Dateien per Drag & Drop in Apex Applikationen ablegen

Franziska Höcker
MT AG
Ratingen

Schlüsselworte

Apex, Dropzone.js, JQuery, JavaScript, Drag & Drop

Einleitung

Drag & Drop Fileupload ist ein Feature, welches in Apex nativ nicht gegeben ist. Mit Hilfe der freien Dropzone.js Bibliothek ist es in Apex möglich Dateien per Drag & Drop hochzuladen.

Auch das gleichzeitige Hochladen von mehreren Dateien ist möglich. Das Hochladen der Dateien erfolgt direkt nach dem „Loslassen“ der Datei und es ist sichtbar wenn die Datei vollständig hochgeladen wurde.

In nachfolgenden wird der komplette Prozess erläutert, welcher von der Einbindung der Dropzone.js Bibliothek, über die Nutzung in Kombination mit den bisherigen „File-Browse-Item“ bis hin zur Verarbeitung der Dateien in der Datenbank funktioniert. Weiter werden Gestaltungsmöglichkeiten der Dropzone dargestellt, sowie Probleme erläutert.

Datei Upload Möglichkeiten in Apex

In Apex ist es bisher nur möglich eine Datei über ein „File-Browse-Item“ hochzuladen. Dies bietet keine Möglichkeiten Dateien hereinzuziehen oder auch mehrere Dateien gleichzeitig hochzuladen. Die Datei muss über ein Button Klick vom Desktop ausgewählt werden.

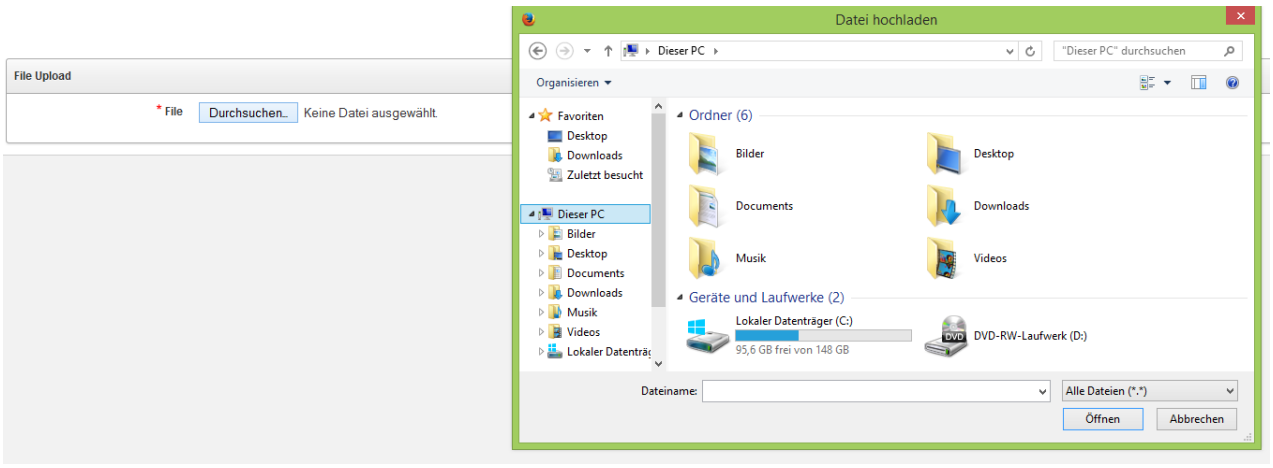

Abbildung 1: Nativer File Upload mit File Browse Item

Die JavaScript Bibliothek Dropzone.js bietet die Möglichkeit Dateien per „Drag & Drop“ in einen definierten Bereich hereinzuziehen. Die Bibliothek ist open Source. Um es zu ermöglichen Dateien per „Drag & Drop“ in eine Apex Applikation hereinzuziehen, kann die dropzone.js in Apex eingebunden werden.

