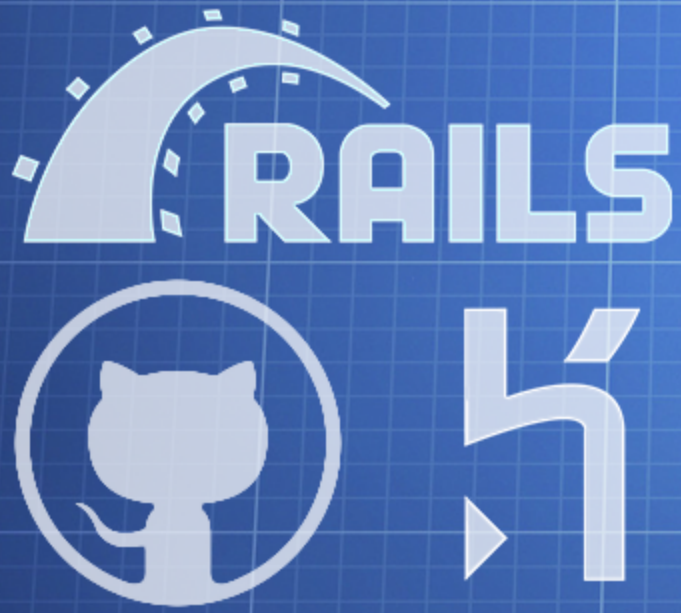
Zero to Deploy: The Basics

May 2017

at ShiftKey Labs

Zero to Deploy: The Basics

Use these tools



Ruby On Rails | GitHub | Heroku

Deploy your app

Hosted by: DevWORKS

PLANNING FOR THE WORKSHOP

https://github.com/DalDevWorks/zero_to_deploy_workshop

PLANNING FOR THE WORKSHOP

- Brief intro to the web stack
- Initialize our address book application
- Version control with Git and GitHub
- Create our first view

PLANNING FOR THE WORKSHOP

- Deploy our application
- Create the actual address book

THE WEB STACK

WEB STACK

- Various definitions
- Developers have different opinions
- We will keep it simple:
  - Front-end development
  - Back-end development

WEB STACK

FRONT-END DEVELOPMENT

BACK-END DEVELOPMENT

FRONT-END DEVELOPMENT

BACK-END DEVELOPMENT

FRAMEWORKS

DATA MANAGEMENT

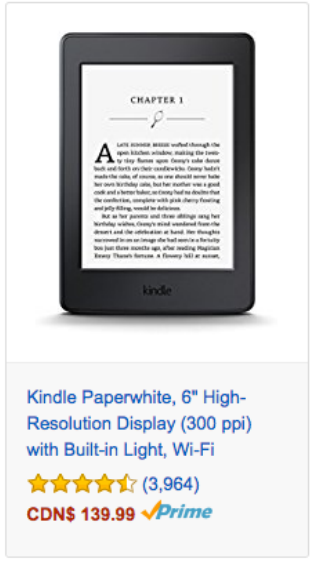
WEB STACK

FRONT-END DEVELOPMENT

BACK-END DEVELOPMENT

FRONT-END DEVELOPMENT

- Visual representation of your application
- What the user interacts with



HTML and CSS to display an Amazon Product

BACK-END DEVELOPMENT

- Data and logical management of your application
- How you manage user actions

```
{"product": {
  "id": 1233452,
  "name": "Kindle Paperwhite(...)",
  "price": 139.99,
  "rating": 4.5,
  "reviews": 3964,
  "prime": true
}}
```

Data sent from the server to the front-end when the user searched “Kindle Paperwhite”

GETTING STARTED

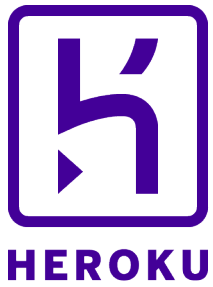
OUR GOALS

- Build a simple address book
- Store information about your contacts, such as: name, address, phone number and notes.
- Manage your contacts (create, read, update, delete)
- Available online (i.e. hosted)

