Setup

We need tools

Sublime Text

A text editor. Your new companion, day & night.

Terminal

Don't fear the command line.

Package manager

You can't imagine how easy it is to install new software with just one command. **Homebrew** on OSX, **Aptitude** on Ubuntu

Git & GitHub

Version Control. Collaboration.

Ruby

Ruby is the programming language. Rails will be the framework to build awesome web app in no time.

Your turn!

Go to github.com/lewagon/setup (https://github.com/lewagon/setup)

Terminal 101

Different names

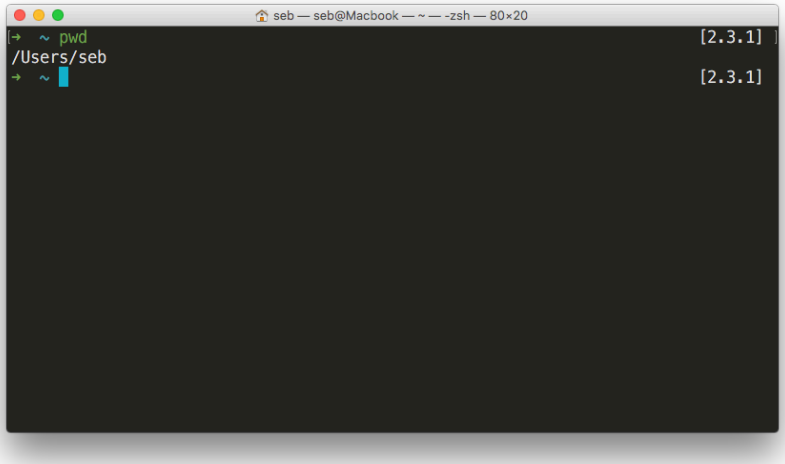
- Command prompt
- Console
- Terminal

Basic commands

Where am I ?

1. Look for the directory name before the prompt, after `➜`
2. Or print the path of the current directory

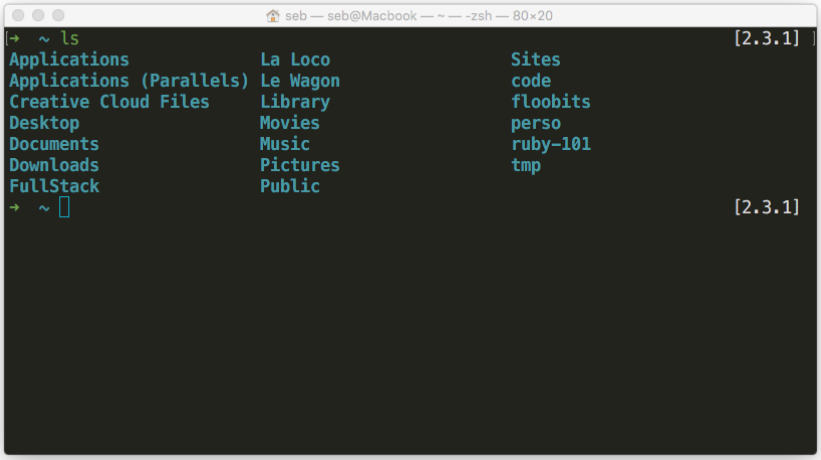
```
pwd
```



That's your `$HOME` directory.

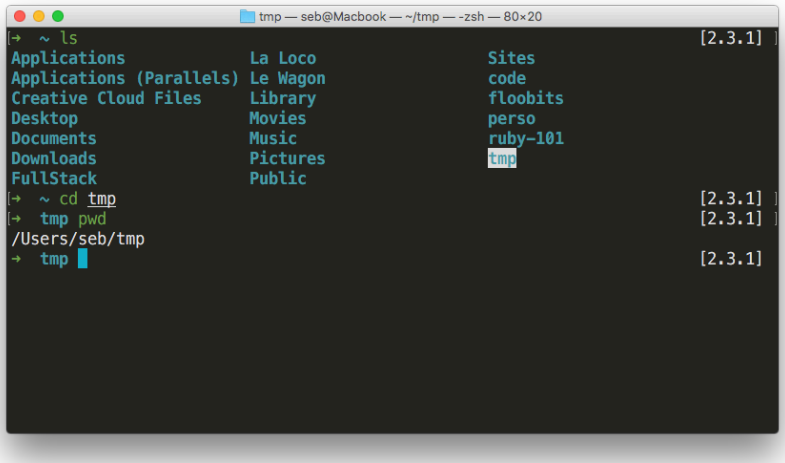
Where can I go ?

`ls` (or **`ll`** , an alias of `ls -lh`)



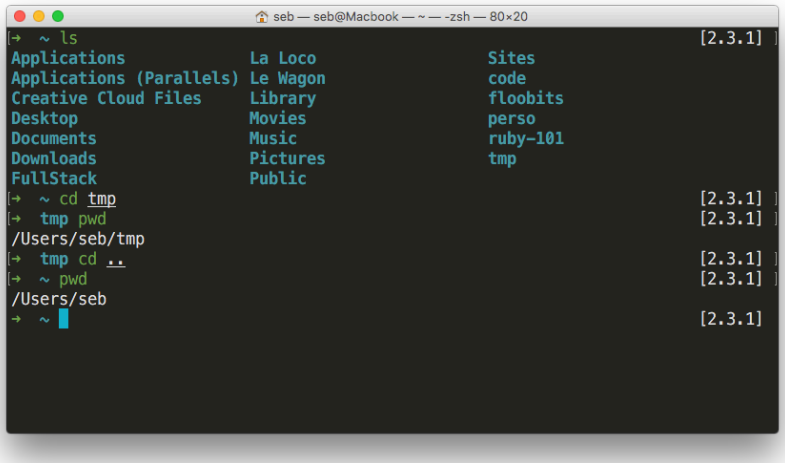
Let's go there

`cd <FOLDER_NAME>`



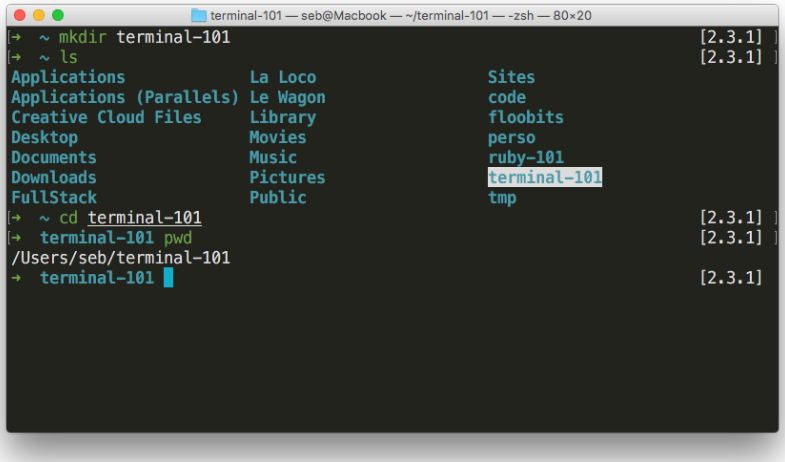
How can I go up?

`cd ..`



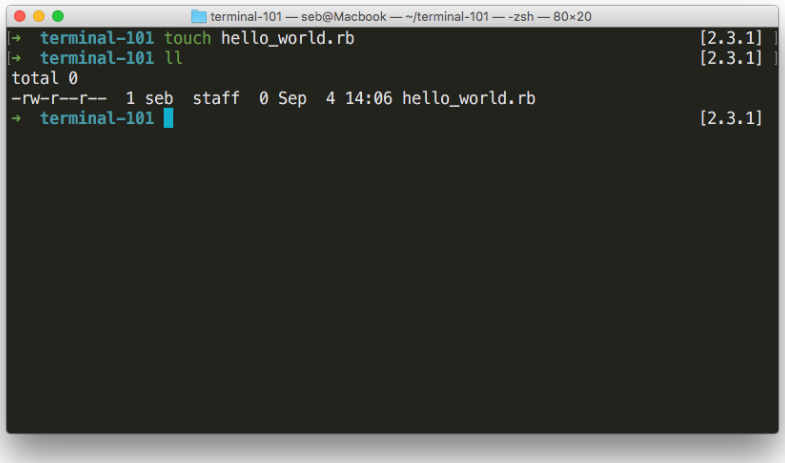
Let's create a directory

`mkdir <NEW_FOLDER>`



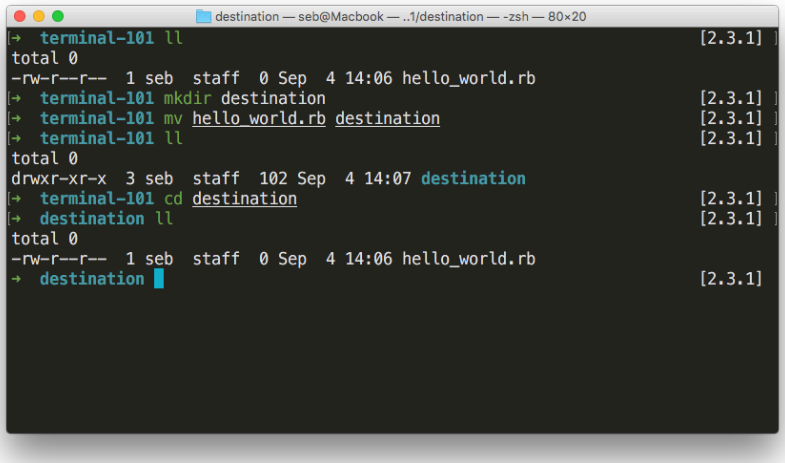
Let's create a file

`touch <FILE_NAME>`



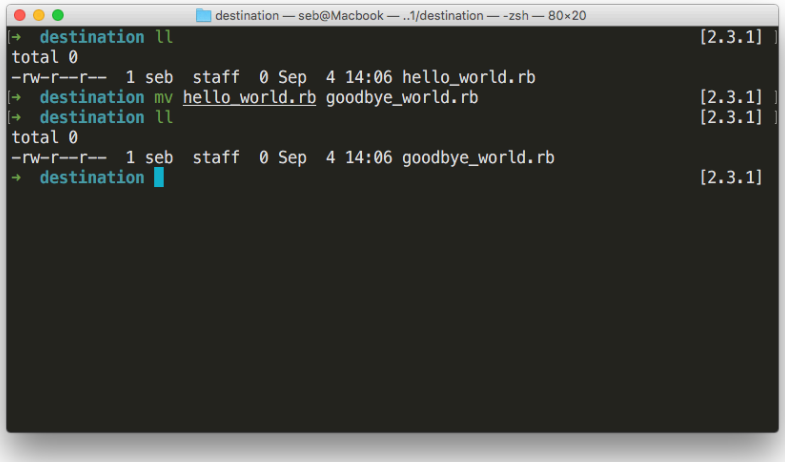
Let's move a file (or directory)

`mv <FILE_NAME> <FOLDER_NAME>`



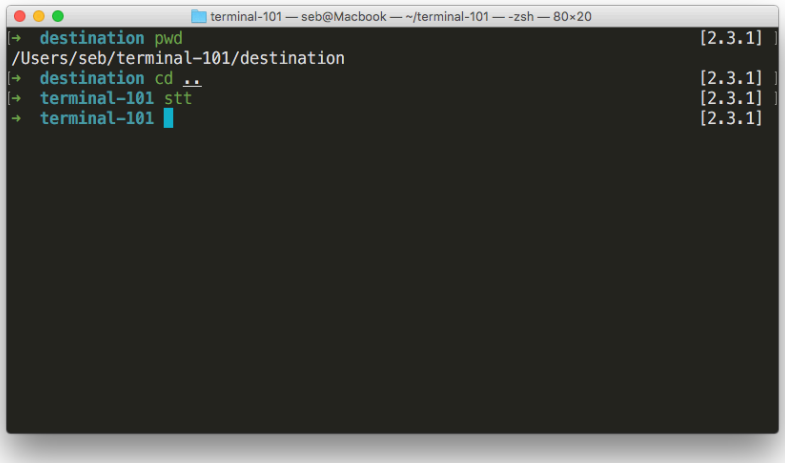
Let's rename a file (or directory)

`mv <FILE_NAME> <NEW_FILENAME>`



Open current directory in Sublime Text

Open your current directory in Sublime with `stt`



Let's view the content of a text file

`cat <FILE_NAME>`

And many more!

Learn Enough Command Line to be Dangerous (https://www.learnenough.com/command-line-tutorial)

Git 101

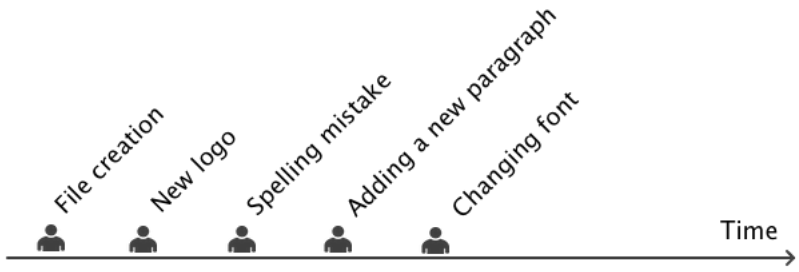
We are knowledge Workers

We create and edit **files** (text, images, etc.)

Everyday workflow

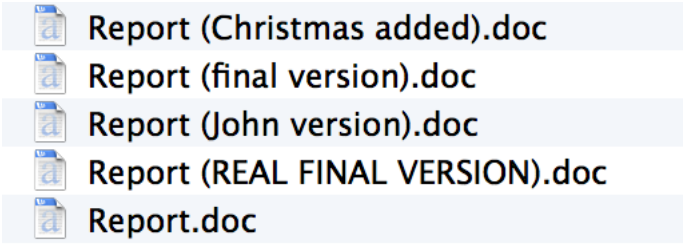
1. Create a file
2. Save it
3. Edit it
4. Save it again
5. etc.

File life



Manual Version Control

How most people keep track of different versions of a file

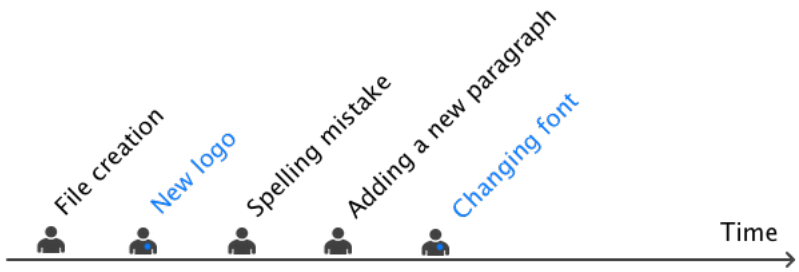


Can we automate this?

For each document version, we need to know:

1. **When** the file was modified
2. **What** changed
3. **Why** it was modified

There's more: Teams



That's one more question:

For each document version, we need to know:

1. When the file was modified
2. What changed
3. Why it was modified
4. **Who** did the change

In a nutshell

We want a tool which:

- tracks document versions
- keeps an history of document changes
- foster team work

That would be

Git basic commands

Starting

```
# From existing repository (on GitHub for instance)
git clone <github_ssh_clone_url>

# Or from scratch
mkdir new_project
cd new_project
git init
```

Status

git can tell you if your folder has some modified files (dirty)

```
git status
```

Commit

A `commit` (a snapshot of the folder) is a 3-steps job.

```
# First check which files have been modified
git status

# Then, add the ones you want to the staging area.
git add <file_1_which_has_been_modified>
git add <file_2_which_has_been_modified>

# You can review your staging area
git status

# Take a snapshot of what is in the staging area.
git commit --message "A meaningful message about this change"
```

Diff

If `git status` tells you something changed, you can inspect exactly what changed:

```
git diff
git diff <a_specific_file_or_folder>
```

Log

Show commit history with:

```
git log

# More fancy command in your ~/.gitconfig
git lg
```

Live-code: git init

Let's create a project and start tracking it

```
mkdir -p ~/code/$GITHUB_USERNAME/git-101 && cd $_
git init
ls -a # it has created a .git hidden folder
```

Live-code: first commit

Let's create an `index.html` file and code some basic HTML content

```
touch index.html
stt # code some basic HTML content
```

Time to commit our work

```
git status # file not staged
git add index.html
git status # file staged, ready to commit
git commit -m "Basic HTML content for home page"
git status
```

Live-code: second commit

Let's add an image in our project

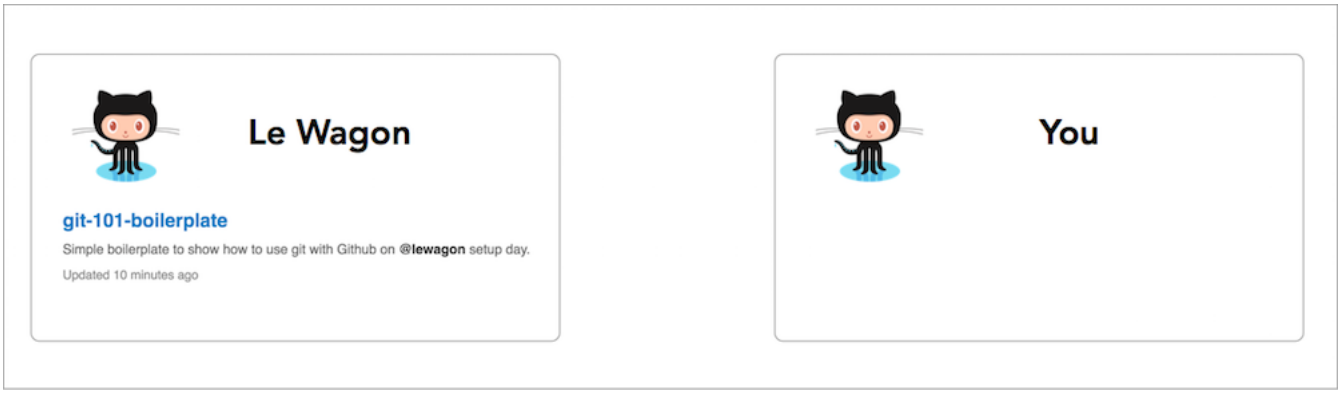
```
curl https://raw.githubusercontent.com/lewagon/karr-images/master/white_logo_red_circle.png > logo.png
stt # add <img src="logo.png"> to your HTML
```

Time to commit our work

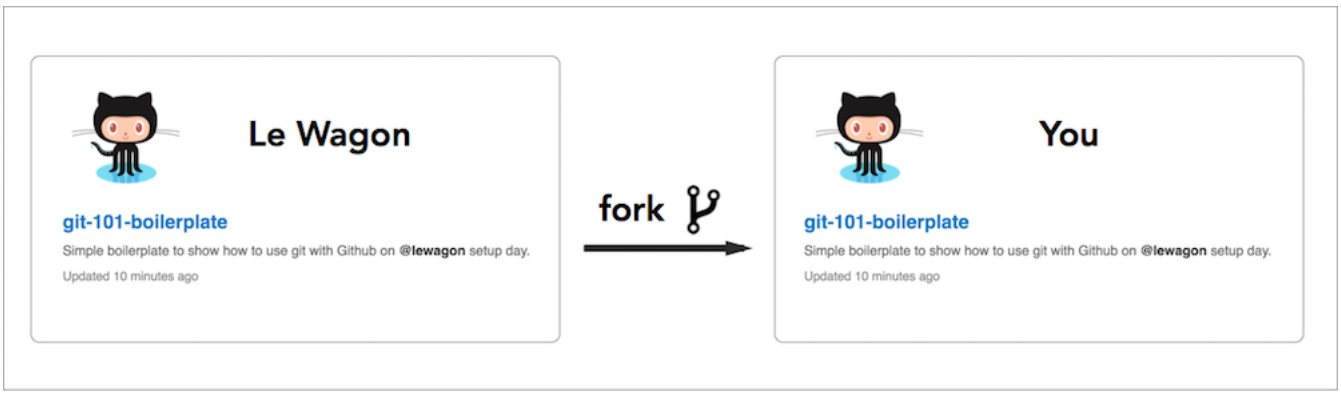
```
git status
git diff index.html # what has changed?
git add index.html
git add logo.png
git status
git commit -m "Adding logo to home page"
git status
git log # check commits history
```

Remote

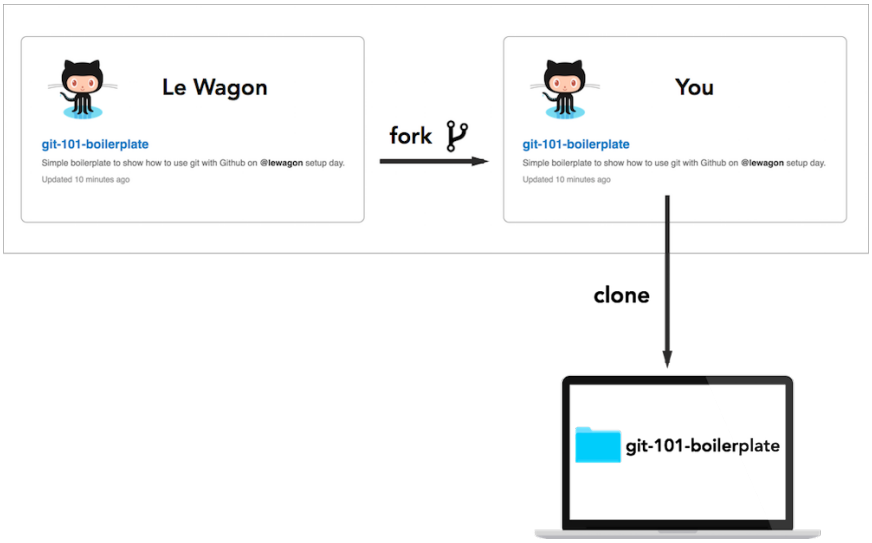
Fork and clone



Fork and clone



Fork and clone



Pushing the changes

Once you've committed your work, push it to Github.

```
# Generic command
git push <remote> <branch>

# What we'll use
git push origin master
```

Live-code: forking and cloning a remote

1. Let's fork Le Wagon's git 101 boilerplate (https://github.com/lewagon/git-101-boilerplate)
2. Then let's clone it with:

```
cd ~/code/$GITHUB_USERNAME
git clone git@github.com:$GITHUB_USERNAME/git-101-boilerplate.git
git status # it's already tracked by git
```

Live-code: commit and push

Let's make a change, commit **and push**

```
stt # change the HTML code
git add index.html
git commit "adding some custom text"
git status
git push origin master # Pushing on Github
```

Check that **project was updated on Github**.

Git advanced

In the next few weeks, we'll see how git can help us with

- Solving conflicts
- collaboration (using branches)
- production deployment (using multiple remotes)

Kitt Demo

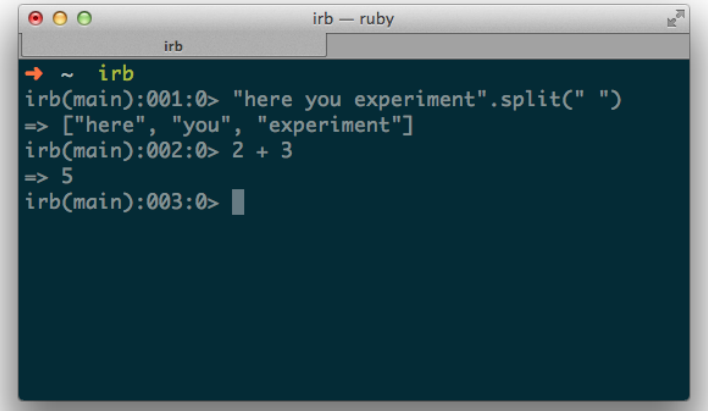
- Navigation
- Lectures
- Videos
- Classmates
- Buddies
- Tickets
- Flashcards: **let's test it on "Terminal and git"**
- Challenges: **let's solve and push `00-Setup/00-Demo`**

Programming basics

How to run your code?

Experiment with IRB

- run `irb` to launch a ruby interpreter
- IRB is your **playground**. So test things, be insane!



Running a ruby file (1)

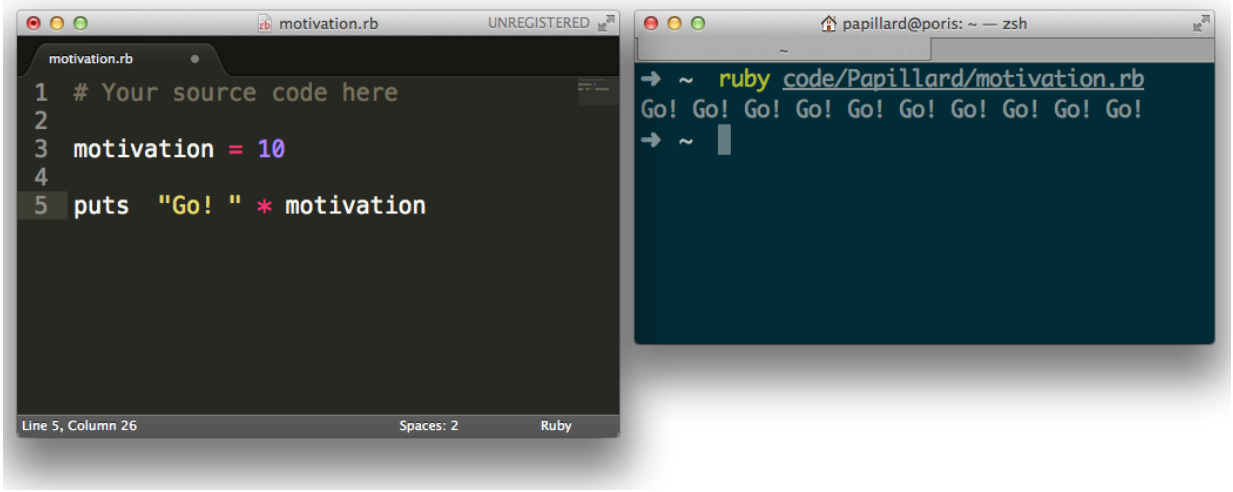
Launch your ruby script with

```
$ ruby path/to/your/file.rb
```

Running a ruby file (2)

- Use `puts` to test results of your script

code/Papillard/motivation.rb **terminal**



Built-in objects

```
"Sponge Bob".class              #=> String
12.class                        #=> Fixnum
3.14.class                      #=> Float
["Sponge Bob", 12, 3.14].class  #=> Array
true.class                      #=> TrueClass
false.class                     #=> FalseClass
(1..100).class                  #=> Range
```

String

```
"yipi yeah".upcase   #=> "YIPI YEAH"
"Hello" == 'Hello'   #=> true
```

Interpolation

```
'two: #{1 + 1}'      #=> "two: #{1 + 1}"
"two: #{1 + 1}"      #=> "two: 2"
```

Conversion to integer

```
'1984'.class        #=> String
'1984'.to_i         #=> 1984
'1984'.to_i.class   #=> Fixnum
```

Fixnum

```
# Standard arithmetic
1 + 2      #=> 3
2 * 4      #=> 8
# Custom methods
20.even?   #=> true
20.odd?    #=> false
```

Conversion to string

```
1984.to_s    #=> "1984"
```

Float

```
3.1416.truncate   #=> 3
```

Array

```
['Sponge Bob', 12, 3.14].size      #=> 3
['Huey', 'Dewey', 'Louie'].sort    #=> ["Dewey", "Huey", "Louie"]
[3, 5, 1].sort                     #=> [1, 3, 5]
```

Shortcut for array of strings

```
%w[Huey Dewey Louie]  #=> ["Huey", "Dewey", "Louie"]
%w(Huey Dewey Louie)  #=> ["Huey", "Dewey", "Louie"]
%w{Huey Dewey Louie}  #=> ["Huey", "Dewey", "Louie"]
```

Range

```
(1..10).to_a       #=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
(1...10).to_a      #=> [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Special values

```
nil
```

```
false
true
```

Tomorrow: we'll combine *booleans* using logic (`&&` , `||` , `!`) and conditions (`if`)

ruby doc is your friend!

Want to perform some basic treatment?

- shuffle an array
- capitalize a word
- split a word into an array of characters
- etc..

Do not re-invent the Wheel

=> http://www.ruby-doc.org/core/

Variables

Deeply understand them!

They are the **elementary blocks of programming**.

Why variables?

Store values to re-use them

```
age = 17
puts "You are #{age} years old"

puts "Lucky you, it's your birthday"
age = age + 1
puts "You are now #{age}"
```

Assignment

```
1  city = “Paris”
2  population = 2000000
```

city	population
“Paris”	2000000

Assignment

```
1  city = “Paris”
2  population = 2000000


3  city_details = “#{city}, France”
```

city	population
“Paris”	2000000

city	population	city_details
“Paris”	2000000	“Paris, France”

Assignment

```
1  city = “Paris”
2  population = 2000000


3  city_details = “#{city}, France”


4  population = population + 30000
```

city	population
“Paris”	2000000

city	population	city_details
“Paris”	2000000	“Paris, France”

city	population	city_details
“Paris”	2030000	“Paris, France”

Methods

Factoring your code

Why methods?

Concise way to call a ruby code

```
def tomorrow
  tomorrow_date = Date.today + 1
  return tomorrow_date.strftime("%B %d")
end

puts tomorrow
```

Why methods?

Apply a ruby code to **dynamic inputs**

```
def full_name(first_name, last_name)
  name = first_name.capitalize + " " + last_name.capitalize
  return name
end

puts full_name("boris", "paillard")
puts full_name("sebastien", "saunier")
```

Combine variables and methods

```
def full_name(first_name, last_name)
  name = first_name.capitalize + " " + last_name.capitalize
  return name
end

boris_first_name = "boris"
boris_last_name = "paillard"
boris_full_name = full_name(boris_first_name, boris_last_name)
puts boris_full_name
```

Parameters vs. arguments

```
def new_population(population, births)
  return population + births
end
```

- `population` and `births` are **parameters**

```
puts new_population(2000000, 300)
```

- `2000000` and `300` are **arguments**
- Arguments are values taken by the parameters

The return convention

A method returns the **last statement executed**.

```
def add(x, y)
  return x + y
end

# is the same as

def add(x, y)
  x + y
end
```

Conventions

Methods and variables in **snake_case**

```
good_method_or_variable_name
```

Conventions

a method ending with a `?` returns `true` or `false`

```
42.even? #=> true
42.odd?  #=> false
```

a method ending with a `!` is destructive or dangerous

```
text = 'Hello'

text.upcase
#=> "HELLO"

text
#=> "Hello"

text.upcase!
#=> "HELLO"

text
#=> "HELLO"
```

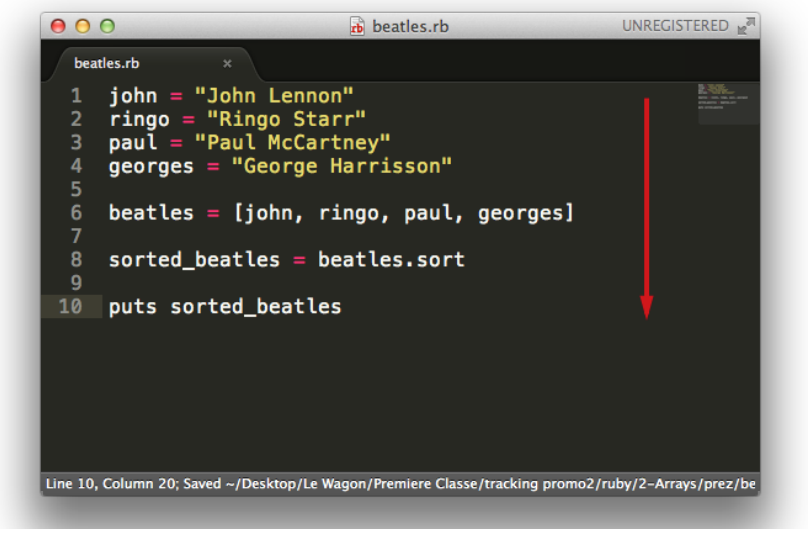
Happy hacking!

Flow & Array

Controlling the flow

Basic flow

Top to bottom / line-by-line



Let's change this flow

if

```
if condition
  # code executed only when condition is "truthy"
end
```

truthy: anything that is different from `false` or `nil`.

0 is true!

if !

```
if !condition
  # code executed only when condition is not "truthy"
end
```

unless

```
unless condition
  # code executed only when condition is not "truthy"
end
```

Live-code - voting age

Can you vote?

```ruby
puts "How old are you?"
print ">"
age = gets.chomp.to_i

if age >= 18
  puts "you can vote!"
end
```

if / else

```ruby
if condition
  # executed when condition is truthy
else
  # executed when condition is not truthy
end
```

Live-code: voting age (enhanced)

Can you vote? enhanced version

```ruby
puts "what's your age?"
print '>'

age = gets.chomp.to_i

if age >= 18
  puts "you can vote!"
else
  puts "too young to vote.."
end
```

Ternary operator

```ruby
condition ? code_when_truthy : code_when_falsey
```

Live-code: let's flip coins

```ruby
puts "heads or tails?"
choice = gets.chomp
coin = ["heads", "tails"].sample

result = (choice == coin) ? "winner" : "looser"
puts "#{result}, that was #{coin}"
```

if / elsif / else

```ruby
if condition
  # ...
elsif other_condition
  # ...
else
  # ...
end
```

Live-code: Morning? Afternoon? Night?

What's wrong with this program?

```ruby
hour = Time.now.hour

if hour < 12
  puts "Good morning!"
elsif hour > 12
  puts "Good afternoon!"
elsif hour >= 20
  puts "Good night!"
else
  puts "Lunch time!"
end
```

case / when / else

Old school UI

```ruby
puts "Which action? [read|write|exit]"
action = gets.chomp

case action
when "read"
  puts 'You are in READ mode'
when "write"
  puts 'You are in WRITE mode'
when "exit"
  puts 'Bye Bye'
else
  puts 'Wrong action'
end
```

Inline conditions

For short single-line statements

```ruby
do_something if condition
do_something unless condition
```

```ruby
number = gets.chomp.to_i
puts "your number #{number} is so even !" if number.even?
```

Multiple conditions - AND

`&&` is the logical AND

```ruby
true  && true             #=> true
false && false            #=> false
true  && false            #=> false
false && true             #=> false
true  && false && true    #=> ?
```

Multiple conditions - OR

`||` is the logical OR

```ruby
true  || true            #=> true
false || false           #=> false
true  || false           #=> true
false || true            #=> true
true  || false || true   #=> ?
```

Live-code - Opening hours

When is the shop opened?

```ruby
hour = Time.now.hour

if (hour > 9 && hour < 12) || (hour > 14 && hour < 18)
  puts "The shop is opened!"
else
  puts "Sorry, the shop is closed.."
end
```

Looping with `while`

```ruby
while condition
  # executed while condition is truthy
end
```

Live-code: find the right price

Let's find the right price!

```ruby
price_to_find = 1 + rand(5)
choice = nil

while choice != price_to_find
  puts "How much (between $1 and $5)?"
  choice = gets.chomp.to_i
end

puts "You won! Price was #{price_to_find}$"
```

Looping with `until`

```ruby
until condition
  # executed until condition is truthy
end
```

Live-code: find the right price (refacto)

Let's find the right price! refacto

```ruby
price_to_find = 1 + rand(5)
choice = nil

until choice == price_to_find
  puts "How much (between $1 and $5)?"
  choice = gets.chomp.to_i
end

puts "You won! Price was #{price_to_find}$"
```

Looping with `for`

```ruby
for num in [1, 2, 3]
  puts num
end
```

Arrays

Define an array

```ruby
empty_array = []                        # an empty array
beatles = ["john", "ringo", "seb"]      # array of 3 strings
```

Get an element with its index

```ruby
beatles = ["john", "ringo", "seb"]
beatles[0]  #=> "john"
beatles[2]  #=> "seb"
beatles[8]  #=> nil
```

indexes start at 0

```ruby
beatles = ["john", "ringo", "seb"]
# index =>    0       1       2
```

Modify an element

```ruby
beatles = ["john", "ringo", "seb"]
beatles[2] = "george"
p beatles #=> ["john", "ringo", "george"]
```

Append an element

```ruby
beatles = ["john", "ringo", "george"]
beatles << "paul"
p beatles #=> ["john", "ringo", "george", "paul"]
```

Delete an element

By **element**:

```ruby
beatles.delete("john")
```

By **index**: `ruby beatles.delete_at(2)`

`size` / `count` / `length`

```ruby
[1, 2, 3].size #=> 3
[1, 2, 3].count #=> 3
[1, 2, 3].length #=> 3
```

`each`

`each` is your new best friend

```ruby
beatles.each do |beatle|
 puts "#{beatle} is in the Beatles"
end

beatles.each { |beatle| puts "#{beatle} is in the Beatles" }
```

And many more...

- http://www.ruby-doc.org/core/Array.html
- http://www.ruby-doc.org/core/Enumerable.html

Happy Hacking!

Iterators & Blocks

---

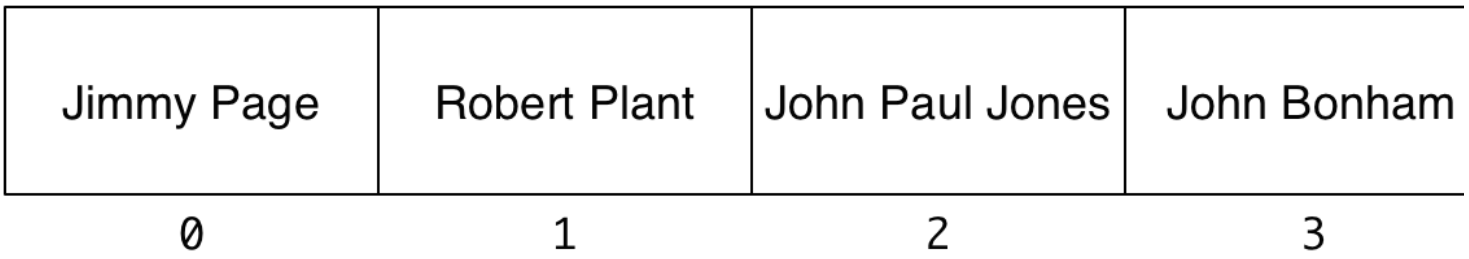
layout: reveal

transition: 'none'

Recap

Array

```ruby
musicians = ['Jimmy Page', 'Robert Plant', 'John Paul Jones', 'John Bonham']
```




```ruby
musicians.size                # => 4
musicians[1]                  # => "Robert Plant"
musicians << 'Miles Davis'    # => [..., 'Miles Davis']
```

Range

Collection of **successive** elements

```ruby
0..10         # including upper bound
# => 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

```ruby
0...10        # excluding upper bound
# => 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

```ruby
(0..10).to_a  # conversion to array
# => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

rubydoc for Range (http://www.ruby-doc.org/core/Range.html)

Array iteration

Looping with `for` on indices

Let's loop through the **indices**

```ruby
for index in 0...(musicians.size)
  musician = musicians[index]
  puts "#{index + 1} - #{musician}"
end

# => 1 - Jimmy Page
#    2 - Robert Plant
#    3 - John Paul Jones
#    4 - John Bonham
```

Looping with `for` on elements

Or directly on the **elements**

```
for musician in musicians
  puts "Listen to #{musician}"
end

# => Listen to Jimmy Page
#    Listen to Robert Plant
#    Listen to John Paul Jones
#    Listen to John Bonham
```

Well, that's not idiomatic.

Iterators

`#each`

Live-code: Let's greet musicians one by one.

```
musicians.each do |musician|
  puts "Hello #{musician}!"
end

# => Hello Jimmy Page!
#    Hello Robert Plant!
#    Hello John Paul Jones!
#    Hello John Bonham!
```

`#each_with_index`

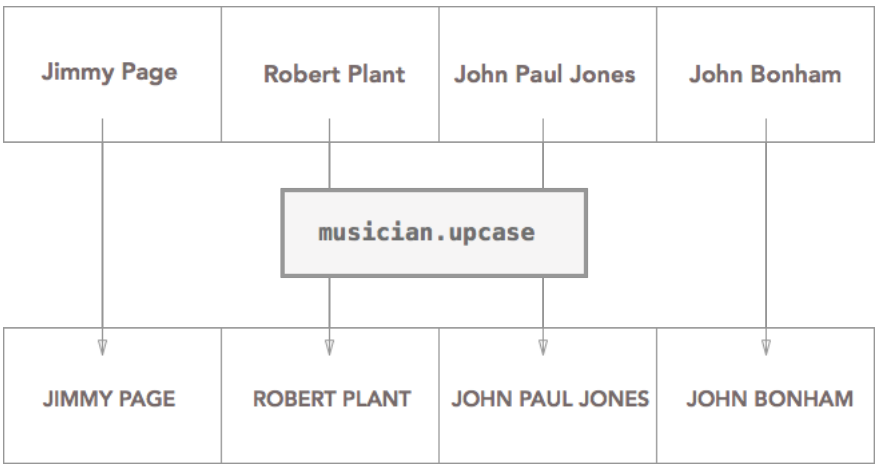
Live-code: Let's display an ordered list of musicians.

```
musicians.each_with_index do |musician, index|
  puts "#{index + 1} - #{musician}"
end

# => 1 - Jimmy Page
#    2 - Robert Plant
#    3 - John Paul Jones
#    4 - John Bonham
```

There's more

`#map`

"Transform" one array to another one by applying some code on **each** element.



`#map` - upcased names

Live-code: Let's build an array of upcased musician names.

```ruby
musicians = ['Jimmy Page', 'Robert Plant', 'John Paul Jones', 'John Bonham']

upcased_musicians = musicians.map do |musician|
  musician.upcase
end

p upcased_musicians
# => ['JIMMY PAGE', 'ROBERT PLANT', 'JOHN PAUL JONES', 'JOHN BONHAM']
```

`#map` - first names

Live-code: Let's build an array of musician first names.

```ruby
musicians = ['Jimmy Page', 'Robert Plant', 'John Paul Jones', 'John Bonham']

musician_first_names = musicians.map do |musician|
  musician.split(" ").first
end

p musician_first_names
# => ['Jimmy', 'Robert', 'John', 'John']
```

`#count`

- Count element of an array for which some code is **true**
- **Live-code**: count musicians starting with "J"

```ruby
musicians = ['Jimmy Page', 'Robert Plant', 'John Paul Jones', 'John Bonham']

j_musicians_count = musicians.count do |musician|
  musician[0] == "J"
end

p j_musicians_count
# => 3
```

`#select`

- Filter from an array elements for which some code is **true**.
- **Live-code**: extract musicians starting with "J"

```ruby
musicians = ['Jimmy Page', 'Robert Plant', 'John Paul Jones', 'John Bonham']

j_musicians = musicians.select do |musician|
  musician[0] == "J"
end

p j_musicians
#=> ['Jimmy Page', 'John Paul Jones', 'John Bonham']
```

And many more

- http://www.ruby-doc.org/core/Array.html
- http://www.ruby-doc.org/core/Enumerable.html

(**Today's first challenge** is to pick the right methods from this documentation)

To summarize...

`#each`, `#map`, `#count`...

- Are called **iterators**.
- **Enumerable** is a **module included** in the `Array` class.
- An iterator is just a **method of Enumerable (http://ruby-doc.org/core/Enumerable.html)**.

Understand what they return

```ruby
# .map      [] -> []
# .count    [] -> Fixnum
# .select   [] -> []
```

Understand how they work

The block of code **has a different role** in each case.

```
musicians.each   { |musician| any_code_with(musician) }
musicians.map    { |musician| transform(musician)     }
musicians.select { |musician| condition_on(musician)  }
```

What the hell was this?

```
do |musician|
  musician.upcase
end
```

This is a piece of code called a **block**

Blocks

Blocks are pieces of code

You can think of blocks as **anonymous** methods.

Block Syntax

1-line syntax

```
{ |num| num * 2 }
```

multi-line syntax

```
do |num|
  num * 2
end
```

Block return

A multi-line block works as a method, it returns the last statement executed.

```
musicians = ['Jimmy Page', 'Robert Plant', 'John Paul Jones', 'John Bonham']

upcased_first_names = musicians.map do |musician|
  first_name = musician.split.first
  capitalized_first_name = first_name.capitalize
  puts "[DEBUG] #{musician}'s first name is #{capitalized_first_name}"

  capitalized_first_name
end
```

Methods using a block

Multi-line syntax

```
method(arg_1, ...) do |block_arg_1, ...| # A block can pass on multiple arguments
  # Some code
end
```

Or with the 1-line syntax

```
method(arg_1, ...) { |block_arg_1, ...| some_code }
```

IMPORTANT

The block is just an argument of the method

Example `#each`

```
[1, 2, 3].each do |number|
  # Do what ever you want with each number
end
```

```
[1, 2, 3].each { |number| some_code }
```

- Remember, those two syntaxes are the same
- The block is **part of the method call**

Define methods using blocks

DISCLAIMER

- Advanced concept, normal if you struggle!
- You'll probably never have to **define** such methods in your ruby/Rails career.
- But you will **use** lots of them (like `each`, `map`, etc..)

`yield` calls the block

- `yield` is a Ruby keyword **executing** the block.
- **Live-code**: let's code a `timer` method

```ruby
def timer
  start_time = Time.now
  yield
  puts "Elapsed time: #{Time.now - start_time}s"
end
```

```ruby
timer() do
  puts "I'm doing something slow..."
  sleep(4)
  puts "I'm done :)"
end
# => I'm doing somethings slow...
# => I'm done :)
# => Elapsed time: 4s
```

Flow of a method called with a block (1)

1. method is called

```ruby
def timer
  start_time = Time.now
  yield
  puts "Elapsed time: #{Time.now - start_time}s"
end

timer() do
  puts "I'm doing something slow..."
  sleep(4)
  puts "I'm done :)"
end
```

```
>
```

Flow of a method called with a block (2)

1. method is called
   with a block

```ruby
def timer
  start_time = Time.now
  yield
  puts "Elapsed time: #{Time.now - start_time}s"
end

                block
timer() do
  puts "I'm doing something slow..."
  sleep(4)
  puts "I'm done :)"
end
```

```
>
```

Flow of a method called with a block (3)

2. set start time

1. method is called with a block

```ruby
def timer
  start_time = Time.now
  yield
  puts "Elapsed time: #{Time.now - start_time}s"
end

timer() do
  puts "I'm doing something slow..."
  sleep(4)
  puts "I'm done :)"
end
```

```
>
```

Flow of a method called with a block (4)

2. set start time
3. yield calls the block

1. method is called with a block

```ruby
def timer
  start_time = Time.now
  yield
  puts "Elapsed time: #{Time.now - start_time}s"
end

block

timer() do
  puts "I'm doing something slow..."
  sleep(4)
  puts "I'm done :)"
end
```

```
> I'm doing something slow…
> I'm done :)
```

Flow of a method called with a block (5)

2. set start time
3. yield calls the block
4. print elapsed time

1. method is called with a block

```ruby
def timer
  start_time = Time.now
  yield
  puts "Elapsed time: #{Time.now - start_time}s"
end

timer() do
  puts "I'm doing something slow..."
  sleep(4)
  puts "I'm done :)"
end
```

```
> I'm doing something slow…
> I'm done :)
> Elapsed time: 4.3s
```

`yield` can be called with parameters

Live-code: let's code a custom greeting method.

```ruby
def greet(first_name, last_name)
  full_name = "#{first_name.capitalize} #{last_name.upcase}"
  return "Hello, #{full_name}"
end

puts greet('john', 'lennon') # "Hello, John LENNON"
```

We want to *customize* this greeting message with a **block**.

Using a block

```ruby
def beautify_name(first_name, last_name)
  full_name = "#{first_name.capitalize} #{last_name.upcase}"
  yield(full_name)
end
```

```ruby
message = beautify_name("john", "lennon") do |name|
  "Greetings #{name}, you look quite fine today!"
end
puts message # "Greetings John LENNON, you look quite fine today!"
```

```ruby
message = beautify_name("john", "lennon") do |name|
  "Bonjour #{name}, comment allez-vous ?"
end
puts message # "Bonjour John LENNON, comment allez-vous ?"
```

```ruby
message = beautify_name("ringo", "starr") do |name|
  "Hey #{name}! Let's play on your #{name} drum kit!"
end
puts message # "Hey Ringo STARR! Let's play on your Ringo STARR drum kit!"
```

Your turn!

Hash & Symbols

Data types we know

```ruby
42                          # Fixnum
1.25                        # Float
true                        # Boolean
"hello world"               # String
[ "a", "e", "i", "o", "u" ] # Array
```

From arrays to hash

Let's take an example

Students

```ruby
students =     [ "Peter", "Mary", "George", "Emma" ]
student_ages = [ 24     , 25    , 22      ,  20    ]
```

Write a program to display a list of students with their age

```ruby
students.each_with_index do |student, index|
  age = student_ages[index]
  puts "- #{student} (#{age} years old)"
end
```

Do you like this solution?

- Imagine with **10k students** in the array.
- You have to know that `"sebastien"` has index `6783` to read his age.
- Hard to maintain, you'll make mistakes when updating or injecting students.

What if we could do?

```
puts students_age["Peter"]
```

We can!

```
students_age = {
  "Peter" => 24,
  "Mary" => 25,
  "George" => 22,
  "Emma" => 20
}
```

Hash

A Hash is a dictionary-like collection of **unique** keys.

For each key, a **value** is associated.

rubydoc (http://www.ruby-doc.org/core/Hash.html)

Reading value

```
paris = {
  "country" => "France",
  "population" => 2211000
}
```

You can access hash value by **keys** with:

```
paris["country"]     # => "France"
paris["population"]  # => 2211000
```

Adding key/value pair

```
paris = {
  "country" => "France",
  "population" => 2211000
}
```

You can add a new key/value to your hash with:

```
paris["star_monument"] = "Tour Eiffel"
```

Updating value

```
paris = {
  "country" => "France",
  "population" => 2211000
}
```

You can update the hash with:

```
paris["population"] = 2211001
```

Deleting key/value pair

```
paris = {
  "country" => "France",
  "population" => 2211000,
  "star_monument" => "Tour Eiffel"
}
```

You can delete a key/value with:

```
paris.delete("star_monument")
```

`#each`

```
paris = { "country" => "France", "population" => 2211000 }

paris.each do |key, value|
  puts "Paris #{key} is #{value}"
end
```

This code will print:

```
# Paris country is France
# Paris population is 2211000
```

Custom methods

```
paris = {
  "country" => "France",
  "population" => 2211000,
  "star_monument" => "Tour Eiffel"
}
p paris.has_key?("country")
p paris.has_key?("language")
p paris.keys
p paris.values
```

Similar to `Array`?

```
cities = [ "London", "Paris", "NYC" ]
city = {
  "name" => "Paris",
  "population" => 2211000
}
```

`Array` are accessed by **indexes**, `Hash` by **keys**

```
cities[0]    # => "London"
city["name"] # => "Paris"
```

More readable for rich data

Which one do you prefer?

```
cities = [ ["London", "England", "Big Ben"], ["Paris", "France", "Tour Eiffel"]]

cities = {
  "London" => { "country" => "England", "monument" => "Big Ben" },
  "Paris" => { "country" => "France", "monument" => "Tour Eiffel" }
}
```

Well, what's the more readable?

```
cities[1][2]
cities["Paris"]["monument"]
```

`Symbol`

Cousin of `String` used for **keywords** of your code

ruby-doc.org/core/Symbol.html (http://www.ruby-doc.org/core/Symbol.html)

Let's take two cities

```
paris = {
  "country" => "France",
  "population" => 2211000
}

london = {
  "country" => "England",
  "population" => 8308000
}
```

`country` and **`population`** are keys of the two hashes. They are **keywords** more than data.

Use symbols for identifiers

In Ruby, when in need of internal **keywords**, we use **symbols**

```
:country # this is a symbol. This is a new data type
```

`Symbol` plays nicely with `Hash`

```
paris = {
  :country => "France",
  :population => 2211000
}
```

is equivalent to:

```
paris = {
  country: "France",
  population: 2211000
}

# New syntax does not change the way we read a key
puts paris[:population]
```

As ruby developers, we prefer the latter syntax.

`Symbol` vs `String`

Strings for **data**. Symbols for **keywords**.

```
# Text data => String
"Sebastien Saunier"
"seb@lewagon.org"
"ruby on Rails"
"Paris"

# Text identifiers => Symbol
:fullname
:email
:skill
:city
```

Great answer on StackOverflow (http://stackoverflow.com/a/16621092/197944)

`Hash` as last method argument

Let's code an HTML generator

```
tag("h1", "Hello world")
# => <h1>Hello world</h1>

tag("h1", "Hello world", { class: "bold" })
# => <h1 class='bold'>Hello world</h1>

tag("a", "Le Wagon", { href: "http://lewagon.org", class: "btn" })
# => <a href='http://lewagon.org' class='btn'>Le Wagon</a>
```

```
def tag(name, content, attrs = {})
  flat_attrs = attrs.map { |key, val| "#{key}='#{val}'" }.join(" ")
  return "<#{name} #{flat_attrs}>#{content}</#{name}>"
end
```

Note: You should not reveal the solution and live code it with students. Start by showing them that we need to do "

" + content "

", then telling them that we will focus on ATTRIBUTES. Usually they will start with #each, and guide them to use #map in the end.

Rails will use similar methods

Data Format

- CSV
- JSON / XML

CSV / `Array`

```
# file.csv
Paris,2211000,"Tour Eiffel"
London,8308000,"Big Ben"
```

```
require "csv"
CSV.foreach("file.csv") do |row|
  # row is an array. For first iteration:
  # row[0] is "Paris"
  # row[1] is 2211000, etc.
end
```

JSON / `Hash`

A JSON document will look like this: `javascript { "name": "Paris", "population": 2211000 }`

And in Ruby, we'll get

```
require "json"
JSON.parse('{ "name": "Paris", "population": 2211000 }')
# => { "name" => "Paris", "population" => 2211000 }
```

JSON is everywhere in APIs

Example: GitHub Api: `/users/ssaunier` (https://api.github.com/users/ssaunier)

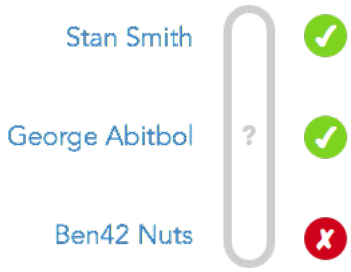
Your turn!

Regular Expressions (Regex)

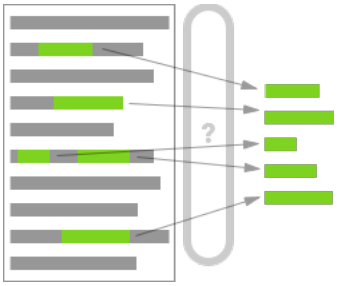
Regular Expressions

Why use it?

Data validation



Data extraction



How does it work?

It's a compact syntax

The following is a ruby `Regexp` (http://ruby-doc.org/core/Regexp.html)

```
/^\+\d{2}\s\d{2}\s\d{4}\s\d{4}$/
```

Matches an international UK phone number like this:

```
+44 20 7946 0234
```

Your Regex Buddy

- Rubular (http://rubular.com)

Delimiters

```
/.../
```

Basics

```
/a/          # a
/ab/         # a directly followed by b
```

Quantifiers

```
/abc?/        # ab followed by 0..1 c
/abc*/        # ab followed by 0..∞ c
/abc+/        # ab followed by 1..∞ c
/abc{3}/      # ab followed by 3 c
```

Grouping

```
/(abc)+/     # 1..∞ abc
/(a|b)c/     # ac OR bc
```

Ranges 1/2

```
/./          # any character
/[aB9]/      # a OR B OR 9
/[0-9]/      # any numeric character
/[a-zA-Z]/   # any alphabetical character
/[^a-c]/     # any char BUT a, b OR c
```

Ranges 2/2

```
/\d/         # like /[0-9]/
/\w/         # like /[a-zA-Z0-9_]/
/\W/         # like /[^a-zA-Z0-9_]/
/\b/         # word boundary (start or end of word)
/\s/         # whitespace (space, tab, line-break, ...)
```

Anchors

```
/^abc/       # starts with abc
/abc$/       # ends with abc
/^abc$/      # contains only abc
```

Modifiers

```
/hello/i         # Will match hello, Hello, HeLlO, HELLO, ...
/hello.world/m   # Will match hello\nworld
```

RegExps in Ruby

Tools

- `irb`
- Rubular (http://rubular.com)

Also:

- Regex 101 (https://regex101.com/)
- RegExr Community (http://regexr.com)
- Debuggex (https://www.debuggex.com/) (Regex are NFA (https://en.wikipedia.org/wiki/Nondeterministic_finite_automaton))

The Regexp (http://ruby-doc.org/core/Regexp.html) class

Regular Expressions are first class citizens in Ruby

```ruby
/hello/.class         # => Regexp
```

```ruby
"hello" =~ /l{2}/       # => 2
"hello" =~ /m{2}/       # => nil
```

Usage in conditionals

```ruby
if phone_number =~ /^\+\d{2}\s\d{2}\s\d{4}\s\d{4}$/
  puts "This is a valid UK international phone number"
else
  puts "It's not valid!"
end
```

RegExps and Ruby

```ruby
text = "
this is a multi-line text
try to match only this line
and not the others
"

text =~ /^try .* line$/       # => 27 ?!
text =~ /\Atry .* line\z/     # => nil
```

`match` (http://ruby-doc.org/core/String.html#method-i-match)

We'll use **capture groups** with **parenthesis**.

```ruby
match_data = "John Doe".match(/^(\w+) (\w+)$/)
puts "Firstname: #{match_data[1]}"
puts "Lastname: #{match_data[2]}"
# => Firstname: John
# => Lastname: Doe
```

You can also **name** capture groups!

```ruby
pattern = /^(?<first_name>\w+) (?<last_name>\w+)$/

match_data = "John Doe".match(pattern)
puts match_data[:first_name]
puts match_data[:last_name]
```

`gsub` (http://ruby-doc.org/core/String.html#method-i-gsub)

```ruby
"hello guys".gsub(/g.{3}/, 'le wagon')
# => "hello le wagon"
```

```ruby
"hello guys".gsub(/^(\w+) (\w+)$/, 'Oh \2, \1!')
# => "Oh guys, hello!"
```

`scan` (http://ruby-doc.org/core/String.html#method-i-scan)

```ruby
"Let's play tic tac toe".scan(/t../)
# => ["t's", "tic", "tac", "toe"]
```

```ruby
"Let's play tic tac toe".scan(/\bt../)
# => ["tic", "tac", "toe"]
```

Doc for word boundary (http://www.regular-expressions.info/wordboundaries.html)

Documentation

- http://www.ruby-doc.org/core/Regexp.html

Happy Regexing!

Parsing & Storing Data

Data formats

3 common data formats: CSV, XML & JSON

Lecture Boilerplate

https://github.com/lewagon/parsing-demo

```
$ cd ~/code/$GITHUB_USERNAME
$ git clone git@github.com:lewagon/parsing-demo.git
$ cd parsing-demo
$ stt
```

CSV

CSV (1)

Filename: `beers.csv`

```
"Name","Appearance","Origin"
"Edelweiss","White","Austria"
"Cuvée des Trolls","Blond","Belgium"
"Choulette Ambrée","Amber","France"
"Gulden Draak","Dark","Belgium"
```

CSV (2)

Comma-Separated Values

- Human readable plain text containing any number of records
- One row per record
- Fields of a record are separated by a comma
- Header row (optional)

CSV (3)

But with a lack of standard

- Delimiters aren't always commas
- Fields aren't always quoted

```
Name;Appearance;"Origin"
Edelweiss;"White";Austria
```

```
Name  Appearance  Origin
Edelweiss  White  Austria
```

XML

XML (1)

Filename: `beers.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beers>
  <title>Great beers</title>
  <beer>
    <name>Edelweiss</name>
    <appearance>White</appearance>
    <origin>Austria</origin>
  </beer>
  <beer>
    <name>Cuvée des Trolls</name>
    <appearance>Blond</appearance>
    <origin>Belgium</origin>
  </beer>
  ...
</beers>
```

XML (2)

Extensible Markup Language

- Human/machine readable text
- XML declaration line
- Focus on a document containing elements and nested sub-documents
- Elements composed of a tag and a content
- Several schemas systems on top of XML

JSON

JSON (1)

Filename: `beers.json`

```
{
  "title": "Great beers",
  "beers": [
    {
      "name": "Edelweiss",
      "appearance": "White",
      "origin": "Austria"
    },
    {
      "name": "Cuvée des Trolls",
      "appearance": "Blond",
      "origin": "Belgium"
    },
    # ...
  ]
}
```

JSON (2)

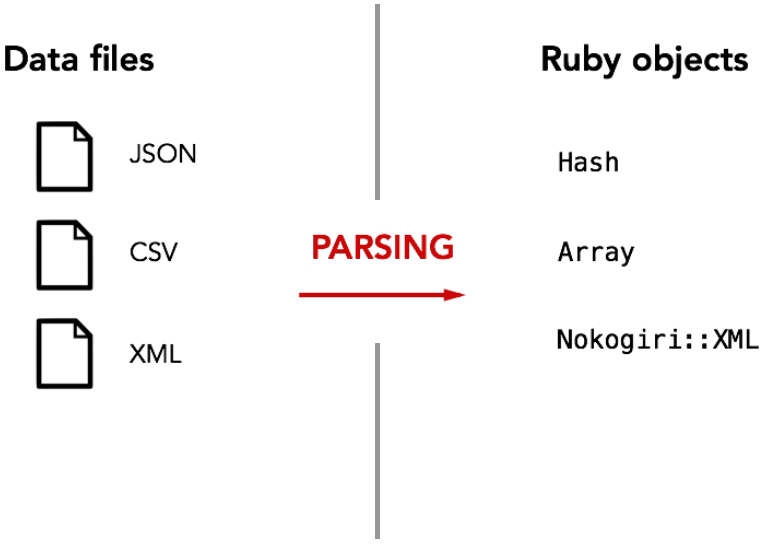
JavaScript Object Notation

- Human readable text
- Format derived from JavaScript but completely independent
- Key-value pair datas
- Just like Ruby hashes

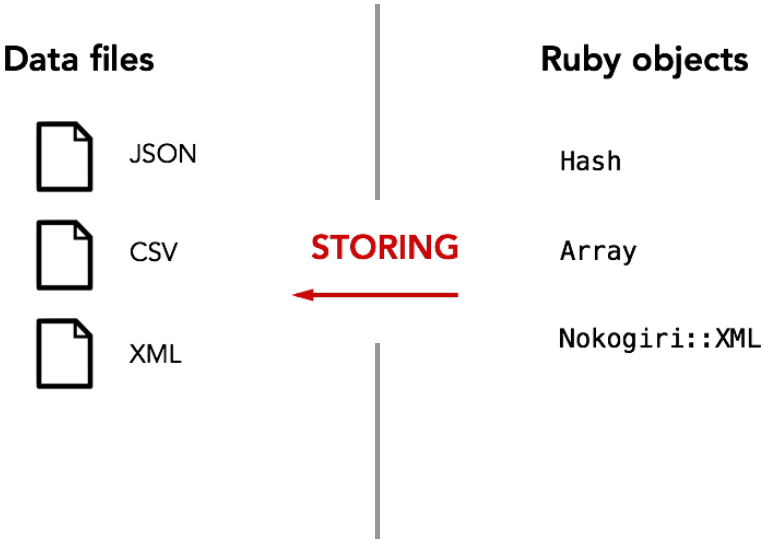
All format heavily used in web services

Parsing & Storing

What's parsing?



What's storing?



Parsing CSV (1)

```
require 'csv'

filepath = 'beers.csv'

CSV.foreach(filepath) do |row|
  # Here, row is an array of columns
  puts "#{row[0]} | #{row[1]} | #{row[2]}"
end
```

Parsing CSV (2)

If your file has headers

```
require 'csv'

csv_options = { col_sep: ',', quote_char: '"', headers: :first_row }
filepath    = 'beers.csv'

CSV.foreach(filepath, csv_options) do |row|
  puts "#{row['Name']}, a #{row['Appearance']} beer from #{row['Origin']}"
end
```

Storing CSV

```
require 'csv'

csv_options = { col_sep: ',', force_quotes: true, quote_char: '"' }
filepath    = 'beers.csv'

CSV.open(filepath, 'wb', csv_options) do |csv|
  csv << ['Name', 'Appearance', 'Origin']
  csv << ['Asahi', 'Pale Lager', 'Japan']
  csv << ['Guinness', 'Stout', 'Ireland']
  # ...
end
```

Parsing JSON

```
require 'json'

filepath = 'beers.json'

serialized_beers = File.read(filepath)

beers = JSON.parse(serialized_beers)

# => beers is a Hash
```

Storing JSON

```
require 'json'

beers = { beers: [
  {
    name:       'Edelweiss',
    appearance: 'White',
    origin:     'Austria'
  },
  {
    name:       'Guinness',
    appearance: 'Stout',
    origin:     'Ireland'
  }
  # etc...
]}

File.open(filepath, 'wb') do |file|
  file.write(JSON.generate(beers))
end
```

Parsing XML

We will use the Nokogiri (http://www.nokogiri.org) gem.

No live-code, just use the code below when you need it :)

```
require 'nokogiri'

file      = File.open('beers.xml')
document  = Nokogiri::XML(file)

document.root.xpath('beer').each do |beer|
  name        = beer.xpath('name').text
  appearance  = beer.xpath('appearance').text
  origin      = beer.xpath('origin').text

  puts "#{name}, a #{appearance} beer from #{origin}"
end
```

Storing XML with Nokogiri

```
require 'nokogiri'

filepath  = 'beers.xml'
builder   = Nokogiri::XML::Builder.new(encoding: 'UTF-8') do
  beers do
    beer do
      name        'Edelweiss'
      appearance  'White'
      origin      'Austria'
    end
    beer do
      # [...]
    end
  end
end

File.open(filepath, 'wb') { |file| file.write(builder.to_xml) }
```

Data from the Web

Using JSON API

Most services have a JSON API.

Example: GitHub API

Example: api.github.com/users/ssaunier (https://api.github.com/users/ssaunier)

```
require 'json'
require 'open-uri'

url = 'https://api.github.com/users/ssaunier'
user_serialized = open(url).read
user = JSON.parse(user_serialized)

puts "#{user["name"]} - #{user["bio"]}"
```

Scraping

Sometimes there is no API. We have to **scrape** the HTML directly

Inspect Element

CSS Selector

```
.the_class
```

```
<div class="the_class">
  Some text
</div>
```

Documentation for selectors (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/Selectors) on MDN

Scraping

```
require 'open-uri'
require 'nokogiri'

ingredient = "chocolate"

html_file = open("http://www.epicurious.com/tools/searchresults?search=#{ingredient}")
html_doc = Nokogiri::HTML(html_file)

html_doc.search('.recipeLnk').each do |element|
  puts element.text
  puts element.attribute('href')
end
```

Happy scraping!

Introduction to OOP

A quick word before

Let's talk about debugging.

How do you debug your program right now?

We can do better than just `puts` or `p`.

Let's use the gem pry-byebug (https://github.com/deivid-rodriguez/pry-byebug)

```
gem install pry-byebug
```

How to

To use the gem, put this line at the top:

```
require "pry-byebug"
```

You can then insert **breakpoints**.

Software Architecture

Because real life programs aren't one file programs

OOP = Data + Behavior

`String`

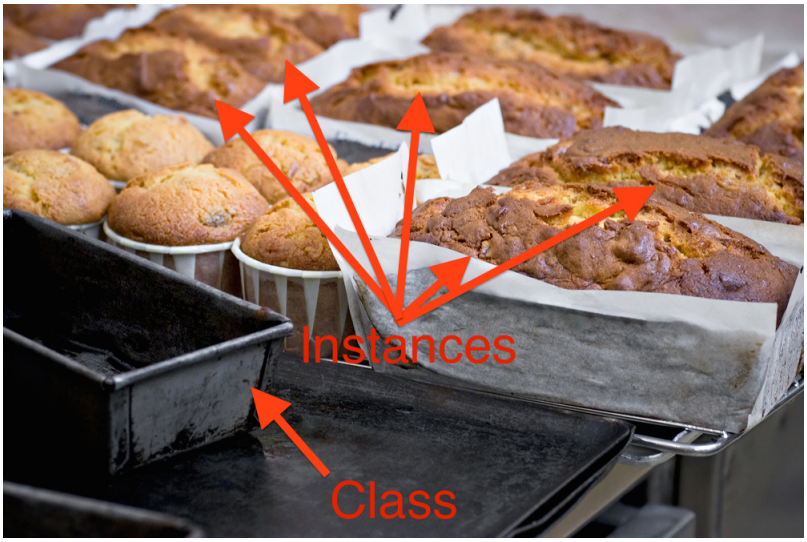
We already have used built-in classes like `String`

```
name = String.new("John Lennon")  # (same as) name = "John Lennon"
name.split # => [ "John", "Lennon" ]
```

- **Data** (or **State**) = the list of characters.
- **Behavior** = the set of [methods](http://www.ruby-doc.org/core/String.html) which can act on the list of characters.

Class and Instances

- A **class** is like a cake mold. It helps to create a cake, but is not a cake
- An **instance** of a class is a cake created from a given cake mold



My first Class

Let's create a `Car` class

```ruby
# car.rb
class Car
end
```

Convention: filename is in lower snake case, class name in upper camel case.

For example: `sports_car.rb` → `class SportsCar`

Instantiation

```ruby
my_car = Car.new
your_car = Car.new
```

We just created two new **instances** of the class `Car`.

Constructor

```ruby
Car.new # => Instantiates the class, then calls the method `initialize`.
```

```ruby
class Car
  def initialize
  end
end
```

Instance variable - Data

```
class Car
  def initialize
    @engine_started = false
  end
end
```

Instance methods - Behavior 1/2

```
class Car
  def initialize
    @engine_started = false
  end

  def engine_started?
    return @engine_started
  end
end
```

```
my_car = Car.new
my_car.engine_started?
# => false
```

Instance methods - Behavior 2/2

```
class Car
  def initialize
    @engine_started = false
  end
  def engine_started?
    return @engine_started
  end
  def start_engine
    @engine_started = true
  end
end
```

```
my_car = Car.new
my_car.engine_started?
# => false
my_car.start_engine
my_car.engine_started?
# => true
```

Setting and accessing state

```
class Car
  def initialize(color)
    @color = color
  end
end
```

```
red_car = Car.new("red")
green_car = Car.new("green")
```

Explicit reader

```
class Car
  def initialize(color)
    @color = color
  end
  def color
    return @color
  end
end
```

```
red_car = Car.new("red")
red_car.color
# => "red"
```

`attr_reader`

```ruby
class Car
  def color
    return @color
  end
end
```

equivalent to

```ruby
class Car
  attr_reader :color
end
```

`attr_reader` convenient shortcut

```ruby
class Car
  def color
    return @color
  end
  def kms
    return @kms
  end
  # etc..
end
```

equivalent to

```ruby
class Car
  attr_reader :color, :kms # etc..
end
```

Explicit writer

```ruby
class Car
  attr_reader :color
  def initialize(color)
    @color = color
  end

  def color=(new_color)
    @color = new_color
  end
end
```

```ruby
my_car = Car.new("red")
my_car.color
# => "red"
my_car.color = "green" # (same as) my_car.color=("green")
my_car.color
# => "green"
```

`attr_writer`

Again there is a shortcut method

```ruby
class Car
  attr_writer :color

  def initialize(color)
    @color = color
  end
end
```

```ruby
my_car = Car.new("red")
my_car.color = "green"  # => car color has been updated
```

`attr_accessor`

`attr_reader + attr_writer = attr_accessor`

```ruby
class Car
  attr_accessor :color  # Can read and write the color property
end
```

```ruby
my_car = Car.new
my_car.color = "red"
my_car.color
# => "red"
```

Interface

Public Interface

A **contract** with the external world

```ruby
class Car
  def start_engine
  end
end
```

Private interface

```ruby
class Car
  def start_engine
    start_petrol_pump
    init_spark_plug
  end

  private

  def start_petrol_pump
  end

  def init_spark_plug
  end
end
```

External world does not know about both private methods.

Summary

- **Everything** in ruby is an object.
- OOP is about **data** (or **state**) and **behavior**.
- State is stored in **instance variables** (`@foo`)
- Behavior is defined by **instance methods** (`def bar` in `class` definition)

Inheritance, class methods, self

Inheritance

Sometime, your classes share some behavior and states

Imagine you had three classes, each one to create a different type of building

```ruby
class House
end

class Skyscraper
end

class Castle
end
```

They share some properties

- They all have a **name**
- They all have a **height**, a **width**, and a **length**
- For every building we have a way to calculate the floor area

Thus, we could do : ```ruby class House attr_accessor :name, :height, :width, :length def initialize(name, height, width, length) @name = name @height, @width, @length = height, width, length end

def floor*area @width * @length end end ruby class Castle attr*accessor :name, :height, :width, :length def initialize(name, height, width, length) @name = name @height, @width, @length = height, width, length end

def floor_area @width * @length end end ```

DRY

Don't **R**epeat **Y**ourself.

Houses, Castles and Skyscrapers are all **buildings**

The `Building` class

```
class Building
  attr_accessor :name, :height, :width, :length
  def initialize(name, height, width, length)
    @name = name
    @height, @width, @length = height, width, length
  end

  def floor_area
    @width * @length
  end
end
```

Inheritance

```
class Castle < Building
end

class House < Building
end
```

```
some_castle = Castle.new('Tower of London', 27, 100, 480)
some_castle.name # => Tower of London
some_castle.floor_area # => 48000
```

```
my_house = House.new('Ker Avel', 5, 10, 12)
my_house.name # => Ker Avel
my_house.floor_area # => 120
```

Specific behavior

```
class Castle < Building
  attr_accessor :butler

  def has_a_butler?
    @butler != nil
  end
end
```

```
my_castle = Castle.new('Moulinsard', 24, 50, 30)
my_castle.has_a_butler? # => false
my_castle.butler = "George"
my_castle.has_a_butler? # => true
```

```
my_house = House.new('Ker Avel', 5, 10, 12)
my_house.has_a_butler? # => Undefined method `has_a_butler?'
```

Change an inherited method

With the `super` keyword, which *calls* the **parent**'s method with the **same** name.

