Image manipulation with Cropit

by Laura Barluzzi

Background

One of the features I worked on during my internship at Cadasta [1], is enabling users of the Cadasta platform [2] to upload a profile image as an avatar. I implemented this feature via the free open-source library Cropit [3]. Although my pull request didn't end up being merged due to specific security restrictions preventing the upload of user-created files to Cadasta servers, I want to share what I learned about using the Cropit library with Django since the library provides a great developer experience and would be useful to many projects out there.

Installation

Cropit is an easy to use library to upload, preview, crop and manipulate pictures. First, have a look at the library yourself! Check out the repository [4] and the demos [5].

To install Cropit you can simply add a script tag in the HTML template where the user will upload their profile picture:

```
<script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/cropit/0.5.1/jquery.cropit.min.js"></script>
```

Integration

To integrate Cropit with a Django application, several steps are required, including a model change, a template change and a form view model change. A full working example is included in the Cadasta repository on Github [6] and we'll step through each part in the sections below.

User model

To start, we need to add an avatar field in the user model. Cadasta built django-buckets [7], a library to link files stored on S3 with Django models, which we'll leverage here to ensure that avatars can be displayed without putting additional burden on the database. As such, I declared a new field on the user model called `avatar` which uses `S3FileField`:

```
avatar = S3FileField(upload_to='avatars',
                     blank=True,
                     accepted_types=['image/png', 'image/jpeg'],
                     default='/static/img/avatar_sm.jpg')
```

Template and Javascript

Image preview and customization

Cropit provides a HTML block that enables you to preview the uploaded image. Below you can see my slightly modified version that forces the preview of `user.avatar.url` whenever the page is loaded. If no custom image is provided, the display defaults to the default user avatar declared earlier on the `avatar` field in the user model. This code snippet should be included in the template you're using to edit the user model.

```
<div class="image-editor well file-well" id="image-editor">
  <div class="cropit-preview-wrapper">
    <div class="cropit-preview">
      <img src={{user.avatar.url}} id="user-avatar"/>
    </div>
  </div>
</div>
```

Zoom-in and zoom-out

In addition to enabling the user to reframe their avatar, Cropit also provides the option to crop the image by zooming in and out. You can integrate this feature by adding an input of type range to your template as shown in next HTML block:

```
<div class="slider-wrapper hidden" id="zoom">
    <span class="glyphicon glyphicon-picture"></span>
    <input type="range" class="cropit-image-zoom-input" id="zoom-range">
    <span class="glyphicon glyphicon-picture large"></span>
</div>
```

Linking HTML and Javascript

In order to tell Cropit to turn the blocks defined above into an image editor, we use the following Javascript snippet. A full list of all supported configuration options is defined at the bottom of this page [4].

```
$('#image-editor').cropit({
    // the following function is fired anytime the image changes
    onFileChange: function() {
        avatarChanged = true;
        $("#user-avatar").hide();
```

```
    },
    onZoomDisabled: function() {
        // hide zoom input range when the picture can't be zoomed
        $("#zoom").addClass("hidden");
    },
    onZoomEnabled: function() {
        // show zoom input range when the picture can be zoomed
        $("#zoom").removeClass("hidden");
    },
   // if the image is smaller than preview box, stretch it and fill the box
   smallImage: 'stretch',
 });
```

The hidden input and cropping to base64

Cropit works on top of 2 input tags:

1. The visible input where a user uploads an image from their computer:
   ```
   <input type="file" class="cropit-image-input" id="upload" accept=".jpg,.gif,.png">
   ```
2. The hidden input where Cropit stores the base64 of the already-cropped image which should get uploaded to the Django backend:
   ```
   <input type="hidden" name="base64" id="hidden-image-data" />
   ```

These two inputs need to be synchronized when the submit button of the form with the user-provided image is clicked such that the client-side cropping via Cropit can be communicated to the server-side processing:

```
$('form').submit(function() {
    var croppedImageBase64 = $('#image-editor').cropit('export');
    $('#hidden-image-data').val(croppedImageBase64);
});
```

