differentiated by which HTTP verb and which URL was used.

Web servers use a ***router*** to decide what ***action*** to take in response to a particular request.

---

# Building Responses

---



The last step in handling a request is building a ***response***.

---

# Responses

---

Responses come in many forms:

- Web Page (HTML/CSS)
  - Dynamic Content (JS/PHP)
  - Data (JSON/XML)
-

# HTML/CSS Responses

---

- Often, responses will be partly note-specific, partly static.
  - ***Templates*** allow us to write the static parts of web pages once, and fill in the note-specific parts later.
-

# HTML Templates

---

```
<html>
  <body>
    <h1>Welcome! <%= note.name %></h1>

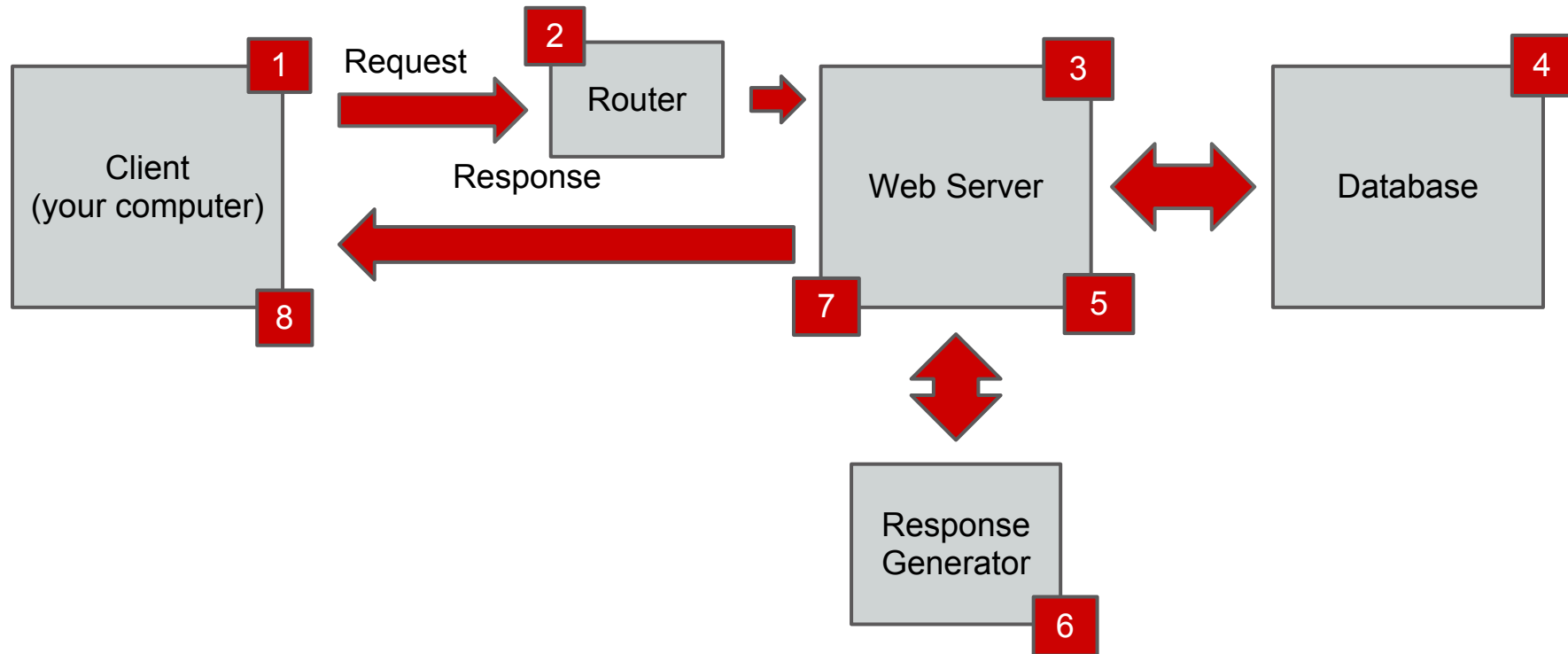
    <p>Our site allows you to....</p>
  </body>
</html>
```

