Open Source Frameworks for Rapid Application Development

Marek Krętowski
Krzysztof Bandurski, Tomasz Łukaszuk, Tomasz Rybak

Software Departament
Faculty of Computer Science
Bialystok University of Technology

m.kretowski@pb.edu.pl
k.bandurski@pb.edu.pl, t.lukaszuk@pb.edu.pl, t.rybak@pb.edu.pl

Lecture topic

Sessions

Sessions: Table of content

Introduction

HTTP as stateless protocol

- HTTP is stateless
- It means that each request should be treated the same way
- It makes it fast
  - Allows caching
  - Allows any server from the farms to create response
- It causes problems with storing data between requests
- E.g. whether user is authenticated and who one is
- Connection details (IP, User Agent, locality settings, plugin versions) cannot be used to distinguish between users

Authentication

- To store who the user is
- To verify those claims
- To ensure that user is authorised to perform particular actions on particular objects
- Authentication requires sessions management to be enabled to work
- It has many parts:
  - Users
  - Permissions
  - Groups
  - Messages
  - Resources

Django authentication

Enabling authentication in Django

1. Enable session framework
2. Add “django.contrib.auth” application to INSTALLED_APPS after sessions application
3. Run manege.py syncdb to create tables storing users information
4. Add django.contrib.auth.middleware.AuthenticationMiddleware to MIDDLEWARE_CLASSES after sessions middleware classes

Managing users

- manage.py createsuperuser –username name –email email
- Asks for password and other details of user
- Helper class django.contrib.auth.models.UserManager

create_user(username, email, password)
make_random_password(length, allowed_chars)

- Default value of allowed_chars does not contain neither letter I and digit 1, nor letter O and digit 0

```
1  user.groups = [group1, group2]
2  from django.contrib.auth.models import User
3  user = User.objects.create_user('tomasz', 't.rybak@pb.edu.pl', 'haslo')
4  user.save()
```

- user.save() is not needed as create_user saves user object to database
- Of course if we change user after creation, explicit saving is necessary

User model fields

Class django.contrib.auth.models.User

username	required, 30 characters or less
first_name	
last_name	
email	
password	required, hash of password
is_staff	boolean, whether user can access Admin module
is_active	boolean; is not checked during login process
is_superuser	boolean
last_login	timestamp
date_joined	timestamp

User methods

is_authenticated()
is_anonymous()
get_full_name()
set_password(pwd)
set_unusable_password() changes hash of password to impossible value, practically disallowing login for user
has_usable_password(p)
check_password(pwd)
get_all_permissions()
has_perm(perm) permission is in form application.permission
has_perms(perms)
has_module_perms(module)
get_and_delete_messages()
email_user(subject, body, from_email)
get_profile() returns additional data for user

Many-to-many relationships of User class

groups user belongs to

user_permissions set of all permissions

- Those two are Python collection
- They can be fully replaced
- One can call methods on those collections

```
1  user.groups.add(administrators)
2  user.groups.remove(staff)
3  user.user_permissions.clear()
```

Permissions

- Custom permissions can be defined in class Meta of the model class
- They describe permissions for this particular class

```
1  class Articles(model.Models):
2      class Meta:
3          permissions = (('name', 'description'),
4      ('commenting', 'User can comment on articles')
5      )
```

Logging in and out of users

authenticate(user, password) returns User object or None

login(request, user) attaches User object to request

- Creates all cookies, etc.
- Function authenticate() must be called before login()

logout(request) does not raise error when user was not logged in

- Clears session state

```
1  from django.contrib.auth import authenticate
2  user = authenticate(username, password)
3  if user is None:
4  # Failed
5  else:
6      if not user.is_active:
7  # Account disabled
8      else:
9  # Everything OK
10         login(request, user)
```

Password storage

- Password is not stored in raw form
- Instead hash of the password is stored
  - crypt (legacy Unix)
  - MD5
  - SHA1
- To defend from attacks using precalculated hashes or rainbow tables each password is salted before hashing
- hashtype$salt$hashvalue
- E.g. sha1$abcd$aabbcc

Anonymous users

- AnonymousUser class implements User interface from models.User
- is_active, is_staff, is_superuser return False
- Collections groups and user_permissions are empty
- is_authenticated and has_perms return False
- Password methods raise NotImplementedError exception
- Storage methods (save, delete) raise NotImplementedError exception

User profile

- Profile can be used to store additional data about user
- Profile class is just a model class
- It must have foreign key pointing to the User class
- Django needs to know which class contains user profile
- AUTH_PROFILE_MODULE = 'application.ProfileClass'
- This will cause method get_profile() to return profile object, instance of class application.models.ProfileClass

