

---

# **Bootstrap-Flask Documentation**

*Release 1.0.4*

**Grey Li**

**Aug 29, 2020**



---

# Contents

---

<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Basic Usage . . . . .	3
1.2	Use Macros . . . . .	5
1.3	Migrate from Flask-Bootstrap . . . . .	13
1.4	Advanced Usage . . . . .	15
1.5	Run the demo application . . . . .	17
<b>2</b>	<b>API Reference</b>	<b>19</b>
2.1	API Reference . . . . .	19
<b>3</b>	<b>Changelog</b>	<b>21</b>
3.1	Changelog . . . . .	21
<b>4</b>	<b>Development</b>	<b>25</b>
<b>5</b>	<b>Authors</b>	<b>27</b>
<b>6</b>	<b>License</b>	<b>29</b>
	<b>Python Module Index</b>	<b>31</b>
	<b>Index</b>	<b>33</b>



Bootstrap 4 helper for Flask/Jinja2.



## 1.1 Basic Usage

### 1.1.1 Installation

```
$ pip install bootstrap-flask
```

This project can't work with Flask-Bootstrap at the same time. If you have already installed Flask-Bootstrap in the same Python environment, you have to uninstall it and then reinstall this project:

```
$ pip uninstall flask-bootstrap bootstrap-flask
$ pip install bootstrap-flask
```

---

**Tip:** See *Migrate from Flask-Bootstrap* to learn how to migrate from Flask-Bootstrap.

---

### 1.1.2 Initialization

```
from flask_bootstrap import Bootstrap
from flask import Flask

app = Flask(__name__)

bootstrap = Bootstrap(app)
```

### 1.1.3 Resources helpers

Bootstrap-Flask provides two helper functions to load Bootstrap resources in the template: `bootstrap.load_css()` and `bootstrap.load_js()`.

Call it in your base template, for example:

```
<head>
...
{{ bootstrap.load_css() }}
</head>
<body>
...
{{ bootstrap.load_js() }}
</body>
```

You can pass `version` to pin the Bootstrap 4 version you want to use. It defaults to load files from CDN. Set `BOOTSTRAP_SERVE_LOCAL` to `True` to use built-in local files. However, these methods are optional, you can also write `<href></href>` and `<script></script>` tags to include Bootstrap resources (from your static folder or CDN) manually by yourself.

### 1.1.4 Starter template

For reasons of flexibility, Bootstrap-Flask doesn't include built-in base templates (this may change in the future). For now, you have to create a base template yourself. Be sure to use an HTML5 doctype and include a viewport meta tag for proper responsive behaviors. Here's an example base template:

```
<!doctype html>
<html lang="en">
  <head>
    {% block head %}
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
    ↳fit=no">

    {% block styles %}
    <!-- Bootstrap CSS -->
    {{ bootstrap.load_css() }}
    {% endblock %}

    <title>Your page title</title>
    {% endblock %}
  </head>
  <body>
    <!-- Your page content -->
    {% block content %}{% endblock %}

    {% block scripts %}
    <!-- Optional JavaScript -->
    {{ bootstrap.load_js() }}
    {% endblock %}
  </body>
</html>
```

Use this in your templates folder (suggested names are `base.html` or `layout.html` etc.), and inherit it in child templates. See [Template Inheritance](#) for more details on inheritance.



## 1.1.5 Macros

Macro	Templates Path	Description
<code>render_field()</code>	<code>bootstrap/form.html</code>	Render a WTForms form field
<code>render_form()</code>	<code>bootstrap/form.html</code>	Render a WTForms form
<code>render_pager()</code>	<code>bootstrap/pagination.html</code>	Render a basic Flask-SQLAlchemy pagination
<code>render_pagination()</code>	<code>bootstrap/pagination.html</code>	Render a standard Flask-SQLAlchemy pagination
<code>render_nav_item()</code>	<code>bootstrap/nav.html</code>	Render a navigation item
<code>render_breadcrumb_item()</code>	<code>bootstrap/nav.html</code>	Render a breadcrumb item
<code>render_static()</code>	<code>bootstrap/utils.html</code>	Render a resource reference code (i.e. <code>&lt;link&gt;</code> , <code>&lt;script&gt;</code> )
<code>render_messages()</code>	<code>bootstrap/utils.html</code>	Render flashed messages send by <code>flash()</code> function
<code>render_table()</code>	<code>bootstrap/table.html</code>	Render a table with given data

