Miles more maintainable:

Building APIs with the middleware pattern

I'm @timrogers 

I'm from London 🇬🇧☔

I'm a Software Engineer at GoCardless

We're building an API-driven bank-to-bank payments network for the internet


VISA


mastercard


AMERICAN
EXPRESS

DIRECT Debit

bankgirot

SEPA
Single Euro Payments Area

```
POST /payments HTTP/1.1
Host: api.gocardless.com
Content-Type: application/json
```

Today, I'm going to talk about building APIs with Rails 🚂

```
rails new gocardless_api --api
```

SOFTWARE ENGINEERS HATE HIM

Man from London discovers how to build maintainable APIs with this one WEIRD TRICK

LEARN THE TRUTH NOW

Maintainability ❤️

Understandable ✅

Testable ✅

Reusable ✅

Adaptable ✅

Understandable ✅

Testable ✅

Reusable ✅

Adaptable ✅

Understandable ✅

Testable ✅

Reusable ✅

Adaptable ✅

Understandable ✅

Testable ✅

Reusable ✅

Adaptable ✅

Understandable ✅

Testable ✅

Reusable ✅

Adaptable ✅

Understandable ✅

Testable ✅

Reusable ✅

Adaptable ✅

Let's start by building a simple API using Rails 🛠️

```
POST /customers HTTP/1.1

Content-Type: application/json
Host: api.gocardless.com
```

```
{
  "data": {
    “email": "tim@gocardless.com",
    "iban": "GB60BARC20000005577991"
  }
}
```

```
class CustomersController < ApplicationController
  # ...
end
```

```ruby
class CustomersController < ApplicationController
  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```ruby
class CustomersController < ApplicationController
  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```ruby
class CustomersController < ApplicationController
  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```ruby
class CustomersController < ApplicationController
  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```ruby
class CustomersController < ApplicationController
  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```ruby
class CustomersController < ApplicationController
  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```
POST /customers HTTP/1.1

Authorization: Bearer my_access_token
Content-Type: application/json
Host: api.gocardless.com
```

```
{
  "data": {
    “email": "tim@gocardless.com",
    "iban": "GB60BARC20000055779911"
  }
}
```

```ruby
class CustomersController < ApplicationController
  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

Understandable ❌

Reusable ❌

Testable ❌

Understandable ❌

Reusable ❌

Testable ❌

Understandable ❌

Reusable ❌

Testable ❌

```
before_action :check_authorization_header

private
  def check_authorization_header
    # …
  end
```

before_action

after_action

around_action

```ruby
before_action :check_authorization_header

private

def check_authorization_header
  header_value = request.headers["HTTP_AUTHORIZATION"]

  return missing_access_token_error unless header_value.present?
  token_type, token = header_value.split(" ", 2)
  return missing_access_token_error unless token_type == "Bearer"

  @access_token = token
end

def missing_access_token_error
  render json: { errors: [I18n.t("errors.missing_access_token")] },
         status: 401
end
```

```ruby
before_action :check_authorization_header

private

def check_authorization_header
  header_value = request.headers["HTTP_AUTHORIZATION"]

  return missing_access_token_error unless header_value.present?
  token_type, token = header_value.split(" ", 2)
  return missing_access_token_error unless token_type == "Bearer"

  @access_token = token
end

def missing_access_token_error
  render json: { errors: [I18n.t("errors.missing_access_token")] },
         status: 401
end
```

```
POST /customers HTTP/1.1

Authorization: Bearer my_access_token
Content-Type: application/json
Host: api.gocardless.com
```

```
{
  "data": {
    “email": "tim@gocardless.com",
    "iban": "GB60BARC20000055779911"
  }
}
```

```ruby
before_action :check_authorization_header

private

def check_authorization_header
  header_value = request.headers["HTTP_AUTHORIZATION"]

  return missing_access_token_error unless header_value.present?
  token_type, token = header_value.split(" ", 2)
  return missing_access_token_error unless token_type == "Bearer"

  @access_token = token
end

def missing_access_token_error
  render json: { errors: [I18n.t("errors.missing_access_token")] },
         status: 401
end
```

```ruby
before_action :check_authorization_header

private

def check_authorization_header
  header_value = request.headers["HTTP_AUTHORIZATION"]

  return missing_access_token_error unless header_value.present?
  token_type, token = header_value.split(" ", 2)
  return missing_access_token_error unless token_type == "Bearer"

  @access_token = token
end

def missing_access_token_error
  render json: { errors: [I18n.t("errors.missing_access_token")] },
         status: 401
end
```

```ruby
before_action :check_authorization_header

private

def check_authorization_header
  header_value = request.headers["HTTP_AUTHORIZATION"]

  return missing_access_token_error unless header_value.present?
  token_type, token = header_value.split(" ", 2)
  return missing_access_token_error unless token_type == "Bearer"