OUR TOOLS

FRONT-END

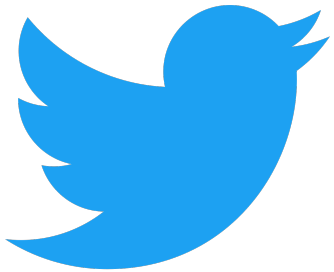
BACK-END



RAILS IN A NUTSHELL

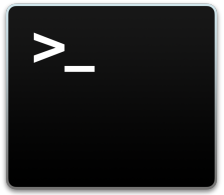
- Released in 2005
- Active open source community and supported by companies like Shopify
- The backend code is written in Ruby
- The framework comes with several tools and packages built-in which are named ‘gems’

APPS BUILT WITH RAILS



TOP OF EACH SLIDE WILL HAVE...

- Where you should be typing:



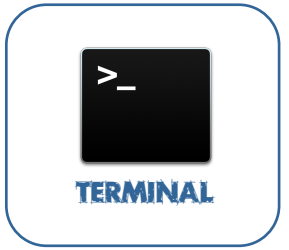
TERMINAL

TEXT EDITOR / IDE

- The file path we are working on:

```
# Application view
app/views/layouts/application.html.erb
```

LET'S GET STARTED



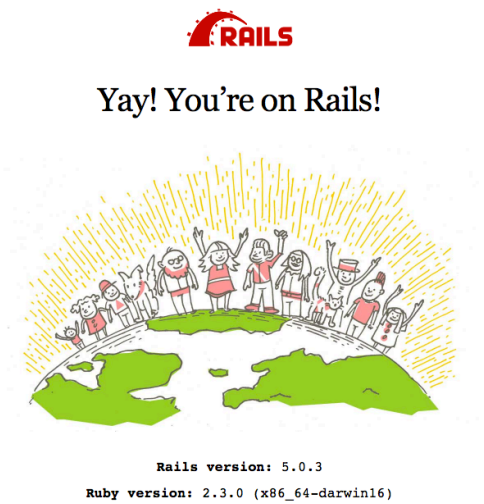

```
# Go to home directory [Optional]
$ cd ~

# Create a workspace folder [Optional]
mkdir Developer
$ cd ~/Developer/

# Create our Rails application in your preferred workspace folder
$ rails new address_book
```

START OUR APP

TERMINAL

~/Developer/address_book

```
# Launch rails app
$ cd address_book
$ rails server

# In your browser, enter `localhost:3000` in the URL bar
```



If all went well you should see these happy Rails developers

MVC PATTERN

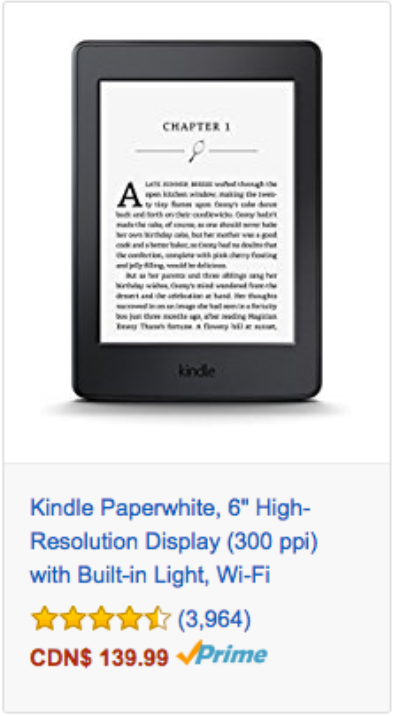
- Model – View – Controller
- Design pattern used by most modern web applications
- Internal information vs. user facing information
  - Model – Controller (internal)
  - View (user facing)

MODEL

- How the data is represented within the application
- An ‘object’ that represents data stored in the database
- The model determines the requirements and validations of an application object (called **resources** in Rails).

MODEL EXAMPLE: AMAZON PRODUCT

- A product resource has the following attributes:
  - Name
  - Rating
  - Number of reviews
  - Price
  - Is *Prime* available (yes/no?)