How to use these macros? It's quite simple, just import them from the corresponding path and call them like any other macro:

```
{% from 'bootstrap/form.html' import render_form %}
{{ render_form(form) }}
```

Go to the [Use Macros](#) page to see the detailed usage for these macros.

## 1.1.6 Configurations

Configuration Variable	Default Value	Description
<code>BOOTSTRAP_SERVE_LOCAL</code>	<code>False</code>	If set to <code>True</code> , local resources will be used for <code>load_*</code> methods.
<code>BOOTSTRAP_BTN_STYLE</code>	<code>'secondary'</code>	Default form button style, will change to <code>primary</code> in next major release
<code>BOOTSTRAP_BTN_SIZE</code>	<code>'md'</code>	Default form button size

**Tip:** See [Form Button Customization](#) to learn how to customize form buttons.

## 1.2 Use Macros

These macros will help you to generate Bootstrap-markup codes quickly and easily.

### 1.2.1 `render_nav_item()`

Render a Bootstrap nav item.

## Example

```
{% from 'bootstrap/nav.html' import render_nav_item %}

<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="navbar-nav mr-auto">
    {{ render_nav_item('index', 'Home') }}
    {{ render_nav_item('explore', 'Explore') }}
    {{ render_nav_item('about', 'About') }}
  </div>
</nav>
```

## API

**render\_nav\_item**(*endpoint*, *text*, *badge=""*, *use\_li=False*, *\*\*kwargs*)

Render a Bootstrap nav item.

### Parameters

- **endpoint** – The endpoint used to generate URL.
- **text** – The text that will displayed on the item.
- **badge** – Badge text.
- **use\_li** – Default to generate `<a></a>`, if set to `True`, it will generate `<li><a></a></li>`.
- **kwargs** – Additional keyword arguments pass to `url_for()`.

## 1.2.2 render\_breadcrumb\_item()

Render a Bootstrap breadcrumb item.

## Example

```
{% from 'bootstrap/nav.html' import render_breadcrumb_item %}

<nav aria-label="breadcrumb">
  <ol class="breadcrumb">
    {{ render_breadcrumb_item('home', 'Home') }}
    {{ render_breadcrumb_item('users', 'Users') }}
    {{ render_breadcrumb_item('posts', 'Posts') }}
    {{ render_breadcrumb_item('comments', 'Comments') }}
  </ol>
</nav>
```

## API

**render\_breadcrumb\_item**(*endpoint*, *text*, *\*\*kwargs*)

Render a Bootstrap breadcrumb item.

### Parameters

- **endpoint** – The endpoint used to generate URL.

- **text** – The text that will displayed on the item.
- **kwargs** – Additional keyword arguments pass to `url_for()`.

### 1.2.3 `render_field()`

Render a form field create by Flask-WTF/WTFForms.

#### Example

```
{% from 'bootstrap/form.html' import render_field %}

<form method="post">
  {{ form.csrf_token() }}
  {{ render_field(form.username) }}
  {{ render_field(form.password) }}
  {{ render_field(form.submit) }}
</form>
```

#### API

**`render_field`**(*field*, *form\_type*="basic", *horizontal\_columns*=('lg', 2, 10), *button\_style*="", *button\_size*="", *button\_map*={})

Render a single form field.