Folgende Tabelle stellt nochmal die beiden Varianten gegenüber:

	Upload Nativ (File Browse Item)	Dropzone.js
Dateien per Klick auswählen	Ja	Ja
Dateien per Drag & Drop hereinziehen	Nein	Ja
Mehrere Dateien gleichzeitig auswählen	Nein	Ja

Tabelle 1: Vergleich Dropzone Upload mit File Browse Item

Die drozone.js Bibliothek beinhaltet kein Upload der Dateien. Dies muss Server-seitig selbst implementiert werden. Dafür wird das File Browse Item als Basis genutzt. Innerhalb dieses Items wird der Speicherort für die Dateien festgelegt. Über das Festlegen der Action der Dropzone bei hereinziehen einer Datei, wird der normale Apex Submit genutzt und somit erfolgt der Upload der Datei nativ, über das File Browse Item in den festgelegten Speicherort.

Einbindung der Dropzone.js Bibliothek in Apex

Die Bibliothek kann direkt auf der Seite http://www.dropzonejs.com/ heruntergeladen werden. Anschließend muss sie als Static File unter Shared Components in Apex hochgeladen werden.

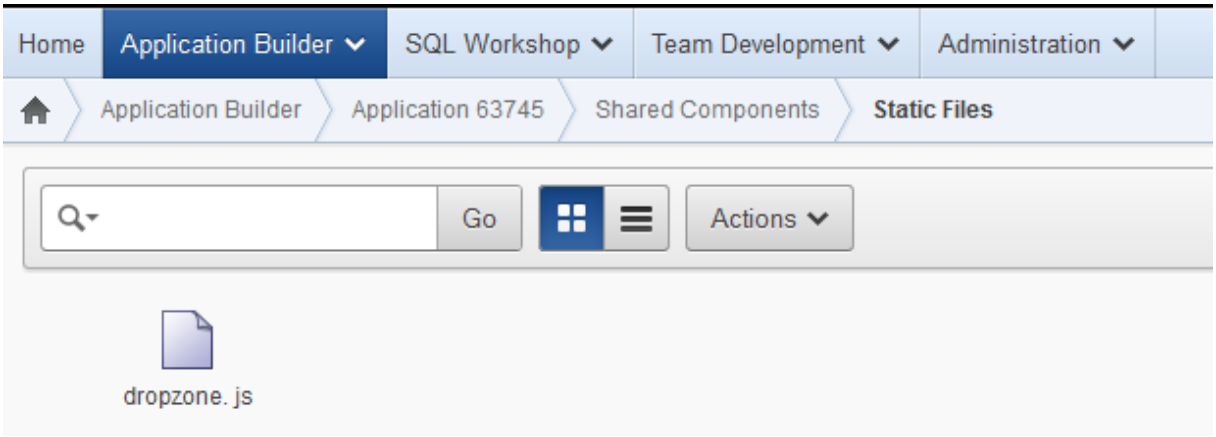

Abbildung 2: dropzone.js als Static File in Apex

Anschließend muss die Bibliothek referenziert werden, dies kann über verschiedene Wege passieren. Ich habe den Aufruf der Bibliothek direkt im HTML Header der entsprechenden Seite (wo später die Dropzone erstellt wird) eingebunden.

```
<script type="text/javascript" src="#APP_IMAGES#dropzone.js"></script>
```

Erstellen einer Dropzone in Apex

Nach Einbindung der Bibliothek kann nun eine Dropzone erstellt werden. Als erstes wird ein File Browse Item benötigt. Dieses wird in einer Region erstellt die nicht angezeigt wird. Im File Browse Item kann nun gewählt werden wo die Datei abgespeichert werden soll. Es muss hier die Tabelle wwv_flow_files (apex_application_files) gewählt werden. Die Dateien können dann später über einen Pl/SQL Prozess in eine individuelle Tabelle auf der Datenbank abgespeichert werden.

Als zweites wird eine neue Region benötigt (HTML Region). Innerhalb dieser Region wird dann ein DIV erstellt.

