Image Browser

Team:
Philipp Wandl
Vlad Gunda
Tobias Watzl
Armin Weihbold

Thema

Kurzbeschreibung

Grundidee ist ein einfacher Image Browser (ähnlich http://www.xnview.com/de/xnviewmp/), der aber durch “Plugins” erweiterbar ist. Ein Plugin bekommt vom Image Browser ein oder mehrere Bilder mit Metadaten, sowie einige Parameter, die der Nutzer im GUI einstellen kann. Ein Plugin kann ist ein einfaches Python Script. Ziel ist eine Basis für das Entwickeln und Testen von Bildverarbeitungs-Filtern, oder Machine Learning Bibliotheken zu haben.

Anwendungsgebiet

Die Anwendung soll in einem ersten Schritt im Bereich der einfachen Bildbearbeitung sein. Jedoch soll es auch für die automatisierte Bildverwaltung erweitert werden können.

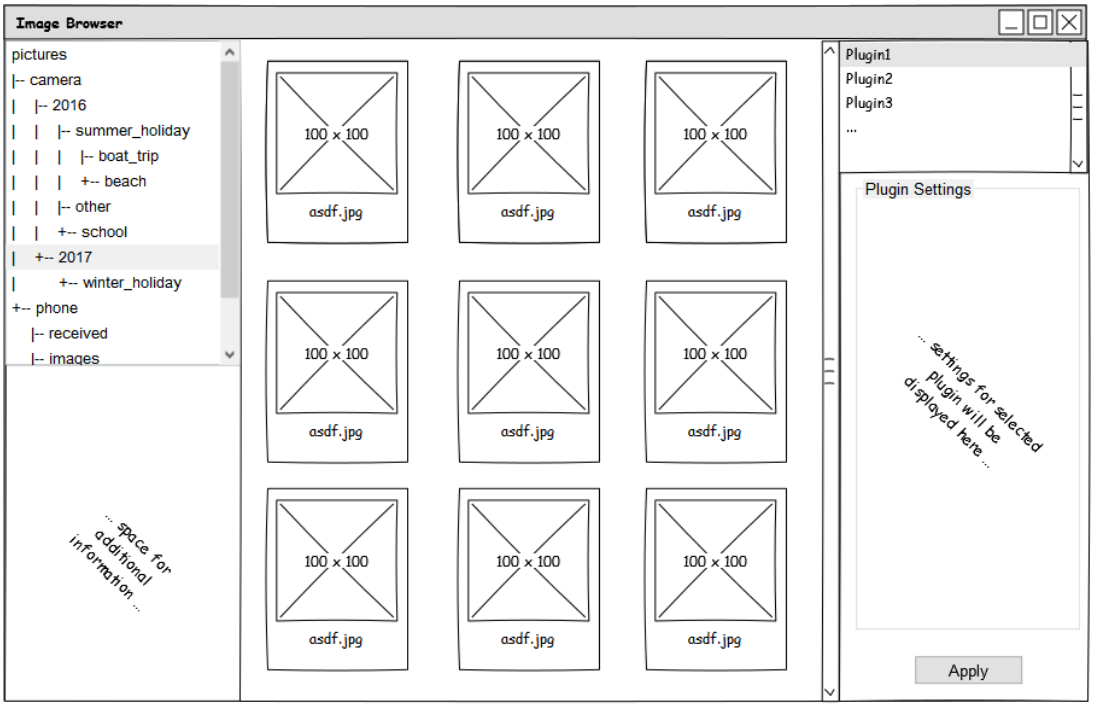
Realisierungsziel

- Laden von Bildern (kann man zumindest von der Architektur auch schon auf Plugins ausrichten, um z.b. RAW Formate unterstützen zu können)
- Anzeige in “File browser” ähnlicher Ansicht
- Anzeige von GUI Elementen zur Einstellung der Plugin Parameter
- Aufruf des Plugins mit ein oder mehreren Bildern und dazugehörigen Metadaten
- Abspeichern der vom Plugin geänderten Metadaten und Bilder
  (Metadaten sind deshalb wichtig, weil es dadurch möglich wird ein auf Machine Learning basiertes Plugin zu erstellen, welche Fotos automatisch tagged)
- Zusatz: Anzeige Plugins (kann durch einfache Ableitung von einer Basisklasse realisiert werden). Dadurch könnte man z.b. Videos abspielbar machen. Bearbeitungs-Plugin, welches statt einem RAW und JPG Bild nur das JPG Bild als Thumbnail angezeigt und beim Bearbeiten der Eigenschaften werden beide Bilder beeinflusst. (Anwendungsfall: eine Kamera die JPG und RAW gleichzeitig aufnimmt)

