

TP : écriture de middleware pour Rack

Archive middleware.tgz :

```
test.ru
middleware/
├── lib
│   ├── rack_cookie_session.rb
│   ├── rack_debug.rb
│   └── rack_session.rb
└── my_middleware.rb
```

pour tester les 2 middlewares et l'application : `test.ru`

```
<pre 'code'>
require 'rack/request'
require 'rack/response'
$: << File.join(File.dirname(__FILE__), "middleware")
require 'my_middleware'

use RackCookieSession
use RackSession
run RackDebug.new
```

application rack :

```
<pre 'code'>
class RackDebug
```

```
def call(env)
  req = Rack::Request.new(env)
  req.session["history"] ||= []
  req.session["history"] << [req.request_method, req.path_info, req.query_string]
  res = Rack::Response.new
  res.write "cookie= #{req.cookies.inspect}</br>"
  res.write "rack.session.id=#{env["rack.session.id"]}</br>"
  res.write "rack.session=#{env["rack.session"].inspect}</br>"
  res.finish
end

end
```

middleware_init/lib/rack_debug.rb

Faites fonctionner l'application :

rackup test.ru

Middleware RackCookieSession

Le but est de positionner automatiquement un cookie pour stocker un identifiant de session.

Les conventions à utiliser :

- l'identifiant de session sera stocké dans `env["rack.session.id"]`
- utilisez la methode fournie `generate_session_id` pour la génération de l'identifiant de session
- le cookie pour stocker l'identifiant de session est nommé `session_id`

```
<pre 'code'>
class RackCookieSession

  def initialize(app, env_session_id_key="rack.session.id", cookie_name="session_id")
    @app = app
    @env_session_id_key = env_session_id_key
    @cookie_name = cookie_name
  end

  def call(env)
    # new session or not
    # if new : generate session_id
    # if not : extract_session_id
    # store session id into env with the key env_session_id_key
    status, headers, body = @app.call(env)
    # set cookie in headers if necessary
```

```
[status, headers, body]
end

def extract_session_id(env)
  # get session id from cookie named @cookie_name
end

def generate_session_id(bit_size=32)
  rand(2**bit_size - 1)
end

end
```

middleware_init/lib/rack_cookie_session.rb

Middleware RackSession :

Le but est d'implémenter un conteneur de session avec un Hash.

- rendre la session courante accessible par `env["rack.session"]` (voir `Rack::Request`)

```
<pre 'code'>
class RackSession
  attr_reader :pool

  def initialize(app)
    @app = app
    @pool = Hash.new
  end

  def call(env)
    load_session(env)
    status, headers, body = @app.call(env)
    commit_session(env)
    [status, headers, body]
  end

  def load_session(env)
    # load session in env["rack.session"]
  end

  def commit_session(env)
    # save session into @pool
  end
end

```

```
end  
end
```

middleware_init/lib/rack_session.rb

Middleware rack et sinatra

Installez Sinatra :

```
gem install sinatra
```

Utilisez les middlewares précédents :

```
<pre 'code'>
require 'sinatra'
$: << File.join(File.dirname(__FILE__), "middleware")
require 'my_middleware'

use RackCookieSession
use RackSession

get '/' do
  # request : Rack::Request
  request.session["history"] ||= []
  request.session["history"] << [request.request_method, request.path_info, request.query_string]
  result = "cookie= #{request.cookies.inspect}</br>"
  result << "rack.session.id=#{env["rack.session.id"]}</br>"
  result << "rack.session=#{env["rack.session"].inspect}</br>"
end
```

sinatra_test.rb

Tester :

```
ruby sinatra_test.rb
```

Gestion de l'authentification

Nous allons utiliser le mécanisme de session pour coder une gestion d'authentification d'utilisateur.

Implémentez les routes suivantes grâce à Sinatra :

- GET /session/new : renvoie un formulaire permettant de renseigner un login/mot de passe
- POST /session, vérifie le login/mot de passe, et redirige sur /protected en cas de succès, en positionnant dans la

Fichier de départ : sinatra_authentication.rb

```
<pre 'code'>
require 'sinatra'
$: << File.join(File.dirname(__FILE__), "", "middleware")
require 'my_middleware'

use RackCookieSession
use RackSession

helpers do
  def current_user
    session["current_user"]
  end

  def disconnect
    session["current_user"] = nil
  end
end

get '/' do
  if current_user
    "Bonjour #{current_user}"
  else
    '<a href="/sessions/new">Login</a>'
  end
end

```

session la variable
current_user

- GET /protected :
teste si current_user
a une valeur dans la
session et affiche un
message, sinon
redirige sur GET
/session/new

Pour les redirection,
utilisez :

```
redirect '/other/path'
```

```
end  
get '/sessions/new' do  
  erb : "sessions/new"  
end
```

sinatra_authentication.rb

Les vues

Les vues sont à mettre dans le répertoire `views`.

Pour le formulaire de création de session, utilisez par exemple `views/sessions/new.erb`

```
<pre 'code'>
<!DOCTYPE html>
<html>
<head>
  <title>login page</title>
</head>
<body>

<h1>New session</h1>

<form action="/sessions" method="post">
  <div class="field">
    <label for="session_login">Login</label><br />

    <input id="session_login" name="login" size="30" type="text" />
  </div>
  <div class="field">
    <label for="session_password">Password</label><br />
    <input id="session_password" name="password" size="30" type="password" />
  </div>
  <div class="actions">
    <input name="commit" type="submit" value="Create Session" />
  </div>
</form>
</body>
</html>
</pre>
```

```
</div>  
</form>  
  
</body>  
</html>
```

authentication/views/sessions/new.erb