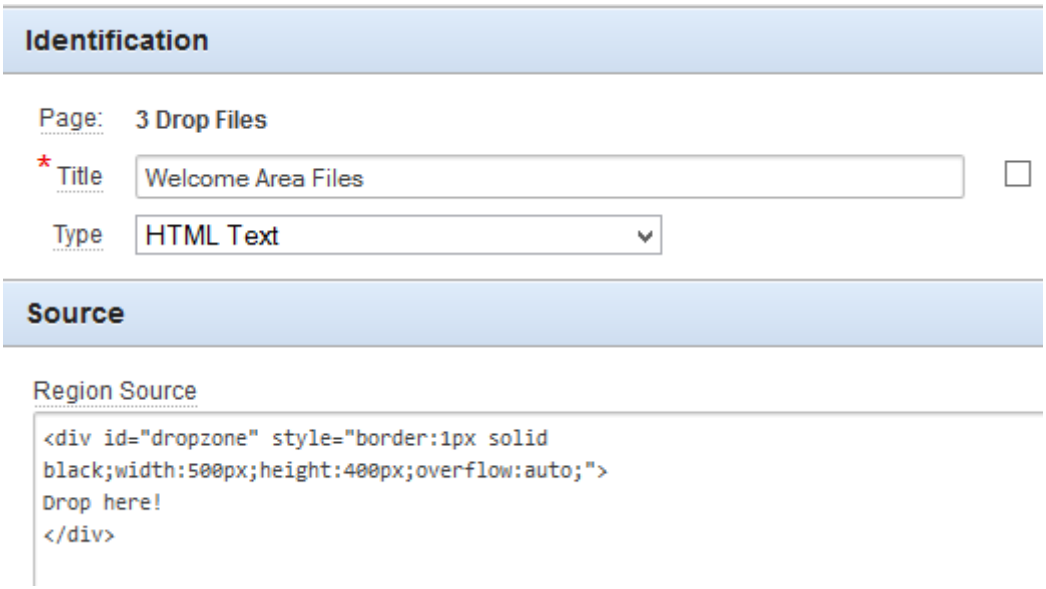

Abbildung 3: Div für Dropzone

Das DIV bekommt eine ID und ein paar Style Eigenschaften zugewiesen. Anhand der ID des DIV's wird dann über JavaScript im Seiten Header eine Dropzone programmatisch erstellt durch die Instanziierung der Dropzone Klasse.

Hier scheint es allerdings einen Bug zu geben, denn die Dropzone erhält nicht die CSS Klasse *dropzone*. Um dies zu beheben siehe Kapitel Layout anpassen.

```
//Create  programmatically a Dropzone-Object & assign some properties
   fDZS = new Dropzone('div#dropzone',
          {
          "url": "wwv_flow.accept", //"Normal" page submit url of APEX
          "params": {
          "p_instance":$('#pInstance').val(),
          "p_flow_id":$('#pFlowId').val(),
          "p_flow_step_id":$('#pFlowStepId').val(),
          "p_page_checksum":$('#pPageChecksum').val(),
          "p_page_submission_id":$('#pPageSubmissionId').val()
          }
          });

   fDZS.options.paramName = vArgName; //Necessary submit-item "upload item name"
```

In diesem Beispiel wird die Dropzone erstellt für das DIV mit der ID dropzone und es wird auch eine URL (Action) übergeben. Hier wird der normale Seiten Submit von Apex genutzt. Außerdem werden noch weitere Parameter (Meta Daten) übergeben die bei einem normalen Apex Post auch notwendig sind. Diese werden alle aus versteckten Variablen der Form der Seite ausgelesen. Zusätzlich muss der ParamName noch gesetzt werden mit dem Namen des Input Items (File Browse Item).

Nach diesen Einstellungen sollte die Dropzone als solche schon funktionieren und es können schon Dateien hereingezogen werden. Allerdings werden diese noch nicht abgespeichert.

Serverseitige Implementierung zum Abspeichern der Daten

Damit die Dateien auch ohne Probleme in die Tabelle wwv_flow_files abgespeichert werden können braucht der Post Prozess noch einige Daten. Ein Post Prozess von einer Seite mit einem File Browse Item mit Save Button sieht wie folgt aus.