---



# Putting it all together...

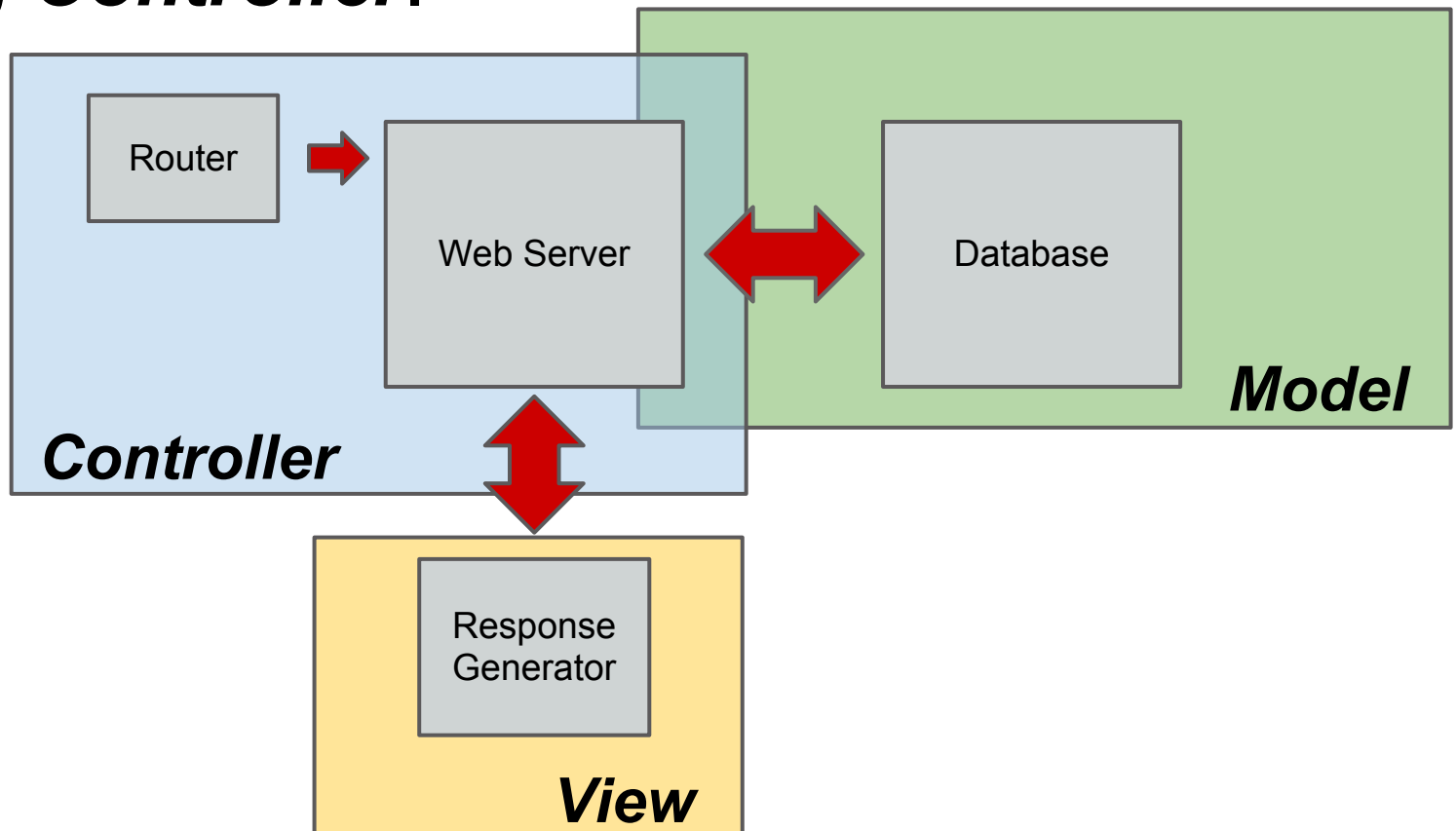
---



# Putting it all together...