```
class Castle < Building
  # A castle always has a garden of 300 sq. m
  def floor_area
    super + 300  # `super` calls `floor_area` of `Building`
  end
end
```

```
some_castle = Castle.new('Tower of London', 27, 100, 480)
some_castle.floor_area # => 48300
```

Real world example : `IO`

```
class IO # Stands for InputOutput
  def read
    # [...]
  end
end
```

```ruby
class File < IO
  def each_line
    # [...]
  end
end

class CSV < IO
  def add_row(content)
    # [...]
  end
end
```

Summary

- *Inheritance* is used when classes need to **share** behavior and properties
- *Subclasses* **inherit methods** and **instance variables** from parents
- On top of that, *subclasses* can define **more** instance variables and methods
- Use `super` to access the **parent method** with the **same name**

Disclosure / Rails

- As with `yield`, when building Rails app, you won't define a lot of parent/child classes. Still you need to know the mechanism to interpret the documentation.
- **But** you'll often **inherit** from Rails classes (`ActiveRecord::Base`, `ActionController::Base`, ...)

Class methods

Reminder: Instance VS Class

A `class` is used to create **`instances`** using the **`new`** method

```ruby
# castle.rb
class Castle # => This is a class
end
```

```ruby
Castle.new # => This is an instance of the Castle class
```

Instance method

```ruby
class Castle < Building
  def has_a_butler?
    @butler != nil
  end
end
```

The `has_a_butler?` method is an **instance method**.

We can call it on a `Castle` **instance** `ruby my_castle = Castle.new my_castle.has_a_butler? # => false`

Not on the `Castle` class directly

```ruby
Castle # => This is the class
Castle.has_a_butler? # => Undefined method `has_a_butler?` for Castle
```

Class methods

They are called on the **class**

```ruby
Castle.categories
```

Class methods - Syntax

```ruby
class Castle < Building
  def self.categories
    ['Medieval', 'Norman', 'Ancient']
  end
end
```

Note the **`self`**.

You can call this method **directly** on the Class. ```ruby puts Castle.categories.join(', ')

=> Medieval, Norman, Ancient

```
You **can't** call class methods on instances!

```ruby
castle = Castle.new
castle.categories # => Undefined method `categories`
```

When to create class methods?

Example

```ruby
class House
  # [...]
end

def price_per_square_meter(city)
  case city
  when "Paris" then 9000
  when "Brussels" then 3000
  else raise Exception.new("No data for #{city}")
  end
end
```

Makes sense to use a class method

```ruby
class House
  def self.price_per_square_meter(city)
    case city
    when "Paris" then 9000
    when "Brussels" then 3000
    else raise Exception.new("No data for #{city}")
    end
  end
end
```

In a nutshell

You can create a **class** method if it does not need an instance

You will **use class methods** more than you **define** them.

Real world examples (1)

```ruby
Time.now # => 2014-07-15 23:19:43 +0200
```

`now` is a **class** method of `Time` which returns an **instance** of `Time`.

Real world examples (2)

```ruby
# `Nokogiri::HTML` is a module, and acts as a namespace
Nokogiri::HTML::Document.parse("<h1>Hello guys</h1>")
```

`parse` is a **class** method of `Nokogiri::HTML::Document` which returns an **instance** of `Nokogiri::HTML::Document`.

Real world examples (3)

```ruby
JSON.parse('{ "key": "value", "other_key": "other_value" }')
```

`parse` is a **class** method of `JSON` which returns an **instance** of `Hash`.

Self

2 use cases:

- Inside an instance method
- Inside a class definition, to define class methods

Inside an instance method

`self` refers to the **current** instance
```

```ruby
class House
  def to_s
    "#{@name} owned by #{self.owner_name}"
  end

  def owner_name
    @owner.name
  end
end
```

`self` is not mandatory in that case.

When do we actually need it explicitly?

```ruby
class Butler
  def initialize(house)
    @house = house
  end

  def clean_house
    puts "#{@house.name} cleaned!"
  end
end
```

```ruby
class House
  attr_reader :name

  def initialize(name)
    @name = name
    @butler = Butler.new(???)
  end
end
```

```ruby
class House
  attr_reader :name

  def initialize(name)
    @name = name
    @butler = Butler.new(self)
  end
end
```

Inside a class

```ruby
class House
  def self.price_per_square_meter(city)
    # [...]
  end
end
```

Good luck!

Building Software - MVC

How to figure out a class

Real-life object

`Vehicle`

```ruby
class Vehicle
  def initialize(color)
    @color = color
    @engine_started = false
  end

  def start_engine
    @engine_started = true
  end

  def engine_started?
    return @engine_started
  end
end
```

`Vehicle` is written in the singular form, not plural.

Recipe

```ruby
class Recipe
  attr_reader :ingredients

  def initialize(name)
    @name = name
    @ingredients = []
  end

  def add_ingredient(ingredient)
    @ingredients << ingredient
  end
end
```

Again, `Recipe` is written in the singular form, not plural.

Data structures

- It's a list of plates
- You can take a plate on top (`pop`)
- You can put a new plate on top (`push`)

Stack

```ruby
class Stack
  def initialize
    @elements = []
  end

  def push(element)
    @elements << element
  end

  def pop
    @elements.delete_at(-1)
  end
end
```

Ruby is awesome (1)

`Array` already has a **stack** behavior with `push` (ruby-doc.org/core/Array.html#method-i-push) and `pop` (ruby-doc.org/core/Array.html#method-i-pop)

```ruby
beatles = %w(john paul george)
```

```ruby
# PUSH: => element is added at the end of the array.
beatles.push("ringo")

beatles.length
# => 4
p beatles
# => [ "john", "paul", "george", "ringo" ]
```

```ruby
# POP: last element of the array is picked.
beatle = beatles.pop
# => "ringo"

p beatles
# => [ "john", "paul", "george" ]
```

Queue

```ruby
class Queue
  def initialize
    @elements = []
  end

  def enqueue(element)
    @elements << element
  end
  def dequeue
    @elements.delete_at(0)
  end
end
```

Ruby is awesome (2)

`Array` already has a **queue** behavior with `shift` (ruby-doc.org/core/Array.html#method-i-shift) and `push` (ruby-doc.org/core/Array.html#method-i-push)

```ruby
beatles = %w(john paul george)
```

```ruby
# ENQUEUE: => element is added at the end of the array.
beatles.push("ringo")

beatles.length
# => 4
p beatles
# => [ "john", "paul", "george", "ringo" ]
```

```ruby
# DEQUEUE: first element of the array is picked.
beatle = beatles.shift
# => "john"

p beatles
# => [ "paul", "george", "ringo" ]
```

Let's build a TODO manager

And it's gonna be a live code!

```ruby
class Task
  attr_reader :description, :done

  def initialize(description)
    @description = description
    @done = false
  end

  def done?
    @done
  end

  def mark_as_done!
    @done = true
  end
end
```

```ruby
class TaskRepository
  def initialize
    @tasks = []
  end

  def add(task)
    @tasks << task
  end

  def all
    @tasks
  end

  def find(index)
    @tasks[index]
  end

  def remove(index)
    @tasks.delete_at(index)
  end
end
```

```ruby
class TasksView
  def display(tasks)
    tasks.each_with_index do |task, index|
      done = task.done? ? "[x]" : "[ ]"
      puts "#{done} #{index + 1} - #{task.description}"
    end
  end

  def ask_user_for_description
    puts "What do you want to do?"
    return gets.chomp
  end

  def ask_user_for_index
    puts "Index?"
    return gets.chomp.to_i - 1
  end
end
```

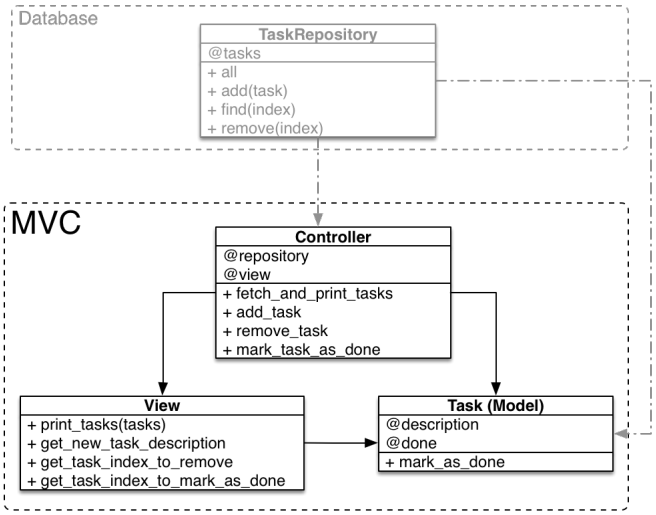
```ruby
require_relative 'task'
require_relative 'tasks_view'

class TasksController
  def initialize(repository)
    @repository = repository
    @view = TasksView.new
  end

  def list
    display_tasks
  end

  def create
    # 1. Get description from view
    description = @view.ask_user_for_description
    # 2. Create new task
    task = Task.new(description)
    # 3. Add to repo
    @repository.add(task)
  end

  def mark_as_done
    # 1. Display list with indices
    display_tasks
    # 2. Ask user for index
    index = @view.ask_user_for_index
    # 3. Fetch task from repo
    task = @repository.find(index)
    # 4. Mark task as done
    task.mark_as_done!
  end

  def destroy
    # 1. Display list with indices
    display_tasks
    # 2. Ask user for index
    index = @view.ask_user_for_index
    # 3. Remove from repository
    @repository.remove(index)
  end

  private

  def display_tasks
    # 1. Fetch tasks from repo
    tasks = @repository.all
    # 2. Send them to view for display
    @view.display(tasks)
  end
end
```

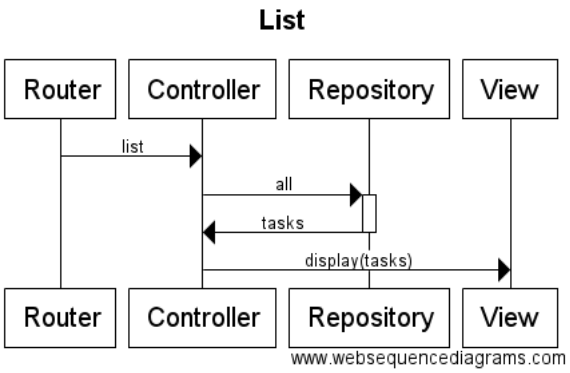
```ruby
class Router
  def initialize(controller)
    @controller = controller
  end

  def run
    loop do
      print_actions
      action = gets.chomp.to_i
      dispatch(action)
    end
  end

  private

  def print_actions
    puts "\n---"
    puts 'What do you want to do?'
    puts '1 - Display tasks'
    puts '2 - Add a new task'
    puts '3 - Mark a task as done'
    puts '4 - Remove a task'
    puts '---'
  end

  def dispatch(action)
    case action
    when 1 then @controller.list
    when 2 then @controller.create
    when 3 then @controller.mark_as_done
    when 4 then @controller.destroy
    else
      puts "Please type 1, 2, 3 or 4 :)"
    end
  end
end
```

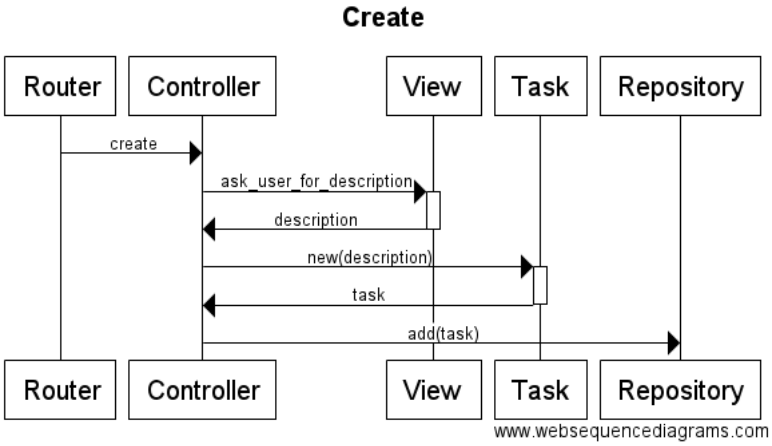
Start the program

```ruby
require_relative 'lib/task_repository'
require_relative 'lib/tasks_controller'
require_relative 'lib/router'

repository = TaskRepository.new
controller = TasksController.new(repository)
router = Router.new(controller)
router.run
```

Get the code on GitHub! lewagon/oop-todolist (https://github.com/lewagon/oop-todolist)







Modeling data

Real apps have several models

Example: a Hospital

1st model: Patient

Let's live-code!

```
class Patient
  attr_reader :name, :cured

  def initialize(attributes = {})
    @name = attributes[:name]
    @cured = attributes[:cured] || false
  end

  def cure
    @cured = true
  end
end
```

2nd model: Room

Let's live-code!

```
class Room
  def initialize(attributes = {})
    @capacity = attributes[:capacity] || 0
    @patients = attributes[:patients] || []
  end

  def full?
    @patients.size == @capacity
  end
end
```

Assigned room to patient?

```
patient = Patient.new(name: "Bob")
patient.room
# => NoMethodError
```

```ruby
class Patient
  attr_accessor :room
  # [...]
end
```

```ruby
class Room
  # [...]
  def add_patient(patient)
    fail Exception, "Room is full!" if full?
    patient.room = self
    @patients << patient
  end
end
```

Persistence

How will you store that in the database (CSV)?

Let's **open** Excel / Google Spreadsheet.

ids!

Rooms CSV file content

	A	B
1	id	capacity
2	1	2
3	2	2
4	3	1
5	4	4
6	5	2

Patients CSV file content

	A	B	C	D
1	id	name	cured	room_id
2	1	Paul	false	1
3	2	John	true	1
4	3	Ringo	false	3

```ruby
class Room
  attr_accessor :id
end
```

```ruby
class Patient
  attr_accessor :id
end
```

There will be a `RoomRepository` and a `PatientRepository` setting the `id` with an auto-increment strategy.

Aparté

If you have this `patients.csv` file:

```
id,name,cured
1,paul,true
2,john,false
```

You can write the following ruby code:

```ruby
patients = []
csv_options = { headers: :first_row, header_converters: :symbol }
CSV.foreach(csv_file, csv_options) do |row|
  row[:id]    = row[:id].to_i          # Convert column to Fixnum
  row[:cured] = row[:cured] == "true"  # Convert column to boolean
  patients << Patient.new(row)
end

p patients
```

Auto-increment

Loading data

```ruby
class PatientRepository
  def initialize(csv_file, room_repository)
    # [...]
    load_csv
  end

  def load_csv
    # Getting @patients from CSV
    @next_id = @patients.empty? ? 1 : @patients.last.id + 1
  end
end
```

Adding a new patient

```ruby
class PatientRepository
  def initialize
    @patients = []
  end

  def add(patient)
    patient.id = @next_id
    @next_id += 1
    @patients << patient
    save_csv
  end
end
```

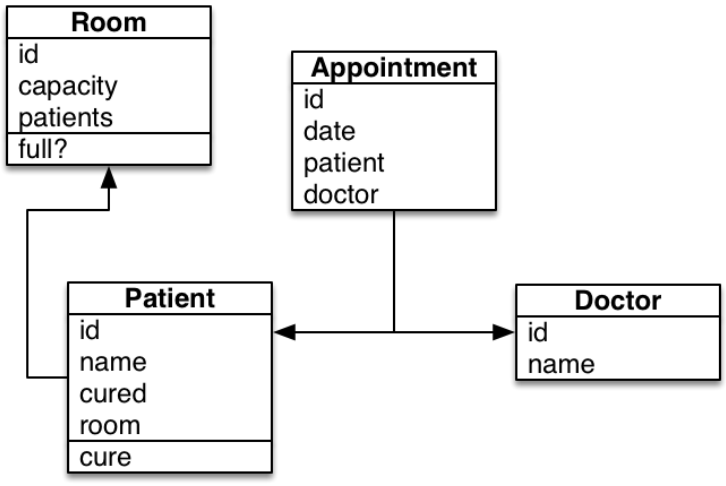
```ruby
class Patient
  attr_accessor :room
end
```

The `PatientRepository` will have to store the `room.id` in the CSV. Plus, when loading from the CSV, it will convert the `room_id` column into a `Room` instance fetched from `RoomRepository`

Serialization / Deserialization

Storing a column `room_id` in `patients.csv` is a serialization of the reference to the `Room` instance from `Patient`'s `room` instance variable.

Adding more models



```ruby
class Doctor
  attr_accessor :id

  def initialize(attributes)
    @id = attributes[:id]
    @name = attributes[:name]
  end
end
```

```ruby
class Appointment
  attr_accessor :id, :doctor, :patient, :date

  def initialize(attributes = {})
    @id = attributes[:id]
    @doctor = attributes[:doctor]
    @patient = attributes[:patient]
    @date = attributes[:date]
  end
end
```

	A	B	C	D
1	id	date	doctor_id	patient_id
2	1	03/11/14 18:20	3	

```ruby
class AppointmentsController
  def create_appointment
    doctor_id = @view.ask_for_doctor_id
    patient_id = @view.ask_for_patient_id
    appointment = Appointment.new
    appointment.doctor = @doctor_repository.find(doctor_id)
    appointment.patient = @patient_repository.find(patient_id)
    @appointment_repository.add(appointment)
  end
end
```

Good luck!

Relational Database & SQL

Why do we need databases?

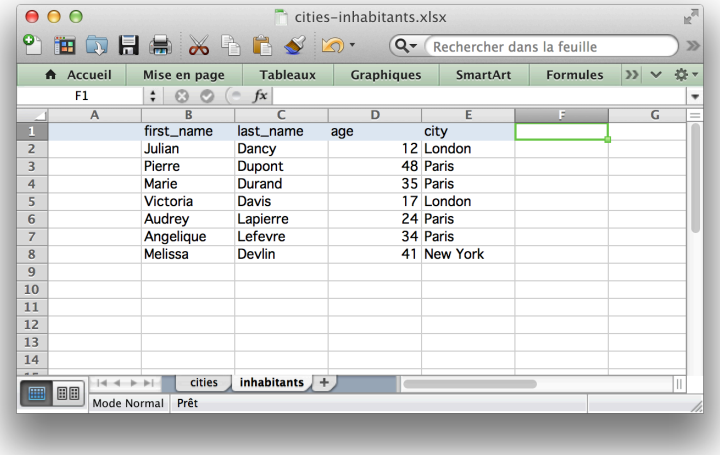
Store data. Persistently.

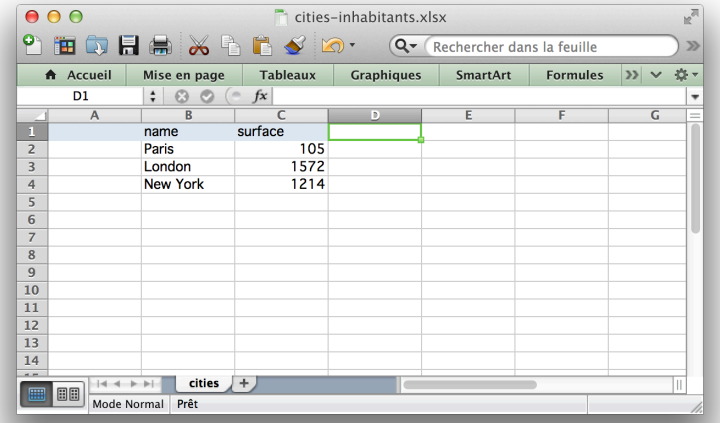
Excel

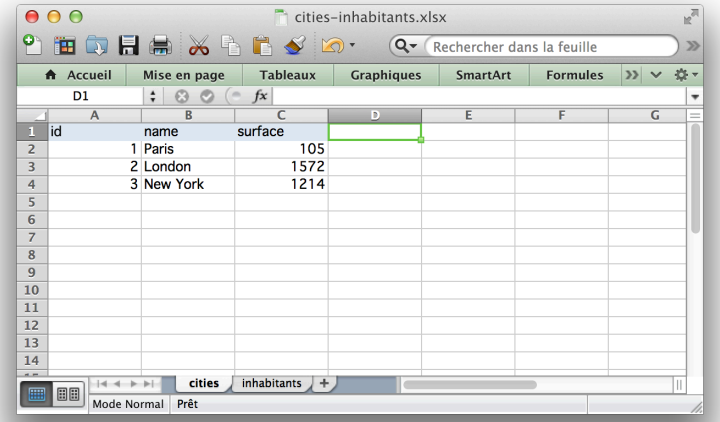
Let's start with something we all know

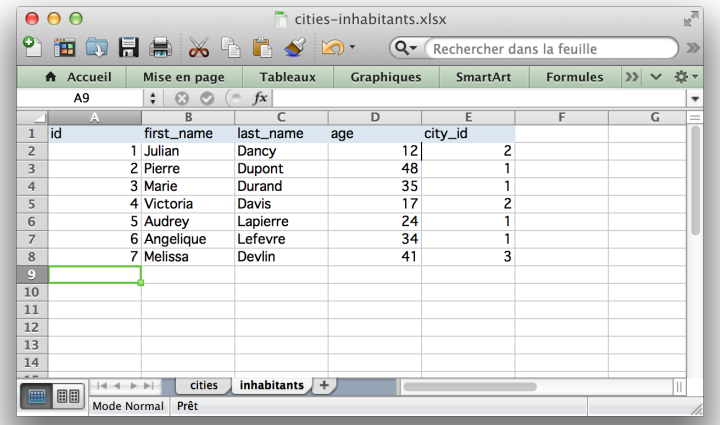
Example

Let's store **cities** and their **inhabitants** using Excel. How would you do it?







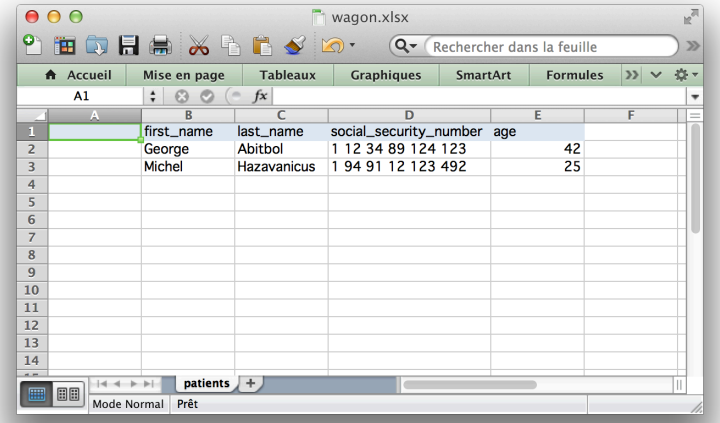


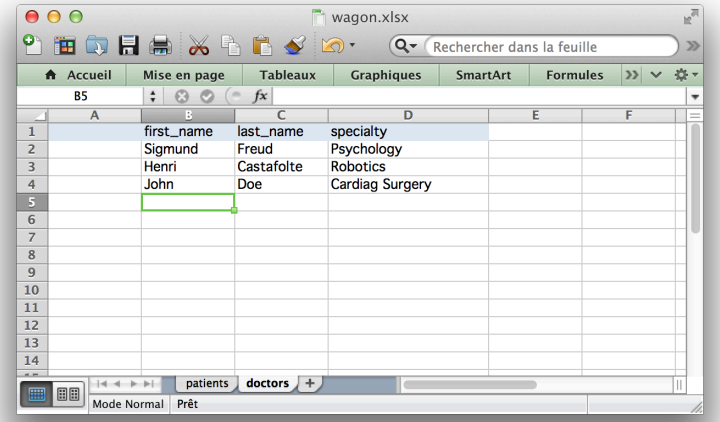
1:n relation (one to many)

An inhabitant **belongs to** one city (or has one city)

Excel++

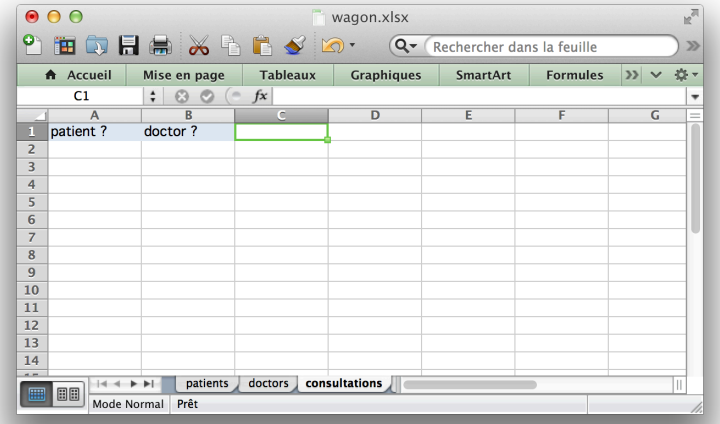
Let's go further

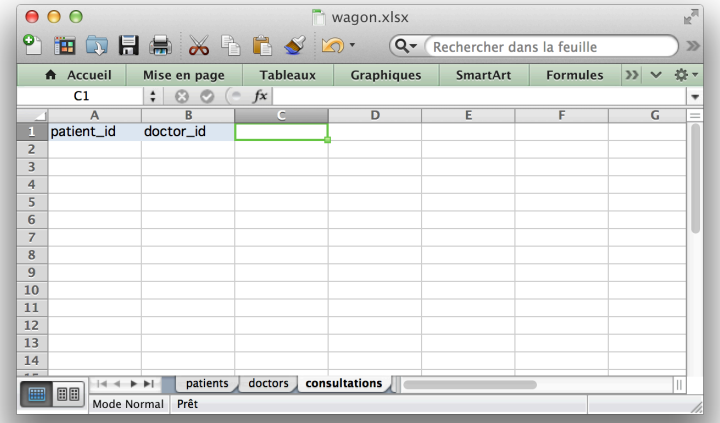


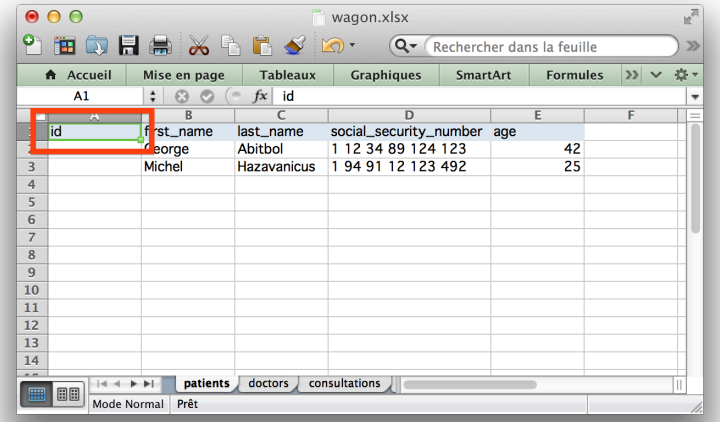


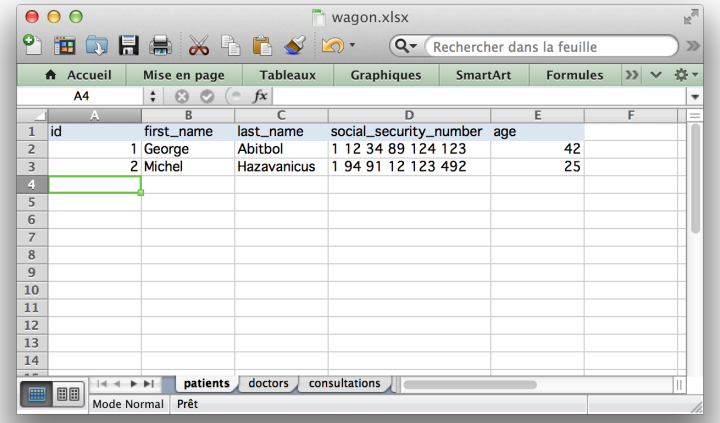
Consultations ?

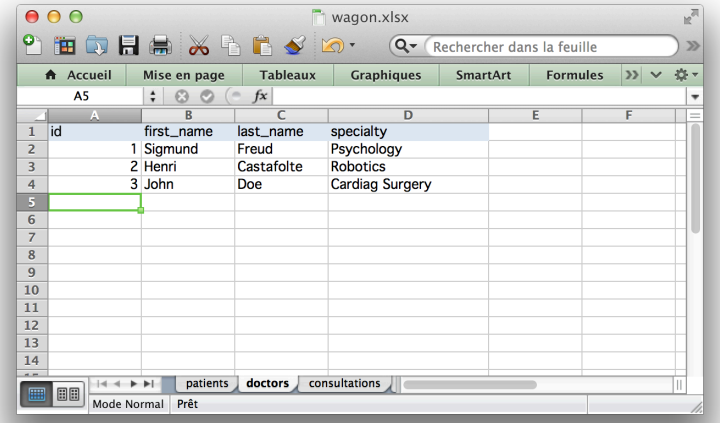
- **One** doctor can have **many** patients
- **One** patient can see **many** doctors

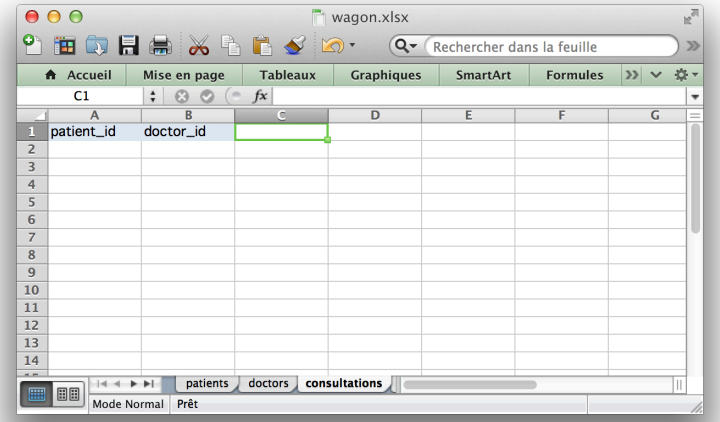



wagon.xlsx
Rechercher dans la feuille
Accueil
Mise en page
Tableaux
Graphiques
SmartArt
Formules
C1
patient_id
doctor_id
patients
doctors
consultations
Mode Normal
Prêt

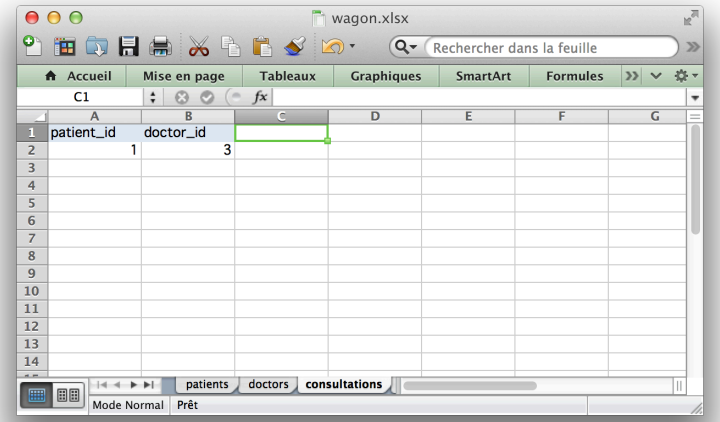

wagon.xlsx
Rechercher dans la feuille
Accueil
Mise en page
Tableaux
Graphiques
SmartArt
Formules
A1
id
id
first_name
last_name
social_security_number
age
George
Abitbol
1 12 34 89 124 123
42
Michel
Hazavanicus
1 94 91 12 123 492
25
patients
doctors
consultations
Mode Normal
Prêt


wagon.xlsx
Rechercher dans la feuille
Accueil
Mise en page
Tableaux
Graphiques
SmartArt
Formules
A4
id
first_name
last_name
social_security_number
age
1
George
Abitbol
1 12 34 89 124 123
42
2
Michel
Hazavanicus
1 94 91 12 123 492
25
patients
doctors
consultations
Mode Normal
Prêt





George Abitbol (`id = 1`) has seen Doctor John Doe (`id = 3`)



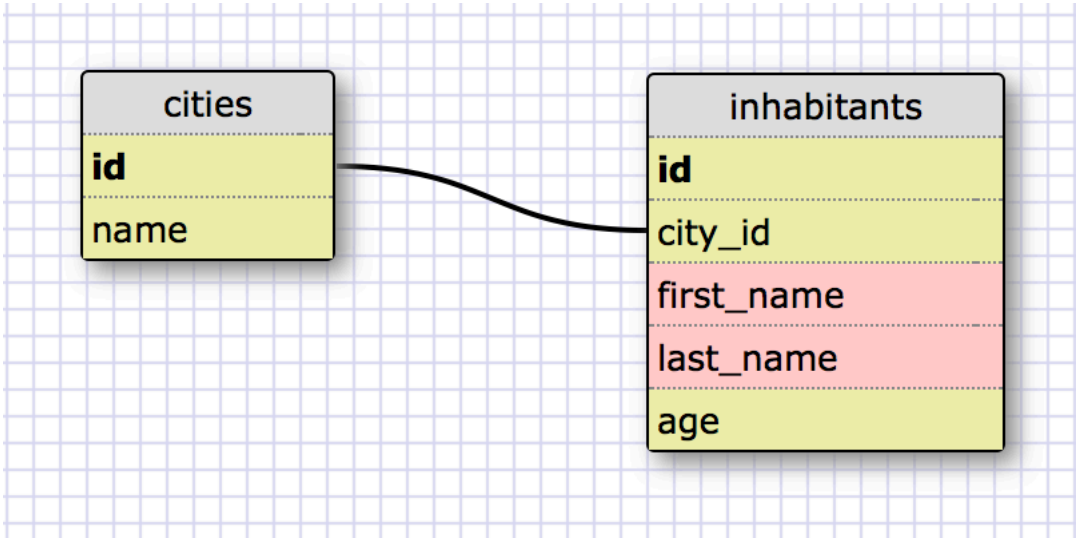
n:n relation (many to many)

A patient **has many** doctors and a doctor **has many** patients.

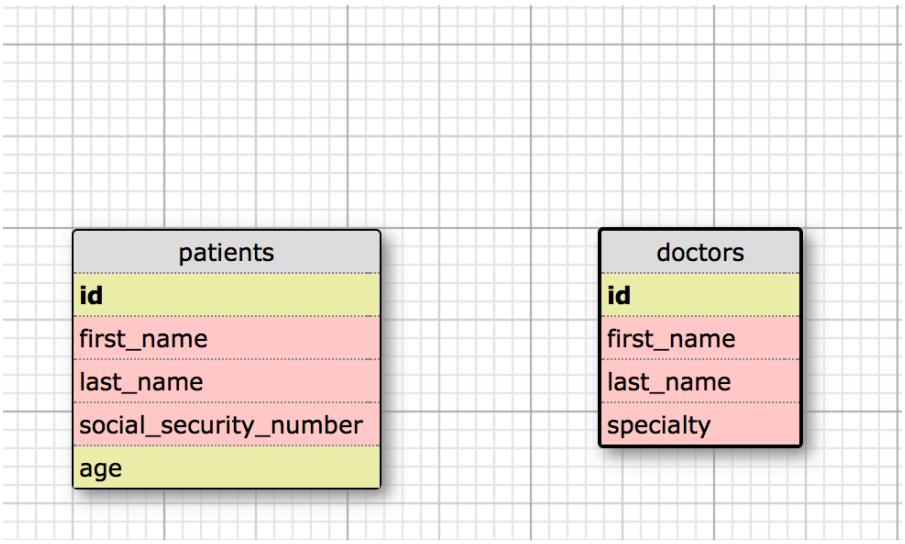
You can download this example: consultations.xslx (/assets/patients-doctors-56eaccffe9a741fbebec5b9c09922abdfd39055dcd1880eb5e0bbdc547e558b6.xlsx)

Relational Database

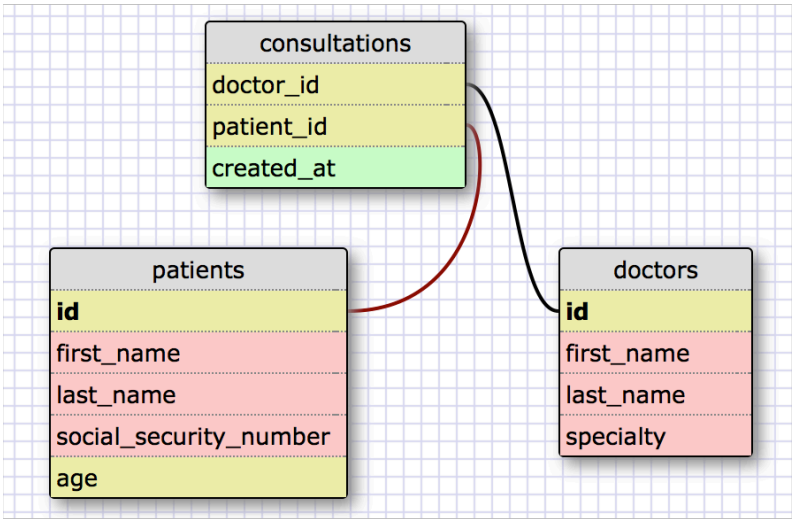
1:n



n:n ?



n:n



Vocabulary

- A schema is composed of **tables**.
- Each table has a set of **columns**.
- When inserting data in a table, you create a **record** in a new **row**.

DB Schema Composer

db.lewagon.org (http://db.lewagon.org/)

You can save/load schemas. Try with patients-doctors.xml (/assets/patients-doctors-2ee663685cecec3f9fcf22ef7a9ffef7a0dfe0dbde62a8ea5d893940b042cf70.xml)

Querying

Retrieve data using the schema.

SQL

Structured Query Language

Give me all patient names

```
SELECT first_name, last_name FROM patients
```

Give me all doctor names

```
SELECT first_name, last_name FROM doctors
```

Give me all you got about patients

```
SELECT * FROM patients
```

Give me all patients of age 21

```
SELECT * FROM patients WHERE age = 21
```

Give me all doctors of cardiac surgery specialty

```
SELECT * FROM doctors WHERE specialty = 'Cardiac Surgery'
```

Give me all surgery doctors

```
SELECT * FROM doctors WHERE specialty LIKE '%Surgery'
```

Give me all cardiac surgery doctors named Steve

```
SELECT * FROM doctors
WHERE specialty = 'Cardiac Surgery'
AND first_name = 'Steve'
```

Give me all patients ordered by age

```
SELECT * FROM patients ORDER BY age ASC
```

```
SELECT * FROM patients ORDER BY age DESC
```

How many doctors do I have?

```
SELECT COUNT(*) FROM doctors
```

Count cardiac surgery doctors

```
SELECT COUNT(*) FROM doctors WHERE specialty = 'Cardiac Surgery'
```

Count all doctors per specialty

```
SELECT COUNT(*), specialty
FROM doctors
GROUP BY specialty
```

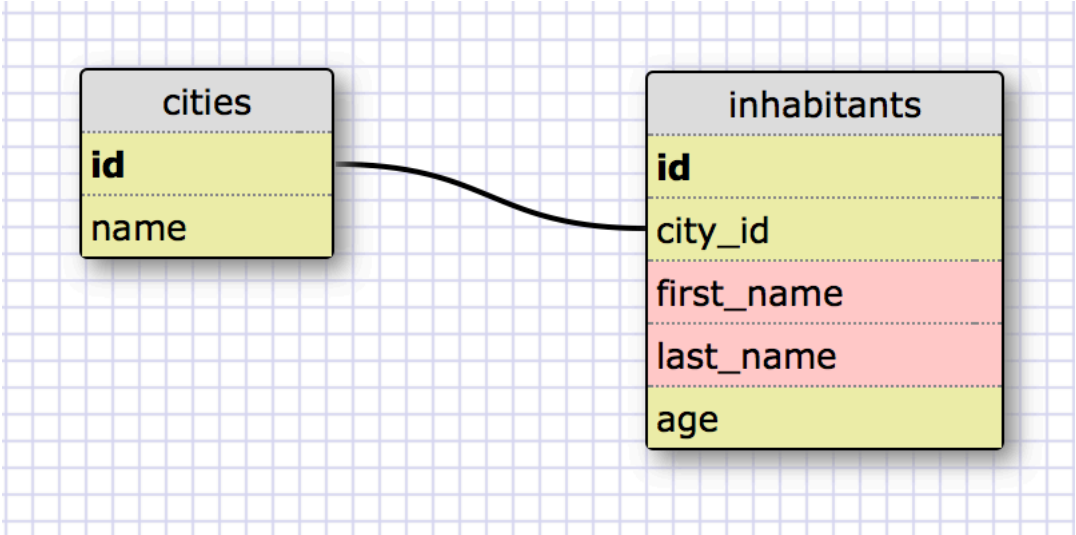
Count all doctors per specialty, order by specialty

You need to rename result column, with `AS` .

```
SELECT COUNT(*) AS c, specialty
FROM doctors
GROUP BY specialty
ORDER BY c DESC
```

Using 2 or more tables at once

Given this cities/inhabitants schema...



... and this data

cities

id	name
1	Paris
2	Brussels
3	Lille
4	Beirut
5	Bordeaux

inhabitants

id	city_id	first_name	last_name	age
1	3	Sophia	Smith	21
2	2	Jackson	Williams	30
3	5	Emma	Johnson	29
4	1	Aiden	Brown	18
5	1	Olivia	Jones	24
6	4	Liam	Miller	39
7	4	Ava	Davis	41
8	1	Lucas	Garcia	43
9	2	Isabella	Rodriguez	20
10	1	Noah	Wilson	20

Question: Give me all the inhabitants from Paris

```
SELECT * FROM inhabitants
JOIN cities ON cities.id = inhabitants.city_id
WHERE cities.name = 'Paris'
```

cities

id	name
1	Paris
2	Brussels
3	Lille
4	Beirut
5	Bordeaux

inhabitants

id	city_id	first_name	last_name	age
1	3	Sophia	Smith	21
2	2	Jackson	Williams	30
3	5	Emma	Johnson	29
4	1	Aiden	Brown	18
5	1	Olivia	Jones	24
6	4	Liam	Miller	39
7	4	Ava	Davis	41
8	1	Lucas	Garcia	43
9	2	Isabella	Rodriguez	20
10	1	Noah	Wilson	20

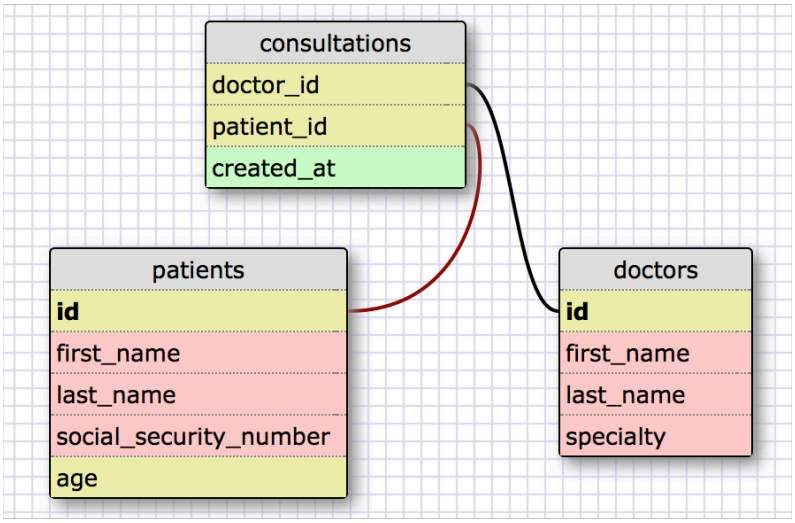
Query Result

c.id	c.name	i.id	i.city_id	i.first_name	i.last_name	i.age
1	Paris	4	1	Aiden	Brown	18
1	Paris	8	1	Lucas	Garcia	43
1	Paris	10	1	Noah	Wilson	20

Question: Give me all the adults living Paris

```
SELECT * FROM inhabitants
JOIN cities ON cities.id = inhabitants.city_id
WHERE inhabitants.age >= 18
AND cities.name = 'Paris'
```

Given this consultations schema



Question: For each consultation, give me its date, patient and doctor names

```
SELECT c.created_at, p.first_name, p.last_name, d.first_name, d.last_name
FROM consultations c
JOIN patients p ON c.patient_id = p.id
JOIN doctors d ON c.doctor_id = d.id;
```

Going further

You can read more about `INNER` (the default one), `LEFT OUTER`, `RIGHT OUTER` or `FULL OUTER JOIN` here (http://www.techonthenet.com/sql/joins.php) and there (http://www.lornajane.net/posts/2011/inner-vs-outer-joins-on-a-many-to-many-relationship).

SQLite

It is a simple database storing everything in **one** file. Great to quickly test, but not suited for production.

Installation

On OSX, run this:

```
$ brew install sqlite
```

On Ubuntu, run this:

```
$ sudo apt-get install sqlite libsqlite3-dev
```

Quick start

Create a new folder, and got into it.

Create a DB and start typing SQL queries:

```
$ sqlite3 db.sqlite
```

It will create the `db.sqlite` file.

Table creation

```
CREATE TABLE `cities` (
  `id` INTEGER PRIMARY KEY AUTOINCREMENT,
  `name` VARCHAR
);
```

Inserting and Querying

```
sqlite> INSERT INTO cities ('name') VALUES ('Paris');
sqlite> INSERT INTO cities ('name') VALUES ('London');
```

```
sqlite> .mode column
sqlite> .headers on
```

```
sqlite> SELECT * FROM cities;
id          name
----------  ----------
1           Paris
2           London
```

Help

```
sqlite> .help
```

```
.exit                 Exit this program
.header(s) ON|OFF     Turn display of headers on or off
.read FILENAME        Execute SQL in FILENAME
.schema ?TABLE?       Show the CREATE statements
                        If TABLE specified, only show tables matching
                        LIKE pattern TABLE.
.show                 Show the current values for various settings
.tables ?TABLE?       List names of tables
                        If TABLE specified, only list tables matching
                        LIKE pattern TABLE.
[...]
```

In 2 weeks: PostgreSQL

CRUD with SQL

CRUD

- **C**reate
- **R**ead
- **U**pdate
- **D**elete

Our example : A Doctors table

Table Schema

```
CREATE TABLE `doctors` (
  `id`  INTEGER PRIMARY KEY AUTOINCREMENT,
  `name` TEXT,
  `age` INTEGER,
  `specialty` TEXT
);
```

Sample Data

```
sqlite> .mode column
sqlite> .headers on
sqlite> SELECT * FROM doctors;
id          name        age         specialty
----------  ----------  ----------  ----------------
1           John Smith  39          Anesthesiologist
2           Emma Reale  31          Cardiologist
```

You can download 02*sql*crud_doctors.db (/assets/02_sql_crud_doctors-135696377f22460ebf3508ef121c633c4221a61f1a451ba81db94c5a334ce931.db)

READ

SQL keyword: **`SELECT`**

`SELECT` (http://www.techonthenet.com/sql/select.php) keyword to **fetch records** from the DB.

Selecting all the rows

```
SELECT * FROM doctors
```

Selecting a specific row

```
SELECT * FROM doctors WHERE id = 2
```

No quotes around `2` as it's a number.

CREATE

SQL keyword: **`INSERT`**

`INSERT` (http://www.techonthenet.com/sql/insert.php) keyword to **add records** to a table.

```
INSERT INTO doctors (name, age, specialty)
VALUES ('Dr. Dolladille', 45, 'Dentist')
```

inserts a new record:

```
sqlite> SELECT * FROM doctors;
id          name        age         specialty
----------  ----------  ----------  ----------------
1           John Smith  39          Anesthesiologist
2           Emma Reale  31          Cardiologist
3           Dr. Dollad  45          Dentist
```

Autoincrement

Remember the schema definition:

```
CREATE TABLE `doctors` (
  `id`  INTEGER PRIMARY KEY AUTOINCREMENT,
  `name` TEXT,
  `age` INTEGER,
  `specialty` TEXT
);
```

Let the DB handle `id`

Ensure that all records have a different id.

UPDATE

SQL keyword: **`UPDATE`**

`UPDATE` (http://www.techonthenet.com/sql/update.php) keyword to **update records** in a table.

Update one record

```
UPDATE doctors SET age = 40, name = 'John Smith' WHERE id = 3
```

DELETE

SQL keyword: **`DELETE`**

`DELETE` (http://www.techonthenet.com/sql/delete.php) keyword to **delete records** from a table.

Delete a single record

The most used

```
DELETE FROM doctors WHERE id = 32
```

Delete every record

```
DELETE FROM doctors
```

Deletes every record in the `doctors` table.

SQlite and Ruby

Some tips for the exercise

Model

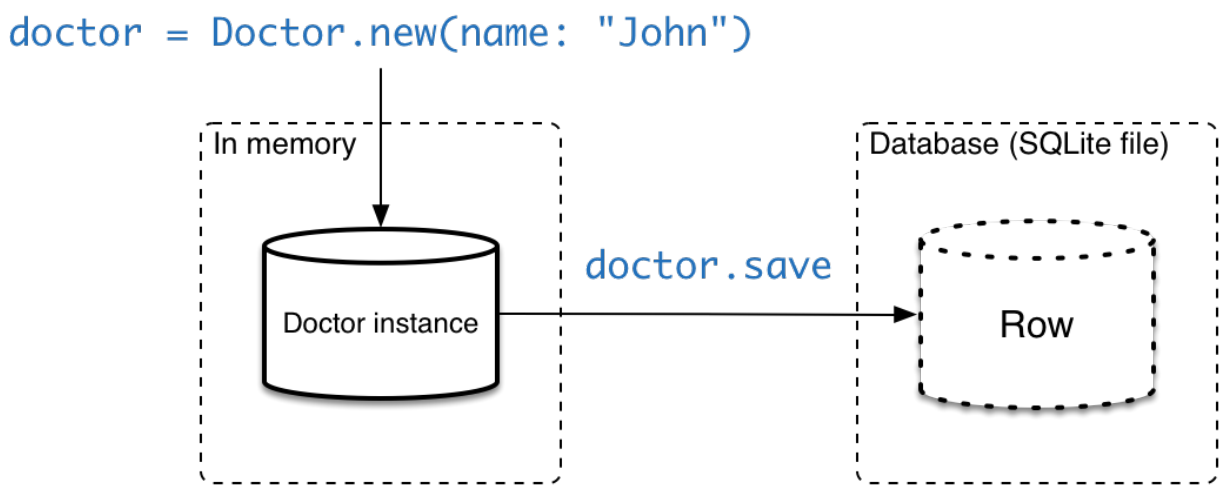
We will use a model class (the M in MVC).

```ruby
# app/models/doctor.rb
class Doctor
  attr_reader :id

  def initialize(attributes = {})
    @id = attributes[:id]
    # TODO: store other attributes as instanced variable (exercise)
  end
end
```

Example

```ruby
doctor = Doctor.new(name: 'John', age: 42)
doctor.id
# => nil
```

In-memory vs Stored in Database



You'll have to write the `save` method in your model.

DB connection

For today, we'll provide a global variable named `DB`

```ruby
require 'sqlite3'
DB = SQLite3::Database.new("db/doctors.db")
```

Then you can use it in the model to run queries:

```ruby
rows = DB.execute('SELECT * FROM doctors')
```

What about the id?

Not your job to set it. Your job to retrieve it and update the `@id` accordingly upon first save.

```ruby
DB.execute("INSERT INTO doctors (name, age) VALUES ('John', 42)")
DB.last_insert_row_id
```

Something useful...

```ruby
doctors = DB.execute("SELECT name, age FROM doctors")
# => [
#      [ "John Smith", 39 ],
#      [ "Emma Reale", 31 ]
#    ]
```

```ruby
DB.results_as_hash = true
doctors = DB.execute("SELECT name, age FROM doctors")
# => [
#      { "name" => "John Smith", "age" => 39 , 0 => "John Smith", 1 => 39 },
#      { "name" => "Emma Reale", "age" => 31 , 0 => "Emma Reale", 1 => 31 }
#    ]
```

```ruby
doctor = doctors.first
name = doctor["name"]
age = doctor["age"]
puts "Doctor #{name} is #{age} years old"
```

ActiveRecord Basics

Old School: SQL commands in code

We did it. Painful

What we **don't want** to write

```
DB = SQLite3::Database.new("db/database.sqlite3")
DB.execute("INSERT INTO restaurants SET name = 'La Tour d\'Argent'")
```

What we **want** to write

```
restaurant = Restaurant.new(name: "La Tour d'Argent")
restaurant.save
```

Both will result in adding a row in database:

```
+----+--------------------+
| id | name               |
+----+--------------------+
| 1  | "La Tour d'Argent" |
+----+--------------------+
```

We'll use the Active Record Pattern (http://en.wikipedia.org/wiki/Active_record_pattern) (2003)

> The active record pattern is a pattern found in software that stores its data in relational databases. [...] The interface of an object conforming to this pattern would include functions such as [`save` , `destroy`], plus [**instance variables**] that correspond [...] to the **columns** in the underlying database table. *Wikipedia*

An **object** is represented as a **row** of a table in a relational database.

Active Record - Setup

```
gem install activerecord -v 5.0.3
# Successfully installed activerecord-5.x
# 1 gem installed
```

Let's start from a boilerplate

```
cd ~/tmp
git clone git@github.com:lewagon/active-record-boilerplate.git
cd active-record-boilerplate
bundle install
```

`rake`

`rake` is a **task launcher**. All tasks are defined in a `Rakefile` .

```
rake -T
# rake rubocop     # Look for style guide offenses in your code
# rake spec        # Run RSpec code examples
```

Default task:

```
# Rakefile
task default: [:rubocop, :spec]
```

Create the database

First specify a database

```
# config/database.yml
development:
  adapter: sqlite3                  # Database vendor (soon, postgres)
  database: db/development.sqlite3  # File path of the database
```

Then create it, run this in your terminal

```
rake db:create
ll db
```

Drop the database

To get rid of your database and **all its data** (!!)

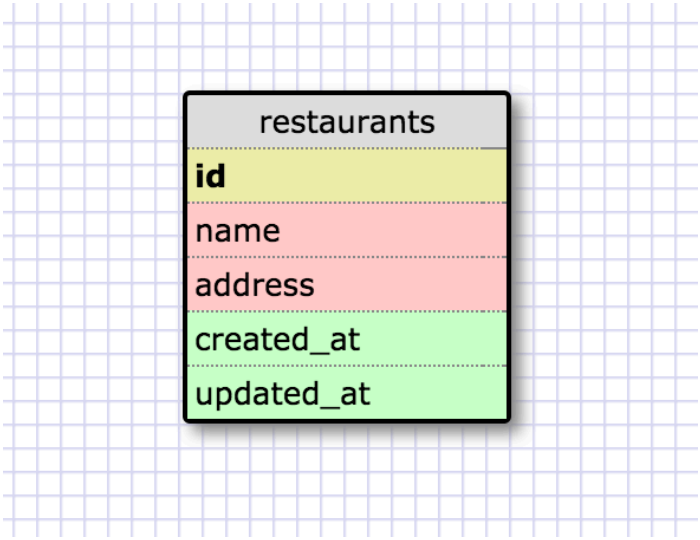
```
rake db:drop
ll db
```

Migrations

`rake db:create` created a database with an empty schema.

You need to add tables, the one you figured out with db.lewagon.org (http://db.lewagon.org)

Let's model a Restaurant!



Each time you change your schema (add / remove table), (add / remove column), you need to create a **schema migration**.

It's a change in the **structure** of your schema, **not the data**.

```
CREATE TABLE ...
UPDATE TABLE ...
DROP TABLE ...
```

Table creation migration

ActiveRecord helps us to write schema migration with **ruby**, not SQL.

```
TIMESTAMP=`rake db:timestamp`
touch db/migrate/${TIMESTAMP}_create_restaurants.rb
```

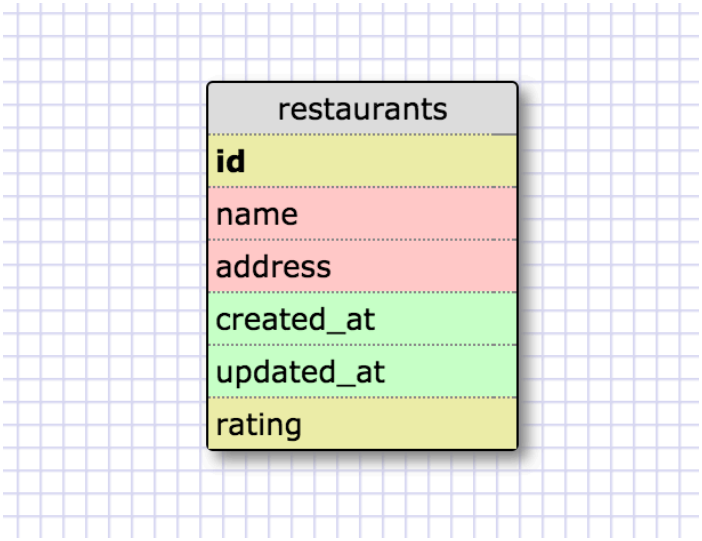
```
# db/migrate/**********_create_restaurants.rb
class CreateRestaurants < ActiveRecord::Migration[5.0]
  def change
    create_table :restaurants do |t|
      t.string    :name
      t.string    :address
      t.timestamps # add 2 columns, `created_at` and `updated_at`
    end
  end
end
```

Run the migrations

```
rake db:migrate
```

ActiveRecord figures out which migration to run on your schema (the ones not already run).

Oops, the client wants ratings!



Add a column migration

```
TIMESTAMP=`rake db:timestamp`
touch db/migrate/${TIMESTAMP}_add_rating_to_restaurants.rb
```

```
# db/migrate/********_add_rating_to_restaurants.rb
class AddRatingToRestaurants < ActiveRecord::Migration[5.0]
  def change
    add_column :restaurants, :rating, :integer, default: 0, null: false
  end
end
```

```
rake db:migrate
```

Schema is ready!

```
sqlite3 db/development.sqlite3
```

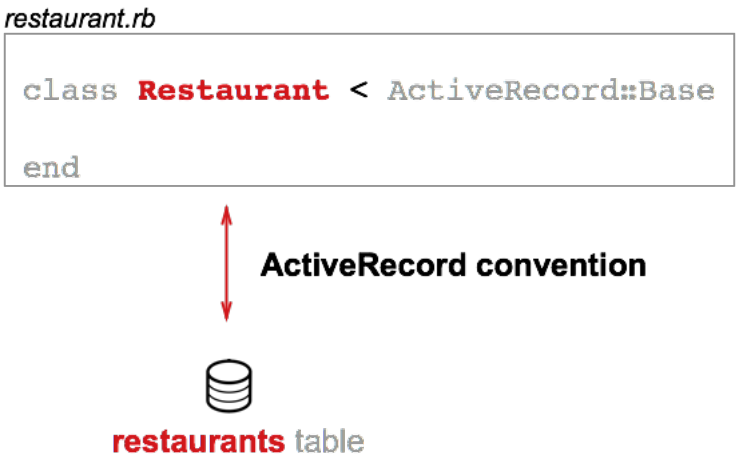
```
sqlite> .schema
sqlite> .mode column
sqlite> .headers on
sqlite> SELECT * FROM schema_migrations;
```

Model

We don't have a `Restaurant` class yet for our `restaurants` table.

```
# app/models/restaurant.rb
class Restaurant < ActiveRecord::Base
end
```

ActiveRecord naming convention



Remember the convention

- Table name: **lowers*nake*case**, **plural** form (store several rows)
- Model class name: **UpperCamelCase**, **singular** form (mapped to 1 row)

Rails is full of **Convention** over **Configuration**.

Unleash the magic

Creating a record

```
rake console
```

```
restaurant = Restaurant.new(
  name: "La Tour d'Argent",
  address: "15 Quai de la Tournelle, 75005 Paris"
)
restaurant.save
# => INSERT INTO restaurants SET name = 'La Tour d\'Argent', address = '[...]'
```

```
# app/models/restaurant.rb
class Restaurant < ActiveRecord::Base

end
```

> Where's the code? *A Java Developer*

Retrieving all records

```
restaurants = Restaurant.all
# => SELECT * FROM restaurants
```

Calling `all` - a **class method** - on your model gives you an `Array` of all your records for this model. Elements of the array are **instances** of the model class `Restaurant`.

Counting records of a table

```
Restaurant.count
```

(Faster than `Restaurant.all.length`, why?)

Retrieving a specific record

```
restaurant = Restaurant.find(1)
# => SELECT * FROM restaurants WHERE id = 1
```

The `restaurant` variable is an **instance** of the model class `Restaurant`.