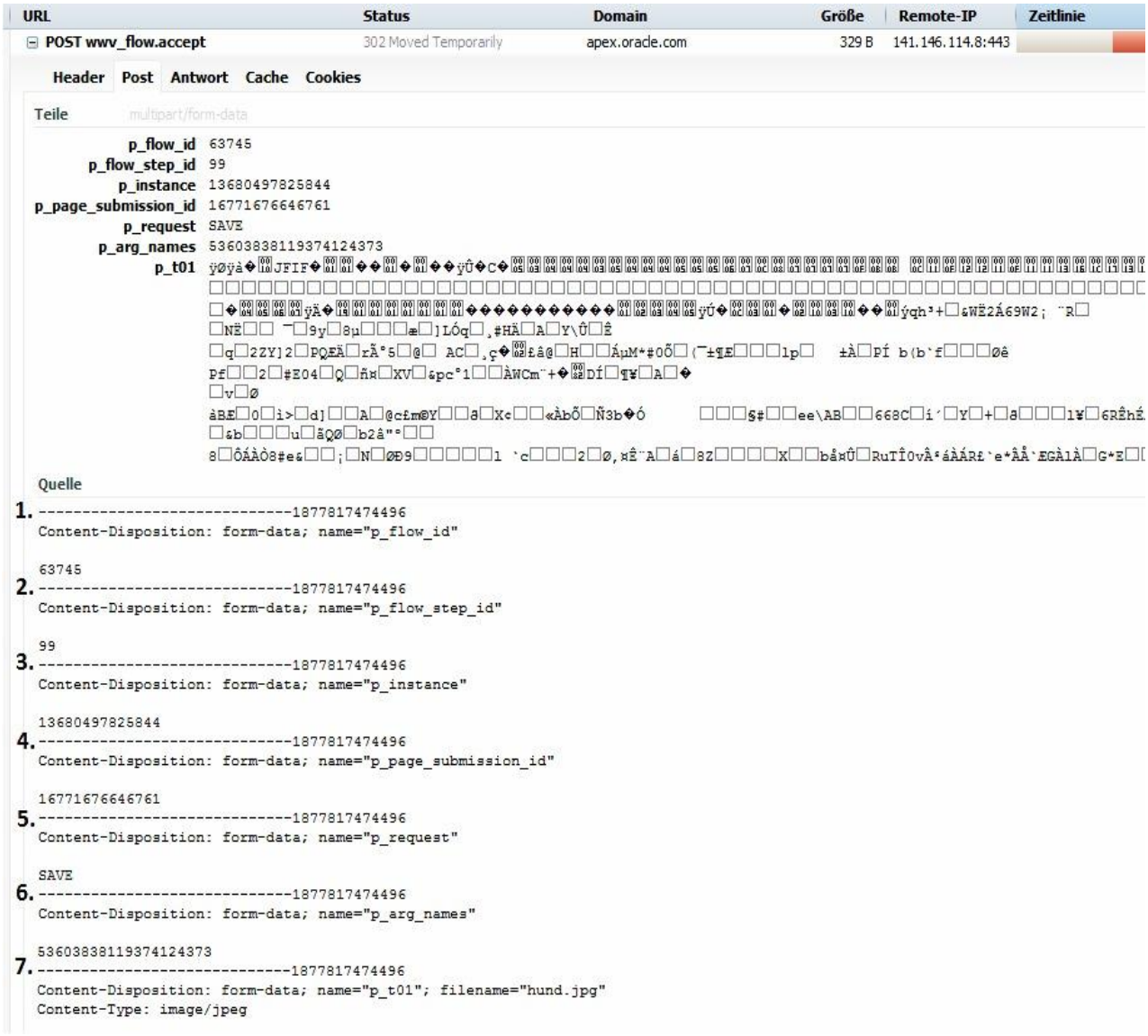

Abbildung 4: Post Prozess für ein File Browse Item

Unter dem Bereich Quelle kann eingesehen werden welche Daten der Post benötigt um eine Submit der Seite auszuführen.

Die ersten vier sind versteckte Variablen innerhalb der Form auf der jeweiligen Apex Seite. Diese können über JQuery abgefragt werden. Beim programmatischen erstellen der Dropzone werden die Werte der ersten vier Variablen übergeben (siehe Abbildung 2). Zusätzlich wird noch der Wert der Variable p_page_Checksum mit übergeben, welche sich im Footer der Form befindet und auch über JQuery abgefragt werden kann.

Abbildung 5: Apex Post eines File Browse Items

Die Variablen 5. und 6. stammen von dem Button Save den es auf dieser Seite gab. Diese Werte sind also in dem Falle für die Dropzone nicht vorhanden.