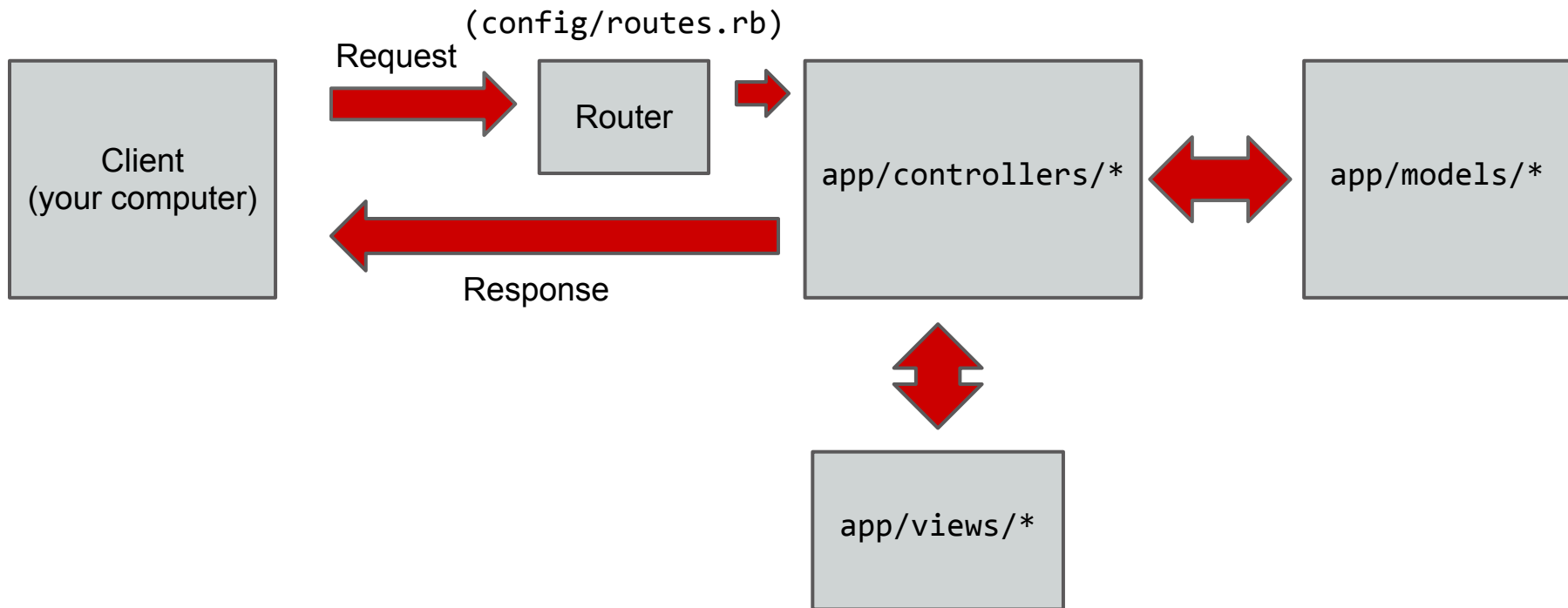
---

The common name for this pattern is ***Model, View, Controller***.