Entwicklung

Mockup

Als erstes wurde ein UI entworfen um eine Vorstellung des fertigen Projekts zu haben.



Verwendete Technologien

Zur Realisierung dieses Projekts wurden hauptsächlich folgende Technologie eingesetzt:

- Angular, Angular-CLI: für die Darstellung des Frontends im Browser und zum Aufsetzen des Projekts
- Express: als Javascript basierter Server, welcher verschiede REST Endpoints zur Ausführung der Python Plugins und zur Verwaltung des Filesystems bietet
- Python: als Umgebung zur Ausführung der Plugins aus Express mit der Javascript Bibliothek PythonShell
- Typescript: als Erweiterung von Javascript (Typsicherheit)
- Npm: zur Verwaltung der verschiedenen Bibliotheken
- Git, Github: zur Koordinierung des verteilten Arbeitens (https://github.com/mms-imagebrowser/image-browser)

Projektaufbau

Das Projekt wurde mittels Angular-CLI eingerichtet, siehe https://github.com/angular/angular-cli. Wodurch innerhalb von Minuten ein lauffähiges Projekt erstellt werden kann.
Dabei wurde für die 4 Hauptkomponenten eine eigene Component erstellt. Jeweils für den Filetree, die Preview, die Selektion und das Plugin Handling, wobei dieses wiederum aus Selektion und Settings besteht.
Zur Kommunikation mit dem Server wurden 3 Services erstellt, welche in den jeweiligen Components mithilfe der Dependency Injection von Angular verwendet werden können.

Der Server befindet sich ebenfalls im selben Projekt im Ordner api. Die Einrichtung in Kombination wurde von diesem Projekt: Demo.Express.Angular-CLI, übernommen und entsprechend angepasst. Dabei wurde ebenfalls auf gulp zum Starten des Servers verwendet, welches aber einfach mit einem npm task im package.json aufgerufen werden kann.

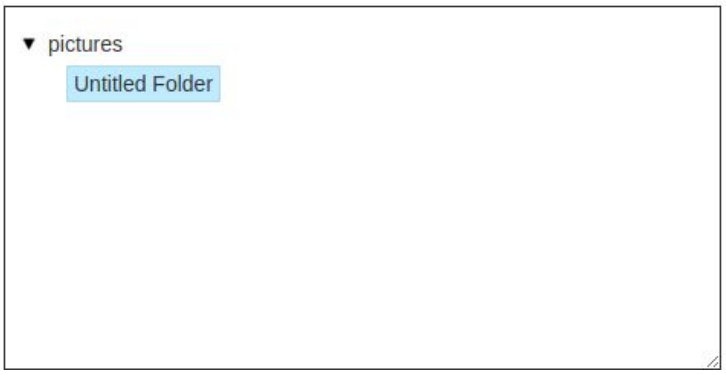
File Tree

Zur Realisierung des File Trees wurde ein externe Bibliothek zur Darstellung von Trees in Angular verwendet: https://github.com/500tech/angular-tree-component

Die Abfrage der Daten erfolgt dabei mittels eines einfachen list abfrage am Server die wiederum eine externe Bibliothek zum Aufbau eines trees verwendet https://github.com/mihneadb/node-directory-tree.
Der Request wird dabei vom FileSystemService ausgeführt.

Da diese Trees jedoch nicht genau kompatibel sind muss die Antwort vom Server noch in die von angular-tree-component benötigte Form konvertiert werden. Dies geschieht durch Iteration des erhaltenen Trees und hinzufügen der einzelnen Element mittels addTreeNode.