  @access_token = token
end

def missing_access_token_error
  render json: { errors: [I18n.t("errors.missing_access_token")] },
         status: 401
end
```

```
before_action :check_authorization_header
before_action :check_access_token

private

def check_access_token
  @user = User.find_by(access_token: @access_token)
  return invalid_access_token_error unless @user.present?
end

def invalid_access_token_error
  render json: { errors: [I18n.t("errors.invalid_access_token")] },
         status: 401
end
```

```ruby
before_action :check_authorization_header
before_action :check_access_token

private

def check_access_token
  @user = User.find_by(access_token: @access_token)
  return invalid_access_token_error unless @user.present?
end

def invalid_access_token_error
  render json: { errors: [I18n.t("errors.invalid_access_token")] },
         status: 401
end
```

```ruby
before_action :check_authorization_header
before_action :check_access_token

private

def check_access_token
  @user = User.find_by(access_token: @access_token)
  return invalid_access_token_error unless @user.present?
end

def invalid_access_token_error
  render json: { errors: [I18n.t("errors.invalid_access_token")] },
         status: 401
end
```

```ruby
class CustomersController < ApplicationController
  before_action :check_authorization_header
  before_action :check_access_token

  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```
POST /customers HTTP/1.1

Accept-Language: uk-UA
Authorization: Bearer my_access_token
Content-Type: application/json
Host: api.gocardless.com
```

```
{
   "data": {
     “email": "tim@gocardless.com",
     "iban": “GB60BARC20000055779911"
   }
}
```

```
around_action :with_locale

def with_locale
  I18n.with_locale(request.headers["HTTP_ACCEPT_LANGUAGE"]) do
    yield
  end
end
```

```
around_action :with_locale

def with_locale
  I18n.with_locale(request.headers["HTTP_ACCEPT_LANGUAGE"]) do
    yield
  end
end
```

```
around_action :with_locale

def with_locale
  I18n.with_locale(request.headers["HTTP_ACCEPT_LANGUAGE"]) do
    yield
  end
end
```

Welcome to

before action

hell

```
around_action :with_locale, except: [:cancel]
before_action :only_allow_html, only: [:authorize]
before_action :build_oauth_params_from_params, only: [:authorize]
before_action :build_oauth_params_from_session, except: %i[authorize cancel]

# You would shiver to see the number of instance variables set in here...
before_action :set_instance_variables

before_action :check_client_id
before_action :check_redirect_uri
before_action :check_scope
before_action :check_response_type
before_action :redirect_to_cancel, except: %i[authorize cancel]

before_action :persist_oauth_params_to_session, only: [:authorize]
before_action :set_permitted_params
```

```
around_action :with_locale, except: [:cancel]
before_action :only_allow_html, only: [:authorize]
before_action :build_oauth_params_from_params, only: [:authorize]
before_action :build_oauth_params_from_session, except: %i[authorize cancel]

# You would shiver to see the number of instance variables set in here...
before_action :set_instance_variables

before_action :check_client_id
before_action :check_redirect_uri
before_action :check_scope
before_action :check_response_type
before_action :redirect_to_cancel, except: %i[authorize cancel]

before_action :persist_oauth_params_to_session, only: [:authorize]
before_action :set_permitted_params
```

Action-specific logic

vs.

Reusable logic

What's the problem? 🤔

Understandable ❌

Reusable ❌

Testable ❌

Understandable ❌

Reusable ❌

Testable ❌

Understandable ❌

Reusable ❌

Testable ❌

The single responsibility principle

Understandable ✅

Reusable ✅

Testable ✅

"Every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class"

Is there a Rails way?

Move our filters from `CustomersController` to `ApplicationController`

```ruby
class ApplicationController < ActionController::Base
  before_action :check_authorization_header
  before_action :check_access_token

  private

  def check_authorization_header
    # ...
  end

  def check_access_token
    # ...
  end

  def with_locale
    # ...
  end
end

class CustomersController < ApplicationController
  around_action :with_locale
end
```

```ruby
class ApplicationController < ActionController::Base
  before_action :check_authorization_header
  before_action :check_access_token

  private

  def check_authorization_header
    # ...
  end

  def check_access_token
    # ...
  end

  def with_locale
    # ...
  end
end

class CustomersController < ApplicationController
  around_action :with_locale
end
```

```
class ApplicationController < ActionController::Base
  before_action :check_authorization_header
  before_action :check_access_token

  private

  def check_authorization_header
    # ...
  end

  def check_access_token
    # ...
  end

  def with_locale
    # ...
  end
end

class CustomersController < ApplicationController
  around_action :with_locale
end
```

Use a plain old Ruby module or an `ActiveSupport::Concern`

```ruby
module API::AuthorizationHeader
  def check_authorization_header
    # ...
  end
end

class CustomersController < ApplicationController
  include API::AuthorizationHeader

  before_action :check_authorization_header
end
```

```ruby
module API::AuthorizationHeader
  def check_authorization_header
    # ...
  end
end
```

