Usernado

Release 0.3

Reganto

Aug 19, 2022

DOCUMENTATION:

Usernado is a Tornado extension to make life easier!

CHAPTER

ONE

QUICK LINKS

- Current version: 0.3 (download from PyPI, Change Log)
- Source Github
- Found a bug? don't hesitate to report it here

CHAPTER

TWO

HELLO, WORLD

Here is a simple "Hello, world" example for Usernado:

```
from usernado.helpers import api_route
from usernado import APIHandler
from tornado import web, ioloop


@api_route("/hello", name="hello")
class Hello(APIHandler):
    def get(self):
        self.response({"message": "Hello, world"})

def make_app():
    return web.Application(api_route.urls, autoreload=True)


def main():
    app = make_app()
    app.listen(8000)
    ioloop.IOLoop.current().start()


if __name__ == "__main__":
    main()
```

Then you can test it via Curl or other HTTP Clients.

```
$ curl localhost:8000/hello
```

2.1 Intro

2.1.1 Why Usernado

- Why not?
- I like
- For every project I wrote with Tornado, I had to repeat the code for some parts. For example, to authenticate users, I had to write repeated code every time. So I decided to write an extension to solve my need.

2.1.2 Features

- REST support
- Websocket easier than ever
- ORM agnostic authentication
- Humanize datetime in templates
- Better exception printer thanks to tornado-debugger

2.2 Getting started

2.2.1 Installation

You can install **Usernado** with `pip` or poetry.

With pip

```
(.venv) $ pip install usernado
```

With poetry

```
(.venv) $ poetry add usernado
```

From Github

```
(.venv) $ pip install git+https://github.com/reganto/usernado
```

2.3 Web

If you want to add support for other ORM:

1. Create a class which implement *`IAuth`* interface with name convention like `_OrmNameAuth`.
2. Override *`register`* and *`login`* methods
3. Add model check logic in *`WebHandler`*'s *`register`* and *`login`* methods.

2.3.1 Authentication interface

class usernado.web.**IAuth**

Every ORM specific authentication class MUST implement this Interface and override `register` and `login` mehtods.

abstract static login(*request: HTTPRequest*, *model: Union[peewee.Model, sqlalchemy.orm.declarative_base]*, *username: str*, *password: str*) → bool

Abstract login method.

Parameters

- **request** (*tornado.httpclient.HTTPRequest*) – Incoming HTTP request.
- **model** (*Union[peewee.Model, sqlalchemy.orm.declarative_base]*) – ORM model.
- **username** (*str*) – Username.
- **password** (*str*) – Password.

Returns
True if user registration done successfully otherwise False.

Return type
bool

abstract static register(*request: HTTPRequest*, *model: Union[peewee.Model, sqlalchemy.orm.declarative_base]*, *username: str*, *password: str*) → bool

Abstract register method.

Parameters

- **request** (*tornado.httpclient.HTTPRequest*) – Incoming HTTP request.
- **model** (*Union[peewee.Model, sqlalchemy.orm.declarative_base]*) – ORM model.
- **username** (*str*) – Username.
- **password** (*str*) – Password.

Returns
True if user registration done successfully otherwise False.

Return type
bool

2.3.2 WebHandler

class usernado.web.**WebHandler**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Every HTTP request handler MUST inherit from `WebHandler`.

property authenticate: bool

Check if current user is authenticated?

Return type
bool

get_current_user() → Optional[bytes]

To implement user authentication we need to override this method.

for more information, take a look at Tornado documentation.

Returns
A secure cookie.

Return type
Optional[bytes]

get_escaped_argument(*name: str*, *default: Optional[str] = None*, *strip: bool = True*) → str

Returns the xhtml escaped value of the argument with the given name.

Parameters

- **name** (*str*) – Name of the desired argument.
- **default** (*Optional[str], optional*) – Default value for non existing argument, defaults to None.
- **strip** (*bool, optional*) – Strip argument value, defaults to True.

Returns
Escaped argument.

Return type
str

login(*model: Union[peewee.Model, sqlalchemy.orm.declarative_base]*, *username: str*, *password: str*) → bool

Signin user with provided username and password.

Parameters

- **model** (*Union[peewee.Model, sqlalchemy.orm.declarative_base]*) – ORM model.
- **username** (*str*) – Username.
- **password** (*str*) – Password.

Raises
UnsupportedUserModelError – Raised when auth operation for `model` was not provided.

Returns
True if user login done successfully otherwise False.

