django-htcpcp-tea

Release 0.7.0

Aug 29, 2019

Contents:

Django HTCPCP-TEA is a Django app that implements the TEA extension to HTCPCP as defined in RFC 7168.

CHAPTER 1

Installation

1.1 Getting the package

Note: Regardless of the installation method that you choose, it is recommended that you use a virtual environment to keep your Python environment clean. If you are using Python 3.6 or greater, the tools you will to create virtual environments ship with the interpreter as the venv module. For older versions of Python, or for further customization, see the virtualenv and virtualenvwrapper packages.

The recommended way to install Django HTCPCP-TEA is via pip:

```
$ pip install django-htcpcp-tea
```

Alternatively, you can install Django HTCPCP-TEA from its source:

```
$ git clone git://github.com/blueschu/django-htcpcp-tea.git
$ cd django-htcpcp-tea
$ python3 setup.py install
```

1.2 Installing the Django App

To load Django HTCPCP-TEA into an existing Django project, add `'django_htcpcp_tea'` to the `INSTALLED_APPS` list in your setting file:

```
INSTALLED_APPS = [
    # ...
    'django_htcpcp_tea',
    # ...
]
```

If you intend to use interactive pot sessions, make sure that the Django sessions framework is also installed and properly configured (it's installed by default).

Next, enable the Django HTCPCP-TEA middleware in your settings:

```
MIDDLEWARE = [
    # ...
    'django_htcpcp_tea.middleware.HTCPCPTeaMiddleware',
    # ...
]
```

1.3 Configuring the URL patterns

It is recommend to install the URL configuration for Django HTCPCP-TEA in one of two fashions: the "Standard" Installation, where this app behaves like a standard Django app and is isolated to the URL namespace you include it under, or the "Schema Compliant" Installation, which makes this app work extra hard to follow the HTCPCP URI schema.

1.3.1 Standard Installation

You can include the Django HTCPCP-TEA urls in your URL configuration in the same way as any Django app:

```
from django.urls import include, path

urlspatterns = [
    # ...
    path('htcpcp/', include('django_htcpcp_tea.urls')),
    # ...
]
```

With this installation method, all HTCPCP URIs must be prefixed with the string `/htcpcp`. For example, the request line a of BREW request to Pot #1 would look like

```
BREW /htcpcp/pot-1/ HTTP/1.1
```

1.3.2 Schema Compliant Installation

For an installation that closely follows that HTCPCP standard, append the Django HTCPCP-TEA URLs directly to your URL patterns:

```
from django.urls import include, path
from django_htcpcp_tea import urls as htcpcp_urls

urlspatterns = [
    # ...
]

urlpatterns += htcpcp_urls.urlpatterns
```

Then, add the following options to your project's Django settings file:

```
HTCPCP_OVERRIDE_ROOT_URI = True
```

This will allow Django HTCPCP-TEA to override default URL dispatcher when it receives a request that is unambiguously an HTCPCP request. Making requests to the root url (`/`) or top-level HTCPCP URIs (e.g. `/pot-1/earl-grey/`) will behave as defined in RFCs 2324 and 7168 with no `/htcpcp` prefix required.

1.4 (Optional) Loading the demo data fixture

Django HTCPCP-TEA ships will a few data fixtures to help you explore the app after installation:

- `rfc_2324_additions`: All of the beverage additions types listed as examples in RFC 2324.
- `rfc_7168_additions`: The Sugar addition types added to the HTCPCP standard in RFC 7168.
- `rfc_7168_teas`: The tea types listed as examples in RFC 7168.
- `demo_pots`: A hand-craft selection of coffee- and teapots to demonstrate the HTCPCP protocol (depends on `rfc_2324_additions` and `rfc_7168_teas`).
- `demo_forbidden_combinations`: Common sense rules that forbid combinations of additions contrary to the sensibilities of a consensus of drinkers (depends on `rfc_2324_additions` and `rfc_7168_teas`).

Each of these data fixtures can be loaded using the following manage.py command:

```
$ ./manage.py loaddata FIXTURE
```

CHAPTER 2

Configuration

Django HTCPCP-TEA offers a few mechanisms to customize the behavior of your HTCPCP service.