```ruby
class CustomersController < ApplicationController
  include API::AuthorizationHeader

  before_action :check_authorization_header
end
```

```ruby
module API::AuthorizationHeader
  def check_authorization_header
    # ...
  end
end
```

```ruby
class CustomersController < ApplicationController
  include API::AuthorizationHeader

  before_action :check_authorization_header
end
```

```ruby
module API::AuthorizationHeader
  def check_authorization_header
    # ...
  end
end
```

```ruby
class CustomersController < ApplicationController
  include API::AuthorizationHeader

  before_action :check_authorization_header
end
```

```ruby
module API::AuthorizationHeader
  extend ActiveSupport::Concern

  included do
    before_action :check_authorization_header
  end

  def check_authorization_header
    # ...
  end
end

class CustomersController < ApplicationController
  include API::AuthorizationHeader
end
```

```ruby
module API::AuthorizationHeader
  extend ActiveSupport::Concern

  included do
    before_action :check_authorization_header
  end

  def check_authorization_header
    # ...
  end
end

class CustomersController < ApplicationController
  include API::AuthorizationHeader
end
```

```ruby
module API::AuthorizationHeader
  extend ActiveSupport::Concern

  included do
    before_action :check_authorization_header
  end

  def check_authorization_header
    # ...
  end
end

class CustomersController < ApplicationController
  include API::AuthorizationHeader
end
```

Use a filter class

```ruby
module API::AuthorizationHeader
  def self.before(controller)
    header_value = controller.request.headers["HTTP_AUTHORIZATION"]

    return missing_access_token_error unless header_value.present?
    token_type, access_token = header_value.split(" ", 2)
    return missing_access_token_error unless token_type == "Bearer"

    controller.access_token = access_token
  end

  def self.missing_access_token_error(controller)
    controller.render json: { errors: ["Access token not provided"] },
                      status: 401
  end
end

class CustomersController < ApplicationController
  attr_writer :access_token

  before_action API::AuthorizationHeader
end
```

```ruby
module API::AuthorizationHeader
  def self.before(controller)
    header_value = controller.request.headers["HTTP_AUTHORIZATION"]

    return missing_access_token_error unless header_value.present?
    token_type, access_token = header_value.split(" ", 2)
    return missing_access_token_error unless token_type == "Bearer"

    controller.access_token = access_token
  end

  def self.missing_access_token_error(controller)
    controller.render json: { errors: ["Access token not provided"] },
                      status: 401
  end
end

class CustomersController < ApplicationController
  attr_writer :access_token

  before_action API::AuthorizationHeader
end
```

```ruby
module API::AuthorizationHeader
  def self.before(controller)
    header_value = controller.request.headers["HTTP_AUTHORIZATION"]

    return missing_access_token_error unless header_value.present?
    token_type, access_token = header_value.split(" ", 2)
    return missing_access_token_error unless token_type == "Bearer"

    controller.access_token = access_token
  end
  def self.missing_access_token_error(controller)
    controller.render json: { errors: ["Access token not provided"] },
                      status: 401
  end
end

class CustomersController < ApplicationController
  attr_writer :access_token

  before_action API::AuthorizationHeader
end
```

```ruby
module API::AuthorizationHeader
  def self.before(controller)
    header_value = controller.request.headers["HTTP_AUTHORIZATION"]

    return missing_access_token_error unless header_value.present?
    token_type, access_token = header_value.split(" ", 2)
    return missing_access_token_error unless token_type == "Bearer"

    controller.access_token = access_token
  end

  def self.missing_access_token_error(controller)
    controller.render json: { errors: ["Access token not provided"] },
                      status: 401
  end
end

class CustomersController < ApplicationController
  attr_writer :access_token

  before_action API::AuthorizationHeader
end
```

```ruby
module API::AuthorizationHeader
  def self.before(controller)
    header_value = controller.request.headers["HTTP_AUTHORIZATION"]

    return missing_access_token_error unless header_value.present?
    token_type, access_token = header_value.split(" ", 2)
    return missing_access_token_error unless token_type == "Bearer"

    controller.access_token = access_token
  end
  def self.missing_access_token_error(controller)
    controller.render json: { errors: ["Access token not provided"] },
                      status: 401
  end
end

class CustomersController < ApplicationController
  attr_writer :access_token

  before_action API::AuthorizationHeader
end
```

```ruby
module API::AuthorizationHeader
  def self.before(controller)
    header_value = controller.request.headers["HTTP_AUTHORIZATION"]

    return missing_access_token_error unless header_value.present?
    token_type, access_token = header_value.split(" ", 2)
    return missing_access_token_error unless token_type == "Bearer"

