Django File Picker Documentation

Release 0.5

Caktus Consulting Group LLC

Oct 31, 2017

Contents

django-file-picker is a pluggable Django application used for uploading, browsing, and inserting various forms of media into HTML form fields.

Using jQuery Tools, django-file-picker integrates seamlessly into pre-existing pages by installing an overlay that lists file details and, when applicable, image thumbnails. New files can also be uploaded from within the overlay (via AJAX Upload).

django-file-picker provides a few optional extensions to help get started, including `file_picker.uploads`, an app with pre-built Image and File models, and `file_picker.wymeditor`, an app that integrates with WYMeditor, a web-based WYSIWYM (What You See Is What You Mean) XHTML editor. These extensions are provided for convenience and can easily be replaced by custom modules.

For complete documentation checkout, http://django-file-picker.readthedocs.org

CHAPTER 1

Dependencies

Required

- Python 2.7, 3.4, 3.5 and 3.6
- Django 1.8 to 1.11 (inclusive)
- sorl-thumbnail==12.4a1
- jQuery 1.4.x
- jQuery Tools 1.2.x
- AJAX Upload (included)

Optional

- django.contrib.staticfiles
- WYMeditor 0.5

  If you are using `django.contrib.staticfiles`, then add `file_picker` to your INSTALLED_APPS to include the related css/js.

  Otherwise make sure to include the contents of the static folder in your project's media folder.

CHAPTER 2

Basic Installation

1. Add `file_picker` to INSTALLED_APPS in settings.py:

   ```
   INSTALLED_APPS = (
       'file_picker',
       'file_picker.uploads', # file and image Django app
       'file_picker.wymeditor', # optional WYMeditor plugin
   )
   ```

   `file_picker.uploads` will automatically create two pickers named 'images' and 'files'.

2. Add the `file_picker` URLs to urls.py, e.g.:

   ```
   from django.conf.urls import include, url
   import file_picker

   urlpatterns = [
       # ...
       url(r'^file-picker/', include(file_picker.site.urls)),
       # ...
   ]
   ```

Development sponsored by Caktus Consulting Group, LLC..

CHAPTER 3

Contents

Basic Setup

1. Use or create a model for storing images and/or files. For simplicity here we will use the models in `file_picker.uploads`: Image and File.
2. Use or create another model to contain the text field(s) to be inserted by the picker. Here we will use the Post model from `sample_project.article`. It has two text fields, Body and Teaser.
3. To use the pickers on both the teaser and body fields use a *formfield_override* to override the widget with the `file_picker.widgets.SimpleFilePickerWidget`:

```
import file_picker
from django.contrib import admin
from sample_project.article import models as article_models

class PostAdmin(admin.ModelAdmin):
    formfield_overrides = {
        models.TextField: {
            'widget': file_picker.widgets.SimpleFilePickerWidget(pickers={
                'image': "images", # a picker named "images" from file_picker.
↪uploads
                'file': "files", # a picker named "files" from file_picker.uploads
            }),
        },
    }

    class Media:
        js = ("http://cdn.jquerytools.org/1.2.5/full/jquery.tools.min.js",)

admin.site.register(article_models.Post, PostAdmin)
```

Simple File Picker Widget

class `file_picker.widgets.`**`SimpleFilePickerWidget`**

To use the simple file picker widget, override the desired form field's widget. It takes a dictionary with expected keys *"image"* and/or *"file"* which define which link to use, i.e. "Add Image" and/or "Add File".

The Uploads App

The uploads app is designed to make it easy to get File Picker up and running without having to add models or register them with `file_picker`. The uploads app includes two simple pickers which can be attached to your own project's text fields. For installation instructions, check out *Basic Setup*

FilePicker

class `file_picker.uploads.file_pickers.`**`FilePicker`**

This is a subclass of `file_picker.FilePickerBase` which is connected to the *File* model and can be found in the Uploads admin section.

ImagePicker

class `file_picker.uploads.file_pickers.`**`ImagePicker`**

This is a subclass of `file_picker.ImagePickerBase` which is connected to the *Image* model and can be found in the Uploads admin section.

Simple File Picker Widget

These pickers can be used like any other. Below is an example of how to add them on a single text field:

```
body = forms.TextField(
            widget=file_picker.widgets.SimpleFilePickerWidget(pickers={
                'file': "files",
                'image': "images",
            }))
```

The *"file"* and *"image"* keywords add classes to the inputs, so that the links for the overlay can be added. They can also be added to all fields in a form by using the *formfield_overrides* as in:ref:*setup*.

