

MerbAuth

Darwinian Authentication



In The Beginning

- ⦿ Conditions for Authentication were ripe
- ⦿ Merb
- ⦿ Plugins
- ⦿ Rubigen with scopes (6th Nov 2007)
- ⦿ Restful Authentication was missing



merbful_authentication

merbful_authentication

- Released 4th January 2008
- Direct port of Restful Authentication
- Supported DataMapper & ActiveRecord
- Supported RSpec & Test::Unit

Pros

- First plugin with multi-ORM support in Merb
- A lot of applications used it

Cons

- ⌚ Generated code
- ⌚ Very complex
- ⌚ Maintenance sucked

The Catalyst

- ➊ Slices were born



Original merb-auth

merb-auth

- ⦿ Mostly direct port of merbful_authentication
- ⦿ Used brand new slices plugin
- ⦿ Moved to Library Code not Generated
- ⦿ ORM support via mixins
- ⦿ Forgotten Passwords
- ⦿ 17 June 2008

merb-auth - Pros

- Live code. Not generated
- Minimal Application Configuration
- Implemented as a slice
- Easier to maintain (still sucked)

merb-auth - Cons

- ⌚ User model hidden
- ⌚ Hard to please all through configuration
- ⌚ Unclear how to customize it
- ⌚ Tied to one model type
- ⌚ Dictates user model
- ⌚ Extensions difficult (No OpenID)
- ⌚ Difficult to change logic

Evolutionary Step
Required

The Catalyst

- ⦿ Adam French proposed:
 - ⦿ Authenticating Sessions
 - ⦿ Simple session based api
 - ⦿ Using Exceptions to refuse entry
 - ⦿ Provides correct status code

ExceptionalAuthentication

- ⦿ Adam created a prototype
- ⦿ ExceptionalAuthentication
- ⦿ Application including his proposals

Session API

- ➊ session.authenticated?
- ➋ session.authenticate!
- ➌ session.user
- ➍ session.user=
- ➎ session.abandon!

Exceptional Authentication

- ➊ Originally a DataMapper based system
- ➋ Decided to allow arbitrary “user” objects



Code Named - Mauth

Mauth - What is it?

- ⦿ Authentication Framework
- ⦿ Cascading authentication “Strategies”
- ⦿ Authenticates user objects
- ⦿ Now MerbAuth (in merb-plugins)
- ⦿ Supports user objects such as
 - ⦿ DM, AR, Sequel, Hash, String, File, IO, or just plain old Object

MerbAuth - What it's Not

- ➊ A user management system
- ➋ Tell MerbAuth how to store and retrieve the User object from the session

```
class Authentication
  def fetch_user(session_info)
    User.get(session_info)
  end

  def store_user(user)
    user.nil? ? user : user.id
  end
end
```

MerbAuth

- Uses cascading Strategies
- Strategy is a class that implements run!
- Each strategy is run in order
- Success == First Strategy to return object
- Failure == No Strategies return object
- Only run strategies if !session.authenticated?

What is a Strategy?

- Sub-class of Authentication::Strategy

```
class PasswordStrategy < Authentication::Strategy

  def run!
    if params[:password] && params[:login]
      User.authenticate!(params[:login], params[:password])
    end
  end

end
```

- Re-order Authentication.default_strategy_order

- Declare many. One for each login type

Protect Actions

- Controller helper :ensure_authenticated

```
class MyController < Application
  before :ensure_authenticated
  #...
end
```

- Control Which Strategies are used

```
class MyController < Application
  before :ensure_authenticated, :with => [OpenId, Password]
  #...
end
```

What Happens on Fail?

- ➊ Raises Unauthenticated
- ➋ Uses Merbs Exception Handling
 - ➌ Exceptions#unauthenticated
- ➍ Sets correct status code

Failure Messages

- ➊ `Authentication#error_message (Overwrite)`
- ➋ `before :ensure_authenticated, :with => { :message => "Fail" }`
- ➌ `session.authentication.errors.add(:label, "message")`
- ➍ `error_messages_for session.authentication`

How To Login?

- Setup login form at `Exceptions#unauthenticated`
- Protect a method and try to access it

How to Logout?

- session.abandon!

Sessions Controller Example

```
class Sessions < Application
  before :ensure_authenticated

  def create
    redirect_back_or url(:home)
  end

  def destroy
    session.abandon!
    redirect url(:home)
  end
end
```

Routes

```
to(:controller => "exceptions")
  match("/login", :method => :get).
    to(:action => "unauthenticated").
      name(:login)
end

to(:controller => "sessions") do
  match("/login", :method => :post).to(:action => "create")
  match("/logout", :method => :delete).to(:action => "destroy").
    name(:logout)
end
```

What Strategies Are There?

- ⦿ Password based form login
- ⦿ Open ID
- ⦿ Basic Authentication
- ⦿ Require a default strategy to load / define it
- ⦿ Monkey patch parts of it you need to change

Advanced Strategies

- YourStrategy#redirect!
- YourStrategy.abstract!
- Overwrite YourStrategy#user_class for different types of users

Mixins

- Use mixins to extend your User or MerbAuth
- Salted User mixin
- Redirect Mixin (`redirect_back_or`)

What About Users

- You should make your own user class
 - Default password based login requires
`<User>.authenticate(login, password)`
 - Salted Password Mixin (DM, AR & Sequel)

Make it Easy

- Slices are being written to make it easy for people who want a drop in user / authentication solution
- So Far
 - Password Slice
 - Activation Slice

Overview of Setup

- require 'merb_auth-core'
- Define <User> model
- Setup session storage
- Declare strategies
- Protect methods
- Setup login / logout actions

Where next?

- ➊ Implement slices
- ➋ Implement meta gem to include sensible defaults

Resources

- http://adam.speaksoutofturn.com/articles/authentication_vs_authorization.html
- http://github.com/wycats/merb-plugin/merb_auth
- <http://github.com/ck/cookbook>
- <http://github.com/RichGuk/merb-auth-example>