    controller.access_token = access_token
  end
  def self.missing_access_token_error(controller)
    controller.render json: { errors: ["Access token not provided"] },
                      status: 401
  end
end

class CustomersController < ApplicationController
  attr_writer :access_token

  before_action API::AuthorizationHeader
end
```

The Rails way doesn't get us where we want to be 😪

Let's see how the middleware pattern can help us ⭐

"Middleware is software that's assembled into an application pipeline to handle requests and responses"

Each middleware can:

Choose whether to pass the request to the next middleware in the pipeline

Perform work before and after the next middleware in the pipeline is invoked


JS

What's so great about the middleware pattern?

The single responsibility principle

Understandable ❌

Reusable ❌

Testable ❌

Understandable ❌

Reusable ❌

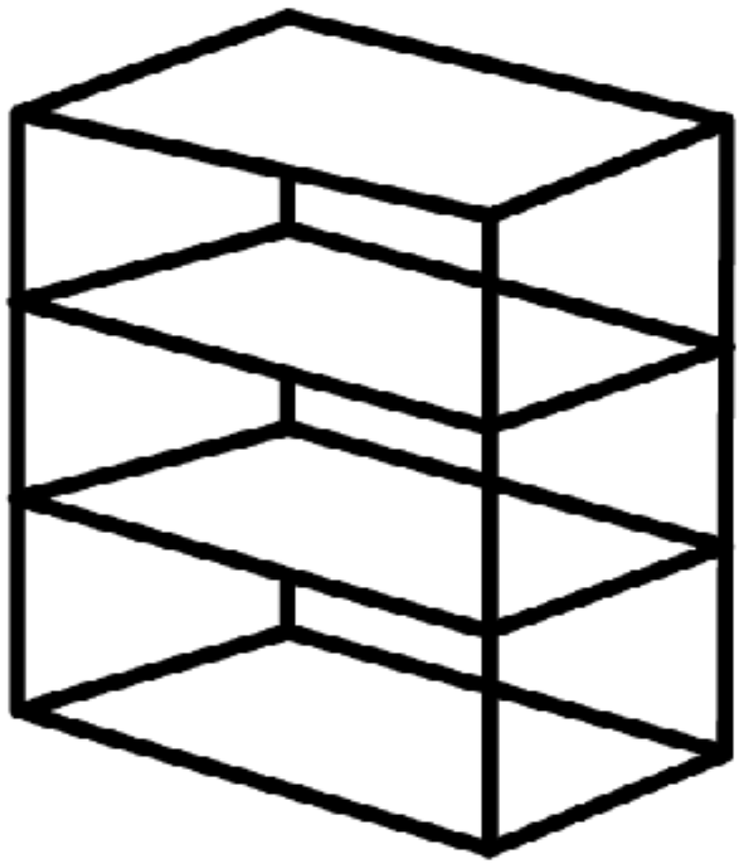
Testable ❌

Understandable ❌

Reusable ❌

Testable ❌

Hidden from view, middleware are actually behind how Rails works 👻



rack

powers web applications

```ruby
require "securerandom"

module ActionDispatch
  class RequestId
    X_REQUEST_ID = "X-Request-Id".freeze

    def call(env)
      req = ActionDispatch::Request.new env
      req.request_id = internal_request_id
      @app.call(env).tap do |_status, headers, _body|
        headers[X_REQUEST_ID] = req.request_id
      end
    end

    private

    def internal_request_id
      SecureRandom.uuid
    end
  end
end
```

```ruby
require "securerandom"

module ActionDispatch
  class RequestId
    X_REQUEST_ID = "X-Request-Id".freeze

    def call(env)
      req = ActionDispatch::Request.new env
      req.request_id = internal_request_id
      @app.call(env).tap do |_status, headers, _body|
        headers[X_REQUEST_ID] = req.request_id
      end
    end

    private

    def internal_request_id
      SecureRandom.uuid
    end
  end
end
```

```ruby
require "securerandom"

module ActionDispatch
  class RequestId
    X_REQUEST_ID = "X-Request-Id".freeze

    def call(env)
      req = ActionDispatch::Request.new env
      req.request_id = internal_request_id
      @app.call(env).tap do |_status, headers, _body|
        headers[X_REQUEST_ID] = req.request_id
      end
    end

    private

    def internal_request_id
      SecureRandom.uuid
    end
  end
end
```

```ruby
require "securerandom"

module ActionDispatch
  class RequestId
    X_REQUEST_ID = "X-Request-Id".freeze

    def call(env)
      req = ActionDispatch::Request.new env
      req.request_id = internal_request_id
      @app.call(env).tap do |_status, headers, _body|
        headers[X_REQUEST_ID] = req.request_id
      end
    end