```
restaurant.name
# => "La Tour d'Argent"

restaurant.address
# => "15 Quai de la Tournelle, 75005 Paris"
```

Updating an existing record

```
restaurant = Restaurant.find(1)
restaurant.address = '14 Quai de la Tournelle, 75005 Paris'
restaurant.save
# => UPDATE articles SET address = '14 Quai [...]' WHERE id = 1
```

Deleting a record

```
restaurant = Restaurant.find(1)
restaurant.destroy
```

Filtering records

```
restaurants = Restaurant.where(rating: 3)
# => SELECT * FROM restaurants WHERE rating = 3
```

As for the `all` class method, `where` returns an `Array` of class model instances.

`where like`

```
restaurants = Restaurant.where("name LIKE ?", "%tour%")
# => SELECT * FROM restaurants WHERE name LIKE '%tour%'
```

Finding by attribute

`find` => Returns **one** record

```
restaurant = Restaurant.find_by_name("La Tour d'Argent")
# => SELECT * FROM restaurants WHERE name = 'La Tour d\'Argent' LIMIT 1
```

First and Last

```
Article.first # Returns the first article in DB
Article.last  # Returns the last article in DB
```

Depends on the primary key (`id` column).

Seeds

The Rails Way

```
# db/seeds.rb
puts 'Creating restaurants...'
tour_d_argent = Restaurant.new(name: "La Tour d'Argent")
tour_d_argent.save!

chez_gladines = Restaurant.new(name: "Chez Gladines")
chez_gladines.save!
puts 'Finished!'
```

To run the seed, open your terminal:

```
rake db:seed
```

Easily fake data

```
gem install faker
```

```
# db/seeds.rb
require 'faker'

puts 'Creating 100 fake restaurants...'
100.times do
  restaurant = Restaurant.new(
    name:    Faker::Company.name,
    address: "#{Faker::Address.street_address}, #{Faker::Address.city}",
    rating:  (0..5).to_a.sample
  )
  restaurant.save!
end
puts 'Finished!'
```

See also

- `Faker::Address` / `Faker::Name` / `Faker::Lorem`
- `Faker::Business` / `Faker::Commerce` / `Faker::Company`
- `Faker::Internet` / `Faker::Number` / `Faker::PhoneNumber`

https://github.com/stympy/faker

Growth Hacking

Web scrapping (nokogiri (http://www.nokogiri.org/)) or API calls (rest-client (https://github.com/rest-client/rest-client))

Fetch GitHub repos from @lewagon

```
gem install rest-client
```

```
require "json"
require "rest-client"

response = RestClient.get "https://api.github.com/users/lewagon/repos"
repos = JSON.parse(response)
# => repos is an `Array` of `Hashes`.

repos.size
# => 30
```

Browser extension to install: JSON Formatter on Chrome (https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa?hl=en).

Read this great doc

http://guides.rubyonrails.org/active*record*basics.html

Skip the chapter 6 (Validations) and 7 (Callbacks) for now.

Your turn!

ActiveRecord Advanced

Topics

- Associations
- Validations

Let's start from a boilerplate

```
cd ~/tmp
git clone git@github.com:lewagon/active-record-boilerplate.git
cd active-record-boilerplate
bundle install
```

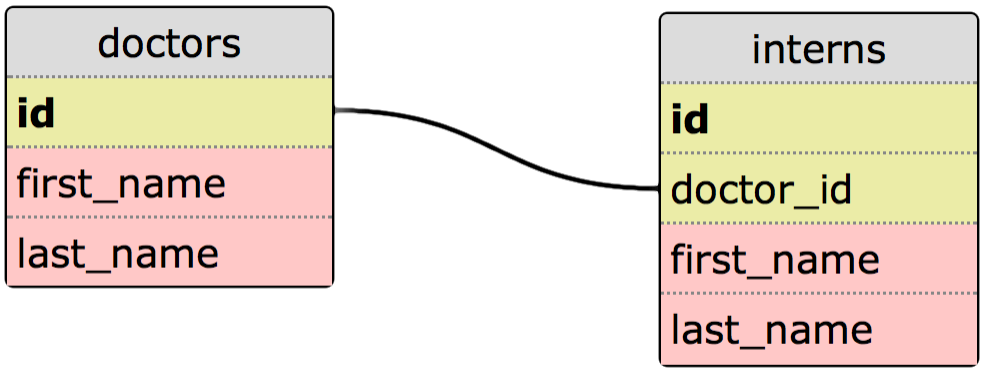
Associations

Example: a Hospital

Same procedure. Always

1. Draw the tables in db.lewagon.org (http://db.lewagon.org)
2. Create the migrations
3. Create the models

A doctor has many interns



Migration to create doctors

```ruby
# db/migrate/20141027100300_create_doctors.rb
class CreateDoctors < ActiveRecord::Migration[5.0]
  def change
    create_table :doctors do |t|
      t.string      :first_name
      t.string      :last_name
      t.timestamps
    end
  end
end
```

Migration to create interns

```ruby
# db/migrate/20141027100400_create_interns.rb
class CreateInterns < ActiveRecord::Migration[5.0]
  def change
    create_table :interns do |t|
      t.string      :first_name
      t.string      :last_name
      t.references  :doctor, foreign_key: true, index: true
      t.timestamps
    end
  end
end
```

Note the `t.references` .

Create 2 models

```ruby
# app/models/doctor.rb
class Doctor < ActiveRecord::Base
  has_many :interns
end
```

```ruby
# app/models/intern.rb
class Intern < ActiveRecord::Base
  belongs_to :doctor
end
```

Read it:

- A **`Doctor has_many interns`**
- An **`Intern belongs_to`** a **`doctor`**

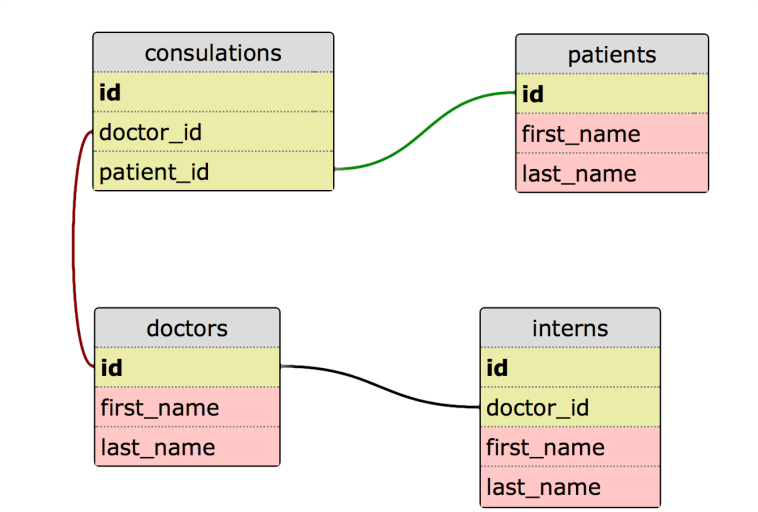
Create interns

```ruby
doctor = Doctor.new(first_name: "Gregory", last_name: "House")
doctor.save

intern = Intern.new(first_name: "Allison", last_name: "Cameron")
intern.doctor = doctor
intern.save
```

Retrieve interns for a given doctor:

```ruby
doctor = Doctor.last
doctor.interns
# => `Array` of `Intern` instances
```

Let's add `patients` / `consultations` table



Migration to create patients

```
# db/migrate/20141027114700_create_patients.rb
class CreatePatients < ActiveRecord::Migration[5.0]
  def change
    create_table :patients do |t|
      t.string      :first_name
      t.string      :last_name
      t.timestamps
    end
  end
end
```

Migration to create consultations

```
# db/migrate/20141027114800_create_consultations.rb
class CreateConsultations < ActiveRecord::Migration[5.0]
  def change
    create_table :consultations do |t|
      t.references :doctor, foreign_key: true, index: true
      t.references :patient, foreign_key: true, index: true
      t.timestamps
    end
  end
end
```

Create 2 new models

```
# app/models/patient.rb
class Patient < ActiveRecord::Base
  has_many :consultations
end
```

```
# app/models/consultation.rb
class Consultation < ActiveRecord::Base
  belongs_to :patient
  belongs_to :doctor
end
```

What about the `Doctor` model?

```
# app/models/doctor.rb
class Doctor < ActiveRecord::Base
  has_many :interns

  # Do not forget to add this! :)
  has_many :consultations
end
```

How can I retrieve patients for a given doctor?

```ruby
doctor = Doctor.last

patients = []
doctor.consultations.each do |consultation|
  patients << consultation.patient
end
# => `Array` of `Patient` instance
```

Use `:through`

```ruby
# app/models/doctor.rb
class Doctor < ActiveRecord::Base
  has_many :interns
  has_many :consultations

  has_many :patients, through: :consultations
end
```

Then you can do:

```ruby
doctor = Doctor.last
doctor.patients
# => `Array` of `Patient` instance
```

Migration to update a table

Right now we've always created migrations to `create` new tables. But 90% of the time, we write migrations to `update` a table schema (adding or removing columns).

Adding / Removing a column

```ruby
# db/migrate/20141027200800_add_age_to_patients.rb
class AddAgeToPatients < ActiveRecord::Migration[5.0]
  def change
    add_column :patients, :age, :integer
  end
end
```

```ruby
# db/migrate/20141027201000_remove_age_from_patients.rb
class RemoveAgeFromPatients < ActiveRecord::Migration[5.0]
  def change
    remove_column :patients, :age, :integer
  end
end
```

Adding a foreign key

Say that an intern has to take care of many patients.

```ruby
# db/migrate/20141027201300_add_intern_reference_to_patients.rb
class AddInternReferenceToPatients < ActiveRecord::Migration[5.0]
  def change
    add_reference :patients, :intern, foreign_key: true, index: true
  end
end
```

Documentation

http://guides.rubyonrails.org/association_basics.html

(There is also `has_one` which is rarely used, and `has_and_belongs_to_many` that you should not use (http://blog.flatironschool.com/post/35346328762/why-you-dont-need-h
and-belongs-to-many))

Validations

Concept

We want to **check the validity** of a model instance before saving it to the Database

Example

Suppose we have this:

```ruby
doctor = Doctor.new(first_name: "Gregory")
```

We want to enforce that all doctors have a `last_name` .

Therefore `doctor` is invalid.

```
doctor.last_name = "House"
```

Now, it's valid.

Presence validation

```
# app/models/doctor.rb
class Doctor
  has_many :interns
  has_many :consultations
  has_many :patients, through: :consultations

  validates :last_name, presence: true
end
```

Checking if a model is valid:

```
doctor = Doctor.new(first_name: "Gregory")
doctor.valid?
# => false
doctor.errors.messages
# => { last_name: [ "can't be blank" ] }
```

```
doctor.last_name = "House"
doctor.valid?
# => true
```

Automatically happens

When calling the `save` method when your instance is invalid, the record **won't** be inserted in the database!

```
doctor = Doctor.new(first_name: "Gregory")
doctor.save
# => false
# The record has not been inserted
```

`save` returns `true` if the record has been inserted.

`save!` **raises** an Exception if the record is invalid.

Validations type

We've just seen the presence validation

Uniqueness

All doctors must have a unique `last_name` .

```
class Doctor < ActiveRecord::Base
  validates :last_name, uniqueness: true
end
```

Or: All doctors must have a unique `first_name last_name` combination.

```
class Doctor < ActiveRecord::Base
  validates :first_name, uniqueness: { scope: :last_name }
end
```

Length

Doctor last names must be at least 3 characters.

```
class Doctor < ActiveRecord::Base
  validates :last_name, length: { minimum: 3 }
end
```

Format

Doctor email (column not yet created, migration!) must match a Regex.

```
class Doctor < ActiveRecord::Base
  validates :email, format: { with: /\A.*@.*\.com\z/ }
end
```

For all possibilities, read the documentation (http://guides.rubyonrails.org/active_record_validations.html)

Happy Hacking!

HTML & CSS basics

Objectives of today

- Learn HTML/CSS basics
- Code a cool profile page (http://lewagon.github.io/html-css-challenges/04-advanced-selectors/)
- Put it online with Github Pages (https://pages.github.com/)

For most advanced (http://lewagon.github.io/html-css-challenges/05-fixed-sidebar/)

Front-developer setup

Sublime Packages

1. Open Sublime Text
2. `Ctrl + Shift + p` => type `"Package Control: Install Package"`
3. Search for packages in Package Control

Cool packages

- Emmet
- Color Picker (`⌘ ⇧ C`)
- SCSS
- Color Highlighter

Browser plugins

- FontFace Ninja (http://fontface.ninja/)
- Colorzilla (http://www.colorzilla.com/) (use Sip (https://itunes.apple.com/fr/app/sip/id507257563?mt=12) on Mac)
- JSON Formatter (https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa?hl=en) to have nice JSON display in browser.
- Postman (https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm) to simulate HTTP requests (useful when we will play with AJAX).

Live-code

Let's create a project with:

- an `index.html` file
- a `style.css` file
- an `images` folder

Let's play 30 seconds with Emmet (http://emmet.io/) in the HTML and Color Picker (http://weslly.github.io/ColorPicker/) in the CSS.

Front-end languages

The languages **your browser speaks**

Content

This is what Google cares about. Think of SEO.

Appearance

How does it look?

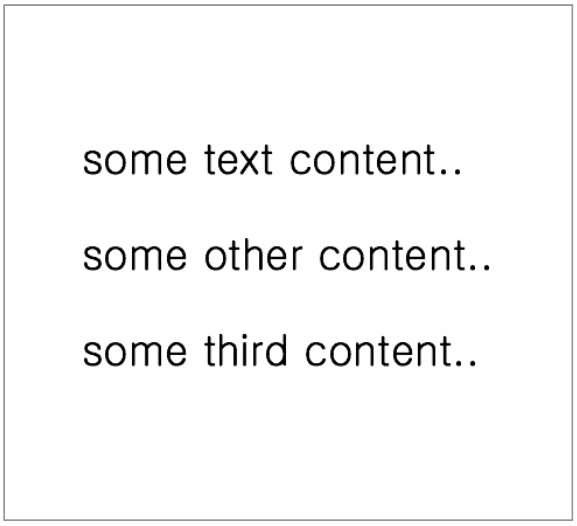
Dynamic Behavior

Animate stuff on screen. Client-side form validation.

It's a **markup** language (== structure)

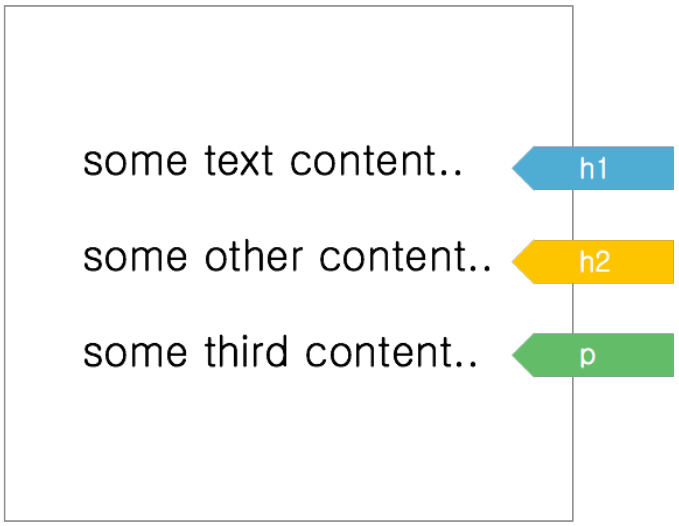
HTML

Your page has different contents



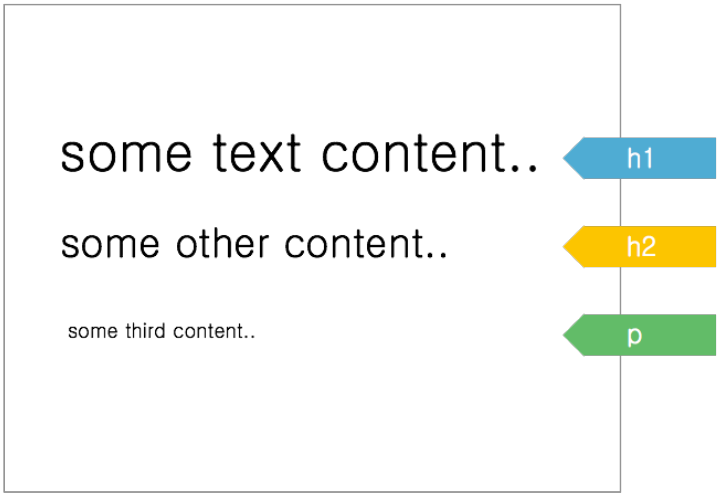
HTML

HTML tags help you **identify** content



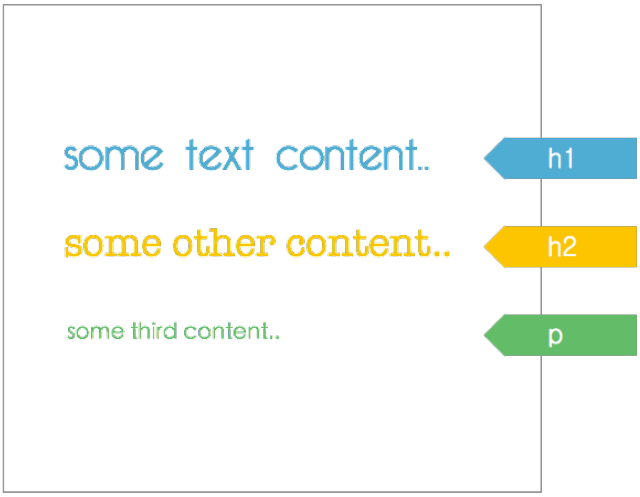
HTML

Hence, **browser default styles** will apply



HTML

And you will also be able to apply **your own style rules** if you want



HTML Skeleton

```
<!DOCTYPE html>




<!-- end of file -->
```

HTML Skeleton

```
<!DOCTYPE html>
<html>




</html>
<!-- end of file -->
```

HTML Skeleton

```
<!DOCTYPE html>
<html>
  <head>


  </head>
  <body>

  </body>
</html>
<!-- end of file -->
```

HTML Skeleton - head

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"></meta>
    <title>TEST</title>
  </head>
  <body>

  </body>
</html>
<!-- end of file -->
```

HTML Skeleton - head & Google

```
<head>
  <!-- Google result text-->
  <title>Le Wagon - Apprendre à coder - Bootcamp en ruby on Rails à Paris</title>
  <!-- Google result description-->
  <meta name="description" content="Le Wagon est la meilleure formation au développement web..">
</head>
```

protip: max 55 characters for title & 160 characters for description.

HTML Skeleton - head & Facebook

```
<head>
  <meta property="og:title" content="Le Wagon - The French innovative coding school">
  <meta property="og:image" content="facebook-card.jpg">
  <meta property="og:description" content="Le Wagon is the best French coding school for entrepreneurs. Checkout by yourself the projects of
ur students. You will be impressed.">
  <meta property="og:site_name" content="Le Wagon"/>
</head>
```

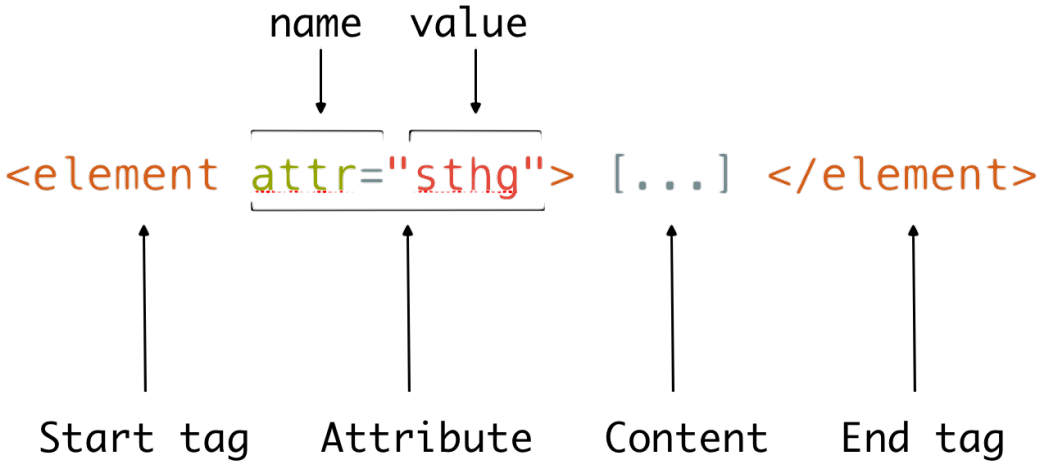
HTML Skeleton - head & Twitter

```
<head>
  <meta name="twitter:card" content="summary_large_image">
  <meta name="twitter:site" content="@Lewagonparis">
  <meta name="twitter:title" content="Le Wagon - The French innovative coding school">
  <meta name="twitter:description" content="Le Wagon is the best French coding school for entrepreneurs. Checkout by yourself the projects c
our students. You will be impressed.">
  <meta name="twitter:creator" content="@Lewagonparis">
  <meta name="twitter:image:src" content="http://twitter-card.jpg">
</head>
```

HTML Skeleton - body

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello world</title>
  </head>
  <body>
    <h1>Hello buddies!</h1>
  </body>
</html>
<!-- end of file -->
```

Basic syntax



Example

```
<a href="http://www.lewagon.org" target="_blank">
  Le Wagon
</a>
```

Result: Le Wagon (http://www.lewagon.org)

Quizz

- What is the element's name?
- What is the element's content?
- What are the 2 attributes (name and value)?

Titles

```
<h1>[...]</h1>  <!-- Only one per page! SEO important -->

<h2>[...]</h2>
<h3>[...]</h3>
<h4>[...]</h4>
<h5>[...]</h5>
<h6>[...]</h6>
```

Paragraphs

```
<p>
  Lorem ipsum dolor sit amet, consectetur adipisicing elit.
  Veritatis laboriosam mollitia autem at ab omnis iure quis
  asperiores inventore eos nam aut iusto officiis deserunt
  nihil, sequi tempore impedit quae?
</p>
```

Emphasize

```
<p>
  You can emphasize <em>some words</em>,
  and even <strong>more if needed</strong>
</p>
```

Lists

```
<h2>Shopping List</h2>
<ul>
  <li>Milk</li>
  <li>Butter</li>
</ul>

<h2>World Cup 2014</h2>
<ol>
  <li>Germany</li>
  <li>Argentina</li>
  <li>Netherlands</li>
  <li>Brazil</li>
</ol>
```

Images

```
<img src="logo.png" alt="Le Wagon logo">
```

Tables

```
<table>
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Boris</td>
      <td>Paillard</td>
    </tr>
    [...]
  </tbody>
</table>
```

Forms

```
<form>
  <input type="email">
  <input type="password">
  <input type="submit" value="Log in">
</form>
```

Much more

- codeguide.co (http://codeguide.co)
- MDN reference (https://developer.mozilla.org/en/docs/Web/HTML/Element)

Live-code

Let's **live-code** a HTML profile page without design (http://lewagon.github.io/html-css-challenges/01-profile-content/)

Web without CSS ?

Cut the <head> on lewagon.com (http://lewagon.com) with Chrome dev tool.

Linking stylesheet to HTML page

index.html

```html
<head>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Hello</h1>
</body>
```

style.css

```css
h1 {
  color: red;
}
```

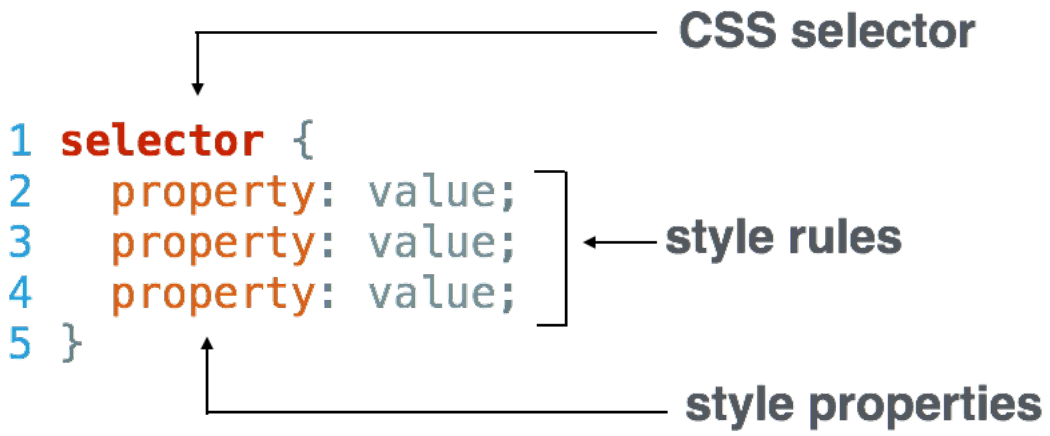
CSS syntax

```
selector {
  property: value;
  property: value;
  property: value;
}
```

selection of one or more elements of the page..

..on which we define style rules

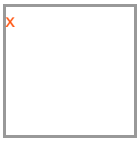
CSS vocabulary



Example

Colors

```
color: #FF530D;
color: rgb(255, 83, 13);
color: rgba(255, 83, 13, 1.0);
```



Colors - Tips

```
body {
  color: rgb(10, 10, 10);
}
```

- **RGB** stands for **R**ed **G**reen **B**lue
- each value is between 0 and 255
- for same values of R, G and B, you are on the grey scale

Text vs Background

text color

```
1 body {
2  color: orange;
3  background-color: yellow;
4 }
```

background color

Background image

```
1 body {
2  color: orange;
3  background-image: url("background.jpg");
4 }
```

background-image

Fonts - family (1)

font-family
font-size
line-height
font-weight
letter-spacing
color
text-decoration
text-align
font-style
text-transform

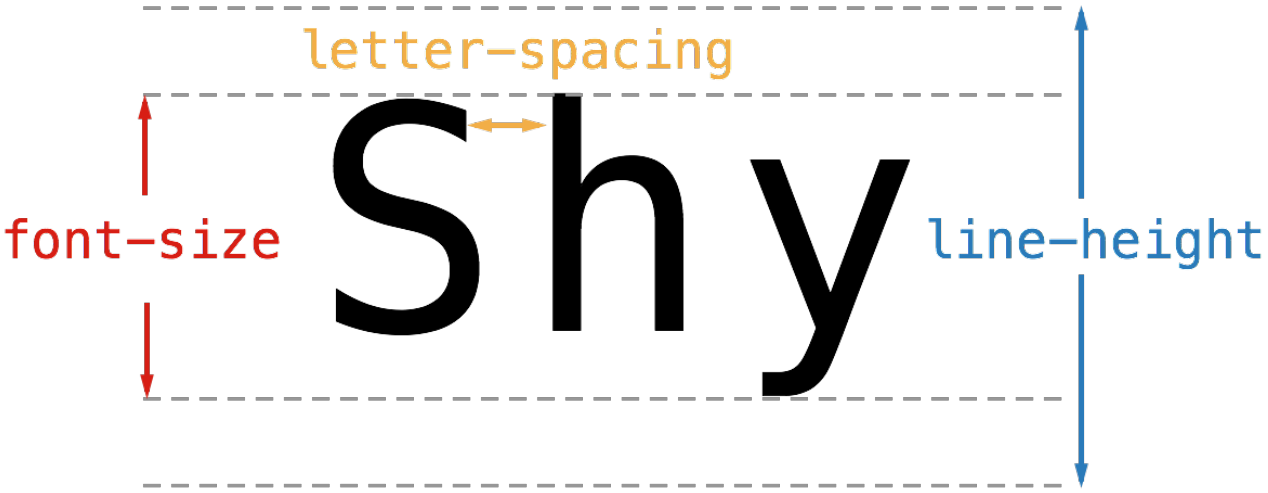
Fonts - family (2)

font-family
font-size
line-height
font-weight
letter-spacing
color
text-decoration
text-align
font-style
text-transform

Fonts - family (3)

font-family
font-size
line-height
font-weight
letter-spacing
color
text-decoration
text-align
font-style
text-transform

Fonts - size and spacing



Fonts - colors

color: green;

font-family
font-size
line-height
font-weight
letter-spacing
color
text-decoration
text-align
font-style
text-transform

Fonts - decoration

text-decoration: underline;

font-family
font-size
line-height
font-weight
letter-spacing
color
text-decoration
text-align
font-style
text-transform

Fonts - alignment

this is a
paragraph
centered with text-align

text-align: center;

font-family
font-size
line-height
font-weight
letter-spacing
color
text-decoration
text-align
font-style
text-transform

Fonts - weight

Helvetica Neue 25 Ultra Light
Helvetica Neue 35 Thin
Helvetica Neue 45 Light
Helvetica Neue 55 Roman
Helvetica Neue 65 Medium
Helvetica Neue 75 Bold
Helvetica Neue 85 Heavy
Helvetica Neue 95 Black

Fonts - Google fonts

Make your shopping on Google fonts (http://www.google.com/fonts).

For main typo

- **Open-Sans** (don't beat around the bush)

For headers

- **Raleway**
- **Montserrat**
- **Varela Round**
- **Museo**

Fonts - Fontawesome

Fontawesome (http://fortawesome.github.io/Font-Awesome/) is a font of icons, really useful!

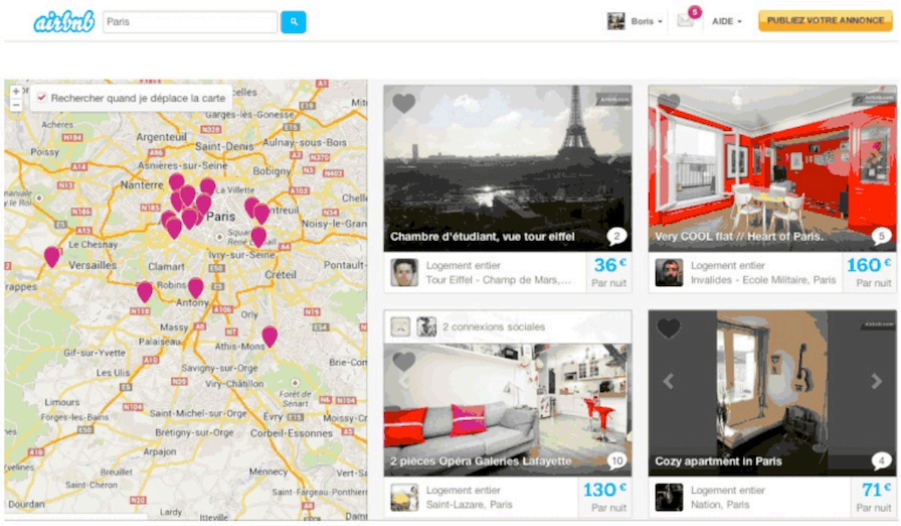
```
<!-- cdn link to paste in your <head> -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.6.1/css/font-awesome.min.css">
```

Live-code

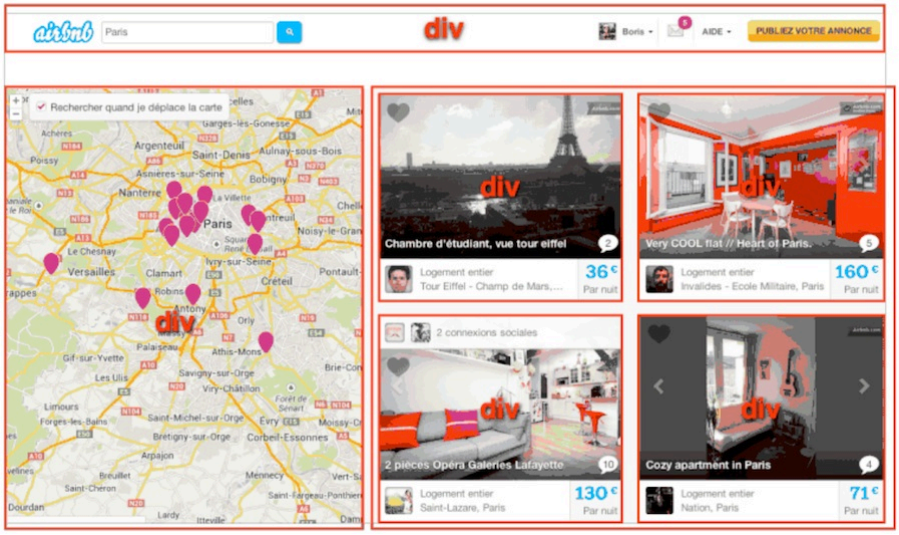
Let's **fix font and color design** like in this page (http://lewagon.github.io/html-css-challenges/02-fonts-colors/) using Google fonts.

Div and box model

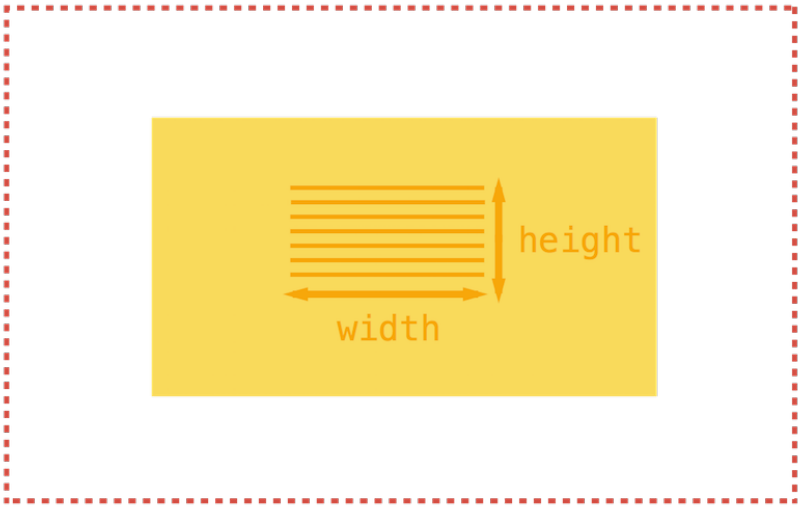
Real life...



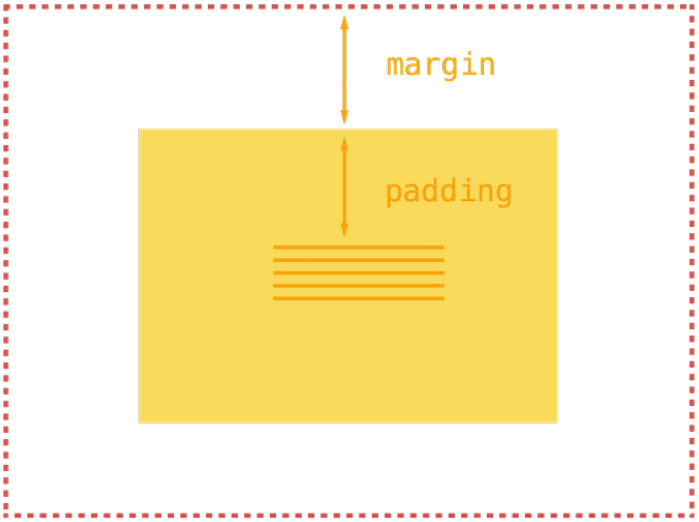
... is made of `<div>`



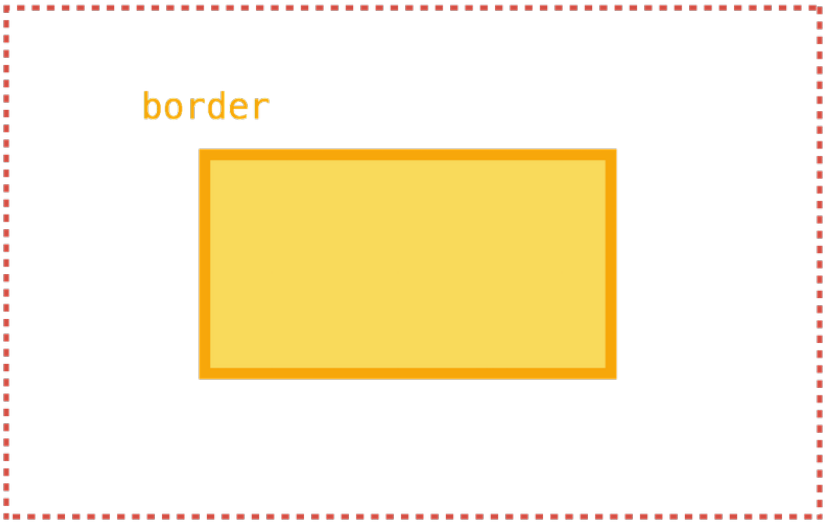
Box model - content



Box model - padding & margin



Box model - border



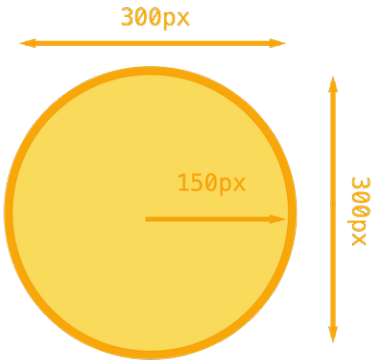
Borders

```
div {
  border-top: 1px solid red;
  border-right: 2px dotted black;
  border-bottom: 1px dashed green;
  border-left: 2px dotted black;
}
```

Border radius


```
border-radius: 5px;
```

Border radius




```
width: 300px;
height: 300px;
border-radius: 150px;
```

Box shadow

```
box-shadow: 2px 2px 3px blue;
```

Units

```
/* Absolute */
p {
  width: 50px;
}

/* Relative to parent */
p {
  width: 50%;
}

/* Relative to font size */
p {
  width: 2em;
}
```

Div design tips

```
background: white;                          /* White background */
padding: 30px;                              /* Internal space */
border-radius: 4px;                         /* Small radius */
box-shadow: 1px 1px 3px rgba(0, 0, 0, 0.1); /* Subtle shadow */
border: 1px solid rgba(0, 0, 0, 0.1);       /* OR subtle border */
```

Result:

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eos voluptatibus, quis iure vel aliquam veritatis architecto fugia necessitatibus? Quidem error explicabo nemo maiores voluptatem odio delectus ad, esse reprehenderit animi.

Div centering technique

```
width: 300px;       /* Set the width */
margin: 0 auto;     /* Set automatic margins on right/left */
```

Result:

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eos voluptatibus, quis iure vel aliquam veritatis architecto fugiat necessitatibus? Quidem error explicabo nemo maiores voluptatem odio delectus ad, esse reprehenderit animi.

Live-code

Let's **add divs** in our HTML and play with the box model with Chrome Dev tool.

---

id and class

How do you style only the logo?

HTML

```
<img src="logo.png">

<ul>
  <li><img src="paul.png"></li>
  <li><img src="ringo.png"></li>
  <li><img src="george.png"></li>
  <li><img src="john.png"></li>
</ul>
```

CSS

```
img {
  width: 40px;
}
```

Name your tag with `id`

HTML

```
<img src="logo.png" id="logo">

<ul>
  <li><img src="paul.png"></li>
  <li><img src="ringo.png"></li>
  <li><img src="george.png"></li>
  <li><img src="john.png"></li>
</ul>
```

CSS

```
#logo {
  width: 40px;
}
```

How do you style your staff pictures?

HTML

```html
<img src="logo.png" id="logo">

<ul>
   <li><img src="paul.png"></li>
   <li><img src="ringo.png"></li>
   <li><img src="john.png"></li>
   <li><img src="george.png"></li>
</ul>
```

Name your tags with `class`

HTML

```html
<img src="logo.png" id="logo">

<ul>
   <li><img src="paul.png" class="img-circle"></li>
   <li><img src="ringo.png" class="img-circle"></li>
   <li><img src="john.png" class="img-circle"></li>
   <li><img src="george.png" class="img-circle"></li>
</ul>
```

CSS

```css
.img-circle {
  border-radius: 50%;
}
```

id or class?

unique

HTML

```html
<img src="logo.png" id="logo">

<ul>
   <li><img src="paul.png" class="img-circle"></li>
   <li><img src="ringo.png" class="img-circle"></li>
   <li><img src="john.png" class="img-circle"></li>
   <li><img src="george.png" class="img-circle"></li>
</ul>
```


re-usable

Combine (1)

```html
<img src="paul.png" class="img-circle">
```

```css
.img-circle {
  border-radius: 50%;
}
```

Combine (2)

```html
<img src="paul.png" class="img-circle shadowed">
```

```css
.img-circle {
  border-radius: 50%;
}

.shadowed {
  box-shadow: 5px 5px 3px black;
}
```

Combine (3)

```html
<img src="paul.png" class="img-circle shadowed" id="leader">
```

```css
.img-circle {
  border-radius: 10px;
}

.shadowed {
  box-shadow: 5px 5px 3px black;
}

#leader {
  border: 5px solid red;
}
```

Class naming - quizz

Which one is **more explicit** (tells what it does)?

- `.btn-red` or `.btn-signup` ?
- `.background-blue` or `.background-home` ?
- `.img-user` or `.img-circle` ?

Change your mindset => **think graphical**

Class naming - convention

.component-shape

```
/* Examples*/
.text-center
.text-justify
.btn-red
.btn-green
.btn-big
.list-inline
.form-horizontal
.img-rounded
.img-circle
```

Selectors Summary

Element Selector

```
<!-- index.html -->
[...]
<body>
  <h1>Hello World</h1>
</body>
```

combined with `css /* style.css */ h1 { color: red; font-weight: bold; }` makes the `h1` elements red and bold.

Class Selector

```
<!-- index.html -->
[...]
<body>
  <p>This paragraph is not justified</p>
  <p class="text-justify">This one is</p>
  <p class="text-justify">This one also</p>
</body>
```

combined with

```
/* style.css */
.text-justify {
  text-align: justify;
}
```

will make only the second and third paragraphs justified.

Id Selector

```
<!-- index.html -->
<body>
  <div id="banner">
    <h1>Le Wagon</h1>
    <p>We bring tech skills to creative people</p>
  </div>
</body>
```

combined with

```
/* style.css */
#banner {
  background-image: url("example.jpg");
  background-size: cover;
}
```

will put an image background on the **unique** div with `id="banner"` .

Descendant Selectors

```
<!-- index.html -->
<body>
  <div id="banner">
    <h1>Le Wagon</h1>
    <p>We bring tech skills to creative people</p>
  </div>
</body>
```

combined with

```
/* style.css */
#banner h1 {
  color: white;
}
```

`h1` **children** of the element `id="banner"` will be white.

Direct Children

```
<!-- index.html -->
<body>
  <ul id="navigation">
    <li><a href="#">Home</a></li>
    <li><a href="#">Team</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</body>
```

combined with

```
/* style.css */
#navigation > li > a {
  color: blue;
}
```

`a` **direct children** of `li` **direct children** of `id="navigation"` will be blue.

Grouping

```
/* style.css */
h1, h2, h3 {
  font-weight: bold;
}
```

is a shortcut syntax for

```
/* style.css */
h1 {
  font-weight: bold;
}
h2 {
  font-weight: bold;
}
[...]
```

Pseudo Classes

```
/* style.css */
a {
  color: red;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}
```

will make links underlined when the mouse hovers over them.

See other pseudo classes (https://developer.mozilla.org/en/docs/Web/CSS/Pseudo-classes)

Quizz #1

```
<!-- index.html -->
<body>
  <p class="text-red">
    Lorem ipsum dolor sit amet, consectetur adipisicing elit.
  </p>
</body>
```

combined with

```
/* style.css */
p {
  color: black;
}
.text-red {
  color: red;
}
```

Quizz #2

```
<!-- index.html -->
<body>
  <p id="bio" class="text-red">
    Lorem ipsum dolor sit amet, consectetur adipisicing elit.
  </p>
</body>
```

combined with

```
/* style.css */
.text-red {
  color: red;
}
#bio {
  color: green;
}
```

Specificity of Selectors

```
p {                /* least specific */
  color: black;
}
.text-red {        /*        ↓       */
  color: red;
}
#bio {             /* most specific  */
  color: green;
}
```

Check out the specificity calculator (http://specificity.keegan.st/)

Live-code

Let's **finish our live-code** and get this final result (http://lewagon.github.io/html-css-challenges/04-advanced-selectors/)

Tonight, we will put our profile **online** using Github Pages (https://pages.github.com/).

Let's build your profile!

CSS components design

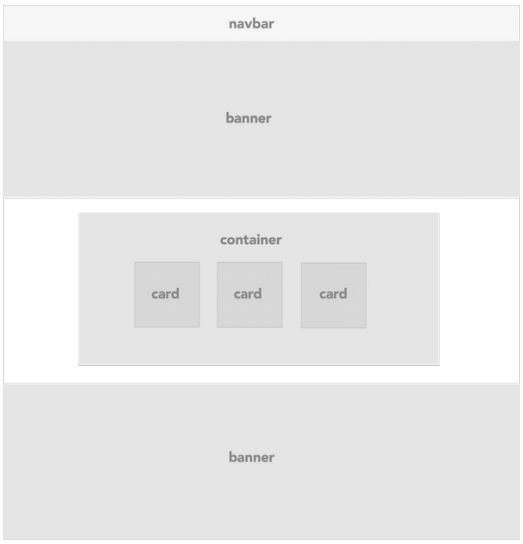
Objectives of today

- Improve your components culture
- Code your own CSS library
- Build pages using components

- **First goal**: code this CSS components library (http://lewagon.github.io/ui-components)
- **Second goal**: build pages like Airbnb Home (http://lewagon.github.io/html-css-challenges/10-homepage-with-cards/) or Product Hunt dashboard (http://lewagon.github.io/html-css-challenges/12-profile-with-products-bis/) using components.

Components = lego bricks

Used to build pages

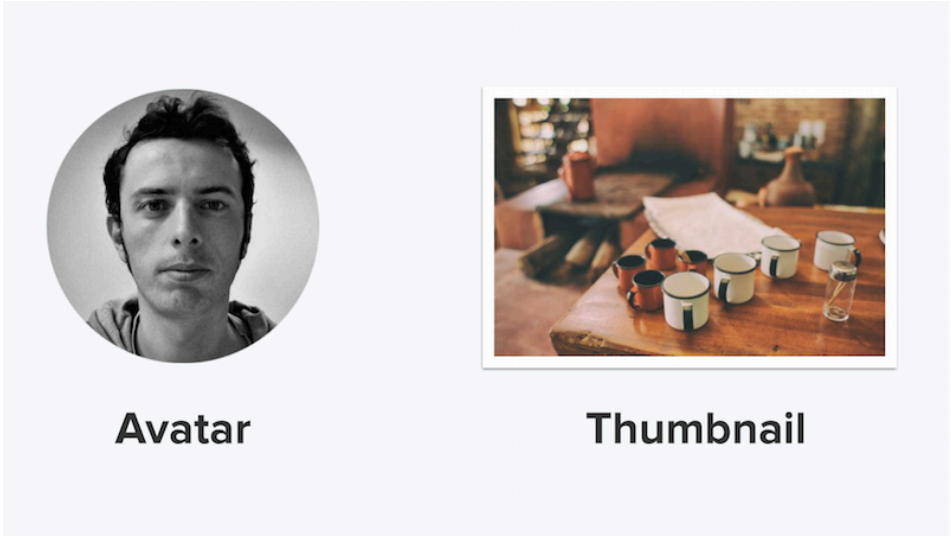


10 components => 90% of your app design

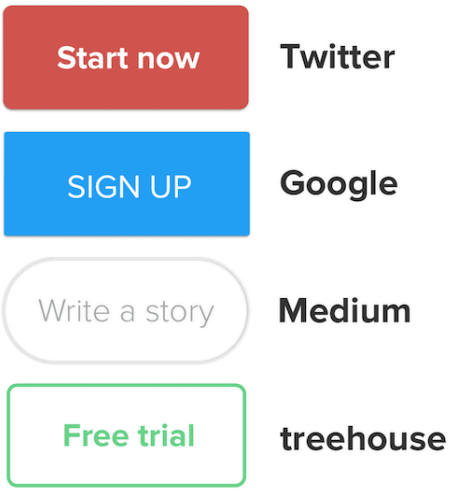
That's the secret of frontend developers

Components culture

Images

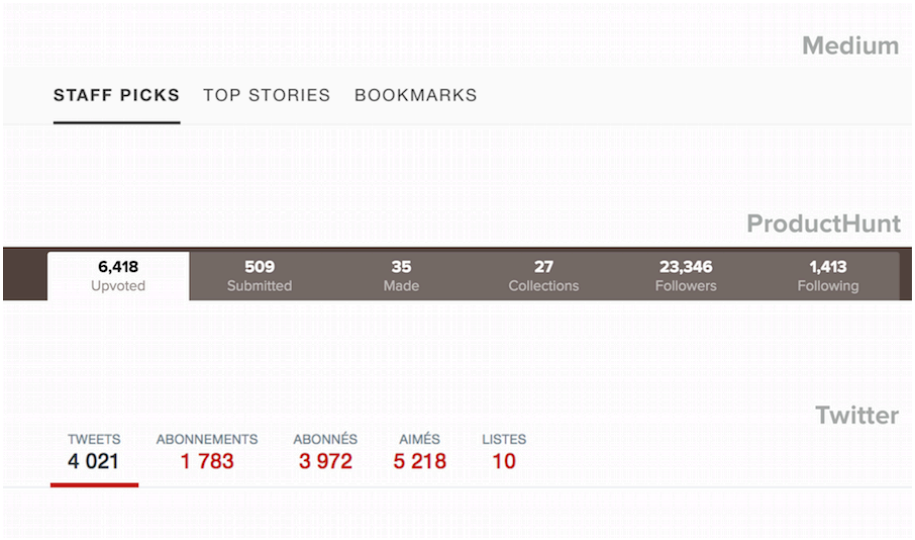


Buttons

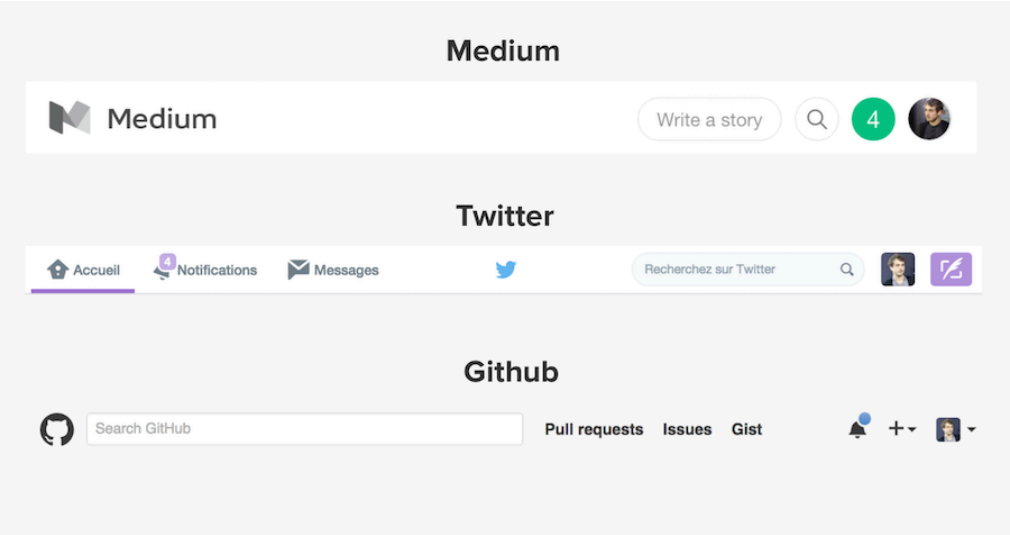


Forms (Vertical or Inline)

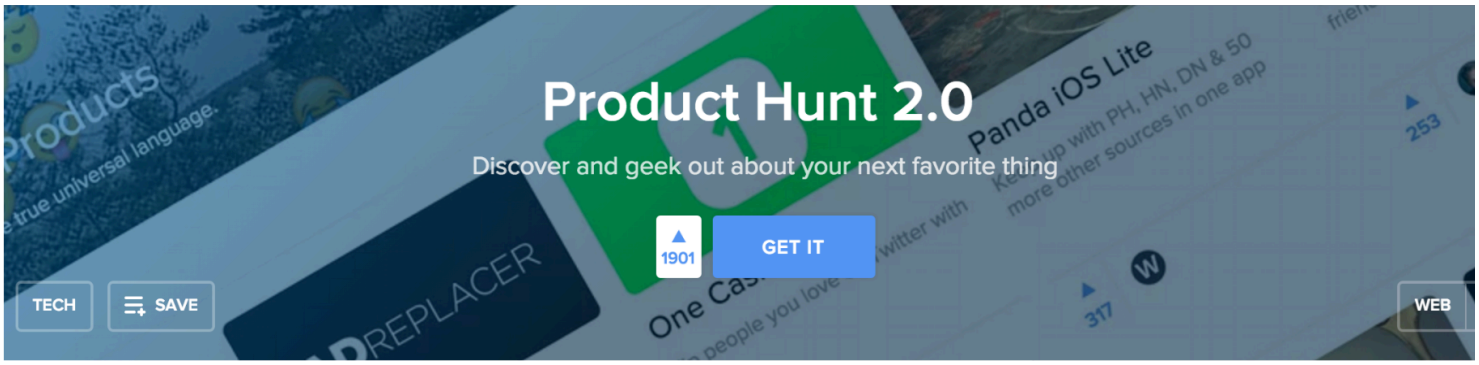
Tabs / Nav



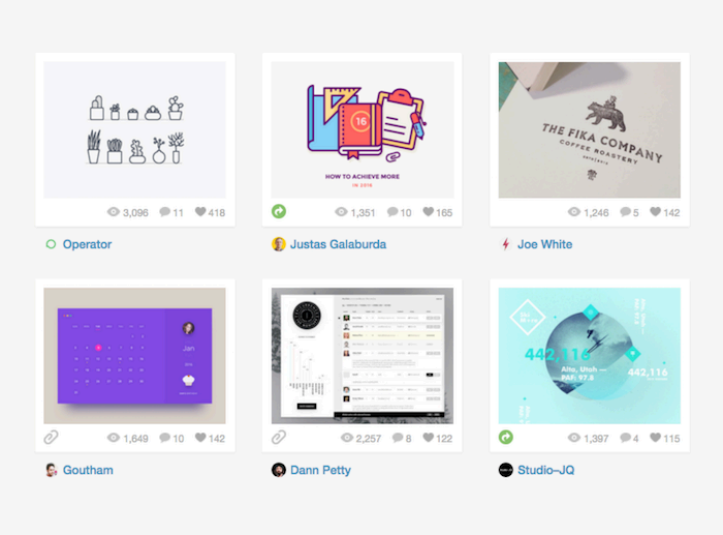
Navbar



Banner

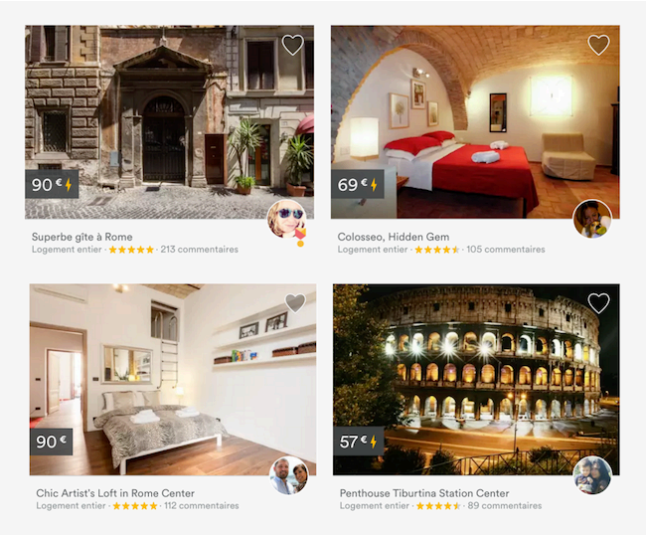


Card



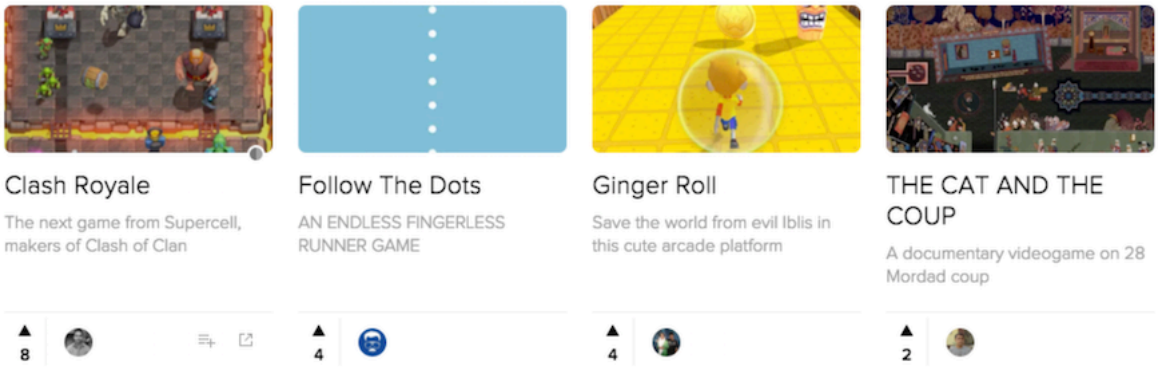
Dribbble

Card



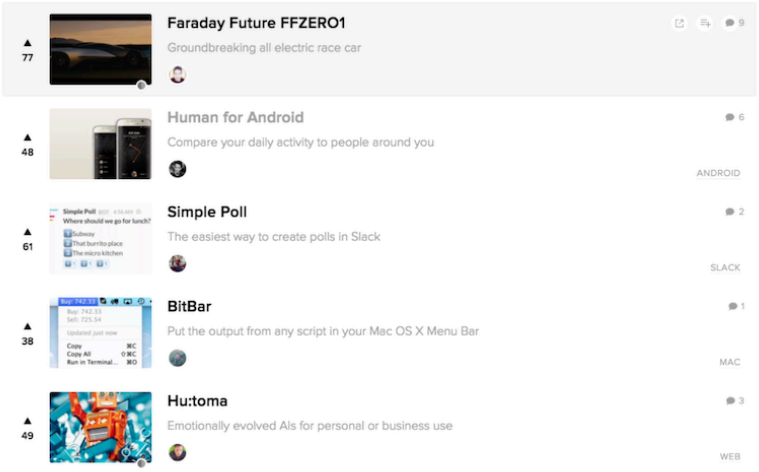
Airbnb

Card



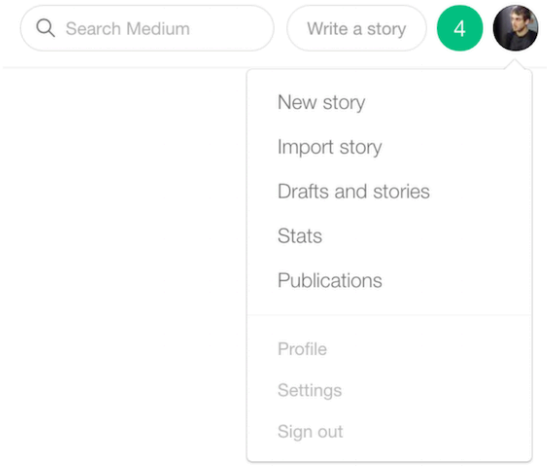
Product Hunt

List



Product Hunt

Dropdown



Medium

Components - CSS organization

Organize CSS by components

- Yesterday: one file `style.css`
- Today: one file **per component**

```
.
├── css
│   ├── components
│   │   ├── avatar.css
│   │   ├── button.css
│   │   └── card.css
│   └── style.css
└── index.html
```

Style.css imports all other CSS files

```
/* style.css */
/* --- Import Google Fonts stylesheets --- */
@import url("http://fonts.googleapis.com/
css?family=Open+Sans:400,300,700|Montserrat:400,700");

/* --- Import Components stylesheets --- */
@import url("components/avatar.css");
@import url("components/button.css");

/* --- Fonts --- */
body {
  font-family: "Open Sans", "Helvetica", "sans-serif";
}
h1, h2, h3 {
  font-family: "Montserrat", "sans-serif";
}
```

Then **one CSS link only** in the HTML page

```
<link rel="stylesheet" href="css/style.css">
```

Basic component

Let's **live-code** avatar (http://lewagon.github.io/ui-components/#avatar) and button (http://lewagon.github.io/ui-components/#button) components

Start now

Avatar Button

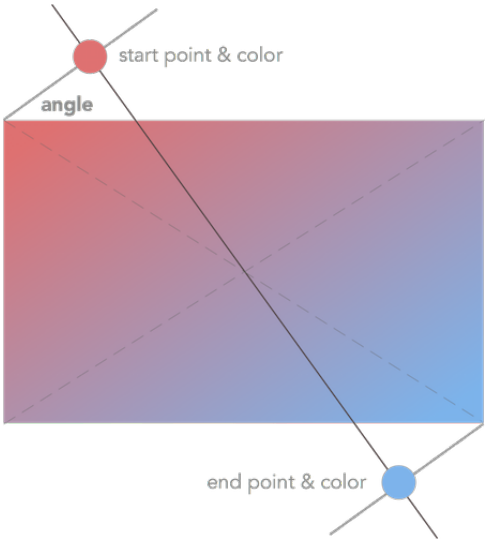
Advanced CSS patterns

For advanced components, we need advanced techniques

1 - Gradient filter

- Background pictures are too contrasted or of poor quality.
- They need a filter.

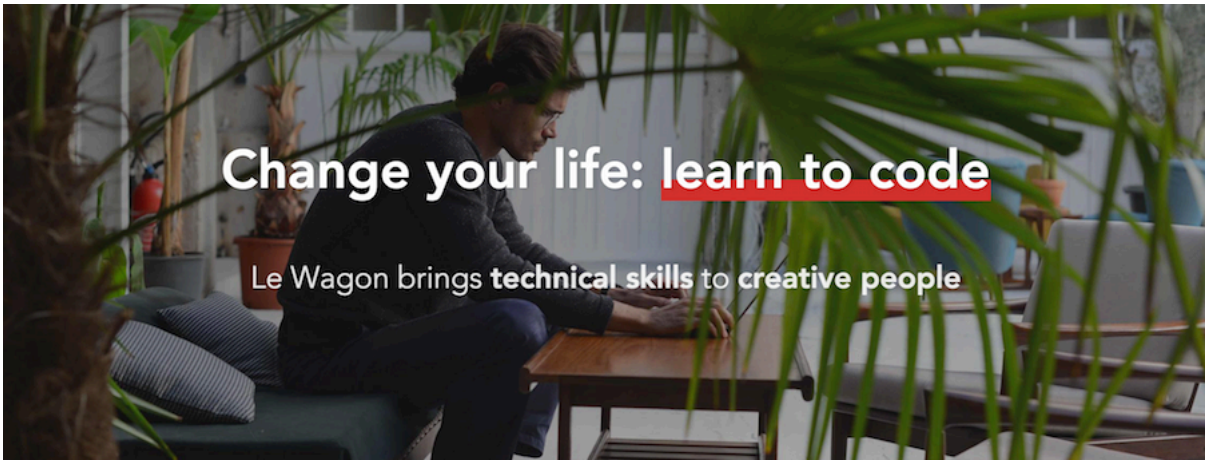
Linear gradient - theory



Forget about theory, **just play with parameters in the browser!**

Linear gradient - code

```
.banner {
  background-image: linear-gradient(135deg, rgba(0,0,0,0.8) 0%, rgba(0,0,0,0.2) 50%),
                    url('background.jpg');
  background-size: cover;
}
```



User-uploaded pictures

```
.banner {
  background-size: cover;
}
```

Put background image and gradient **in your HTML**

```
<div class="banner" style="background-image: linear-gradient(135deg, rgba(0,0,0,0.8) 0%, rgba(0,0,0,0.2) 50%), url('uploaded_by_user.jpg');"
  ...
</div>
```

2 - Relative > Absolute pattern

When you need to position elements precisely in a div

Position relative

- Offset from **initial** position
- A way to **pin an element**

Let's offset the logo on https://www.lewagon.com/

Position absolute

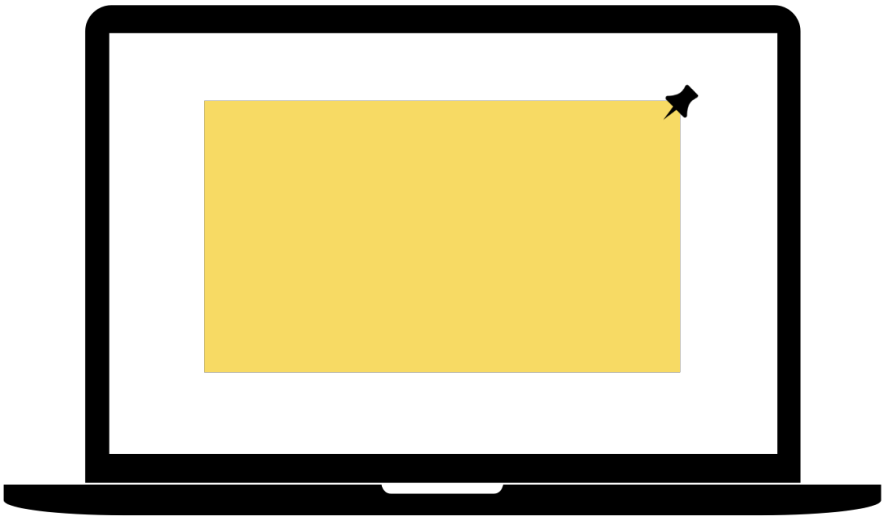
- Offset from **1st positioned parent**
- If no pinned parent => **body**

Relative > Absolute

The **ultimate CSS pattern**.

Relative > Absolute

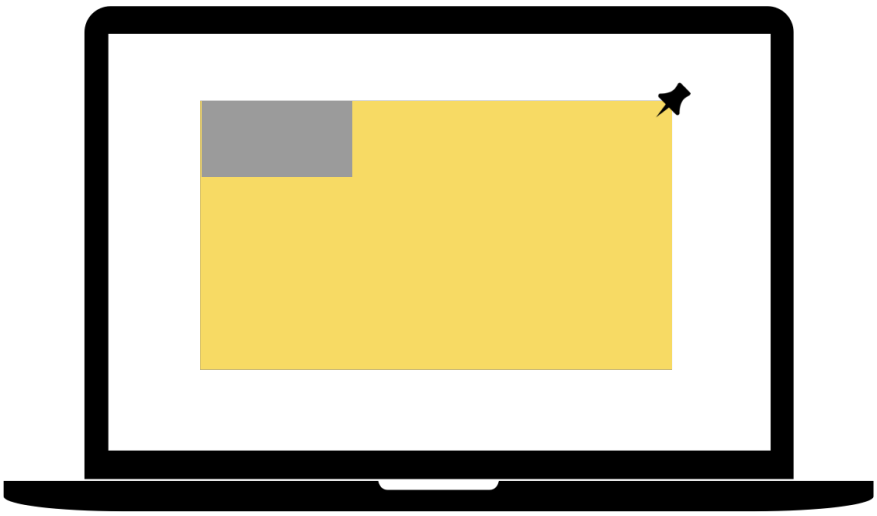
1) **Pin** a div with `position: relative`



```
#relative {
  position: relativ
}
```

Relative > Absolute

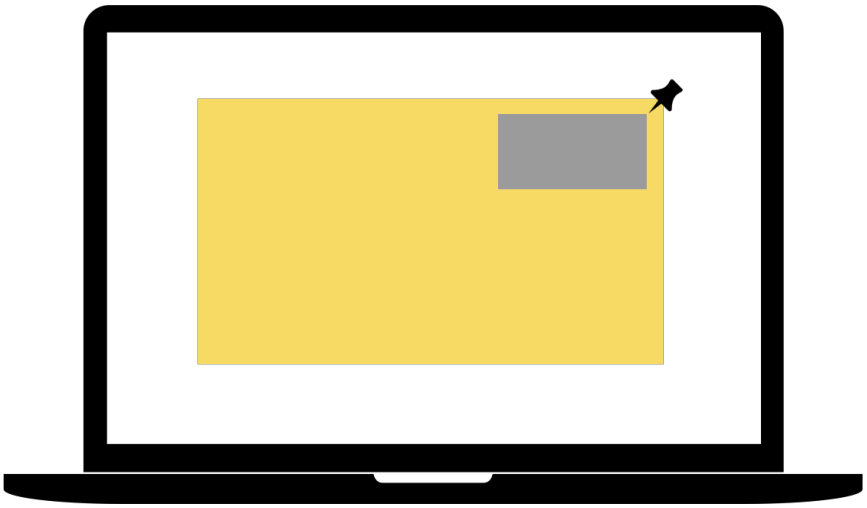
2) Put an inside div with `position: absolute`



```
#relative {
  position: relativ
}

#absolute {
  position: absolut
}
```

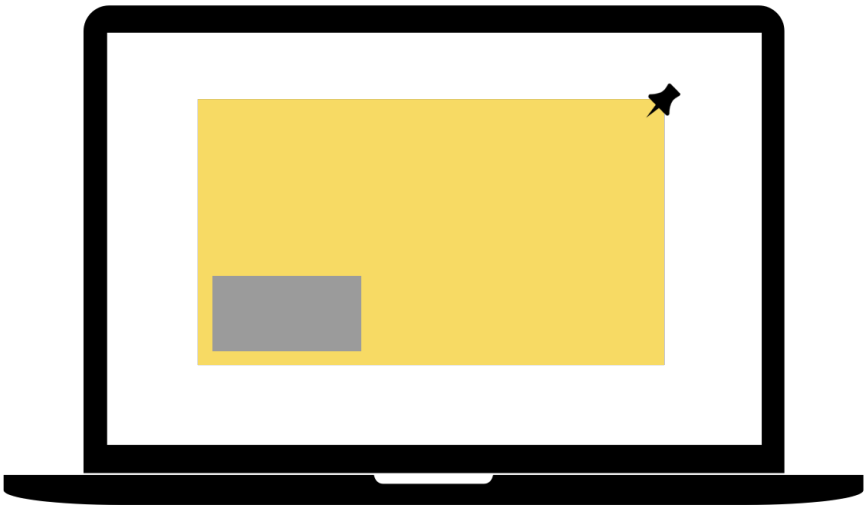
Relative > Absolute

3) Now you can play with inside div



```
#relative {
  position: relativ
}

#absolute {
  position: absolut
  top: 10px;
  right: 10px;
}
```

Relative > Absolute



```
#relative {
  position: relativ
}

#absolute {
  position: absolut
  bottom: 10px;
  left: 10px;
}
```

Card design

Let's **live-code** a card (http://lewagon.github.io/ui-components/#card) component using:

1. A linear gradient filter
2. The **Relative > absolute** pattern

Flexbox

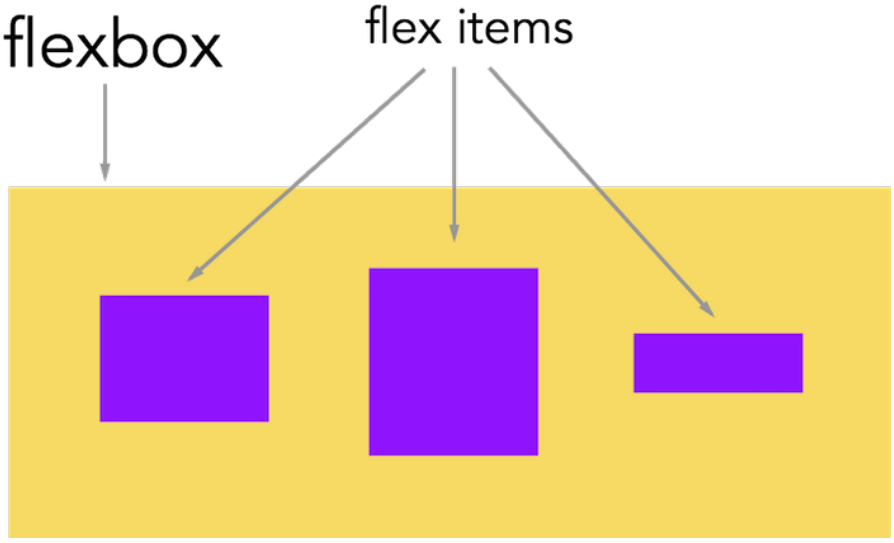
Awesome for:

- Vertically align items
- Build small component grids (ex: for tabs)

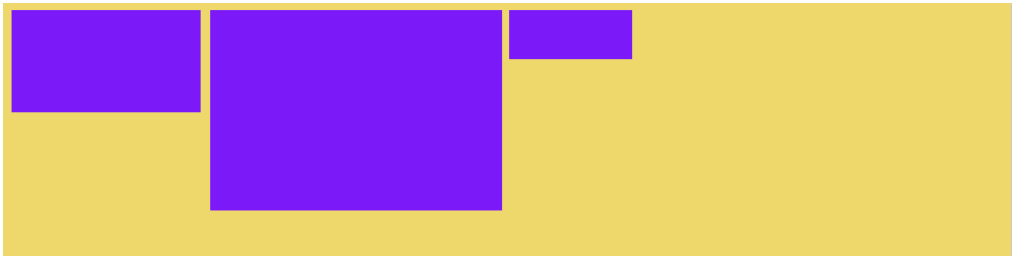
Not adapted for:

- Positioning in corners: use the Relative > Absolute pattern
- For the main responsive grid, use Bootstrap

Flexbox - vocabulary

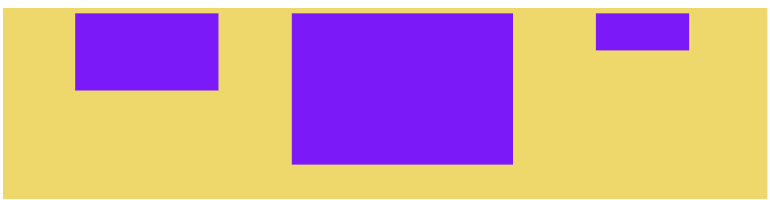


Flexbox - basics



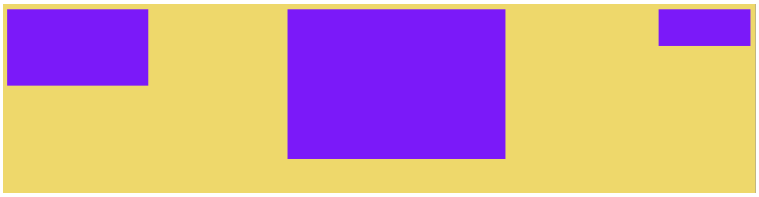
```
.flexbox {
  display: fle
}
```

Flexbox - space around



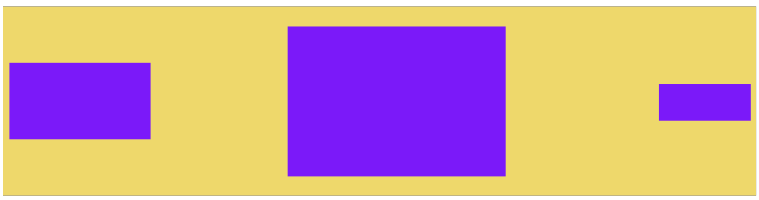
```
.flexbox {
  display: flex;
  justify-content: space-arour
}
```

Flexbox - space between



```
.flexbox {
  display: flex;
  justify-content: space-betwee
}
```

Flexbox - align items



```
.flexbox {
  display: flex;
  justify-content: space-betwee
  align-items: center;
}
```

Flexbox - item behavior

```
.flex-item {
  flex: X X 200px;
}
```

flex-shrink: can I shrink?

flex-grow: can I grow?

Flexbox - grow example

```
.flexbox {
  display: flex;
  align-items: center;
}
.flex-item {
  flex: 0 0 20%;
}
.flex-body {
  flex: 1 0 auto;
}
```

Flex quiz - badge?