View decorator

- In case of many applications most pages require for user to be logged in
- Repeating code checking for login status in each view function violated DRY principle
- Django authors provide us with function solving this problem
- django.contrib.auth.decorators.login_required
- If user is logged in view is executed without any changes
- If not, browser is redirected to login page
- User is redirected to settings.LOGIN_URL page
- After successful login user is redirected to previous page
- URL is passed in parameter which name is passed in parameter “redirect_field_name”

Views available for logging purposes

- Available in module django.contrib.auth.views
- login(req, template_name, redirect_field_name)
- Default template is registration/login.html
- Template receives four variables

form	Form representing login form
next	URL to redirect to when login is successful
site	
site_name	

- logout(req, template_name, redirect_field_name)
  - Default template is registration/logged_out.html
- logout_then_login(req, login_url)
  - Logs a user out, then redirects to the login page

Password-related views

- password_change(req, template_name, post_change_redirect)
  - Default template is registration/password_change_form.html
- password_change_done(req, template_name)
  - Default template is registration/password_change_done.html
- password_reset()
- password_reset_done(req, template_name)
- password_reset_confirm()
- password_reset_complete(req, template_name)

Forms used in logging process

AdminPasswordChangeForm

AuthenticationForm

PasswordChangeForm

PasswordResetForm

SetPasswordForm

UserChangeForm

UserCreationForm

Sessions in Django

Storage of state

- Need to solve problem with statelesness of HTTP
- Need to store information between requests
- Usually cookies are used to do it
- Cookie can store little bits of information in the browser
- They are send by browser on each request to the server
- They are set of key=value pairs

Cookies

- Cookies are used to store identity and other details of identity of user on particular site
- This means that they can be used to trace users
- It also means that getting cookie of logged user one can impersonate him or her
- There are rules of sending cookies for sites by browsers
- Same-domain policy
  - example.com
  - *.example.com
- User agents need not to store cookies
- User can remove or change stored cookies

Cookie in HTTP

- HTTP response contains header field “Set-Cookie” containing cookie from server to this particular client
- Client returns cookie to the server in header “Cookie”
- Django stores cookies in request.COOKIES dictionary
- Key of dictionary is used as key of dictionary
- Value contains cookie object
- It means that each cookie can have different expirations date, path and domain, etc.
- This is used by many frameworks

Cookie properties

max_age	number of seconds cookie will last
expires	date cookie will expire
path	path in URL cookie is valid for
domain	domain cookie is valid for
secure	boolean, if True cookie is only send over encrypted connection

Sending cookies in the response

- response.set_cookie(key, value)
- Additional parameters can be added to manage life of cookies
- We are responsible for sending back in response all cookies that were send in the request

Testing for cookies

- As noted previously not all clients need to support cookies
- Although those that ignore them are rather rare
- Nonetheless one needs to check whether client supports cookies
- Methods in request.session

set_test_cookie()

test_cookie_worked() need to be called in the next response

delete_test_cookie()

Sessions in Django

- Management of cookies, values, etc. can be tedious
- Django provides us with functionality of managing sessions

1. Add application “django.contrib.sessions” to INSTALLED_APPS
2. Add django.contrib.sessions.middleware.SessionMiddleware to MIDDLEWARE_CLASSES
3. Run manege.py syncdb to create tables storing session data

Sessions in Django

- request.session dictionary
- Contains pairs key/value
- Key should be string
- Strings starting with underscore are reserved for Django usage
- Session dictionary should not be replaced with another object nor any attributes of it should be accessed
- Values from session dictionay can be removed by using “del” keyword
- Django uses cookies to retrieve correct session object from database and attach it to request of particular user

Sessions in Django

- Session is model class defined in django.contrib.sessions.models
- Primary key identifying particular session object is sent to user in the cookie
- Session object is saved to database after each modification of the value stored in dictionary
- Only direct modifications are detected and cause save of session object

expire_date

session_data dictionary encoded to textual format

get_decoded() returns session object

Session configuration

SESSION_EXPIRE_AT_BROWSER_CLOSE boolean, causes cookie identifying Django session to disappear after browser window is closed

SESSION_COOKIE_DOMAIN domain to tie session cookie to

SESSION_COOKIE_NAME name of the cookie value storing identity of the session (primary key); default is “sessionid”

SESSION_COOKIE_SECURE whether “secure” cookies should be used to store session identity

SESSION_SAVE_EVERY_REQUEST boolean, if set to True Django will save session to database every time, even if nothing was changed

Session in RoR

- As in Django in RoR session is dictionary-like structure allowing for storing data for particular client between requests
- Each session is identified by session_id
- session_id is 32 characters generated by MD5 based
  - time
  - constant string
  - random value
- Session is not full dictionary (hash) — not all methods work