2.1 Settings

Each of the follow settings can be configured in your project's settings module to control the behavior of Django HTCPCP-TEA.

2.1.1 HTCPCP_ALLOW_DEPRECATED_POST

Default: `True`

Whether to allow clients to use the POST method for brewing requests.

The use of the POST method for HTCPCP requests is officially deprecated in RFC 2324 section 2.1.1. However, the standard also stipulates that, HTCPCP servers must accept POST and BREW requests equivalently, and so setting this configuration to `False` is not recommended.

2.1.2 HTCPCP_CHECK_FORBIDDEN

Default: `True`

Whether to check for forbidden addition combinations as defined in RFC 7168 seciton 2.3.2.

When set to `True`, clients will receive a 403 Forbidden with an explanatory message upon requesting a forbidden addition combination.

2.1.3 HTCPCP_RESPONSE_CONTENT_TYPE

Default: `None`

The MIMI type to use for HTCPCP responses. Specify this setting if you override this app's templates to return content other than HTML, such as JSON or XML data.

If set to a string, the value will be used as the MIME type for the response body.

If left as `None`, the content type provided by Django (usually `text/html`) will be used.

2.1.4 HTCPCP_DISABLE_CSRF

Default: `True`

When set to `True`, this app's views will be exempt from Django's CSRF protection.

In order for HTCPCP requests to pass the CSRF checks, clients would need to set valid Referrer headers and correctly store and transmit CSRF tokens. For most use cases, this is an unnecessary burden, and so these checks are disabled by default.

If you set this configuration to `False`, you will need to update the templates to include CSRF tokens and ensure that your clients know to include the required headers and data in their requests.

2.1.5 HTCPCP_GET_ADDITIONS

Default: `True`

When set to `True`, clients will be able to request beverage additions by listing additions after the `?` in a url.

The URI scheme specified in RFC 2324 section 3 suggests that beverage additions can be provided as list in the query string of the request uri. When this setting is enabled, additions requested in the Accept-Additions header and in the query string are given equal precedence.

Do note that setting this configuration to `True` may enable CSBF (Cross-Site Beverage Forgery) attacks as HTCPCP urls could contain malicious beverage additions that will be requeted with the client knowledge (such as Sea-Salt).

Setting this option to `True` *does not* enable HTCPCP GET requests.

2.1.6 HTCPCP_OVERRIDE_ROOT_URI

Default: `False`

Whether to override the the root URI (`/`) for HTCPCP requests.

This option is included to allow for the creation of highly compliant HTCPCP services. Setting this option to `True` will allow your webapp to serve a proper Alternates header and page when an HTCPCP request is directed to the root URI (see RFC 7168 section 2.1.1). This is equivalent to requesting the `''` URI specified in this app's URL patterns.

Since overriding the root page of a webapp can be a surprising behavior, enabling `HTCPCP_OVERRIDE_ROOR_URI` requires opting-in to stricter HTCPCP validation checks to make sure clients _really_ want a root request to be treated as an HTCPCP request. Namely, the `HTCPCP_STRICT_MINE_TYPES` setting MUST also be enabled in order for this setting to have an effect.

2.1.7 HTCPCP_OVERRIDE_SERVER_NAME

Default: `True`

Whether to override the `Server` header field for HTCPCP requests.

When set to `True`, the Server header will be set to `'HTCPCP-TEA {SERVER_SOFTWARE}'`, where `{SERVER_SOFTWARE}` is the server software string that is added to the environment by a WSGI server, such

as the reference WSGI implementation used by the Django testing server. If the environment providess no `{SERVER_SOFTWARE}`, then the string `Python` will be used.

When set to a callable, the provided callable will be invoked with the request and response objects received by the middleware. The return value will used as the Server header.

Example:

```
def HTCPCP_OVERRIDE_SERVER_NAME(request, response):
    import sys
    from platform import python_implementation

    return 'Teapot {}/{}'.format(
        python_implementation(),
        sys.version.split()[0],
    )
```

When set to a string, the provided string will be formatted with all of the context from `request.META`.