```
<div class="badge">
  3
</div>
```

```css
.badge {
  display: flex;
  justify-content: space-around;
  align-items: center;
}
```

Flex quiz - banner?


```html
<div class="banner">
  <div class="banner-content">...</div>
</div>
```

```css
.banner {
  display: flex;
  justify-content: space-around;
  align-items: center;
}
```

Flex quiz - cards?


```html
<div class="card">
  <img src="...">
  <div>
    <h2>Sebastien Saunier</h2>
    <p>CTO, Le Wagon</p>
  </div>
</div>
```

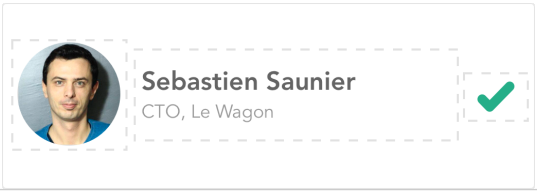
```css
.card {
  display: flex;
  justify-content: space-around;
  align-items: center;
}
```

Flex quiz - cards?


```html
<div class="card">
  <img src="...">
  <div>...</div>
</div>
```

```css
.card {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```

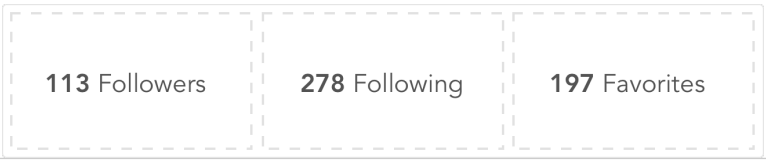
Flex quiz - cards?




```
<div class="card">
  <img src="...">
  <div class="card-body">...</div>
  <i class="fa fa-check"></i>
</div>
```

```
.card {
  display: flex;
  align-items: center;
}
.card-body {
  flex-grow: 1;
}
```

Flex quiz - tabs?

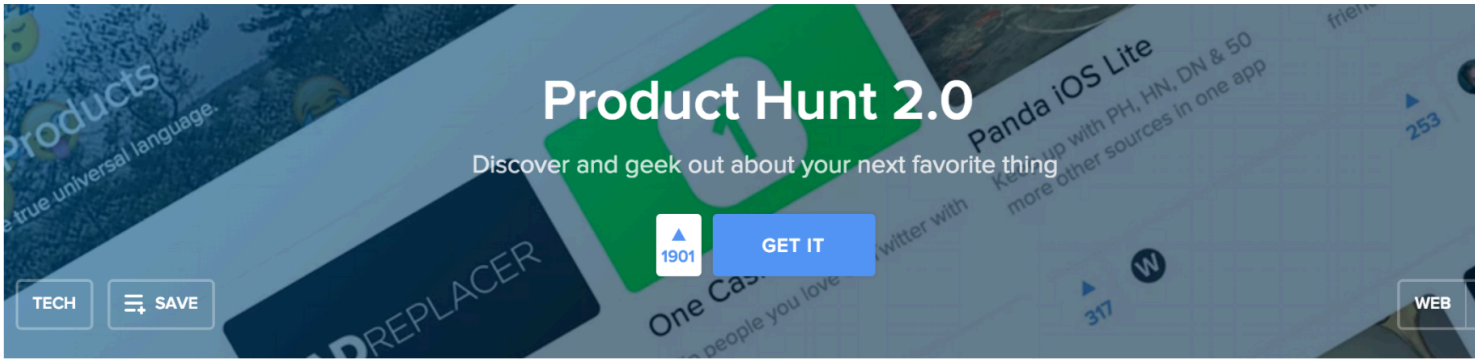



```
<div class="tabs">
  <a href="..." class="tab"><strong>113</strong> Followers</a>
  <a href="..." class="tab"><strong>278</strong> Following</a>
  <a href="..." class="tab"><strong>197</strong> Favorites</a>
</div>
```

```
.tabs {
  display: flex;
}
.tab {
  flex: 1 1 auto;
}
```

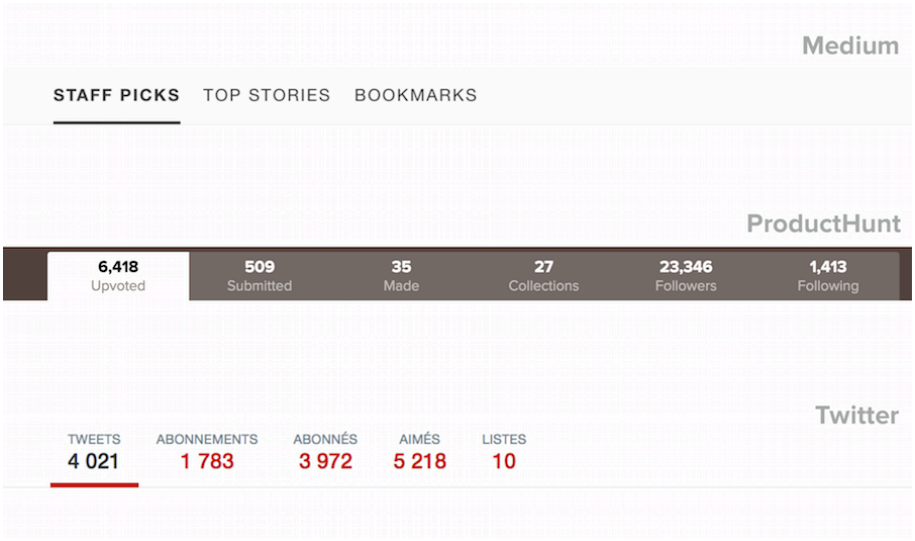
Flex live-code - Banner

Let's **live-code** a banner (http://lewagon.github.io/ui-components/#banner) (centering content on horizontal & vertical axis)



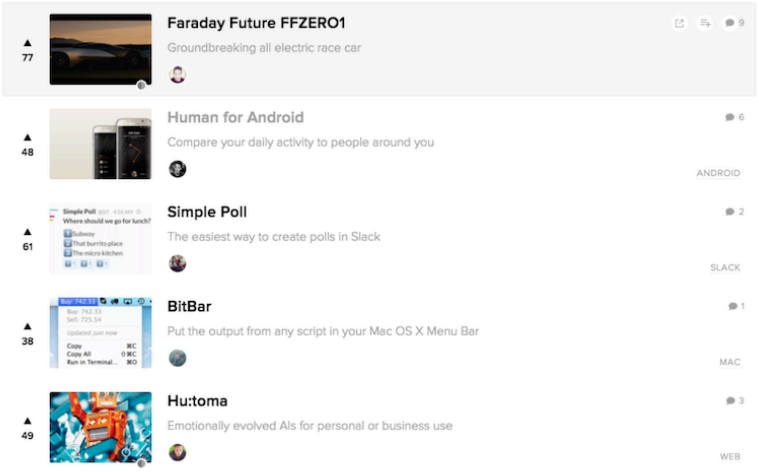
Flexbox live-code - Tabs

Let's **live-code** a tabs (http://lewagon.github.io/ui-components/#tabs) building a simple grid in flexbox



Flexbox live-code - List

Let's **live-code** list items (http://lewagon.github.io/ui-components/#list) setting a `flex-grow` on the central description.



Flexbox - resources

- Flexbox with gifs (https://medium.freecodecamp.com/an-animated-guide-to-flexbox-d280cf6afc35#.kbhv9jwdb)
- Flexbox froggy (http://flexboxfroggy.com/)

Building pages with components

What's a layout?

- Components = **elementary bricks**
- Layout = **global structure**

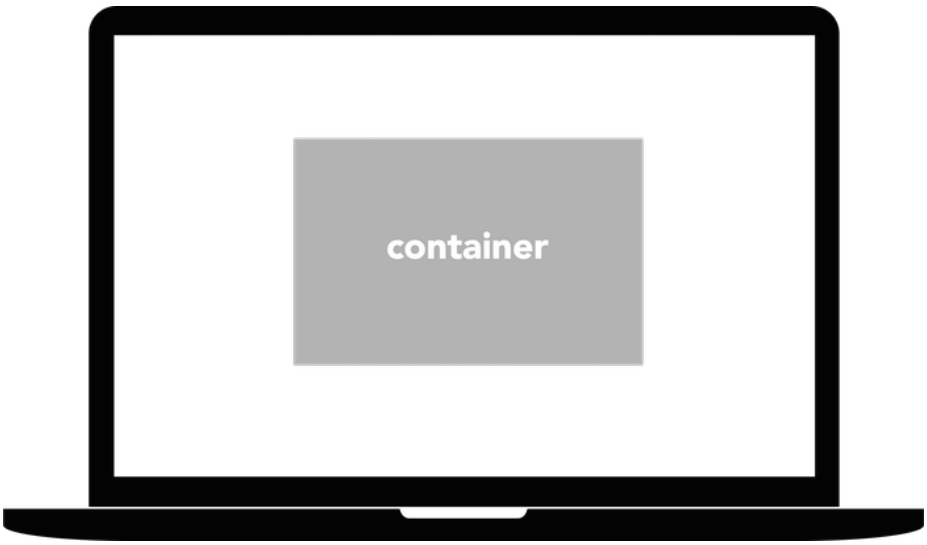
Two fundamental layout components

Container & Wrapper

Container

Full-width content is ugly. You should put page content in a container.

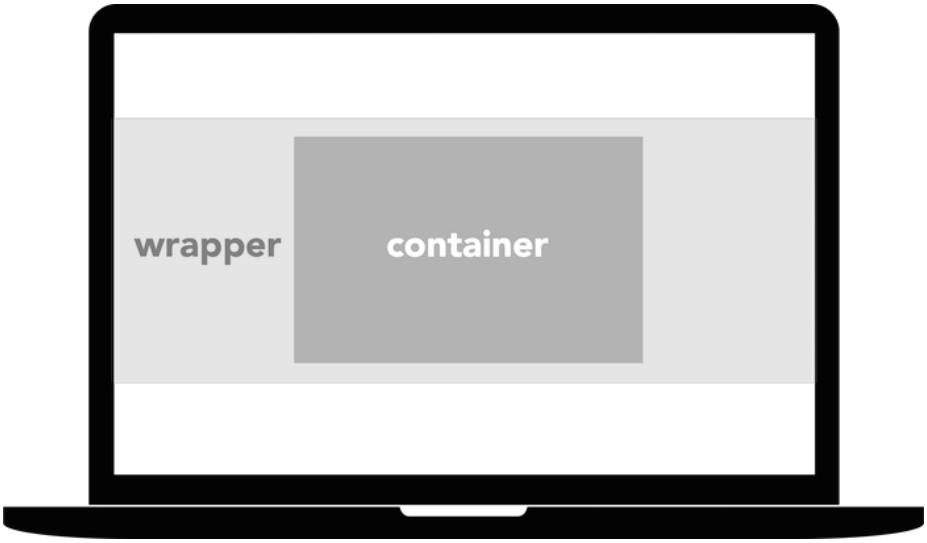
```
.container {
  margin: 0 auto;
  width: 970px;
}
```



Wrapper

You often need a background color (full-width) around content.

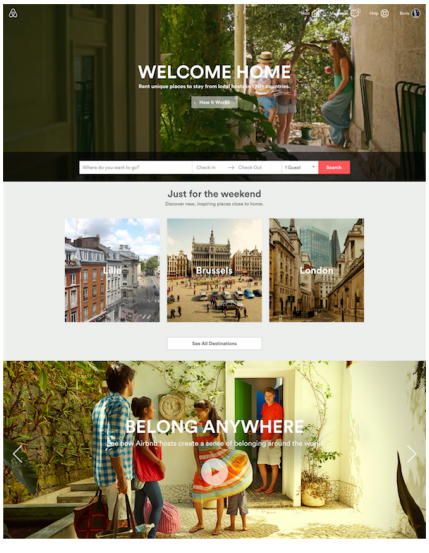
```
.wrapper {
  background-color: lightgrey;
}
```



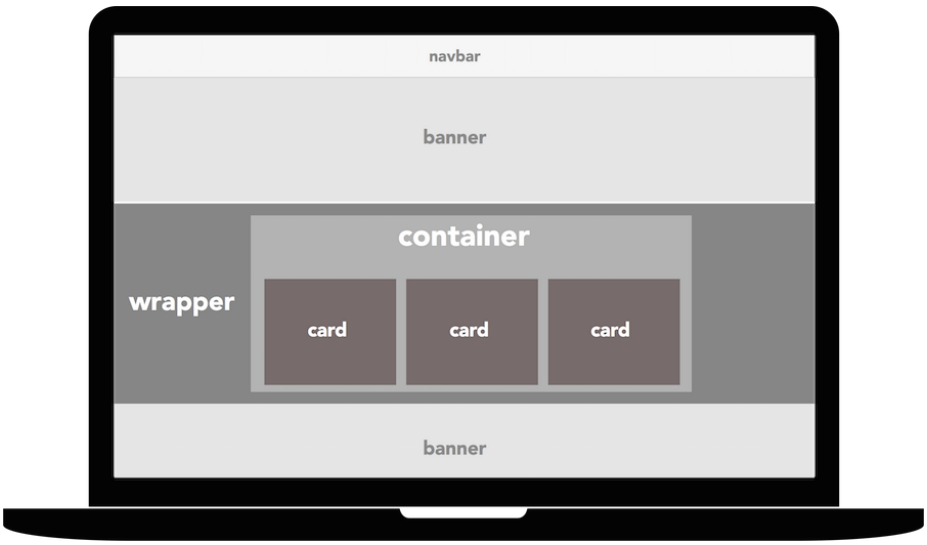
Layout - use cases

Can you decompose each page into simple components?

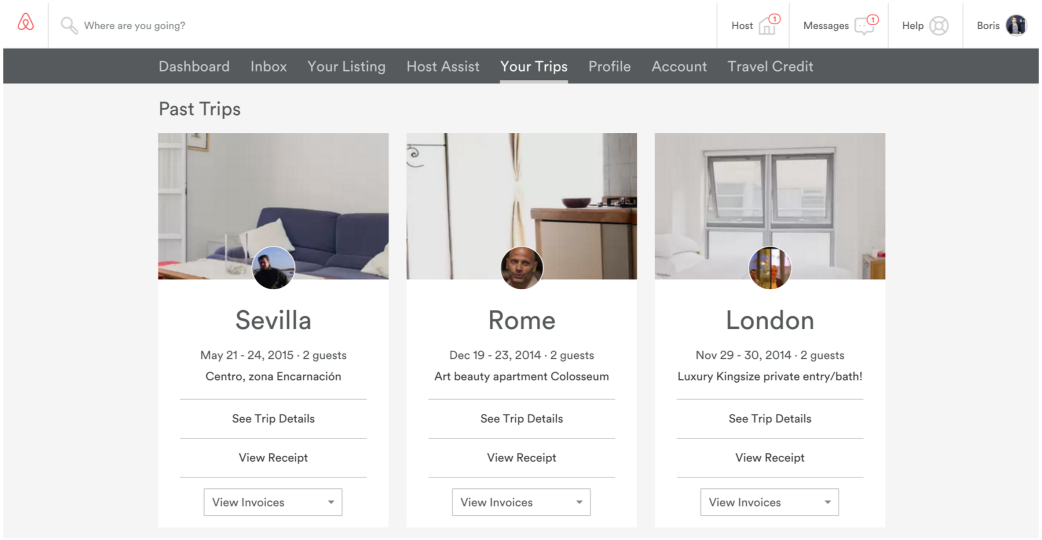
Airbnb home?



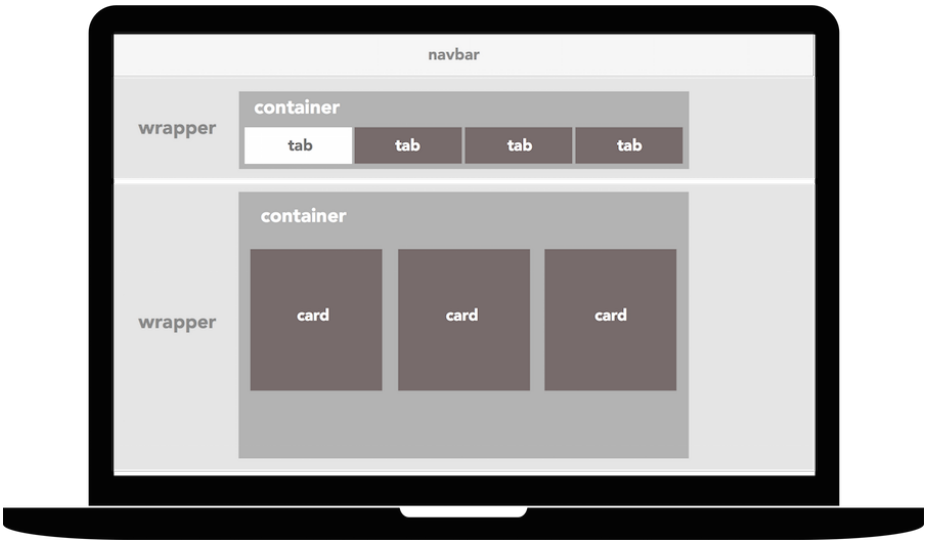
Solution



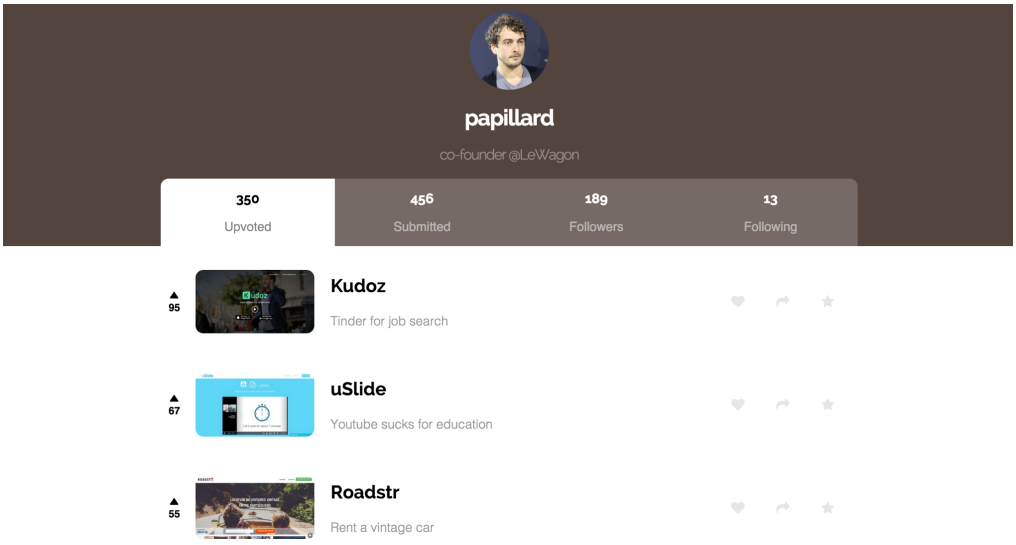
Airbnb dashboard?



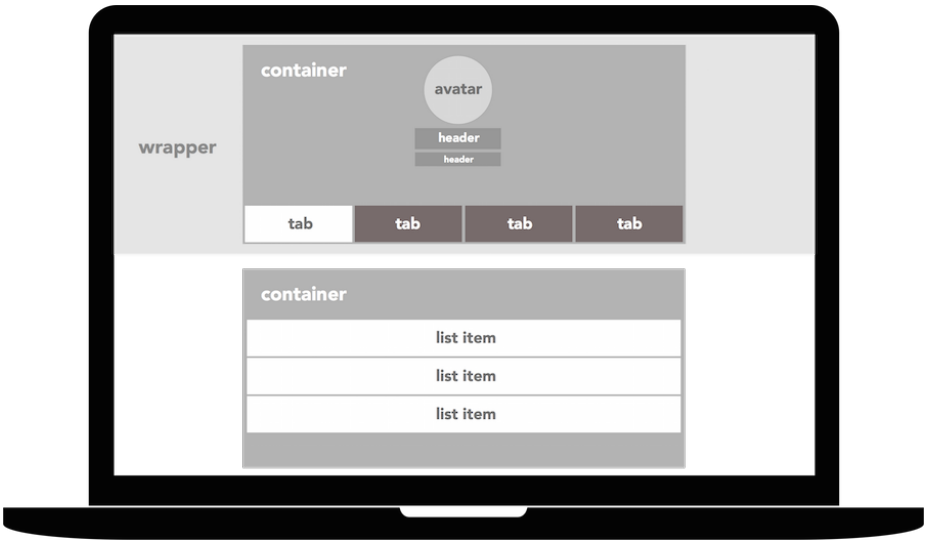
Solution



Product Hunt dashboard?



Solution



More Advanced layout

Wait for Bootstrap grid tomorrow.

Let's build your CSS components library!

Design resources & tips

Graphical tool - Sketch

bohemiancoding.com/sketch (http://bohemiancoding.com/sketch/)

Free web-based alternative: Figma (https://www.figma.com/)

UI do's and don'ts

https://goodui.org/ (https://goodui.org/)

Pictures

- Flickr (https://www.flickr.com/search/?q=quai%20de%20seine): exhaustive
- Pexels (https://www.pexels.com/): selective
- Unsplash.it (http://unsplash.it/), Bunny Holder (https://www.bunnyholder.com/): placeholders
- Medium - stock photos that don't suck (https://medium.com/@dustin/stock-photos-that-dont-suck-62ae4bcbe01b)

Colors

- Coolors (http://coolors.co/)
- Color Hunt (http://colorhunt.co/)

Background gradient

UI gradients (http://uigradients.com/#)

Fonts - Google fonts

Make your shopping on Google fonts (http://www.google.com/fonts). Our advice:

For main typo

- **Open-Sans**

For headers

- **Raleway**
- **Montserrat**
- **Varela Round**
- **Museo**

Fonts - Hack on Github

Look for `.ttf/.eot/.woff/etc..` files on Github.

- **Avenir**
- **Proxima Nova**
- **Gotham Rounded**
- **Brandon**

Icons

- Fontawesome (http://fortawesome.github.io/Font-Awesome/): font of icons
- Material Design Icons (http://materializecss.com/icons.html): fonts of icons from Google
- TheNounProject (http://thenounproject.com/): historical icon resource
- Iconstore.co (https://iconstore.co/): best icon resource
- Nucleo (https://nucleoapp.com/premium-colored-icons/): cool free pack

---

HTML/CSS snippets

- Codepen (http://codepen.io/)

Inspiration

- https://dribbble.com/ => for everything
- http://www.awwwards.com/ => for websites
- http://www.calltoidea.com/ => for UI components
- https://uimovement.com/ => for dynamic UI components

Of course - The best source

http://alumni.lewagon.org

Bootstrap

Why Bootstrap?

- For rich components, we have our own CSS library (http://lewagon.github.io/ui-components/)
- For standard ones, let's not re-invent the wheel!
- We also need a responsive grid system

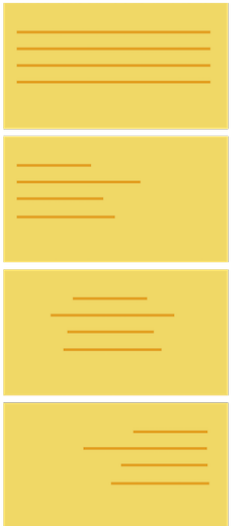
Start with Bootstrap boilerplate

https://github.com/lewagon/bootstrap-boilerplate

Basic components

Basic components - semantic scheme

Basic components - alignment



```
<div class='text-justify'></div>
<div class='text-left'></div>
<div class='text-center'></div>
<div class='text-right'></div>
```

Basic components - list inline

```
<ul class='list-inline'>
  <li>
    <a href='http://twitter.com/lewagonparis'>
      <i class='fa fa-twitter'></i>
    </a>
  </li>
  <li>
    <a href='http://facebook.com/lewagonformation'>
      <i class='fa fa-facebook'></i>
    </a>
  </li>
  <!-- etc.. -->
</ul>
```

Basic components - images (http://getbootstrap.com/css/#images)


```
<img class='img-rounded' src='ex.png'>
```


```
<img class='img-circle' src='ex.png'>
```


```
<img class='img-thumbnail' src='ex.png'>
```

Basic components - buttons (http://getbootstrap.com/css/#buttons)

Go
```
<a class='btn btn-primary' href='#'>Go</a>
```

Go
```
<a class='btn btn-warning' href='#'>Go</a>
```

Go
```
<a class='btn btn-danger' href='#'>Go</a>
```

Go
```
<a class='btn btn-info' href='#'>Go</a>
```

Go
```
<a class='btn btn-success' href='#'>Go</a>
```

Basic components - forms (http://getbootstrap.com/css/#forms)

alan@turing.org

*****

Apply

```
<form action='/'>
  <input type='email' class='form-control' placeholder='alan@turing.org
  <input type='password' class='form-control' placeholder='*****'>
  <input type='submit' value='Apply' class='btn btn-primary'>
</form>
```

Basic components - forms (http://getbootstrap.com/css/#forms) with label

Your email

alan@turing.org

Your password

*****

Apply

```
<form action='/'>
  <div class='form-group'>
    <label for='email'>Your email</label>
    <input id='email' type='email' class='form-control' placeholder='alan@turing.or
  </div>
  <div class='form-group'>
    <label for='password'>Your password</label>
    <input id='password' type='password' class='form-control' placeholder='*****'>
  </div>
  <input type='submit' value='Apply' class='btn btn-primary'>
</form>
```

Basic components - forms (http://getbootstrap.com/css/#forms) inline

alan@turing.org ***** Apply

```
<form action='/' class='form-inline'>
  <input type='email' class='form-control' placeholder='alan@turing.or
  <input type='password' class='form-control' placeholder='*****'>
  <input type='submit' value='Apply' class='btn btn-primary'>
</form>
```

Documentation

http://getbootstrap.com/css/

Advanced Components

Advanced Components - labels (http://getbootstrap.com/components/#labels)

Challenge opt.

```
<h2>
  Challenge <span class='label label-warning'>opt.</spa
</h2>
```

Challenge todo

```
<h2>
  Challenge <span class='label label-success'>todo</spa
</h2>
```

Advanced Components - nav (http://getbootstrap.com/components/#nav)

Ugly!!!! let's use Le Wagon tabs (http://lewagon.github.io/ui-components/#tabs) instead

Advanced Components - alerts (http://getbootstrap.com/components/#alerts)

You password **is not secure enough** x

```
<div class='alert alert-danger alert-dismissible' role="alert">
  <button type="button" class="close" data-dismiss="alert" aria-label="Close">
    <span aria-hidden="true">&times;</span>
  </button>
  You password <strong>is not secure enough</strong>
</div>
```

You are **successfully** logged in x

```
<div class='alert alert-success alert-dismissible' role="alert">
  <button type="button" class="close" data-dismiss="alert" aria-label="Close">
    <span aria-hidden="true">&times;</span>
  </button>
  You are <strong>successfully</strong> logged in
</div>
```

Advanced Components - navbar (http://getbootstrap.com/components/#navbar

Ugly!!!! let's use Le Wagon navbar (http://lewagon.github.io/ui-components/#navbar) instead

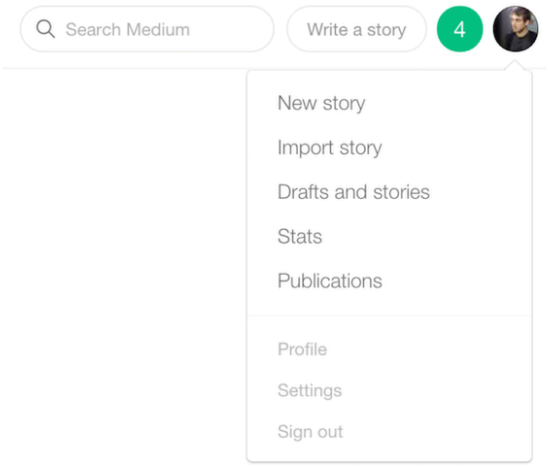
---

Dynamic components

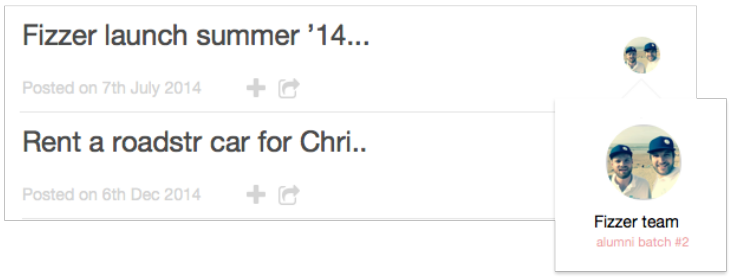
Dynamic components - modals (http://getbootstrap.com/javascript/#modals)


```
<!-- Button to trigger modal -->
<button data-toggle="modal" data-target="#sign-up">Join</butt

<!-- Modal -->
<div class="modal fade" id="sign-up">
  <!-- Modal content with Sign up form ->
</div>
```

Dynamic components - dropdown (http://getbootstrap.com/javascript/#dropdowns)



Dynamic components - popovers (http://getbootstrap.com/javascript/#popovers




```
<img src="fizzer.png"
     class="popover-img"
     data-content="<img src='fizzer.png'>
                   <h2>Fizzer Team</h2>
                   <p>alumni batch #2</p>

<script>
  $(function () {
    $(".popover-img").popover({
      html: true,
      trigger: "hover"
    });
  });
</script>
```

Documentation

http://getbootstrap.com/javascript/

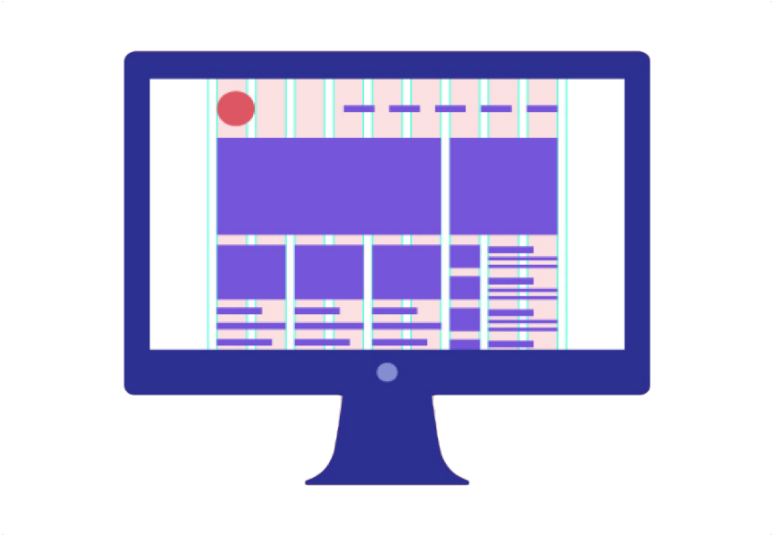
Live-code

Let's live-code a first Airbnb home (http://lewagon.github.io/bootstrap-challenges/08-Final-airbnb-home-without-grid/) in 5 minutes

- Using **a maximum of Bootstrap components** (`btn-primary` , `form-inline` , `list-inline` ..)

- Using your own CSS components **when really needed** (e.g. banner, footer)

Bootstrap grid



Definition

Bootstrap includes a **responsive, mobile first** fluid **grid** system that appropriately scales up to **12 columns** as the **device** size increases.

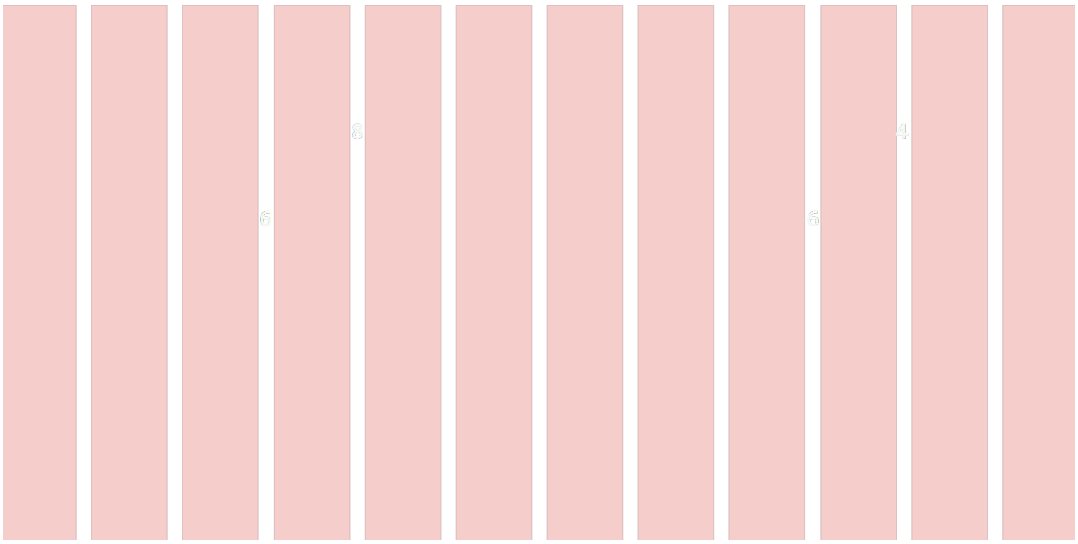
Responsive

Mobile-first

How it works?

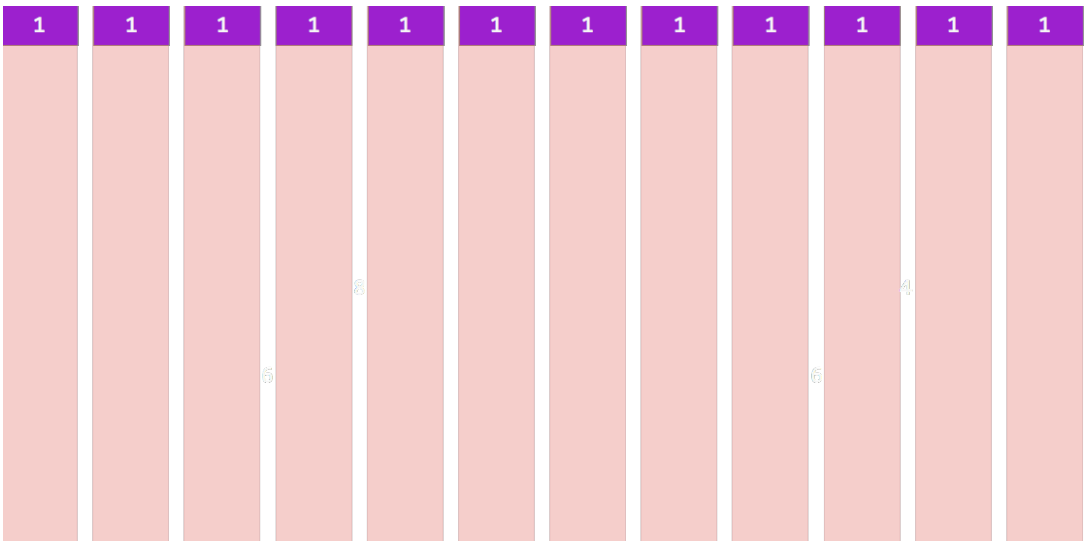
Container first

```
<div class="container">

  <!--Define a container-->

</div>
```

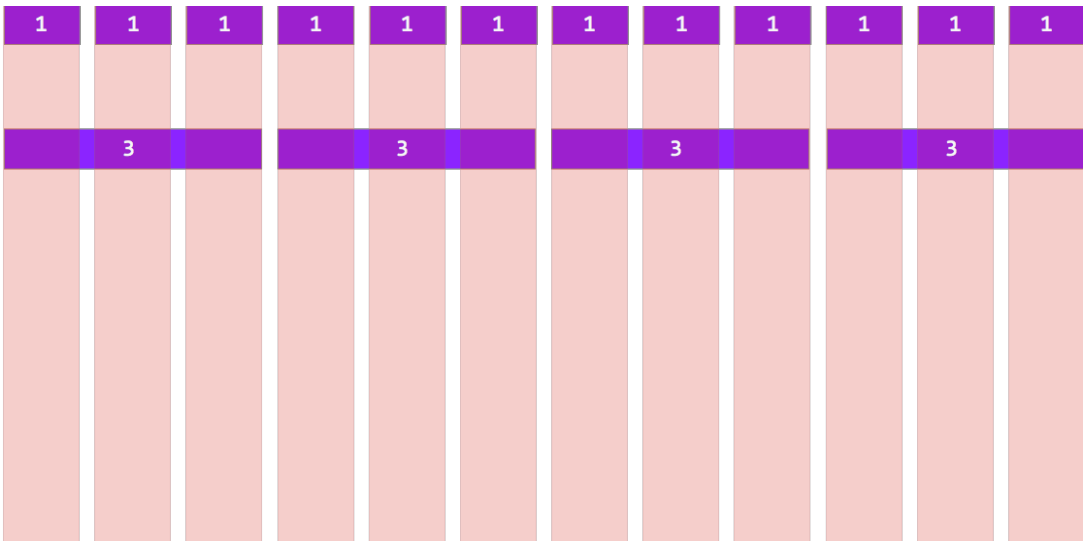


Then insert rows

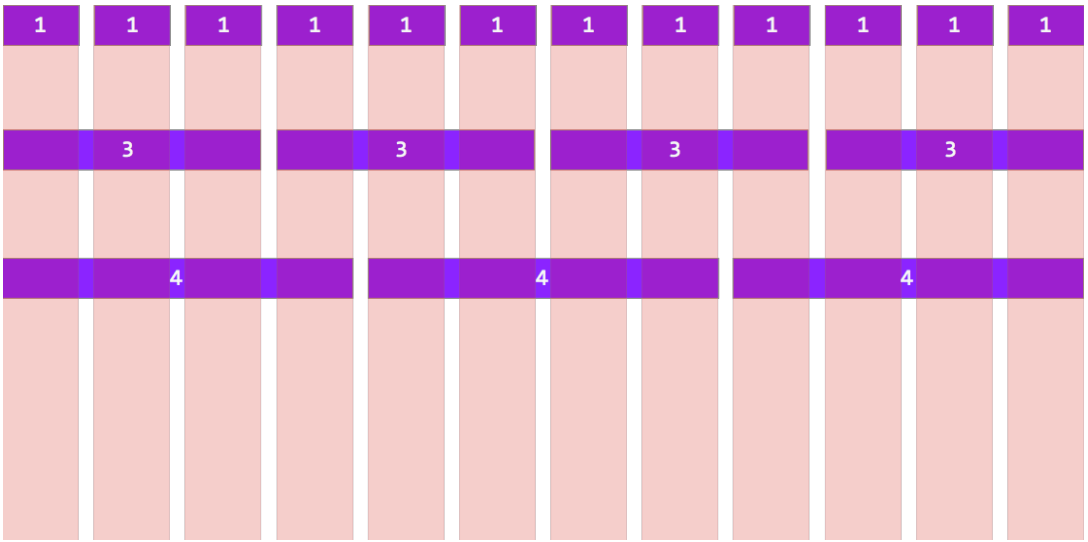
```
<div class="container">

  <div class="row">
    <!-- First row -->
  </div>

</div>
```




```
<div class="container">

  <div class="row">
    <!-- First row -->
  </div>

  <div class="row">
    <!-- Second row -->
  </div>

</div>
```




```
<div class="container">

  <div class="row">
    <!-- First row -->
  </div>

  <div class="row">
    <!-- Second row -->
  </div>

  <div class="row">
    <!-- Third row -->
  </div>

</div>
```



What's in the row?

```
<div class="col-xs-6">
</div>
```

- **col**: because it fits a number of columns
- **xs**: screen width
- **6**: number of columns taken by the block (**max 12**)

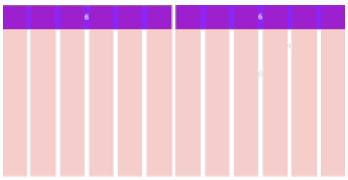
Media

- **xs**: Extra small devices (Phones < 768px)
- **sm**: Small devices (Tablets > 768px)
- **md**: Medium devices (> 992px)
- **lg**: Large devices (Desktops > 1200px)

Example (2 columns)

Code

```
<div class="container">
  <div class="row">

    <div class="col-xs-6">
    </div>

    <div class="col-xs-6">
    </div>

  </div>
</div>
```



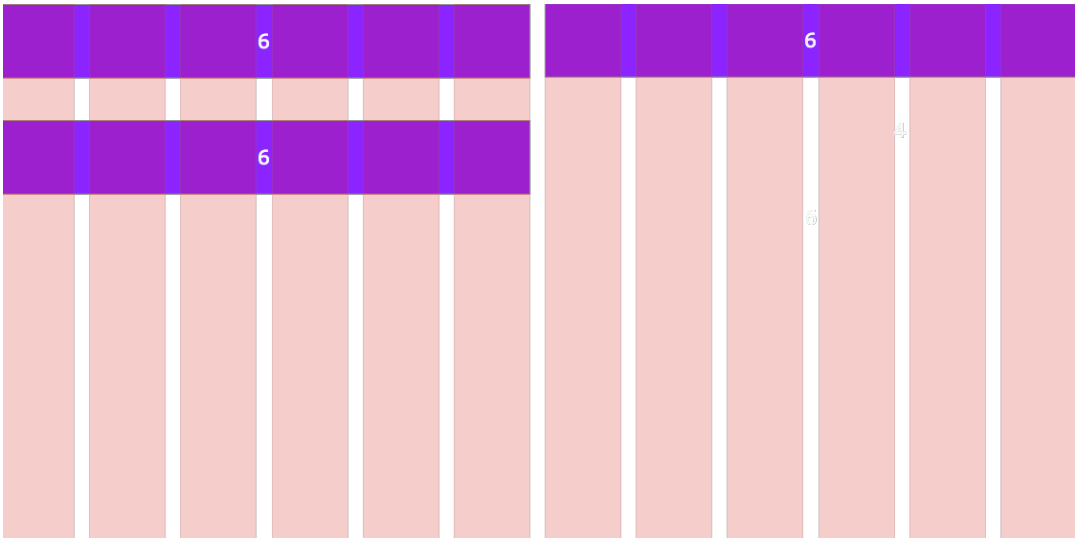
Mobile first

- if you just give the `.col-xs` classes
- it will apply on all larger screens

Now what happens?

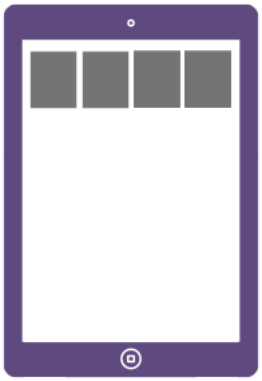
```
<div class="container">
  <div class="row">

    <div class="col-xs-6">
    </div>

    <div class="col-xs-6">
    </div>

    <div class="col-xs-6">
    </div>

  </div>
</div>
```

Just return on new line



Responsive example

We want 4 blocks for tablets or larger screens



We want 2 lines for mobiles with 2 blocks on each line

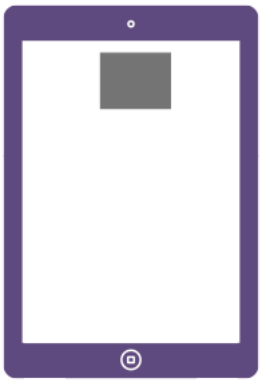
How will you do ?

Code

```
<div class="container">
  <div class="row">
    <div class="col-xs-6 col-sm-3"></div>
    <div class="col-xs-6 col-sm-3"></div>
    <div class="col-xs-6 col-sm-3"></div>
    <div class="col-xs-6 col-sm-3"></div>
  </div>
</div>
```

Yeah baby !

Offsetting blocks



Can we offset a block ?

Yes we can

```
<div class="container">
  <div class="row">
    <div class="col-sm-offset-4 col-sm-4">
      <!-- Le contenu de la div centrée -->
    </div>
  </div>
</div>
```

Useful technique for responsive form (http://lewagon.github.io/bootstrap-challenges/10-Login/)

Grid dont's

Don't break **`.container > .row > .col` succession**

Don't change CSS code of `.container`, `.row` or `.col` elements

For full-width background

Place a wrapper **around** the `.container`

```
<div class="wrapper-grey">
  <div class="container">
    <div class="row">
      <div class="col-xs-6">
      </div>
      <div class="col-xs-6">
      </div>
    </div>
  </div>
</div>
```

For inside cards

Put cards **inside** the `.col`

```
<div class="container">
  <div class="row">
    <div class="col-xs-6">
      <div class="card">
        ...
      </div>
    </div>
    <div class="col-xs-6">
      <div class="card">
        ...
      </div>
    </div>
  </div>
</div>
```

Then write your CSS on these divs.

As you don't touch grid elements, you won't break the grid.

```
.wrapper-grey {
  background: #CCCCCC;
  padding: 30px 0;
}
.card {
  border: 1px solid grey;
  padding: 30px;
  background: white;
}
```

Live-code

Let's **live-code** 2 grid examples for features and flats, like in this Airbnb home (http://lewagon.github.io/bootstrap-challenges/09-Final-airbnb-home/index.html)

Pitch Prep

What's a product pitch?

Target

Who are my customers?

Pain

What do they suffer from?

Solution

How do I answer their need?

Target, be specific!

anyone
adults
women
hipsters
employees
...

festival attendees
sport amateur players
fashionistas
long-distance couple
fast food manager
...

A pain has to hurt...

As sport amateur player,
organising a 30 players competition

As long-distance couple,
planning affordable weekend

Keep the solution simple

Social network for amateur players

Opodo copycat

Tournament generation tool

Curated destinations explorer

Cherry on the cake

Find a **new angle**, be different from competitors

Let’s pitch Le Wagon in 2 minutes

Good pitch
makes
Good product

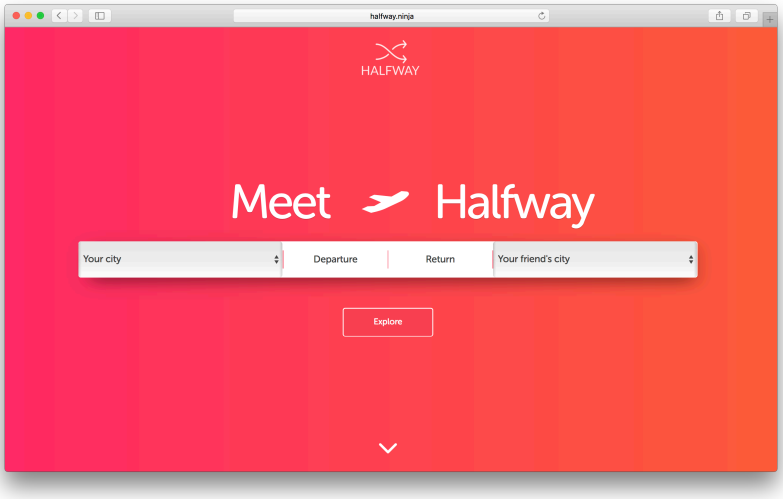
Obvious pain = clear product

Simple solution = few pages, amazing front

New Angle = more creativity, more fun

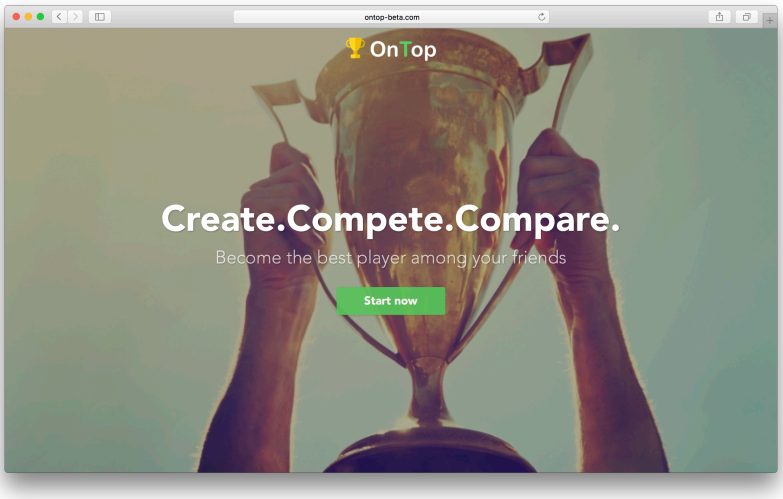
See by yourself :)

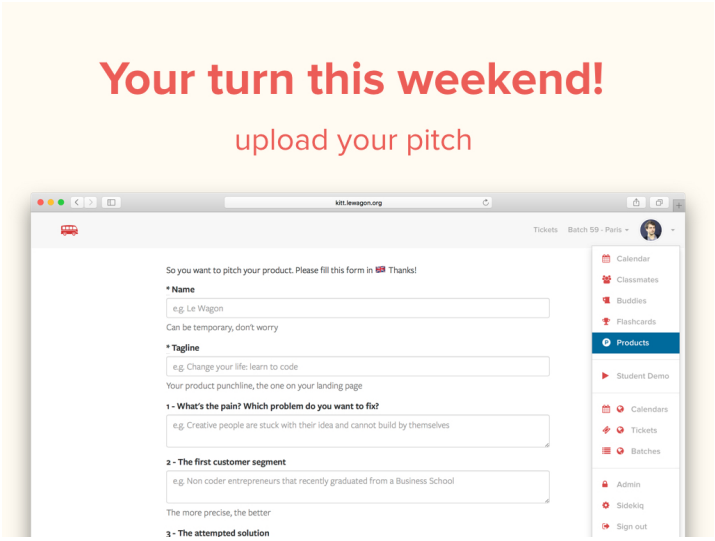
HalfWay (https://www.lewagon.com/demoday/43/halfway)



See by yourself :)

OnTop (https://www.lewagon.com/demoday/41/ontop)





Middleman

Install Middleman


```
$ gem install middleman -v 3.4.1
```

Objectives

- Build a static Airbnb (http://lewagon.github.io/middleman-airbnb/) with Middleman
- Understand ERB (layout, partial, loop) and SCSS (variables, Bootstrap over-ride)

HTML/CSS are not adapted

- Copy/paste of HTML code for navbar, footer, cards, etc..
- 1000 lines of CSS to over-ride Bootstrap.

Sadly, your browser **speaks only HTML/CSS**

ERB

ERB (**E**mbedded **R**uby) comes with

- **layout**
- **partial**
- **loops**

You will **refacto your HTML** using ruby!

SCSS

SCSS (http://sass-lang.com/guide) comes with **variables**

You will **over-ride Bootstrap** with variables.

Advanced setup

Since our browser is stupid, we need a tool:

- to compile ERB into HTML
- to compile SCSS into CSS

Middleman

Middleman (http://middlemanapp.com/) = static site generator

Setup

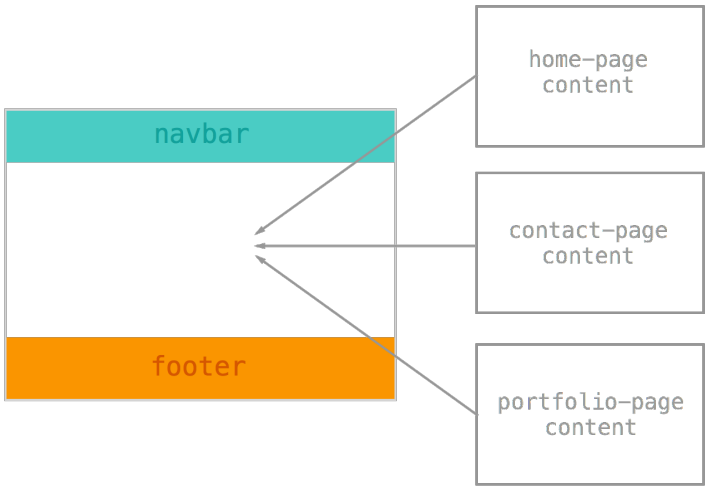
Fork our boilerplate (https://github.com/lewagon/frontend-advanced-boilerplate) & rename it in settings. Then:

```
$ cd ~/code/$GITHUB_USERNAME
$ git clone git@github.com:$GITHUB_USERNAME/REPO_NAME.git
$ cd REPO_NAME
$ bundle install
$ middleman server # To launch local server
$ middleman deploy # To build HTML/CSS and push it on Github Pages
```

ERB basics

- Middleman & Rails use ERB
- There are other templates (slim, handlebar, liquid, etc..).

ERB - layout (https://middlemanapp.com/basics/templates/)




```
<html>
  <head>
    <title>My Site</head>
  </head>
  <body>
    <nav class='navbar'>…</nav>
      <%= yield %>
    <div id='footer'>…</div>
  </body>
<html>
```

Live-code

- Let's add a `team.html.erb` page & illustrate layout.
- Let's add a footer partial to our layout.
- Let's play with ERB in our team page.

URLs are tricky

Depend on current file location.

```
<img src="images/logo.png" alt="Logo">
<img src="../images/logo.png" alt="Logo">
<img src="../../images/logo.png" alt="Logo">

<a href="team.html">Team</a>
<a href="../team.html">Team</a>
<a href="../../team.html">Team</a>
```

Are there nice methods **generating `<img>` and `<a>` for us**?

ERB - helpers (https://middlemanapp.com/basics/helpers/)

```
<%= link_to "Press", "press.html" %>
<!-- => <a href="/press.html">Press</a> -->
```

```
<%= image_tag "logo.png" %>
<!-- => <img src="/images/logo.png"> -->
```

Helpers with class

```
<%= link_to "Press", "press.html", class: "btn btn-primary" %>
<!-- => <a href="/press.html" class="btn btn-primary">Press</a> -->
```

```
<%= image_tag "logo.png", id: "logo", class: "img-circle" %>
<!-- => <img src="/images/logo.png" id="logo" class="img-circle"> -->
```

Link with rich content

When your link contains **more than text**

```
<% link_to "index.html", class: "navbar-brand" do %>
  <%= image_tag "logo.png", class: "img-circle" %>
<% end %>
```

```
<a href="index.html" class="navbar-brand">
  <img src="logo.png" alt="" class="img-circle">
</a>
```

Helper in CSS

```
.banner {
  background-image: image-url('background.jpg');
}
```

Live-code

Let's import our Airbnb Home (http://lewagon.github.io/bootstrap-challenges/11-Airbnb-search-form/) in Middleman (just the HTML).

- Put common parts (navbar/footer) in the layout
- Replace **`<a>`** and **`<img>`** with **`link_to`** and **`image_tag`**

SCSS basics

SASS vs SCSS

.sass

```
li
  color: $emerald
```

.scss

```
li {
  color: $emerald;
}
```

- **SASS** => **new** syntax (no overlap with CSS)
- **SCSS** (Sassy CSS) => **extended** syntax (accept raw CSS)
- Both are processed to regular CSS

SCSS - partials

SCSS - variables

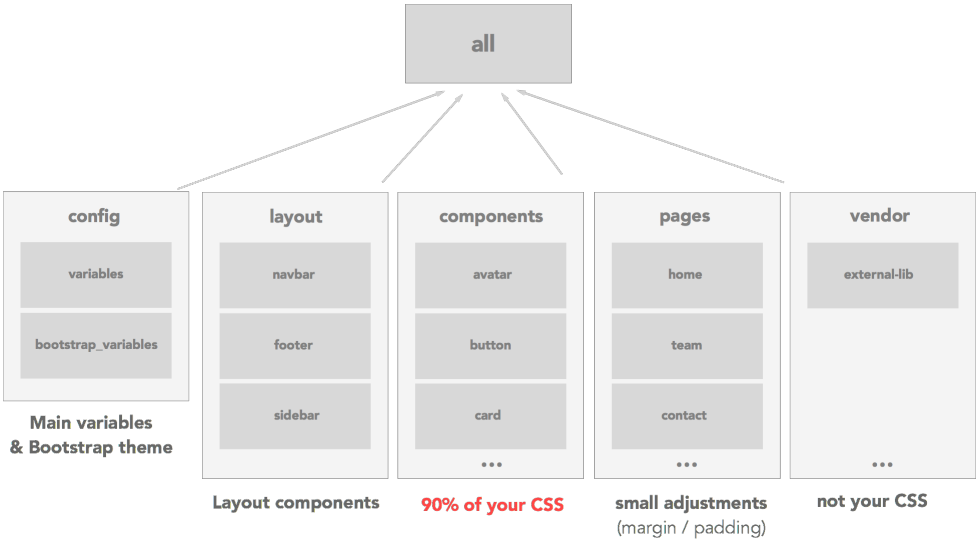
example.css

```
h1, h2, h3{
  color: #EE5F5B;
}

a:hover{
  color: #EE5F5B;
}

ul.list-inline > li > a{
  color: #EE5F5B;
}
```

example.css.scss

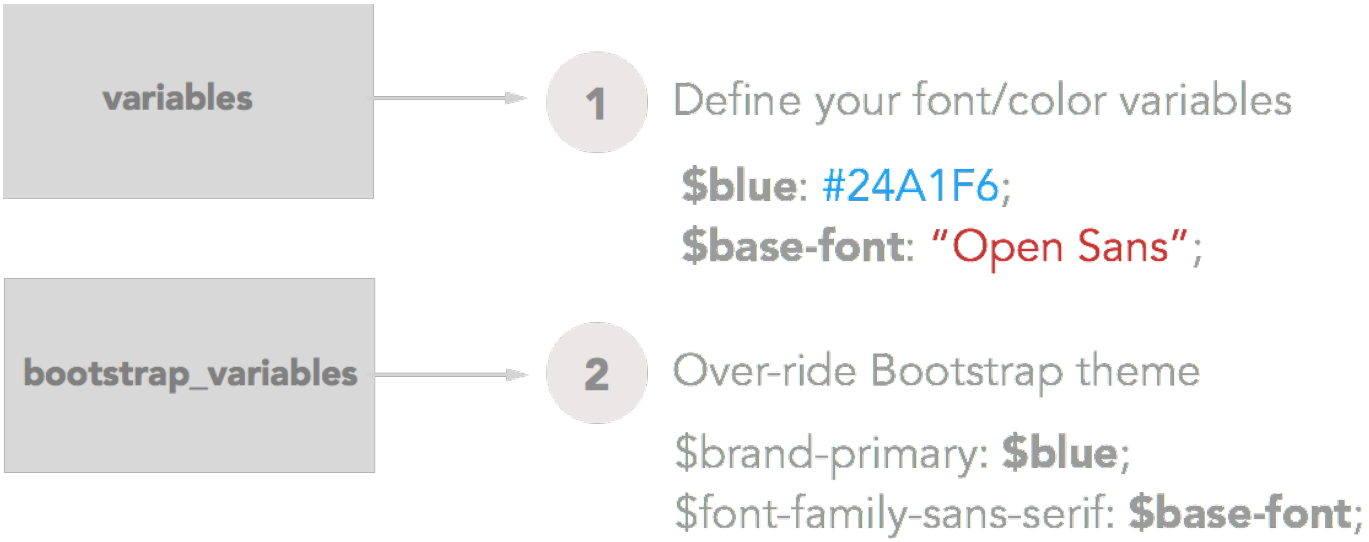
```
$red: #EE5F5B;

h1, h2, h3{
  color: $red;
}

a:hover{
  color: $red;
}

ul.list-inline > li > a{
  color: $red;
}
```

SCSS - partials organization



SCSS - Bootstrap over-ride

Here are all Bootstrap variables (http://getbootstrap.com/customize/#less-variables) you can over-ride.



SCSS - nesting

home.css

```
#banner{
  background: image-url('background.jpg');
  padding: 100px;
  text-align: center;
}

#banner h1{
  font-family: 'Montserrat', sans-serif;
  color: #EE5F5B;
}

#banner p{
  color: center;
}
```

home.css.scss

```
#banner{
  background: image-url('background.jpg');
  padding: 100px;
  text-align: center;
  h1{
    font-family: $montserrat;
    color: $red;
  }
  p{
    color: center;
  }
}
```

SCSS - chaining

example.css

```
a{
  color: $red;
}

a:hover{
  color: $yellow;
}
```

example.css.scss

```
a{
  color: $red;
  &:hover{
    color: $yellow;
  }
}
```

Live-code

Let's import our Airbnb Home CSS (http://lewagon.github.io/bootstrap-challenges/11-Airbnb-search-form/) in Middleman (just the CSS).

- Put the code in the good folder (`layout` ? `components` ? `pages` ?).
- Replace colors by main variables.
- Play with Bootstrap variables and change theme.

Local data

Website = card repetition

- Medium **X** posts
- Airbnb **X** flats
- Product Hunt **X** products
- Epicurious **X** recipes

Middleman data (https://middlemanapp.com/advanced/local-data/)

Create as many **`data/database.yml`** as you want (`flats.yml` , `team.yml` ..)

YAML file - Array

```
# team.yml
- name: Romain
  role: COO
- name: Seb
  role: CTO
```

Parsed by Middleman as?

```
data.team #=> [{"name" => "Romain", "role" => "COO"}, {"name" => "Seb", "role" => "CTO"}]
```

YAML file - Hash

```
# team.yml
romain:
  picture: romain.png
  role: COO
seb:
  picture: sebastien.png
  role: COO
```

Parsed by Middleman as?

```
data.team #=> {
          #     "romain" => {"picture" => "romain.png", "role" => "COO"},
          #     "seb" => {"picture" => "sebastien.png", "role" => "CTO"}
          #   }
```

Looping on data

```
<div class="container">
  <div class="row">
    <% data.team.each do |member, infos| %>
      <div class="col-md-4">
        <div class="card">
          <%= image_tag infos.picture, class: "img-thumbnail" %>
          <h2><%= member.capitalize %></h2>
          <p><%= infos.role %></p>
        </div>
      </div>
    <% end %>
  </div>
</div>
```

Middleman dynamic pages (https://middlemanapp.com/advanced/dynamic_pages/)

```
# in config.rb
["anne", "seb", "romain"].each do |member|
  proxy "/flats/#{member}.html", "/flats/show.html", :locals => { :owner => member }
end
```

Then in **`source/flats/show.html.erb`**

```
<%= image_tag data.team[owner].picture %>
<h1><%= owner.capitalize %>'s flat</h1>
```

Live-code

- Replace 3 static flats on Home by data read from a `team.yml` file.
- Add a dynamic template `/flats/show.html` showing each member's flat.

Your turn!

JavaScript & jQuery

JavaScript

(*Vanilla* JavaScript)

The Client-side Programming Language

What's with the name?