Return type
bool

logout() → None

Logout user.

redirect_to_route(*name: str*, **args: Any*) → None

Redirect to particular route.

Parameters
name (*str*) – Named route

register(*model: Union[peewee.Model, sqlalchemy.orm.declarative_base]*, *username: str*, *password: str*) → bool

Signup user with provided username and password.

Parameters

- **model** (*Union[peewee.Model, sqlalchemy.orm.declarative_base]*) – ORM model.
- **username** (*str*) – Username.
- **password** (*str*) – Password.

Raises
UnsupportedUserModelError – Raised when auth operation for `model` was not provided.

Returns
True if user registration done successfully otherwise False.

Return type
bool

2.3.3 Exceptions

exception usernado.web.**UserDoesNotExistError**

exception usernado.web.**UserAlreadyExistError**

exception usernado.web.**PermissionDeniedError**

exception usernado.web.**UnsupportedUserModelError**

2.4 API

2.4.1 APIHandler

class usernado.api.**APIHandler**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Every API handler MUST inherit from `APIHandler`.

Actually `APIHandler` is a *WebHandler* with extra two methods. To use API functionalities you can decorate `APIHandler` inherited classes with *api_route decorator*.

get_json_argument(*name: str*, *default: Optional[str] = None*) → str

Get json argument from incoming request.

Parameters

- **name** (*str*) – Name of the argument.
- **default** (*str, optional*) – Default value for argument if not presented, defaults to None

Raises
DataMalformedOrNotProvidedError –

Returns
Particular JSON argument that comes with current request.

Return type
str

get_json_arguments() → Dict[Any, Any]

Get all json arguments from incoming request.

Raises
DataMalformedOrNotProvidedError –

Returns
All JSON argument that comes with current request

Return type
Dict[Any, Any]

response(*message: Optional[Dict[str, Union[str, bytes]]] = None*, *headers: Optional[Dict[str, str]] = None*, *status_code: int = 200*) → None

Send JSON response to the client.

Parameters

- **message** (*_Message*) – Response body.

- **headers** (*Optional[Dict[str, str]], optional*) – Response headers, defaults to None
- **status_code** (*int, optional*) – Response status code, defaults to 200

2.4.2 Exceptions

exception usernado.api.**DataMalformedOrNotProvidedError**

2.5 Websocket

2.5.1 WebSocketHandler

class usernado.websocket.**WebSocketHandler**(*application: Application*, *request: HTTPServerRequest*, ***kwargs: Any*)

Every websocket handler MUST inherit from `WebSocketHandler`.

broadcast(*participants: Set[WebSocketHandler]*, *message: Union[bytes, str, Dict[str, Any]]*, *binary: bool = False*) → None

Broadcast a message to all participants.

Parameters

- **participants** (*Set[usernado.WebSocketHandler]*) – Participants to send message.
- **message** (*Union[bytes, str, Dict[str, Any]]*) – Message to send.
- **binary** (*bool, optional*) – Type of message, defaults to False.

send(*message: Union[bytes, str, Dict[str, Any]]*, *binary: bool = False*) → None

Send a message to the particular participant.

Parameters

- **message** (*Union[bytes, str, Dict[str, Any]]*) – Message to send.
- **binary** (*bool, optional*) – Type of the message, defaults to False.

2.6 Helpers

2.6.1 Pluralize UIModule

class usernado.helpers.**Pluralize**(*handler: RequestHandler*)

Pluralize a string based on a value.

You Must set `Pluralize` as a valid UIModule In `ui_modules` setting like so

```
ui_modules=dict('pluralize': Pluralize)
```

and then use this uimoduele in templates like so

```
{% module pluralize(post, post_counts) %}
```

2.6.2 humanize decorator

@usernado.helpers.**humanize**(*func: Callable[[...], Any]*) → Callable[[...], Any]

Humanize datetime in templates.

To use `humanize` you have to create a DateTimeField in your model then create a method in your model decorated with `humanize` like so

```
@humanize
def diff_for_humans(self):
    return self.created_at
```

then use `humanize` in your templates like so:

```
{{ obj.diff_for_humans() }}
```

2.6.3 api_route decorator

@usernado.helpers.**api_route**(*url: str*, *name: Optional[str] = None*) → Callable[[...], Any]

Usernado API router class.

You can decorate *APIHandler* inherited classes with `api_route` decorator.

See also:

For further information take a look at examples

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FOUR

CONTACT ME

tell.reganto[at]gmail.com

PYTHON MODULE INDEX

u

INDEX