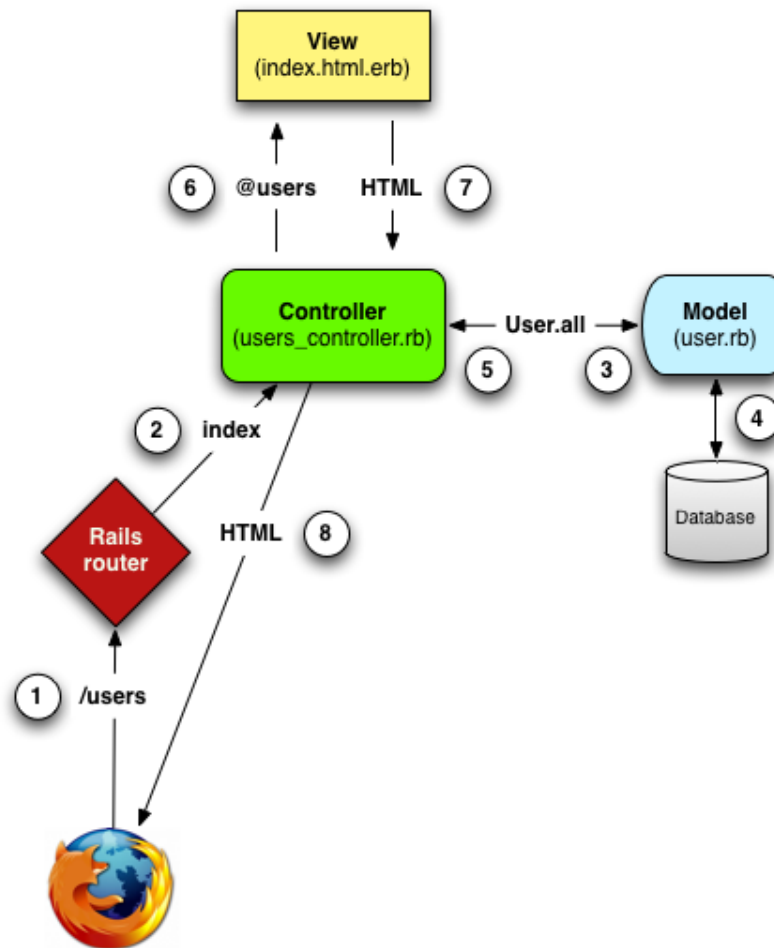
# MVC in Rails

---



# MVC in Rails

---



# rails start sample\_app

---

```
$ rails start <app_name>
```

Creates a directory <app\_name>, with several folders beneath to help you organize all of your code.

**Note:** '\$' means a command prompt.

---

# rails start sample\_app

---

```
$ ls sample_app
```

|              |          |           |        |        |
|--------------|----------|-----------|--------|--------|
| Gemfile      | Rakefile | config.ru | lib    | script |
| Gemfile.lock | app      | db        | log    | test   |
| README.rdoc  | config   | doc       | public | tmp    |

```
$ ls sample_app/app
```

|        |             |         |         |        |
|--------|-------------|---------|---------|--------|
| assets | controllers | helpers | mailers | models |
| views  |             |         |         |        |

# Starting the Server

---

```
$ rails s[erver]
```

More information:

```
$ rails server --help
```

**Note:** [ ] = optional text

---

# Static Pages

---

Type [localhost:3000](http://localhost:3000) into your browser.

Currently, this shows the file `public/index.html`

Create the page `public/hello.html`:

```
<html>
  <body>
    <h1>Hello World!</h1>
  </body>
</body>
```

Can be viewed at [localhost:3000/hello.html](http://localhost:3000/hello.html)



# Static Pages

---

Limitations:

- No templating
  - Truly static (can't have any note-specific content).
-

# A Static Pages Controller

---

```
$ rails g[enerate] controller StaticPages home about
      create  app/controllers/static_pages_controller.rb
      route   get "static_pages/about"
      route   get "static_pages/home"
      invoke  erb
      create   app/views/static_pages
      create   app/views/static_pages/home.html.erb
      create   app/views/static_pages/about.html.erb
      ...
```

This creates our first controller. This will enable us to make use of templates, which will make it much easier to add site-wide content to our site (e.g. a navigation bar).

---

# Static Pages

---

**File:** `app/views/static_pages/about.html.erb`

`.erb` stands for ***embedded ruby***. It enables us to write ruby code inside out HTML pages, which allows us to add dynamic content (more on this later).

---

# Aside: The Asset Pipeline

---

html.erb

css.scss

js.coffee

Assets are ***compiled*** (e.g. translated into a new form). The file-types determine which kinds of compilation are attempted. Files are read from right to left (so .html.erb is first run through the embedded ruby compiler).

In production, this enables us to do JS + CSS ***minification***, a process by which files are made smaller so that they take less time to send to the client.

---

# Routes

---

**File:** config/routes.rb

```
...  
root to: 'static_pages#home'  
match 'about', to: 'static_pages#about'  
...
```

These lines make [localhost:3000](http://localhost:3000) point to app/views/static\_pages/home.html.erb and [localhost:3000/about](http://localhost:3000/about) point to app/views/static\_pages/about.html.erb.

**Note:** need to delete public/index.html

---

# Adding a new static page

---

**File:** `app/controllers/static_pages_controller.rb`

```
def time  
end
```

**File:** `config/routes.rb`

```
match 'time', to: 'static_pages#time'
```

---

# Adding a new static page

---

Go to [localhost:3000/time](http://localhost:3000/time), no page yet!  
(template is missing).

**File:** `app/views/static_pages/time.html.erb`

```
<h1>The time is now: </h1>
```

---

# Adding a template

---

**File:** `app/view/static_pages/time.html.erb`

`<h1>The time is now: <%= Time.now %></h1>`

But, this isn't very pretty... We need a way to ***format*** the time to make it more readable.

