Seshat Documentation

Release 2.0.0

Joshua P Ashby

May 07, 2014

Contents

Seshat is a toy web framework built by JoshAshby over the past few years. It's aimed at being somewhat opinionated, and most definetly full of bad practices but it gets the job done with running a few smaller sites.

Build status - Master: Build status - Dev: Gittip if you like the work I do and would consider a small donation to help fund me and this project:

CHAPTER 1

A Few Minor Warnings

1. I have litterally NO clue what I am doing. Use at your own risk.
2. I'm only a second year university student, and software isn't even my major; I'm working towards an Electrical and Computer Engineering degree, so not only do I have limited time to keep this maintained, but I also probably won't write the best code ever.
3. This project follows the semantic versioning specs. All Minor and patch versions will not break the major versions API, however an bump of the major version signifies that backwards compatibility will most likely be broken.

CHAPTER 2

Quick Start

Getting started is fairly easy, take a look at the included *example.py*:

```
from waitress import serve
import seshat.dispatch as dispatch

from seshat.route import route
from seshat.controller import Controller
from seshat.actions import NotFound


@route()
class index(Controller):
  def GET(self):
    name = self.request.get_param("name", "World!")
    return "Hello, " + name


@route()
class wat(Controller):
  def GET(self):
    return Redirect("/?name=Wat")


serve(dispatch.dispatch)
```

This starts a full web app on port 8080 that you can navigate your browser to localhost that will serve a basic page displaying the text "Hello, World". Navigating to localhost:8080/wat will redirect you back to the index, with the name now as "Wat".

2.1 Contributing

All code for this can be found online at github. If something is broken, or a feature is missing, please submit a pull request or open an issue. Most things I probably won't have time to get around to looking at too deeply, so if you want it fixed, a pull request is the way to go. Besides that, I'm releasing this under the GPLv3 License as found in the `LICENSE.txt` file. Enjoy!

2.2 Doc Contents

2.2.1 controller

No app built with Seshat does much without controllers. This module provides a base controller class which can be used right away in its current state, or can be inherited from to create more advanced or custom controllers.

Basic use is like so:

```
from seshat.controller import Controller

class index(Controller):
  def GET(self):
    return "<h1>WAT</h1>"
```

By default all controllers have HEAD and GET methods. HEAD simply calls GET but strips the reponse body. (HEAD is probably broken for now but I'll fix it eventually).

class seshat.controller.**Controller**(*request=None*, *session=None*)

The parent of all controllers which Seshat will serve.

To use this to make a controller, override or add the request method (in all caps) which will be called for this controller. Eg:

```
from seshat.controller import Controller

class index(Controller):
  def GET(self):
    return "<h1>WAT</h1>"
```

then all GET based requests to this controller will return with the text *<h1>WAT</h1>* however all POST, PUT, DELETE calls will a 405 Method Not Supported error.

post_init_hook()

Called at the end of *__init__* this allows you to customize the creation process of your controller, without having to override *__init__* itself.

This should accept nothing and return nothing.

pre_content_hook()

Called before the request method is called and should return either *None* or Action object.

If there is a returned value other than None, this will skip calling the request method and simply return directly to dispatch, so make sure it returns an Action.

A good example of the use for this hook would be for authentication. You could for example, check a parameter set through a cookie and return something like a 401 Unauthorized if the param doesn't represent a logged in user:

```
return actions.Unauthorized()
```

Return type Action or *None*

post_content_hook(*content*)

Gets called after the content generating request method has been called. This can be to further modify the content which is returned, or perform some other action after each request.

Parameters **content** (*str*) – the content from the content generating request method that was called.

Returns The original or modified content

Return type *str*

HEAD()
Will be called if the request method is HEAD

By default this will call *GET()* but return nothing, so that only the Headers are returned to the client.