The WYMeditor App

The WYMeditor app is included to make it easy to integrate a File Picker with a popular WYSIWYG interface. WYMeditor is a Javascript based editor. Its documentation can be found here. This application offers an extra form widget for applying WYMeditor to a text field with buttons for files and images if desired.

WYMeditorWidget

class `file_picker.wymeditor.widgets.`**`WYMeditorWidget`**

To use the WYMeditorWidget, override the desired form field's widget. It takes in a dictionary with expected keys *"image"* and/or *"file"* which define which button is used to call the overlay, either an image or a paper clip icon respectively.

Example TextField Override

An example using a *formfield_override* in an admin class using WYMeditor and a File Picker for each *TextField* in the form:

```
class PostAdmin(admin.ModelAdmin):
        formfield_overrides = {
            models.TextField: {
                'widget': file_picker.wymeditor.widgets.WYMeditorWidget(
                    pickers={
                        'image': "images",
                        'file': "files",
                    }
                ),
            },
        }

        class Media:
            js = ("http://cdn.jquerytools.org/1.2.5/full/jquery.tools.min.js",)
```

Creating Custom Pickers

File Picker offers many ways to create custom pickers and assign them to your own models.

The FilePickerBase Class

The base file picker class has a mixture of class based views and helper functions for building the colorbox on the page. File pickers should be included in the *file_pickers.py* file in the root directory of any app so that the auto-discovery process can find it.

Attributes

Each picker can take a set of attributes for easy customization.:

```
from myapp.models import CustomModel
from myapp.forms import CustomForm
import file_pickers

class CustomPicker(file_picker.FilePickerBase):
    form = CustomForm
    page_size = 4
    link_headers = ['Insert File',]
    columns = ['name', 'body', 'description',]
    extra_headers = ['Name', 'Body', 'Description',]
    ordering = '-date'

file_picker.site.register(CustomModel, CustomPicker, name='custom')
```

None of these attributes are required and they all have sane defaults.

- *form*- If not set, a *ModelForm* is created using the model defined in the register function. This is used to build the form on the Upload tab.
- *link_headers*- Defines the headers for the first set of columns which are used to insert content into the textbox or WYSIWYG widget of your choice. By default, it converts _ to ' ' and capitalizes the first letter of the field's name.
- *columns*- Defines the fields you want to be included on the listing page and their ordering.
- *extra_headers*- This list is used to display the headers for the columns and needs to be in the same order as *columns*.
- *ordering*- Defines the order of items on the listing page. In this example, the code would run `query_set.order_by('-date')`. If no ordering is provided, we'll order by `-pk`.

Methods

The three main methods consist of *append*, *list*, and *upload_file*. *List* and *upload_file* take a request object and act as views, while *append* takes a model instance and builds the JSON output for list. Other methods are available but typically do not need to be modified.

append(obj)

This method takes *obj* which is a model instance and returns a dictionary to be used by *list*. This dictionary is of the form:

```
{
    'name': 'Name for the object.',
    'url': 'The url to the file.',
    'extra': {
        'column_name_1': 'value',
        'column_name_2': 'value',
    },
    'insert': [
        'text or html to insert if first link is clicked',
        'text or html to insert if second link is clicked',
    ],
    'link_content': [
        'string to show on first insert link',
        'string to show on second insert link',
    ],
}
```

The *link_content* list, *insert* list, and *extra* dictionary must all be the same length, and must match the length of the *link_headers* of your custom FilePicker.

list(request)

This takes a *request* object and returns:

```
{
    'page': 1-based integer representing current page number,
    'pages': List of pages,
    'search': The current search term,
    'result': List returned from *append*,
    'has_next': Boolean telling paginator whether to show the next button,
    'has_previous': Boolean telling paginator whether to show the previous button.,
    'link_headers': List of link headers defined by the Picker attribute (or␣
↪generated if not defined),
    'extra_headers': List of the extra_headers specified by the Picker attribute,
    'columns': List of column names specified by the Picker attribute,
}
```

upload_file(request)

This takes a *request* object and builds the upload file form, which is used to upload files in two steps: first the file, and then the other form parameters.

If called without POST data it returns a JSON dictionary with the key `form` containing the HTML representation of the form.

If called with a file and then with the POST data, it performs a two step process. If the form validates successfully on the second step it returns the result of *append* for the object which was just created.

Screenshots

Motivation

The deep concern while building a file picker has been flexibility. Too many projects focus on wrapping everything together so that they can make deep connections.