MODEL EXAMPLE: AMAZON PRODUCT

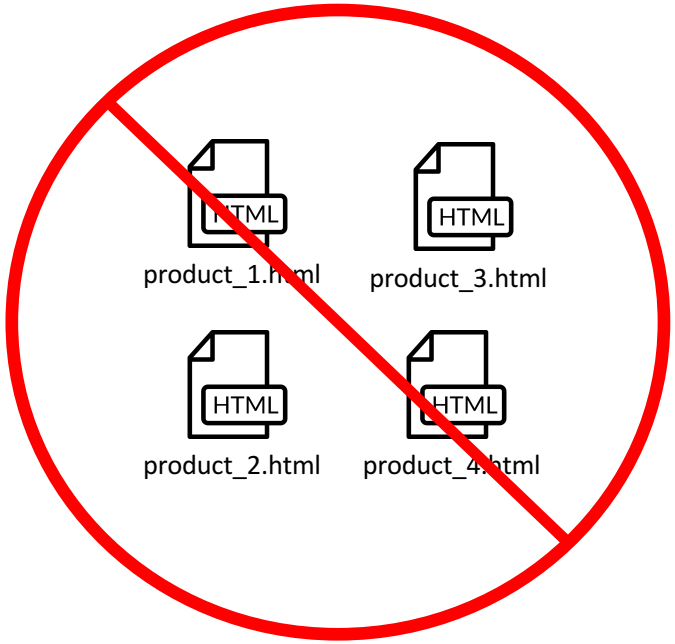
- Product:
  - Name : string
  - Rating : float
  - Number of reviews : integer
  - Price : float
  - Prime : boolean


```
{"product": {
  "id": 1233452,
  "name": "Kindle Paperwhite(...)",
  "price": 139.99,
  "rating": 4.5,
  "reviews": 3964,
  "prime": true
}}
```


Example of the product object (JSON)

VIEW

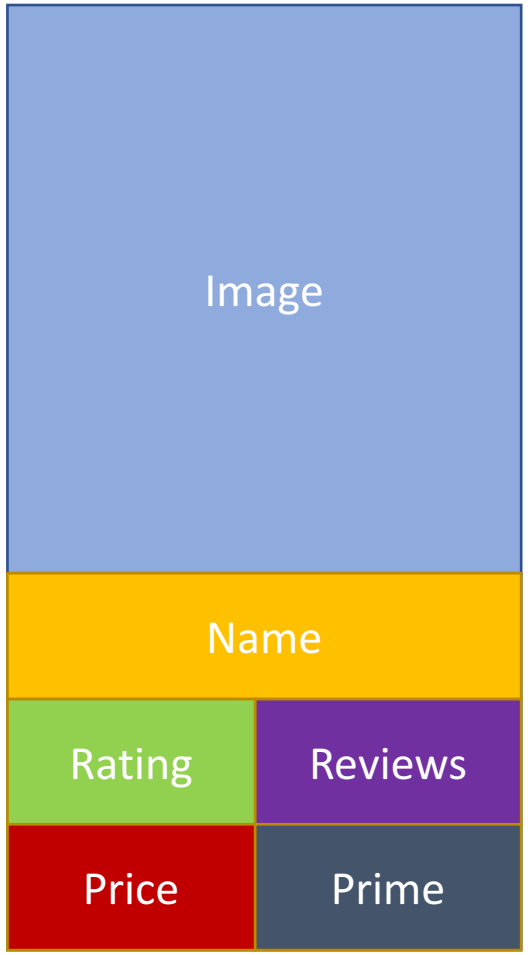
- The visual representation of data to the user
- Generic templates to display data
- Why use generic templates?



No need to create a view for *each* product

One template to represent a resource

```
{"product": {
  "id": 1233452,
  "name": "Kindle Paperwhite(...)",
  "price": 139.99,
  "rating": 4.5,
  "reviews": 3964,
  "prime": true
}}
```

Retrieve the data for a given resource

Render the data in the HTML template

Consistent display for all products

HOW DO YOU ASSOCIATE A MODEL WITH A VIEW?