2.1.8 HTCPCP_POT_SESSIONS

Default: `True`

Whether to track user interactions with the server's pots using the Django session framework.

When set to `True`, this app will track when beverage a particular user from each pot to enable statefull interactions with the server. Clients will need to follow a complete HTCPCP request cycle, including a start, stop, and optional 'WHEN' request for each beverage the client requests. Invalid HTCPC request sequences (such as requesting a new beverage in a pot that is already brewing a beverage) will result in errors.

When set to `False`, this app will naively simulate an HTCPCP server without tracking user sessions. Start, stop, and 'WHEN' requests will be accepted even if their ordering is not logical (e.g. saying 'WHEN' before requesting any beverage).

2.1.9 HTCPCP_STRICT_MIME_TYPE

Default: `True`

When set to `True`, HTCPCP requests will be ignored unless they have a content type of either `message/coffeepot` or `message/teapot`.

Set this configuration to `False` if modifying HTTP Content-Type header for HTTP requests is not convenient for your use case.

2.1.10 HTCPCP_STRICT_REQUEST_BODY

Default: `False`

When set to `True`, HTCPCP requests must have a body constistly solely of `start` or `stop`.

By default, this configuration is set to `False` since it is understood that some clients may want to include additional content in the request entity, such as "please" and "thank you".

2.1.11 HTCPCP_USE_SAFE_HEADER_EXT

Default: `True`

Whether to use the extension to the `Safe` header field defined in RFC 2324 section 2.2.1.1.

When set to `True`, the decorators that this app provides for managing the `Safe` header will modify the header's value according to its extended semantics in the HTCPCP standard.

2.2 Templates

The default templates provided by Django HTCPCP-TEA are designed to be minimal so that HTCPCP responses are readable from a terminal window.

All of the templates used by Django HTCPCP-TEA live in the template directory `templates/django_htcpcp_tea`, including the error code templates such as `403.html`. The one exceptions to this is the 404 response code, for which the root 404 template is used to help HTCPCP services "blend in" with the normal functionality of a web app.

2.2.1 base.html

The base template for all HTCPCP templates. Override this template if you would like to incorporate your HTCPCP instances into the natural flow of your web app.

This template must define a single block, `htcpcp_content`, which is where all HTCPCP related content is placed by default.

2.2.2 base_beverage.html

The base template for successful brewing sequences. By default, this template mirrors `base.html`.

2.2.3 brewing.html

The template used when a pot successfully begins brewing a new beverage.

Context variables:

- `pot`: The Pot model that the request was directed to.
- `beverage`: The name of the beverage being brewed.
- `additions`: The additions that were requested for the beverage.

If the request resulted in a new pot of coffee being brew, the following context variable will also be made available in order to comply with RFC 7168 section 2.1.1:

- `alternatives`: The (uri, content-type) pairs for the available alternate beverages.

2.2.4 finished.html

The template used when a pot successfully finishes brewing a beverage.

Context variables:

- `pot`: The Pot model that the request was directed to.

- `beverage`: The name of the beverage being brewed.
- `additions`: The additions that were requested for the beverage.

2.2.5 options.html

The template used to display a list of beverage options when brewing does not begin.

This template will be used when an HTCPCP request is made to the root URI, or when a request is made of a specific pot with the `message/teapot` content type.

Context variables:

- `alternatives`:

2.2.6 pouring.html

The template used when a pot begins to poour milk into a beverage.

Inserting relevant graphics into this templates is *highly* recommended.

Context variables:

- `pot`: The Pot model that the request was directed to.
- `beverage`: The name of the beverage being brewed.
- `additions`: The additions that were requested for the beverage.

2.2.7 base_error.html

The base template for all HTCPCP errors.

This template must define two template blocks: `error_title` and `error_body`.

2.2.8 400.html

The template used for HTCPCP requests with invalid semantics, such as starting a beverage with a `WHEN` request, or attempting to start a new beverage while milk is being poured.

Context variables:

- `error_reason`: An error message explaining why the client's request was not valid.
- `pot`: The Pot model that the request was directed to.
- `beverage`: The name of the beverage being brewed.
- `additions`: The additions that were requested for the beverage.