GET()
Will be called if the request method is GET

2.2.2 route

Along with needing to have controllers, an app also has to have routes to those controllers. This is accomplished through the `route()` decorator. This decorator will, if no arguments are supplied, use `controller_folder` and the file hierarchy of the controller to auto generate a route. Or you can optionally supply a custom url pattern. This pattern then gets converted to a *dict* when the url is matched. The structure of the url pattern is simple, variabled within the url are denoted by a colon. For example, in the pattern:

```
/user/:name/:email
```

the returned *dict* will be:

```
{"name": something,
 "email": something_else}
```

seshat.route.**controller_folder** = ''
The folder where the controllers are located in. Since the auto route generation uses folder hierarchy, this setting allows to you to have controllers in a single folder but not have that folder end up as the route prefix.

seshat.route.**route**(*r=None*, *s=None*)
Class decorator that will take and generate a route table entry for the decorated `Controller` class, based off of its name and its file hierarchy if no route pattern is specified.

Use like so:

```
from seshat.controller import Controller
from seshat.route import route

@route()
class index(Controller):
    pass
```

which will result in a route for "/" being made for this controller.

controllers whose name is *index* automatically get routed to the root of their folders, so an index controller in "profiles/" will have a route that looks like "/profiles"

Controllers whose name is *view* will automatically get routed to any index route that has an attached ID. Eg:

```
# In folder: profiles/
class view(Controller):
  pass
```

will be routed to if the request URL is "/profiles/5" and the resulting id will be stored in `Controller.request.url_params`

class seshat.route.**Route**(*controller=None*, *route=None*, *subdomain=None*)
Provides a base route table entry which is generated by the `route()` decorator described below.

controller = None
The controller object, of type `Controller` which this route represents

Type `Controller`

class `seshat.route_table.`**RouteTable**

add(*container*)
Adds the given route container to the route table.

Parameters **r_container** (`Route`) – The route container which contains the url and controller for a route.

get(*request*)
Attempts to find the closest match to the given url, through comparing lots of regexs for a match against the url.

Parameters **request** (*urlparse.ParseResult*) – The requested url

Returns `Controller` or *None*

2.2.3 actions

Actions allow you to write code that looks like:

```
class RandomController(Controller):
  def GET(self):
    return Redirect("/")
```

This module provides a few common Action classes to use, along with a base Action class which can be inherited to create your own Actions by overriding the *__init__* function.

class `seshat.actions.`**Action**
Provides a base for creating a new object which represents an HTTP Status code.

All returned data is checked if it is of type `Action` and if so, the data/actions head is returned rather than the controllers head. This allows for a syntax like:

```
return NotFound()
```

which will cause the controller to return a 404 status code.

To create a new action, inherit this class then make a new *__init__(self, *kargs)* which sets *self.response* to a `Reponse` object (or just call super), and adds any headers or status changes to that `Response` object.

class `seshat.actions.`**Redirect**(*loc*)
Returns a 303 See Other status code along with a *location* header back to the client.

Parameters **loc** (*str*) – The location to which the client should be redirect to

class `seshat.actions.`**BadRequest**
Returns a 400 BAD REQUEST

class `seshat.actions.`**Unauthorized**
Returns a 401 UNAUTHORIZED back to the client

This should probably also include a WWW-Authenticate header, but I'll leave that for later right now.

class `seshat.actions.`**Forbidden**
Returns a 403 FORBIDDEN

class seshat.actions.**NotFound**
Returns a 404 Not Found

class seshat.actions.**MethodNotAllowed**(*allow*)
Returns a 405 METHOD NOT ALLOWED

Parameters **allow** (*list*) – A *list* of allowable methods

class seshat.actions.**InternalServerError**(*e=None*, *tb=None*)
Returns a 500 INTERNAL SERVER ERROR

Parameters

- **e** (*Exception*) – The Exception
- **tb** (*str*) – The traceback of the exception

2.2.4 request

class seshat.request.**FileObject**(*file_obj*)
Provides a File like object which supports the common file operations, along with providing some additional metadata which is sent from the client.