WITH CONTROLLERS

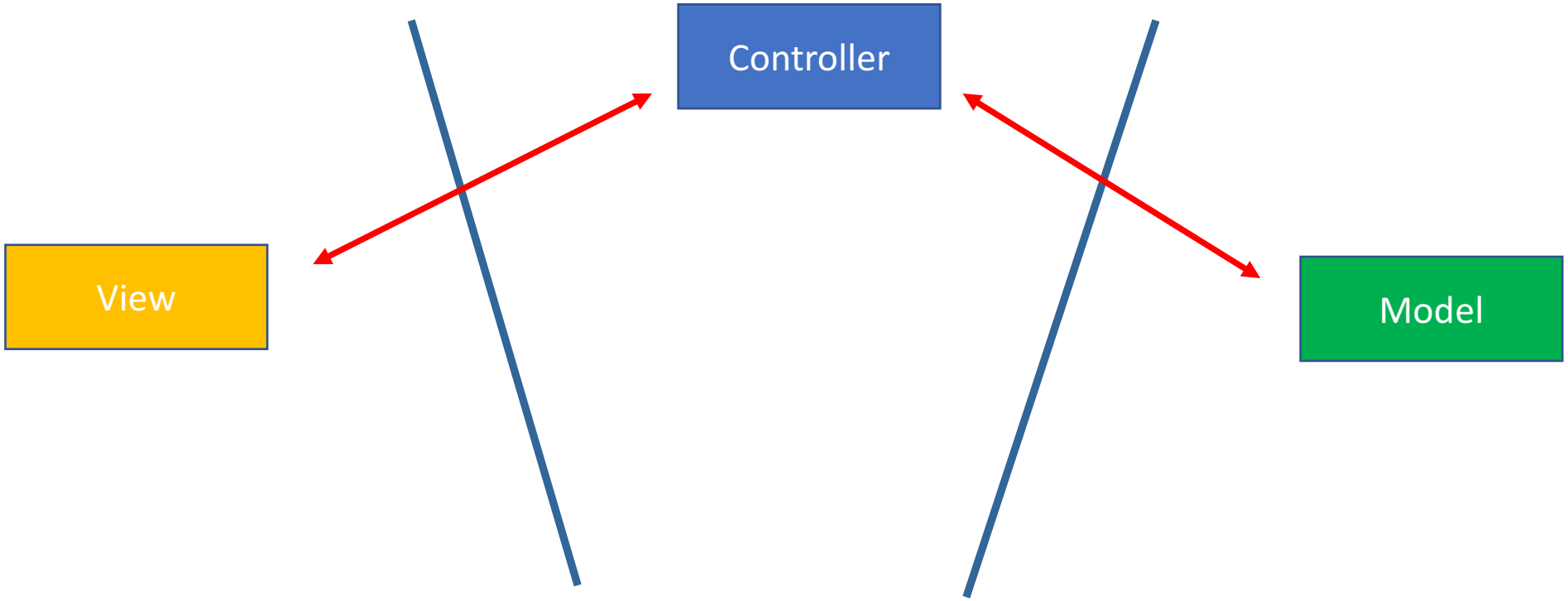
CONTROLLER

A controller maps a user action to an application action

- **User action**:
  - GET all e-readers (by clicking on a filter link)

- **Application action**:
  - Fetch from database all e-readers from the *Products* table
  - Return the necessary data and template to represent that data

CONTROLLER IN ACTION



Important: The view and the model never interact with each other

OK, THAT'S A LOT – LET'S WRITE SOME CODE.