##### Parameters

- **field** – The form field (attribute) to render.
- **form\_type** – One of basic, inline or horizontal. See the Bootstrap docs for details on different form layouts.
- **horizontal\_columns** – When using the horizontal layout, layout forms like this. Must be a 3-tuple of (column-type, left-column-size, right-column-size).
- **button\_style** – Accept Bootstrap button style name (i.e. primary, secondary, outline-success, etc.), default to secondary (e.g. btn-secondary). This will overwrite config `BOOTSTRAP_BTN_STYLE`.
- **button\_size** – Accept Bootstrap button size name: sm, md, lg, block, default to md. This will overwrite config `BOOTSTRAP_BTN_SIZE`.
- **button\_map** – A dictionary, mapping button field name to Bootstrap button style names. For example, `{'submit': 'success'}`. This will overwrite `button_style` and `BOOTSTRAP_BTN_STYLE`.

---

**Tip:** See `button_customization` to learn how to customize form buttons.

---

### 1.2.4 `render_form()`

Render a form object create by Flask-WTF/WTFForms.

## Example

```
{% from 'bootstrap/form.html' import render_form %}

{{ render_form(form) }}
```

## API

**render\_form** (*form*, *action=""*, *method="post"*, *extra\_classes=None*, *role="form"*, *form\_type="basic"*, *horizontal\_columns=('lg', 2, 10)*, *enctype=None*, *button\_style=""*, *button\_size=""*, *button\_map={}*, *id=""*, *novalidate=False*, *render\_kw={}*)

Outputs Bootstrap-markup for a complete Flask-WTF form.

### Parameters

- **form** – The form to output.
- **action** – The URL to receive form data.
- **method** – `<form>` method attribute.
- **extra\_classes** – The classes to add to the `<form>`.
- **role** – `<form>` role attribute.
- **form\_type** – One of `basic`, `inline` or `horizontal`. See the Bootstrap docs for details on different form layouts.
- **horizontal\_columns** – When using the horizontal layout, layout forms like this. Must be a 3-tuple of (`column-type`, `left-column-size`, `right-column-size`).
- **enctype** – `<form>` enctype attribute. If `None`, will automatically be set to `multipart/form-data` if a `FileField` or `MultipleFileField` is present in the form.
- **button\_style** – Accept Bootstrap button style name (i.e. `primary`, `secondary`, `outline-success`, etc.), default to `secondary` (e.g. `btn-secondary`). This will overwrite config `BOOTSTRAP_BTN_STYLE`.
- **button\_size** – Accept Bootstrap button size name: `sm`, `md`, `lg`, `block`, default to `md`. This will overwrite config `BOOTSTRAP_BTN_SIZE`.
- **button\_map** – A dictionary, mapping button field name to Bootstrap button style names. For example, `{'submit': 'success'}`. This will overwrite `button_style` and `BOOTSTRAP_BTN_STYLE`.
- **id** – The `<form>` id attribute.
- **novalidate** – Flag that decide whether add `novalidate` class in `<form>`.
- **render\_kw** – A dictionary, specifying custom attributes for the `<form>` tag.

**form\_errors** (*form*, *hiddeens=True*)

Renders paragraphs containing form error messages. This is usually only used to output hidden field form errors, as others are attached to the form fields.

### Parameters

- **form** – Form whose errors should be rendered.
- **hiddeens** – If `True`, render errors of hidden fields as well. If `'only'`, render *only* these.

---

**Tip:** See *Form Button Customization* to learn how to customize form buttons.

---

## 1.2.5 render\_form\_row()

Render a row of a grid form.

### Example

```
{% from 'bootstrap/form.html' import render_form_row %}

<form method="post">
  {{ form.csrf_token() }}
  {{ render_form_row([form.username, form.password]) }}
  {{ render_form_row([form.remember]) }}
  {{ render_form_row([form.submit]) }}
  {# Custom col which should use class col-md-2, and the others the defaults: #}
  {{ render_form_row([form.title, form.first_name, form.surname], col_map={'title':
→'col-md-2'}) }}
  {# Custom col which should use class col-md-2 and modified col class for the_
→default of the other fields: #}
  {{ render_form_row([form.title, form.first_name, form.surname], col_class_default=
→'col-md-5', col_map={'title': 'col-md-2'}) }}
</form>
```

### API

**render\_form\_row**(*fields*, *row\_class*='form-row', *col\_class\_default*='col', *col\_map*={}, *button\_style*="", *button\_size*="", *button\_map*={})

Render a bootstrap row with the given fields.