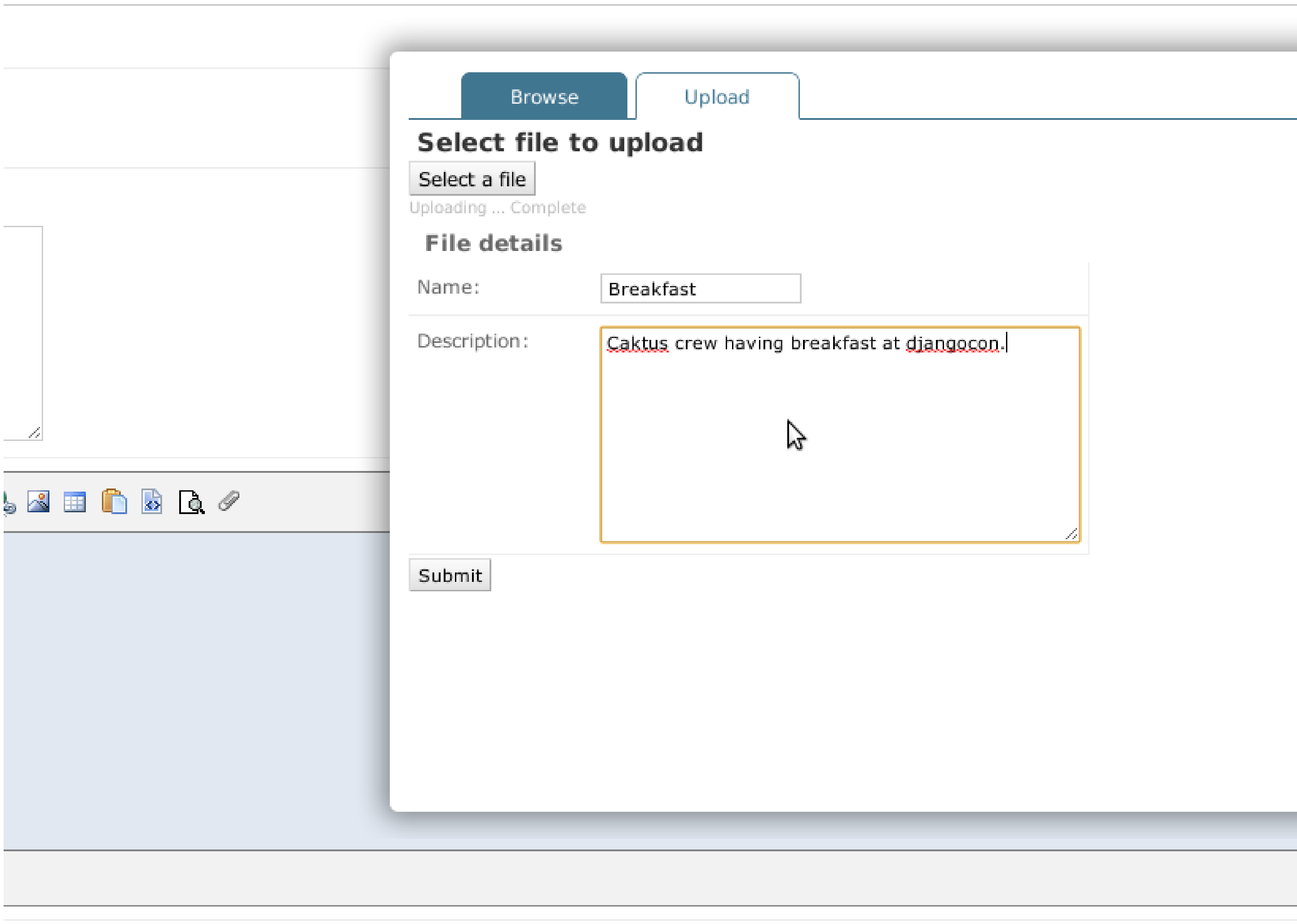
Our main goal has been to build an application that is easy to plug into a wide variety of applications and models.

Running the Sample Project

django-file-picker includes a sample project to use as an example. You can run the sample project like so:

```
~$ mkvirtualenv filepicker-sample
(filepicker-sample)~$ pip install "django>=1.8,<2.0"
(filepicker-sample)~$ git clone git://github.com/caktus/django-file-picker.git
(filepicker-sample)~$ cd django-file-picker/
(filepicker-sample)~/django-file-picker$ python setup.py develop
(filepicker-sample)~/django-file-picker$ cd sample_project/
(filepicker-sample)~/django-file-picker/sample_project$ ./manage.py migrate
(filepicker-sample)~/django-file-picker/sample_project$ ./manage.py createsuperuser
(filepicker-sample)~/django-file-picker/sample_project$ ./manage.py runserver
```

Then go to the admin, log in, and add an Article Post. There will be 2 links to 'Insert File' and 'Insert Image' which will pop up the File Picker dialog.