Note: If the error was due to the client using a `WHEN` request with a `start` body, the `pot`, `beverage`, and `additions` context variables will *not* be available.

2.2.9 403.html

The template used when a forbidden combination of additions is requested.

Context variables:

- `matched_combinations`: The ForbiddenCombination instances that prohibit some part of the requested additions

2.2.10 406.html

The template used when unsupported beverage additions are requested.

Context variables:

- `supported_additions`: The Addition instances that are supported by the pot in question.

2.2.11 418.html

The template used when a client attempts to brew coffee in a teapot.

No context variables are made available.

2.2.12 503.html

The template used when the beverage request cannot be serviced due to the current state of the pot.

Context variables:

- `error_reason`: An error message explaining why the client's request could not be serviced.

If this error occurs due to a new beverage being requested while a pot is busy, the following context variables will also be made available:

- `pot`: The Pot model that the request was directed to.
- `beverage`: The name of the beverage being brewed.
- `additions`: The additions that were requested for the beverage.

2.2.13 includes/additions.html

The template used to render lists of beverage additions.

Context variables:

- `additions`: The Addition instances to be rendered.

2.2.14 includes/alternatives.html

The template used to render lists of beverage alternatives.

Context variables:

- `alternatives`: The (uri, content-type) pairs of available alternate beverages to be rendered.

CHAPTER 3

Examples

This document highlights the basic usage of Django HTCPCP-TEA by interacting with it over a command line interface. It is worth noting that since Django HTCPCP-TEA simulates an HTTP extension, there are many other ways to use it beyond a CLI.

In these examples, I will be using the GNU netcat utility to transmit HTTP requests. Netcat is available for most Unix systems, including MacOS and Linux. Similar utilities are also available for Windows systems.

For the sake of brevity, this document assumes that you have already installed Django HTCPCP-TEA will the "Schema Compliant" URL configuration (see *Schema Compliant Installation*). If you want to follow these examples for with "Standard" installation, add the appropriate prefix to the request line's URI in each HTCPCP request.

Note: These examples interact with a locally run instance of the Django testing server located at `example.localhost:8080`. To duplicate this setup, add `example.localhost` as a loopback address on your system. On Unix systems, this can be accomplished by adding:

```
127.0.0.1 example.localhost
```

to your `/etc/hosts` file. Then, run the following command from your Django project's root:

```
$ ./manage.py runserver example.localhost:8080
```

3.1 Basic HTCPCP

The simplest way to use Django HTCPCP-TEA is with pot sessions disabled. If you have set the `HTCPCP_POT_SESSION` setting to `False`, Django HTCPCP-TEA will not try to keep track of your transaction history with the servers pots, so you do not have to worry about keeping track of a session id.

Let's begin using Django HTCPCP-TEA in this capacity by creating a file named `request.http` with the following contents:

```
BREW / HTTP/1.1
Host: example.localhost
Content-Type: message/coffeepot
Content-Length: 5

start
```

Note: The HTCPCP request listed above uses the `BREW` request method from the HTCPCP standard. If this method is not available to you in your HTTP client, the `POST` method can be used in place with no loss of functionality.

This is a simple HTCPCP request that will prompt the server to return a list of available beverages. To the untrained eye, this file will look like an ordinary HTTP request: it has a well-formed request line, a Host header, and some entity body. A closer look, however, makes it clear that this request makes us of some unconventional HTTP. Most notably, the request method, found at the beginning of the request line, is `BREW` (not a particular common HTTP verb). Moreover, the content type is the HTCPCP specialize `message/coffee` pot, and the content itself is just the word "start". These three elements are key to creating a legal HTCPCP request. Get them wrong, and you will most likely be ignored by your digital barista.