JavaScript != Java

JS Bin

jsbin.com/?js,console (http://jsbin.com/?js,console)

Outside the Browser

`node` can be used as an equivalent of `irb` and `ruby`.

```
$ node foo.js # Will execute the JS code inside `foo.js` (think `ruby foo.rb`)
$ node        # Gives you a REPL to execute JS code (like `irb` for Ruby code)
>
# Ctrl-C twice to exit
```

To install on OSX:

```
brew update && brew install node
```

On Ubuntu:

```
sudo apt-get install nodejs
sudo ln -s /usr/bin/nodejs /usr/local/bin/node
```

Language Basics

Types

Ruby

```
"Hello Le Wagon"            # String

42                          # Fixnum
42.0                        # Float

true != false               # Boolean
```

JavaScript

```
"Hello Le Wagon"            // string

42                          // number
42.0                        // number

true != false               // boolean
```

Checking type

Ruby

```
name = "Boris"
name.class
# => String
```

JavaScript

```
var name = "Boris";
typeof(name);
// => 'string'
```

More types

Ruby

```ruby
['Hello', 'Le', 'Wagon', 42]      # Array

{ name: 'bob', age: 42 }          # Hash with symbol as keys
{ 'name' => 'bob', 'age' => 42 }  # Hash with string as keys
```

JavaScript

```javascript
['Hello', 'Le', 'Wagon', 42]      // Array

{ name: 'bob', age: 42 }          // Object
{ 'name': 'bob', 'age': 42 }      // Object (the exact same)
```

Variables

Ruby

```ruby
count = 12          # Assignment
count = count + 1   # Re-assignment
```

JavaScript

```javascript
var count = 12;
count = count + 1;
```

`if`

Ruby

```ruby
if condition
  # [...]
end
```

JavaScript

```javascript
if (condition) {
  // [...]
}
```

`for` (1)

Ruby

```ruby
10.times { |i| do_something(i) }

for i in 0...10
  do_something(i)
end
```

JavaScript

```javascript
for (var i = 0; i < 10; i += 1) {
  doSomething(i);
}
```

`forEach`

Ruby

```ruby
beatles = %w(john paul ringo george)
puts beatles[0]

beatles.each do |beatle|
  puts beatle
end
```

JavaScript

```javascript
var beatles = [ "john", "paul", "ringo", "george"];
console.log(beatles[0]);

beatles.forEach(function(beatle) {
  console.log(beatle);
});
```

`Object.keys` (https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Object/keys)

Ruby

```ruby
instruments = { "john" => "guitar", "paul" => "bass", "ringo" => "drums", "george" => "guitar" }

instruments.each do |beatle, instrument|
  puts "#{beatle} played the #{instrument}"
end
```

JavaScript

```javascript
var instruments = { "john": "guitar", "paul": "bass", "ringo": "drums", "george": "guitar" };

Object.keys(instruments).forEach(function(beatle) {
  console.log(beatle + " played the " + instruments[beatle]);
});
```

while

Ruby

```ruby
i = 0
while i < 10
  puts i
  i += 1
end
```

JavaScript

```javascript
var i = 0;
while (i < 10) {
  console.log(i);
  i += 1;
}
```

Functions

Ruby

```ruby
def full_name(first_name, last_name)
  "#{first_name} #{last_name}"
end
```

JavaScript

```javascript
function fullName(firstName, lastName) {
  return firstName + " " + lastName;
}

// or

var fullName = function(firstName, lastName) {
  return firstName + " " + lastName;
}
```

Keywords

Ruby

```ruby
nil
self
```

JavaScript

```javascript
null
this
```

Variable scope

```javascript
function concatenate(a, b) {
  var result = a + b;
  return result;
}

console.log(result)
```

```
// ReferenceError: result is not defined
```

Objects

Trickier than Ruby

Built-in objects

```
var firstName = String("boris");
```

You can call functions on objects.

```
firstName.length;
// => 5
```

More built-ins (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects)

Built-in objects (2)

```
Math.random();
// => 0.555923983729
Date.now();
// => 1422875735069
```

Your own class (Ruby)

```
class User
  def initialize(first_name, last_name)
    @first_name = first_name
    @last_name = last_name
  end

  def full_name
    "#{@first_name} #{@last_name}"
  end
end
```

```
john = User.new("John", "Smith")
john.full_name
# => "John Smith"
```

Your own object (JavaScript)

```
var User = function(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

User.prototype.fullName = function() {
  return this.firstName + " " + this.lastName;
}
```

```
var john = new User("John", "Smith");
john.fullName()
// => "John Smith"
```

Read more about prototypes (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript#Object-oriented_programming)

Using JavaScript with HTML

```
<!DOCTYPE html>
<html>
  <head>

  </head>
  <body>
    <!-- Your content -->

    <!-- Your script tags here -->
    <!-- <script src="first_file.js"></script> -->
    <!-- <script src="other_file.js"></script> -->
  </body>
</html>
```

Loading a JS file is **blocking** the page rendering => Put it at the end.

---

jQuery

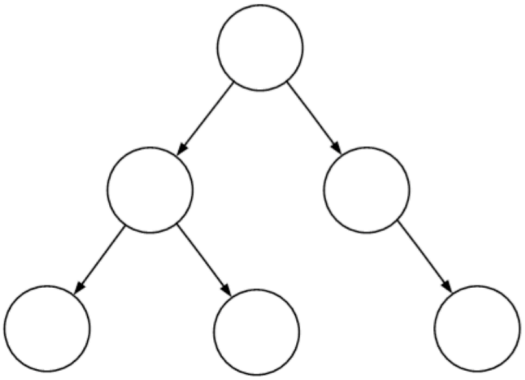
A JavaScript library to ease your life.

Cross-Browser Standardization

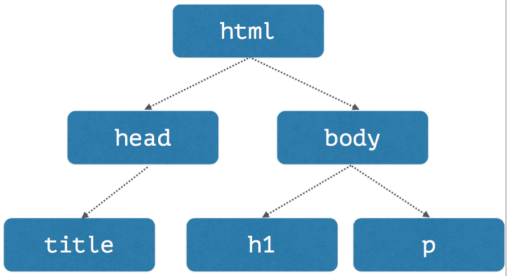
DOM

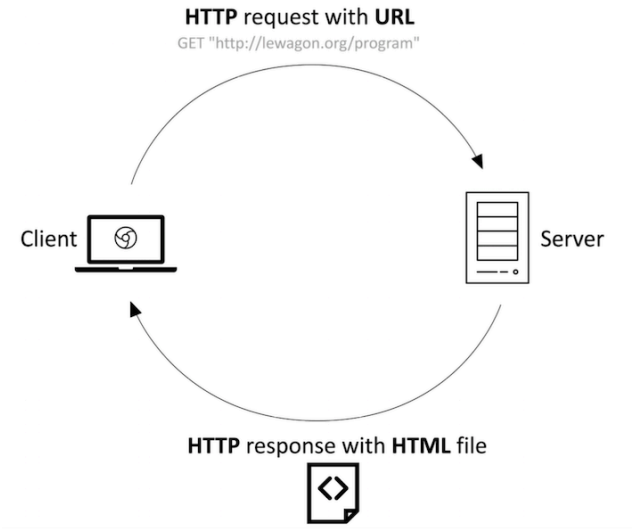
Document Object Model

wikipedia.org/Tree(*data*structure) (https://en.wikipedia.org/wiki/Tree_%28data_structure%29)



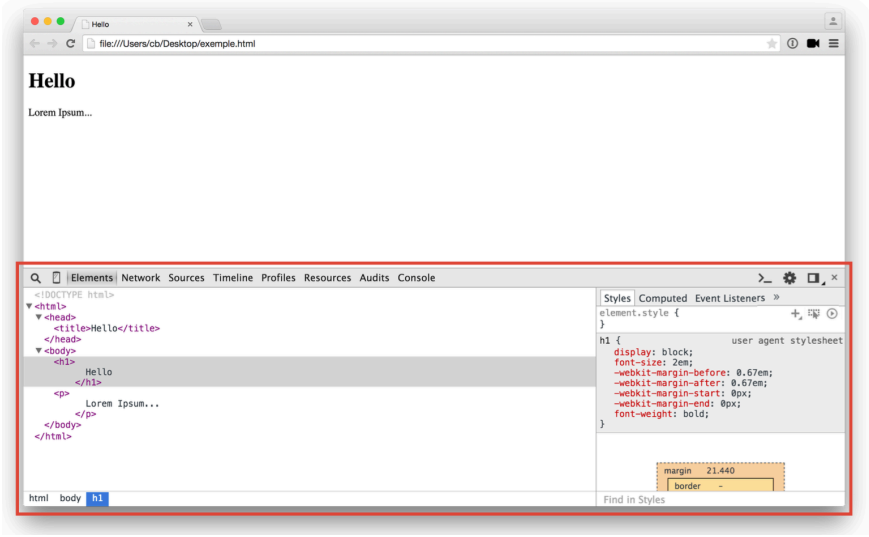
```
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <h1>
      Hello
    </h1>
    <p>
      Lorem Ipsum..
    </p>
  </body>
</html>
```





The browser **parses** the HTML response and **creates** the DOM from it.

You can visualize the DOM in the Chrome Inspector, in **Elements** tab.



First usage : Manipulate the DOM

The `$()` function

```
$('div')                // finds every <div> in the page
```

```
$('.hello')             // finds every tag with a "hello" class
```

```
$('#wagon')             // finds the one element with id="wagon"
```

```
$('td.title > a')          // any CSS selector will do
```

Find all selectors on MDN (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/Selectors)

The `$(selector)` methods return a **jQuery object** containing

a **collection** of DOM elements.

This collection can contain **zero**, **one** or **many** elements.

```
$('body').length
// => 1

$('a').length
// => 24
```

If you matched several elements, you can loop with `$.each()` (http://api.jquery.com/jquery.each/)

```
$('a').each(function(i, item) {
  console.log($(item).attr('href'));
});
```

DO NOT use `forEach` here, `$(a)` does not return a regular array.

```
// As a reminder, use forEach for regular arrays.
var beatles = [ 'john', 'paul', 'ringo'];
beatles.forEach(function(beatle) {
  console.log(beatle);
});
```

Hide & Show elements

```
<div id="wagon">
  Lorem ipsum...
</div>
```

```
$('#wagon').hide();
$('#wagon').show();
```

Doc: `hide` (http://api.jquery.com/hide/) and `show` (http://api.jquery.com/show/)

Add / Remove classes

```
<ul>
  <li class="lead">Paul</li>
  <li class="lead">John</li>
  <li>George</li>
  <li>Ringo</li>
</ul>
```

```
$('.lead').addClass('bold');
```

Doc: `addClass` (http://api.jquery.com/addClass/), `removeClass` (http://api.jquery.com/removeClass/) and `toggleClass` (http://api.jquery.com/toggleClass/)

Read/Write inputs

```
<!-- Some HTML -->
<input name="email" id="email" value="bob@sponge.com" />
```

Read

```
var email = $('#email').val();
// => 'bob@sponge.com'
```

Write

```
$('#email').val('patrick@star.com');
```

Extract text

```
<!-- Some HTML -->
<a href="http://www.lewagon.org" id="home">Le Wagon <em>rocks</em></a>
```

Read

```
var content = $('#home').text();
// => 'Le Wagon rocks'
```

Read/Write HTML

```
<!-- Starting HTML -->
<div id="intro">
  Hello guys
</div>
```

```
$('#intro').html('<a href="http://www.lewagon.org">Le Wagon</a>')
```

```
<!-- Result HTML -->
<div id="intro">
  <a href="http://www.lewagon.org">Le Wagon</a>
</div>
```

Difference between `.text()` and `html()` : check this out (http://jsfiddle.net/hossain/sUTVg/)

Create new elements

```
var paragraph = $('<p>').text("Hello I'm new here");

$('body').append(paragraph);
```

```
<!-- Result HTML -->
<body>
  <p>Hello I'm new here</p>
</body>
```

Second Usage - Events

(Tomorrow)

Third usage - Animations

(Tomorrow)

Fourth usage - AJAX

(In 2 days)

Happy Hacking!

Events/Callbacks & jQuery Plugins

Events

Demo (http://jsbin.com/ropemi/edit?html,js,output)

What happened?

- The JavaScript put a *microphone* on all `img` elements to catch `click` events.
- The user clicked on an image.
- The browser fired the `click` event on the `img` element.
- Some JavaScript code was executed to add the `img-circle` class to the `img` .

What's a callback?

Registering a callback

```
$('img').on('click', function(event) {
  $(this).toggleClass('img-circle');
});
```

the `on()` method takes **2 arguments**:

- Name of the event to listen to, the **String** `'click'`
- JavaScript code to run when the event is triggered, a `function`. `this` is the **DOM element** on which the event happened.

Wait for it

```
$(document).ready(function() {
 // the DOM is now ready, you can traverse it,
 // and register events with on().
});
```

Event types

There's a lot (https://developer.mozilla.org/en-US/docs/Web/Events). jQuery has shortcuts (http://api.jquery.com/category/events/) for most of them

`click`

```
<img class="clickable" src="..." />
```

```
$(".clickable").on("click", function(event) {
  // Do something. `this` is the clicked <img /> DOM node.
});
```

Alternative syntax:

```
$(".clickable").click(function(event) {

});
```

`preventDefault()` (https://api.jquery.com/event.preventDefault/)

```
<a href="#" id="foo">Click here</a>
```

Default behavior is to go **back to top**.

```
$('#foo').on('click', function(event) {
  event.preventDefault(); // Won't go back to top.
  // Do something else
});
```

`hover`

```
<div id="#one"></div>
```

```
$("#one").hover(
  function(event) {
    // Do something when entering the div one.
  },
  function(event) {
    // Do something when leaving the div one.
  }
);
```

Demo (http://jsbin.com/suyira/edit?html,css,js,output)

submit

```
$("#apply").on("submit", function(event) {
  // Code executed when a form is submitted

  return false; // prevent it from being submitted (errors, etc.)
});
```

scroll

```
$(window).on("scroll", function(event) {
  // Code executed for each pixel scrolled
});
```

`focus / blur / change`

`keydown / keyup`

Bootstrap JavaScript

getbootstrap.com/javascript (http://getbootstrap.com/javascript/)

Let's clone lewagon/frontend-advanced-boilerplate (http://github.com/lewagon/frontend-advanced-boilerplate).

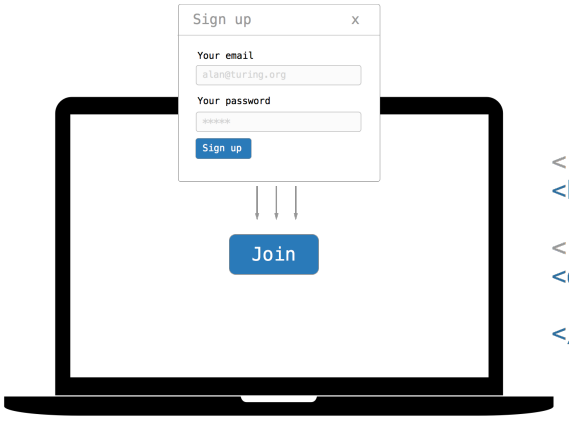
There's a gem (https://github.com/twbs/bootstrap-sass) for that

```
# Gemfile
gem 'bootstrap-sass'
```

```
// all.css.scss
@import "bootstrap-sprockets";
@import "bootstrap";
```

```
// all.js
//= require jquery
//= require bootstrap-sprockets
```

Bootstrap Modal




```
<!-- Button to trigger modal -->
<button data-toggle="modal" data-target="#sign-up">Join</butt

<!-- Modal -->
<div class="modal fade" id="sign-up">
  <!-- Modal content with Sign up form -->
</div>
```

Other useful JS components

Dropdown / Tooltip / Popover / Alert / Carousel

Read the doc (http://getbootstrap.com/javascript/)

Datepicker

Beware of this on **Chrome**

```
<input type="date">
```

Open this (http://jsbin.com/viruhi/edit?html,output) in Firefox/Safari, then Chrome.

Plugin: bootstrap-datepicker

eternicode/bootstrap-datepicker (https://github.com/eternicode/bootstrap-datepicker) and its demo (http://eternicode.github.io/bootstrap-datepicker)

How can I import it in Middleman/Rails?

Is it on Bower? Yes! (https://libraries.io/bower/bootstrap-datepicker)

We can use rails-assets.org (https://rails-assets.org)

```
# Gemfile
source "https://rails-assets.org" do
  gem 'rails-assets-bootstrap-datepicker'
end
```

Then:

```
// all.js

//= require bootstrap-datepicker
```

```
/* all.scss */

@import "bootstrap-datepicker";
```

Let's live-code adding a datepicker to Airbnb home!

```
$ git clone git@github.com:lewagon/middleman-airbnb.git
```

Other components

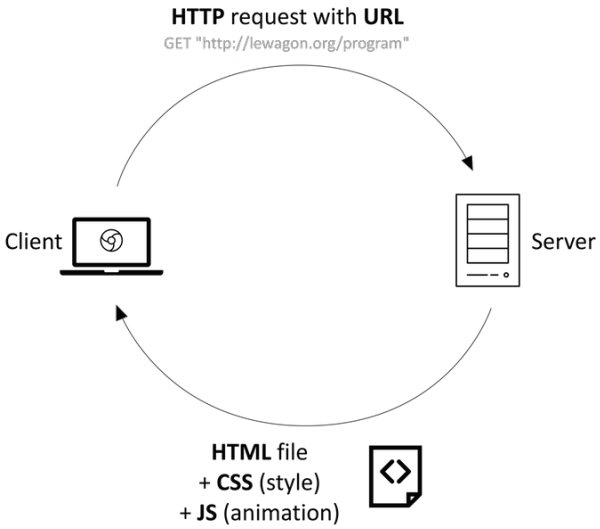
(All available on Bower, thus rails-assets)

swipebox (http://brutaldesign.github.io/swipebox/) / scrollreveal (https://scrollrevealjs.org/) / smoothscroll (https://libraries.io/bower/smoothscroll)

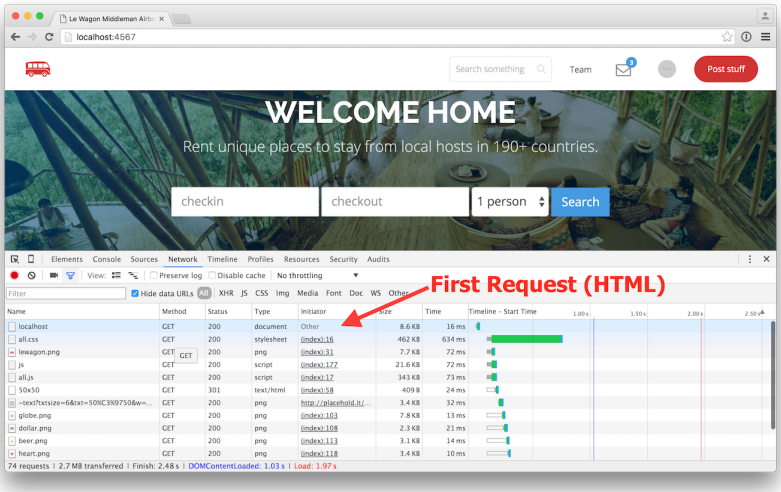
Happy Hacking!

HTTP & AJAX

How does the web work?



1 HTTP request / file (`file.html` , `file.css` , `file.js` , `file.png`)



HTTP

Base protocol of the **Web** (!= Internet)

Hyper

Text

Transfer

Protocol

What is it?

- A **protocol** to transfer **resources**
- Resource means **file** (HTML, CSS, images, JS, ...)
- Based on a system of **request / response**
- Between 2 machines, **a client and a server**

Let's look at an HTTP request

It's **more** than just an URL

1 - Request Method

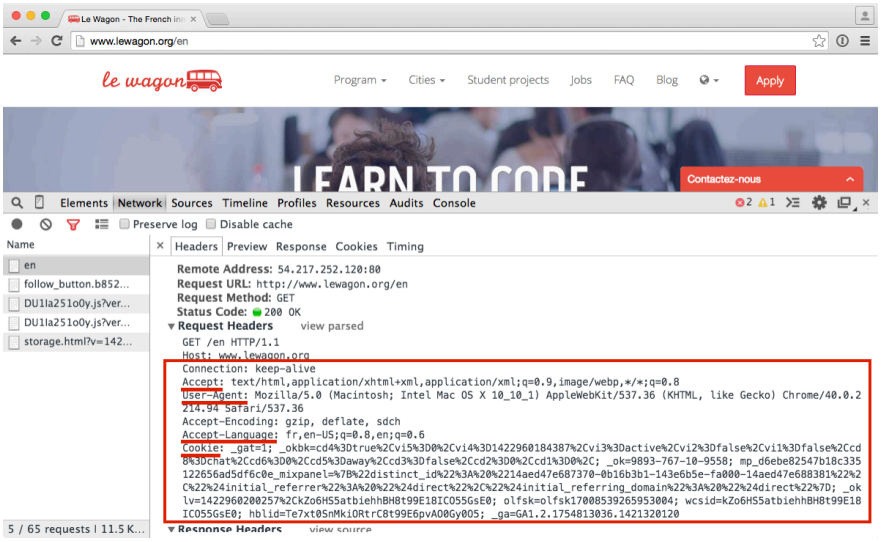
- **GET**: get a page
  `=> I want to read page http://www.google.com`
- **POST**: send new data
  `=> Here are my email and password, I want to sign up!`

  Later, we'll see **PATCH** and **DELETE**.

2 - Request URL

`http://www.lewagon.org`

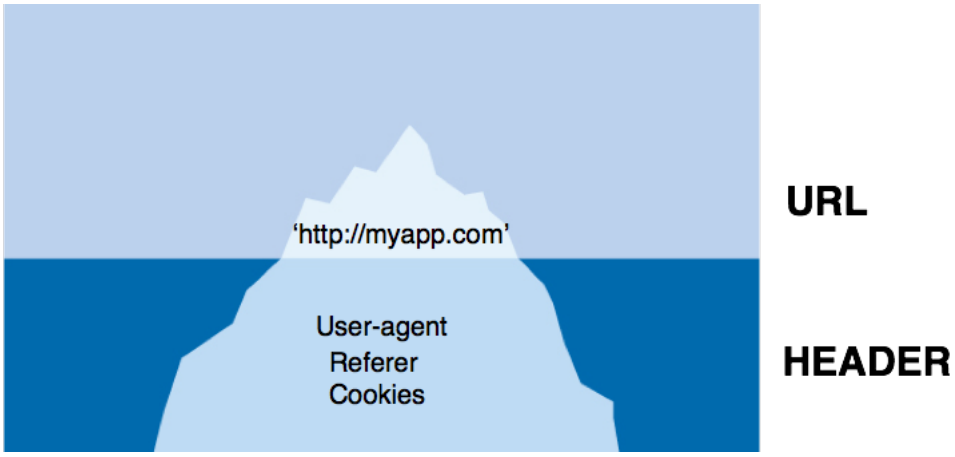
3 - Request Headers



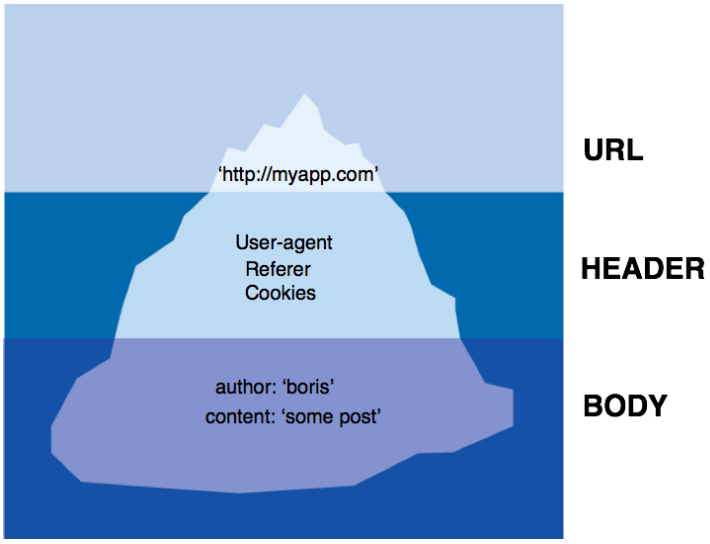
4 - Request Body

- No body for a `GET` Request
- Body for a `POST` or `PATCH` request (usually with content from **form**)

GET



POST



Get - params in URL

- No body in GET request. No way to hide params in the body
- **Query Parameters** appear directly in the URL
- More convenient to share a search with your friends :)

Example: https://www.airbnb.com/s/Torino?checkin=10%2F02%2F2015&checkout=27%2F02%2F2015&guests=4 (https://www.airbnb.com/s/Torino?checkin=10%2F02%2F2015&checkout=27%2F02%2F2015&guests=4)

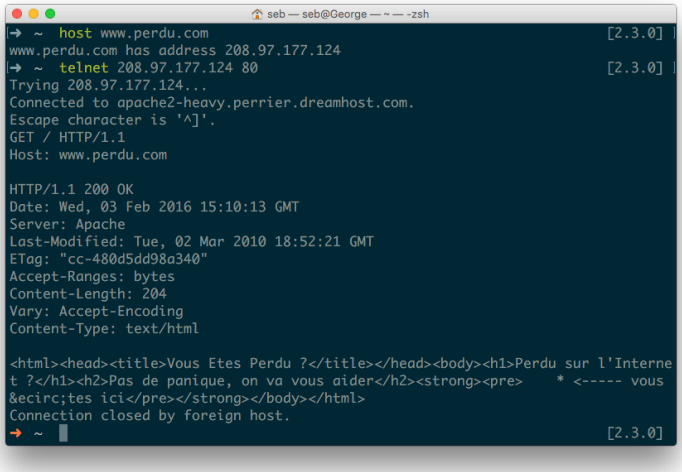
Post - params in body

```
<form action="/users/sign_in" method="POST">
  <input name="username" type="text" >
  <input name="password" type="password">
  <input type="submit">
</form>
```

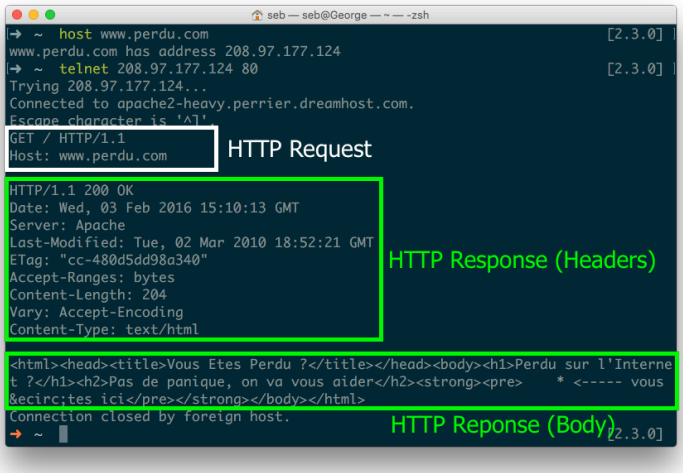
When posting this form, we'll generate this HTTP request:

```
POST /users/sign_in HTTP/1.1
Content-Type: application/x-www-form-urlencoded

username=boris&password=secret
```

Speak HTTP (no browser required)



Press `Enter` twice



`curl`

```
$ curl http://www.perdu.com
```

```
$ curl http://www.lewagon.com
$ curl -I http://www.lewagon.com
```

AJAX

- **A**synchronous **J**avascript **A**nd **X**ML
- Historical naming, no more links with XML

What is it?

- Allows to make **HTTP requests in Javascript**

- AJAX requests are **asynchronous** (non-blocking), **after** page loaded.
- Build by Microsoft in 1998 for Internet Explorer 5

`$.ajax` (https://api.jquery.com/jQuery.ajax/) with `GET`

Let's **read** some info about GitHub repositories owned by `lewagon`.

```
$.ajax({
  type: "GET",
  url: "https://api.github.com/users/lewagon/repos",
  success: function(data) {
    console.log(data);
  },
  error: function(jqXHR) {
    console.error(jqXHR.responseText);
  }
});
```

Demo (https://jsfiddle.net/ssaunier/4c90e805/3/)

Shortcut: `$.get()` (https://api.jquery.com/jQuery.get)

```
$.get("https://api.github.com/users/lewagon/repos", function(data) {
  // success callback;
  console.log(data);
})
```

`$.ajax` with `POST`

Let's **create** a new gist (see API doc (https://developer.github.com/v3/gists/#create-a-gist))

```
var gist = JSON.stringify({
  "description": "A gist from AJAX",
  "public": true,
  "files": {
    "README.md": { "content": "It worked!" }
  }
});
$.ajax({
  type: "POST",
  url: "https://api.github.com/gists",
  data: gist,
  success: function(data) {
    console.log("Successfully created gist at " + data.html_url);
  }
});
```

Demo (https://jsfiddle.net/ssaunier/72zmm6zh/11/)

Your turn!

Product Design Sprint

Product Design Sprint

https://github.com/lewagon/product-design

Slides

- Download keynote (https://github.com/lewagon/product-design/blob/master/Le%20Wagon%20-%20Product%20Design%20Sprint.key?raw=true)
- Download PDF (https://github.com/lewagon/product-design/raw/master/Le%20Wagon%20-%20Product%20Design%20Sprint.pdf)

Sketch Demo

(mockup)

Marvelapp Demo

(prototype)

Happy prototyping!

Rails Basics

Install Rails

```
gem install rails -v 5.0.3
```

Restart your terminal (⌘ + Q) Then, check setup with:

```
rails --version
```

History

Created in 2003 by David Heinemeier Hansson, while working on Basecamp.

Extracted Ruby on Rails and released it as open source code in July of 2004

3 principles

- Ruby
- MVC
- Programmer happiness

Releases

```
1.0   December 13, 2005
1.2   January 19, 2007

2.0   December 7, 2007
2.1   June 1, 2008
2.2   November 21, 2008
2.3   March 16, 2009

3.0   August 29, 2010
3.1   August 31, 2011
3.2   January 20, 2012

4.0   June 25, 2013
4.1   April 8, 2014
4.2   December 19, 2014

5.0   June 30, 2016
```

Philosophy

- Convention over Configuration
- Don't Repeat Yourself

(both mean write less code)

Going strong

4500+ contributors (http://contributors.rubyonrails.org/).

Code is on GitHub at rails/rails (http://github.com/rails/rails)

Community

+100k gems at rubygems.org (http://rubygems.org/). How do I know which one to use?

- Ask teachers
- Browse the Ruby Toolbox (https://www.ruby-toolbox.com/categories/by_group)

How every new rails project starts

Create a new rails app

First, go to your personal code folder:

```
cd ~/code/$GITHUB_USERNAME
```

Then create a new rails app

```
rails new lacuillere -T
```

This creates a new folder `~/code/$GITHUB_USERNAME/lacuillere` .

Set up git

```
cd lacuillere
pwd
# => ~/code/$GITHUB_USERNAME/lacuillere
git init
git add .
git commit -m "Starting awesome development with Rails :)"
```

Push project to GitHub

```
git remote -v
# => No remotes yet! Cannot push!
```

Install the `hub` binary, with `brew install hub` (Mac) or `gem install hub` (Linux). Then, just run:

```
hub create
```

```
git remote -v
# => An `origin` remote is now set!
git push -u origin master  # Push the generated rails app to GitHub
hub browse                 # Will open your browser to the new directory
```

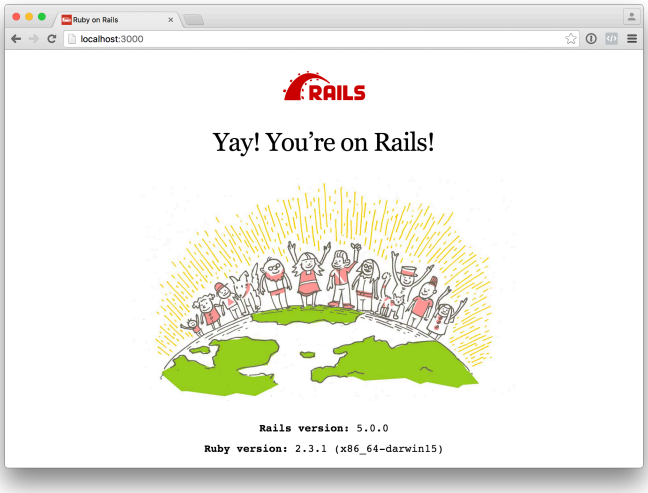
Launch the rails server

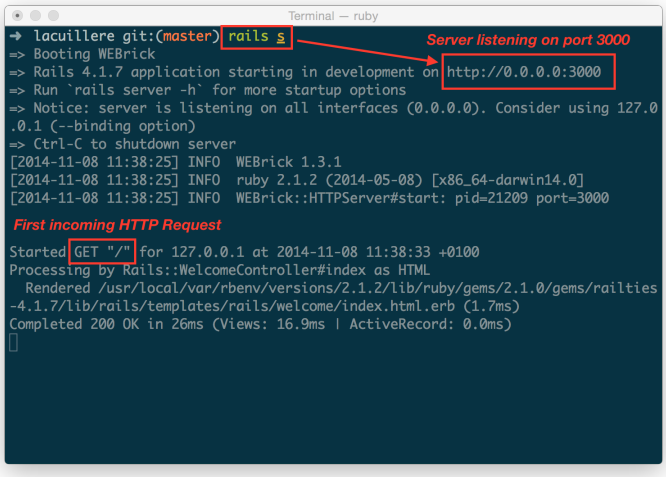
Open a **new tab** in your terminal:

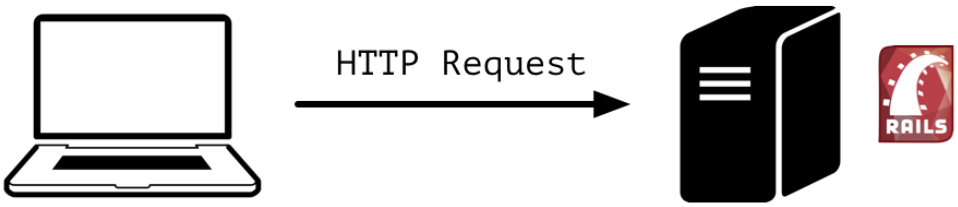
```
cd ~/code/$GITHUB_USERNAME/lacuillere # if not already there
rails s
```

Keep this tab **opened**!

Open your terminal and go to http://localhost:3000 (http://localhost:3000)








```
GET http://localhost:3000/
```

```
verb  scheme  host  port  path
```

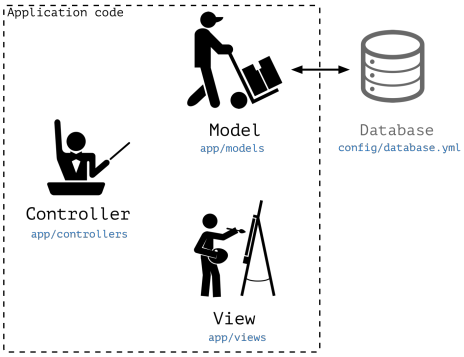
Rails Architecture

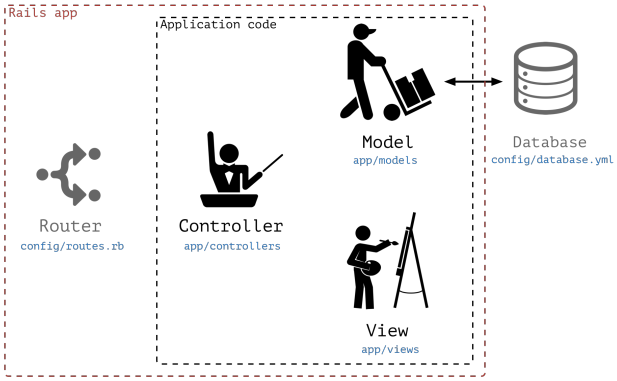
Open the project in Sublime Text:

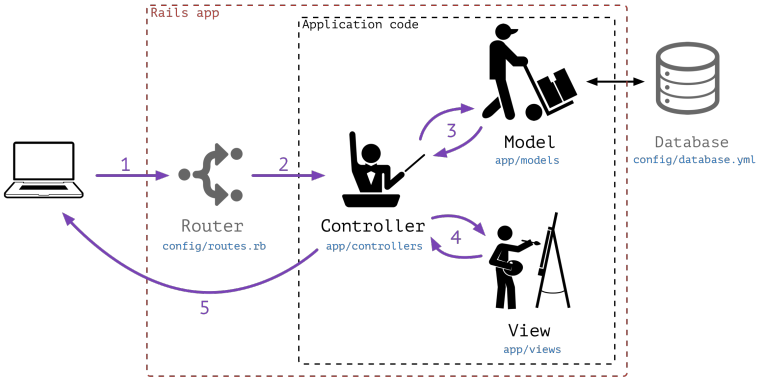
```
pwd
# => ~/code/$GITHUB_USERNAME/lacuillere
stt
```

```
.
├── app
│   ├── controllers
│   │   └── application_controller.rb
│   ├── models
│   └── views
│       └── layouts
│           └── application.html.erb
└── config
    ├── database.yml
    └── routes.rb
```

MVC Revision







Controller

Let's add basic pages to our app (contact, about).

We need a new controller, which we'll generate:

```
rails generate controller pages contact about
#      create  app/controllers/pages_controller.rb
#       route  get 'pages/about'
#       route  get 'pages/contact'
#      invoke  erb
#      create    app/views/pages
#      create    app/views/pages/contact.html.erb
#      create    app/views/pages/about.html.erb
```

I can now navigate to:

- http://localhost:3000/pages/contact (http://localhost:3000/pages/contact)
- http://localhost:3000/pages/about (http://localhost:3000/pages/about)

The generator created 2 routes, you can find them in `config/routes.rb` .

```
# config/routes.rb
Rails.application.routes.draw do
  get 'pages/contact'
  get 'pages/about'
end
```

```ruby
# app/controllers/pages_controller.rb
class PagesController < ApplicationController
  def contact
  end

  def about
  end
end
```

```
.
└── app
    └── views
        └── pages
            ├── about.html.erb
            └── contact.html.erb
```

Customizing routes

```ruby
# config/routes.rb
Rails.application.routes.draw do
  get 'about', to: 'pages#about'
  get 'contact', to: 'pages#contact'

  # Generic syntax:
  # verb 'path', to: 'controller#action' (action is an instance method)
end
```

We ditched the `/pages` from the URL path:

- http://localhost:3000/about (http://localhost:3000/about)
- http://localhost:3000/contact (http://localhost:3000/contact)

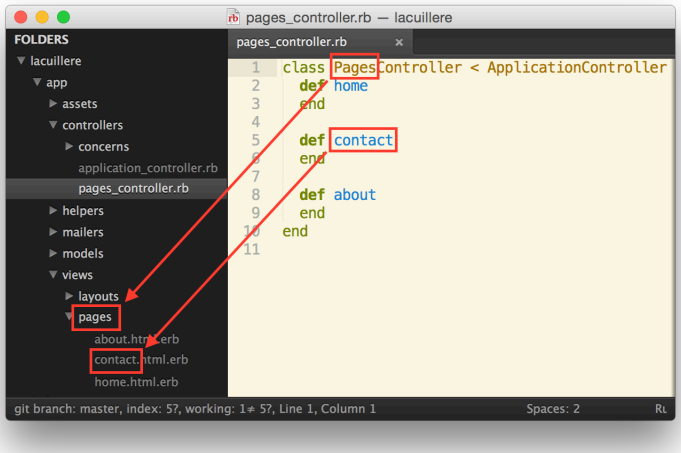
Root path

```ruby
# config/routes.rb
Rails.application.routes.draw do
  # [...]
  root to: 'pages#home'
end
```

```ruby
# app/controllers/pages_controller.rb
class PagesController < ApplicationController
  def home
  end
  # [...]
end
```

```
.
└── app
    └── views
        └── pages
            └── home.html.erb
```

Convention over Configuration



And if you get lost

```
rails routes
```

```
 Prefix Verb URI Pattern        Controller#Action
   root GET  /                  pages#home
  about GET  /about(.:format)   pages#about
contact GET  /contact(.:format) pages#contact
```

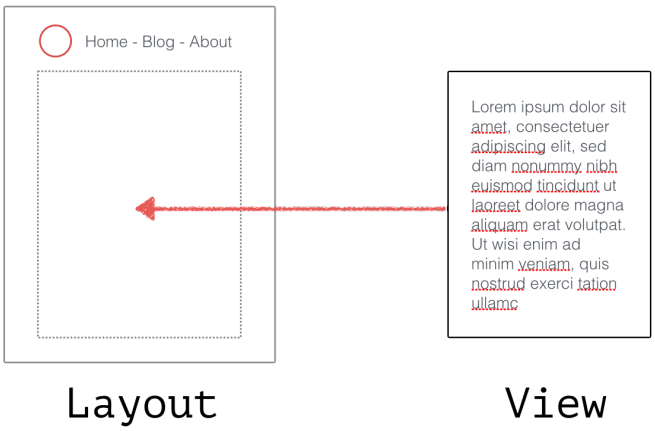
Live-code - One more time

Let's add a new route to list our restaurants:

```
Verb    URI Pattern    Controller#Action
GET     /restaurants   restaurants#index
```

View

```
.
└── app
    └── views
        ├── layouts
        │   └── application.html.erb
        ├── pages
        │   ├── about.html.erb
        │   ├── contact.html.erb
        │   └── home.html.erb
        └── restaurants
            └── index.html.erb
```




```
<!-- app/views/layouts/application.html.erb -->

<!DOCTYPE html>
<html>
<head>
  <title>Lacuillere</title>
  <%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track' => true %>
  <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
  <%= csrf_meta_tags %>
</head>
<body>

<h1>La Cuillere</h1>

<%= yield %>

<p>A very simple footer</p>

</body>
</html>
```

A typical generated view

```
<!-- app/views/pages/home.html.erb -->
<h1>Pages#home</h1>
<p>Find me in app/views/pages/home.html.erb</p>
```

View is inserted in its layout at the line:

```
<%= yield %>
```

ERB

View files are `.html.erb` (".erb" stands for "**e**mbedded **r**u**b**y").

We will mix Ruby inside HTML.

Syntax

- You can write standard HTML
- You can execute ruby code inside `<% %>`
- You can execute ruby code and add it to the HTML with `<%= %>` (~ `puts`)

It's dynamic!

We can use some Ruby :)

```
<p>Now is <%= Time.now %><p>
```

```
<p>Now is 20XX-XX-XX XX:XX:XX +0000</p>
```

Loop

```
<% categories = [ 'japanese', 'indian', 'french' ] %>

<h2>Find lots of restaurants</h2>
<ol>
<% categories.each do |category| %>
  <li><%= category.capitalize %></li>
<% end %>
</ol>
```

```
<h2>Find lots of restaurants</h2>
<ol>
  <li>Japanese</li>
  <li>Indian</li>
  <li>French</li>
</ol>
```

Controller <=> View

Controller instance variables...

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  # Let's fake a DB
  RESTAURANTS = [
    { name: "Dishoom", address: "Shoreditch, London", category: "indian" },
    { name: "Sushi Samba", address: "City, London", category: "japanese" }
  ]
  def index
    @restaurants = RESTAURANTS
  end
end
```

...are accessible by the associated action view.

```
<!-- app/views/restaurants/index.html.erb -->
<ul>
<% @restaurants.each do |restaurant| %>
  <li><%= restaurant[:name]%> (<%= restaurant[:address] %>)</li>
<% end %>
</ul>
```

Where's the model?

Let's anticipate tomorrow's lecture.

Tomorrow with ActiveRecord

```
class RestaurantsController < ApplicationController
  def index
    @restaurants = Restaurant.all # no more fake DB
  end
end
```

Controller instance variables will often be model instances or array of model instances.

Params

Params coming from the Query String

```
<!-- app/views/pages/home.html.erb -->
<form action="/restaurants" method="get">
  <input type="text" name="food_type" placeholder="What kind of food?">
  <input type="submit">
</form>
```

Clicking on the submit button, the browser will make the following request:

```
GET /restaurants?food_type=something_you_typed
```

The controller can then retrieve this parameter passed in the **query string**.

```
# app/controllers/restaurants_controller.rb
class RestaurantsController
  def index
    @category = params[:food_type]
    @restaurants = RESTAURANTS.select {|r| r[:category] == @category }
  end
end
```

Query string

Everything between the `?` and the `#` in the URL.

```
GET /some_path?first_name=alan&last_name=turing#some-facultative-anchor
```

```
# params is the following hash:
{
  first_name: "alan",
  last_name: "turing"
}
```

Params coming from the request body

When do we have a request body?

POST

```
<!-- app/views/pages/home.html.erb -->
<form action="/restaurants" method="post">
  <%= hidden_field_tag :authenticity_token, form_authenticity_token %>
  <input type="text" name="name">
  <input type="text" name="address">
  <input type="submit">
</form>
```

Clicking on the submit button, the browser will make the following request:

```
Header: POST /restaurants
Body: name=name_you_typed&address=address_you_typed
```

```
# config/routes.rb
Rails.application.routes.draw do
  post '/restaurants', to: 'restaurants#create'
end
```

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    render plain: "Add to DB restaurant '#{params[:name]}' with address '#{params[:address]}'"
  end
end
```

Tomorrow with ActiveRecord

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    @restaurant = Restaurant.new(name: params[:name], address: params[:address])
    @restaurant.save
  end
end
```

Params coming from the URL path

```ruby
# config/routes.rb
Rails.application.routes.draw do
  get 'restaurants/:id', to: 'restaurants#show'
end
```

When the browser navigates to the following URL:

```
GET /restaurants/23
```

The controller can then retrieve this parameter passed in the **path**.

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def show
    @restaurant = RESTAURANTS[params[:id].to_i]
  end
end
```

```erb
<!-- app/views/restaurants/show.html.erb -->
<h1>More infos on <%= @restaurant[:name] %></h1>
<p>Find us at <%= @restaurant[:address] %></p>
```

Tomorrow with ActiveRecord

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def show
    @restaurant = Restaurant.find(params[:id])
  end
end
```

Summary

The `params` hash is populated from 3 sources:

- The URL **query string** arguments
- The **body** of a `POST` request
- The URL **path** of parametric routes

Links

```erb
<a href="ANCHOR_URL">ANCHOR_TEXT</a>
```

Use `link_to`

```erb
<%= link_to ANCHOR_TEXT, ANCHOR_URL %>
```

`ANCHOR_URL` will use a **path helper** based on the route name.

```ruby
# config/routes.rb
Rails.application.routes.draw do
  get 'about',       to: 'pages#about'
  get 'contact',     to: 'pages#contact'
  get 'restaurants', to: 'restaurants#index'
end
```

```
$ rails routes
     Prefix Verb URI Pattern               Controller#Action
      about GET  /about(.:format)          pages#about
    contact GET  /contact(.:format)        pages#contact
restaurants GET  /restaurants(.:format)    restaurants#index
```

You can use in the view:

```
<%= link_to "Know more about us", about_path %>
<%= link_to "Discover all our restaurants", restaurants_path %>
```

Define your own helper

```
# config/routes.rb
Rails.application.routes.draw do
  get 'hello', to: 'pages#welcome', as: :welcome
end
```

```
rails routes
 Prefix Verb URI Pattern        Controller#Action
welcome GET  /hello(.:format)   pages#welcome
```

You can use in the view:

```
<%= link_to "Say hi", welcome_path %>
```

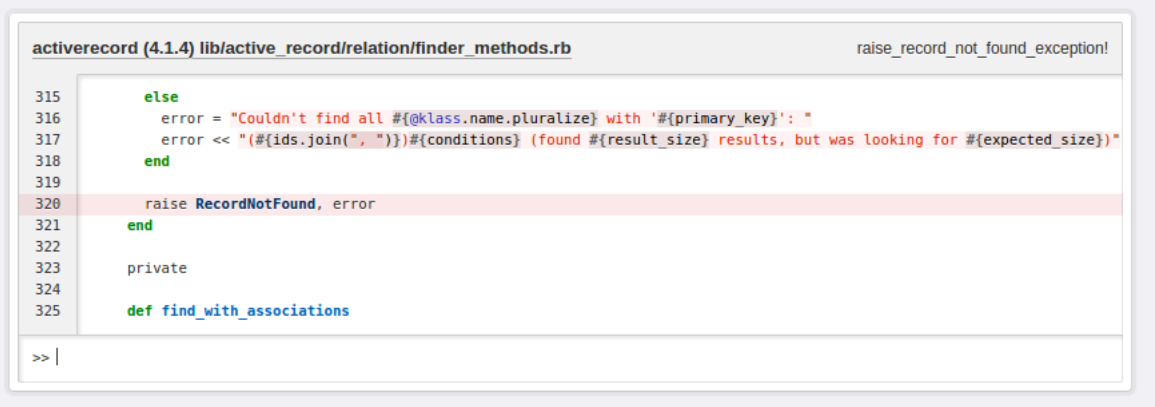
Useful tools

Better errors gem

```
# Gemfile
group :development do
  gem "better_errors"
  gem "binding_of_caller"
end
```

Then run in your terminal

```
bundle install
```



Will basically give you an IRB in your browser each time you get an error.

Your turn! Dive into the **Routing**/**Controller**/**View** with the exercises.

Tomorrow, we'll add the **Model** layer with the well-known **ActiveRecord**!

Rails CRUD

ActiveRecord in Rails

- **The same good old ActiveRecord** you **already know**
- Rails just has some useful generators for models & migrations.

Model Generator

```
rails generate model Restaurant name:string stars:integer
```

```
      invoke  active_record
      create    db/migrate/20141108192211_create_restaurants.rb
      create    app/models/restaurant.rb
```

Auto-generated files

```ruby
# db/migrate/20141108192211_create_restaurants.rb
class CreateRestaurants < ActiveRecord::Migration[5.0]
  def change
    create_table :restaurants do |t|
      t.string :name
      t.integer :stars

      t.timestamps
    end
  end
end
```

```ruby
# app/models/restaurant.rb
class Restaurant < ApplicationRecord
end
```

Do not forget: apply the migration to schema

```
rails db:migrate
```

This will create (and update then) the `db/schema.rb` file which holds all your table definitions. Refer to it when in doubt about a table name or a column name.

And of course

```
git status
git add .
git commit -m "Add restaurant model"
```

`rails destroy`

Just after the `rails g model`, if you made a typo on your model name, run:

```
rails destroy model Restaurant
```

It will remove the migration and the `app/models/restaurant.rb`.

Migration generator

Not the same as **model** generator! Use it to **update** an **existing** model.

```
rails g migration AddAddressToRestaurants
```

```
      invoke  active_record
      create    db/migrate/20141108192916_add_address_to_restaurants.rb
```

This generator creates a file skeleton. There's still work to do.

Write your migration code

The migration is empty by default. Populate the `change` method.

```ruby
# db/migrate/20141108192916_add_address_to_restaurants.rb

class AddAddressToRestaurants < ActiveRecord::Migration[5.0]
  def change
    add_column :restaurants, :address, :string
  end
end
```

```
rails db:migrate
```

```
git add .
git commit -m "Add address to Restaurant"
```

Methods to be used in `change`

- `add_column`
- `change_column`
- `rename_column`
- `remove_column`

(And for 1:n, `add_reference`)

Read more on Migrations (http://guides.rubyonrails.org/migrations.html)

Rake DB tasks

- **`rails db:drop`** - Drop the database (lose all your data!)
- **`rails db:create`** - Create the database with an empty schema
- **`rails db:migrate`** - Run pending migrations on the database schema
- **`rails db:rollback`** - Revert the last migration
- **`rails db:reset`** - Drop database + replay all migration

Rails console

```
rails console  # or `rails c`
```

You can type ruby, with the **rails environment loaded**.

```
[1] pry(main)> Restaurant.all
#  Restaurant Load (1.4ms)  SELECT "restaurants".* FROM "restaurants"
# => #<ActiveRecord::Relation []>
```

No more `require_relative`!

Use the console to create or explore your records.

Rails console tips

If you updated your model, do not restart the console. Just type:

```
[1] pry(main)> reload!
```

To exit the console, just type:

```
[1] pry(main)> exit
```

Sandbox

```
rails console --sandbox
```

All the changes you did (create / update / delete) will be rollbacked at exit.

CRUD

Read (all)

http://localhost:3000/restaurants (http://localhost:3000/restaurants)

```
# config/routes.rb
Rails.application.routes.draw do
  get "restaurants", to: "restaurants#index"
end
```

```
class RestaurantsController < ApplicationController
  def index
    @restaurants = Restaurant.all
  end
end
```

Read (one)

http://localhost:3000/restaurants/4 (http://localhost:3000/restaurants/4)

```
# config/routes.rb
Rails.application.routes.draw do
  get "restaurants/:id", to: "restaurants#show"
end
```

```
class RestaurantsController < ApplicationController
  def show
    @restaurant = Restaurant.find(params[:id])
  end
end
```

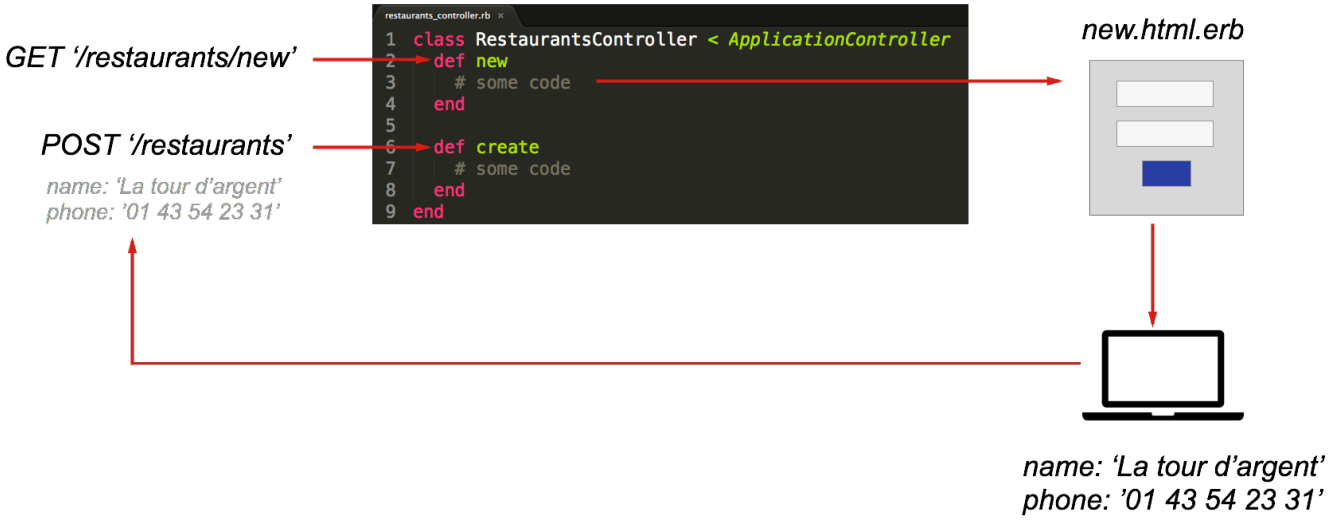
Create

What are the steps?

1. `GET` the restaurant creation form = **1 request**
2. `POST` the parameters to create a new restaurant = **1 request**

How many requests?

2 requests



Create - Step 1, `GET` the form

`GET` http://localhost:3000/restaurants/new (http://localhost:3000/restaurants/new)

```
# config/routes.rb
Rails.application.routes.draw do
  get "restaurants/new", to: "restaurants#new"
end
```

```
class RestaurantsController < ApplicationController
  def new
    ?  # We'll see that in a moment.
  end
end
```

Create - Step 2, `POST` the form

`POST` http://localhost:3000/restaurants (http://localhost:3000/restaurants)

```
# config/routes.rb
Rails.application.routes.draw do
  post "restaurants", to: "restaurants#create"
end
```

```
class RestaurantsController < ApplicationController
  def create
    @restaurant = ?  # We'll see that in a moment.
  end
end
```

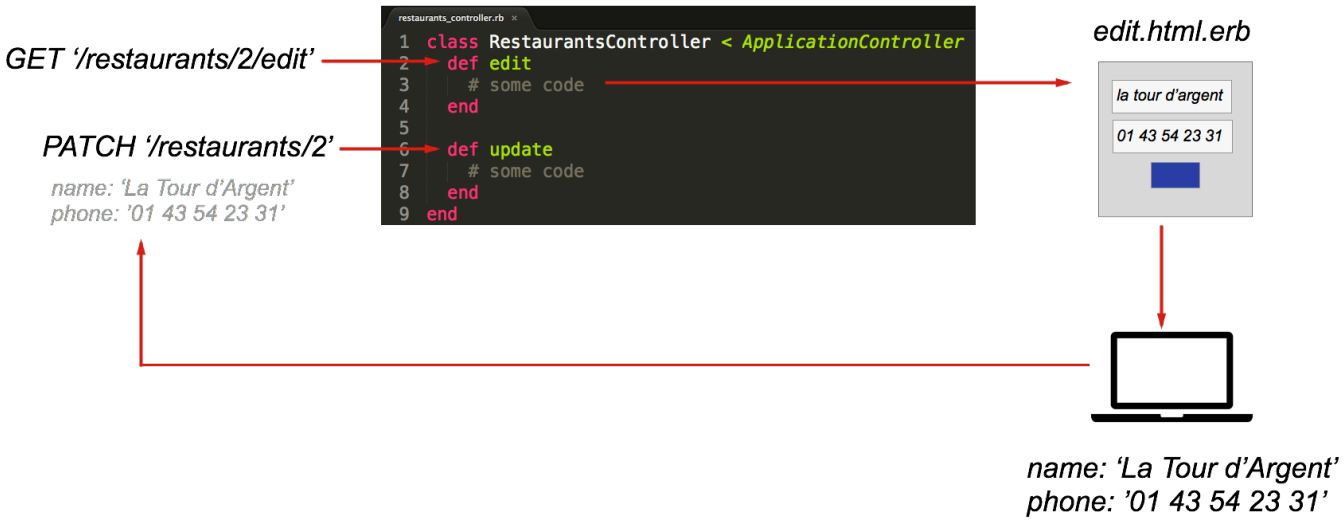
Update

What are the steps?

1. `GET` the HTML form (pre-filled with restaurant attributes) for editing = **1 request**
2. `PATCH` the parameters to update an existing restaurant = **1 request**

How many requests?

2 requests again!



Update - Step 1, `GET` the form

`GET` http://localhost:3000/restaurants/4/edit (http://localhost:3000/restaurants/4/edit)

```
# config/routes.rb
Rails.application.routes.draw do
  get "restaurants/:id/edit", to: "restaurants#edit"
end
```

```
class RestaurantsController < ApplicationController
  def edit
    @restaurant = Restaurant.find(params[:id])
  end
end
```

Update - Step 2, `PATCH` the form

`PATCH` http://localhost:3000/restaurants/4 (http://localhost:3000/restaurants/4)

```
# config/routes.rb
Rails.application.routes.draw do
  patch "restaurants/:id", to: "restaurants#update"
end
```

```
class RestaurantsController < ApplicationController
  def update
    @restaurant = Restaurant.find(params[:id])
    @restaurant.? # We'll see that in a moment.
  end
end
```

Delete

`DELETE` http://localhost:3000/restaurants/4 (http://localhost:3000/restaurants/4)

```
# config/routes.rb
Rails.application.routes.draw do
  delete "restaurants/:id", to: "restaurants#destroy"
end
```

```
class RestaurantsController < ApplicationController
  def destroy
    @restaurant = Restaurant.find(params[:id])
    @restaurant.destroy
  end
end
```

Summary

There are 7 CRUD routes

```
Rails.application.routes.draw do
  get    "restaurants",          to: "restaurants#index"

  get    "restaurants/:id",      to: "restaurants#show"

  get    "restaurants/new",      to: "restaurants#new"
  post   "restaurants",          to: "restaurants#create"

  get    "restaurants/:id/edit", to: "restaurants#edit"
  patch  "restaurants/:id",      to: "restaurants#update"

  delete "restaurants/:id",      to: "restaurants#destroy"
end
```

Know them by heart

But don't write them all!

You can express the 7 routes shown before with just one line in `config/routes.rb` :

```
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants
end
```

Subset of the 7 routes

```
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants, only: [:create, :index, :destroy]
end
```

You can also use the `except` keyword.

Controller <=> View

7 standard CRUD routes => 7 instance methods

```
# app/controller/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def index         # GET /restaurants
  end

  def show          # GET /restaurants/:id
  end

  def new           # GET /restaurants/new
  end

  def create        # POST /restaurants
  end

  def edit          # GET /restaurants/:id/edit
  end

  def update        # PATCH /restaurants/:id
  end

  def destroy       # DELETE /restaurants/:id
  end
end
```

Read (all)

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def index
    @restaurants = Restaurant.all
  end
end
```

```
<!-- app/views/restaurants/index.html.erb -->
<% @restaurants.each do |restaurant| %>
  <h2><%= restaurant.name %></h2>
  <p><%= restaurant.address %></p>
<% end %>
```

Read (one)

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def show
    @restaurant = Restaurant.find(params[:id])
  end
end
```

```
<!-- app/views/restaurants/show.html.erb -->
<h2><%= @restaurant.name %></h2>
<p><%= @restaurant.address %></p>
```

Create & Update

Remember, 2 requests for create:

- The first one to `GET` the empty HTML form
- The second one to `POST` this form

And for update, 2 requests as well:

- The first one to `GET` the pre-populated HTML form of existing record
- The second one to `PATCH` this form

Building the `new` form, by hand

```
<form action="/restaurants" method="post">
  <label for="name">Name: </label>
  <input type="text" name="restaurant[name]" id="name">

  <label for="address">Address: </label>
  <input type="text" name="restaurant[address]" id="address">

  <input type="submit">
</form>
```

On `POST`, the `params` will be populated with

```
# params
{
  restaurant: {
    name: "La tour d'argent",
    address: "15 Quai de la Tournelle"
  }
}
```

Building the `edit` form, by hand

Same as `new` form, but with `value` attributes:

```
<label for="name">Name: </label>
<input type="text" name="restaurant[name]" id="name"
       value="La Tour d'Argent">
```

DRY

`form_for`

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

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

  <%= f.submit %>
<% end %>
```

disclaimer: behaves differently for **new** record or **existing** record.

`@restaurant`

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def new
    @restaurant = Restaurant.new
  end

  def edit
    @restaurant = Restaurant.find(params[:id])
  end
end
```

new.html.erb **AND** edit.html.erb

```ruby
<%= form_for(@restaurant) do |f| %>
  <%= f.label :name %>
  <%= f.text_field :name %>
  <%= f.label :address %>
  <%= f.text_field :address %>
  <%= f.submit %>
<% end %>
```

DRY

Partial View

Define the `_form` partial view (partials starts with a `_`) `erb <!-- app/views/restaurants/_form.html.erb --> <%= form_for(@restaurant) do |f| %> <%= f.label :name %> <%= f.text_field :name %> <%= f.label :address %> <%= f.text_field :address %> <%= f.submit %> <% end %>`

Use this partial in `new` and `edit` views: `erb <!-- app/views/restaurants/new.html.erb --> <%= render 'form' %>`

```ruby
<!-- app/views/restaurants/edit.html.erb -->
<%= render 'form' %>
```

What now?

We built the form.

We need to handle the `POST` and `PATCH` requests coming from them!

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    @restaurant = Restaurant.new(params[:restaurant])
    @restaurant.save
    # Will raise ActiveModel::ForbiddenAttributesError
  end

  def update
    @restaurant = Restaurant.find(params[:id])
    @restaurant.update(params[:restaurant])
    # Will raise ActiveModel::ForbiddenAttributesError
  end
end
```

Strong Params

Why?

Suppose we have this schema and this app.

```ruby
# db/schema.rb
create_table "users" do |t|
  t.string   "email"
  t.string   "encrypted_password"
  t.boolean  "admin"
  t.datetime "created_at"
  t.datetime "updated_at"
end
```

```ruby
<!-- app/views/users/_form.html.erb -->
<%= form_for(@user) do |f| %>
  <%= f.text_field :email %>
  <%= f.submit %>
<% end %>
```

```ruby
<form action="/users/1">
  <input type="text" name="user[email]" value="bob@leponge.com">
  <input type="submit">
</form>
```

A malicious user could **update** the HTML in the browser

```
<form action="/users/1">
  <input type="text" name="user[email]" value="bob@leponge.com">
  <input type="hidden" name="user[admin]" value="true">
  <input type="submit">
</form>
```

And submit the form

```
# app/controllers/users_controller.rb
class UsersController < ApplicationController
  def update
    @user = User.find(params[:id])
    @user.update(params[:user])
    # Will raise ActiveModel::ForbiddenAttributesError
  end
end
```

```
# app/controllers/users_controller.rb
class UsersController < ApplicationController
  def update
    @user = User.find(params[:id])
    @user.update(user_params)
  end

  private

  def user_params
    params.require(:user).permit(:email)
  end
end
```

Summary: Never trust user data

Rails forces you to **whitelist** the `params` with `require` and `permit`.

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    @restaurant = Restaurant.new(restaurant_params)
    @restaurant.save
  end

  def update
    @restaurant = Restaurant.find(params[:id])
    @restaurant.update(restaurant_params)
  end

  private

  def restaurant_params
    # *Strong params*: You need to *whitelist* what can be updated by the user
    # Never trust user data!
    params.require(:restaurant).permit(:name, :address)
  end
end
```

What next?

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    @restaurant = Restaurant.new(restaurant_params)
    @restaurant.save

    # no need for app/views/restaurants/create.html.erb
    redirect_to restaurant_path(@restaurant)
  end

  def update
    @restaurant = Restaurant.find(params[:id])
    @restaurant.update(restaurant_params)

    # no need for app/views/restaurants/update.html.erb
    redirect_to restaurant_path(@restaurant)
  end

  private

  def restaurant_params
    params.require(:restaurant).permit(:name, :address)
  end
end
```

Delete

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def destroy
    @restaurant = Restaurant.find(params[:id])
    @restaurant.destroy

    # no need for app/views/restaurants/destroy.html.erb
    redirect_to restaurants_path
  end
end
```

```ruby
<%= link_to "Delete", restaurant_path(restaurant),
      method: :delete,
      data: { confirm: "Are you sure?" } %>
```

Filters

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def show
    @restaurant = Restaurant.find(params[:id])
    # ...
  end
  def edit
    @restaurant = Restaurant.find(params[:id])
    # ...
  end
  def update
    @restaurant = Restaurant.find(params[:id])
    # ...
  end
  def destroy
    @restaurant = Restaurant.find(params[:id])
    # ...
  end
end
```

DRY

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  before_action :set_restaurant, only: [:show, :edit, :update, :destroy]

  def show
    # ...
  end
  def edit
    # ...
  end
  def update
    # ...
  end
  def destroy
    # ...
  end

  private

  def set_restaurant
    @restaurant = Restaurant.find(params[:id])
  end
end
```

Links and Route names

`link_to` helper

```erb
<%= link_to(title, url) %>
```

```erb
<!-- app/views/restaurants/index.html.erb -->
<% @restaurants.each do |restaurant| %>
  <%= link_to restaurant.name, restaurant_path(restaurant) %>
<% end %>
```

```erb
<a href="/restaurants/1">Chez Gladines</a>
<a href="/restaurants/2">Le Temps des Cerises</a>
<a href="/restaurants/3">La Tour d'Argent</a>
```

`rake routes`

```
         Prefix Verb   URI Pattern                    Controller#Action
    restaurants GET    /restaurants(.:format)         restaurants#index
                POST   /restaurants(.:format)         restaurants#create
 new_restaurant GET    /restaurants/new(.:format)     restaurants#new
edit_restaurant GET    /restaurants/:id/edit(.:format) restaurants#edit
     restaurant GET    /restaurants/:id(.:format)     restaurants#show
                PATCH  /restaurants/:id(.:format)     restaurants#update
                DELETE /restaurants/:id(.:format)     restaurants#destroy
```

Add `_path` to the route prefix (left column).

Your turn! Let's build a simple CRUD app with just one model.

Beyond CRUD

Quick setup

Debug gems

```ruby
group :development, :test do
  gem "better_errors"
  gem "binding_of_caller"
end
```

Bootsrap sass

```ruby
# Gemfile
gem "bootstrap-sass"
```

```
// application.js (require bootstrap AFTER jquery)
//= require jquery
//= require bootstrap-sprockets
```

```
/* application.scss (change file extension to .scss) */
@import "bootstrap-sprockets";
@import "bootstrap";
```

```
<!-- application.html.erb -->
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
</head>
```

Simple form

```
gem "simple_form"
```

Then, run:

```
$ bundle install
$ rails generate simple_form:install --bootstrap
```

First commit

```
$ git init
$ git add .
$ git commit -m "rails new with debug and frontend gems"
```

CRUD

(Previous lecture)

```
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants
end
```

```
         Prefix Verb   URI Pattern                    Controller#Action
    restaurants GET    /restaurants(.:format)         restaurants#index
                POST   /restaurants(.:format)         restaurants#create
 new_restaurant GET    /restaurants/new(.:format)     restaurants#new
edit_restaurant GET    /restaurants/:id/edit(.:format) restaurants#edit
     restaurant GET    /restaurants/:id(.:format)     restaurants#show
                PATCH  /restaurants/:id(.:format)     restaurants#update
                DELETE /restaurants/:id(.:format)     restaurants#destroy
```

Scaffold generator

- Useful for **quick demo** but not for real projects
- It generates useless files (`scaffold.css.scss` ..)
- You don't always need all of the 7 CRUD actions

```
rails g scaffold Restaurant name:string address:string description:text stars:integer
```

```
rails db:migrate
```

```ruby
# db/seeds.rb
puts 'Cleaning database...'
Restaurant.destroy_all

puts 'Creating restaurants...'
restaurants_attributes = [
  {
    name:         "Epicure au Bristol",
    address:      "112 rue du Fg St-Honoré 75008 Paris",
    description:  "Face au jardin, on découvre une salle lumineuse... et la cuisine d'Éric Frechon.",
    stars:        3
  },
  {
    name:         "La truffière",
    address:      "4 rue Blainville 75005 Paris",
    description:  "Une valeur sûre que cette belle maison du 17e et les recettes de Jean-Christophe Rizet",
    stars:        1
  },
  {
    name:         "Le pré catelan",
    address:      "route de Suresnes 75016 Paris",
    description:  "Oeil vif, geste sûr: impossible de distinguer, dans les créations de Frédéric Anton...",
    stars:        3
  }
]
Restaurant.create!(restaurants_attributes)
puts 'Finished!'
```

```
rails db:seed
```

Beyond CRUD

You are not limited to the **seven** routes that RESTful routing creates by default.

Say you want a route to list **three stars** restaurants. Something like:

```
GET /restaurants/top
```

```ruby
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants do
    collection do                       # collection => no restaurant id in URL
      get 'top', to: "restaurants#top"  # RestaurantsController#top
    end
  end
end
```

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def top
    @restaurants = Restaurant.where(stars: 3)
  end
end
```

Now we want a route to **display info about the chef of a restaurant**.

```
GET /restaurants/42/chef
```

Add a chef to Restaurants table

```
rails generate migration AddChefToRestaurants chef:string
rails db:migrate
```

```ruby
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants do
    member do                           # member => restaurant id in URL
      get 'chef', to: "restaurants#chef"  # RestaurantsController#chef
    end
  end
end
```

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  before_action :find_restaurant, only: [ :chef ]

  def chef
    # TODO: you can use @restaurant here
  end

  private

  def find_restaurant
    @restaurant = Restaurant.find(params[:id])
  end
end
```

Nested Resources

Now we want routes to **add a review on a restaurant**

- `GET /restaurants/42/reviews/new`
- `POST /restaurants/42/reviews`

Models

```
rails generate model Review content:string restaurant:references
rails db:migrate
```

```ruby
# app/models/restaurant.rb
class Restaurant < ApplicationRecord
  has_many :reviews, dependent: :destroy
end
```

```ruby
# app/models/review.rb
class Review < ApplicationRecord
  belongs_to :restaurant
end
```

Routing

```ruby
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants do
    resources :reviews, only: [ :new, :create ]
  end
end
```

Will add 2 new routes

Prefix	Verb	Path	Controller#Action
new_restaurant_review	GET	/restaurants/:restaurant_id/reviews/new	reviews#new
restaurant_reviews	POST	/restaurants/:restaurant_id/reviews	reviews#create

```ruby
# app/controllers/reviews_controller.rb
class ReviewsController < ApplicationController
  def new
    # we need @restaurant in our `simple_form_for`
    @restaurant = Restaurant.find(params[:restaurant_id])
    @review = Review.new
  end

  def create
    @review = Review.new(review_params)
    # we need `restaurant_id` to asssociate review with corresponding restaurant
    @review.restaurant = Restaurant.find(params[:restaurant_id])
    @review.save
  end

  private

  def review_params
    params.require(:review).permit(:content)
  end
end
```

View

Use `simple_form_for` with nested resources

```
# app/views/reviews/new.html.erb
<%= simple_form_for [@restaurant, @review] do |f| %>
  <%= f.input :content %>
  <%= f.submit "add a review", class: "btn btn-primary" %>
<% end %>
```

Helper	HTTP request
`form_for(@review)`	POST **`reviews_path`**
`form_for([@restaurant, @review])`	POST **`restaurant_reviews_path`**

Question

Do we **have to** nest routes **because** we have a 1:N relation between 2 models?

🤔

Answer

NO WE **DON'T**!!!

⛔

It's just a convenient way to have **restaurant's id** in **params**!

Guess the appropriate routing

Get one restaurant's reviews

Guess the appriopriate routing

We need corresponding restaurant's id!