---



# Adding dynamic content

---

```
$ rails c[onsole]
> Time.now
> Time.now.strftime
```

Documentation: <http://www.ruby-doc.org/core-2.0/>

**File:** app/views/static\_pages/time.html.erb

```
<h1>The time is now: <%= Time.now.strftime  
('%B %d, %Y: %H:%M:%S') %></h1>
```

---

# Adding dynamic content

---

**File:** app/views/static\_pages/time.html.erb

```
<h1>The time is now:</h1>
```

```
<h2>
```

```
  <%= Time.now.strftime('%B %d, %Y: %H:%M:%S')  
  %>
```

```
</h2>
```

---

# Gemfile

---

A Gemfile enables you to specify the ***dependencies*** (other software libraries) that your code uses. In ruby, packages of software can be installed as ***gems***.

---

# Adding styling with Bootstrap

---

**File:** Gemfile

```
group :assets do
  ...
  gem 'bootstrap-sass', '2.1'
  ...
end
```

---

# Adding styling with Bootstrap

---

Install new gems added to Gemfile:

```
$ bundle install
```

**File:** app/assets/stylesheets/custom.css.scss

```
@import "bootstrap"
```

**Note:** server restart needed

---

# Styling with Bootstrap

---

**File:** app/views/static\_pages/time.html.erb

```
<div class="container">
  <div class="hero-unit">
    <h1>The time now is:</h1>
    <h2 class="centered">...</h2>
  </div>
</div>
```

**File:** app/assets/stylesheets/custom.css.scss

```
.centered {
  text-align: center;
}
```

---

# Aside: The Asset Pipeline

---

For ***development***, we want to make sure that our files are easy to read in the browser. For ***production***, we want files to be as small as possible, even if they aren't readable.

```
$ rails s -e production
$ rake assets:precompile
$ rails s -e production
```

**File:** config/environments/production.rb

```
config.assets.compile = true
```

---

# Aside: The Asset Pipeline

---

```
$ rake assets:precompile  
rails s -e production
```

Now, all of our JS and CSS are compiled into just two files, that are basically impossible to read (this has the added benefit of somewhat protecting intellectual property (IP)).

---



# Partials

---

**File:** `app/views/layouts/application.html.erb`

This file gives us structure for the entire site.  
Add this line:

```
<%= render 'layouts/header %>
```

**File:** `app/views/layouts/_header.html.erb`

```
<%= render 'layouts/header %>
```

---

# Header Partial

---

**File:** app/views/layouts/\_header.html.erb

(see: <http://twitter.github.com/bootstrap/examples/starter-template.html>)

```
<div class="navbar navbar-inverse navbar-fixed-top">
  <div class="navbar-inner">
    <div class="container">
      <div class="nav-collapse collapse">
        <ul class="nav">
          <li><a href="/">Home</a></li>
          <li><a href="/time/">Time</a></li>
        </ul>
      </div>
    </div>
  </div>
</div>
```

---

# Cleaning up home

---

What do we edit to make [localhost:3000](http://localhost:3000) look better?

Where do we add it?

```
<div class="container">  
  <div class="hero-unit">  
    <%= yield %>  
  </div>  
</div>
```

**File:** app/views/layouts/application.html.erb

---

# Adding models

---

We're going to make a simple note taking app. It allows notes to leave notes on a page.

notes can go look at notes, edit specific ones, or leave more.

---

# Sketching out Notes

---

A note has:

- a. Title (string)
- b. Content (string)
- c. Author (string (for now...))

**Form:** rails g [model\_name] [attr]:[data\_type]

```
$ rails g model Note title:string content:string author:string
  invoke  active_record
  create   db/migrate/20130405164422_create_notes.rb
  create   app/models/note.rb
  invoke   test_unit
  create   test/unit/note_test.rb
  create   test/fixtures/notes.yml
```

---

# Migrations

---

```
db/migrate/20130405164422_create_notes.rb
```

Migration files allow us to write ruby code to manipulate our database. This saves us from having to learn a specific syntax for a specific database, and enables us to switch between different databases easily.

sqlite3 is used in development by default

---

# The Notes Model

---

**File:** `app/models/note.rb`

**File:** `config/routes.rb`

`resources :notes`

The resources line gives us some default URLs for manipulating notes. Notice we wrote `:notes` instead of `:note`. In general, Ruby allows us to use pluralization where it is appropriate.