    private

    def internal_request_id
      SecureRandom.uuid
    end
  end
end
```

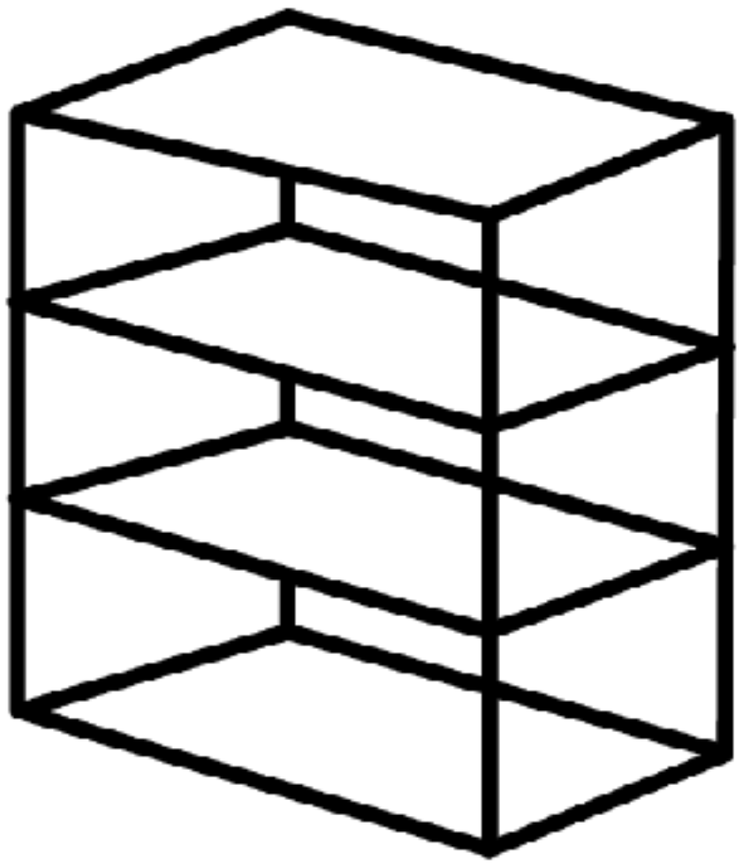
```ruby
require "securerandom"

module ActionDispatch
  class RequestId
    X_REQUEST_ID = "X-Request-Id".freeze

    def call(env)
      req = ActionDispatch::Request.new env
      req.request_id = internal_request_id
      @app.call(env).tap do |_status, headers, _body|
        headers[X_REQUEST_ID] = req.request_id
      end
    end

    private

    def internal_request_id
      SecureRandom.uuid
    end
  end
end
```



rack

powers web applications

Rails doesn't make it easy for us to plug middleware in on a per-request basis :(

```
config.middleware.insert_after ActionDispatch::RequestId,
MyFancyMiddleware
```

Let's see how we can refactor our API using middleware

The middleware pattern in about 25 lines of Ruby ⚡

https://github.com/timrogers/simple_middleware

```
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```ruby
module SimpleMiddleware
  class Middleware
    # @param [#call] the next middleware in the chain, which can be anything
    #   that responds to #call, accepting the current state, an
    #   `Immutable::Hash`, as a parameter
    def initialize(next_middleware)
      @next_middleware = next_middleware
    end

    # Runs the middleware, either returning a result (probably a Rack response)
    # or calling the next middleware in the chain, giving it the opportunity to
    # return a result
    #
    # @param [Immutable::Hash] the current state
    def call(state)
      raise NotImplementedError
    end

    private

    attr_reader :next_middleware

    def render(status:, headers: [], body: nil)
      [status, headers, body]
    end
  end
end
```

```ruby
module SimpleMiddleware
  class Middleware
    # @param [#call] the next middleware in the chain, which can be anything
    #   that responds to #call, accepting the current state, an
    #   `Immutable::Hash`, as a parameter
    def initialize(next_middleware)
      @next_middleware = next_middleware
    end

    # Runs the middleware, either returning a result (probably a Rack response)
    # or calling the next middleware in the chain, giving it the opportunity to
    # return a result
    #
    # @param [Immutable::Hash] the current state
    def call(state)
      raise NotImplementedError
    end

    private

    attr_reader :next_middleware

    def render(status:, headers: [], body: nil)
      [status, headers, body]
    end
  end
end
```

```ruby
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```
#<Middleware::Locale @next_middleware=
  #<Middleware::AuthorizationHeader @next_middleware=
    #<Middleware::AccessToken @next_middleware=
     #<Create @next_middleware=nil>
    >
  >
>
```

