Tester une application rack/sinatra

Il faut pouvoir simuler un dialogue HTTP. Pour cela, on utilise rack-test

```
1 require 'rack/test'
2
3 # Rack::Test est maintenant disponible !
4
```

Rack::Test::Methods

Les méthodes `get`, `post`, `put`, `delete` et `head` sont disponibles dans ce module. Chacune simule la requête HTTP eponyme.

Ces méthodes s'utilisent de la manière suivante :

```
1   get '/path', params={}, rack_env={}
2
```

- `/path` : chemin de la ressource ciblée par la requête, avec éventuellement des options (`?param1=val1&&param2=val1,val2`)

- `params` : hash codant les paramètres (requête GET ou POST)
- `rack_env` : hash représentant l'environnement rack, qui peut être utilisé pour positionner des headers ou d'autres informations relatives à la requête, comme des données de session. Voir la spec de Rack pour les clefs/valeurs possibles.

Après l'appel d'une de ces méthodes, vous avez accès à :

- `last_response` : une instance de Rack::MockResponse
- `last_request` : une instance de Rack::MockRequest

Configuration

Il faut récupérer la définition des méthodes `get`, `post`, `put`, `delete` et `head` :

```
1  include Rack::Test::Methods
2
```

Il faut positionner l'application Rack à tester pour qu'elle soit disponible en appelant la méthode app.

Application Sinatra

```
1 def app
2   Sinatra::Application
3 end
4
```

Configuration

Application Rack

```
def app
  MyRackApp.new(options)
end

```

Middleware

```
def app
  Rack::Builder.app do
    use MiddlewareToTest, options
    run lambda{ |env| [404, {'env' => env}, ['HELLO']] }
  end
end

```

Exemple

```ruby
require 'hello_world'  # <-- your sinatra app
require 'spec'
require 'rack/test'

set :environment, :test

describe 'The HelloWorld App' do
  include Rack::Test::Methods

  def app
    Sinatra::Application
  end

  it "says hello" do
    get '/'
    last_response.should be_ok
    last_response.body.should == 'Hello World'
  end
end

```

TP

À vous de réaliser le middleware rack `AuthApi` dont le but est le suivant :

- si le header `HTTP_AUTHORIZATION` est positionné, les identifiants passés par ce headers doivent être vérifiés par l'appel d'un code externe.
- le header `HTTP_AUTHORIZATION` est effacé dans tous les cas
- si les identifiants sont corrects, la clef `rack.api.user` est positionnée à un Hash contenant un identifiant unique de l'utilisateur.
- le middleware peut être initialisé avec des options permettant de fixer la classe et la méthode permettant de faire la recherche de l'utilisateur.

Schéma général de l'interface souhaité avec la classe codant l'authentification : `User.find(key)` doit renvoyer un `Hash` contenant au moins une clef pour stocker l'identifiant de l'utilisateur, par exemple `{ :uid => 'bob', :info => { :name => "Bob l'éponge" }}`

À vous de fixer le schéma du Hash qui est renvoyé et stocké ensuite dans `env['rack.api.user']`

Le nom de la classe, la méthode à appeler, la clef permettant de stocker l'identifiant dans le Hash doivent être des paramètres pour l'initialisation du

middleware.

La clef à passer dans l'appel `User.find(key)` est le mot de passe contenu dans l'entête `HTTP_AUTHORIZATION`.

Récupérez l'archive rach-auth-api.tgz qui contient le squelette du middleware.

Utilisation du middleware

Utilisez ce middleware dans l'application sinatra développée dans le précédent TP. Spécifiez puis modifiez cette application pour obtenir le comportement suivant :

- si le middleware renvoie un identifiant correct, créez automatiquement une session en cas de besoin
- sinon retombez sur le comportement précédent (création de session par vérification de l'identité soumise par un formulaire)

Authentification HTTP Basic

Le principe est d'utiliser l'entête `HTTP_AUTHORIZATION` pour stocker des identifiants login/mot de passe. Le codage de l'entête est le suivant :

```
Basic: encodage_base64("login:pass")
Basic: bG9naW46cGFzcw
```

Pour coder/décoder en base 64 en ruby, on peut utiliser les méthodes `String#unpack` et `Array#pack`

```
1 ["login:pass"].pack('m')
2 => "bG9naW46cGFzcw==\n"
3 "bG9naW46cGFzcw==\n".unpack('m')
4 => ["login:pass"]
5
```