---

# Notes Routes

---

Shows all URIs your router responds to:  
\$ rake routes

...

|           |        |                           |               |
|-----------|--------|---------------------------|---------------|
| notes     | GET    | /notes(.:format)          | notes#index   |
|           | POST   | /notes(.:format)          | notes#create  |
| new_note  | GET    | /notes/new(.:format)      | notes#new     |
| edit_note | GET    | /notes/:id/edit(.:format) | notes#edit    |
| note      | GET    | /notes/:id(.:format)      | notes#show    |
|           | PUT    | /notes/:id(.:format)      | notes#update  |
|           | DELETE | /notes/:id(.:format)      | notes#destroy |

...

---



# Creating Notes

---

```
$ rails c[onsole]
```

```
> n = Note.new # should cause a DB error
```

Need to first create Notes table:

```
$ rake db:migrate
```

```
$ rails c
```

```
> n = Note.new
```

```
> n = Note.new title: "Hello, There", author:  
"Samuel Messing", content: "Lorem ipsum  
dolor..."
```

```
> n.save
```

---

# Model Validations

---

**File:** app/models/note.rb

```
validates :title, presence :true
```

```
before_save do |note|  
  note.title = note.title.titlecase  
end
```

```
$ rails c
```

```
> n = Note.new title: "MY AWESOMEST NOTE"
```

```
> n.save
```

```
> n.title
```

```
=> "My Awesomest Note"
```

---

# Notes URLs

---

Example:

| HTTP Verb | URI      | Action  | Purpose                      |
|-----------|----------|---------|------------------------------|
| GET       | /notes   | index   | List all notes on website    |
| GET       | /notes/1 | show    | Display details about note 1 |
| POST      | /notes   | create  | Create a new note            |
| PUT       | /notes/1 | update  | Update note 1                |
| DELETE    | /notes/1 | destroy | Delete note 1                |

---

# Notes Controller

---

Why does [localhost:3000/notes](http://localhost:3000/notes) not currently work?

No controller! (The part of the server that actually builds the response).

**Form:**

```
rails generate controller <ControllerName>
```

---

# Notes Controller

---

```
rails g controller Notes
      create app/controllers/notes_controller.rb
      invoke erb
      create app/views/notes
      invoke test_unit
      create test/functional/notes_controller_test.rb
      invoke helper
      create app/helpers/notes_helper.rb
      invoke test_unit
      create test/unit/helpers/notes_helper_test.rb
      invoke assets
      invoke coffee
      create app/assets/javascripts/notes.js.coffee
      invoke scss
      create app/assets/stylesheets/notes.css.scss
```

---

# Notes Controller

---

Now [localhost:3000/notes](http://localhost:3000/notes) gives us an action undefined error. Need to add the action "index" to the notes controller.

**File:** app/controllers/notes\_controller.rb

```
def index
end
```

---

# Notes Index Template

---

Now [localhost:3000/notes](http://localhost:3000/notes) gives us a template not found error. Need to add an index template to app/views/notes.

**File:** app/views/notes/index.html.erb

```
<ul class="notes-list">
  <% @notes.each do |note| %>
    <h1><%= note.title %></h1>
    <h2>By <%= note.author %></h2>
    <hr noshade/>
    <p><%= note.content %></p>
  <% end %>
</ul>
```

---

# Passing Notes into the template

---

Need to define `@notes` for the template. This is done in the controller.

**File:** `app/controllers/notes_controller.rb`

```
def index
  @notes = Note.all
end
```



# Adding a form for new notes

---

```
$ rake routes
```

Visiting [localhost:3000/notes/new](http://localhost:3000/notes/new) gives us errors.

How do we fix the first error?

Once it's fixed, how do we fix the second?

---

# A form to make new notes

---

File: app/views/notes/new.html.erb

```
<%= form_for(@note) do |f| %>
  <%= f.label :title %>
  <%= f.text_field :title %>

  <%= f.label :author %>
  <%= f.text_field :author %>

  <%= f.label :content %>
  <%= f.text_field :content %>

  <%= f.submit "Create note", class: "btn btn-large
btn-primary" %>
<% end %>
```

---

# Controller Updates

---

**File:** app/controllers/notes\_controller.rb

```
def create
  @note = Note.new(params[:note])
  if @note.save
    # do something...
  else
    render 'new'
  end
end
```

---

# Adding Error Feedback

---

```
$ rails c  
> n = Note.new  
> n.save  
> n.errors
```

Rails by default adds an errors object to models that have failed to be saved. This enables us to give feedback to a user trying to make a new note.