You can send this request to an HTCPCP server by running the following netcat command.

```
$ nc example.localhost 8080 < request.http
```

You should receive a response the resembles the listing below.

```
HTTP/1.1 300 Multiple Choices
Date: Tue, 23 Jul 2019 17:14:51 GMT
Content-Type: text/html; charset=utf-8
Alternates: {"/pot-1/" {type message/coffeepot}},
            --snip--
            {"/pot-3/earl-grey" {type message/teapot}},
            --snip--
Server: HTCPCP-TEA WSGIServer/0.2
X-Frame-Options: SAMEORIGIN
Content-Length: 769

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>django_htcpcp_tea</title>
</head>
<body>
    <h1>Options</h1>
    <ul>
        <li><a href="/pot-1/">/pot-1/</a> (type message/coffeepot)</li>
        --snip--
        <li><a href="/pot-3/earl-grey/">/pot-3/earl-grey/</a> (type message/teapot)</
↪li>
        --snip--
    </ul>
</body>
</html>
```

Once again, this is all pretty standard HTTP. The important bits for our purposes are the `Alternates` header and the response body. You'll note that the `Alternates` header field contains a listing of all of the beverages that are available from each of the pots hosted by the server. A similar, more human-readable listing of the same information is

found in the response's body, which is formatted as HTML by default (see *Templates* for details on how to customize the format of HTCPCP responses).

From this response, we can see that Pot 1 on the server supports brewing coffee on the `/pot-1/` uri, and Pot 3 supports brewing tea on the `/pot-3/earl-grey/` uri. This is all the information we need to start requesting beverages from the HTCPCP server.

To brew your first beverage, change the request uri in `request.http` to `/pot-1/`, while leaving the rest of the content the same:

```
BREW /pot-1/ HTTP/1.1
Host: example.localhost
Content-Type: message/coffeepot
Content-Length: 5

start
```

Send this new request to the server with the same netcat command. You should be greeted with a different output:

```
HTTP/1.1 202 Accepted
Date: Tue, 23 Jul 2019 16:43:17 GMT
Content-Type: text/html; charset=utf-8
Alternates: --snip--
Server: HTCPCP-TEA WSGIServer/0.2
X-Frame-Options: SAMEORIGIN
Content-Length: 878

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>django_htcpcp_tea</title>
</head>
<body>
    <p>Brewing coffee...</p>
        <h2>Alternatives, in case you change your mind...</h2>
        <ul>
            --snip--
        </ul>
</body>
</html>
```

This response indicates that you have successfully asked the server to start brewing a pot of coffee. We still received a list of alternatives beverages despite having requested a cup of coffee due to stipulation in RFC 7168 section 2.1.1, which safeguards against the selection of "inferior caffeinated beverages".

Note: Since pot session are disabled for now, repeating the `BREW` request above will result in precisely the same response. The server will not remember that it is "already brewing a pot of coffee." This functionality will change once the `HTCPCP_POT_SESSION` setting in enabled in Django.

To tell the server to stop brewing your pot of coffee, send the following request by updating `request.http` and running the same netcat command:

```
BREW /pot-1/ HTTP/1.1
Host: example.localhost
Content-Type: message/coffeepot
```

(continues on next page)

(continued from previous page)

```
Content-Length: 4

stop
```

You should receive the following response:

```
HTTP/1.1 201 Created
Date: Tue, 23 Jul 2019 17:32:09 GMT
Content-Type: text/html; charset=utf-8
Server: HTCPCP-TEA WSGIServer/0.2
X-Frame-Options: SAMEORIGIN
Content-Length: 298

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>django_htcpcp_tea</title>
</head>
<body>
    <p>Finished brewing your coffee. Please come and collect your beverage.</p>
    <h2>Additions</h2>
        <p>Your beverage has no additions.</p>
</body>
</html>
```

And voila! Your is coffee is finished and ready for pick-up. You will note, however, that it just black: we did not request any beverage additions yet. Lucky for us, the HTCPCP protocol supports beverage fixations from milk and sugar to spice and booze. This aspect of HTCPCP will be covered in greater detail in Adding Additions to your Requests.

3.2 Bringing HTCPCP to Life

Smarter servers means smarter coffee, right?

To truly reap the benefits of Django HTCPCP-TEA, we'll want to enable session tracking for the coffee pots. This can be accomplished setting `HTCPCP_POT_SESSIONS` to `True` in your Django project settings.