```ruby
require "immutable/hash"

module SimpleMiddleware
  # Runs a chain of middlewares with the provided initial state, returning the
  # result of the chain of middlewares
  #
  # @param [Hash, Immutable::Hash] the initial state to be passed into the chain of
  #   middlewares
  # @param [Array<SimpleMiddleware::Middleware>] the middlewares to be run
  def self.call(initial_state: {}, middlewares:)
    middleware_chain = middlewares.reverse.reduce(nil) do |next_middleware, middleware|
      middleware.new(next_middleware)
    end

    middleware_chain.call(Immutable::Hash.new(initial_state))
  end
end
```

```ruby
require "immutable/hash"

module SimpleMiddleware
  # Runs a chain of middlewares with the provided initial state, returning the
  # result of the chain of middlewares
  #
  # @param [Hash, Immutable::Hash] the initial state to be passed into the chain of
  #   middlewares
  # @param [Array<SimpleMiddleware::Middleware>] the middlewares to be run
  def self.call(initial_state: {}, middlewares:)
    middleware_chain = middlewares.reverse.reduce(nil) do |next_middleware, middleware|
      middleware.new(next_middleware)
    end

    middleware_chain.call(Immutable::Hash.new(initial_state))
  end
end
```

```ruby
require "immutable/hash"

module SimpleMiddleware
  # Runs a chain of middlewares with the provided initial state, returning the
  # result of the chain of middlewares
  #
  # @param [Hash, Immutable::Hash] the initial state to be passed into the chain of
  #   middlewares
  # @param [Array<SimpleMiddleware::Middleware>] the middlewares to be run
  def self.call(initial_state: {}, middlewares:)
    middleware_chain = middlewares.reverse.reduce(nil) do |next_middleware, middleware|
      middleware.new(next_middleware)
    end

    middleware_chain.call(Immutable::Hash.new(initial_state))
  end
end
```

```ruby
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```
#<Middleware::Locale @next_middleware=
  #<Middleware::AuthorizationHeader @next_middleware=
    #<Middleware::AccessToken @next_middleware=
     #<Create @next_middleware=nil>
    >
  >
>
```

```
#<Middleware::Locale @next_middleware=
  #<Middleware::AuthorizationHeader @next_middleware=
    #<Middleware::AccessToken @next_middleware=
     #<Create @next_middleware=nil>
    >
  >
>
```

```
#<Middleware::Locale @next_middleware=
  #<Middleware::AuthorizationHeader @next_middleware=
    #<Middleware::AccessToken @next_middleware=
      #<Create @next_middleware=nil>
    >
  >
>
```

```
#<Middleware::Locale @next_middleware=
  #<Middleware::AuthorizationHeader @next_middleware=
    #<Middleware::AccessToken @next_middleware=
      #<Create @next_middleware=nil>
    >
  >
>
```

https://github.com/timrogers/simple_middleware

Time to refactor our API using middleware 🎉

https://github.com/timrogers/simple_middleware_example

```ruby
class CustomersController < ApplicationController
  around_action :with_locale
  before_action :check_authorization_header
  before_action :check_access_token

  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```ruby
class CustomersController < ApplicationController
  around_action :with_locale
  before_action :check_authorization_header
  before_action :check_access_token

  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```ruby
require "rails_helper"

RSpec.describe CustomersController, type: :controller do
  # ...
end
```

```ruby
class CustomersController < ApplicationController
  around_action :with_locale
  before_action :check_authorization_header
  before_action :check_access_token

  def create
    customer = Customer.new(customer_params.merge(user: @user))

    if customer.save
      render json: customer
    else
      render json: { errors: customer.errors.full_messages }, status: 422
    end
  end

  private

  def customer_params
    params.require(:data).permit(:email, :iban)
  end
end
```

```
class API::AuthorizationHeader < SimpleMiddleware::Middleware
  def call(state)
    # ...
  end
end
```

```
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```ruby
def call(state)
  header_value = state[:request].headers["HTTP_AUTHORIZATION"]
  # ...
end
```

```ruby
def call(state)
  header_value = state[:request].headers["HTTP_AUTHORIZATION"]

  return missing_access_token_error unless header_value.present?
  token_type, token = header_value.split(" ", 2)
  return missing_access_token_error unless token_type == "Bearer"

  # ...
end

private

def missing_access_token_error
  render status: 401,
         headers: { "Content-Type" => "application/json" },
         body: JSON.generate(errors: [I18n.t("errors.missing_access_token")])
end
```

```ruby
def call(state)
  header_value = state[:request].headers["HTTP_AUTHORIZATION"]

  return missing_access_token_error unless header_value.present?
  token_type, token = header_value.split(" ", 2)
  return missing_access_token_error unless token_type == "Bearer"

  # ...
end

private

def missing_access_token_error
  render status: 401,
         headers: { "Content-Type" => "application/json" },
         body: JSON.generate(errors: [I18n.t("errors.missing_access_token")])
end
```

```
[
  401,
  { "Content-Type" => "application/json" },
  "{\"errors\":[\"Missing access token\"]}"
]
```

```
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```ruby
def call(state)
  header_value = state[:request].headers["HTTP_AUTHORIZATION"]

  return missing_access_token_error unless header_value.present?
  token_type, token = header_value.split(" ", 2)
  return missing_access_token_error unless token_type == "Bearer"

  new_state = state.put(:access_token, token)
  next_middleware.call(new_state)
end
```