---

# Adding Error Feedback

---

**File:** app/views/shared/\_error\_messages.html.erb

```
<% if @note.errors.any? %>
  <div id="error_explanation">
    <div class="alert alert-error">
      The form contains <%= pluralize(@note.errors.
count, "error") %>.
    </div>
    <ul>
      <% @note.errors.full_messages.each do |msg| %>
        <li>* <%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>
```

---

# Showing individual notes

---

We want to go to [localhost:3000/notes/<note\\_id>](http://localhost:3000/notes/<note_id>) (e.g. [localhost:3000/notes/3](http://localhost:3000/notes/3)). Right now this doesn't work.

```
File: app/controllers/notes_controller.rb  
  def show  
    @note = Note.find(params[:id])  
  end
```

What have we forgotten to do?

---

# Note template

---

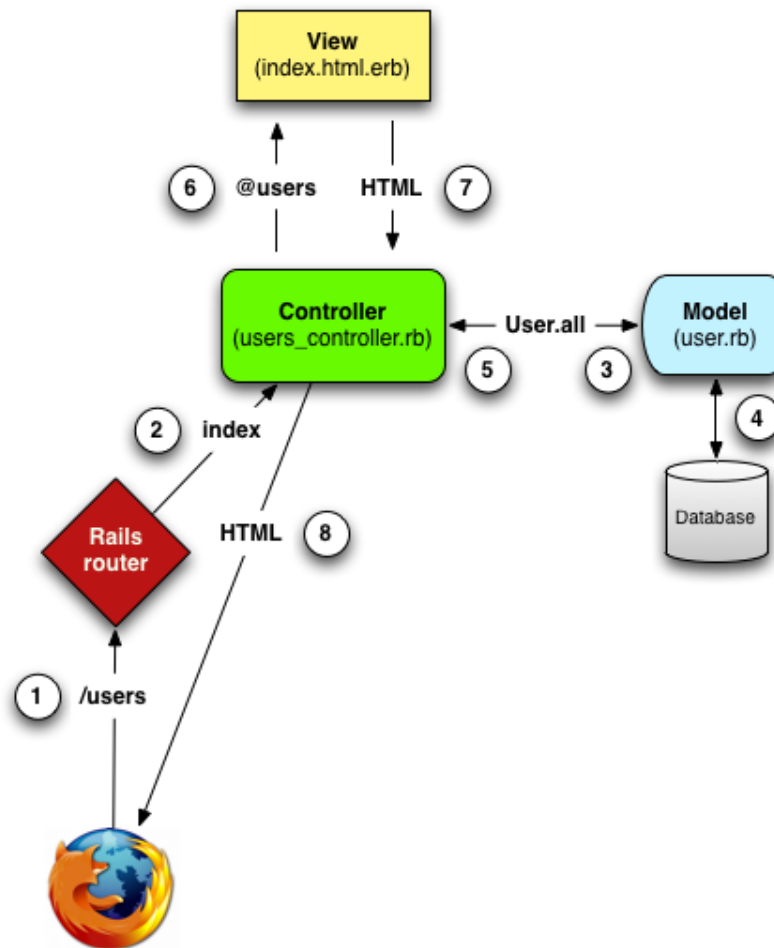
**File:** app/views/notes/show.html.erb

```
<h1><%= @note.title %></h1>
<h2>by <%= @note.author %></h2>
<hr noshade/>
<p><%= @note.content %></p>
```

---

# MVC in Rails

---





# Deleting Notes

---

**File:** app/views/notes/index.html.erb

```
<%= link_to("Delete", note, method: delete,  
class: 'action') %>
```

---

# Deleting Notes (cont'd)

---

**File:** app/controllers/notes\_controller.rb

```
def destroy
  @note = Note.find(params[:id])
  @note.destroy
  @notes = Notes.all
  render 'index'
end
```

---

# Not Covered

---

- Testing (this is HUGE)
  - Security (ditto)
  - Adding client-side code (JS)
  - Handling data requests (JSON/XML)
  - Deployment (AWS, Heroku)
  - Version Control (git, subversion, etc.)
-

# Resources

---

RailsCast

<http://ruby.railstutorial.org/>

---