```
Rails.application.routes.draw do
  get "restaurants/:restaurant_id/reviews", to: "reviews#index"
end
```

Or, even simpler, display reviews in restaurant's show:

```
Rails.application.routes.draw do
  get "restaurants/:id",                    to: "restaurants#show"
end
```

Guess the appriopriate routing

Get one review's details

Guess the appriopriate routing

We **just** need corresponding review's id!

```
Rails.application.routes.draw do
  get "reviews/:id", to: "reviews#show"
end
```

Guess the appriopriate routing

Get one review's edit form page

Guess the appriopriate routing

Again, we **just** need corresponding review's id!

```
Rails.application.routes.draw do
  get "reviews/:id/edit", to: "reviews#edit"
end
```

etc...

Wrapping it up (1)

No need to nest **every** reviews CRUD route in restaurants!

Ask yourself what do you **need** in **params** and you'll know if nesting is appropriate

```
Rails.application.routes.draw do
  get "restaurants/:restaurant_id/reviews",     to: "reviews#index"
  get "restaurants/:restaurant_id/reviews/new", to: "reviews#new"
  post "restaurants/:restaurant_id/reviews",    to: "reviews#create"
  get "reviews/:id",                            to: "reviews#show"
  get "reviews/:id/edit",                       to: "reviews#edit"
  patch "reviews/:id",                          to: "reviews#update"
  delete "reviews/:id",                         to: "reviews#destroy"
end
```

Wrapping it up (2)

Using `resources`

```
Rails.application.routes.draw do
  resources :restaurants do
    resources :reviews, only: [ :index, :new, :create ]
  end
  resources :reviews, only: [ :show, :edit, :update, :destroy ]
end
```

Namespaced routing

What if?

- You want `restaurants#index` to list **all** restaurants
- You want another `restaurants#index` to list only restaurants **you have created**

Routing

```
# config/routes.rb
Rails.application.routes.draw do
  namespace :admin do
    resources :restaurants, only: [:index]
  end
end
```

Will add 1 new route

Prefix	Verb	Path	Controller#Action
admin_restaurants	GET	/admin/restaurants	admin/restaurants#index

2 Restaurants Controller

Usual one

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def index
    @restaurants = Restaurant.all
  end
end
```

The new one

```
# app/controllers/admin/restaurants_controller.rb
class Admin::RestaurantsController < ApplicationController
  def index
    # Let's anticipate on next week (with login)
    @restaurants = current_user.restaurants
  end
end
```

View

```
# app/views/admin/restaurants/index.html.erb
<h1>My list of restaurants</h1>
<% @restaurants.each do |restaurant| %>
  <h2><%= restaurant.name %></h2>
  <p><%= restaurant.address %></p>
<% end %>
```

All about routing (http://guides.rubyonrails.org/routing.html)

Validation

```
# app/models/restaurant.rb
class Restaurant < ApplicationRecord
  validates :stars, inclusion: { in: [1,2,3], allow_nil: false }
  validates :name, uniqueness: true, presence: true
  validates :address, presence: true
end
```

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    @restaurant = Restaurant.new(restaurant_params)
    @restaurant.save
    # Unless @restaurant.valid?, #save will return false,
    # and @restaurant is not persisted.
    # TODO: present the form again with error messages.
    redirect_to restaurant_path(@restaurant)
  end

  private

  def restaurant_params
    params.require(:restaurant).permit(:name, :address)
  end
end
```

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    @restaurant = Restaurant.new(restaurant_params)
    if @restaurant.save
      redirect_to restaurant_path(@restaurant)
    else
      render :new
    end
  end

  private

  def restaurant_params
    params.require(:restaurant).permit(:name, :address)
  end
end
```

In the view, you can use:

```
@restaurant.errors
```

```
<!-- app/views/restaurants/_form.html.erb -->
<%= form_for(@restaurant) do |f| %>
  <% if @restaurant.errors.any? %>
    <div id="error_explanation">
      <ul>
      <% @restaurant.errors.full_messages.each do |message| %>
        <li><%= message %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <!-- [...] all the fields -->
<% end %>
```

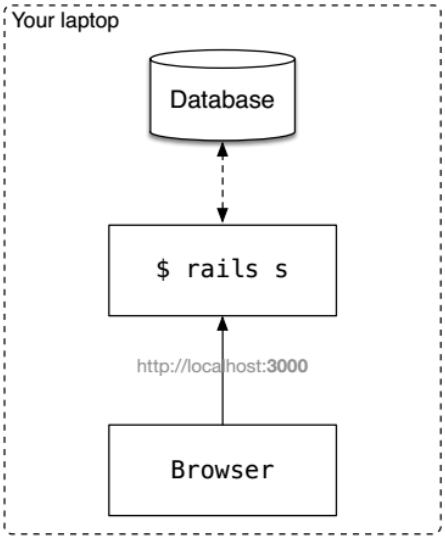
Do not forget to add the same logic to the `update` method in your controller!

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def update
    if @restaurant.update(restaurant_params)
      redirect_to restaurant_path(@restaurant)
    else
      render :edit
    end
  end
end
```

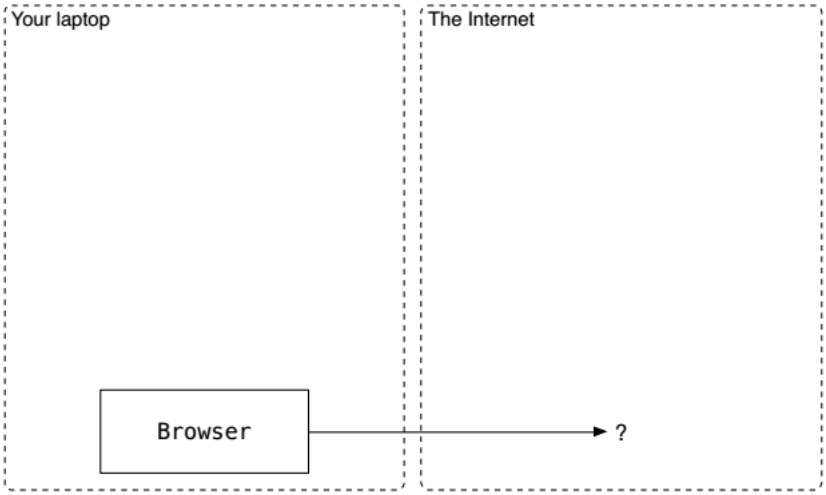
Documentation (http://guides.rubyonrails.org/active_record_validations.html#displaying-validation-errors-in-views)

Hosting & Deployment

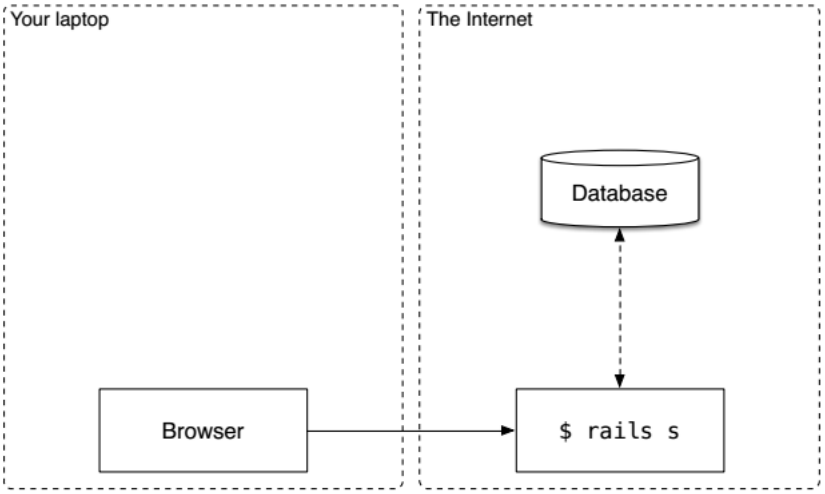
Development



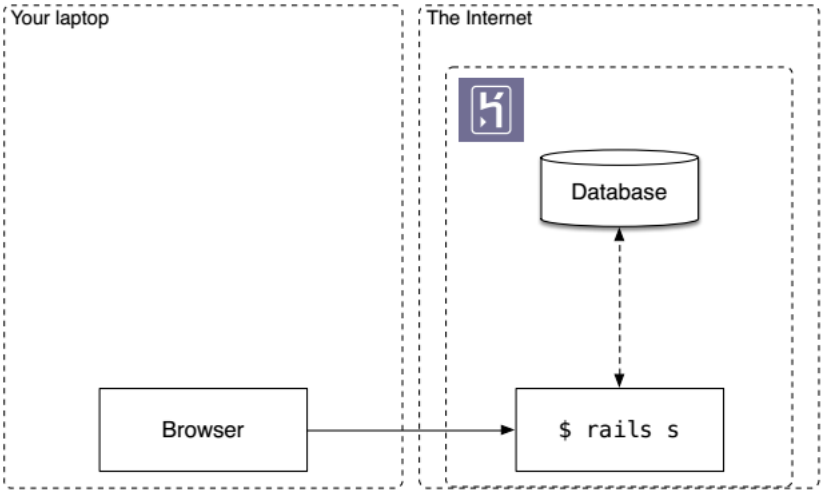
Production



Production



Production



SQLite is over. Say hello to

Install on OS X

Should already be there from the first day setup. In the terminal, you can run:

```
psql -d postgres
# psql (9.5.3)
# Type "help" for help.

# postgres=#
# Type \q to quit
```

Otherwise:

```
brew install postgresql
brew services start postgresql
```

Install on Linux

Same, should already be installed. If not, go here (https://github.com/lewagon/setup/blob/master/UBUNTU.md#postgresql)

New Rails application

Generate the rails app with the `--database` flag:

```
cd ~/code/$YOUR_GITHUB_USERNAME
rails new $YOUR_APP_NAME -T --database=postgresql
cd $YOUR_APP_NAME
```

Init and push your git repo

```
git init
git add .
git commit -m "Initial commit"
hub create
git push origin master
```

Create the database on PostgreSQL

```
rails db:create
```

Look at your `Gemfile` :

```
# Gemfile
gem 'pg'  # No more `gem "sqlite"` thanks to `--database=postgresql`
```

Scaffold a Post model

Open `Gemfile` , and comment out the `jbuilder` gem.

```
# gem 'jbuilder'
```

Then generate a **scaffold** of `article` .

```
rails generate scaffold article title body:text
rails db:migrate
```

```
git add .
git commit -m "Add Article scaffold"
```

```
rails s
# Check that everything's working at http://localhost:3000/articles
```

`root`

Your app does not have any route for `/` . Add one!

```
# config/routes.rb
Rails.application.routes.draw do
  # [...]

  root 'articles#index'
end
```

Prepare for Heroku

Open your Gemfile and add this line

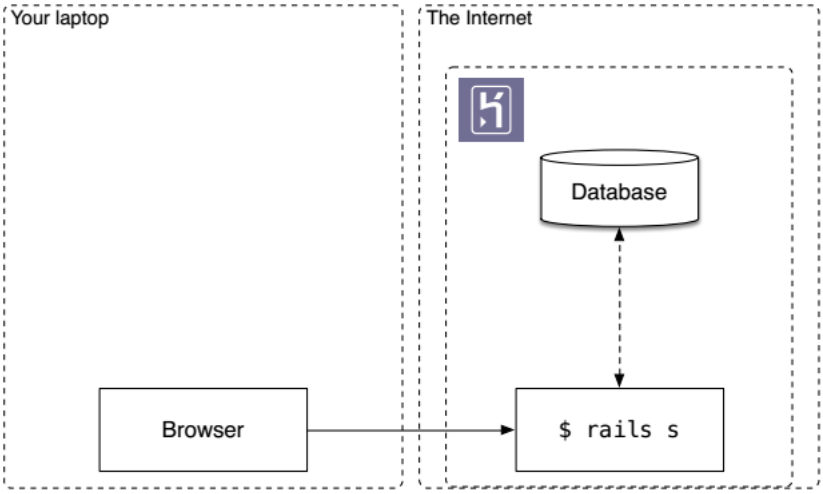
```
# Gemfile
ruby '2.3.3'
```

Commit your changes:

```
git add .
git commit -m "Prepare for Heroku"
```

Let's deploy this blog!

Production



Sign up

https://id.heroku.com/signup

Install on OS X

```
# OS X
brew install heroku
```

Install on Ubuntu

```
wget -qO- https://toolbelt.heroku.com/install-ubuntu.sh | sh
```

Help

```
heroku
Usage: heroku COMMAND [--app APP] [command-specific-options]
Primary help topics, type "heroku help TOPIC" for more details:
  addons     # manage addon resources
  apps       # manage apps (create, destroy)
  auth       # authentication (login, logout)
  config     # manage app config vars
  domains    # manage custom domains
  logs       # display logs for an app
  ps         # manage dynos (dynos, workers)
  releases   # manage app releases
  run        # run one-off commands (console, rake)
  sharing    # manage collaborators on an app
```

Login

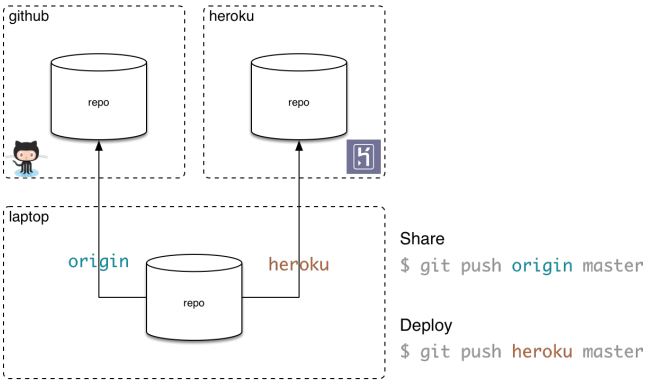
```
heroku login
```

Create an Heroku app

```
heroku create $YOUR_APP_NAME --region eu
```

What happened?

```
git remote -v
```



Let's deploy!

Push your code to Heroku

```
git push heroku master
```

Run a command on Heroku

```
heroku run rails db:migrate
```

Done!

Some useful commands

```
heroku open         # open in your browser
```

```
heroku logs --tail  # show the app logs and keep listening
```

```
heroku run rails c  # Run the production console
```

Happy Hacking!

Frontend setup

Assets pipeline

What is an asset?

- CSS file
- JS file
- Image

Pipeline

5 steps to make your assets **production-ready**

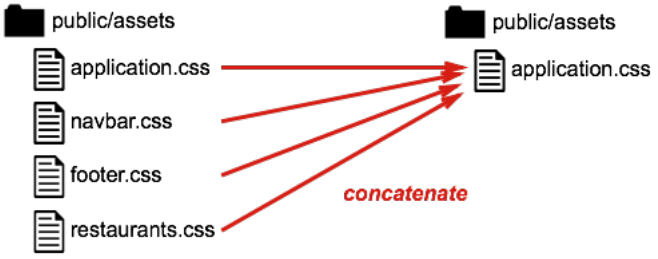
- Precompile (coffee => js, scss => css)
- Concatenate
- Minify
- Cache (stamp)
- Compress

Precompile

Your **production assets** go in `public/assets`

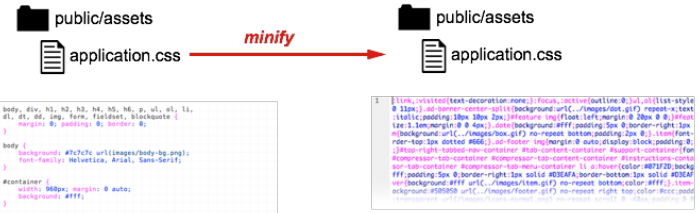
Concatenate

1 file = **1 HTTP request** to load your CSS



Minify

Less characters = **Smaller file**



Stamps

- Add a new stamp every time the file's binary code changes
- Without this stamp, browser would always use files in cache.

Compress

Browser will use the compressed version for too big files.

Rails front-end setup

Extensive version

Gemfile

```
# Gemfile
gem "bootstrap-sass"
gem "font-awesome-sass"
gem "simple_form"
gem "autoprefixer-rails"
```

```
bundle install
rails generate simple_form:install --bootstrap
```

- https://github.com/twbs/bootstrap-sass
- https://github.com/FortAwesome/font-awesome-sass
- https://github.com/plataformatec/simple_form

Stylesheets

We don't like Rails choices for organizing CSS. Let's use lewagon/rails-stylesheets (https://github.com/lewagon/rails-stylesheets)

```
rm -rf app/assets/stylesheets
curl -L https://github.com/lewagon/rails-stylesheets/archive/master.zip > stylesheets.zip
unzip stylesheets.zip -d app/assets && rm stylesheets.zip && mv app/assets/rails-stylesheets-master app/assets/stylesheets
```

Javascript

- `require` directives **are not** javascript comments
- They load js files (such as `require` in ruby)
- Add `bootstrap-sprockets` after `jquery`.

```
// app/assets/javascripts/application.js

//= require jquery
//= require jquery_ujs
//= require bootstrap-sprockets
//= require_tree .
```

Layout

Don't forget viewport meta in the layout for Bootstrap responsiveness.

```
<!-- app/views/layouts/application.html.erb -->
<head>
  <!-- Add these line for detecting device width -->
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
</head>
```

External CSS/JS plugins

If you're lucky, there's a good gem (like `bootstrap-sass`)

External CSS/JS plugins

Otherwise, look in Rails assets (https://rails-assets.org/), then

```
# Gemfile
source 'https://rails-assets.org' do
  gem 'rails-assets-plugin'
end
```

```
// assets/stylesheets/application.scss
@import "plugin";
```

```
// assets/javascripts/application.js
//= require jquery
//= require plugin
//= require_tree .
```

Warning: plugin js file should be loaded **after jQuery** but **before your own JS files**.

Rails Helpers - advanced use

`link_to`

- Never use `<a>` tags with absolute URLs
- URLs should be defined in `routes.rb` only!
- Use `link_to` with path helpers (ex: `something_path`)
- Add HTML attributes (like `class` and `id`) in last hash argument

```
<%= link_to "All articles", articles_path, {class: "btn btn-primary"} %>
```

```
<a href="/articles" class="btn btn-primary">All articles</a>
```

```
<!-- #<Article: id=1, title="My first blog post", body="Lorem ipsum"> -->
<%= link_to @article.title, article_path(@article), {class: "btn btn-primary"} %>
```

```
<a href="/articles/1" class="btn btn-primary">My first blog post</a>
```

```
<!-- #<Article: id=1, title="My first blog post", body="Lorem ipsum"> -->
<%= link_to article_path(@article) do %>
  <%= @article.title %>
  <span class="label label-primary">NEW</span>
<% end %>
```

```
<a href="/articles/1">
  My first blog post <span class="label label-primary">NEW</span>
</a>
```

`image_tag`

- Never use `<img>` tags with absolute sources
- Rails will add MD5 stamp in production
- Use `image_tag`
- Add HTML attributes (like `class` and `id`) in last hash argument

In **development** mode

```
<%= image_tag "logo.png", {class: "img-circle", id: "logo"}%>
```

```
<img href="assets/logo.png" class="img-circle" id="logo">
```

Later in **production**

```
<%= image_tag "logo.png", {class: "img-circle", id: "logo"}%>
```

```
<img src="assets/logo-f800df98aa2605c552400667353f47d2.png"
     class="img-circle" id="logo">
```

`form_for`

- Don't use `<form>` tags with absolute URLs
- Rails must add an authenticity token to your form
- Don't use `form_for` because it sucks
- Use `simple_form_for` . Like `form_for` , but simpler

Using `simple_form_for`

```
<%= simple_form_for(@article) do |f| %>
  <%= f.error_notification %>

  <%= f.input :title %>
  <%= f.input :body %>

  <%= f.button :submit %>
<% end %>
```

- Only one method `=> input`
- Automatic labels
- With Bootstrap config, automatic `form-group` and `form-control` classes

Simple form - Cheat Sheet

```
<!-- Deactivate label -->
<%= f.input :name, label: false %>
<!-- Change default label -->
<%= f.input :name, label: "Entrez votre nom" %>
<!-- Add placeholder -->
<%= f.input :name, placeholder: "Bob" %>
<!-- Add HTML attributes on input -->
<%= f.input :name, input_html: {class: "input-lg"} %>
<!-- Add HTML attributes on label -->
<%= f.input :name, label_html: {class: "label-lg"} %>
<!-- Use collection for select -->
<%= f.input :stars, collection: 1..5, prompt: "Nb of stars", selected: 3 %>
```

You can combine them all! Read the doc (https://github.com/plataformatec/simple_form)

Front-End best practices

1 - Use Bootstrap SASS variables

- Bootstrap SASS has a lot of variables (http://getbootstrap.com/customize/#less-variables)
- Override them in **`config/_bootstrap_variables.scss`**
- Don't write 1000 lines of CSS if there's a variable for that..

2 - Structure your layout with shared partials

```
<!-- app/views/layouts/application.html.erb -->
<html>
  <head>
    <!-- [...] -->
  </head>
  <body>
    <%= render "shared/navbar" %>
    <%= yield %>
    <%= render "shared/footer" %>
    <!-- For inserting some analytics script (GA, Mixpanel..) -->
    <%= render "shared/analytics" %>
  </body>
</html>
```

```
└── views
    ├── layouts
    │   └── application.html.erb
    └── shared
        ├── _navbar.html.erb
        ├── _footer.html.erb
        └── _analytics.html.erb
```

3 - Use our Navbar

- Integrate our ERB template (https://github.com/lewagon/awesome-navbars/blob/master/templates/_navbar_wagon_without_login.html.erb) in your `_navbar.html.erb` p

4 - Use our UI components

- Pick SASS partials from Le Wagon UI components (http://lewagon.github.io/ui-components/)
- Add them in `stylesheets/components`

5 - `content_for` & title

```
<!-- app/views/layouts/application.html.erb -->
<head>
  <title><%= yield(:title_tag).blank? ? "My Blog" : yield(:title_tag) %></title>

  <!-- [...] -->
</head>
```

```erb
<!-- app/views/articles/show.html.erb -->
<% content_for(:title_tag) do %>
  Read <%= @article.title %> on My Blog
<% end %>

<h1><%= @article.title %></h1>
```

6 - `content_for` & description

```erb
<!-- app/views/layouts/application.html.erb -->
<head>
  <meta name="description"
        content="<%= (yield(:description) || "").squish %>">

  <!-- [...] -->
</head>
```

```erb
<!-- app/views/articles/show.html.erb -->
<% content_for(:description) do %>
  <%= @article.meta_description %>
<% end %>

<h1><%= @article.title %></h1>
```

7 - `content_for` & js

What if you need to interpolate ruby in your JS ?

```erb
<!-- app/views/layouts/application.html.erb -->
<body>
  <!-- [...] -->

  <%= javascript_include_tag 'application' %>
  <%= yield(:js) %>
</body>
```

```erb
<!-- app/views/controller/action.html.erb -->
<!-- [...] -->

<% content_for(:js) do %>
  <%= javascript_tag do %>
    $(function(){
      $(".target").text("<%= some_ruby %>");
    });
  <% end %>
<% end %>
```

8 - Use different layouts

Ex: We want a specific layout for `articles#show` .

```ruby
class ArticlesController < ApplicationController
  layout "blog", only: [ :show ]

  # ...
end
```

Action `#show` in `ArticlesController` will use `app/views/layouts/blog.html.erb` layout.

9 - Pass locals to your partial

```erb
<!-- app/views/articles/index.html.erb -->
<% @articles.each do |a| %>
  <%= render "article", article: a %>
<% end %>
```

```erb
<!-- app/views/articles/_article.html.erb -->
<div class="card">
  <div class="card-header">
    <%= image_tag article.image_url %>
  </div>
  <div class="card-body">
    <h2><%= article.title %></h2>
    <p class="article-content"><%= article.body %></p>
  </div>
</div>
```

Image Upload

User-side upload. Not **assets**.

Uploading / Storing to Heroku?

You *can't*: the dyno file system is **ephemeral** (https://devcenter.heroku.com/articles/dynos#ephemeral-filesystem).

We need an external service.

Cloudinary...

Let's start

Create a new app with Le Wagon Minimal template (https://github.com/lewagon/rails-templates)

Security

(What the template did)

```
# Gemfile
gem 'figaro'
```

```
bundle install
bundle binstubs figaro
figaro install
spring stop
```

```
git status # Should ignore config/application.yml — Do not add it!
git add .
git commit —m "Add figaro — Protect my secret data in config/application.yml"
```

Create a Cloudinary account

Sign up (https://cloudinary.com/users/register/free) for a free account

⚠️ You may need to sign up using **tethering** as they might detect a "sign up spam".

Cloudinary & Environment

```
# Gemfile
gem 'cloudinary'
```

```
bundle install
```

```
# config/application.yml
CLOUDINARY_URL: "cloudinary://298522699261255:Qa1Zf04syfbOC-**********************8"
```

Let's upload one picture

```
curl http://lorempixel.com/800/600/city/9/ > san_francisco.jpg
rails c
irb> Cloudinary::Uploader.upload("san_francisco.jpg")
irb> exit
rm san_francisco.jpg
```

And then go to cloudinary.com/console/media_library (https://cloudinary.com/console/media_library)

Let's display it

```
<!-- app/views/pages/home.html.erb -->
<%= cl_image_tag("THE_IMAGE_ID_FROM_LIBRARY",
      width: 400, height: 300, crop: :fill) %>

<!-- for face detection -->
<%= cl_image_tag("IMAGE_WITH_FACE_ID",
      width: 400, height: 300, crop: :thumb, gravity: :face) %>
```

Crop modes (http://cloudinary.com/documentation/image_transformations#crop_modes): `scale`, `fit`, `fill`, `limit`, `pad`, `crop`.

You even have `thumb` with Face detection (http://cloudinary.com/documentation/image_transformations#face_detection)

Transformation reference

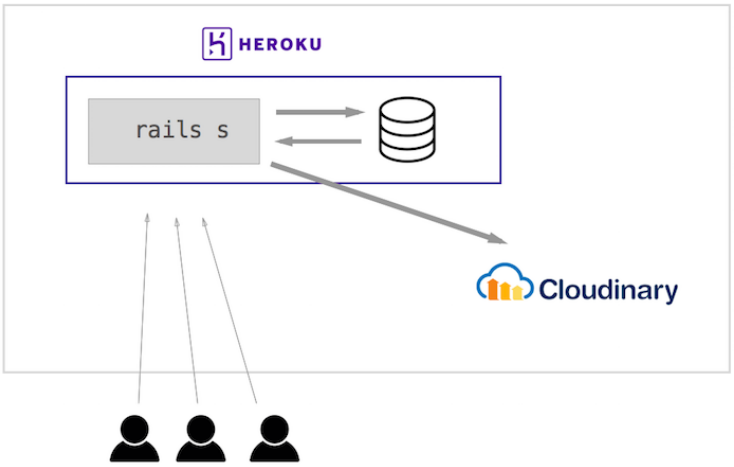
This exhaustive documentation (http://cloudinary.com/documentation/image_transformations#transformations_reference) lists all you can do.

You can change the **quality**, put some **effects** (like Instagram), add an overlay (watermark)

You can upload also other file formats (PDFs (http://cloudinary.com/blog/uploading_converting_and_generating_thumbnails_for_pdf_documents), etc.)

Carrierwave

It's a gem (https://github.com/carrierwaveuploader/carrierwave) to upload files associated to **Models**.




```
# Gemfile
gem 'carrierwave', '~> 0.11.2'
```

```
bundle install
```

Product

Let's say we have a `Product` model like this:

```
rails g scaffold Product name description:text
rails db:migrate
```

Creating an uploader for photos

```
rails g uploader Photo
spring stop # if the new `app/uploaders` folder is not picked up.
```

Open the created uploader and **simplify it** to:

```
# app/uploaders/photo_uploader.rb
class PhotoUploader < CarrierWave::Uploader::Base
  include Cloudinary::CarrierWave
end
```

Mount it in model

```
rails g migration AddPhotoToProduct photo:string
rails db:migrate
```

```
# app/models/product.rb
class Product < ApplicationRecord
  mount_uploader :photo, PhotoUploader
end
```

Form

Simple form version:

```
<!-- app/views/products/_form.html.erb -->
<%= simple_form_for(@product) do |f| %>
  <%= f.input :name %>
  <%= f.input :description %>
  <%= f.input :photo %>
  <%= f.input :photo_cache, as: :hidden %>
  <%= f.submit %>
<% end %>
```

With a regular rails form, use `f.file_field :photo` and `f.hidden_field :photo_cache` .

Strong params

```
# app/controllers/products_controller.rb
def product_params
  params.require(:product).permit(:name, :description, :photo, :photo_cache)
end
```

View (Creating an `<img />`)

```
<!-- app/views/products/show.html.erb -->
<%= cl_image_tag @product.photo, height: 300, width: 400, crop: :fill %>
```

Usage in `background-image`

Let's add a dynamic background image to our Card component (http://lewagon.github.io/ui-components/#card)

```
<div class="card"
  style="background-image: linear-gradient(rgba(0,0,0,0.3), rgba(0,0,0,0.2)),
         url('<%= cl_image_path @product.photo, height: 300, width: 400, crop: :fill %>')">
  <div class="card-category">Popular</div>
  <div class="card-description">
    <h2>Some stuff</h2>
    <p>Very cool stuff, useful and smart</p>
  </div>
  <a class="card-link" href="#" ></a>
</div>
```

Uploader with versions

```
# app/uploaders/photo_uploader.rb
class PhotoUploader < CarrierWave::Uploader::Base
  include Cloudinary::CarrierWave

  process eager: true  # Force version generation at upload time.

  process convert: 'jpg'

  version :standard do
    resize_to_fit 800, 600
  end

  version :bright_face do
    cloudinary_transformation effect: "brightness:30", radius: 20,
      width: 150, height: 150, crop: :thumb, gravity: :face
  end
end
```

Show view with versions

```
<!-- app/views/products/show.html.erb -->
<%= image_tag @product.photo.url(:bright_face) %>
```

Seed

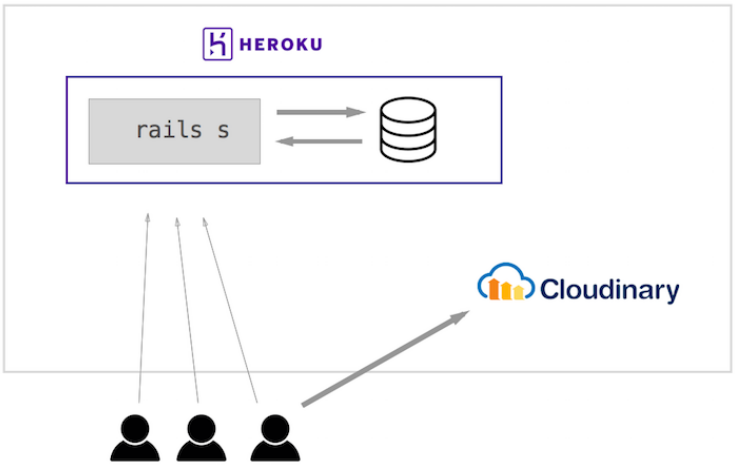
You can upload from a URL (https://github.com/carrierwaveuploader/carrierwave/wiki/How-to:-Upload-remote-image-urls-to-your-seedfile).

```
url = "http://static.giantbomb.com/uploads/original/9/99864/2419866-nes_console_set.png"
product = Product.new(name: 'NES')
product.remote_photo_url = url
product.save
```

(`remote_photo_url` because it's `mount_uploader :photo`)

Attachinary

Alternative gem (https://github.com/assembler/attachinary) to Carrierwave



Config (1)

```
# Gemfile
gem 'cloudinary', '1.1.7'
gem 'attachinary', github: 'assembler/attachinary'
gem 'jquery-fileupload-rails'
gem 'coffee-rails'
```

```
bundle update cloudinary
rails attachinary:install:migrations
rails db:migrate
```

```
# config/application.rb
# [...]
require "sprockets/railtie"
require "attachinary/orm/active_record"  # <= Add this line
```

```
# config/routes.rb
mount Attachinary::Engine => "/attachinary"
```

Config (2)

```
<!-- app/views/layouts/application.html.erb -->
<%= cloudinary_js_config %> <!-- Append this after the main `javascript_include_tag`. -->
```

```
// app/assets/javascripts/application.js

//= require jquery-fileupload/basic
//= require cloudinary/jquery.cloudinary
//= require attachinary
```

```
// app/assets/javascripts/init_attachinary.js
$(document).ready(function() {
  $('.attachinary-input').attachinary();
});
```

Usage - One attachment

```ruby
# app/models/product.rb
class Product < ApplicationRecord
  has_attachment :photo
end
```

```ruby
# app/controllers/products_controller.rb
def product_params
  params.require(:product).permit(:name, :description, :photo)
end
```

Simple form:

```erb
<!-- app/views/products/_form.html.erb -->
<%= f.input :photo, as: :attachinary %>
```

View (Creating an `<img />`)

```erb
<!-- app/views/products/show.html.erb -->
<% if @product.photo? %>
  <%= cl_image_tag @product.photo.path, width: 200, height: 300, crop: :fill %>
<% end %>
```

Usage in `background-image`

Again, let's add a dynamic background image to our Card component (http://lewagon.github.io/ui-components/#card)

```erb
<div class="card"
  style="background-image: linear-gradient(rgba(0,0,0,0.3), rgba(0,0,0,0.2)),
         url('<%= cl_image_path @product.photo.path, height: 300, width: 400, crop: :fill %>')">
  <div class="card-category">Popular</div>
  <div class="card-description">
    <h2>Some stuff</h2>
    <p>Very cool stuff, useful and smart</p>
  </div>
  <a class="card-link" href="#" ></a>
</div>
```

Usage - Multiple attachments

```ruby
class Product < ApplicationRecord
  has_attachments :photos, maximum: 2
end
```

```ruby
# app/controllers/products_controller.rb
def product_params
  params.require(:product).permit(:name, :description, photos: [])
end
```

Simple form:

```erb
<!-- app/views/products/_form.html.erb -->
<%= f.input :photos, as: :attachinary %>
```

```erb
<!-- app/views/productsshow.html.erb -->
<% @product.photos.each do |photo| %>
  <%= cl_image_tag photo.path, width: 300, height: 200, crop: :fill %>
<% end %>
```

Seed

You can upload a picture from a remote URL:

```ruby
# has_attachement :photo
url = "http://img.clubic.com/07791435-photo-playstation.jpg"
product = Product.new(name: 'Playstation')
product.save!
product.photo_url = url # Upload happens here
```

```ruby
# has_attachement :photos  (many)
urls = [
  'http://img.clubic.com/08254724-photo-xbox-console.jpg',
  'http://compass.xbox.com/assets/a5/d3/a5d3e0e4-38fd-42ab-90f4-e7b5112af4d1.png'
]

product = Product.new(name: 'Xbox')
product.save!
product.photo_urls = urls # Multi-upload happens here
```

Production

Make sure to push your `CLOUDINARY_URL` env variable to Heroku:

```
figaro heroku:set -e production
```

Check with:

```
heroku config
```

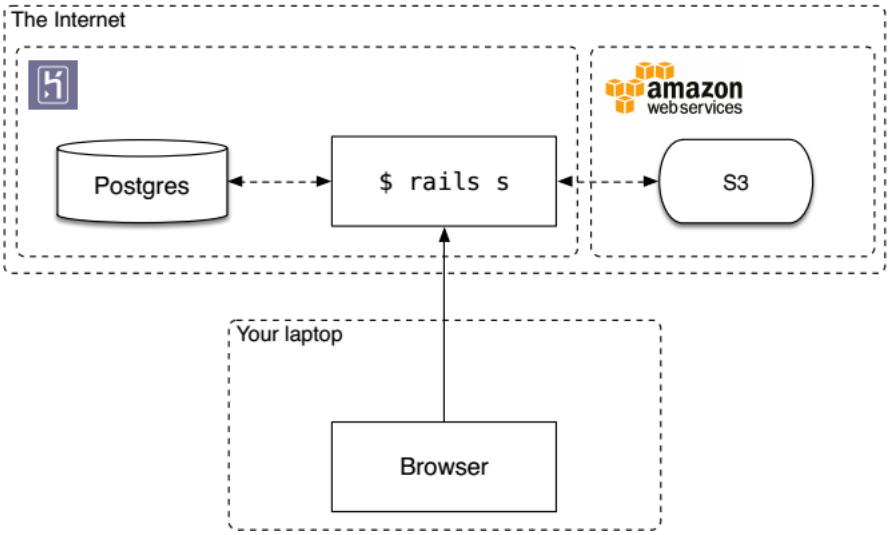
AWS S3

Alternative to Cloudinary

(No live-code, just a reference if you need it)

What is S3?

S3, is a “highly durable and available **store**” and can be used to reliably store application content such as media files, static assets and **user uploads**.

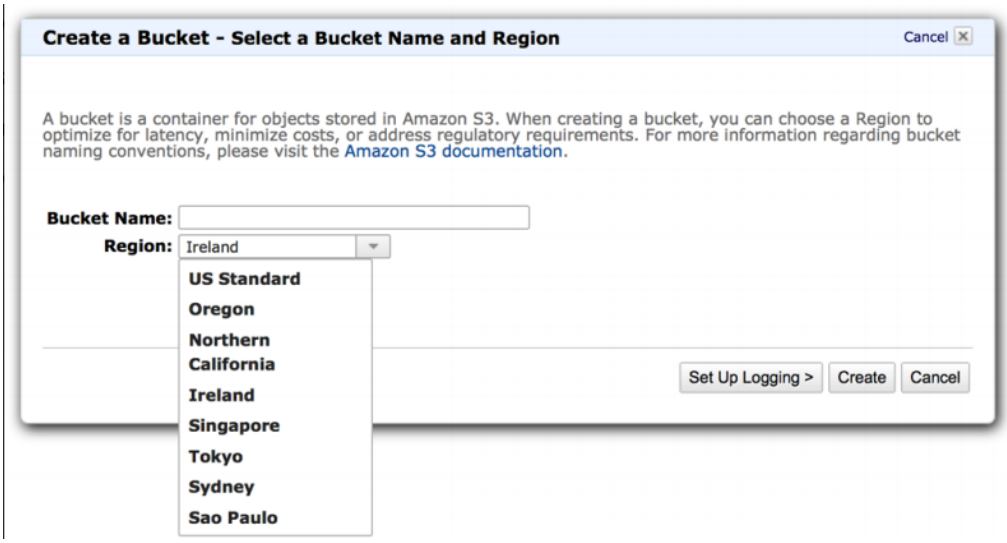


Sign up

https://aws.amazon.com/

You can sign up with your **existing** Amazon account

Buckets



Upload

or

Cyberduck

By default, uploaded files are **private**.

Security

This is **serious**. You don't want to lose thousands of dollars overnight (http://www.devfactor.net/2014/12/30/2375-amazon-mistake/).

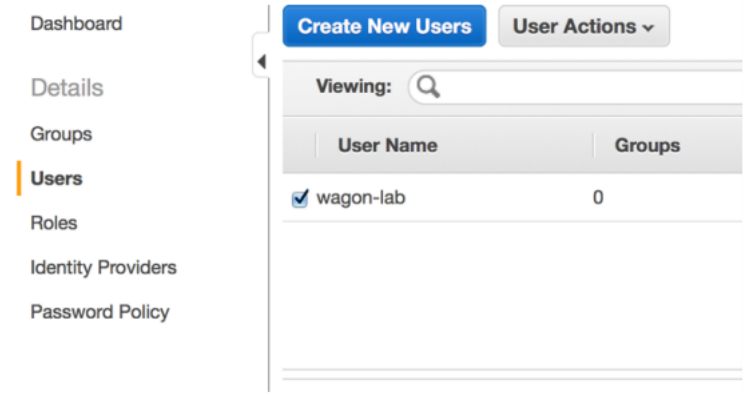
So pay attention.

Security (1)

Go to AWS Identity & Access Management
(IAM, https://aws.amazon.com/iam/)

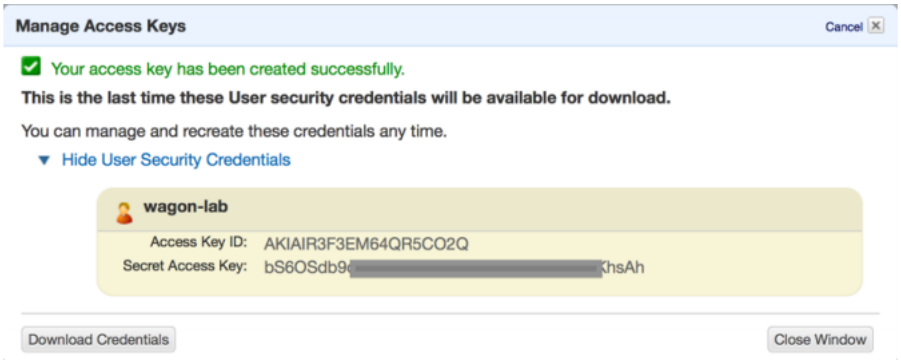
Security (2)

For each rails app, create a new user (same name as rails bucket). Download the credentials file to your computer.



Security (3)

Click on "Download Credentials" for the generated user.



Security (4)

Click on the created user. In Permissions, create a new **inline policy**.

Choose **Custom Policy** on the next screen. Copy paste this json (do not forget to replace the string `REPLACE_THIS_WITH_YOUR_BUCKET_NAME`)

```
{
  "Version": "2012-10-17", # Amazon policy service version, (given by Amazon, try to get the last one to use last features of Amazon policy)
  "Statement": [{
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "arn:aws:s3:::*"
    }, {
      "Action": "s3:*",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::REPLACE_THIS_WITH_YOUR_BUCKET_NAME",
        "arn:aws:s3:::REPLACE_THIS_WITH_YOUR_BUCKET_NAME/*"
      ]
  }]
}
```

Security (5)

Make sure that figaro (https://github.com/laserlemon/figaro) is setup and `config/application.yml` is in the `.gitignore` .

Security (6)

Paste these infos in your config/application.yml

```
# Change those values with yours!
AWS_REGION: "eu-west-1"  # If your bucket is created in Ireland Datacenter
AWS_S3_BUCKET_NAME: "your-bucket-name"
AWS_ACCESS_KEY_ID: "AK..............Q"
AWS_SECRET_ACCESS_KEY: "bS6OSdb9...hsAh"
```

Remember, this file **should not** be tracked by git!

Share this file with your teamates over a **secure** channel.

Push configuration to Heroku

```
# View current environment variables
heroku config
```

```
# Add environment variables from `config/application.yml`
figaro heroku:set -e production
```

```
# Check that again!
heroku config
```

Image Magick

Transformations of images will happen on your computer in dev, so you will need the ImageMagick:

```
# On MacOS X
brew install imagemagick
```

```
# On Ubuntu
sudo apt-get install imagemagick
```

Carrierwave with S3

```
# Gemfile
gem 'carrierwave'
gem 'mini_magick'
gem 'fog-aws'
```

```
bundle install
```

```
# config/initializers/carrierwave.rb
CarrierWave.configure do |config|
  config.fog_provider = 'fog/aws'
  config.fog_credentials = {
    :provider              => 'AWS',
    :aws_access_key_id     => ENV['AWS_ACCESS_KEY_ID'],
    :aws_secret_access_key => ENV['AWS_SECRET_ACCESS_KEY'],
    :region                => ENV['AWS_REGION']
  }

  config.fog_directory    = ENV['AWS_S3_BUCKET_NAME']
  config.fog_public       = false  # <- Private URLS (with expiration) are generated
end
```

```
# app/uploaders/photo_uploader.rb
class PhotoUploader < CarrierWave::Uploader::Base
  include CarrierWave::MiniMagick # <- Notice this

  storage :fog                    # <- and this

  version :standard do
    resize_to_fill 400, 300
  end

  version :thumb do
    resize_to_fill 100, 100
  end
end
```

Happy Uploading!

Refacto in Rails (slides only)

Partials

First usage: isolate code

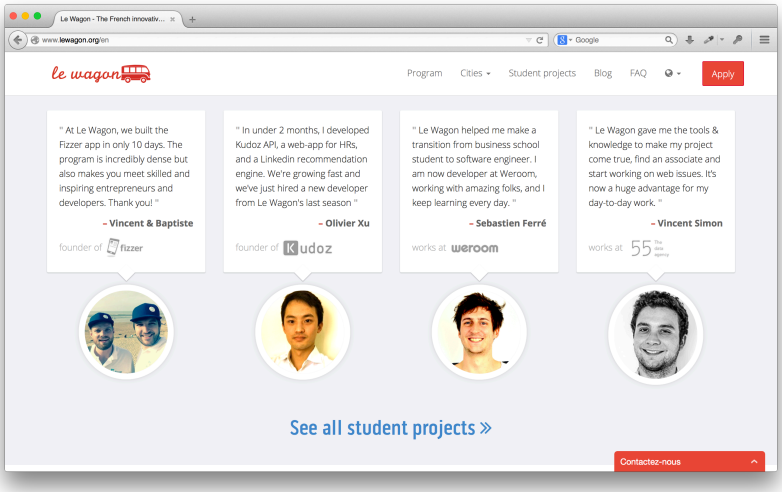
```
<!-- app/views/layouts/application.html.erb -->
<html>
  <head>
    <!-- [...] -->
  </head>
  <body>

    <%= render 'shared/navbar' %>

    <%= yield %>

  </body>
</html>
```

`render` calls the file `app/views/shared/_navbar.html.erb`

```
<!-- app/views/shared/_navbar.html.erb -->
<navbar>
  <!-- [...] Bootstrap Navbar, 50 lines [...] -->
</navbar>
```

You can also create a `app/views/shared/_footer.html.erb`

Second usage: DRY



```
<!-- app/views/pages/home.html.erb -->

<% @testimonials.each do |testimonial| %>
  <div>
    <%= testimonial.text %>
    <%= testimonial.author.name %>
    <%= image_tag testimonial.author.image %>
  </div>
<% end %>
```

What if I want to reuse these cards on another page?

Create the following partial:

```
<!-- app/views/shared/_testimonial.html.erb -->
<div>
  <%= testimonial.text %>
  <%= testimonial.author.name %>
  <%= image_tag testimonial.author.image %>
</div>
```

And use it:

```
<% @testimonials.each do |t| %>
  <%= render "shared/testimonial", testimonial: t %>
<% end %>
```

We passed **locals** to the partial.

Helpers

Sometimes you have complex ruby code in your views.

```
<!-- app/views/layouts/application.html.erb -->
<head>
  <title>
    <%= content_for?(:title) ? yield(:title) : "The default title for your site" %>
  </title>
</head>
```

We want to do something simpler:

```
<!-- app/views/layouts/application.html.erb -->
<head>
  <title>
    <%= yield_with_default(:title, "The default title for your site") %>
  </title>
</head>
```

Helper creation

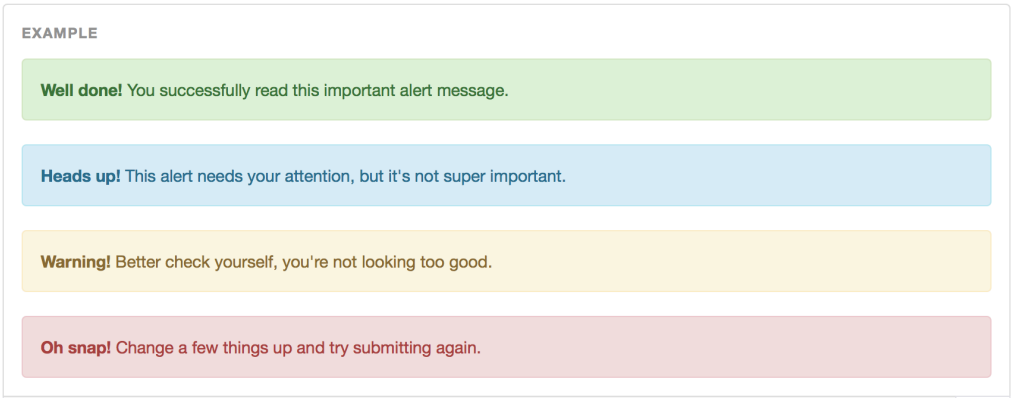
```
# app/helpers/application_helper.rb
module ApplicationHelper
  def yield_with_default(holder, default)
    if content_for?(holder)
      content_for(holder).squish
    else
      default
    end
  end
end
```

Other example - the `flash`

```
class RestaurantsController < ApplicationController
  def create
    @restaurant = Restaurant.create(restaurant_params)
    flash[:notice] = "Restaurant #{@restaurant.name} has been created"
    redirect_to restaurent_path(@restaurant)
  end
end
```

Bootstrap alert component

getbootstrap.com/components/#alerts (http://getbootstrap.com/components/#alerts)




```
<div class="alert alert-success" role="alert">...</div>
<div class="alert alert-info" role="alert">...</div>
<div class="alert alert-warning" role="alert">...</div>
<div class="alert alert-danger" role="alert">...</div>
```

Copy

In the **layout**, we need to render the flash:

```
<body>
  <% if flash[:notice] %>
    <div class="alert alert-info fade in">
      <button class="close" data-dismiss="alert">×</button>
      <%= flash[:notice] %>
    </div>
  <% end %>

  <% if flash[:alert] %>
    <div class="alert alert-error fade in">
      <button class="close" data-dismiss="alert">×</button>
      <%= flash[:alert] %>
    </div>
  <% end %>

  <%= yield %>
</body>
```

```
# app/models/application_helper.rb
module ApplicationHelper
  def bootstrap_class_for(flash_type)
    case flash_type
    when :alert   then "alert-error"
    when :notice  then "alert-success"
    end
  end
end
```

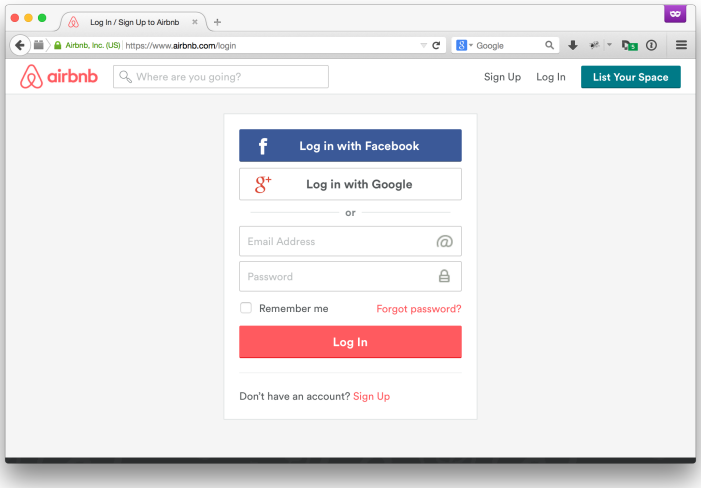
```
<!-- app/views/layouts/application.html.erb -->
<body>
  <% flash.select { |type| %i(alert notice).include?(type) } .each do |type, message| %>
    <div class="alert <%= bootstrap_class_for(type) %> fade in">
      <button class="close" data-dismiss="alert">×</button>
      <%= message %>
    </div>
  <% end %>

  <%= yield %>
</body>
```

We should even push the view code to a **partial**.

```
<!-- app/views/layouts/application.html.erb -->
<body>
  <%= render 'shared/flash' %>
  <%= yield %>
</body>
```

```
<!-- app/views/shared/_flash.html.erb -->
<% flash.select { |type| %i(alert notice).include?(type) } .each do |type, message| %>
  <div class="alert <%= bootstrap_class_for(type) %> fade in">
    <button class="close" data-dismiss="alert">×</button>
    <%= message %>
  </div>
<% end %>
```

Authentication



What we'll use

Devise An authentication gem for Rails

https://github.com/plataformatec/devise/

Start with a rails minimal template

github.com/lewagon/rails-templates#minimal (https://github.com/lewagon/rails-templates#minimal)

How-to

Installation

```
# Gemfile
gem 'devise'
```

```
bundle install
rails generate devise:install
```

Configuration

```
Some setup you must do manually if you haven't yet:

  1. Ensure you have defined default url options in your environments files. Here
     is an example of default_url_options appropriate for a development environment
     in config/environments/development.rb:

       config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }

     In production, :host should be set to the actual host of your application.

  2. Ensure you have defined root_url to *something* in your config/routes.rb.
     For example:

       root to: "home#index"

  3. Ensure you have flash messages in app/views/layouts/application.html.erb.
     For example:

       <p class="notice"><%= notice %></p>
       <p class="alert"><%= alert %></p>
```

Initializer

```
# config/initializers/devise.rb

# ==> Mailer Configuration
# Configure the e-mail address which will be shown in Devise::Mailer,
# note that it will be overwritten if you use your own mailer class
# with default "from" parameter.
config.mailer_sender = 'please-change-me-at-config-initializers-devise@example.com'
```

User Model

Generate the `User` model with the **devise** generator (not `model` nor `scaffold`).

```
rails generate devise User
#      invoke  active_record
#      create    db/migrate/TIMESTAMP_devise_create_users.rb
#      create    app/models/user.rb
#      insert    app/models/user.rb
#       route  devise_for :users
```

This **creates** the `User` model and its migration. So run:

```
rails db:migrate
```

Look! (1)

```
# app/models/user.rb

class User < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable
end
```

Look! (2)

```
rails routes
rails routes | grep new_
```

Let's sign up!

```
rails s
```

And go to localhost:3000/users/sign_up (http://localhost:3000/users/sign_up)

```
rails c
[1] pry(main)> User.count
[2] pry(main)> User.last
```

Views

Helpers

```
user_signed_in?
# => true / false

current_user
# => User instance / nil
```

Navbar

You need a Log in / Logout logic.

You can start from this navbar template (https://github.com/lewagon/awesome-navbars/blob/master/templates/_navbar_wagon.html.erb)

```
mkdir app/views/shared
curl https://raw.githubusercontent.com/lewagon/awesome-navbars/master/templates/_navbar_wagon.html.erb > app/views/shared/_navbar.html.erb
curl https://raw.githubusercontent.com/lewagon/karr-images/master/white_logo_red_circle.png > app/assets/images/logo.png
```

```
<!-- app/views/layouts/application.html.erb -->
<%= render 'shared/navbar' %>
```

Alerts

```
<!-- app/views/layouts/application.html.erb -->
<%= render 'shared/flashes' %>
```

```
<!-- app/views/shared/_flashes.html.erb -->
<% if notice %>
  <div class="alert alert-info alert-dismissible" role="alert">
    <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button>
    <%= notice %>
  </div>
<% end %>
<% if alert %>
  <div class="alert alert-warning alert-dismissible" role="alert">
    <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button>
    <%= alert %>
  </div>
<% end %>
```

Customize devise views

Sign in / Sign up / Forget password / [...]

```
rails g devise:views
```

And look into `app/views/devise` folder.

Controller

White-list approach

Protect **every route** by default.

```
# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  protect_from_forgery
  before_action :authenticate_user!
end
```

Skipping login for some pages

```
# app/controllers/pages_controller.rb
class PagesController < ApplicationController
  skip_before_action :authenticate_user!, only: :home

  def home
  end
end
```

Adding attributes to `User`

What if I want to add a `first_name` and `last_name` at sign up?

```
class ApplicationController < ActionController::Base
  # [...]
  before_action :configure_permitted_parameters, if: :devise_controller?

  def configure_permitted_parameters
    # For additional fields in app/views/devise/registrations/new.html.erb
    devise_parameter_sanitizer.permit(:sign_up, keys: [:first_name, :last_name])

    # For additional in app/views/devise/registrations/edit.html.erb
    devise_parameter_sanitizer.permit(:account_update, keys: [:username])
  end
end
```

Read more in the documentation (https://github.com/plataformatec/devise#strong-parameters)

Happy Authenticating!

Coding as a team

Public project

Free on GitHub but be aware that the project will be **visible** to everyone.

Private project

GitHub **paid** plan (https://github.com/pricing). Project visible only to **repo collaborators**.

Cheapest plan, "*Personal*": $7/month, unlimited repos.

Alternatives (Free):

- BitBucket (https://bitbucket.org)
- GitLab (https://gitlab.com)

You can use multiple cloud hosting for your git repositories!

GitHub Organization

Example: github.com/lewagon (https://github.com/lewagon)

- GitHub - User and organization differences (https://help.github.com/articles/what-s-the-difference-between-user-and-organization-accounts)
- GitHub - Converting a user into an organization (https://help.github.com/articles/converting-a-user-into-an-organization)

Create a repo

Hacker's version

```
cd ~/code/YOUR_GITHUB_USERNAME
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/minimal.rb \
  rbnb
cd rbnb
```

Reminder: the minimal template **includes** the following steps (you **do not** need to run them):

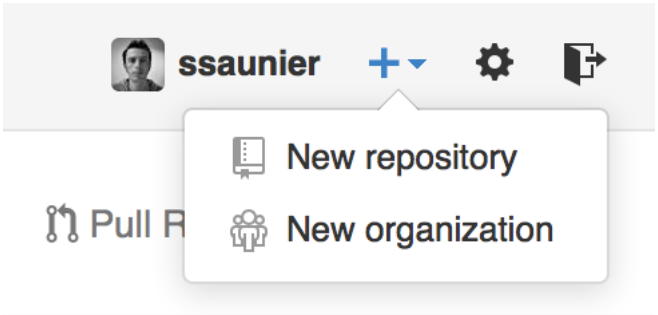
```
git init
git add .
git commit -m "Initial commit with minimal template from https://github.com/lewagon/rails-templates"
```

You still need to create your Github repo with:

```
hub create
```

Which creates the repo on Github and adds the `origin` remote to your local repo.

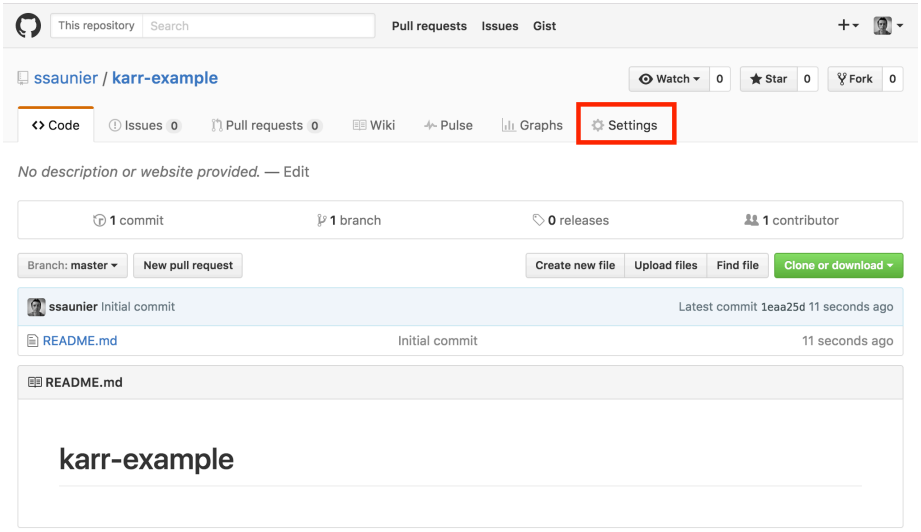
Alternative version

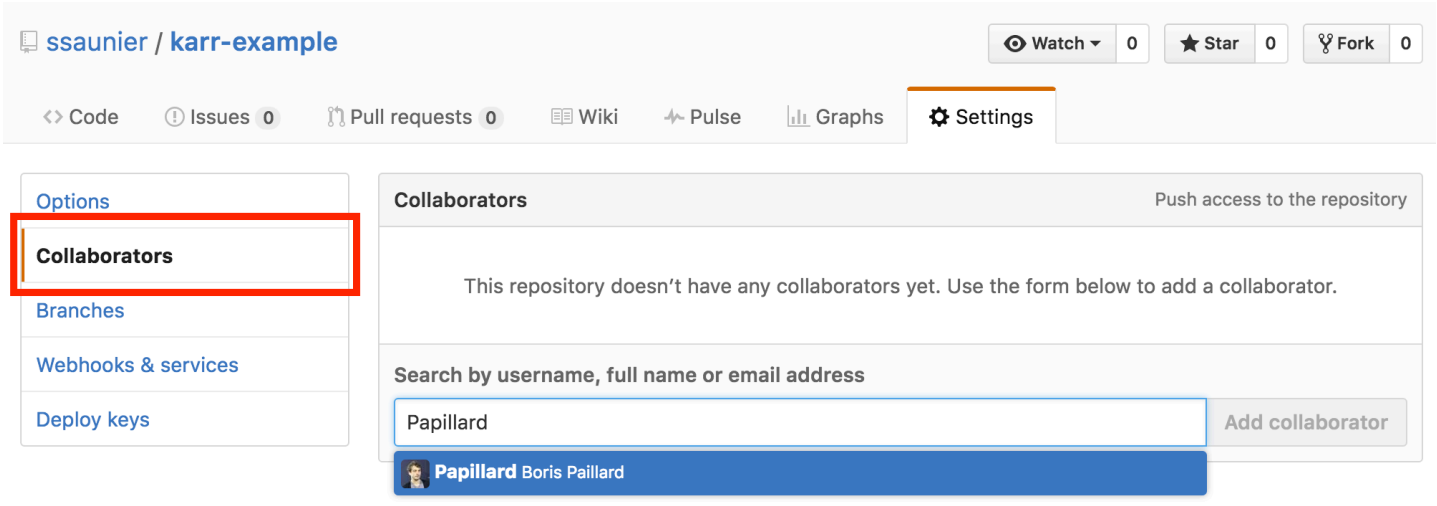


Then you need to add a remote:

```
git remote add origin git@github.com:ssaunier/karr-example.git
git push origin master
```

Adding collaborators





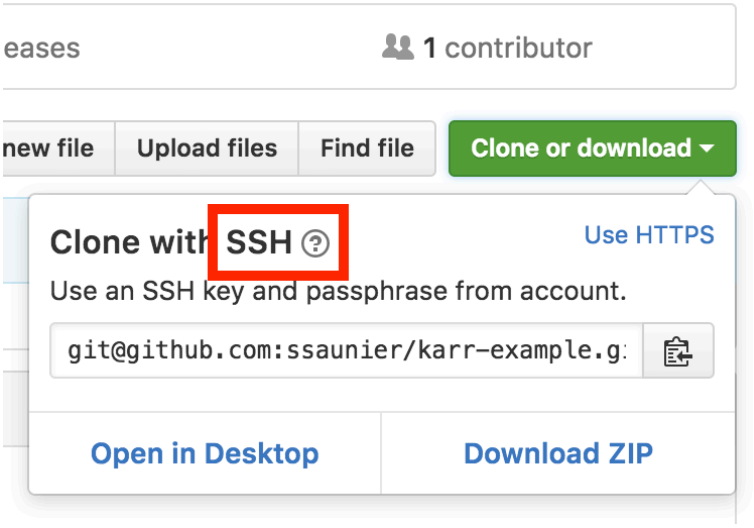
You gave them **push** access to the repository.

(and other (https://help.github.com/articles/permission-levels-for-a-user-account-repository/#collaborator-access-on-a-repository-owned-by-a-user-account) rights)

Now, collaborators can **clone** the project on their own computer:

```
mkdir ~/code/OWNER_GITHUB_USERNAME
cd ~/code/OWNER_GITHUB_USERNAME
git clone git@github.com:OWNER_GITHUB_USERNAME/PROJECT_NAME.git
cd PROJECT_NAME
```

Clone with **SSH** , **not** HTTPS.



Working as a team

Think of features (user stories)

```
As a <ROLE>
I can <ACTION>
So that <VALUE>
```

Problems

- Overlap (we both need to change the `RestaurantsController`)
- Dependency (I need your feature done to start mine)

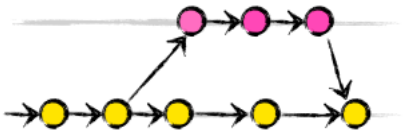
Solutions

- Technical (`git` branching)
- Human (communication)

Git Branching

When cloning a repository, you're by default on a branch, `master`.

One rule: one feature == one branch.



Listing local branches

```
git branch
```

Working on a new branch

```
git checkout -b update-call-to-action-color
git branch
```

We've created a new local branch called **update-call-to-action-color**.

Any new commit will only be applied to this branch.

Pushing a branch to GitHub

```
git branch -a
git push origin update-call-to-action-color
git branch -a
```

Finishing a feature

Using branches, we work on different parts of a project at the same time.

When a feature is finished, we'd like to **merge** commits back in `master`.

How?

GitHub Pull Requests

Usage

- Get feedback on code written in a given branch (code review)
- Merge the branch back into master

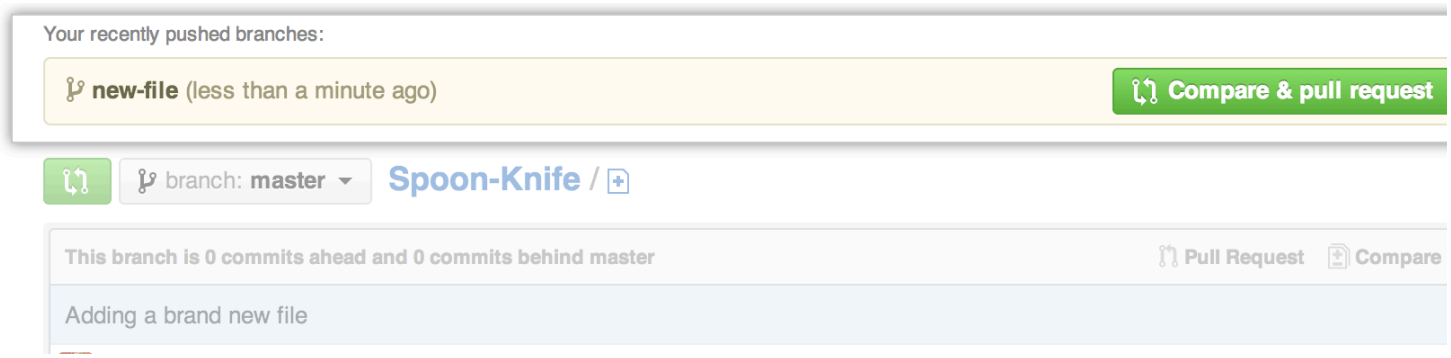
A Pull Request is a conversation

Example: lewagon/www#240 (https://github.com/lewagon/www/pull/240).

Creating a Pull Request (1)

As soon as you push a new branch, a pull request button appears on your GitHub repository page.

Just click on this button, review the diff and add clear title and description.

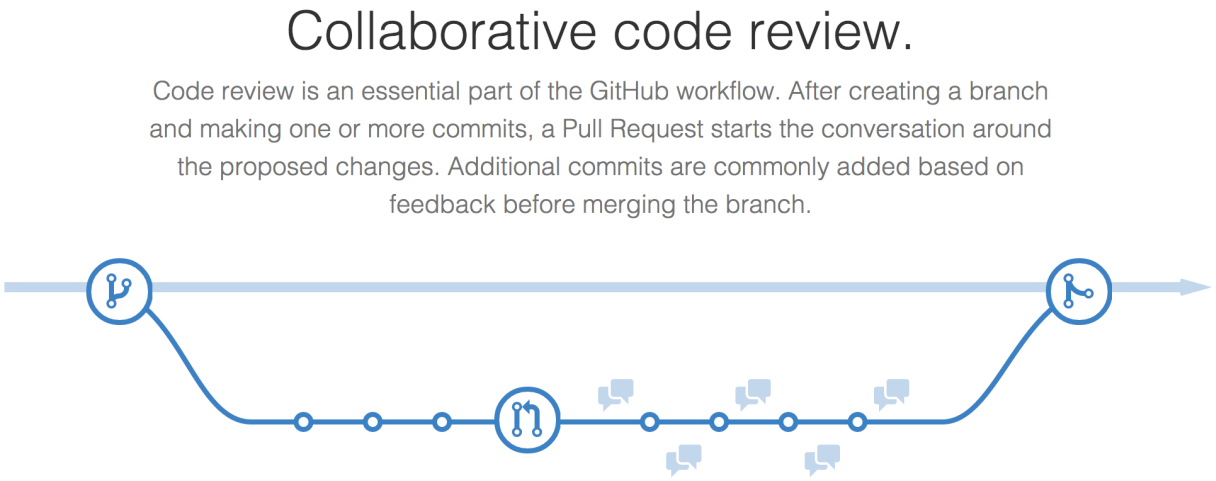


Creating a Pull Request (2)

- Take some time to write a proper **title** and **description**
- Explain what you did on the branch (gem added, code tricks, etc...)
- Ease the reviewer's job
- You can poke people with `@...` , like `@ssaunier` or `@papillard` to get their feedback

Reviewing a Pull Request

- Look at the diff, comment on errors (indentation, style, useless code, etc.)
- Comment **inline** or at the pull request level
- When done:
  - If code is fine, click on "Merge Pull Request" then "Delete Branch"
  - If not, add a general comment "Review done"



Looping over

Getting `master` up to date

When a Pull Request is **merged**, a new commit is created on `master` .

You need to fetch it on your **local** repository.

Be very careful

First, you need a **CLEAN** git status.