With pot sessions enabled, let's try repeating our brew request from the previous sections. Using netcat, send the following HTCPCP request:

```
BREW /pot-1/ HTTP/1.1
Host: example.localhost
Content-Type: message/coffeepot
Content-Length: 5

start
```

You should receive a response nearly identical to that produced by the sessionless server, with the excpetion of some added `Cookie` headers:

```
HTTP/1.1 202 Accepted
Date: Wed, 31 Jul 2019 15:44:05 GMT
Content-Type: text/html; charset=utf-8
```

(continues on next page)

(continued from previous page)

```
Alternates: --snip--
Server: HTCPCP-TEA WSGIServer/0.2
X-Frame-Options: SAMEORIGIN
Content-Length: 346
Vary: Cookie
Set-Cookie:  sessionid=mx2ijezvoxid0g4sjrwg2e4l7tssjg2e; expires=Wed, 14 Aug 2019␣
↪15:44:05 GMT; HttpOnly; Max-Age=1209600; Path=/; SameSite=Lax

<!DOCTYPE html>
    --snip--
    <p>Brewing coffee...</p>
    --snip--
</html>
```

These new `Cookie` headers denote your Django session ID (precise values will value), which allows the Django session framework to keep track of users between requests.

Let's try repeating the same brew request, but this time add your `sessionid` cookie to the request headers:

```
BREW /pot-1/ HTTP/1.1
Host: example.localhost
Content-Type: message/coffeepot
Content-Length: 5
Cookie: sessionid=YOUR_DJANGO_SESSION_ID

start
```

Resubmitting this request the server will result in a different response:

```
HTTP/1.1 503 Service Unavailable
Date: Wed, 31 Jul 2019 15:52:35 GMT
Content-Type: text/html; charset=utf-8
Server: HTCPCP-TEA WSGIServer/0.2
X-Frame-Options: SAMEORIGIN
Content-Length: 236
Vary: Cookie

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>django_htcpcp_tea</title>
</head>
<body>
    <h1>503 Service Unavailable</h1>
    <p>Pot is busy and cannot start a new beverage.</p>
</body>
</html>
```

The server rejected our request since the pot we specified is currently busy. We can only brew at most one beverage in a given pot at a time.

To finish our beverage, repeat the same stop request as before, but be sure to add the `sessionid` cookie in the request headers:

```
BREW /pot-1/ HTTP/1.1
Host: example.localhost
```

(continues on next page)

(continued from previous page)

```
Content-Type: message/coffeepot
Content-Length: 4
Cookie: sessionid=YOUR_DJANGO_SESSION_ID

stop
```

As before, we receive a simple "beverage finihsed" notice:

```
HTTP/1.1 201 Created
Date: Tue, 23 Jul 2019 17:32:09 GMT
Content-Type: text/html; charset=utf-8
Server: HTCPCP-TEA WSGIServer/0.2
X-Frame-Options: SAMEORIGIN
Content-Length: 298

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>django_htcpcp_tea</title>
</head>
<body>
    <p>Finished brewing your coffee. Please come and collect your beverage.</p>
    <h2>Additions</h2>
        <p>Your beverage has no additions.</p>
</body>
</html>
```

After finishing our beverage, Pot 1 is no longer in use and is free to begin serving other HTCPCP requests. For the sake of example, let's try repeating out "stop" request, even though no beverage is being brewed. You should receive the following error message:

```
HTTP/1.1 400 Bad Request
Date: Wed, 31 Jul 2019 15:59:03 GMT
Content-Type: text/html; charset=utf-8
Server: HTCPCP-TEA WSGIServer/0.2
X-Frame-Options: SAMEORIGIN
Content-Length: 379
Vary: Cookie

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>django_htcpcp_tea</title>
</head>
<body>
    <h1>400 Bad Request</h1>
    <p>The operator of the coffee pot could not understand the request.</p>
    <p> Reason: No beverage is being brewed by this pot, but the request did not␣
↪indicate that a new beverage should be brewed</p>
</body>
</html>
```

Oops. We can't stop a beverage when no beverage is being brewed. That's simple enough to remember.

3.3 Requesting Tea

To be documented.

3.4 Adding Additions to HTCPCP Requests

To be documented.

3.5 Pouring Milk

To be documented.

3.6 Other Errors You'll Find in the Wild

To be documented.