```
Immutable::Hash[
 :access_token => "your_access_token",
 :request => #<ActionDispatch::Request>,
 :params => #<ActionController::Parameters>
]
```

```ruby
def call(state)
  header_value = state[:request].headers["HTTP_AUTHORIZATION"]

  return missing_access_token_error unless header_value.present?
  token_type, token = header_value.split(" ", 2)
  return missing_access_token_error unless token_type == "Bearer"

  new_state = state.put(:access_token, token)
  next_middleware.call(new_state)
end
```

```ruby
require "rails_helper"

RSpec.describe Middleware::AuthorizationHeader do
  # ...
end
```

```ruby
subject(:instance) { described_class.new(next_middleware) }
let(:next_middleware) { double(call: true) }

let(:state) { Immutable::Hash.new(request: request) }
let(:headers) { {} }
let(:request) do
  instance_double(ActionDispatch::Request, headers: headers)
end
```

```ruby
subject(:instance) { described_class.new(next_middleware) }
let(:next_middleware) { double(call: true) }

let(:state) { Immutable::Hash.new(request: request) }
let(:headers) { {} }
let(:request) do
  instance_double(ActionDispatch::Request, headers: headers)
end
```

```
let(:instance) { described_class.new(next_middleware) }
let(:next_middleware) { double(call: true) }

let(:state) { Immutable::Hash.new(request: request) }
let(:headers) { {} }
let(:request) do
  instance_double(ActionDispatch::Request, headers: headers)
end
```

```ruby
context "with no Authorization header" do
  let(:headers) { {} }

  it "renders an error" do
    expect(instance.call(state)).to eq([
      401,
      { "Content-Type" => "application/json" },
      "{\"errors\":[\"Missing access token\"]}"
    ])
  end
end
```

```
context "with an invalid Authorization header" do
  let(:headers) { { "HTTP_AUTHORIZATION" => "foo bar" } }

  it "renders an error" do
    expect(instance.call(state)).to eq([
      401,
      { "Content-Type" => "application/json" },
      "{\"errors\":[\"Missing access token\"]}"
    ])
  end
end
```

```
context "with a correctly-structured Authorization header" do
  let(:headers) { { "HTTP_AUTHORIZATION" => "Bearer your_access_token" } }

  it "calls the next middleware, passing on the access token" do
    expect(next_middleware).to receive(:call).
      with(Immutable::Hash.new(request: request,
                               access_token: "your_access_token"))

    instance.call(state)
  end
end
```

```
context "with a correctly-structured Authorization header" do
  let(:headers) { { "HTTP_AUTHORIZATION" => "Bearer your_access_token" } }

  it "calls the next middleware, passing on the access token" do
    expect(next_middleware).to receive(:call).
      with(Immutable::Hash.new(request: request,
                               access_token: "your_access_token"))

    instance.call(state)
  end
end
```

```ruby
class Middleware::AccessToken < SimpleMiddleware::Middleware
  def call(state)
    user = User.find_by(access_token: state[:access_token])

    return invalid_access_token_error unless user.present?

    next_middleware.call(state.put(:user, user))
  end

  private

  def invalid_access_token_error
    # ...
  end
end
```

```ruby
class Middleware::AccessToken < SimpleMiddleware::Middleware
  def call(state)
    user = User.find_by(access_token: state[:access_token])

    return invalid_access_token_error unless user.present?

    next_middleware.call(state.put(:user, user))
  end

  private

  def invalid_access_token_error
    # ...
  end
end
```

```ruby
class Middleware::AccessToken < SimpleMiddleware::Middleware
  def call(state)
    user = User.find_by(access_token: state[:access_token])

    return invalid_access_token_error unless user.present?

    next_middleware.call(state.put(:user, user))
  end

  private

  def invalid_access_token_error
    # ...
  end
end
```

```ruby
require "rails_helper"

RSpec.describe Middleware::AccessToken do
  let(:instance) { described_class.new(next_middleware) }
  let(:next_middleware) { double(call: true) }
  let(:state) { Immutable::Hash.new(access_token: access_token) }

  # ...
end
```

```
context "with a non-existent access token" do
  let(:access_token) { "dummy_access_token" }

  it "renders an error" do
    expect(instance.call(state)).to eq([
      401,
      { "Content-Type" => "application/json" },
      "{\"errors\":[\"Invalid access token\"]}"
    ])
  end
end
```