#### Parameters

- **fields** – An iterable of fields to render in a row.
- **row\_class** – Class to apply to the div intended to represent the row, like `form-row` or `row`
- **col\_class\_default** – The default class to apply to the div that represents a column if nothing more specific is said for the div column of the rendered field.
- **col\_map** – A dictionary, mapping `field.name` to a class definition that should be applied to the div column that contains the field. For example: `col_map={'username': 'col-md-2'}`
- **button\_style** – Accept Bootstrap button style name (i.e. `primary`, `secondary`, `outline-success`, etc.), default to `secondary` (e.g. `btn-secondary`). This will overwrite config `BOOTSTRAP_BTN_STYLE`.
- **button\_size** – Accept Bootstrap button size name: `sm`, `md`, `lg`, `block`, default to `md`. This will overwrite config `BOOTSTRAP_BTN_SIZE`.
- **button\_map** – A dictionary, mapping button field name to Bootstrap button style names. For example, `{'submit': 'success'}`. This will overwrite `button_style` and `BOOTSTRAP_BTN_STYLE`.

**Tip:** See *Form Button Customization* to learn how to customize form buttons.

---

## 1.2.6 render\_pager()

Render a pagination object create by Flask-SQLAlchemy.

### Example

```
{% from 'bootstrap/pagination.html' import render_pager %}

{{ render_pager(pagination) }}
```

### API

**render\_pager** (*pagination*, *fragment=""*, *prev*=(`<span aria-hidden="true">&larr;</span> Previous`)\safe, *next*=(`<span aria-hidden="true">&rarr;</span>`)\safe, *align=""*, *\*\*kwargs*)

Renders a simple pager for query pagination.

#### Parameters

- **pagination** – *Pagination* instance.
- **fragment** – Add url fragment into link, such as #comment.
- **prev** – Symbol/text to use for the “previous page” button.
- **next** – Symbol/text to use for the “next page” button.
- **align** – Can be ‘left’, ‘center’ or ‘right’, default to ‘left’.
- **kwargs** – Additional arguments passed to `url_for`.

## 1.2.7 render\_pagination()

Render a pagination object create by Flask-SQLAlchemy.

### Example

```
{% from 'bootstrap/pagination.html' import render_pagination %}

{{ render_pagination(pagination) }}
```

### API

**render\_pagination** (*pagination*, *endpoint=None*, *prev*='«', *next*='»', *ellipses*='...', *size=None*, *args={}*, *fragment=""*, *align=""*, *\*\*kwargs*)

Render a standard pagination for query pagination.

#### Parameters

- **pagination** – Pagination instance.
- **endpoint** – Which endpoint to call when a page number is clicked. `url_for()` will be called with the given endpoint and a single parameter, `page`. If `None`, uses the requests current endpoint.
- **prev** – Symbol/text to use for the “previous page” button. If `None`, the button will be hidden.
- **next** – Symbol/text to use for the “next page” button. If `None`, the button will be hidden.
- **ellipses** – Symbol/text to use to indicate that pages have been skipped. If `None`, no indicator will be printed.
- **size** – Can be ‘sm’ or ‘lg’ for smaller/larger pagination.
- **args** – Additional arguments passed to `url_for()`. If `endpoint` is `None`, uses `args` and `view_args`
- **fragment** – Add url fragment into link, such as `#comment`.
- **align** – The align of the pagination. Can be ‘left’, ‘center’ or ‘right’, default to ‘left’.
- **kwargs** – Extra attributes for the `<ul>`-element.