Now, whenever the user profile form gets submitted, the hidden input tag will be set to the value of our cropped image in base64 that we can access later on. Don't forget to add the new hidden image data field to your user form view model so that the backend can access the value after it got submitted:

```
base64 = forms.CharField(required=False)
```

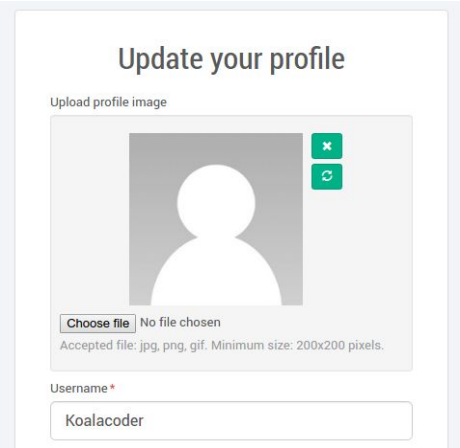
From base64 to PNG in S3

Now that we can access the serialized value of the cropped avatar via the `base64` field on the form view model, we need to convert the serialized cropped image into an actual image that we can then upload to S3 via the previously defined `S3FileField` on the user model.

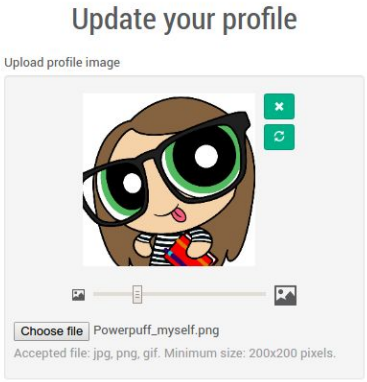
Inside the user form view class, we implement a function `clean_avatar` that enables us to deserialize the base64-serialized cropped avatar into a real file before we store the avatar in the user model:

```
from base64 import b64decode
from tempfile import NamedTemporaryFile

CORRECT_AVATAR_FORMAT = ''

class ProfileForm(forms.ModelForm):
    # ...
    def clean_avatar(self):
        base64 = self.data.get('base64')
        avatar = self.instance.avatar
        if base64:
            if not base64.startswith(CORRECT_AVATAR_FORMAT):
                raise forms.ValidationError('Image data-url format not valid.')
            base64_bytes = base64[len(CORRECT_AVATAR_FORMAT):]
            image_bytes = b64decode(base64_bytes)
            image_file = NamedTemporaryFile('w+b', prefix='avatar-',  suffix='.png')
            image_file.write(image_bytes)
            image_file.seek(0)  # ensure the image can be read later by S3FileField
            avatar.file = image_file
        return avatar
```

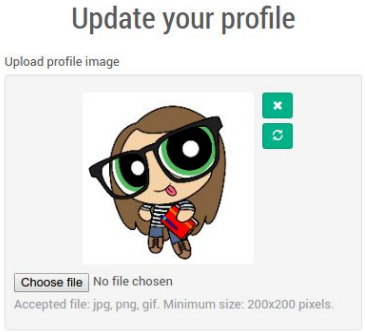
Some screenshots



Preview of the default avatar for a user who didn't already upload a custom image.



Preview of an uploaded image ready to be repositioned and cropped before being saved.



Preview of image currently saved as user avatar. This image has been already uploaded and cropped, therefore in this default view there is no zoom available.

Conclusion

I really enjoyed working with Cropit. I'd advise everyone to use Cropit in case you want to provide a good user experience for avatar uploads while also enabling the developer to personalize and customize the design of the upload control. Another reason why this library is worth trying is the fact that it already provides the base64 of the cropped image. Some other libraries tend to return the coordinates of the cropped section instead, forcing you to crop the image on the server side later on which is an expensive operation.

References

[1] http://cadasta.org
[2] https://github.com/Cadasta/cadasta-platform
[3] https://github.com/scottcheng/cropit
[4] https://github.com/scottcheng/cropit
[5] http://scottcheng.github.io/cropit
[6] https://github.com/Cadasta/cadasta-platform/pull/1643
[7] https://github.com/cadasta/django-buckets