Als letzte Variable wird der Name des File Browse Items und p_arg_names des Items (Submission ID des Items) übergeben.

Im Seiten Header werden per JQuery die verschiedenen Variablen ausgelesen und die Werte gespeichert und nach dem programmatischen Erstellen einer Dropzone werden die Werte mit der Funktion on über das Event sending übergeben.

Das Auslesen der noch benötigten Werte findet vor programmatischen Erstellen Dropzone statt:

```
//Get the name of the FileUpload field
var vArgName = $('#P4_FILEUPLOAD').attr('name');

//Get the "encoded" p_arg_names parameter of this item
var vArgNameEnc =
    $('#P4_FILEUPLOAD').prev('input[type="hidden"][name="p_arg_names"]').val();
```

Übergeben der ausgelesenen Werte:

```
//The onSending eventhandler is necessary to adding "p_arg_names" entries to the
//FormData object
fDZ.on('sending',function(file, xhr, formData)
      {
       formData.append("p_arg_names",vArgNameEnc);
      });
```

Der vArgName wurde bereits bei der Erstellung der Dropzone als ParamName mit übergeben (siehe oben). Die Dateien werden nun in der Tabelle wwv_flow_files abgespeichert. Der Post Prozess für den Upload einer Datei über die Dropzone sieht dann wie folgt aus:

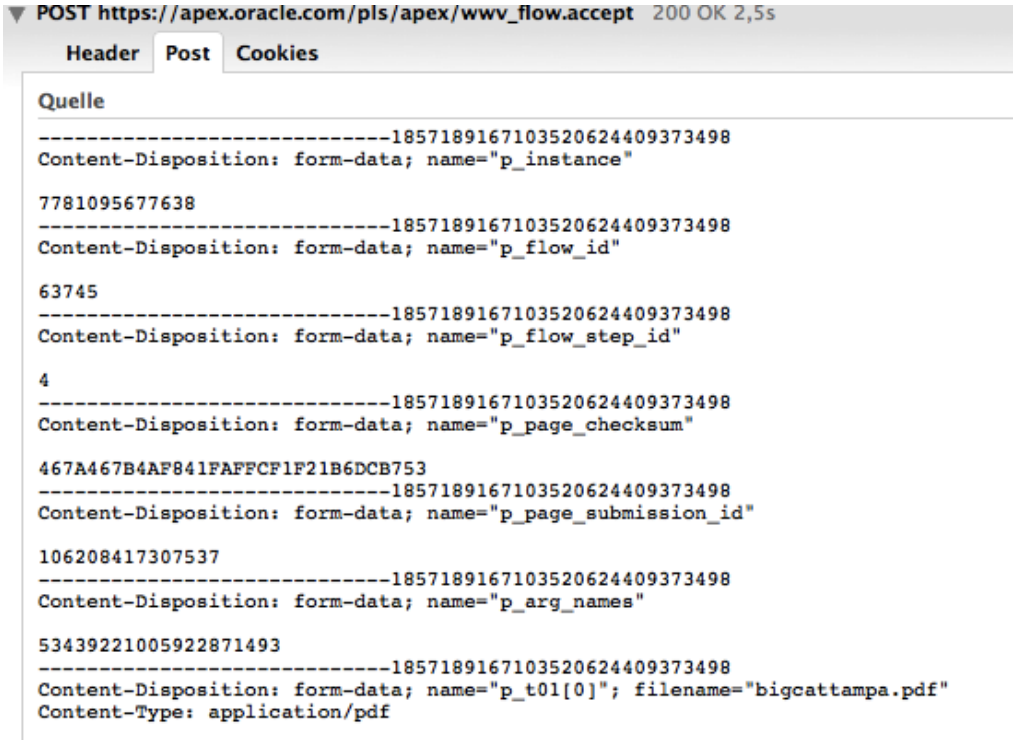

Abbildung 6: Apex **Post für Dropzone Upload**

Allerdings wird es in den meisten Fällen notwendig sein die Dateien in eine eigene Tabelle auf der Datenbank abzulegen.

