Rails From the Ground Up

with @nateberkopec

Rails is

bloated

Rails is over. It's a bloated meta framework that requires enormous amounts of peripheral knowledge to understand.

— **Hacker News**

[Lotus] aims to bring back Object Oriented Programming to web development

— Lotus web framework

Lotus is made of standalone frameworks (controllers, views, etc.)

— Lotus web framework

Lotus is lightweight, fast and testable.

— **Lotus web framework**

Rails is a massive project

- Rails: 49,104 commits and 2,551 contributors
- Sinatra: 2,664 commits and 225 contributors
- Node: 10,222 commits and 548 contributors
- Express: 4,974 commits and 165 contributors
- Ruby: 38,223 commits and 38 (?) contributors

"Rails will become more modular, starting with a rails-core, and including the ability to opt in or out of specific components."

— Yehuda Katz, 2008

Rails is bloated*

***If you let it.**

Lets build
a Rails app
in 15 lines!

What do we get when we use rails new?

- Empty folders, reminding us where Rails expects to find things
- Placeholder files like application.js and application.css,
  application.html.erb, the application helper and application controller, a
  locale file, seeds.rb.

What do we get when we use rails new?

- Public folder with a favicon, 404/500 pages, robots.txt
- Initializers and config files for different environments
- Gemfile
- Rakefile

What do we get when we use rails new *that matters?*

- config.ru
- config/routes.rb
- config/application.rb
- config/boot.rb
- config/environment.rb

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

Gemfile

```
source "https://rubygems.org"

gem "rails", "~> 4.2"
```

This puts a *lot* of stuff in the Gemfile.lock

```ruby
# normally happens in application.rb via "require 'rails/all'"
require "rails"
require "action_controller"
# require "active_record"
# require "action_view"
# require "action_mailer"
# require "active_job"
# require "rails/test_unit"
# require "sprockets"
# also happens in application.rb
class MyApp < Rails::Application
  # config/routes.rb
  routes.append { root "hello#world" }

  # We need a secret token for session, cookies, etc.
  # Usually via config/secrets.yaml
  config.secret_key_base = "insecure"
end
```

Recap

```ruby
require "rails"
require "action_controller"

class MyApp < Rails::Application
  routes.append { root "hello#world" }
  config.secret_key_base = "insecure"
end
```

```ruby
class HelloController < ActionController::Metal
  include AbstractController::Rendering
  include ActionController::Rendering

  def world
    render text: "Hello world!"
  end
end
```

ActionController::Metal

AbstractController::Rendering,AbstractController::Translation,AbstractController::AssetPaths,Helpers,HideActions,UrlFor,Redirecting,ActionView::Layouts,
Rendering,Renderers::All,ConditionalGet,EtagWithTemplateDigest,RackDelegation,Caching,MimeResponds,ImplicitRender,StrongParameters,Cookies,Flash,
RequestForgeryProtection,ForceSSL,Streaming,DataStreaming,HttpAuthentication::Basic::ControllerMethods,
HttpAuthentication::Digest::ControllerMethods,HttpAuthentication::Token::ControllerMethods,

AbstractController::Rendering

ActionController::Rendering

Recap

```ruby
require "rails"
require "action_controller"

class MyApp < Rails::Application
  routes.append { root "hello#world" }
  config.secret_key_base = "insecure"
end

class HelloController < ActionController::Metal
  include AbstractController::Rendering
  include ActionController::Rendering

  def world
    render text: "Hello world!"
  end
end
```

```ruby
require "rails"
require "action_controller"
class MyApp < Rails::Application
  routes.append { root "hello#world" }
  config.secret_key_base = "insecure"
end
class HelloController < ActionController::Metal
  include AbstractController::Rendering
  include ActionController::Rendering

  def world
    render text: "Hello world!"
  end
end
MyApp.initialize!
run MyApp
```

```ruby
# config/environment.rb
MyApp.initialize!

# config.ru
run MyApp
```

What do we get in return?

- Remote IP spoofing protection, timing attack prevention via *ActionDispatch::RemoteIp*
- Automatic reloading in development
- Environments
- Excellent logging (*ActionDispatch::RequestId*, *ActionDispatch::DebugExceptions*)
- Parameter parsing via *ActionDispatch::ParamsParser*
- Conditional GET (*Rack::ConditionalGet*)

What do we get in return?

- Caching (*Rack::Cache* and *Rack::ETag*)
- HEAD requests to GET via *Rack::Head*
- Resourceful routes
- URL generation and URL helpers
- Basic, Token, Digest HTTP auth
- A great instrumentation API
- Generators
- Incredibly simple extensibility
- Access to the Rails ecosystem (Engines, gems)

Memory differences (Thin)

- 40.1 MB lightweight Rails
- 70.7 MB stock Rails
- 26.7 MB Sinatra

Speed differences from stock Rails on a microbench

- Lightweight Rails ~10% faster
- Ultra Lightweight Rails ~90% faster (remove all middleware, log to stdout)
- Sinatra ~100% faster

But these differences are on the order of single-digit milliseconds. App code > Framework code.

Expanding: ActiveRecord

Expanding: ActionView

Expanding: Rails Server

Expanding: ActionMailer

Expanding: Tests

Which decisions *matter?*

Why is this modularity interesting?

- Improves your understanding of Rails internals
- Faster and uses less memory
- Win arguments with internet haters

Your homework

- Don't use rails/all
- Try starting from a single file the next time your start a Rails app

Bonus: tweet-length Rails apps

```ruby
require "rails/all"
run Class.new (Rails::Application) do
  routes.append{root to:proc{[200,{},[]]}}
end.initialize!
```

This example requires a secrets.yml and gemfile

Bonus: tweet-length Rails apps

```
%w[rails rack_test action_controller].map{|r|require r}
run Class.new (Rails::Application) do
  config.secret_key_base=1
  routes.append{root to:proc{[200,{},[]]}}
end.initialize!
```

This example can be run from a single file!