```
git status
# On branch master
# Your branch is up-to-date with 'origin/master'.
# nothing to commit, working directory clean
```

Get the latest `master`

```
git checkout master
git pull origin master
```

Then you can clean up local unused branches

```
git sweep
```

Merging `master` in your branches

Do you need something in `master` back into your current branch?

```
git status
# MAKE SURE THIS IS CLEAN
git checkout add-description-to-restaurant
git merge master
```

2 rules

- **Never** commit directly to `master`. Use feature branches
- **Always** make sure `git status` is **clean** before `pull`, `checkout` or `merge`.

In case of conflict

Sometimes a Pull Request won't be mergeable.

Why? `master` changed between the time you created the branch and now.

```
git status # ⚠️ ⚠️ ⚠️ Make sure it's clean before proceeding
git checkout master
git pull origin master          # pull the latest changes
git checkout unmergeable-branch # switch back to your branch
git merge master                # merge the new changes from master into your branch

# 😱 Conflicts will appear. It's normal!
# 🤞 Open sublime and solve conflicts (locate them with cmd + shift + f `<<<<<`)
# When solved, we need to finish the merge

git add .                           # add the files in conflict
git commit --no-edit                # commit using the default commit message
git push origin unmergeable-branch  # push our branch again
```

Project Management

Week objective: Implement your own version of AirBnB

Teams of 3 to 4

Day One

1. Write 5 to 10 user stories (week backlog) + Validate with teacher
2. Brainstorm Data Model + Validate with teacher
3. Brainstorm Routes / Draw Mockups + Validate with teacher
4. Lead Dev creates rails app, github repo, invite collaborators
5. Development starts. Pair program if too much dependencies at the beginning

Start with Pen & Paper! Print this (https://github.com/lewagon/fullstack-images/raw/master/rails/rails-user-stories.pdf) to help you.

DO NOT commit to `master`. Exception, the rails app creation:

```
cd ~/code/$GITHUB_USERNAME
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/minimal.rb \
  rbnb
cd rbnb
git init
git add .
git commit -m "Initial commit"
hub create
git push origin master
# And now, `checkout -b` a new branch BEFORE writing code.
```

Happy Collaborating!

Facebook Connect

Facebook Connect

- Based on OAuth (https://en.wikipedia.org/wiki/OAuth)
- Takes charge of the Facebook Connect process:

1. Sign in to Facebook
2. Authorize app

3. Callback

Start with a rails devise template

github.com/lewagon/rails-templates#devise (https://github.com/lewagon/rails-templates#devise)

Setup

- Devise already include an `omniauthable` extension.
- We assume that both **devise** and **figaro** gems are installed

```
# Gemfile
gem 'omniauth-facebook'
```

```
bundle install
```

Facebook App

Create the FB app

- Sign in on FB developer platform (https://developers.facebook.com)
- Create your app (specify a name)
  - In Settings, add a website platform with http://yourapp.herokuapp.com (or your domain)
- Create a test app (click on your app name on the top left corner)
  - In settings, set the website platform to `http://localhost:3000`

Integration in Rails

What follows is a summary of the devise omniauth wiki (https://github.com/plataformatec/devise/wiki/OmniAuth:-Overview).

Set your API keys

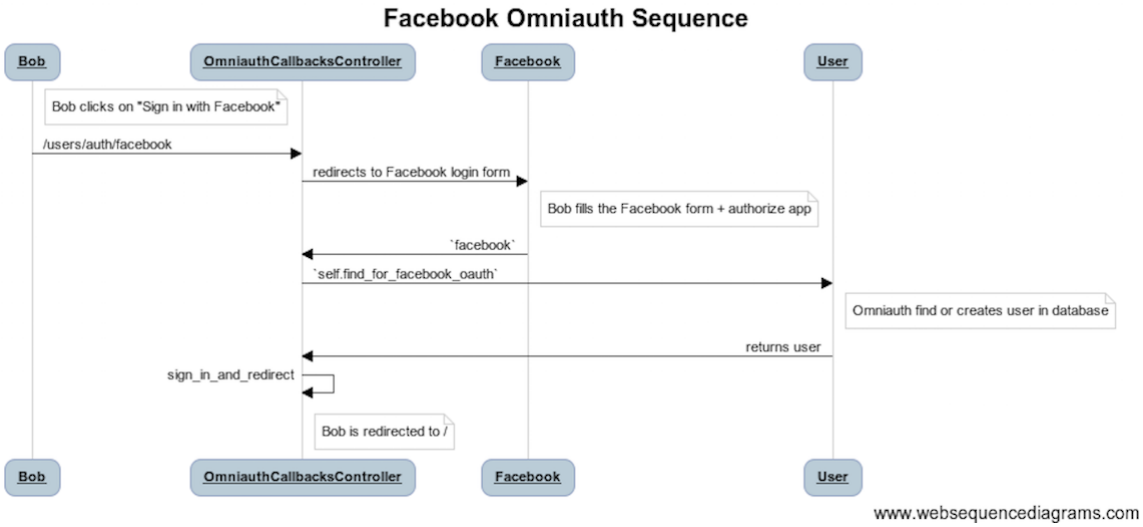
1. Remember, you need the `figaro` (https://github.com/laserlemon/figaro#getting-started) gem installed before, to have an ignored `config/application.yml` file (don't p
   your credentials to GitHub!)
2. Copy/paste your FB keys in `config/application.yml` as follows

```
development:
  FB_ID: "3************1"
  FB_SECRET: "4************e"

production:
  FB_ID: "3************2"
  FB_SECRET: "5************f"
```

Configure devise

```
# config/initializers/devise.rb

Devise.setup do |config|
  config.omniauth :facebook, ENV["FB_ID"], ENV["FB_SECRET"],
    scope: 'email',
    info_fields: 'email, first_name, last_name',
    image_size: 'square',  # 50x50, guaranteed ratio
    secure_image_url: true
end
```

See the omniauth config (https://github.com/mkdynamic/omniauth-facebook#configuring) doc for more.



Prepare to receive the FB callback

Replace your old devise route by the following one

```
# config/routes.rb

Rails.application.routes.draw do
  devise_for :users,
    controllers: { omniauth_callbacks: 'users/omniauth_callbacks' }
end
```

Make your `User` model omniauthable

```
# app/models/user.rb

class User < ApplicationRecord
  devise :omniauthable, omniauth_providers: [:facebook]
end
```

Add new attributes to your users

```
rails g migration AddOmniauthToUsers \
    provider uid facebook_picture_url first_name last_name token token_expiry:datetime
rails db:migrate
```

```
# app/models/user.rb
class User < ApplicationRecord
  # [...]
  def self.find_for_facebook_oauth(auth)
    user_params = auth.slice(:provider, :uid)
    user_params.merge! auth.info.slice(:email, :first_name, :last_name)
    user_params[:facebook_picture_url] = auth.info.image
    user_params[:token] = auth.credentials.token
    user_params[:token_expiry] = Time.at(auth.credentials.expires_at)
    user_params = user_params.to_h

    user = User.find_by(provider: auth.provider, uid: auth.uid)
    user ||= User.find_by(email: auth.info.email) # User did a regular sign up in the past.
    if user
      user.update(user_params)
    else
      user = User.new(user_params)
      user.password = Devise.friendly_token[0,20]  # Fake password for validation
      user.save
    end

    return user
  end
end
```

Callback Controller

Create a new controller to handle Omniauth callback requests

```
mkdir app/controllers/users
touch app/controllers/users/omniauth_callbacks_controller.rb
```

```ruby
# app/controllers/users/omniauth_callbacks_controller.rb

class Users::OmniauthCallbacksController < Devise::OmniauthCallbacksController
  def facebook
    user = User.find_for_facebook_oauth(request.env['omniauth.auth'])

    if user.persisted?
      sign_in_and_redirect user, event: :authentication
      set_flash_message(:notice, :success, kind: 'Facebook') if is_navigational_format?
    else
      session['devise.facebook_data'] = request.env['omniauth.auth']
      redirect_to new_user_registration_url
    end
  end
end
```

Display the picture in navbar

```erb
<% avatar_url = current_user.facebook_picture_url || "http://placehold.it/30x30" %>
<%= image_tag avatar_url, class: "avatar dropdown-toggle", id: "navbar-wagon-menu", "data-toggle" => "dropdown" %>
```

You should use a helper for that!

Keep a regular sign-up

User may disable access to his/her email => Devise can't register user!

Production

```
figaro heroku:set -e production
```

Happy Omniauth!

(There's Twitter, GitHub, Linkedin, etc.)

Geocoding

Geocoding

Address -> GPS Coordinates

```
"Eiffel Tower" → { lat: 48.8582, lng: 2.2945 }
```

Start with the Rails Devise template

github.com/lewagon/rails-templates#devise (https://github.com/lewagon/rails-templates#devise)

A new gem for your `Gemfile`

Let's add the geocoder (https://github.com/alexreisner/geocoder) gem.

```ruby
# Gemfile
gem "geocoder"
```

```
bundle install
rails generate geocoder:config
```

Some configuration first

Open `config/initializers/geocoder.rb` and set some values:

```ruby
Geocoder.configure(
  # [...]
  :units => :km,
  # [...]
)
```

Suppose we already have a `Flat`

```
rails g model Flat name address

# for teacher's demo only:
rails g scaffold Flat name address
```

The `address` is just a string, like `"16 Villa Gaudelet, Paris"`.

Add columns to your model

```
# app/models/flat.rb
class Flat < ApplicationRecord
end
```

Add **two columns** to your model you want to geocode:

(Replace `Flat` with the model you want to geocode!)

```
rails g migration AddCoordinatesTo*Flats* latitude:float longitude:float
```

```
rails db:migrate
```

Geocoding

Suppose that your store the address in a `address` column:

```
# app/models/flat.rb
class Flat < ApplicationRecord
  geocoded_by :address
  after_validation :geocode, if: :address_changed?
end
```

Result

```
rails c
```

```
Flat.create(address: '16 Villa Gaudelet, Paris', name: 'Le Wagon HQ')
# => #<Flat:0x007fad2e720898
#  [...]
#  name: "Le Wagon HQ",
#  latitude: 48.8648482,
#  longitude: 2.3798534,
#  address: "16 Villa Gaudelet, Paris">
```

Search

```
Flat.near('Tour Eiffel', 10)      # venues within 10 km of Tour Eiffel
Flat.near([40.71, 100.23], 20)    # venues within 10 km of a point
```

⚠️ This won't work in production, **yet**

(as API requests are **anonymous**. We'll cover API auth later)

Reminder on JavaScript

Before we try to display a map with markers of our flats.

Objective: we want to `alert` a dynamic message.

Problem: content in `app/assets/javascripts/` is **static**

```
// app/assets/javascripts/application.js

alert('Sadly this is static...');
```

```
# app/controllers/flats_controller.rb
class FlatsController < ApplicationController
  def show
    @flat = Flat.find(params[:id])
    @alert_message = "You are viewing #{@flat.name}"
  end
end
```

```
<!-- app/views/flats/show.html.erb -->

<%= javascript_tag do %>
  $(document).ready(function() {
    alert('<%= j @alert_message %>');
  });
<% end %>
```

The `j` is a shortcut for `escape_javascript` (http://api.rubyonrails.org/classes/ActionView/Helpers/JavaScriptHelper.html#method-i-escape_javascript). Escapes carriage retu
and single and double quotes for JavaScript segments.

Default rails layout + View is:

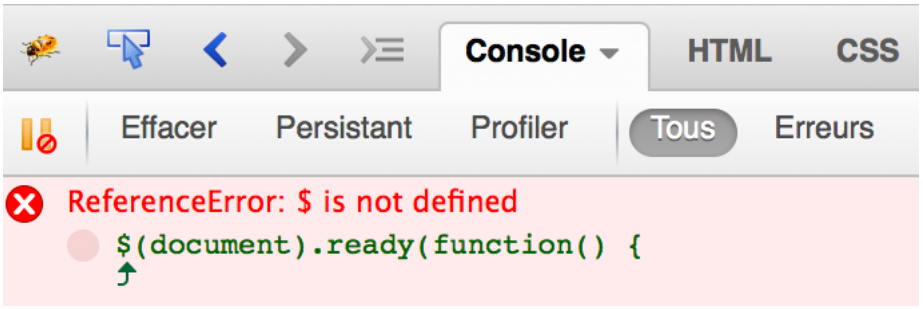
```
<!-- app/views/layouts/application.html.erb -->
<!DOCTYPE html>
<html>
  <head>
    <%= stylesheet_link_tag    'application', media: 'all' %>
    <%= javascript_include_tag 'application' %>
  </head>
  <body>

    <%= yield %>

  </body>
</html>
```

BUT, for **performance** reasons, `javascript_include_tag` should move to the bottom, just before `</body>`

```
<!-- app/views/layouts/application.html.erb -->
<!DOCTYPE html>
<html>
  <head>
    <%= stylesheet_link_tag    'application', media: 'all' %>
  </head>
  <body>

    <%= yield %>

    <%= javascript_include_tag 'application' %>
  </body>
</html>
```

New issue: if view contains some javascript, jQuery & libraries are not available.

Why? Appended at the `yield` , *before* `javascript_include_tag` .



Solution: Adding a `content_for`

```
<!-- app/views/layouts/application.html.erb -->
<!DOCTYPE html>
<html>
  <head>
    <%= stylesheet_link_tag    'application', media: 'all' %>
  </head>
  <body>

    <%= yield %>

    <%= javascript_include_tag 'application' %>
    <%= yield(:after_js) %>
  </body>
</html>
```

```
<!-- app/views/flats/show.html.erb -->

<% content_for(:after_js) do %>
  <%= javascript_tag do %>
    $(document).ready(function() {
      alert('<%= j @alert_message %>');
    });
  <% end %>
<% end %>
```

raw

```ruby
class FlatsController < ApplicationController
  def show
    @flat = Flat.find(params[:id])
    @flat_coordinates = { lat: @flat.latitude, lng: @flat.longitude }
  end
end
```

```erb
<% content_for(:after_js) do %>
  <%= javascript_tag do %>
    $(document).ready(function() {
      var coord = <%= raw @flat_coordinates.to_json %>;
      alert('Flat coordinates are ' + coord.lat + ', ' + coord.lng);
    });
  <% end %>
<% end %>
```

Google Maps (Static)

Objective: Display a google map with markers of flats

1. Demo on GitHub Pages (http://lewagon.github.io/google-maps-markers-static/)
2. Code on GitHub (https://github.com/lewagon/google-maps-markers-static/blob/gh-pages/index.html)

Google Maps (Dynamic)

Demo

1. Demo on Heroku (https://wagon-google-maps-markers.herokuapp.com)
2. Code on GitHub (https://github.com/lewagon/rails-google-maps)

Using the `gmaps4rails` gem

```ruby
# Gemfile
gem "coffee-rails"
gem "gmaps4rails"

source 'https://rails-assets.org' do
  gem "rails-assets-underscore"
end
```

```
bundle install
```

JavaScript dependencies

Add this after `jquery` and `bootstrap` :

```
// app/assets/javascripts/application.js

//= require underscore
//= require gmaps/google
```

Layout

```
<!-- app/views/layouts/application.html.erb -->

  <!-- [...] -->
  <%= javascript_include_tag "https://maps.google.com/maps/api/js" %>
  <%= javascript_include_tag "https://cdn.rawgit.com/printercu/google-maps-utility-library-v3-read-only/master/markerclusterer/src/markerclu
erer_compiled.js" %>
  <%= javascript_include_tag "https://cdn.rawgit.com/printercu/google-maps-utility-library-v3-read-only/master/infobox/src/infobox_packed.js
%>

  <%= javascript_include_tag "application" %>

  <%= yield(:after_js) %>
</body>
```

Make sure you only have one `javascript_include_tag "application"` (remove it from the `<head />` if there is still one)

Controller

You want to display a map on the `flats#index` route:

```
# app/controllers/flats_controller.rb

class FlatsController < ApplicationController
  def index
    @flats = Flat.where.not(latitude: nil, longitude: nil)

    @hash = Gmaps4rails.build_markers(@flats) do |flat, marker|
      marker.lat flat.latitude
      marker.lng flat.longitude
      # marker.infowindow render_to_string(partial: "/flats/map_box", locals: { flat: flat })
    end
  end
end
```

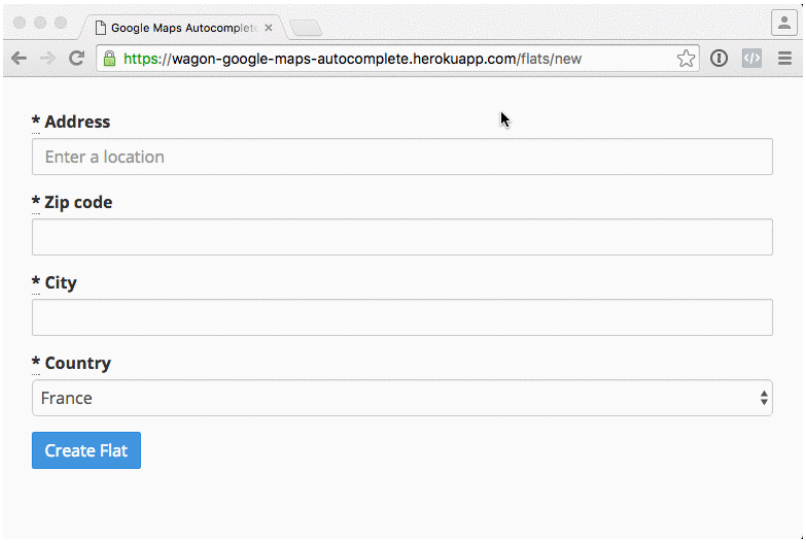
View

```
<!-- app/views/flats/index.html.erb -->

<div id="map" style="width: 100%; height: 600px;"></div>

<% content_for(:after_js) do %>
  <%= javascript_tag do %>
    $(document).ready(function() {
      var handler = Gmaps.build('Google');
      handler.buildMap({ internal: { id: 'map' } }, function() {
        markers = handler.addMarkers(<%= raw @hash.to_json %>);
        handler.bounds.extendWith(markers);
        handler.fitMapToBounds();
        if (markers.length == 0) {
          handler.getMap().setZoom(2);
        } else if (markers.length == 1) {
          handler.getMap().setZoom(14);
        }
      });
    });
  <% end %>
<% end %>
```

Documentation

For more, read Drawing on the maps (https://developers.google.com/maps/documentation/javascript/overlays)

Address autocomplete



1. Demo (https://wagon-google-maps-autocomplete.herokuapp.com)
2. Source code (https://github.com/lewagon/google-maps-autocomplete)

We won't dive into too much details, have a look at this sample code if you want to implement it.

Production

1 rails app == 1 new project

Go to console.developers.google.com (https://console.developers.google.com/apis/dashboard) to create a new project.

1. Config for `geocoder`

Geocoder uses the Google Geocoding API.

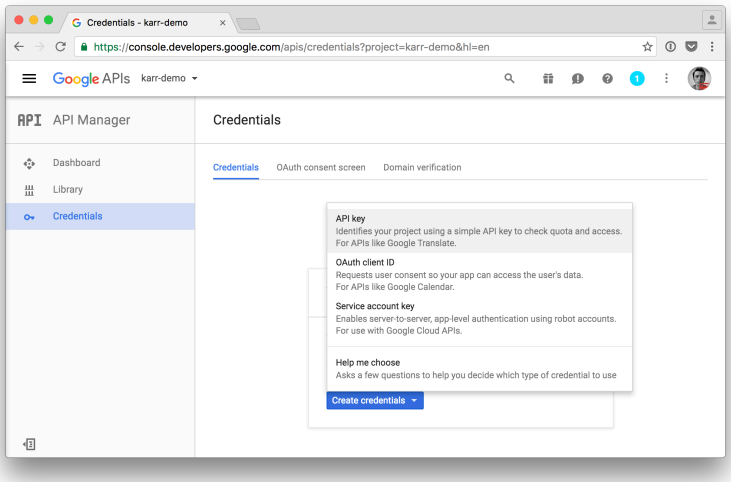
Limited number of requests per day (2500) in the free plan.

Go to **Library**, and look for:

```
Google Maps Geocoding API
```

Click on it and **Enable it** (top blue button)

Create a new API key



API key

AIzaSyCPnktscG3VLi50L53ccPG_Pu4GBc4Ww9A

Name

Server Key - Geocoding

Key restriction

This key is unrestricted. To prevent unauthorized use and quota theft, restrict your key.
Key restriction lets you specify which web sites, IP addresses, or apps can use this key. Learn more

- None
- HTTP referrers (web sites)
- IP addresses (web servers, cron jobs, etc.)
- Android apps
- iOS apps

Note: It may take up to 5 minutes for settings to take effect

Save Cancel

It is a **server** key, which belongs in Figaro's `config/application.yml`.

```
# config/application.yml
GOOGLE_API_SERVER_KEY: "AIzaSyDMg1*****************8wxJqE"
```

```
# config/initializers/geocoder.rb
Geocoder.configure(
  :lookup     => :google,
  :api_key    => ENV['GOOGLE_API_SERVER_KEY'],
  :use_https => true,
  # [...]
)
```

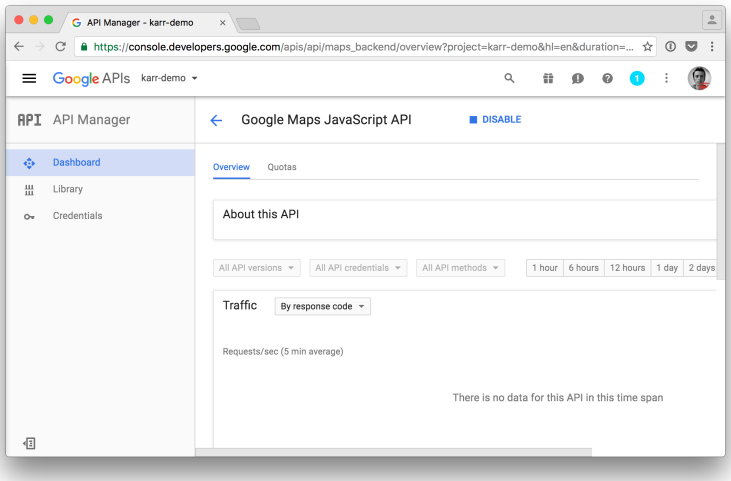
You should push this new key to Heroku:

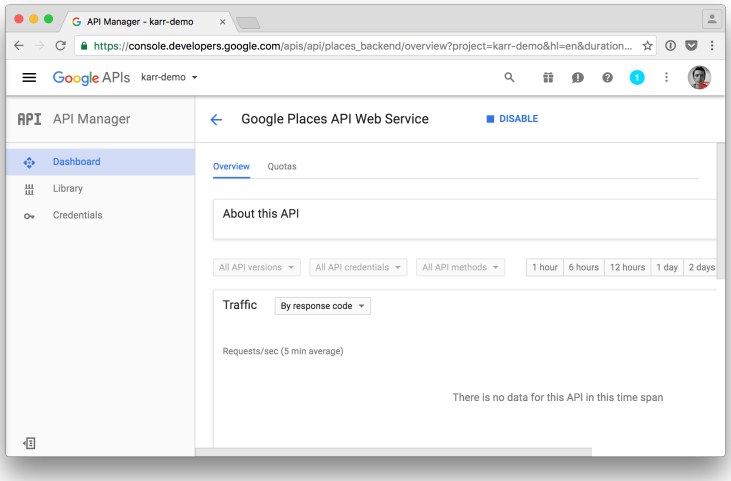
```
figaro heroku:set -e production
```

2. Config for Autocomplete & Google Maps

You must enable two APIs in the console:

1. `Google Maps JavaScript API` (free quota: 25,000 requests / day)
2. `Google Places API Web Service` (free quota: 1,000 requests / day)

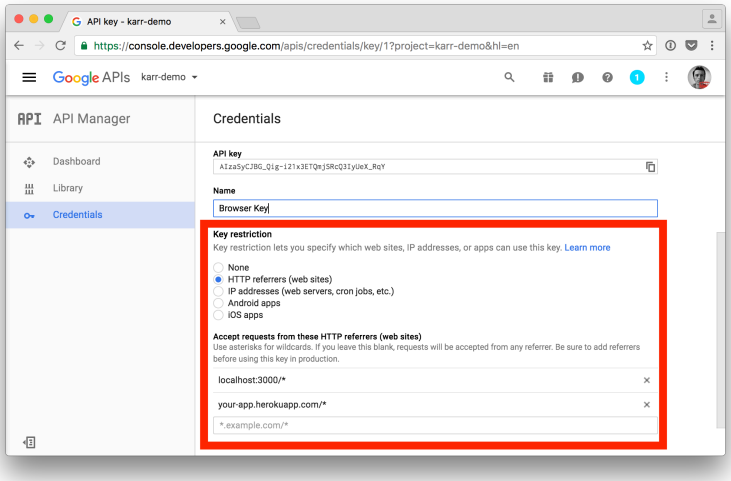




We must create another key, a **browser** key, which will be protected by **referrer**.

We need 2 referrers:

```
localhost:3000/*
your-app.herokuapp.com/*
```



Copy/paste your browser key in `config/application.yml` as follows

```
# config/application.yml
GOOGLE_API_BROWSER_KEY: "Sz1MDgyAaI******************EJ8xwq"
```

Update your javascript include tag:

```
<%= javascript_include_tag "https://maps.google.com/maps/api/js?libraries=places&key=#{ENV['GOOGLE_API_BROWSER_KEY']}" %>

<!-- sorry for the long line... -->
```

Push configuration to Heroku

```
figaro heroku:set -e production
```

Google Skins

Snazzy Maps (https://snazzymaps.com/)

Use the `provider` key when building the map.

```
var styles = [ /* the style copied from Snazzy maps */ ];

var handler = Gmaps.build('Google');
handler.buildMap({
  provider: {
    styles: styles
    // pass in other Google Maps API options here
    // => https://developers.google.com/maps/documentation/javascript/reference#MapOptions
  },
  internal: { id: 'map' } }, function() {
  // [ ... ]
});
```

Google Alternatives

Mapbox (https://www.mapbox.com/), uses OpenStreetmap (http://openstreetmap.fr/)

Happy geocoding!

Mailing

Transactional Email

SMTP

What is SMTP?

Simple Mail Transfer **Protocol**.

(HTTP is another protocol you know)

Start with a rails devise template

github.com/lewagon/rails-templates#devise (https://github.com/lewagon/rails-templates#devise)

We want to send a **welcome email** to our user.

ActionMailer (Rails)

Think of Controller + View

```
rails generate mailer UserMailer welcome
#      create  app/mailers/user_mailer.rb
#      invoke  erb
#      create    app/views/user_mailer
#      create    app/views/user_mailer/welcome.text.erb
#      create    app/views/user_mailer/welcome.html.erb
```

Customize Default Mailer

```
class ApplicationMailer < ActionMailer::Base
  default from: 'your-email@example.com'

  layout 'mailer'
end
```

Customize User Mailer

```
# app/mailers/user_mailer.rb

class UserMailer < ApplicationMailer
  def welcome(user)
    @user = user  # Instance variable => available in view

    mail(to: @user.email, subject: 'Welcome to Le Wagon')
    # This will render a view in `app/views/user_mailer`!
  end
end
```

Email content

To set the email content, you define it like views

```
<!-- app/views/user_mailer/welcome.text.erb -->

Hi <%= @user.name %>,

Today we'll talk about SMTP and Postmark!

See ya!
```

You can send HTML email if you have a view

`app/views/user_mailer/welcome.html.erb`

Testing

```
mkdir -p test/mailers/previews
```

```
# test/mailers/previews/user_mailer_preview.rb
class UserMailerPreview < ActionMailer::Preview
  def welcome
    user = User.first
    UserMailer.welcome(user)
  end
end
```

Go to localhost:3000/rails/mailers/user_mailer/welcome (http://localhost:3000/rails/mailers/user_mailer/welcome)

Development

We don't really send emails.

Letter Opener

```
# Gemfile

gem "letter_opener", group: :development
```

```
bundle install
```

```
# config/environments/development.rb
Rails.application.configure do
  # [...]
  config.action_mailer.delivery_method = :letter_opener
  config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
  # [...]
end
```

A new tab will open in the browser to show the mail you just sent.

Delivering the mail

```
user = User.first
UserMailer.welcome(user).deliver_now
```

From model (on callback)

```
class User < ApplicationRecord
  after_create :send_welcome_email

  private

  def send_welcome_email
    UserMailer.welcome(self).deliver_now
  end
end
```

Another example : Restaurant Mailer

```ruby
# app/mailers/restaurant_mailer.rb
class RestaurantMailer < ApplicationMailer
  def creation_confirmation(restaurant)
    @restaurant = restaurant

    mail(
      to:       @restaurant.user.email,
      subject:  "Restaurant #{@restaurant.name} created!"
    )
  end
end
```

From controller (on action)

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def create
    @restaurant = current_user.restaurants.build(restaurant_params)

    if @restaurant.save
      RestaurantMailer.creation_confirmation(@restaurant).deliver_now
      redirect_to restaurant_path(@restaurant)
    else
      render :new
    end
  end
end
```

Production

Case 1 - You don't own a domain.

You can use your Gmail address (not recommended).

Gmail Configuration (1)

First generate an App Password (https://support.google.com/mail/answer/185833?hl=en).

```ruby
# config/application.yml
GMAIL_ADDRESS: "your_address@gmail.com"
GMAIL_APP_PASSWORD: "abcdefghijklmnop"
```

```ruby
# config/initializers/smtp.rb
ActionMailer::Base.smtp_settings = {
  address: "smtp.gmail.com",
  port: 587,
  domain: 'gmail.com',
  user_name: ENV['GMAIL_ADDRESS'],
  password: ENV['GMAIL_APP_PASSWORD'],
  authentication: :login,
  enable_starttls_auto: true
}
```

Gmail Configuration (2)

```ruby
# config/environments/production.rb
Rails.application.configure do

  config.action_mailer.delivery_method = :smtp
  config.action_mailer.default_url_options = { host: "yourapp.herokuapp.com" }
  # or your custom domain name eg. "www.yourdomain.com"

end
```

Use `_url` path helpers in your email views!

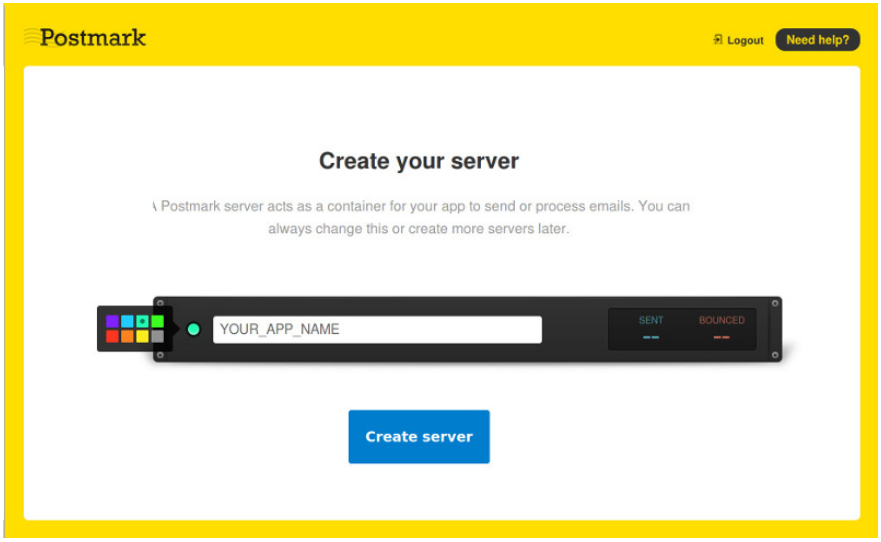
Update the development `delivery_method` to try it!

Case 2 - You own a domain.

Sign Up on Postmark (https://account.postmarkapp.com/sign_up)

Free trial: first 25,000 emails free

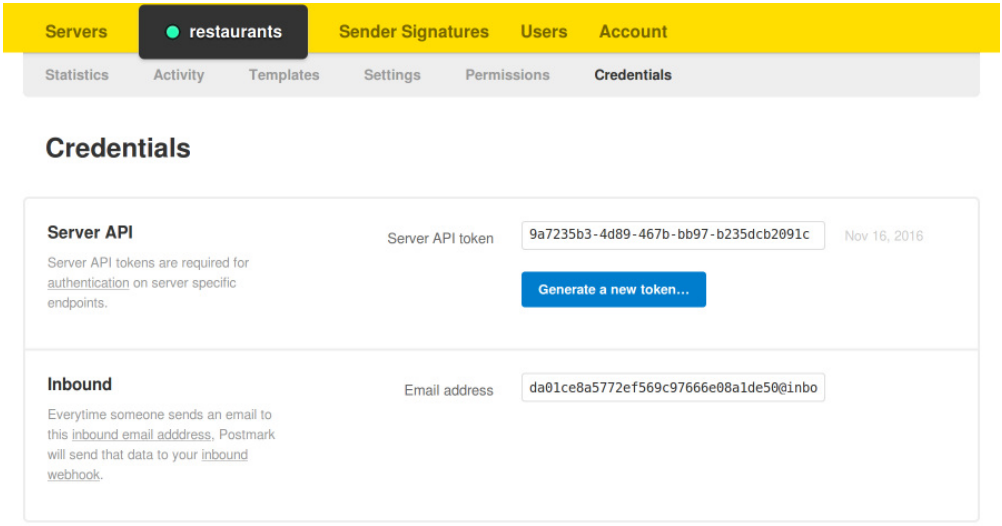
Create a Postmark mail server



Then configure your signature and **confirm** the mail!

Postmark Configuration (1)

Get the Postmark Server API key.



Postmark Configuration (2)

Push it to Heroku

```
heroku config:set POSTMARK_API_KEY="9a7235b3-**********************091c"
```

Add Postmark gem

```
# Gemfile
gem 'postmark-rails'
```

```
bundle install
```

Configure Postmark for production

```
# config/environments/production.rb
Rails.application.configure do
  # ...
  config.action_mailer.delivery_method     = :postmark
  config.action_mailer.postmark_settings   = { api_key: ENV['POSTMARK_API_KEY'] }
  config.action_mailer.default_url_options = { host: "yourapp.herokuapp.com" }
end
```

Use `_url` path helpers in your email views!

Deliverability from your domain

Add DKIM and SPF for your domain

Mail interceptor

Useful if you want to send real emails in `development` mode.

```
# lib/email_interceptor.rb
class EmailInterceptor
  def self.delivering_email(message)
    message.subject = "#{message.to} #{message.subject}"
    message.to = [ 'seb@lewagon.org' ]
  end
end
```

```
# config/initializers/development_email_interceptor.rb
if Rails.env.development?
  require "email_interceptor"
  ActionMailer::Base.register_interceptor(EmailInterceptor)
end
```

Documentation

Read more about Action Mailer on Rails Guides (http://guides.rubyonrails.org/action_mailer_basics.html)

Marketing Email

MailChimp

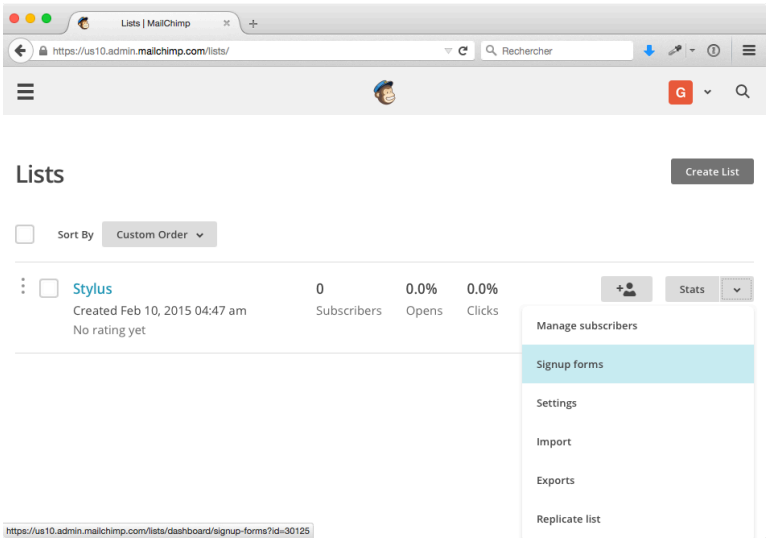
Create an account. Free for 2000 subscribers.

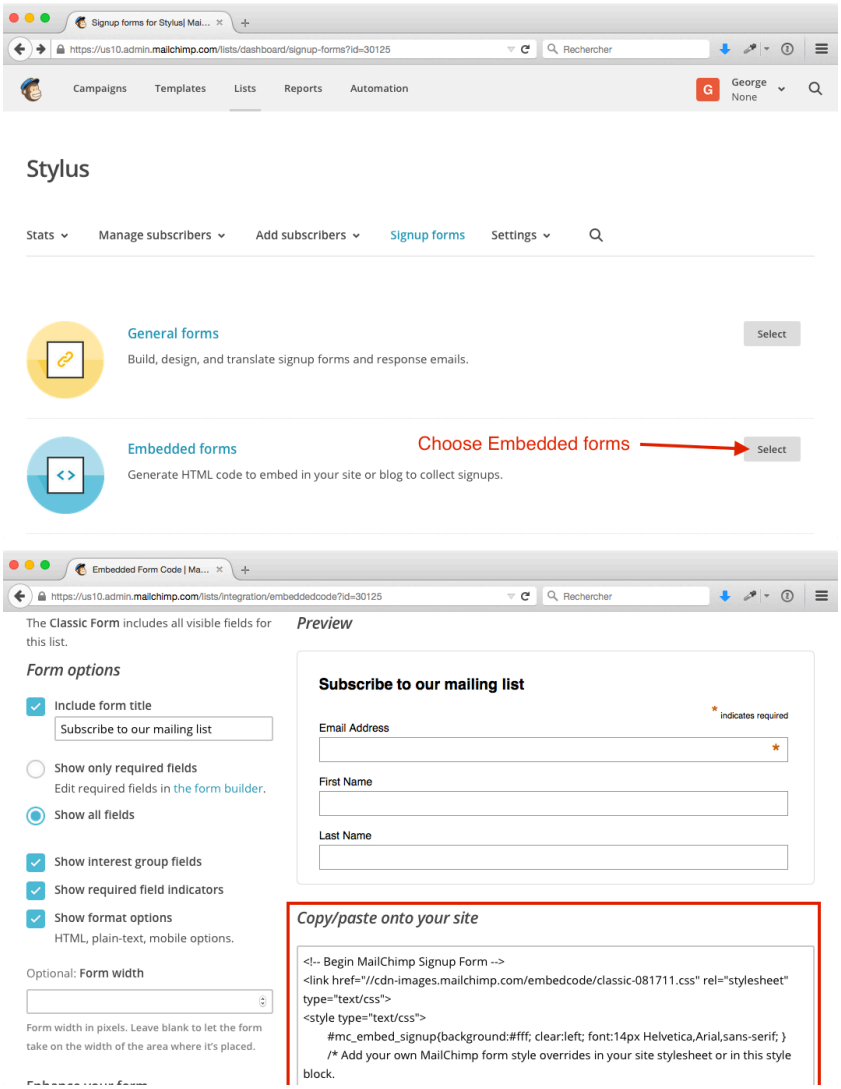
Alternatives: Mailjet, Campaign Monitor, InfusionSoft, etc.

Static website

For instance, one hosted on GitHub Pages

You need a client-side solution => HTML/JS





Dynamic website

For instance, a Rails app

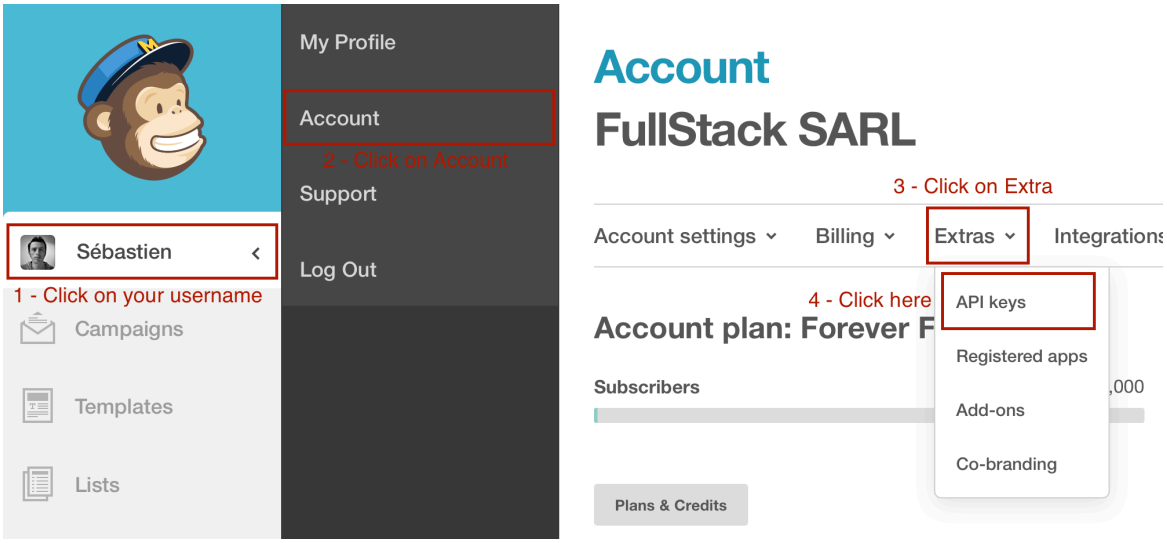
Use the API!

`gibbon` gem

```
# Gemfile
gem 'gibbon'
```

```
bundle install
```

Create an API key

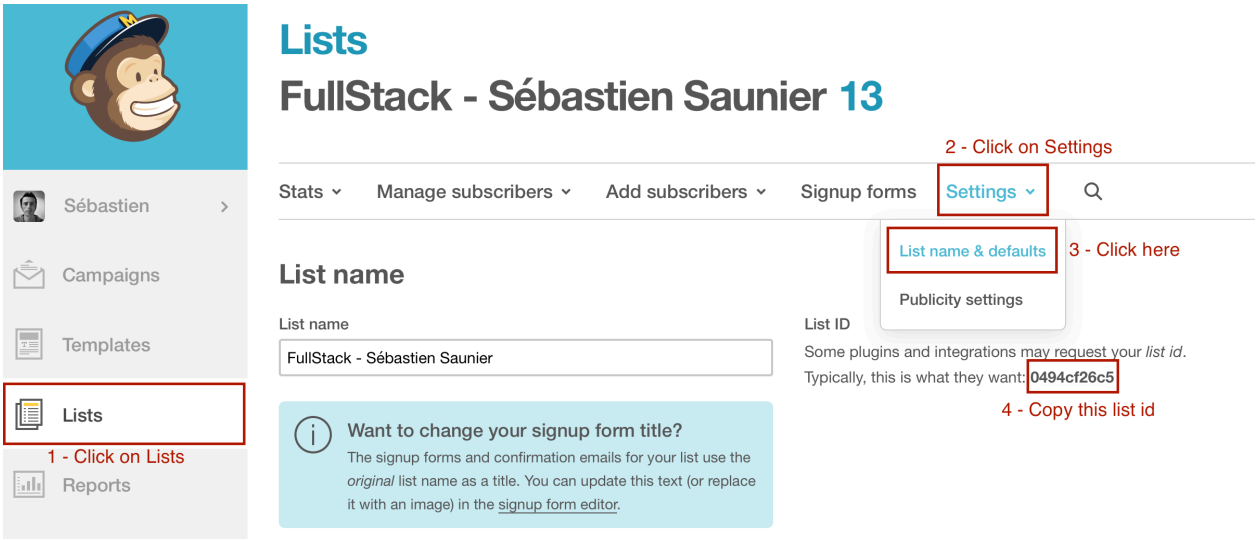


Click on the *Create an API key* button. Copy the key.

Sensitive data

As the API key is sensitive data, you should not store it in the code directly Supposing you have the figaro (https://github.com/laserlemon/figaro#getting-started) gem installed:

```
# config/application.yml

MAILCHIMP_API_KEY: "ehrv************************123"
```



Configuration data

Put this id in your configuration file as well:

```
# config/application.yml

MAILCHIMP_LIST_ID: "0494cf26c5"
```

Business Logic

```ruby
# app/services/subscribe_to_newsletter_service.rb
class SubscribeToNewsletterService
  def initialize(user)
    @user = user
    @gibbon = Gibbon::Request.new(api_key: ENV['MAILCHIMP_API_KEY'])
    @list_id = ENV['MAILCHIMP_LIST_ID']
  end

  def call
    @gibbon.lists(@list_id).members.create(
      body: {
        email_address: @user.email,
        status: "subscribed",
        # merge_fields: {
        #   FNAME: @user.first_name,
        #   LNAME: @user.last_name
        # }
      }
    )
  end
end
```

Register user

```ruby
# app/models/user.rb
class User < ApplicationRecord
  after_create :subscribe_to_newsletter

  private

  def subscribe_to_newsletter
    SubscribeToNewsletterService.new(self).call
  end
end
```

More stuff

Read the `gibbon` gem README (https://github.com/amro/gibbon)

Happy Mailing!

Ajax in Rails

Let's start from scratch

Generate new Restaurant app

```
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/devise.rb \
  restaurants_ajaxified
```

```
cd restaurants_ajaxified
```

Models

```
rails g model Restaurant name address
rails g model Review content:text restaurant:references
rails db:migrate
```

```ruby
# app/models/restaurant.rb
class Restaurant < ApplicationRecord
  has_many :reviews, dependent: :destroy
end
```

```ruby
# app/models/review.rb
class Review < ApplicationRecord
  belongs_to :restaurant
  validates :content, length: { minimum: 20 }
end
```

```
git add app/models db/*
git commit -m "Generate Restaurant & Review models"
```

Seed

```
# db/seeds.rb
puts 'Creating restaurants...'
Restaurant.create!({
  name: "Le Dindon en Laisse",
  address: "18 Rue Beautreillis, 75004 Paris, France"
})
Restaurant.create!({
  name: "Neuf et Voisins",
  address: "Van Arteveldestraat 1, 1000 Brussels, Belgium"
})
puts 'Finished!'
```

```
rails db:seed
```

```
git add db/seeds.rb
git commit -m "Seed 2 restaurants"
```

Restaurant Controller & Routes

```
rails g controller restaurants
```

```
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants, only: [ :index, :show ]
end
```

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def index
    @restaurants = Restaurant.all
  end

  def show
    @restaurant = Restaurant.find(params[:id])
  end
end
```

Restaurant Index View

```
<!-- app/views/restaurants/index.html.erb -->
<h1>Restaurants</h1>

<ul>
  <% @restaurants.each do |restaurant| %>
    <li><%= link_to restaurant.name, restaurant_path(restaurant) %></li>
  <% end %>
</ul>
```

Restaurant Show View

```
<!-- app/views/restaurants/show.html.erb -->
<h1><%= @restaurant.name %></h1>
<%= image_tag "https://maps.googleapis.com/maps/api/staticmap?zoom=15&size=400x300&sensor=false&maptype=roadmap&markers=color:red|#{@restaur
t.address}" %>
<h2>
  <%= pluralize @restaurant.reviews.size, "review" %>
</h2>

<div id="reviews">
  <% if @restaurant.reviews.blank? %>
    Be the first to leave a review for <%= @restaurant.name %>
  <% else %>
    <% @restaurant.reviews.each do |review| %>
      <%= render 'reviews/show', review: review %>
    <% end %>
  <% end %>
</div>
```

```
<!-- app/views/reviews/_show.html.erb -->
<p><%= review.content %></p>
```

```
git status
git add .
git commit -m "Add Restaurant index & show views"
```

We want to add reviews to a restaurant, but with the form inlined in the restaurant show view.

Add a new route

```
# config/routes.rb
Rails.application.routes.draw do
  resources :restaurants, only: [ :index, :show ] do
    resources :reviews, only: :create
  end
end
```

```
rails g controller reviews
```

```
# app/controllers/reviews_controller.rb
class ReviewsController < ApplicationController
  def create
    @restaurant = Restaurant.find(params[:restaurant_id])
    @review = Review.new(review_params)
    @review.restaurant = @restaurant
    if @review.save
      redirect_to restaurant_path(@restaurant)
    else
      render 'restaurants/show'
    end
  end

  private

  def review_params
    params.require(:review).permit(:content)
  end
end
```

Review form

```
class RestaurantsController < ApplicationController
  # [...]
  def show
    @restaurant = Restaurant.find(params[:id])
    @review = Review.new  # <-- You need this now.
  end
end
```

```
<!-- app/views/restaurants/show.html.erb -->

<!-- [...] -->

<%= render 'reviews/form', restaurant: @restaurant, review: @review %>
```

```
<!-- app/views/reviews/_form.html.erb -->
<%= simple_form_for([ restaurant, review ]) do |form| %>
  <%= form.input :content, as: :text %>
  <%= form.button :submit %>
<% end %>
```

It works!

```
git status
git add .
git commit -m "Add review to restaurant"
```

Let's ajaxify this app

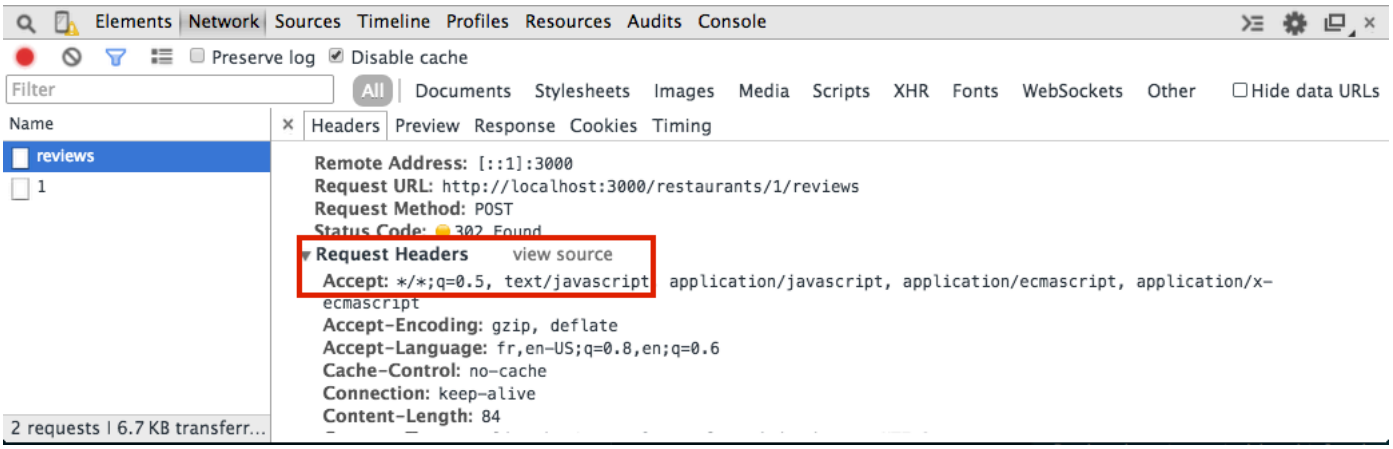
Configuration

Make sure this is set to `true` :

```
# config/application.rb

# [...]
  class Application < Rails::Application
    config.action_view.embed_authenticity_token_in_remote_forms = true
    # [...]
  end
```

```
remote: true
```

```
<!-- app/views/reviews/_form.html.erb -->
<%= simple_form_for([ restaurant, review ], remote: true) do |form| %>
  <%= form.input :content %>
  <%= form.button :submit %>
<% end %>
```




```
# app/controllers/reviews_controller.rb
class ReviewsController < ApplicationController
  def create
    # [...]
    if @review.save
      respond_to do |format|
        format.html { redirect_to restaurant_path(@restaurant) }
        format.js  # <-- will render `app/views/reviews/create.js.erb`
      end
    else
      respond_to do |format|
        format.html { render 'restaurants/show' }
        format.js  # <-- idem
      end
    end
  end

  # [...]
end
```

```
// app/views/reviews/create.js.erb
// Here you will write *javascript* that would be executed by the browser
<% if @review.errors.any? %>
  // Render new review form with errors
  $('#new_review').html("<%= j render 'reviews/form', restaurant: @restaurant, review: @review %>");
<% else %>
  // Create a paragraph for the new review. But hide it.
  var new_review = $("<%= j render 'reviews/show', review: @review %>").hide();
  // Add it to the DOM, at the end of the existing reviews. It's still hidden.
  $('#reviews').append(new_review);
  // Show the new review with an animation!
  new_review.slideDown();
  // Reset the form
  $('#new_review').html("<%= j render 'reviews/form', restaurant: @restaurant, review: Review.new %>");
<% end %>
```

That's it! Your form works with JavaScript enabled **or** disabled.

Unobtrusive JavaScript FTW.

One more thing

You can use this technique with `link_to` as well.

```
<!-- app/views/restaurants/index.html.erb -->
<ul>
  <% @restaurants.each do |restaurant| %>
    <li data-restaurant-id="<%= restaurant.id %>">
      <%= restaurant.name %>
      <%= link_to "Delete", restaurant, method: :delete, remote: true %>
    </li>
  <% end %>
</ul>
```

```ruby
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  # TODO: Add the `destroy` action to the
  #       `restaurants` resources in `config/routes.rb`
  def destroy
    @restaurant = Restaurant.find(params[:id])
    @restaurant.destroy
  end
end
```

```ruby
// app/views/restaurants/destroy.js.erb
$('[data-restaurant-id=<%= @restaurant.id %>]').slideUp();
```

Happy Ajaxification!

Pundit

Authorization

`!= from Authentication .`

Example

- As a **user**, I **can** create a restaurant.
- As a **user**, I **can** update/destroy a restaurant if I **created** it.

Let's scaffold an example

```
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/devise.rb \
  restaurants_with_pundit
```

```
rails g scaffold restaurant name:string user:references
rails db:migrate
```

Update the scaffold so that `current_user` is used in `RestaurantsController#create` .

Let's create 2 users and create restaurants with these users (use 2 browsers).

Problem: you can delete a restaurant you did not create.

Solution: use the `pundit` (https://github.com/elabs/pundit) gem.

Pundit Setup (1)

```
# Gemfile
gem "pundit"
```

```
bundle install
rails g pundit:install
```

Pundit Setup (2)

```ruby
class ApplicationController < ActionController::Base
  # [...]
  before_action :authenticate_user!
  include Pundit

  # Pundit: white-list approach.
  after_action :verify_authorized, except: :index, unless: :skip_pundit?
  after_action :verify_policy_scoped, only: :index, unless: :skip_pundit?

  # Uncomment when you *really understand* Pundit!
  # rescue_from Pundit::NotAuthorizedError, with: :user_not_authorized
  # def user_not_authorized
  #   flash[:alert] = "You are not authorized to perform this action."
  #   redirect_to(root_path)
  # end

  private

  def skip_pundit?
    devise_controller? || params[:controller] =~ /(^(rails_)?admin)|(^pages$)/
  end
end
```

Usage

Protect `Restaurant`

```ruby
rails g pundit:policy restaurant
# => generates the file `app/policies/restaurant_policy.rb`
```

Question: Who can create a restaurant?

Anyone

```ruby
# app/policies/restaurant_policy.rb
class RestaurantPolicy < ApplicationPolicy
  # [...]

  def create?
    return true
  end
end
```

And in the controller, use

```ruby
authorize @restaurant
```

Update the views!

```ruby
<% if policy(Restaurant).create? %>
  <%= new_restaurant_path %>
<% end %>
```

Passing a **class** to `policy` as we don't have a specific restaurant instance.

Rinse & repeat for `update` and `destroy`

```ruby
# app/policies/restaurant_policy.rb
class RestaurantPolicy < ApplicationPolicy
  def update?
    record.user == user
    # - record: the restaurant passed to the `authorize` method in controller
    # - user:   the `current_user` signed in with Devise.
  end

  def destroy?
    record.user == user
  end
end
```

In the view we can use `policy` with an **instance**.

```ruby
<% @restaurants.each do |restaurant| %>
  <% if policy(restaurant).edit? %>
    <%= link_to "Update", edit_restaurant_path(restaurant) %>
  <% end %>
<% end %>
```

Scope

Special case for `index` action:

```
# app/controllers/restaurants_controller.rb
class RestaurantsController < ApplicationController
  def index
    @restaurants = policy_scope(Restaurant).order(created_at: :desc)
  end
end
```

```
# app/policies/restaurant_policy.rb
class RestaurantPolicy < ApplicationPolicy
  class Scope < Scope
    def resolve
      scope.all

      # For a multi-tenant SaaS app, you may want to use:
      # scope.where(user: user)
    end
  end
end
```

Example code

See lewagon/rails-pundit (https://github.com/lewagon/rails-pundit/compare/app-without-authorization...app-with-authorization?diff=split&name=app-with-authorization)

And read all the doc (https://github.com/elabs/pundit)

Admin users

```
rails g migration AddAdminToUsers admin:boolean
rails db:migrate
```

```
rails c
pry> user = User.where(email: 'seb@lewagon.org').first
pry> user.admin = true
pry> user.save
```

You can now use this `admin` field in your policies!

i18n

Internationalization

```
"nternationalizatio".length == 18  # i18n
```

We'll **abstract** strings out of the code.

i18n is already in your app.

```
# config/locales/en.yml

en:
  hello: "Hello world"
```

Views

```
I18n.translate # Lookup text translations
I18n.localize  # Localize Date and Time objects to local formats
```

```
<!-- app/views/pages/home.html.erb -->

<!-- Looks for the 'hello' key in config/locales/{current_locale}.yml -->
<%= t('hello') %>

<!-- Display the current date in the current locale format -->
<%= l(Time.now) %>
```

You can nest locale keys:

```
en:
  hello: "Hello"
  home:
    welcome: "Welcome here!"
```

```
<!-- app/views/pages/home.html.erb -->

<%= t('home.welcome') %>
```

You will have 2+ locales

You want a way to switch locales, and display the content in the right one.

Not in a cookie or session

> You may be tempted to **store** the chosen locale in a **session** or a **cookie**, however **do not** do this. The **locale should be transparent and a part of the URL**. This way you won't break people's basic assumptions about the web itself: if you send a URL to a friend, they should see the same page and content as you. A fancy word for this would be that you're being RESTful. Read more about the RESTfu approach in Stefan Tilkov's articles. Sometimes there are exceptions to this rule and those are discussed below. *Rubyonrails Guide (http://guides.rubyonrails.org/i18n.html#setting-and-passing-the-locale)*

In the URL

Assuming you want `I18n.default_locale == :en`

```
http://www.awesomeapp.com/restaurants/2
```

```
http://www.awesomeapp.com/fr/restaurants/2
```

```
http://www.awesomeapp.com/es/restaurants/2
```

(You can also do it from the domain name (http://guides.rubyonrails.org/i18n.html#setting-the-locale-from-the-domain-name), requires more configuration on your Domain and Heroku)

```
# config/routes.rb
Rails.application.routes.draw do
  devise_for :users

  scope '(:locale)', locale: /fr|es/ do
    root to: 'pages#home'
    resources :restaurants

    # [...]
  end
end
```

```
# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  before_action :set_locale

  def set_locale
    I18n.locale = params.fetch(:locale, I18n.default_locale).to_sym
  end

  def default_url_options
    { locale: I18n.locale == I18n.default_locale ? nil : I18n.locale }
  end
end
```

Default locale

```
# config/application.rb

# [...]
config.i18n.default_locale = :en
```

Adding translations

Lazy lookup

```
# config/locales/en.yml
en:
  restaurants:
    index:
      title: "List of Restaurants"
```

```
<!-- app/views/restaurants/index.html.erb -->

<%= t('.title') %>
```

(note the **dot**)

Interpolation

```
# config/locales/en.yml
en:
  restaurants:
    show:
      reviews_for: 'Reviews for %{name}'
```

```
<!-- app/views/restaurants/index.html.erb -->

<%= t('.reviews_for', name: @restaurant.name) %>
```

Pluralization

```
# config/locales/en.yml
en:
  restaurants:
    show:
      reviews:
        zero:  'Be the first to review!'
        one:   'One review'
        other: '%{count} reviews'
```

```
<!-- app/views/restaurants/show.html.erb -->

<%= t('.reviews', count: @restaurant.reviews.size) %>
```

Safe HTML

Do not abuse this

```
# config/locales/en.yml
en:
  hello_html: 'Hello <strong>there</strong>!'
```

```
<!-- app/views/home/index.html.erb -->

<%= t('hello_html') %>
```

Time/Date formats

```
<!-- app/views/home/show.html.erb -->

<%= l(@restaurant.created_at, format: :short) %>
<%= l(@restaurant.created_at.to_date, format: :short) %>
```

Default date (https://github.com/svenfuchs/rails-i18n/blob/master/rails/locale/en.yml#L41-L44) and time (https://github.com/svenfuchs/rails-i18n/blob/master/rails/locale/en.yml#L206-L209) formats

```
# Gemfile
gem 'rails-i18n', '5.0.3'
```

Models

Suppose you ran

```
rails g scaffold restaurant name:string
```

Then you can i18n the `Restaurant` model name and attributes with

```
# config/locales/fr.yml
fr:
  activerecord:
    models:
      restaurant: "Resto"
    attributes:
      restaurant:
        name: "Nom"
```

Useful gems

For devise

```
# Gemfile

gem 'devise-i18n'
```

Then you need to import **i18n-compatible** views:

```
rails g devise:i18n:views
```

Why? Compare devise (https://github.com/plataformatec/devise/blob/master/app/views/devise/confirmations/new.html.erb) with devise-i18n-views (https://github.com/mcasimir/devise-i18n-views/blob/master/app/views/devise/confirmations/new.html.erb)

`i18n-tasks`

```
gem install i18n-tasks
```

(**not** for your `Gemfile`). README (https://github.com/glebm/i18n-tasks)

```
i18n-tasks health
i18n-tasks add-missing
i18n-tasks translate-missing  # Requires GOOGLE_TRANSLATE_API_KEY
```

Documentation

You can read the whole guide (http://guides.rubyonrails.org/i18n.html)

Admin Interfaces

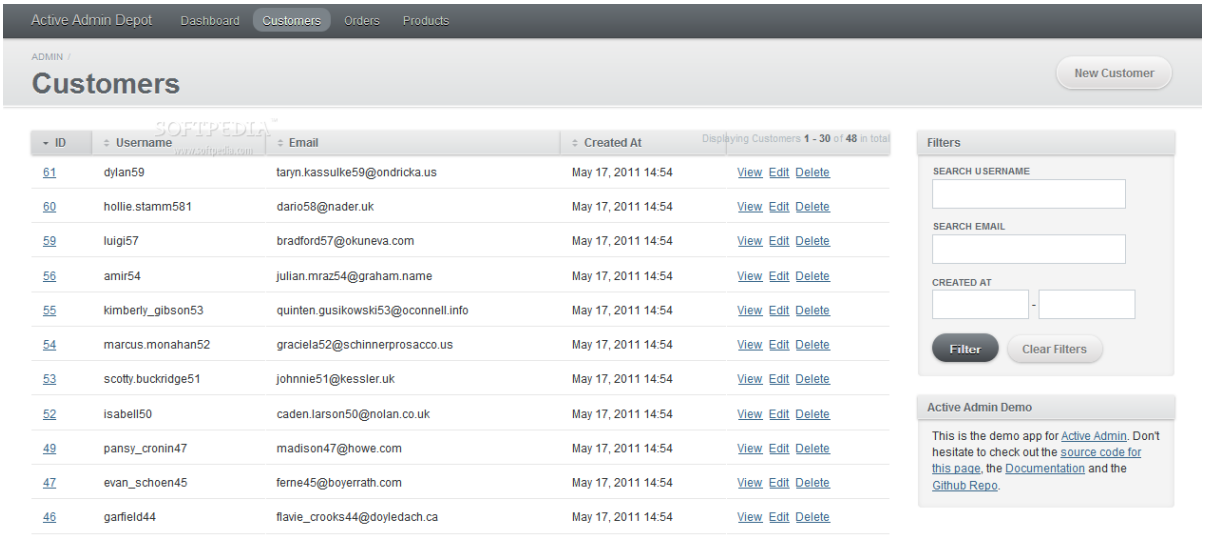
The match

ActiveAdmin vs **RailsAdmin** on Ruby Toolbox (https://www.ruby-toolbox.com/categories/rails_admin_interfaces)

Why do we need them?

`rails console` is cool, what about a web interface?

ActiveAdmin



Example

```
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/devise.rb \
  lafourchette_active_admin
```

```
cd lafourchette_active_admin
rails g model restaurant name:string description user:references
rails g model review content:text restaurant:references user:references
rails db:migrate
```

Then add `has_many` and validations.

Allow a user to be an admin

```
rails g migration AddAdminToUsers
```

Open the newly created `***_add_admin_to_users.rb` migration file:

```
def change
  add_column :users, :admin, :boolean, null: false, default: false
end
```

```
rails db:migrate
```

Let's **commit** (on `master`, only for this lecture) for a checkpoint here.

Setup

Let's branch from here (later we'll install RailsAdmin from here as well).