PL/SQL Prozess zum Speichern der Daten in eine individuelle Tabelle

Um die einzelnen Dateien später aus der Tabelle wwv_flow_files auszulesen werden die Dateinamen benötigt. Das File Browse Item beinhaltet lediglich den Namen der letzten hereingezogenen Datei. Daher wurde ein weiteres Item vom Typ Hidden angelegt. Befüllt wird dieses über die Funktion getFilenames, welches per JQuery die DIV's mit den Dateinamen selektiert und die Namen anschließend konkateniert und in das Item schreibt.

```
function getFilenames() {
names_string="";
$('#dropzone div').find('div.dz-filename').each(function(){
  if (names_string)
  {names_string = names_string + ':' + $(this).text();}
  else{names_string = $(this).text();}
});
$('#P4_FILENAMES').val(names_string);
}
```

Über die Funktion On und das Event Success wird die Funktion getFilenames aufgerufen und anschließend erfolgt ein AJAX Call der das Item mit den Filenames übergibt.

```
fDZS.on('success',function(a,response){
getFilenames();
var ajaxCall = new htmldb_Get(null, &APP_ID.,
               'APPLICATION_PROCESS=PROCESS_UPLOADED_FILES', &APP_PAGE_ID.);
ajaxCall.add('P4_FILENAMES', $(P4_FILENAMES).val());
vRes = ajaxCall.get().trim();
console.log(vRes);
});
```

Der Ajax Call ist in PL/SQL auf der Seite definiert und nutzt die Funktion apex_util.string_to_table um über das Item mit den Filnenames zu iterieren und anschließend pro Filename die Datei aus wwv_flow_files in eine vordefinierte Tabelle auf der Datenbank zu speichern, sofern der Dateiname in der wwv_flow_files Tabelle vorhanden ist. Nach Abspeichern der Datei in eine individuelle Tabelle wird die Datei aus wwv_flow_files gelöscht.

Layout anpassen

Beim programmatischen Erstellen der Dropzone bekommt die Dropzone nicht die CSS Klasse *dropzone* zugeordnet. Die Dropzone sieht daher noch recht „nackt“ aus.

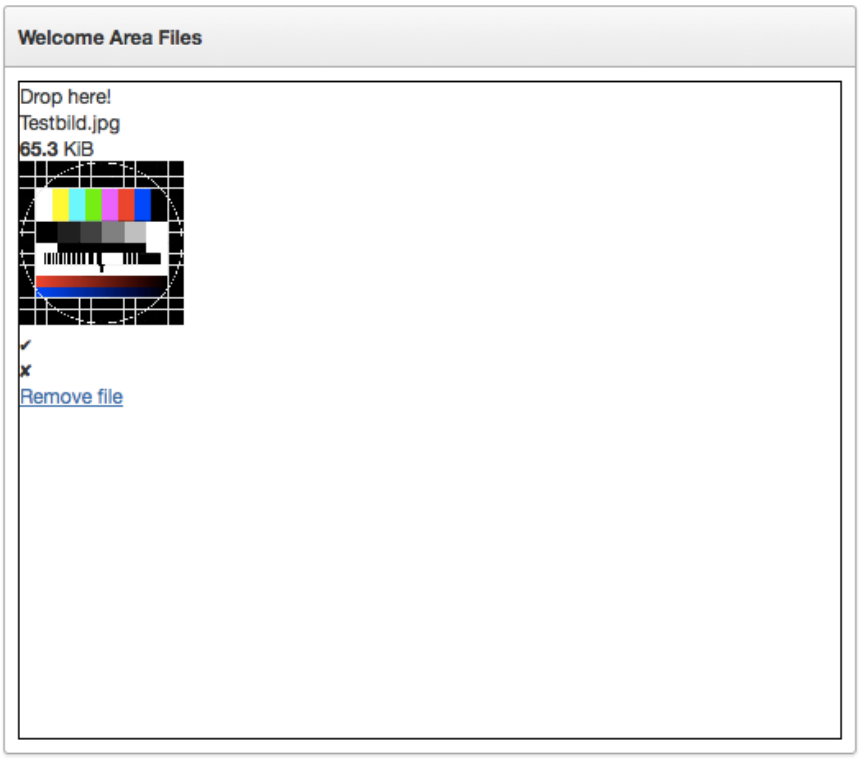

Abbildung 7: Dropzone ohne CSS Klassenzuordnung

Um dafür zu sorgen das die Dropzone die entsprechende Klasse *dropzone* erhält und am Ende so aussieht wie auf der Homepage von Dropzone muss die CSS Datei dropzone.css und jeweils die beiden

vorhandenen Images (spritemap.png, spritemap@2x.png) heruntergeladen werden und anschließend in Apex unter Shared Components hochgeladen werden.
Anschließend kann in der Region wo das DIV erstellt wird die CSS Klasse direkt mit gesetzt werden.