OUR PLAN:

- Create a landing page view
- Create a controller that will handle the root route (i.e. ‘/’)
- Add version control
- Deploy our app

RAILS APP FILE STRUCTURE

/app

- /controllers
- /models
- /views

RAILS APP FILE STRUCTURE

/app

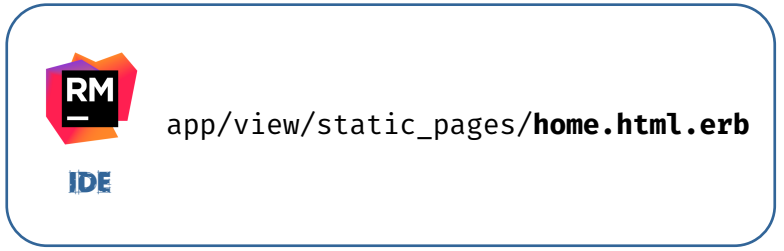
/controllers

/models

/views

/static_pages ← Create this directory

home.html.erb ← Create this file




```
<h1>Rails Address Book</h1>
<h3>Contacts for [Your name]</h3>
<button>Open</button>
```

/app

/controllers

static_pages_controller.rb ← Create this file

/models

/views

app/controllers/static_pages_controller.rb

```ruby
class StaticPagesController < ApplicationController
  def home
  end
end
```

config/routes.rb

```ruby
Rails.application.routes.draw do

  # Set the root of the application
  root 'static_pages#home'

end
```

CHECK IN THE BROWSER

WHAT'S GOING ON IN BACKGROUND

```
Started GET "/" for 127.0.0.1 at 2017-05-27 11:49:41 -0300
Processing by StaticPagesController#home as HTML
  Rendering static_pages/home.html.erb within layouts/application
  Rendered static_pages/home.html.erb within layouts/application (0.3ms)
Completed 200 OK in 184ms (Views: 183.0ms)
```

GIT AND GITHUB

GITHUB

- Head over to GitHub
- Create a new repository

BASIC GIT


```
# Stage all the files
git add –A

# If you get an error do $ git init

# Name the commit
git commit -m "added new landing page"

# Add a remote and push to the repository
git remote add origin git@github.com:ericdesj/address_book.git
git push --set-upstream origin master

# Refresh the repository in your browser and notice all your files are there
```

DEPLOY TO HEROKU

ENABLING SSL

config/environments/production.rb

```
# In config/environments/production.rb
# Uncomment line 48 (remove the '#').
config.force_ssl = true
```

CONFIGURE THE SERVER

config/puma.rb

```
workers Integer(ENV['WEB_CONCURRENCY'] || 2)
threads_count = Integer(ENV['RAILS_MAX_THREADS'] || 5)
threads threads_count, threads_count

preload_app!

rackup      DefaultRackup
port    ENV['PORT']  || 3000
environment ENV['RACK_ENV'] || 'development'

on_worker_boot do
  # Worker specific setup for Rails 4.1+
  # See: https://devcenter.heroku.com/articles/deploying-rails-applications-with-the-puma-web-server#on-worker-boot
  ActiveRecord::Base.establish_connection
end
```

Copy the code from here: https://devcenter.heroku.com/articles/deploying-rails-applications-with-the-puma-web-server

UPDATE OUR GEMFILE

/Gemfile

```
# Find and remove the ‘sqlite3’ gem (delete the whole line and the comment above it)


# Add the gem to the :development AND :test group
group :development, :test do
  gem 'sqlite3’ # Add it here around line 35–37
  (other gems...)
end
```

UPDATE OUR GEMFILE

/Gemfile

```
# Add the following gem group after the last group
group :production do
  gem 'pg', '0.18.4'
end
```

PROCFILE

./Procfile

```
# Create in the project root a file named ‘Procfile’

# Add this line in the file
web: bundle exec puma –C config/puma.rb
```

WRAPPING IT UP


```
# Install our new gems
$ bundle install --without production
$ bundle update

# The '--without production' is to avoid installing gems we don't need in our development environment
$ git add -A

$ git commit –m "added deployment configurations"
```

DEPLOY COMMANDS


```
# Heroku set up
$ heroku login
$ heroku keys:add

# Create new server instance
$ heroku create

# Push add to Heroku instance
$ git push heroku master

# Open app with this command
$ heroku open
$ git push
```

EXPAND OUR FUNCTIONALITY

LET'S THINK ABOUT OUR ADDRESS BOOK

- What is (are) the resource(s) we need?
- What are the user actions we want to support (e.g. view all contacts, create or edit contact)
- What are the views, controllers and model(s) we need?