read()

readline()

seek(*where*)

readlines()

auto_read()

class seshat.request.**Request**(*env*)
Represents the request from the server, and contains various information and utilities.

env = None
The raw environ *dict* which is provided by the wsgi server.

method = None
The HTTP method by which the request was made, in all caps.

remote = None
The clients IP, otherwise *Unknown IP* This is set from HTTP_X_REAL_IP which is a header that I personally have nginx append to the request before handing it to Seshat.

headers = None
Headers which were sent with the request, in the form of a *.RequestHeaders* object

url_params = None
When a match is made in the route table, this is set to a *dict* containing the matched pattern groups in the url.

get_param(*parameter*, *default=''*, *cast=<type 'str'>*)
Allows you to get a parameter from the request. If the parameter does not exist, or is empty, then a default will be returned. You can also choose to optionally cast the parameter.

If a parameter has multiple values then this will return a list of all those values.

Parameters

- **parameter** – The name of the parameter to get
- **default** – The default to return if the parameter is nonexistent or empty

- **cast** – An optional cast for the parameter.

get_file(*name*)
Along with getting parameters, one may wish to retrieve other data such as files sent.

This provides an interface for getting a file like `FileObject` which can be used like a normal file but also holds some meta data sent with the request. If no file by the given name is found then this will return *None*

id
This is more or less a backwards compatability thing. Provides access to the *id* element of *url_params dict* if it is present otherwise returns *None*

class seshat.headers.**RequestHeaders**(*env=None*)
A basic container for all the headers in an HTTP request. Acts like a dictionary.

referer = None
The referrer address which this request originated from. If no referrer is present this will return *None*

referrer = None
The referrer address which this request originated from. If no referrer is present this will return *None*

user_agent = None
The user agent, unparsed, or the string *Unknown User Agent*

Note: This will probably change to a parsed result class later on.

---

authorization = None
Returns an `Authorization` instance if there is an Authorization header in the request. Otherwise this returns *None*

class seshat.headers.**Authorization**(*auth_type*, ***kwargs*)
Basic little class to help represent an Authorization header. Currently only supports HTTP Basic Auth but support for digest auth is slated for later.

This class is mostly unfinished at this time.

class seshat.headers.**Accept**(*s*)
Basic class which can represent any number of the accept headers which commonly take on the form of: type/subtype; q=int

best(*l*)
Determines which item in the provided list is the best match.

If no match is found then it'll return *None*

Parameters **l** (*list*) – A list of strings which are various accept types.

quality(*item*)
Returns the quality of the given accept item, if it exists in the accept header, otherwise it will return *None*.

2.2.5 session

class seshat.session.**Session**(*request=None*)

load()

save(*response*)

2.2.6 response

class seshat.response.**Response**(*status_code=200*, *headers=None*, *body=None*)

status

class seshat.headers.**ResponseHeaders**(*headers=None*)
Represents the headers which will be sent back to the client with the response. This acts a bit like an *list* of *tuples*.

This class is mostly unfinished as of now.

append(*key*, *val*)

2.2.7 dispatch

Dispatch is the actual WSGI app which is served. This module also contains several configuration properties, along with easy access to the apps route table (RouteTable) though *route_table*.

Note: If you would like to see the logs that seshat produces, using the standard library *logging* module, create a handler for *seshat*

---

seshat.dispatch.**request_obj**
The class which should be used to create a new Request object from. Should inherit from Request

alias of Request

seshat.dispatch.**session_obj**
The class which should be used to instantiate a new session object which will be handed to the controller. Should at least inherit from Session

alias of Session

seshat.dispatch.**dispatch**(*env*, *start_response*)
WSGI dispatcher

This represents the main WSGI app for Seshat. To use with *waitress*, for example:

```
from waitress import serve
serve(dispatch)
```

2.2.8 error_catcher

TODO: doc this

2.2.9 Indices and tables

- *genindex*
- *modindex*
- *search*

S