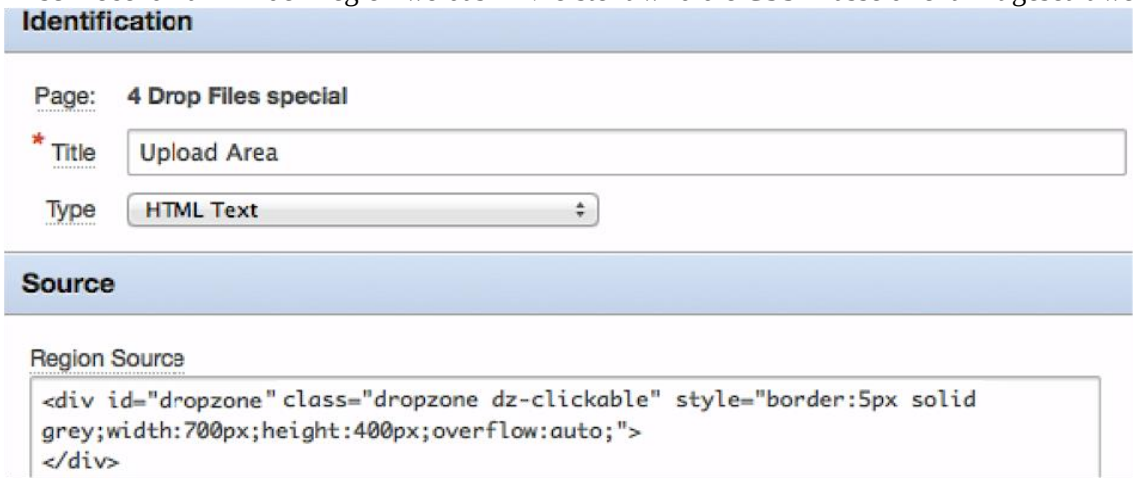

Abbildung 8: Div für Dropzone mit CSS Klasse

Das hat allerdings zur Folge das es einen Fehler gibt („Dropzone schon vorhanden“) beim Erstellen der Dropzone im Seitenheader. Dies kann verhindert werden mit folgendem Aufruf:

```
Dropzone.options.dropzone2 = false;
```

Durch den Aufruf wird die Dropzone nicht weiter betrachtet bei der Ausführung des Javascript Codes im Seitenheader. Es wird die Dropzone programmatisch erstellt und kein Fehler mehr geliefert und die Dropzone hat die Klasse *dropzone*.

Nachdem diese Einstellungen getroffen wurden sieht die Dropzone so aus wie in der Demo auf der Seite von Dropzone.js.