### 1.2.8 `render_static()`

Render a resource reference code (i.e. `<link>`, `<script>`).

#### Example

```
{% from 'bootstrap/utils.html' import render_static %}
{{ render_static('css', 'style.css') }}
```

#### API

**`render_static`** (*type*, *filename\_or\_url*, *local=True*)

Render a resource reference code (i.e. `<link>`, `<script>`).

##### Parameters

- **type** – Resources type, one of `css`, `js`, `icon`.
- **filename\_or\_url** – The name of the file, or the full url when `local` set to `False`.
- **local** – Load local resources or from the passed URL.

### 1.2.9 `render_messages()`

Render flashed messages send by `flask.flash()`.

## Example

Flash the message in your view function with `flash(message, category)`:

```
from flask import flash

@app.route('/test')
def test():
    flash('a info message', 'info')
    flash('a danger message', 'danger')
    return your_template
```

Render the messages in your base template (normally below the navbar):

```
{% from 'bootstrap/utils.html' import render_messages %}

<nav>...</nav>
{{ render_messages() }}
<main>...</main>
```

## API

**render\_messages** (*messages=None, container=False, transform={...}, default\_category='primary', dismissible=False, dismiss\_animate=False*)

Render Bootstrap alerts for flash messages send by `flask.flash()`.

### Parameters

- **messages** – The messages to show. If not given, default to get from `flask.get_flashed_messages(with_categories=True)`.
- **container** – If true, will output a complete `<div class="container">` element, otherwise just the messages each wrapped in a `<div>`.
- **transform** – A dictionary of mappings for categories. Will be looked up case-insensitively. Default maps all Python loglevel names to Bootstrap CSS classes.
- **default\_category** – If a category does not has a mapping in `transform`, it is passed through unchanged. `default_category` will be used when `category` is empty.
- **dismissible** – If true, will output a button to close an alert. For fully functioning dismissible alerts, you must use the alerts JavaScript plugin.
- **dismiss\_animate** – If true, will enable dismiss animate when click the dismiss button.

When you call `flash('message', 'category')`, there are 8 category options available, mapping to Bootstrap 4's alerts type:

primary, secondary, success, danger, warning, info, light, dark.

If you want to use HTML in your message body, just warpper your message string with `flask.Markup` to tell Jinja it's safe:

```
from flask import flash, Markup

@app.route('/test')
def test():
    flash(Markup('a info message with a link: <a href="/">Click me!</a>'), 'info')
    return your_template
```



### 1.2.10 render\_table()

Render a Bootstrap table.

#### Example

```
{% from 'bootstrap/table.html' import render_table %}

{{ render_table(data) }}
```

#### API

**render\_table**(*data*, *titles=None*, *primary\_key='id'*, *primary\_key\_title='#'*, *caption=None*, *table\_classes=None*, *header\_classes=None*, *responsive=False*, *responsive\_class='table-responsive'*)

Render a Bootstrap table.

##### Parameters

- **data** – An iterable of data objects to render. Can be dicts or class objects.
- **titles** – An iterable of tuples of the format (prop, label) e.g [ ('id', '#') ], if not provided, will automatically detect on provided data, currently only support SQLAlchemy object.
- **primary\_key** – Primary key identifier for a single row, default to `id`.
- **primary\_key\_title** – Primary key title for a single row, default to `#`.
- **caption** – A caption to attach to the table.
- **table\_classes** – A string of classes to apply to the table (e.g 'table-small table-dark').
- **header\_classes** – A string of classes to apply to the table header (e.g 'thead-dark').
- **responsive** – Whether to enable/disable table responsiveness.
- **responsive\_class** – The responsive class to apply to the table. Default is 'table-responsive'.

## 1.3 Migrate from Flask-Bootstrap

If you come from Flask-Bootstrap, looking for an alternative that supports Bootstrap 4, well, then you are in the right place.