```
git checkout -b active-admin
```

```
# Gemfile

gem 'activeadmin', github: 'activeadmin/activeadmin'
gem 'inherited_resources', github: 'activeadmin/inherited_resources'
```

```
bundle install
```

Generator

Assuming you **already** have a devise `User` model.

```
rails generate active_admin:install --skip-users
#     create  config/initializers/active_admin.rb
#      create  app/admin
#      create  app/admin/dashboard.rb
#       route  ActiveAdmin.routes(self)
#    generate  active_admin:assets
#      create  app/assets/javascripts/active_admin.js.coffee
#      create  app/assets/stylesheets/active_admin.css.scss
#      create  db/migrate/20141119203507_create_active_admin_comments.rb
```

Then run:

```
rails db:migrate
```

Authentication configuration

```
# config/initializers/active_admin.rb

# Add this line at the beginning, before `ActiveAdmin.setup do |config|`
def authenticate_admin!
  redirect_to new_user_session_path unless current_user && current_user.admin
end

# Edit those four lines, to reuse existing `User` model.
ActiveAdmin.setup do |config|
  # [...]
  config.authentication_method = :authenticate_admin!
  # [...]
  config.current_user_method = :current_user
  # [...]
  config.logout_link_path = :destroy_user_session_path
  # [...]
  config.logout_link_method = :delete
end
```

Beware

In your `application.js`, `require_tree` will also ActiveAdmin JS for your entire website.

Switch to using specifics `require` for each file.

Grant existing user admin access

```
rails c  # Start a rails console, then type:
```

```
user = User.find_by_email('seb@lewagon.org')
user.admin = true
user.save
```

(You can do that in production with `heroku run rails c`)

Aren't you happy that **strong params** exist?

Access our brand new back-end

Go to `localhost:3000/admin` (http://localhost:3000) and enjoy.

Registering models

You need to tell ActiveAdmin which models it will handle.

Folder

`app/admin/`

Register a new model

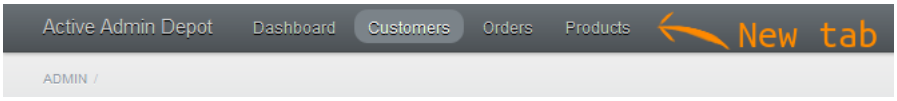
Say you have a `Product` model, then run

```
rails generate active_admin:resource product
#     create  app/admin/product.rb
```

And configure strong params for ActiveAdmin (https://github.com/activeadmin/activeadmin/blob/master/docs/2-resource-customization.md#setting-up-strong-parameters):

```
# app/admin/product.rb
ActiveAdmin.register Product do
  permit_params :name, :price # etc...
end
```

This adds the resource to your back-end:



Register the `User` model!

Of course :)

```
rails generate active_admin:resource user
```

DRY

ActiveAdmin understands `has_many` relations.

ActiveAdmin reuses your validations.

Advanced

Customization

User index view has too many columns!

Customize the index view

```
# app/admin/user.rb
ActiveAdmin.register User do
  index do
    selectable_column
    column :id
    column :email
    column :name
    column :created_at
    column :admin
    actions
  end
end
```

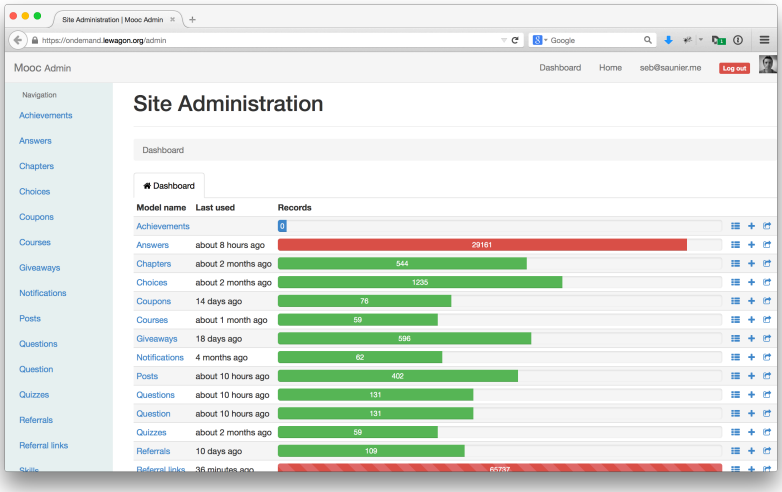
User creation/edition form has too many fields!

Customize the form

```
# app/admin/user.rb
ActiveAdmin.register User do
  form do |f|
    f.inputs "Identity" do
      f.input :name
      f.input :email
    end
    f.inputs "Admin" do
      f.input :admin
    end
    f.actions
  end

  permit_params :name, :email, :admin
end
```

More goodies in the documentation (http://activeadmin.info/docs/5-forms.html)

Rails Admin



Setup

For the lecture, let's go back on `master` and create `rails-admin` branch.

```
# Gemfile

gem 'remotipart', github: 'mshibuya/remotipart'
gem 'rails_admin', '>= 1.0.0.rc'
```

```
bundle install
```

```
rails g rails_admin:install
```

Configuration

BEWARE: Rails admin is accessible to **every logged in user**... Put this:

```
# in config/initializer/rails_admin.rb

RailsAdmin.config do |config|
  config.authorize_with do |controller|
    redirect_to main_app.root_path unless current_user && current_user.admin
  end

  # [...]
end
```

(Relaunch your `rails s`)

Your turn!

- activeadmin.info/docs/documentation.html (http://activeadmin.info/docs/documentation.html)
- github.com/sferik/rails_admin (https://github.com/sferik/rails_admin)

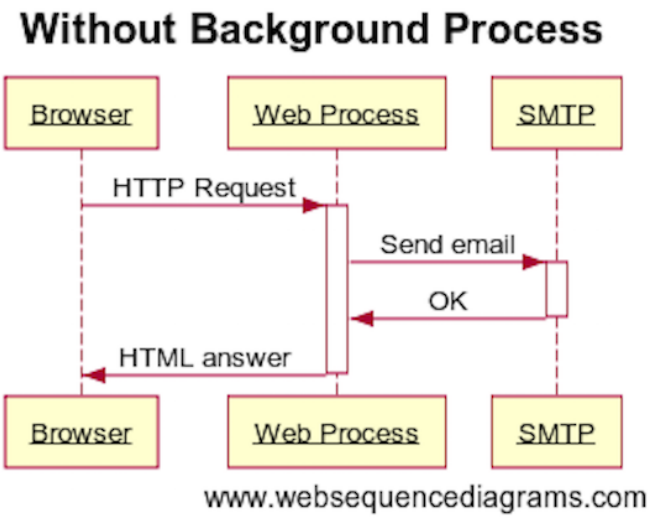
Background Jobs

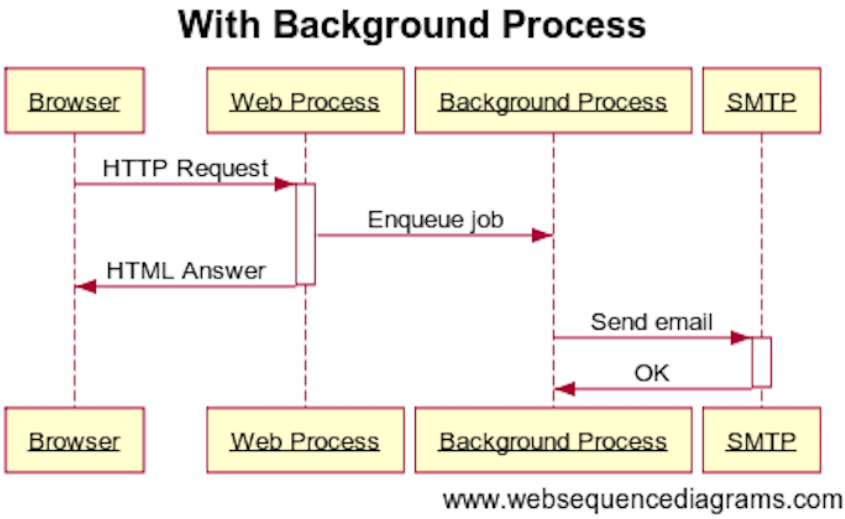
What for?

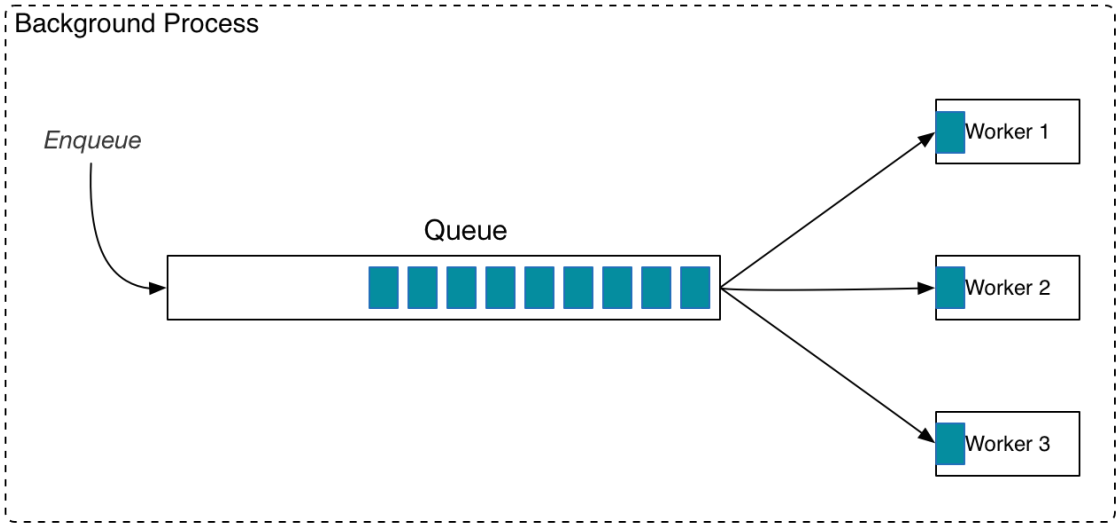
Push outside of the controller **time-consuming processes**.

Examples

- Delivering emails
- Posting to an API (Slack)
- Cleaning up
- etc.







Active Job

Rails Guide (http://edgeguides.rubyonrails.org/active_job_basics.html)

Starting from scratch

```
rails new \
    -T --database postgresql \
    -m https://raw.githubusercontent.com/lewagon/rails-templates/master/devise.rb \
    jobs_example
```

```
# We need admins to protect the jobs dashboard.
rails g migration AddAdminToUsers
```

```
def change
  add_column :users, :admin, :boolean, null: false, default: false
end
```

```
rails db:migrate
```

Our first job

```
rails generate job fake
# create  app/jobs/fake_job.rb
```

```ruby
class FakeJob < ApplicationJob
  queue_as :default

  def perform
    puts "I'm starting the fake job"
    sleep 3
    puts "OK I'm done now"
  end
end
```

Run the job

It will get run **synchronously**. Open a `rails c` and run:

```ruby
FakeJob.perform_now
```

Asynchronicity

How to enqueue jobs to be run later.

Back-end: Sidekiq

sidekiq.org (http://sidekiq.org/)

Alternatives: ActiveJob / QueueAdapters (http://api.rubyonrails.org/classes/ActiveJob/QueueAdapters.html)

Dependency over Redis

Redis (http://redis.io/) is a **key-value store**. Think of it as a PostgreSQL database storing **hashes** in memory. Sidekiq needs it.

```
# On OSX
brew update
brew install redis
brew services start redis
```

```
# Ubuntu
sudo apt-get install redis-server
```

(Teacher, you may want to run `redis-cli FLUSHALL`)

Set up

```
# Gemfile
gem 'sidekiq'
gem 'sidekiq-failures', github: 'mhfs/sidekiq-failures'
```

```
bundle install
bundle binstub sidekiq
```

```
# config/application.rb
class Application < Rails::Application
  # [...]
  config.active_job.queue_adapter = :sidekiq
end
```

```
# config/routes.rb
Rails.application.routes.draw do
  # Sidekiq Web UI, only for admins.
  require "sidekiq/web"
  authenticate :user, lambda { |u| u.admin } do
    mount Sidekiq::Web => '/sidekiq'
  end
end
```

```
# config/sidekiq.yml
:concurrency: 3
:timeout: 60
:verbose: true
:queues:  # Queue priority: https://github.com/mperham/sidekiq/wiki/Advanced-Options
  - default
  - mailers
```

Using ActiveJob

Let's start Sidekiq

Open a **new** terminal tab and run:

```
sidekiq
```

Enqueue our first job

Open a rails console with `rails c` and run:

```
FakeJob.perform_later
```

Arguments

You can pass arguments to the `perform_later` method.

BEWARE: Arguments will be serialized to json, so pass id, string, not full objects.

```
class UpdateUserJob < ApplicationJob
  queue_as :default

  def perform(user_id)
    user = User.find(user_id)
    puts "Calling Clearbit API for #{user.email}..."
    # TODO: perform a time consuming task like Clearbit's Enrinchment API.
    sleep 2
    puts "Done! Enriched #{user.email} with Clearbit"
  end
end
```

Enqueue from a model

```
# app/models/user.rb
class User < ApplicationRecord
  # [...]

  after_save :async_update # Run on create & update

  private

  def async_update
    UpdateUserJob.perform_later(self.id)
  end
end
```

Enqueue from a controller

```
# app/controllers/profiles_controller.rb
class ProfilesController < ApplicationController
  def update
    if current_user.update(user_params)
      UpdateUserJob.perform_later(current_user.id)  # <- The job is queued
      flash[:notice] = "Your profile has been updated"
      redirect_to root_path
    else
      render :edit
    end
  end

  private

  def user_params
    # Some strong params of your choice
  end
end
```

Enqueue from a rake task (1)

```
rails g task user update_all
```

```ruby
# lib/tasks/user.rake
namespace :user do
  desc "Enriching all users with Clearbit (async)"
  task :update_all => :environment do
    users = User.all
    puts "Enqueuing update of #{users.size} users..."
    users.each do |user|
      UpdateUserJob.perform_later(user.id)
    end
    # rake task will return when all jobs are _enqueued_ (not done).
  end
end
```

```ruby
# Look, there's a new task!
rails -T | grep user

# We can run it with
rails user:update_all
```

Enqueue from a rake task (2)

```ruby
# lib/tasks/user.rake
namespace :user do
  desc "Enriching a given user with Clearbit (sync)"
  task :update, [:user_id] => :environment do |t, args|
    user = User.find(args[:user_id])
    puts "Enriching #{user.email}..."
    UpdateUserJob.perform_now(user.id)
    # rake task will return when job is _done_
  end
end
```

```ruby
# Update user of id 42 with this command
noglob rails user:update[42]
```

Mailers

No need to write any job, juste call `deliver_later` .

```ruby
# UserMailer.welcome(user).deliver_now

UserMailer.welcome(user_id).deliver_later
```

(Enqueued to the `mailers` queue)

BEWARE: Again, arguments will be serialized to json, so pass id, string, not full objects.

Delay the job

```ruby
FakeJob.set(wait: 1.minute).perform_later

FakeJob.set(wait_until: Day.tomorrow.noon).perform_later
```

By default, Sidekiq checks for scheduled job every **15 seconds** (see doc (https://github.com/mperham/sidekiq/wiki/Scheduled-Jobs#checking-for-new-jobs)).

Heroku

We'll use Redis Cloud (https://elements.heroku.com/addons/rediscloud)

```
heroku addons:create rediscloud
```

```ruby
# config/initializers/redis.rb
$redis = Redis.new

url = ENV["REDISCLOUD_URL"]

if url
  Sidekiq.configure_server do |config|
    config.redis = { url: url }
  end

  Sidekiq.configure_client do |config|
    config.redis = { url: url }
  end
  $redis = Redis.new(:url => url)
end
```

Update your `Procfile` (https://devcenter.heroku.com/articles/procfile) file

```
# Procfile
web: bundle exec puma -C config/puma.rb
worker: bundle exec sidekiq -C config/sidekiq.yml
```

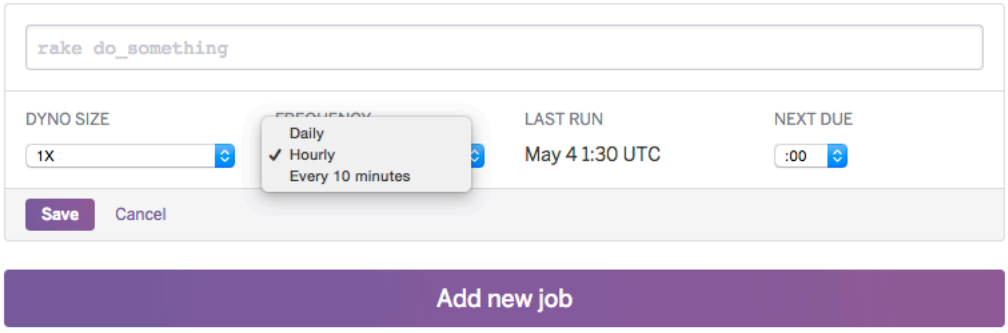
Then you must:

1. Commit
2. Deploy
3. Go to Heroku Dashboard to set dynos. Or your cli:

```
heroku ps:scale worker=1
heroku ps # Check worker dyno is running
```

Heroku Cron jobs

Use the Heroku Scheduler (https://devcenter.heroku.com/articles/scheduler)



Happy Asynchronizing!

Build an API

Intro

- Build a iOS app
- Provide service to customers
- Build a platform
- etc.

No big deal

It's just a new set of **Controllers**...

...which return **JSON** instead of HTML (the `.html.erb` views).

Starting from Scratch

With **users**

```
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/devise.rb \
  restaurants_api
cd restaurants_api
```

Install Pundit (/lectures/rails/pundit/#/2) then

```
rails g model restaurant name address user:references
rails g model comment content:text restaurant:references user:references
# Add the `has_many` to models
# Add some validations

rails db:migrate
rails g pundit:policy restaurant
```

API Setup

```
mkdir -p app/controllers/api/v1
touch app/controllers/api/v1/base_controller.rb
```

```ruby
# app/controllers/api/v1/base_controller.rb
class Api::V1::BaseController < ActionController::Base
  include Pundit

  after_action :verify_authorized, except: :index
  after_action :verify_policy_scoped, only: :index

  rescue_from StandardError,                with: :internal_server_error
  rescue_from Pundit::NotAuthorizedError,   with: :user_not_authorized
  rescue_from ActiveRecord::RecordNotFound, with: :not_found

  private

  def user_not_authorized(exception)
    render json: {
      error: "Unauthorized #{exception.policy.class.to_s.underscore.camelize}.#{exception.query}"
    }, status: :unauthorized
  end

  def not_found(exception)
    render json: { error: exception.message }, status: :not_found
  end

  def internal_server_error(exception)
    if Rails.env.development?
      response = { type: exception.class.to_s, error: exception.message }
    else
      response = { error: "Internal Server Error" }
    end
    render json: response, status: :internal_server_error
  end
end
```

1st endpoint: `index`

```
GET /api/v1/restaurants  # unauthenticated
```

```ruby
# config/routes.rb
Rails.application.routes.draw do
  # [...]
  namespace :api, defaults: { format: :json } do
    namespace :v1 do
      resources :restaurants, only: [ :index ]
    end
  end
end
```

Controller (1)

```
touch app/controllers/api/v1/restaurants_controller.rb
mkdir -p app/views/api/v1/restaurants
```

Controller (2)

```ruby
# app/controllers/api/v1/restaurants_controller.rb
class Api::V1::RestaurantsController < Api::V1::BaseController
  def index
    @restaurants = policy_scope(Restaurant)
  end
end
```

View (json)

The view now renders **json** and is built using the **jbuilder** gem.

```
touch app/views/api/v1/restaurants/index.json.jbuilder
```

```ruby
# app/views/api/v1/restaurants/index.json.jbuilder
json.array! @restaurants do |restaurant|
  json.extract! restaurant, :id, :name, :address
end
```

```
rails s
```

```
curl -s http://localhost:3000/api/v1/restaurants | jq
```

(using `jq` (https://stedolan.github.io/jq/download/) binary)

2nd endpoint: `show`

```
GET /api/v1/restaurants/:id # unauthenticated
```

```
# config/routes.rb
Rails.application.routes.draw do
  # [...]
  namespace :api, defaults: { format: :json } do
    namespace :v1 do
      resources :restaurants, only: [ :index, :show ]
    end
  end
end
```

```
# app/controllers/api/v1/restaurants_controller.rb
class Api::V1::RestaurantsController < Api::V1::BaseController
  before_action :set_restaurant, only: [ :show ]

  def show
  end

  private

  def set_restaurant
    @restaurant = Restaurant.find(params[:id])
    authorize @restaurant  # For Pundit
  end
end
```

```
# app/views/api/v1/restaurants/show.json.jbuilder
json.extract! @restaurant, :id, :name, :address
json.comments @restaurant.comments do |comment|
  json.extract! comment, :id, :content
end
```

```
curl -s http://localhost:3000/api/v1/restaurants/1 | jq
```

3rd endpoint: `update`

```
PATCH /api/v1/restaurants/:id # authenticated
```

The API should make a call **on behalf** of a given user.

In the browser, there is a **session** stored in a **cookie**.

For an API, we'll use a **token**.

```
# Gemfile
gem "simple_token_authentication"
```

```
bundle install
rails g migration AddTokenToUsers "authentication_token:string{30}:uniq"
rails db:migrate
```

```
# app/models/user.rb
class User < ApplicationRecord
  acts_as_token_authenticatable
  # [...]
end
```

```
# config/routes.rb
Rails.application.routes.draw do
  # [...]
  namespace :api, defaults: { format: :json } do
    namespace :v1 do
      resources :restaurants, only: [ :index, :show, :update ]
    end
  end
end
```

```ruby
# app/controllers/api/v1/restaurants_controller.rb
class Api::V1::RestaurantsController < Api::V1::BaseController
  acts_as_token_authentication_handler_for User, except: [ :index, :show ]
  before_action :set_restaurant, only: [ :show, :update ]

  def update
    if @restaurant.update(restaurant_params)
      render :show
    else
      render_error
    end
  end

  private

  def restaurant_params
    params.require(:restaurant).permit(:name, :address)
  end

  def render_error
    render json: { errors: @restaurant.errors.full_messages },
      status: :unprocessable_entity
  end
end
```

Usage

```ruby
# rails c
user = User.find_by_email("seb@lewagon.org")
user.save  # The user did not have any token yet. This call generated one.
user.reload.authentication_token
# => "a6hYpzsfNJdYC6zEMxs3"
```

```
curl -i -X PATCH                                    \
       -H 'Content-Type: application/json'          \
       -H 'X-User-Email: seb@lewagon.org'           \
       -H 'X-User-Token: a6hYpzsfNJdYC6zEMxs3'      \
       -d '{ "restaurant": { "name": "New name" } }'    \
       http://localhost:3000/api/v1/restaurants/1
```

You can also use a browser "Rest Client" (Like Postman) or ruby code (http://stackoverflow.com/a/12161762/197944).

4th endpoint: `create`

```
POST /api/v1/restaurants # authenticated
```

```ruby
# config/routes.rb
Rails.application.routes.draw do
  # [...]
  namespace :api, defaults: { format: :json } do
    namespace :v1 do
      resources :restaurants, only: [ :index, :show, :update, :create ]
    end
  end
end
```

```ruby
# app/controllers/api/v1/restaurants_controller.rb
class Api::V1::RestaurantsController < Api::V1::BaseController
  def create
    @restaurant = Restaurant.new(restaurant_params)
    @restaurant.user = current_user
    authorize @restaurant
    if @restaurant.save
      render :show, status: :created
    else
      render_error
    end
  end
end
```

Usage

```
curl -i -X POST                                                 \
     -H 'Content-Type: application/json'                        \
     -H 'X-User-Email: seb@lewagon.org'                         \
     -H 'X-User-Token: a6hYpzsfNJdYC6zEMxs3'                    \
     -d '{ "restaurant": { "name": "New restaurant", "address": "Paris" } }' \
     http://localhost:3000/api/v1/restaurants
```

5th endpoint: `destroy`

```
DELETE /api/v1/restaurants/:id # authenticated
```

```
# config/routes.rb
Rails.application.routes.draw do
  # [...]
  namespace :api, defaults: { format: :json } do
    namespace :v1 do
      resources :restaurants, only: [ :index, :show, :update, :create, :destroy ]
    end
  end
end
```

```
# app/controllers/api/v1/restaurants_controller.rb
class Api::V1::RestaurantsController < Api::V1::BaseController
  before_action :set_restaurant, only: [ :show, :destroy ]  # Re-use this.

  def destroy
    @restaurant.destroy
    head :no_content
    # No need to create a `destroy.json.jbuilder` view
  end
end
```

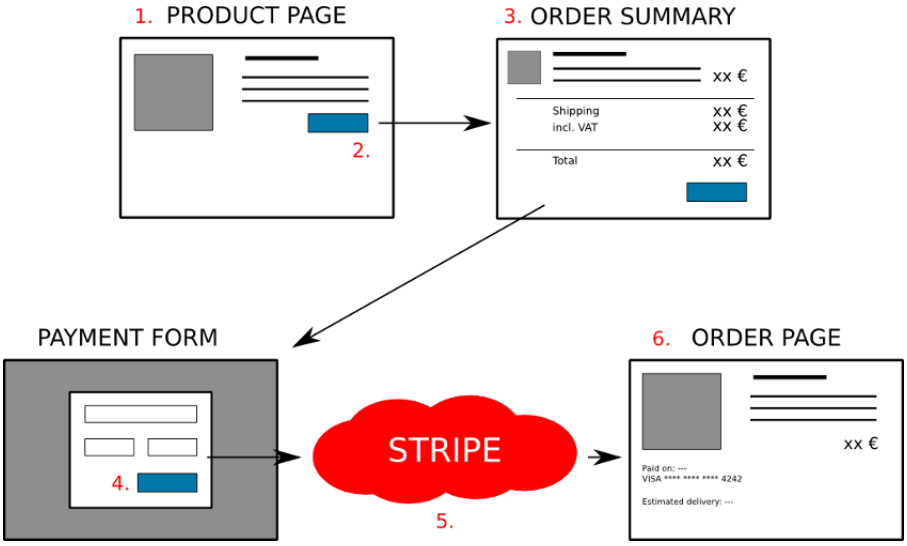
Usage

```
curl -i -X DELETE                             \
     -H 'X-User-Email: seb@lewagon.org'       \
     -H 'X-User-Token: a6hYpzsfNJdYC6zEMxs3'  \
     http://localhost:3000/api/v1/restaurants/1
```

Happy Api-ing!

Stripe

Payment workflow



New app

We'll build a Teddies shop!

```
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/devise.rb \
  teddies_shop
cd teddies_shop
```

Add the `coffee-script` gem to your Gemfile & `bundle install`.

Models

```
rails g model Category name:string
rails g model Teddy sku:string name:string category:references photo_url:string
```

And run migrations

```
rails db:migrate
```

Seeds

```
# db/seeds.rb
puts 'Cleaning database...'
Category.destroy_all
Teddy.destroy_all

puts 'Creating categories...'
geek = Category.create!(name: 'geek')
kids = Category.create!(name: 'kids')

puts 'Creating teddies...'
Teddy.create!(sku: 'original-teddy-bear', name: 'Teddy bear', category: kids, photo_url: 'http://onehdwallpaper.com/wp-content/uploads/2015/
/Teddy-Bears-HD-Images.jpg')

Teddy.create!(sku: 'jean-mimi', name: 'Jean-Michel - Le Wagon', category: geek, photo_url: 'https://pbs.twimg.com/media/B_AUcKeU4AE6ZcG.jpg:
rge')
Teddy.create!(sku: 'octocat',   name: 'Octocat -  GitHub',      category: geek, photo_url: 'https://cdn.shopify.com/s/files/1/0051/4802/proc
ts/mona-2_1024x1024.jpg?v=1447180277')
puts 'Finished!'
```

And run them

```
rails db:seed
```

Listing our Teddies

Routes & Controller

```
rails g controller teddies
```

Edit routes

```
# config/routes.rb
root 'teddies#index'
resources :teddies, only: [:index, :show]
```

```
# app/controllers/teddies_controller.rb
skip_before_action :authenticate_user!

def index
  @teddies = Teddy.all
end

def show
  @teddy = Teddy.find(params[:id])
end
```

```
<!-- app/views/teddies/index.html.erb -->
<div class="container">
  <div class="row">
    <h1>Teddies</h1>
    <% @teddies.each do |teddy| %>
      <div class="col-sm-6 col-md-4">
        <div class="thumbnail">
          <%= link_to image_tag(teddy.photo_url), teddy_path(teddy) %>
          <div class="caption">
            <h3><%= link_to teddy.name, teddy_path(teddy) %></h3>
            <p><%= teddy.category.name %></p>
            <p>$$$</p>
          </div>
        </div>
      </div>
    <% end %>
  </div>
</div>
```

```
<!-- app/views/teddies/show.html.erb -->
<div class="container">
  <div class="row">
    <div class="col-sm-12">
      <h1><%= @teddy.name %></h1>
    </div>
  </div>
  <div class="row">
    <div class="col-sm-12 col-md-6">
      <%= image_tag(@teddy.photo_url, width: '100%') %>
    </div>
    <div class="col-sm-12 col-md-6">
      <p>Some awesome description of our amazing teddy.</p>
      <p>$$$</p>
    </div>
  </div>
</div>
```

Prices

Money-rails (https://github.com/RubyMoney/money-rails)

Add money rails gem

```
# Gemfile
gem 'money-rails'
```

Set default currency

```
# config/initializers/money.rb
MoneyRails.configure do |config|
  config.default_currency = :eur  # or :gbp, :usd, etc.
end
```

Add price to teddies

```
rails g migration add_price_to_teddies
```

Edit migration

```
add_monetize :teddies, :price, currency: { present: false }
```

And run it

```
rails db:migrate
```

Enable monetize

```
# app/models/teddy.rb
class Teddy < ApplicationRecord
  # [...]
  monetize :price_cents  # or :price_pennies
end
```

Seeds

Don't forget to set price in your seeds!

Displayng price

```
<!-- app/views/teddies/index.html.erb -->
<p>Amount: <%= humanized_money_with_symbol(teddy.price) %></p>
```

```
<!-- app/views/teddies/show.html.erb -->
<p>Amount: <%= humanized_money_with_symbol(@teddy.price) %></p>
```

Stripe

http://stripe.com/

```
# Gemfile
gem 'stripe'
```

```
bundle install
```

Add stripe initializer

```
# config/initializers/stripe.rb
Rails.configuration.stripe = {
  publishable_key: ENV['STRIPE_PUBLISHABLE_KEY'],
  secret_key:      ENV['STRIPE_SECRET_KEY']
}

Stripe.api_key = Rails.configuration.stripe[:secret_key]
```

Add your keys

```
# config/application.yml
development:
  STRIPE_PUBLISHABLE_KEY: "pk_**H"
  STRIPE_SECRET_KEY: "sk_**f"
```

Making an Order

Model

```
rails g model Order state:string teddy_sku:string amount:monetize payment:json
```

Edit migration to remove currency

```
t.monetize :amount, currency: { present: false }
```

Don't forget to run the migration

```
rails db:migrate
```

Enable monetize

```
# app/models/order.rb
monetize :amount_cents  # or :amount_pennies
```

Routes & Controller

Add orders controller

```
rails g controller orders
```

Edit routes

```
# config/routes.rb
resources :orders, only: [:show, :create]
```

Edit Teddy page

Add a form to create an Order

```
# app/views/teddies/show.html.erb
<%= form_tag orders_path do %>
  <%= hidden_field_tag 'teddy_id', @teddy.id %>
  <%= submit_tag 'Purchase', class: 'btn btn-primary' %>
<% end %>
```

Edit Orders controller

```
# app/controllers/orders_controller.rb
def create
  @teddy = Teddy.find(params[:teddy_id])
  order  = Order.create!(teddy_sku: @teddy.sku, amount: @teddy.price, state: 'pending')

  redirect_to new_order_payment_path(order)
end
```

Payments

Routes & Controller

```
rails g controller payments
```

Edit routes

```
# config/routes.rb
resources :orders, only: [:show, :create] do
  resources :payments, only: [:new, :create]
end
```

Edit Payment controller (1/2)

```
# app/controllers/payments_controller.rb
class PaymentsController < ApplicationController
  before_action :set_order

  def new
  end

  def create
    # ...
  end

private

  def set_order
    @order = Order.where(state: 'pending').find(params[:order_id])
  end
end
```

Edit payment form

```
<!-- app/views/payments/new.html.erb -->
<h1>Purchase of teddy <%= @order.teddy_sku %></h1>
<%= form_tag order_payments_path(@order) do %>
  <article>
    <label class="amount">
      <span>Amount: <%= humanized_money_with_symbol(@order.amount) %></span>
    </label>
  </article>

  <!-- # Commented for the lecture, remove comments.
  <script src="https://checkout.stripe.com/checkout.js" class="stripe-button"
    data-key="<%= Rails.configuration.stripe[:publishable_key] %>"
    data-name="My Teddy"
    data-email="<%= current_user.email %>"
    data-description="Teddy <%= @order.teddy_sku %>"
    data-amount="<%= @order.amount_cents %>"
    data-currency="<%= @order.amount.currency %>"></script>
  -->
<% end %>
```

Payment controller (2/2)

```
# app/controllers/payments_controller.rb
def create
  customer = Stripe::Customer.create(
    source: params[:stripeToken],
    email:  params[:stripeEmail]
  )

  charge = Stripe::Charge.create(
    customer:     customer.id,   # You should store this customer id and re-use it.
    amount:       @order.amount_cents, # or amount_pennies
    description:  "Payment for teddy #{@order.teddy_sku} for order #{@order.id}",
    currency:     @order.amount.currency
  )

  @order.update(payment: charge.to_json, state: 'paid')
  redirect_to order_path(@order)

rescue Stripe::CardError => e
  flash[:error] = e.message
  redirect_to new_order_payment_path(@order)
end
```

Edit Orders controller

```
# app/controllers/orders_controller.rb
def show
  @order = Order.where(state: 'paid').find(params[:id])
end
```

Edit order page

```
<!-- app/views/orders/show.html.erb -->
<h1>Order</h1>
<ul>
  <li>Status: <%= @order.state %></li>
  <li>Item: <%= @order.teddy_sku %></li>
  <li>Amount: <%= humanized_money_with_symbol(@order.amount) %></li>
<ul>
```

Testing

```
4242 4242 4242 4242
```

More cards: https://stripe.com/docs/testing#cards

Happy Striping!

Search

This is not enough:

```
Movie.where("title ILIKE ?", "%#{params[:search]}%")
```

`superman batman` in search won't return *Batman v Superman: Dawn of Justice*

Postgresql Fulltext Search

```
=# SELECT 'a fat cat sat on a mat and ate a fat rat' ILIKE '%cat rat%';
 ?column?
----------
 f
(1 row)
```

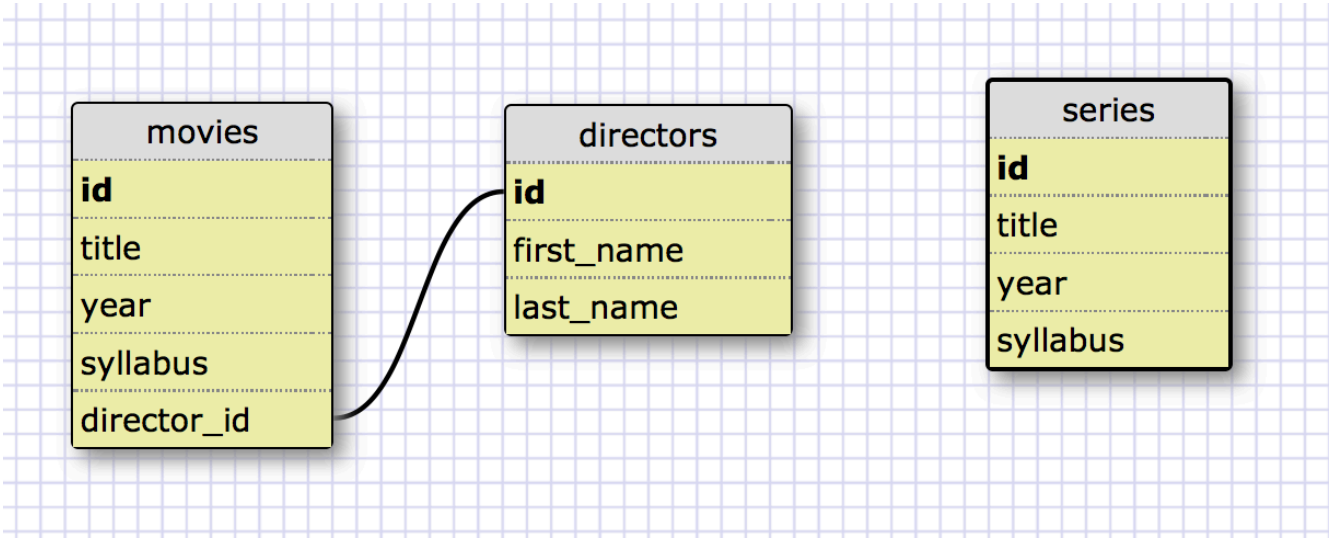
```
=# SELECT 'a fat cat sat on a mat and ate a fat rat' @@ 'cat rat';
 ?column?
----------
 t
(1 row)
```

Let's start from scratch

```
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/minimal.rb \
  imdb_search
```

```
cd imdb_search
```

Schema



Migrations

```
rails g model director first_name last_name
rails g model movie title year:integer syllabus:text director:references
rails g model serie title year:integer syllabus:text
```

```
rails db:migrate
```

And add `has_many :movies` to `Director`.

Seed (https://gist.github.com/ssaunier/25920c896baa0e4495fd)

```
# db/seeds.rb
require "open-uri"
require "yaml"

file = "https://gist.githubusercontent.com/ssaunier/25920c896baa0e4495fd/raw/9c26ce104c15fc237126bf6b136b8e318368f8ad/imdb.yml"
sample = YAML.load(open(file).read)

puts 'Creating directors...'
directors = {}  # slug => Director
sample["directors"].each do |director|
  directors[director["slug"]] = Director.create! director.slice("first_name", "last_name")
end

puts 'Creating movies...'
sample["movies"].each do |movie|
  director = directors[movie["director_slug"]]
  Movie.create! movie.slice("title", "year", "syllabus").merge(director: director)
end

puts 'Creating series...'
sample["series"].each do |serie|
  Serie.create! serie
end
puts 'Finished!'
```

`pg_search` gem

github.com/Casecommons/pg_search (https://github.com/Casecommons/pg_search)

```
# Gemfile
gem 'pg_search'
```

Searching one model (scope)

```
class Movie < ApplicationRecord
  include PgSearch
  pg_search_scope :search_by_title_and_syllabus, against: [ :title, :syllabus ]

  # [...]
end
```

You can add weighting (https://github.com/Casecommons/pg_search#tsearch-full-text-search) to each field searched.

Searching through association

Still only **one** model.

```ruby
class Movie < ApplicationRecord
  include PgSearch
  pg_search_scope :global_search,
    against: [ :title, :syllabus ],
    associated_against: {
      director: [ :first_name, :last_name ]
    }

  belongs_to :director
end
```

Multi-search - Setup

```
rails g pg_search:migration:multisearch
rails db:migrate
```

```ruby
class Movie < ApplicationRecord
  include PgSearch
  multisearchable against: [:title, :syllabus]
end

class Serie < ApplicationRecord
  include PgSearch
  multisearchable against: [:title, :syllabus]
end
```

Multi-search - Usage

```ruby
# rails c
PgSearch::Multisearch.rebuild(Movie)
PgSearch::Multisearch.rebuild(Serie)
results = PgSearch.multisearch('superman')

results.each do |result|
  puts result.searchable
end
```

A lot more

Read the doc (https://github.com/Casecommons/pg_search).

Option #2: Elasticsearch

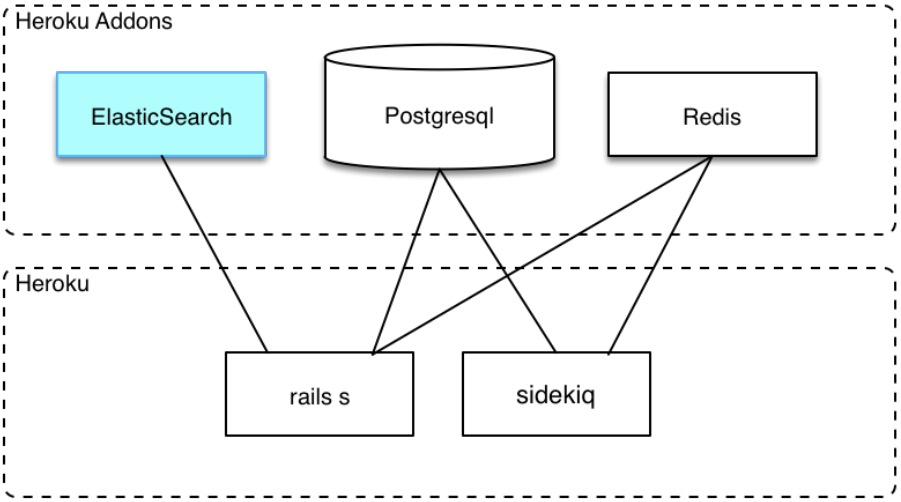
When use it?

You need one or more of these features:

- Scoring complexity (boost some fields)
- Mispelling
- Suggestions
- Highlighting
- Facets
- Autocomplete
- Geo-aware search

Infrastructure

You need to install ElasticSearch on each developer's workstation + on Heroku.




```
# OSX
brew install elasticsearch
brew services start elasticsearch
```

`searchkick` gem

github.com/ankane/searchkick (https://github.com/ankane/searchkick)

```
# Gemfile
gem 'searchkick', '~> 1.3'
```

```
bundle install
```

```
# app/models/movie.rb
class Movie < ApplicationRecord
  searchkick
  # [...]
end
```

```
# Launch a `rails c`
Movie.reindex # Once
Movie.search("superman")
```

Option #3: Algolia

When use it?

- You don't want to mess with complex ElasticSearch install
- You want **speed**
- Awesome autocompletion (hn.algolia.com (https://hn.algolia.com/))
- Fault-tolerance to typos
- Awesome on short texts (user names, movie titles).

If you index huge documents (like PDF, web pages, etc.), ElasticSearch will do a better job.

You can achieve Algolia awesomeness with ElasticSearch but it will take a lot of time and expertise.

`algolia-search` gem

github.com/algolia/algoliasearch-rails (https://github.com/algolia/algoliasearch-rails)

Testing

Setup

Let's create (once again) a Product Hunt clone, from scratch.

Let's just **code**, no `rails s` , no browser.

```
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/devise.rb \
  product_hunt_clone
cd product_hunt_clone
```

```
rails g migration AddNameToUsers first_name last_name
rails g model Product name tagline
rails db:migrate
```

```
# Gemfile
gem 'faker'
```

```
# db/seeds.rb
puts 'Creating products...'
5.times do
  Product.create!(
    name: Faker::Company.name,
    tagline: Faker::Company.catch_phrase
  )
end
puts 'Finished!'
```

```
rails db:seed
```

```
rails g controller products index
```

```
# config/routes.rb
Rails.application.routes.draw do
  devise_for :users
  root to: 'products#index'
end
```

```
curl https://raw.githubusercontent.com/lewagon/ui-components/master/source/stylesheets/components/_product_item.scss \
  > app/assets/stylesheets/components/_product_item.scss
```

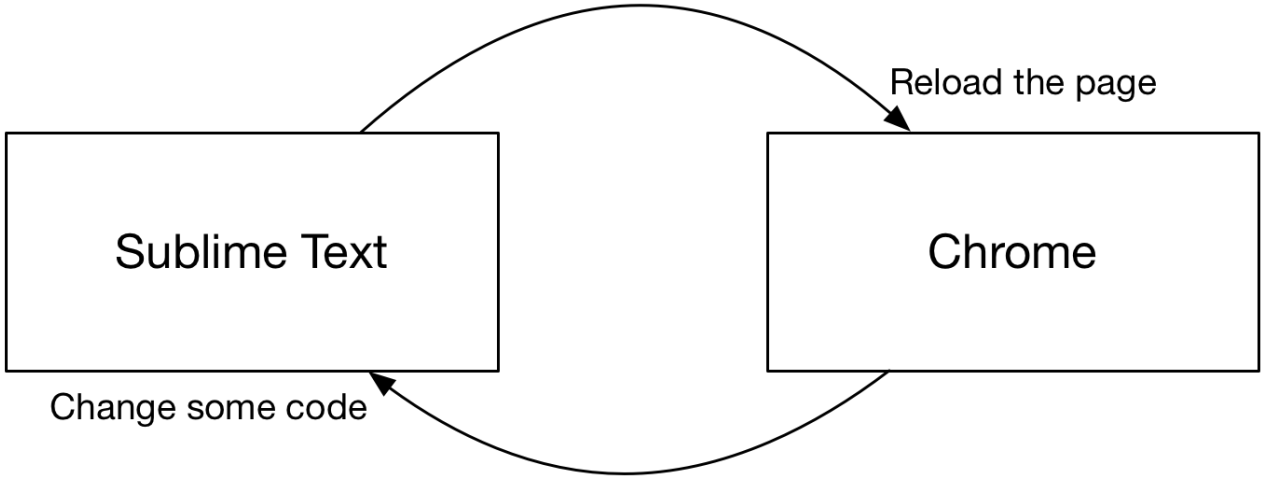
```
// app/assets/stylesheets/components/_index.scss
// [...]
@import "product_item";
```

```
class ProductsController < ApplicationController
  skip_before_action :authenticate_user!, only: :index

  def index
    @products = Product.all
  end
end
```

```
<!-- app/views/products/index.html.erb -->
<h1>Products</h1>
<% @products.each do |product| %>
  <div class="product">
    <div class='product-upvote'>
      <div class="product-arrow"></div>
      <div class='product-count'>0</div>
    </div>
    <div class='product-body'>
      <h3>Kudoz</h3>
      <p>Your career is ahead</p>
    </div>
  </div>
<% end %>
```

Now what?

Manual Testing




```
rails s
```

Looks good? Deploy!

```
git status
git add Gemfile Gemfile.lock app config db
git commit -m "Add Product Hunt list"
```

```
heroku create --region=eu
```

```
git push heroku master
heroku run rails db:migrate db:seed
heroku open
```

So far, so good.

Let's modify the view to display product name.

That's **so easy**! Let's code, commit and deploy!

```
<!-- [...] -->
<div class='product-body'>
  <h3><%= product.title %></h3>
  <p><%= product.tagline %></p>
</div>
```

```
git status
git add app/views
git commit -m "Put tagline below product title"
git push heroku master
```

```
heroku open
```

😱

```
heroku rollback

# Did you have migrations run in the last release?
heroku run rails db:rollback
```

😕

```
heroku logs --tail
```

```
heroku[router]: at=info method=GET path="/" host=serene-depths-42608.herokuapp.com request_id=5ea475f9-6db3-42b1-8ccd-2d0fd86e56b3 fwd="81.6
135.232" dyno=web.1 connect=1ms service=63ms status=500 bytes=1669
app[web.1]: Processing by ProductsController#index as HTML
app[web.1]: Started GET "/" for 81.64.135.232 at 2016-05-31 16:08:00 +0000
app[web.1]:   Rendering products/index.html.erb within layouts/application
app[web.1]: Completed 500 Internal Server Error in 24ms (ActiveRecord: 7.7ms)
app[web.1]:   Rendered products/index.html.erb within layouts/application (20.4ms)
app[web.1]:   Product Load (1.2ms)  SELECT "products".* FROM "products"
app[web.1]:      8:             <div class='product-count'>0</div>
app[web.1]:     11:             <h3><%= product.title %></h3>
app[web.1]:     10:           <div class='product-body'>
app[web.1]:      9:           </div>
app[web.1]:     13:           </div>
app[web.1]: app/views/products/index.html.erb:11:in `block in _app_views_products_index_html_erb___3319655114577111177_70175408985480'
app[web.1]: ActionView::Template::Error (undefined method `title' for #<Product:0x007fa5f7fdf8f0>):
app[web.1]: app/views/products/index.html.erb:4:in `_app_views_products_index_html_erb___3319655114577111177_70175408985480'
app[web.1]:
app[web.1]:
app[web.1]:     12:             <p><%= product.tagline %></p>
app[web.1]:     14:         </div>
```

Replace `title` with `name`, `commit` and re-deploy.

Automated Testing

```
rake
```

We'll use the **rails way** (http://guides.rubyonrails.org/testing.html), Minitest (http://www.rubydoc.info/gems/minitest/2.3.1/MiniTest/Assertions).

Setup (3 steps)

For a rails app generated with the –T flag. Skip this if not.

```
# config/application.rb
require "rails/test_unit/railtie"  # Uncomment this line
```

```
# config/environments/test.rb
config.action_dispatch.show_exceptions = true  # Update this line
```

We'll simulate a real browser. We'll do **Integration Testing**.

```
brew install phantomjs  # on OSX only
                        # Linux: see https://gist.github.com/julionc/7476620
```

```
# Gemfile
group :development do # Stuff you do not want in :test env
  gem 'better_errors'
  gem 'binding_of_caller'
end

group :development, :test do
  gem 'capybara'
  gem 'poltergeist'
  gem 'launchy'
  gem 'minitest-reporters'
  # [...]
end
```

```
bundle install
```

```ruby
# test/test_helper.rb
ENV['RAILS_ENV'] ||= 'test'
require File.expand_path('../../config/environment', __FILE__)
require 'rails/test_help'
require 'minitest/reporters'
Minitest::Reporters.use! [Minitest::Reporters::SpecReporter.new]

class ActiveSupport::TestCase
  fixtures :all
end

require 'capybara/rails'
class ActionDispatch::IntegrationTest
  include Capybara::DSL
  def teardown
    Capybara.reset_sessions!
    Capybara.use_default_driver
    Warden.test_reset!
  end
end

require 'capybara/poltergeist'
Capybara.default_driver = :poltergeist

include Warden::Test::Helpers
Warden.test_mode!
```

Test Principle

4 phases

```ruby
class TestSomeObject
  def test_something_on_object
    # setup
    # exercise
    # verify
    # teardown
  end
end
```

Our first integration test

```
rails generate integration_test home
```

```ruby
# test/integration/home_test.rb
require "test_helper"

class HomeTest < ActionDispatch::IntegrationTest
  test "loads correctly" do
    visit "/"
    assert_equal 200, page.status_code
    assert page.has_content?("Products")
  end
end
```

```
rake
```

🤔

Our test does not catch the `title / name` mix-up.

We need products in the database.

There are **2** different databases (cf `config/database.yml`)

Let's test with products!

```
touch test/fixtures/products.yml
```

```yaml
# test/fixtures/products.yml
kudoz:
  name: "Kudoz"
  tagline: "Tinder for Jobs"
```

```
rake
```

🔴 Tests are now broken if we use `title`.

Debugging your test

```
puts page.html
```

```
save_and_open_page
```

```
save_and_open_screenshot
```

Refine test

```
test "loads correctly" do
  visit "/"
  assert_equal 200, page.status_code
  assert page.has_selector?('.product', count: Product.count)
end
```

You can put **erb** inside your fixtures!

```
# products.yml
<% 1.upto(5) do |i| %>
product_<%= i %>:
  name: <%= Faker::Company.name %>
  tagline: <%= Faker::Company.catch_phrase %>
<% end %>
```

TDD

Test Driven Development

Let's add a new feature: **add a new product**.

```
touch test/fixtures/users.yml
```

```
# test/fixtures/users.yml
george:
  email: "george@abitbol.com"
  first_name: "George"
  last_name: "Abitbol"
```

```
rails g integration_test product
```

```
# test/integration/product_test.rb
require 'test_helper'

class ProductTest < ActionDispatch::IntegrationTest
  test "lets a signed in user create a new product" do
    login_as users(:george)
    visit "/products/new"

    fill_in "product_name", with: "Le Wagon"
    fill_in "product_tagline", with: "Change your life: Learn to code"
    click_button 'Create Product'

    # Should be redirected to Home with new product
    assert_equal root_path, page.current_path
    assert_equal 200, page.status_code
    assert page.has_content?("Change your life: Learn to code")
  end
end
```

And now, let's code!

We'll follow error messages. Until we make the test green!

Note that we should also write a test to handle the **error** scenario.

Unit Test

Example: add a method `#full_name` to the `User` model.

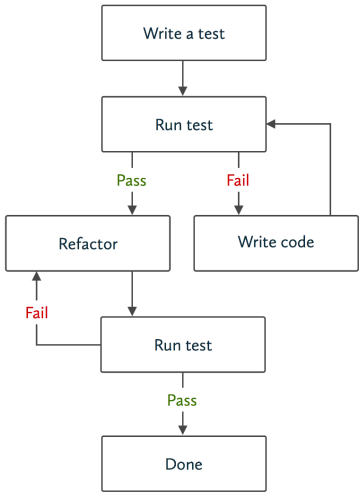
Let's use TDD!

```
touch test/models/user_test.rb
```

```ruby
# test/models/user_test.rb
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  test "full_name returns the capitalized fist name and last name" do
    user = User.new(first_name: "john", last_name: "lennon")
    assert_equal "John Lennon", user.full_name
  end
end
```

Time to implement the feature, following the test error messages!

```
rake test:models
```



Continuous Integration

Travis CI (https://travis-ci.org/)

```
# 1. We need a special file:
cat >.travis.yml <<YAML
language: ruby
before_script:
  - psql -c 'create database product_hunt_clone_test;' -U postgres
YAML
```

```
# 2. Let's commit our work
```

```
# 3. Let's create a GitHub repo
hub create
```

```
# 4. Go to travis-ci.org and add your repository.
```

```
# 5. Now let's push the code on GitHub:
git push origin master
```

```
# 6. Go back to Travis' Dashboard!
```

Travis integrates with Pull Request 😀

Going Further

- Must read: guides.rubyonrails.org/testing.html (http://guides.rubyonrails.org/testing.html)
- RSpec (http://rspec.info/)
- Webmock (https://github.com/bblimke/webmock)

- ...

One last thing

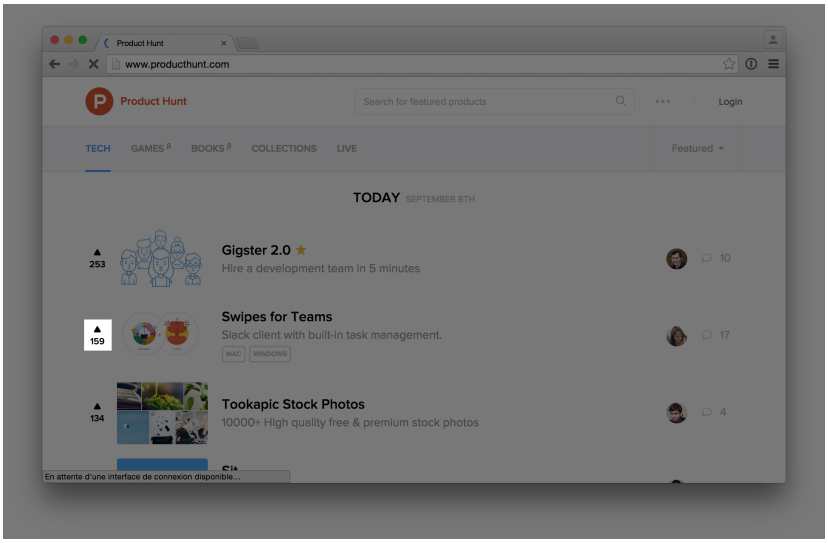
You need **Monitoring**!

- Uptime Robot (https://uptimerobot.com/)
- Error Tracking (https://elements.heroku.com/addons#errors-exceptions) (Try Raygun or AppSignal)

React (Video)

Why?

Suppose you want to code this upvote



HTML

```
<div class="upvote" data-id="1928382">
  <i class="fa fa-caret-up" />
  <span class="count">
    159
  </span>
</div>
```

JS

```
// app/assets/javascripts/WHERE.JS????
$(document).ready(function() {
  $('.upvote').on('click', function(e) {
    var element = $(this);
    $.ajax({
      type: 'POST',
      url: '/resources/' . element.data('id') . '/upvote',
      success: function(data) {
        element.find('.count').html(data.upvotes);
        if (data.upvoted) {
          element.addClass('upvoted');
        } else {
          element.removeClass('upvoted');
        }
      }
    })
  });
});
```

Rails way

You could use `remote: true` here on a regular `link_to`. You still have to write the `.js.erb` view for the callback.

Problems

- Where should I put my JavaScript?
- How can I reuse my upvote component on another page?
- State is everywhere, stored in HTML and CSS classes (no type)
- JS code mixes event handling, and presentation logic.

Use React when you have dynamic views.

React Basics

Component

- A `render` method
- Props (think immutable instance variables)
- State (think mutable instance variables)

Our first component

We use **JSX** (https://facebook.github.io/react/docs/jsx-in-depth.html). Mixing HTML & JS in the same file.

```
var Hello = React.createClass({
  render: function() {
    return <div>Hello guys</div>;
  }
});
```

We can render it with:

```
React.render(<Hello />, document.body);
```

Live example (http://jsbin.com/veluka/1/edit?html,js,output)

Adding a prop (immutable)

```
var Hello = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});
```

```
React.render(<Hello name="Boris" />, document.body);
```

Adding state

```
var Hello = React.createClass({
  getInitialState: function() {
    return { activated: false };
  },

  render: function() {
    var divClasses = classNames({
      'activated': this.state.activated
    });
    return <div className={divClasses}>Hello {this.props.name}</div>;
  }
});
```

Adding events

```
var Hello = React.createClass({
  getInitialState: function() {
    return { activated: false };
  },

  render: function() {
    var divClasses = classNames({
      'activated': this.state.activated
    });
    return <div onClick={this.activate} className={divClasses}>Hello {this.props.name}</div>;
  },

  activate: function() {
    this.setState({ activated: true });
  }
});
```

Live example (http://jsbin.com/veluka/7/edit?html,js,output)

Combining components

List Item pattern

```
var PostListItem = React.createClass({
  render: function() {
    return <div>
      <h3>{this.props.post.title}</h3>
      <p>{this.props.post.content}</p>
    </div>
  }
});
```

```
var PostList = React.createClass({
  render: function() {
    return <div>
      {this.props.posts.map(function(post) {
        return <PostListItem post={post} />;
      })}
    </div>;
  }
});
```

```
var posts = [
  { title: "The first post", content: "lorem ipsum" },
  { title: "The second post", content: "other lorem" }
]
React.render(<PostList posts={posts} />, document.body);
```

Live Example (http://jsbin.com/veluka/8/edit?html,js,output)

React + Rails

Setup

```
rails new \
  -T --database postgresql \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/master/devise.rb \
  product_hunt_clone_with_react
```

```
cd product_hunt_clone_with_react
```

We'll use react-rails (https://github.com/reactjs/react-rails), js-routes (https://github.com/railsware/js-routes) gems and classnames (https://github.com/JedWatson/classnames) t
package

```
# Gemfile
gem 'js-routes', '~> 1.3'
gem 'react-rails', '~> 1.8'

source 'https://rails-assets.org' do
  gem 'rails-assets-classnames'
end
```

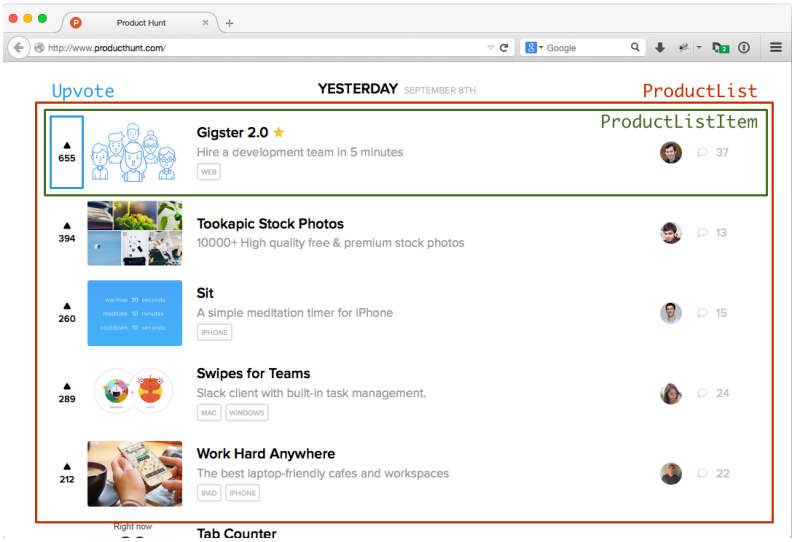
```
bundle install
rails g react:install
```

Config

```
// app/assets/javascripts/components.js

//= require js-routes
//= require classnames
//= require_tree ./components
```

```
# config/initializers/jsroutes.rb

# https://github.com/railsware/js-routes#advanced-setup
JsRoutes.setup do |config|
  # Whitelist routes to include on the Front-End
  # NOTE: if you add a new route here, do not forget to run:
  #       rake tmp:cache:clear
  #       before restarting your `rails s`.
  config.include = [

  ]
end
```

Thinking in components



Model

```
rails g model product name tagline url user:references
rails g migration AddAvatarUrlToUsers avatar_url:string
```

```
# Gemfile
gem 'acts_as_votable'
```

```
bundle install
rails generate acts_as_votable:migration
```

```
# app/models/product.rb
class Product < ApplicationRecord
  acts_as_votable
end
```

```
# app/models/user.rb
class User < ApplicationRecord
  acts_as_voter
end
```

```
# db/seeds.rb
puts 'Creating users...'
john = User.create!(email: 'john@beatles.com', password: 'testtest', avatar_url: 'http://vignette1.wikia.nocookie.net/peel/images/d/d6/John_
nnon.jpg/revision/latest?cb=20130925121148')
paul = User.create!(email: 'paul@beatles.com', password: 'testtest', avatar_url: 'http://beatlesthe.free.fr/img/paul.jpg')
ringo = User.create!(email: 'ringo@beatles.com', password: 'testtest', avatar_url: '')
george = User.create!(email: 'george@beatles.com', password: 'testtest', avatar_url: '')

puts 'Creating products...'
startup_stash = Product.create!(
  name: 'Startup Stash',
  tagline: 'A curated directory of 400 resources & tools for startups',
  url: 'http://startupstash.com',
  user: john
)

startup_launch_list = Product.create!(
  name: 'Startup Launch List',
  tagline: 'Articles you need to read before launching a startup',
  url: 'http://startuplaunchlist.com',
  user: paul
)

puts 'Creating votes...'
john.up_votes startup_stash
paul.up_votes startup_stash
george.up_votes startup_stash

ringo.up_votes startup_launch_list
puts 'Finished!'
```

Let's build the components

```
// app/assets/javascripts/components/product_list_item.js.jsx
var ProductListItem = React.createClass({
  render: function() {
    return (
      <div className="product">
        <div className="product-upvote">TODO</div>
        <div className="product-body">
          <h3>
            <a href={this.props.product.url} target="_blank">{this.props.product.name}</a>
          </h3>
          <p>{this.props.product.tagline}</p>
        </div>
        <div className="product-controls">
          <div className="product-control">
            <div className="user-badge-container ">
              <img src={this.props.product.user.avatar_url} className="avatar"/>
            </div>
          </div>
        </div>
      </div>
    );
  }
});
```

And add some css (https://raw.githubusercontent.com/lewagon/ui-components/master/source/stylesheets/components/_product_item.scss)

```
// app/assets/javascripts/components/product_list.js.jsx
var ProductList = React.createClass({
  render: function() {
    return (
      <div>
        {this.props.products.map(function(product){
          return <ProductListItem product={product} key={product.id} />;
        })}
      </div>
    );
  }
});
```

Building the props (with jbuilder)

```
# app/views/products/_product.json.jbuilder
json.extract! product, :name, :tagline, :url, :id

json.user do
  json.extract! product.user, :avatar_url
end

json.up_votes product.votes_for.count

if user_signed_in?
  json.up_voted current_user.voted_for? product
end
```

```
# app/views/products/index.json.jbuilder
json.products do
  json.array! @products do |product|
    json.partial! "products/product", product: product
  end
end
```

Controller

```
# config/routes.rb
Rails.application.routes.draw do
  root to: 'products#index'
end
```

```
rails g controller products
```

```
# app/controllers/products_controller.rb
class ProductsController < ApplicationController
  def index
    @products = Product.all
  end
end
```

View

```
<!-- app/views/products/index.html.erb -->
<%= react_component "ProductList",
      render(template: 'products/index.json.jbuilder') %>
```

You can prerender (for SEO)!

```
<!-- app/views/products/index.html.erb -->
<%= react_component "ProductList",
      render(template: 'products/index.json.jbuilder'),
      prerender: true %>
```

Let's upvote

```
# config/routes.rb
  resources :products, only: [] do
    member do
      post :upvote
    end
  end
```

```
# app/controllers/products_controller.rb
def upvote
  @product = Product.find(params[:id])
  if current_user.voted_for? @product
    current_user.unvote_for @product
  else
    current_user.up_votes @product
  end
end
```

```
# app/views/products/upvote.json.jbuilder
json.partial! "products/product", product: @product
```

JS Route

```
JsRoutes.setup do |config|
  config.include = [
    /^upvote_product$/
  ]
end
```

```
# Kill your rails s with Ctrl-C
rails tmp:cache:clear
rails s
```

```
// app/assets/javascripts/components/upvote.js.jsx
var Upvote = React.createClass({
  getInitialState: function() {
    return {
      product: this.props.product
    }
  },

  render: function() {
    var divClasses = classNames({
      "product-upvote": true,
      "product-upvote-upvoted": this.state.product.up_voted
    });

    return (
      <div className={divClasses}>
        <div className="product-arrow"></div>
        <div className="product-count">
          {this.state.product.up_votes}
        </div>
      </div>
    );
  }
});
```

With click event handling

```
// [...]
<div className={divClasses} onClick={this.handleClick}>
// [...]
```

```
// [...]
handleClick: function() {
  var that = this;
  $.ajax({
    type: 'POST',
    url: Routes.upvote_product_path(this.props.product.id, { format: 'json' }),
    success: function(data) {
      that.setState({ product: data });
    }
  });
}
// [...]
```

Advanced

Pubsub

To communicate between components *without direct filiation*.

```
# Gemfile
source 'https://rails-assets.org' do
  gem 'rails-assets-pubsub-js'
end
```

```
// app/assets/javascripts/components.js

//= require pubsub-js
```

Documentation on GitHub (https://github.com/mroderick/PubSubJS#pubsubjs)

The future

React Native (https://facebook.github.io/react-native/)