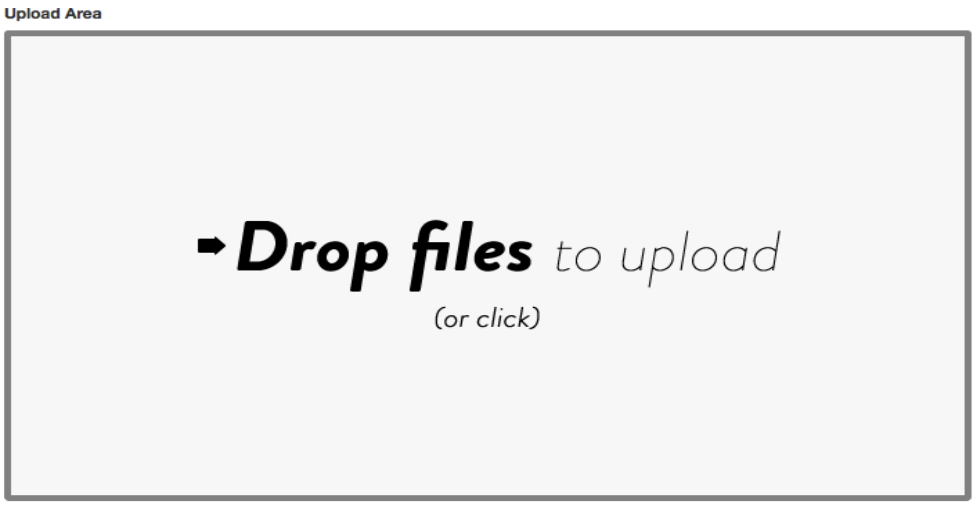

Abbildung 9: Dropzone mit CSS Klasse *dropzone*

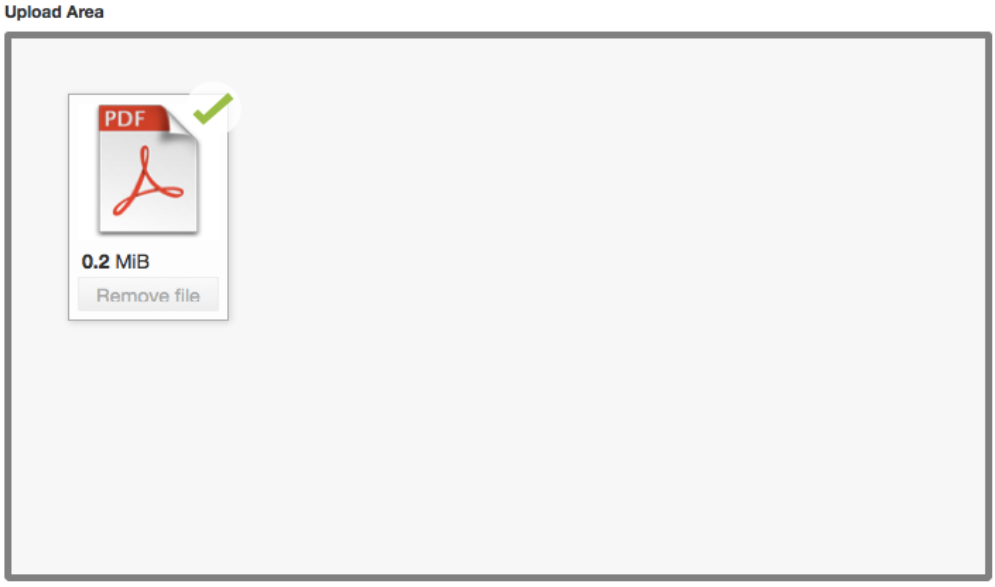

Abbildung 10: Dropzone mit hochgeladener Datei

Am Layout können jederzeit noch Anpassungen durchgeführt werden. Dies kann direkt über die CSS Datei erfolgen. Somit ist die Dropzone noch individualisierbar.

Zum Beispiel sind für diese Dropzone die Thumbnails eingestellt wenn eine Textdatei oder ein PDF hochgeladen wird, wird ein entsprechendes Bild angezeigt.

Weiter Möglichkeiten der Individualiserung

Des Weiteren ist die Dropzone sehr anpassbar. Es gibt viele Einstellungen die getroffen werden können. Einige Beispiele davon sind hier aufgelistet:

- Löschen der Dateien aus Dropzone nach direktem Upload
- Löschen der Datei aus Dropzone bei Klick auf die Datei
- maximale Größe für Dateien

```
- myDropzone.options.maxFilesize = 2;
```

- maximale Anzahl an Dateien

```
- myDropzone.option.maxFiles = 10;
```

- gleichzeitiger Upload von Dateien oder nacheinander

```
- myDropzone.options.uploadMultiple = true;
```

- Remove Link anzeigen zum entfernen der Datei aus Dropzone

```
- myDropzone.options.addRemoveLinks = true;
```

- Thumbnail Anzeige für verschiedene Dateitypen

Quellen

[1] http://www.dropzonejs.com/

Kontaktadresse:

Franziska Höcker11

MT AG

Balcke-Dürr-Allee, 9

D-40882 Ratingen

E-Mail franziska.hoecker@mt-ag.com

Internet: www.mt-ag.com