Select file to upload

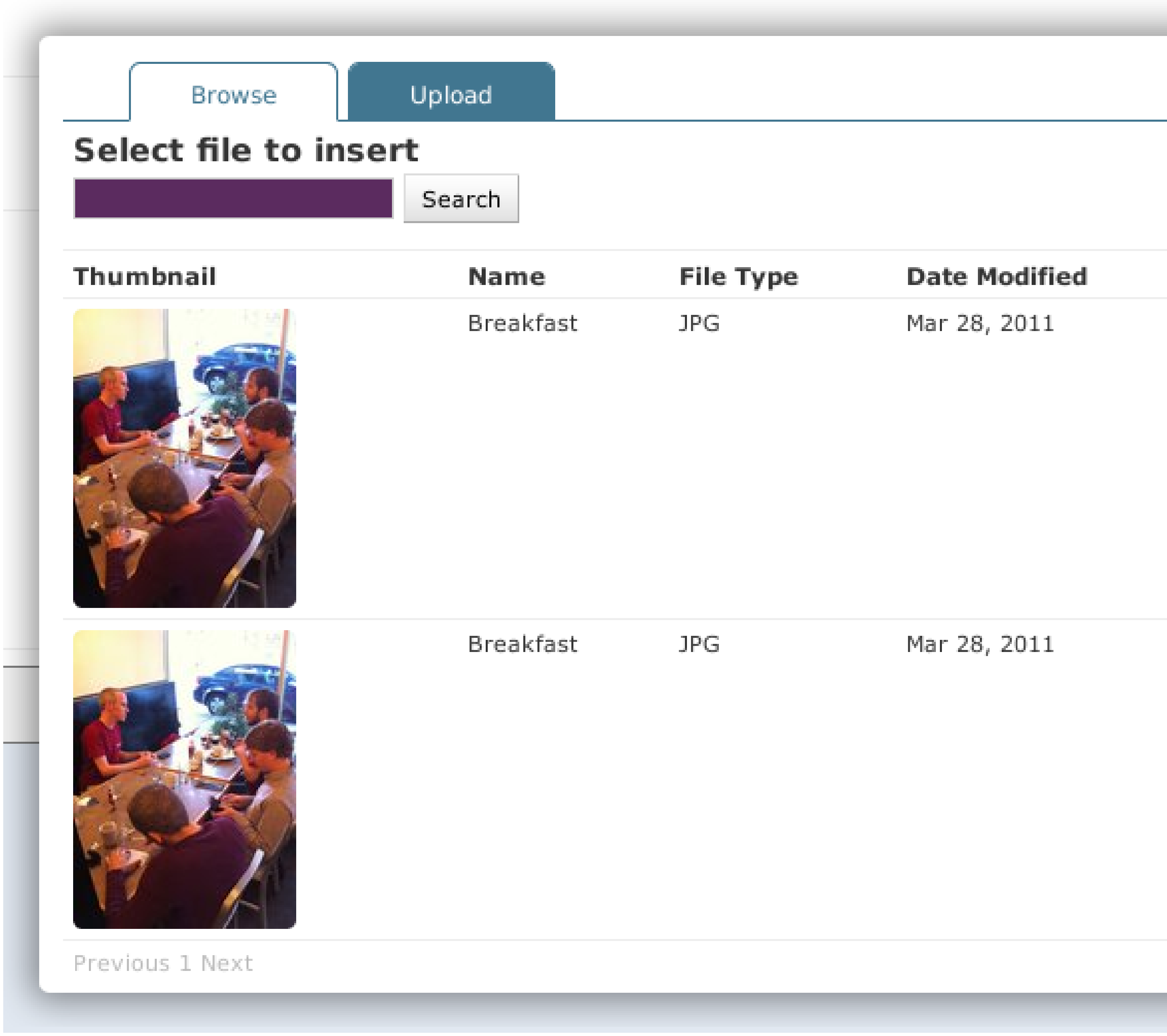
Uploading ... Complete

File details

Name: Breakfast

Description: Caktus crew having breakfast at djangocon.