THINKING WITH HTTP ACTIONS

- **GET** all contacts
- **GET** contact with ID = 1
- **POST** a new contact
- **PATCH** a contact (edit)
- **DELETE** a contact

CONTACT RESOURCE

- Name
- Phone
- Email
- Address
- Note

RAILS GIVES US 'FREE' CODE

- **Generators:** generate boilerplate code for certain types of files (e.g. controller)

- **Scaffolding:** generate all the necessary code and files for a resource

GENERATORS: THE GOOD AND THE BAD

- **The good:**
  - It's free code that works
  - Helps to build an application quick (e.g. Hackathon, project, API)

- **The bad:**
  - You can't rely on generators to do everything
  - Generators don't provide security and validation
  - You should practice by building your app 'from scratch' - you learn more this way.

FOR THE SAKE OF TIME...

- **For this workshop:**
  - We will use scaffold to achieve our goal to deploy an app within two hours
  - We'll look into the code
  - This is more of a 'big picture' workshop rather than focusing on Rails in detail
  - We will provide resources and material for you to keep learning about Rails

CONTACT RESOURCE - TYPES

- Name : string
- Phone : string
- Email : string
- Address : string
- Note : text

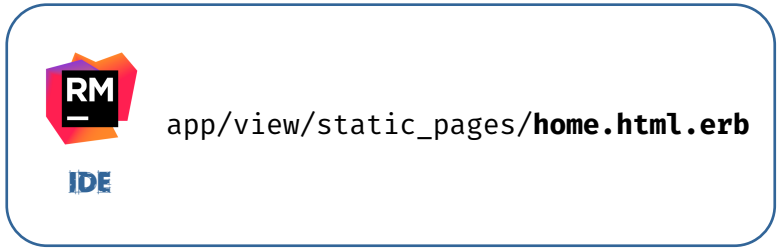
SCAFFOLD COMMANDS


```
# We generate our entire resource (model, controllers, views) with one line
$ rails generate scaffold Contact name:string phone:string email:string
address:string note:text

# We need to create a database migration
$ rails db:migrate

# It's a good time to commit our files
$ git add –A
$ git commit -m "generated scaffold for contacts"
```

THERE IT IS!

```
    invoke  active_record
    create    db/migrate/20170522190236_create_contacts.rb
    create    app/models/contact.rb
    invoke    test_unit
    create      test/models/contact_test.rb
    create      test/fixtures/contacts.yml
    invoke  resource_route
     route    resources :contacts
    invoke  scaffold_controller
    create    app/controllers/contacts_controller.rb
    invoke    erb
    create      app/views/contacts
    create      app/views/contacts/index.html.erb
    create      app/views/contacts/edit.html.erb
    create      app/views/contacts/show.html.erb
    create      app/views/contacts/new.html.erb
    create      app/views/contacts/_form.html.erb
    invoke    test_unit
    create      test/controllers/contacts_controller_test.rb
    invoke    helper
    create      app/helpers/contacts_helper.rb
    invoke      test_unit
    invoke    jbuilder
    create      app/views/contacts/index.json.jbuilder
    create      app/views/contacts/show.json.jbuilder
    invoke  assets
    invoke    coffee
    create      app/assets/javascripts/contacts.coffee
    invoke    scss
    create      app/assets/stylesheets/contacts.scss
    invoke  scss
    create    app/assets/stylesheets/scaffolds.scss
```

SMALL REFACTOR




```
<h1>Rails Address Book</h1>
<h3>Contacts for [Your name]</h3>
<button>Open</button>
<%= link_to "Open Address Book", contacts_path %>
```

TOUR OF THE CODE

FINAL DEPLOY


```
# Commit our recent refactor
$ git add -A
$ git commit –m “refactored home view”

# Deploy the app
$ git push
$ git push heroku

# Run the database migration on the Postgres DB on our server
$ heroku run rails db:migrate

# Open the app in our browser
$ heroku open
```

TIME TO WRAP UP

WE DID A LOT TODAY!

- Web stack development
- MVC
- Basic Git commands
- A sample Rails application
- Deploying a Rails app with Heroku