Bootstrap-Flask was originated as a fork of Flask-Bootstrap, but some APIs were changed, deleted and improved, some bugs were fixed, and on top of all that, some new macros were added. This tutorial will go through all the steps to migrate from Flask-Bootstrap.

### 1.3.1 Uninstall and Install

Flask-Bootstrap and Bootstrap-Flask can't live together, so you have to uninstall Flask-Bootstrap first and then install Bootstrap-Flask:

```
$ pip uninstall flask-bootstrap
$ pip install bootstrap-flask
```

if you accidentally installed both of them, you will need to uninstall them both first:

```
$ pip uninstall flask-bootstrap bootstrap-flask
$ pip install bootstrap-flask
```

If you want to use both Flask-Bootstrap and Bootstrap-Flask for different projects, you can use virtual environment.

### 1.3.2 Initialize the Extension

The initialization of this extension is the same as with Flask-Bootstrap. The package's name is still `flask_bootstrap`, in order to follow the rule of Flask extension development and easy the pain of migration.

```
from flask_bootstrap import Bootstrap
from flask import Flask

app = Flask(__name__)

bootstrap = Bootstrap(app)
```

### 1.3.3 Create Base Template

In Flask-Bootstrap, there is a built-in base template called `bootstrap/base.html`. This extension does not provide one. You have to create it by yourself; an example starter is given here:

```
<!doctype html>
<html lang="en">
  <head>
    {% block head %}
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
    ↳fit=no">

    {% block styles %}
      <!-- Bootstrap CSS -->
      {{ bootstrap.load_css() }}
    {% endblock %}

    <title>Your page title</title>
    {% endblock %}
  </head>
  <body>
    <!-- Your page content -->
    {% block content %}{% endblock %}

    {% block scripts %}
      <!-- Optional JavaScript -->
      {{ bootstrap.load_js() }}
    {% endblock %}
  </body>
</html>
```

Just create a file called `base.html` inside your `templates` folder, copy the contents above into it. There are two resource helper methods used in the example template above (i.e. `bootstrap.load_css()` and `bootstrap.load_js()`). They will generate `<href></href>` and `<script></script>` codes to include Bootstrap's CSS and JavaScript files. These default to load the resources from CDN (provided by jsDelivr). If you set the configuration variable `BOOTSTRAP_SERVE_LOCAL` to `True` the local resources inside the package folder will be used instead.

It's optional to use these resources methods, you can write the codes by yourself to load Bootstrap resources in your application's static folder, or from a different CDN provider that you want to use.

### 1.3.4 Change Template and Macro Name

The template `bootstrap/wtf.html` changed to `bootstrap/form.html`, some macro's name was changed too:

For example, you will need to change the import statement:

```
{% from 'bootstrap/wtf.html' import quick_form, form_field %}
```

to:

```
{% from 'bootstrap/form.html' import render_form, render_field %}
```

The macros below were removed (or not supported yet):

- `ie8()`
- `icon()`
- `form_button()`
- `analytics()`
- `uanalytics()`

There are also some new macros were introduced, check them out at [Macros](#) section.

## 1.4 Advanced Usage

### 1.4.1 Form Button Customization

#### Button Style

When you use form related macros, you have a couple ways to style buttons. Before we start to dive into the solutions, let's review some Bootstrap basics: In Bootstrap 4, you have 9 normal button style and 8 outline button style, so you have 17 button style classes below:

- `btn-primary`
- `btn-secondary`
- `btn-success`
- `btn-danger`
- `btn-warning`
- `btn-info`
- `btn-light`

- btn-dark
- btn-link
- btn-outline-primary
- btn-outline-secondary
- btn-outline-success
- btn-outline-danger
- btn-outline-warning
- btn-outline-info
- btn-outline-light
- btn-outline-dark

Remove the `btn-` prefix, you will get what we (actually, I) called “Bootstrap button style name”:

- primary
- secondary
- success
- danger
- warning
- info
- light
- dark
- link
- outline-primary
- outline-secondary
- outline-success
- outline-danger
- outline-warning
- outline-info
- outline-light
- outline-dark