```
context "with a valid access token" do
  let(:user) { FactoryBot.create(:user) }
  let(:access_token) { user.access_token }

  it "calls the next middleware, passing on the user” do
    expect(next_middleware).to receive(:call).
      with(Immutable::Hash.new(access_token: access_token, user: user))

    instance.call(state)
  end
end
```

```
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```
def create
  response = SimpleMiddleware.call(initial_state: { request: request,
                                                    params: params },
                                   middlewares: [Middleware::Locale,
                                                 Middleware::AuthorizationHeader,
                                                 Middleware::AccessToken,
                                                 Create])

  render_rack_response(response)
end
```

```
class Create < SimpleMiddleware::Middleware
  def call(state)
    customer_params = build_customer_params(state)
    customer = Customer.new(customer_params)

    # ...
  end
end
```

```
def build_customer_params(state)
  state[:params].
    require(:data).
    permit(:email, :iban).
    merge(user: state[:user])
end
```

```
def build_customer_params(state)
  state[:params].
    require(:data).
    permit(:email, :iban).
    merge(user: state[:user])
end
```

```ruby
def build_customer_params(state)
  state[:params].
    require(:data).
    permit(:email, :iban).
    merge(user: state[:user])
end
```

```
def call(state)
 customer_params = build_customer_params(state)
 customer = Customer.new(customer_params)

 if customer.save
   render status: 201,
          headers: { "Content-Type" => "application/json" },
          body: customer.to_json
   else
     render status: 422,
          headers: { "Content-Type" => "application/json" },
          body: JSON.generate(errors: customer.errors.full_messages)
   end
end
```

```ruby
def call(state)
 customer_params = build_customer_params(state)
 customer = Customer.new(customer_params)

 if customer.save
   render status: 201,
          headers: { "Content-Type" => "application/json" },
          body: customer.to_json
 else
   render status: 422,
          headers: { "Content-Type" => "application/json" },
          body: JSON.generate(errors: customer.errors.full_messages)
 end
end
```

```ruby
def call(state)
 customer_params = build_customer_params(state)
 customer = Customer.new(customer_params)

 if customer.save
   render status: 201,
          headers: { "Content-Type" => "application/json" },
          body: customer.to_json
   else
     render status: 422,
          headers: { "Content-Type" => "application/json" },
          body: JSON.generate(errors: customer.errors.full_messages)
   end
end
```

```
require "rails_helper"

RSpec.describe CustomersController, type: :controller do
  # ...
end
```

https://github.com/timrogers/simple_middleware_example

We've learnt to make APIs miles more maintainable with middleware

But I wouldn't *use* Simple Middleware

We can do *way* better in terms of developer experience

But using the simplest implementation possible helps you to understand the concepts

At GoCardless, we built Coach.

```
class Routes::Customers::Create < Coach::Middleware
  uses Middleware::Locale
  uses Middleware::AuthorizationHeader
  uses Middleware::AccessToken

  requires :user

  def call
    # ...
  end
end
```

```
GoCardless::Application.routes.draw do
  match "/customers",
        to: Coach::Handler.new(Routes::Customers::Create),
        via: :post
end
```

```ruby
class API::Middleware::AccessToken < Coach::Middleware
  requires :access_token
  provides :user

  def call
    user = validate_access_token!(access_token)
    return invalid_permissions_error unless user.scope == config[:scope]

    provide(user: user)
    next_middleware.call
  end
end
```

```ruby
class Routes::Customers::Create < Coach::Middleware
  uses API::Middleware::AuthorizationHeader
  uses API::Middleware::AccessToken, required_permissions: "admin"

  requires :user

  def call
    # ...
  end
end
```

```ruby
class API::Middleware::AuthorizationHeader < Coach::Middleware
  provides :access_token

  def call
    # ...

    provide(access_token: access_token)
    next_middleware.call
  end
end

class API::Middleware::AccessToken < Coach::Middleware
  requires :access_token
  provides :user

  def call
    user = validate_access_token!(access_token)

    provide(user: user)
    next_middleware.call
  end
end

class Routes::Customers::Create < Coach::Middleware
  uses API::Middleware::AuthorizationHeader
  uses API::Middleware::AccessToken

  requires :user

  def call
    # ...
  end
end
```

```
class Routes::Customers::Create < Coach::Middleware
  uses API::Middleware::AuthorizationHeader
  uses API::Middleware::AccessToken

  requires :user

  def call
    # ...
  end
end
```

```
it { is_expected.to call_next_middleware }
it { is_expected.to respond_with_status(422) }
```

https://github.com/gocardless/coach

We want our code to be as maintainable as possible ❤️

That is, we want it to be understandable, testable and reusable ✅

Controller filters quickly get out of control

The Single Responsibility Principle points us in the right direction

The middleware pattern can help us build more maintainable APIs

Using Coach, you can hit the ground running with the middleware pattern

We're hiring 👨‍👩‍👧‍👦

https://gocardless.com/jobs

GOCARDLESS

Thanks! 🐣

Grab the slides at https://timrogers.co.uk,
and feel free to chat to me in person or
drop me a tweet - I'm @timrogers