CHAPTER 4

Django HTCPCP-TEA API

Below is an extremely terse summary of the API exposed by Django HTCPCP-TEA.

4.1 Models

class Pot(**args*, ***kwargs*)
A Tea- or Coffee Pot capable of brewing a choice beverage.

name
The name of this pot, e.g. "Joe's Joe Jar" or "Breville (R) BTM800XL"

brew_coffee
Whether this pot can brew coffee.

supported_teas
The types of tea that this pot can brew. May be empty.

supported_additions
The beverage additions that this pot supported. May be empty.

fetch_additions(*addition_names*)
Return the Additions that this pot supports whose names are in the provided sequence.

If this pot does not support an Addition whose name is provided, raise an Addition.DoesNotExist error.

is_teapot
Return True if this pot can serve tea, but cannot serve coffee.

tea_capable
Return True if this pot can serve tea.

class TeaType(**args*, ***kwargs*)
A variety of tea that can be brewed by a pot.

Per the HTCPCP standard, tea may be available as tea bags or tea leaves.

name
The name of this tea type.

slug
The URL slug identifying this tea type.

class Addition(**args*, ***kwargs*)
A beverage addition that may be specified in the Accept-Additions header field of an HTCPCP request.

name
The name of this beverage addition as it would appear in the HTCPCP Accept-Additions header field.

type
The type of this additions

ALCOHOL = 'ACL'

MILK = 'MLK'

OTHER = 'OTR'

SPICE = 'SPC'

SUGAR = 'SUG'

SWEETENER = 'SWT'

SYRUP = 'SYP'

is_milk

class ForbiddenCombination(**args*, ***kwargs*)
A combination of additions that is "contrary to the sensibilities of a consensus of drinkers", either for a specific variety of tea or for all beverages.

tea
The type of tea that this forbidden combination applies to. If null, this forbidden combination applies to all beverages.

additions
The combination of additions that this forbidden combination forbids.

forbids_additions(*requested_additions*)
Return True if the combination of additions that this ForbiddenCombination prohibits is contained in the specified sequence of additions.

4.2 Views

brew_pot(*request*, *pot_designator=None*, *tea_type=None*)

4.3 Decorators

require_htcpcp(*func*)
Decorator to make a view only respond to valid HTCPCP requests.

safe_condition(*token*)
Decorator for adding a `Safe` header to a view's response to indicate client conditions for the safe handling of a device that brews coffee.

The decorator will only have effect if the HTCPCP Safe header extension option is enabled.

See RFC 2324 section 2.2.1.1 for further details.

safe_require_user_awake (*func*)
Decorator for adding an "if-user-awake" safe condition to the Safe header field.

See RFC 2324 section 2.2.1.1 for further details.

4.4 Settings

htcpcp_settings = _HTCPCPTeaSettings('HTCPCP')
Proxy to the standard Django settings that provides defaults for some of this app's settings.

Access to the Django HTCPCP-TEA settings should be done through this instance.

class _HTCPCPTeaSettings (*settings_prefix*)
Proxy to the standard Django settings that provides defaults for some of this app's settings.

ALLOW_DEPRECATED_POST = True

CHECK_FORBIDDEN = True

DISABLE_CSRF = True

GET_ADDITIONS = True

OVERRIDE_ROOT_URI = False

OVERRIDE_SERVER_NAME = True

POT_SESSIONS = True

RESPONSE_CONTENT_TYPE = None

STRICT_MIME_TYPE = True

STRICT_REQUEST_BODY = False

USE_SAFE_HEADER_EXT = True

4.5 Utils

build_alternates (*index_pot=None*)
Generate the Alternates pairs for available beverages, optionally for a specific pot.

find_forbidden_combinations (*requested_additions*, *tea_slug=None*)
Return the list of ForbiddenCombinations that prohibit some part of the requested additions.

render_alternates_header (*alternates_pairs*)
Render (uri, content-type) pairs into an RFC 2295 Alternates header value.

resolve_requested_additions (*request*)
Return the requested additions for the provided request.

Additions may be requested in the `Accept-Additions` header field, or (if the `HTCPCP_GET_ADDITIONS` settings is enabled) in the query string of a uri.