You will use these names in Bootstrap-Flask. First, you configuration variables `BOOTSTRAP_BTN_STYLE` to set a global form button style:

```
from flask import Flask
from flask_bootstrap import Bootstrap

app = Flask(__name__)
bootstrap = Bootstrap(app)

app.config['BOOTSTRAP_BTN_STYLE'] = 'primary' # default to 'secondary'
```

Or you can use `button_style` parameter when using `render_form`, `render_field` and `render_form_row`, this parameter will overwrite `BOOTSTRAP_BTN_STYLE`:

```
{% from 'bootstrap/form.html' import render_form %}
{{ render_form(form, button_style='success') }}
```

Smirily, you can use this way to control the button size. In Bootstrap 4, buttons can have 4 sizes:

- btn-sm
- btn-md (the default size)
- btn-lg
- btn-block

So, the size names used in Bootstrap-Flask will be:

- sm
- md (the default size)
- lg
- block

Now you can use a configuration variable called `BOOTSTRAP_BTN_STYLE` to set global form button size:

```
from flask import Flask
from flask_bootstrap import Bootstrap

app = Flask(__name__)
bootstrap = Bootstrap(app)

app.config['BOOTSTRAP_BTN_SIZE'] = 'sm' # default to 'md'
```

there also a parameter called `button_size` in form related macros (it will overwrite `BOOTSTRAP_BTN_SIZE`):

```
{% from 'bootstrap/form.html' import render_form %}
{{ render_form(form, button_size='lg') }}
```

if you need a **block level small** button (btn btn-sm btn-block), you can just do something hacky like this:

```
app.config['BOOTSTRAP_BTN_SIZE'] = 'sm btn-block'
```

What if I have three buttons in one form, and I want they have different styles and sizes? The answer is `button_map` parameter in form related macros. `button_map` is a dictionary that mapping button field name to Bootstrap button style names. For example, `{'submit': 'success'}`. Here is a more complicate example:

```
{% from 'bootstrap/form.html' import render_form %}
{{ render_form(form, {'submit': 'success', 'cancel': 'secondary', 'delete': 'danger'}
→) }}
```

It will overwrite `button_style` and `BOOTSTRAP_BTN_STYLE`.

## 1.5 Run the demo application

Type these commands in the terminal:

```
$ git clone https://github.com/greyli/bootstrap-flask.git
$ cd bootstrap-flask/examples
$ pip install -r requirements.txt
$ flask run
```

Now go to <http://localhost:5000>.

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

### 2.1 API Reference

**class** flask\_bootstrap.**Bootstrap** (*app=None*)

**init\_app** (*app*)

**static load\_css** (*version='4.3.1'*)

Load Bootstrap's css resources with given version.

New in version 0.1.0.

**Parameters** **version** – The version of Bootstrap.

**static load\_js** (*version='4.3.1', jquery\_version='3.4.1', popper\_version='1.14.0',  
with\_jquery=True, with\_popper=True*)

Load Bootstrap and related library's js resources with given version.

New in version 0.1.0.

**Parameters**

- **version** – The version of Bootstrap.
- **jquery\_version** – The version of jQuery.
- **popper\_version** – The version of Popper.js.
- **with\_jquery** – Include jQuery or not.
- **with\_popper** – Include Popper.js or not.





### 3.1 Changelog

#### 3.1.1 1.4

Release data: 6/15

- Add `render_table` macro to render a Bootstrap table (#71)

#### 3.1.2 1.3.2

Release data: 2020/5/30

- Support display error message for `RadioField` and `BooleanField`, display description for `RadioField`.

#### 3.1.3 1.3.1

Release data: 2020/4/29

- Fix add `field.render_kw.class` to form label class attribute.
- Fix append extra space in class attribute when no `field.render_kw.class` presents (#63).

