Seshat Documentation

Release 1.0.0

Joshua P Ashby

Apr 05, 2017

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 BaseController
from seshat.actions import NotFound


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


@route()
class wat(BaseController):
  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".

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!

Doc Contents

`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 BaseController

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

If you see something along the lines of 'Content Generating Request Method' it will usually mean `GET()`, `POST()`, or any other HTTP method verb which might be given to the controller.

class `seshat.controller.`**`BaseController`**(*request*)

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, with the controller:

```
from seshat.controller import BaseController

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

then all GET method requests to this controller will return with the text *<h1>WAT</h1>* however all POST, PUT, DELETE calls will return as a blank page, since those methods are not overridden.

Note: Support for *Not Supported* status codes may be added later, ironically.

`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 generating request method is called and should return either *None* or *`Head`* or *`BaseAction`* object.

If there is a returned value other than None, this will skip calling the content generating request method and simply return directly to dispatch.

A good example of the use for this hook would be for authentication. You could for example, check the id set through the cookie and compare it to a database entry. If the cookie is not currently in use (ie, user not logged in, or similar) then you could do:

```
return Head("401")
```

or perhaps:

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

Return type `Head` or `BaseAction` 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

Routing

Along with needing to have controllers, an app also has to have routes to those controllers. There is a provided auto route function, described below, that will generate the route pattern based off of the file hierarchy of where the controller which is decorated is located at. If you prefer to make your own routes, then you can use the described `RouteContainer` along with the route tables *add_route()* to make your own routes.

seshat.route_containers.**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**()

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

Use like so:

```
from seshat.controller import BaseController
from seshat.route import route

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

will be routed to if the request URL is "/profiles/5" and the resulting id will be stored in *self.request.id*

class `seshat.route_containers.`**`RouteContainer`**(*url*, *controller*)

Provides a base route table entry which can either be used by itself or inherited from to make a custom process for making a route.

Eg of use is the `AutoRouteContainer` which is used in conjunction with the `route()` decocrator to automatically generate a route url.

`controller` **= None**

The controller object, of type `BaseController` which this route represents

Type `BaseController`

`url` **= None**

The actual url pattern for which this route is for. :type: *str*

Response Headers - `Head`

Often times you hopefully won't have to directly make a `Head` object for your response, since the framework and base controller takes care of this for you for you most of the time. However the `Head`` class provides some utilities to make dealing with response headers a little easier.

class `seshat.head.`**`Head`**(*status='200 OK'*, *headers=None*, *errors=None*)

Gives a basic container for the headers within a request.

`status` **= None**

To change the status at anytime, you can simply just assign it a new value.

`errors` **= None**

If an error was encounters then the stack trace will end up here

`reset_headers`()

Allows you to reset the headers

`add_header`(*key*, *value*)

Allows you to add a new header to the list

Eg:

```
add_header("location", "/")
```

will result in the *tuple* `("location", "/")` being added to the list of headers to be returned.

Parameters

- **`key`** – The header name
- **`value`** – The header value

`actions`

Actions allow you to write code that looks like:

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

which I think looks a lot nicer than:

```
class RandomController(BaseController):
  def GET(self):
    self.head.status = "303 SEE OTHER"
    self.head.append("location", "/")
```

This module provides a few common Action classes to use, along with a BaseAction which can be inherited to create your own Actions.

class `seshat.actions.`**`BaseAction`**

Provides a base for creating a new object which represents an HTTP Status code.

All returned data is checked if it is of type *BaseAction* 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.head* to a `Head` 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.`**`Unauthorized`**

Returns a 401 Unauthorized status code back to the client

class `seshat.actions.`**`NotFound`**

Returns a 404 Not Found code and the resulting 404 error controller to be returned to the client.

BaseRequest

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.`**`BaseRequest`**`(`*env*`)`

Represents the request from the server, and contains various information and utilities. Also the place to store the session object.

`cookie_name` **= 'sid'**

The name of the cookie

`url` **= None**
A *urlparse* result of the requests path

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

`remote` **= None**
The clients IP, otherwise *Unknown IP*

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

`referer` **= None**
The referal URL if it exists, otherwise an empty string.

`accepts`(*t*)
Determines if the given mimetype is accepted by the client.

`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*

`build_session`()
Called during the objects instantiation. Override to set the requests *session* property.

`build_cfg`()
Called during the objects instantiation. Override to set the requests *cfg* property.

`log`(*head*)
Called right at the end of the request when the response is being returned to the client. This is useful for logging to a database or log file.

Parameters **`head`** – The reponses `Head` object which was returned to the client.

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 though *route_table*. Documentation on the route table can be found below: `RouteTable`

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 object which should be used to create a new Request item from. Should inherit from BaseRequest

alias of `BaseRequest`

`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)
```

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

`add_route``(`*r_container*`)`

Adds the given route container to the route table.

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

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index