Note that the returned additions are not guaranteed to be valid additions that are supported by any pot.

CHAPTER 5

Changes

For complete record of changes, see the commit log of the public git repository.

5.1 v0.7.0

Released 2019-08-29

- Add initial example usage documentation
- Add setting to override MIME type for HTCPCP responses
- Add PyPI package version badge to README
- Add Python 3.8 dev build to Travis-CI config
- Change PyPI development status classifier to Beta
- Fix server name being overridden for non-HTCPCP requests
- Remove explicit module names from API documentation
- Remove supported Python and Django version badges from README

5.2 v0.6.0

Released 2019-07-14

- Add units tests for `middleware` module
- Add units test for `admin` module
- Add check to strictly enforce HTCPCP MIME types
- Add setting to override the rot url view for HTCPCP requests
- Add documentation for overriding HTCPCP templates

- Add changelog to Sphinx documentation
- Add API summary to Sphinx documentation
- Add support for the extension to the `Safe` header field from RFC 2324
- Amend minor typo in `utils` module docstrings
- Expand unit tests for `views` module
- Move `require_htcpcp` decorator to the `decorators` module
- Update the package installation instructions

5.3 v0.5.1

Released 2019-07-08

- Fix attribute error when running tests with Django 2.0 / Python 3.4

5.4 v0.5.0

Released 2019-07-08

- Add unit tests for the `views` module
- Add formal support for Python 3.7
- Expand `utils` unit tests
- Fix missing `Alternates` header due to generator exhaustion
- Fix `Server` header override when the WSGI implementation does not populate the `SERVER_SOFTWARE` variable
- Fix detection of supported teas in the request URI
- Refactor handling of `Alternates` header generation
- Refactor the pots data fixture to include a pot that supports a proper subset of available teas

5.5 v0.4.0

Released 2019-07-05

- Add setting to override the `Server` header for HTCPCP responses
- Add support for user-defined forbidden combinations of additions
- Add additional unit tests for the `utils` and `models` modules
- Add data fixture for demo forbidden combinations of additions
- Update README description
- Update package metadata
- Update reStructuredText formatting in the configuration docs
- Optimized model listings on the admin site

5.6 v0.3.1

Released 2019-06-25

- Remove Sphinx build directory from package data

5.7 v0.3.0

Released 2019-06-24

- Add Sphinx documentation for installation and configuration
- Add unit tests for the models module
- Update links in REAME.rst
- Fix typo in Travis-CI build matrix
- Fix error in `utils` module unit tests

5.8 v0.2.2

Released 2019-06-24

- Fix syntax error in Python 3.4
- Fix dependency errors for Travis-CI build jobs (123e022)

5.9 v0.2.1

Released 2019-06-24

- Add Travis-CI and Coveralls reporting to README

5.10 v0.2.0

Released 2019-06-24

- Add informative content to README
- Add data fixture for RFC 2324 additions
- Add data fixture for RFC 7168 additions
- Add data fixture for RFC 7168 teas
- Add data fixture for demo pots
- Add default `coverage` configuration
- Add Travis-CI integration
- Add script to run Django tests
- Add `tests` package

- Add unit tests for `utils` module
- Fix filter override in `admin.PotsServingMixin`
- Fix duplicate tea types being recorded in admin counts
- Refactor template hierarchy
- Improve context visibility in templates
- Refactor logic for determining a pots addition and milk support

5.11 v0.1.2

Released 2019-06-23

- Re-release patch version due to packaging mishap

5.12 v0.1.1

Released 2019-06-23

- Add data files to package manifest

5.13 v0.1.0

Released 2019-06-21

- Add licence
- Add app class
- Add `Pot` model
- Add `TeaType` model
- Add `Addition` model
- Add initial admin site
- Add `settings` module
- Add initial url config
- Add initial HTCPCP middleware
- Add `require_htcpcp` decorator
- Add HTCPCP view
- Add initial templates
- Add `utils` module
- Add setup script

Python Module Index

d

Index

Symbols

A

B

C

D

F

G

H

I

M

N

O

P

R

S

T

U