#### 3.1.4 1.3.0

Release date: 2020/4/23

- Fix `enctype` attribute setting for `WTForms MultipleFileField` (**Flask-Bootstrap #198**<https://github.com/mbr/flask-bootstrap/issues/198><'\_\_').
- Fix `WTForms` field class append bug when using `render_kw={'class': 'my-class'}` (#53).

- Fix WTForms field description not showing for BooleanField (**‘Flask-Bootstrap #197<<https://github.com/mbr/flask-bootstrap/issues/197>>‘**\_\_).
- Add configuration variable `BOOTSTRAP_BTN_STYLE` (default to `primary`) and `BOOTSTRAP_BTN_SIZE` (default to `md`) to set default form button style and size globally.
- Add parameter `button_style` and `button_map` for `render_form` and `render_field` to set button style and size.

### 3.1.5 1.2.0

Release date: 2019/12/5

- Add macro `render_messages` for rendering flashed messages.
- Fix rendering bug for WTForms `FormField` (#34).

### 3.1.6 1.1.0

Release date: 2019/9/9

- Update Bootstrap version to 4.3.1

### 3.1.7 1.0.10

Release date: 2019/3/7

- Added macro `render_form_row` for rendering a row of a bootstrap grid form.

### 3.1.8 1.0.9

Release date: 2018/11/14

- Fix missing error message when form type was horizontal.
- Fix missing input label for `RadioField`.
- Fix `RadioField` grid when form type was horizontal.

### 3.1.9 1.0.8

Release date: 2018/9/6

- Correct macro name used in `templates/bootstrap/form.html`: `form_field`→`render_field`.

### 3.1.10 1.0.7

Release date: 2018/8/30

- Built-in resources loading not based on `‘FLASK_ENV‘`.

### 3.1.11 1.0.6

Release date: 2018/8/7

- Fix unmatched built-in jQuery filename. (#8)

### 3.1.12 1.0.5

Release date: 2018/8/7

- Fix KeyError Exception if ENV isn't defined. (#7)

### 3.1.13 1.0.4

Release date: 2018/7/24

- Add missing `<script>` tag in resources URL. (#3)

### 3.1.14 1.0.3

Release date: 2018/7/22

- Built-in resources will be used when `FLASK_ENV` set to `development`.
- Change CDN provider to jsDelivr.

### 3.1.15 1.0.2

Release date: 2018/7/21

- Include `popper.js` before `bootstrap.js` in `bootstrap.load_js()`. (#2)

### 3.1.16 1.0.1

Release date: 2018/7/1

- Fix local resources path error
- Add basic unit tests

### 3.1.17 1.0

Release date: 2018/6/11

Initialize release.



## CHAPTER 4

---

### Development

---

We welcome all kinds of contributions. You can run test like this:

```
$ python setup.py test
```



## CHAPTER 5

---

Authors

---

Maintainer: [Grey Li](#)

See also the list of [contributors](#) who participated in this project.





## CHAPTER 6

---

### License

---

This project is licensed under the MIT License (see the `LICENSE` file for details).

Some macros were part of [Flask-Bootstrap](#) and were modified under the terms of its BSD License.



**f**

`flask_bootstrap`, 19



## B

Bootstrap (*class in flask\_bootstrap*), 19

## F

flask\_bootstrap (*module*), 19

form\_errors() (*built-in function*), 8

## I

init\_app() (*flask\_bootstrap.Bootstrap method*), 19

## L

load\_css() (*flask\_bootstrap.Bootstrap static method*),  
19

load\_js() (*flask\_bootstrap.Bootstrap static method*),  
19

## R

render\_breadcrumb\_item() (*built-in function*), 6

render\_field() (*built-in function*), 7

render\_form() (*built-in function*), 8

render\_form\_row() (*built-in function*), 9

render\_messages() (*built-in function*), 12

render\_nav\_item() (*built-in function*), 6

render\_pager() (*built-in function*), 10

render\_pagination() (*built-in function*), 10

render\_static() (*built-in function*), 11

render\_table() (*built-in function*), 13