Session management

- Session can be disabled in particular controllers
- This can save memory and processing time
- We can enable session for some controllers
- Or only for only chosen actions

```
1  class MyController < ActionController::Base
2      session :off
3  end
4
5  class OtherController < ActionController::Base
6      session :on, :only => [:create, :update]
7  end
```

Session usage

- Session behaves as an hash
- Keys are strings or constants
- Values can be any type
- If value is not needed it can be removed using “delete” method
- Setting value to “nil” also causes deleting from session
- reset_session() removes all object from session object

```
1  session[:key] = value
2
3  session[:name] = 'Rybak'
4  session[:email] = 'rybak@example.com'
5  session[:messages] = ['Line1', 'Line2']
6
7  session.delete :messages
```

Session storage

- Session values for popular site can take much space
- Management of large amount of data can be challenging
- Proper management may mean difference between working site and site that users must wait for any action
- Ruby on Rails offers four different ways of storing session data
- It is chosen for entire application in "config.action_controller.session_store"

CookieStore stores data on client; cookie size limit of 4kB applies

DRBStore

MemCacheStore

ActiveRecordStore session stored in database

```
1  config.action_controller.session_store = :drb_store
2  config.action_controller.session_store = :active_record_store
```

Flash session

- Variable called “flash”
- It is special part of session
- Like “session” it is hash storing key/value pairs
- Values stored in “flash” are cleared after next request
- To make them available in the current session (e.g. to use flash values in rendering of other action) use “flash.now”
- flash.keep causes values to be persisted to use in next requests
- This function accepts optional list of names to persist

```
1  flash.now[:key] = value
2  flash.keep(:kay)
3  config.action_controller.session_store = :drb_store
4  config.action_controller.session_store = :active_record_store
```

Usage of cookies in sessions

- All sessions store session_id in the cookie
- ActionController::SessionManagement
- Data stores in the cookie store is signed (so tampering is very hard) but is not encrypted, so anyone can read it

Cookies

- As noted earlier, cookies are stored on the client
- Cookie size limit is 4kB
- Client need not to store or return cookies
- Variable “cookies” behaves as an hash, but does not support all dictionary methods
- It is similar to variable “session” in this regard

```
1  cookies[:key] = value
2
3  cookies[:name] = 'Rybak'
4
5  cookies.delete(:messages)
```

Cookie creation

- When creating cookies it is possible to set parameters managing their lifetime options

value	
path	default “/”
domain	
expires	
secure	whether to send cookie only over HTTPS

```
1   cookies[:name] = {:value => 'Rybak', :path = '/administration'}
```

RoR authentication

- Usually controller uses “before_filter” to ensure authorisation of user
- Session is used to store state of the user
- In most cases one implements custom authentication code
- Or uses additional module

```
1  class MyController < ActionController::Base
2      before_filter :require_login
3
4      def require_login
5          if session[:logged] == 1
6      return
7  fi
8       if username == 'tomasz' && password == 'topsecret'
9      session[:logged] = 1
10 fi
11 redirect
12     end
13 end
```

HTTP Basic authentication

- HTTP allows us to use authentication
- Exact messages in previous lectures
- authenticate_or_request_with_http_basic
- It is necessary not to store plain-text password
-

```
1  def authenticate
2      authenticate_or_request_with_http_basic do |username, password|
3          username == USERNAME && Digest::MD5.hexdigest(password) == PASSWORD
4      end
5  end
```

Security

Attacks

- There is many different attacks in the internet
- Most of them are trying to steal identity of the used on one of the sites
- Of course the most valuables are bank sites
- But email and blog accounts are also nice
- Community portals are nice point of interest
- Online games allow for monetizing virtual goods

Session hijacking

- When user is logged in attacker can steal his session
- One of possibilities is to steal cookie with session id
- JavaScript has access to the cookies
- If domain or path are set to generously rogue site can receive cookie with session id
- Embedding iframe in valid site can lead to leak of session
- When user was logged in and site does not clear session immediately after logout, attacker can use old cookie (reply attack)

Session from other source

- Session fixation
- Attacker creates session (and knows its id)
- This session id is send to the victim
- Victim logs in
- Attacker knows id of valid logged in session
- To protect from this attack it is advised to replace old session (either id or entire session) with new one after successful login

Cross Site Request Forgery

- CSRF
- Attacker embeds command causing action on attacked site on another site
- This way when user clicks link or image command is send for example to banking site
- To protect from it in RoR use “:verify” and “protect_from_forgery”

Cross Site Scripting

- XSS
- Application accepts data from external source
- User input is not validated
- It then puts this data into page sent to other users
- This can be done directly or (more dangerous) this data might be saved to database
- This allows for the attacker to put JavaScript or other code to be embedded in the pages displayed to the victims
- Always validate and escape user input!