django-decadence Documentation

Release 0.2

Maciej Janiszewski

Jul 30, 2018

Guides:

Decadence is a library which enables creating dynamic, constantly updating webpages with Django. It consists of three elements:

- Serializable models - so you have a consistent way of displaying and accessing data
- Rendering API - exposes Django's templating engine so you can use it from JS in your frontend
- Updates API - allows fields to be updated in real time over WebSocket protocol

You don't need to use all of these elements if you don't want to.

The main motivation behind this library is that you don't have to switch over to a fancy JS framework nor duplicate rendering logic in both JS and Django templates for content loaded over AJAX/WebSocket protocols.

CHAPTER 1

Getting Started

To use Decadence in your project, first install it using pip

```
pip install -e git://github.com/ksiazkowicz/django-decadence.git#egg=django-decadence
```

Then, add Decadence to `INSTALLED_APPS` in `settings.py`

```
INSTALLED_APPS = {
    ...
    "django_decadence",
    ...
}
```

If you want to use Updates API, you also need to update your `routing.py` file

```
from django_decadence.consumers import UpdateConsumer

channel_routing = [
    ...
    route_class(UpdateConsumer, path=r"^/updates$"),
    ...
]
```

For Rendering API, add following lines to your url_patterns:

```
url(r'^decadence/', include('django_decadence.urls')),
```

To be continued

CHAPTER 2

Using Updates API

Updates API provides automatic content updates, for example, when post title is changed, this change will be broadcast to all users that are currently browsing the page.

2.1 General idea

Updates API is exposed over WebSocket protocol at `ws://localhost/updates`. New values are broadcast to Groups named in this fashion: `[namespace]-[object id]-[path]`:

- namespace - for example, a name of model we're updating,
- object id - pk of this object,
- path - unique identifier that describes a specific field, for example title

For 23rd user's like status , we'd use something like: `post-1-like.23`.

To be a part of such Group, client needs to send a subscription request.

```
{
    "subscribe": true,
    "group": "post-1-like.23",
}
```

Where `subscribe` field can be either true or false (in case client wants to unsubscribe specific group).

Whenever field was updated with new data, server will send a message that looks like this:

```
{
    "type": "update_value",
    "path": "post-1-like.23",
    "value": "1"
}
```

- `type` - type of change Server asks client to apply. Available types are:

  - update_value - value (ex. innerHTML of an Element containing this field) should be updated to value
  - toggle_class - class (ex. hidden of an Element corresponding to this field) defined in class field should be either added or removed to an Element
  - update_attribute - attribute (ex. data-liked of an Element corresponding to this field) named attribute_name should be changed to value
- path - group name (field name)
- value - new value

Optional parameters:

- class - name of class to toggle (only in toggle_class)
- attribute_name - name of attribute, which value will be changed

2.2 Usage (Django)

2.2.1 Template

There is also a template tag available at update_tags called updatable.

Usage:

```
{% updatable post "content" %}
```

This tag will include <span> element with all required data attributes to enable automatic updates using UpdateListener.

You can also add safe=True as argument if you want string to be marked as safe.

2.2.2 Model

General idea is, that if you already use serialization features from Decadence in your model, you will usually send out a few “update_value” requests during save(). To avoid reimplementing this from scratch for each model and simplify integration, push_update method is provided.

To avoid updating certain fields (upload date for blog post?), you can add this field in model:

```
updates_excluded = []
```

Fields from this list will never get checked for changes and won’t trigger any errors.

By default, each change will result in update_value request being broadcast. In case you want to override this behaviour, you can define a list of options with which update() method will be called for specific field. A common case could be, for example, updating an URL to image if it changed. You can either define it as a list, or a method which returns it.

```
# as method, in case you need to override default value or something, in this case␣
↪field is called "is_hidden"
def updates_is_hidden(self):
    return [{
        "type_name": "toggle_class",    # use toggle_class instead of
        "field": "main-div",            # in case you want to override path
```

(continues on next page)

(continued from previous page)

```
        "classname": "hidden",
        "value": not self.is_hidden,    # override value, optional
    }, ]

# as list, in this case field is called "image"
updates_image = [{
    "type_name": "update_attribute",
    "attribute_name": "src",
}, ]
```

One final step is overriding `save()` method, capturing data before `save()` and calling `push_update()` with captured data as an argument.

```
def save(self, *args, **kwargs):
    original_data = None
    if self.pk:
        old = ExampleModel.objects.get(pk=self.pk)
        original_data = old.serialize()
    super(ExampleModel, self).save(*args, **kwargs)
    self.push_update(original_data)
```

2.2.3 Low-level

To send out updates, you need to either override `save()` method on a model or use signals.

Example:

```
from django_decadence import update

...
update(type_name="update_value", path="post-1-like.23", value="1");
...
```

2.3 Usage (TypeScript)

To simplify the process of subscribing to specific fields, there is a client for Updates API available globally under window.UpdateListener.

`UpdateListener` automatically captures all elements in `document` that have `data-update-group` attribute containing a valid Group name. For dynamically created Elements, you need to call UpdateListener again: `window["UpdateListener"].init(element)`. Decadence does this automatically.

CHAPTER 3

Consumers

class django_decadence.consumers.**UpdateConsumer**(*message*, ***kwargs*)

Update Consumer

Handles logic for adding user sessions to specific Groups. Handles subscribe/unsubscribe requests.

connect(*message*, ***kwargs*)

Called when a WebSocket connection is opened.

disconnect(*message*, ***kwargs*)

Called when a WebSocket connection is closed.

receive(*content*, ***kwargs*)

Called with decoded JSON content.

CHAPTER 4

Updater

`django_decadence.helpers.`**`check_template_path`**(*path*)

Checks whether the path is valid. Valid path is a path that: - is not outside DECADENCE_DIR (default: templates/includes/decadence) - …

`django_decadence.helpers.`**`update`**(*type_name='update_value'*, *path=''*, *value=''*, *classname=''*, *attribute_name=''*)

Pushes out content updates to user through our update channel.

Parameters

- **`type_name`** – type of content update:
  - **toggle_class - adds/remove given class from element, if value is** true, it's added
  - update_attribute - replaces html attribute value with given value
  - update_value - updates content of html element with given value
- **`path`** – name of update group
- **`value`** – new value
- **`classname`** – name of class to be added/removed to element (optional)
- **`attribute_name`** – name of attribute which value will be replaced (optional)

CHAPTER 5

Models

class django_decadence.models.**DecadenceModel**(**args*, ***kwargs*)

Implements a generic model that supports Decadence-specific features like serialization.

push_update(*original_data={}*)

Compares changes between old serialization data and new, then pushes out updates through Updates API.

serialize(*user=None*, *fields=None*)

Attempts to generate a JSON serializable dictionary based on current model

updates_excluded = []

list of fields excluded from updates

class django_decadence.models.**SerializableQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

QuerySet extended with Decadence serialization method

CHAPTER 6

Views

class django_decadence.views.**DecadenceListView**(***kwargs*)

Custom ListView that adds "serialized" to context with Decadence-serialized queryset

class django_decadence.views.**DecadenceTableView**(***kwargs*)

Implements a Decadence-serialized paginated table view.

django_decadence.views.**generate_html**(*request*)

Generates HTML code from request. You should send a JSON through post this way: csrfmiddlewaretoken=token&data={"template": "includes/decadence/cancer.html", . . . } Data should have a template + context.

CHAPTER 7

Templatetags

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index