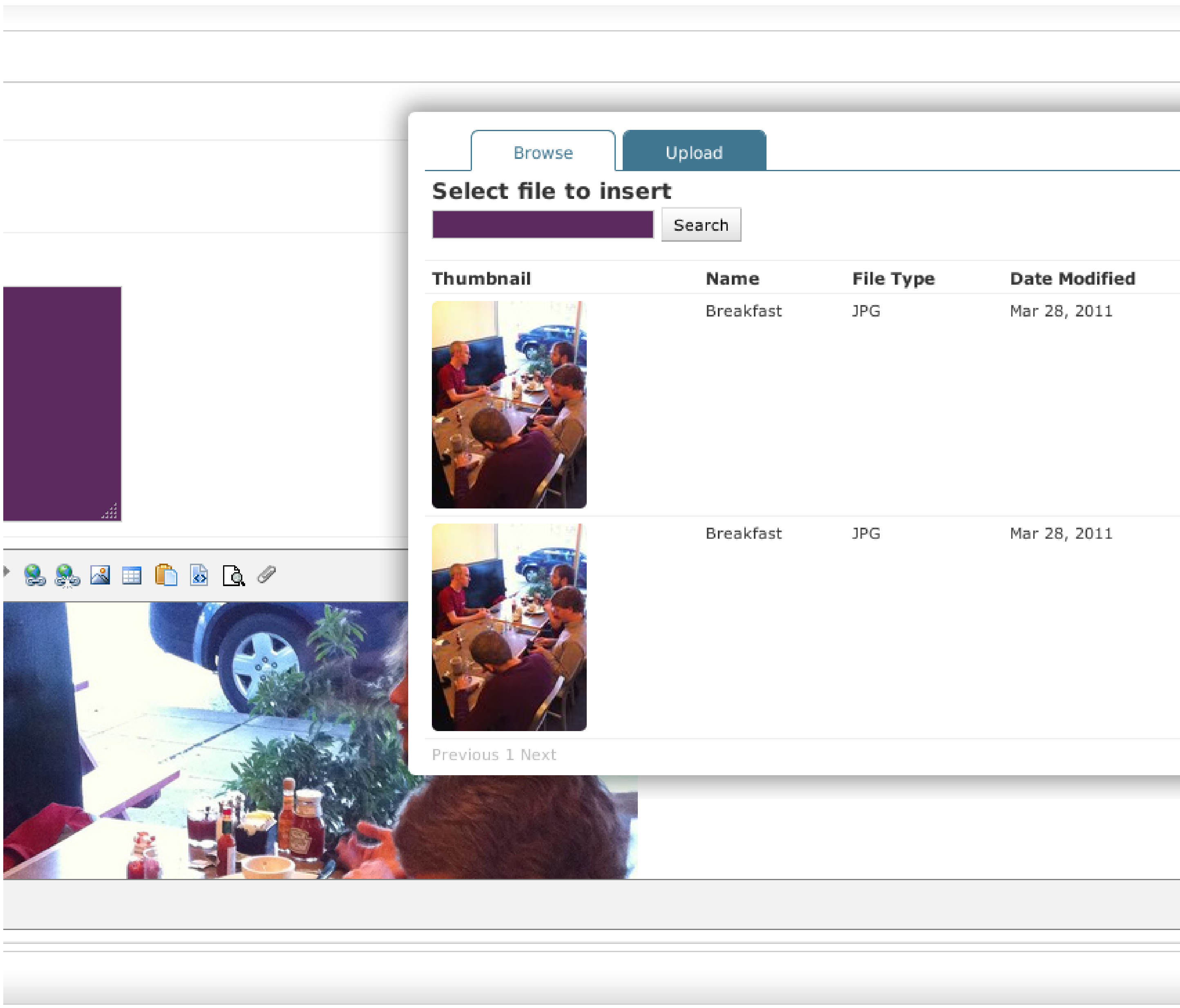
Fig. 3.1: The upload pane from the sample project.



Select file to insert

Thumbnail	Name	File Type	Date Modified
	Breakfast	JPG	Mar 28, 2011
	Breakfast	JPG	Mar 28, 2011

Fig. 3.2: The browser pane from the sample project.



Thumbnail	Name	File Type	Date Modified
	Breakfast	JPG	Mar 28, 2011
	Breakfast	JPG	Mar 28, 2011

Fig. 3.3: An example of inserting an image with File Picker.

Release History

0.9.0 (released 2017-10-31)

- Added Python 3 support

0.8.0 (released 2017-10-27)

- Added support for Django 1.11

0.7.0 (released 2017-10-26)

- Added support for Django 1.8, 1.9 and 1.10
- Dropped support for Django < 1.8

Previous versions

- See commit history for clues.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index