Es wurde aktuell noch nicht viel Wert auf das Design gelegt, dies kann jedoch relativ einfach angepasst werden. Jedoch kann es bereits resized werden und es wird bei zu vielen Elementen eine Scrollbar angezeigt




1g7jb1tbmnvy.jpg

Selektion von Bildern

Die Selektion der Bilder in der Mitte wurde mit einem SelectionService realisiert, die Kommunikation mit den anderen Komponten der Applikation funktioniert mit Hilfe von Observables aus der rxjs Bibliothek. Dies sind Objekte die eine leichte Event-basierte Kommunikation ermöglichen. Die Bilderselektions-Komponente beobachtet über diesen Service Änderungen in der Auswahl der Ordner in der Filetree Komponente. Sie benutzt ebenfalls das node-directory-tree um eine Liste aller Bilders des ausgewählten Ordners zu erhalten. Das responsive frontend für die Darstellung der Liste basiert auf bootstrap-3 https://github.com/twbs/bootstrap. Die Darstellung passt sich an die Gößе des Browserfensters an und funktioniert damit auf Desktop- sowie auch auf Mobilgeräten. Weiters bedient die Komponente ihrerseits ein Observable des SelectionServies das Änderungen der Bilderauswahl an alle Subscriber weitergibt, vor allem an die Plugin-Ausführungs Komponente.

Plugin-System

Das Plugin-System des Image-Browsers bietet eine Möglichkeit, die Anwendung durch Plugins um verschiedenste Funktionalitäten zu erweitern.
Zur Zeit werden Python-Plugins unterstützt, eine Erweiterung des Pluginsystem zur Unterstützung zusätzlicher Technologien wäre denkbar.
Plugins können über eine REST-API verwaltet werden. Die Ausführung der Plugins erfolgt im Backend, auch diese kann via http ausgelöst werden.
Momentan existieren Anzeige- und Transformationsplugins.
Anzeigeplugins dienen als Filter mittels Analyse einzelner Mediendateien. Der Aufrufer kann somit feststellen, ob Bilder momentan angezeigt werden sollen, oder nicht.
Transformationsplugins dienen zur Weiterverarbeitung von Medien. Hierbei können Daten verarbeitet und in veränderter Form wieder gespeichert werden, wobei die Ursprungsdatei jedoch nicht verändert werden sollte. Der Aufruf erhält den Pfad zur Ergebnisdatei, die Anzeige dieser liegt danach in der Verantwortung des Aufrufers.

Die verschiedenen Plugin-Typen können natürlich kombiniert werden, so könnte zum Beispiel ein Transformationsplugin zusätzlich Hinweise geben, welche Ergebnisse angezeigt werden sollen, und welche nicht.
Die Plugin-Schnittstelle wurde bewusst simpel gestaltet, damit zukünftige Plugin-Ideen nicht durch eine restriktive Schnittstelle eingeschränkt werden. Nachteilig bei diesem Ansatz ist allerdings die beschränkte Möglichkeit zur statischen Verifikation.
Plugins können entweder über die API hochgeladen werden, oder einfach direkt in den plugins-Ordner kopiert werden.

Plugin-Management-API

Folgende HTTP-Requests können zur Verwaltung von Plugins im Backend verwendet werden:

GET /api/plugins/
Liefert ein Json-Array mit den verfügbaren Plugins.
POST /api/plugins/
Request-Body:
{ "title": "..", "code:"}
Erstellt oder überschreibt ein Plugin.
GET /api/plugins/{name}
Liefert die gespeicherten Daten eines Plugins im gleichen Format wie beim Upload.
DELETE /api/plugins/{name}
Löscht das Plugin.

Plugin-Execution-API

Die folgende API dient zur Ausführung der vorhandenen Plugins.

POST /api/plugins/{name}name/execute
Query-Paramerter:
- *action: erforderlich, momentan nur "info" und "execute" erlaubt*
- *imgPath: Pfad zur zu verarbeitenden Mediendatei. Grundsätzlich optional, allerdings bei "execute" von den Plugins benötigt*

Request-Body: Zusätzliche Optionen, die an unverändert an das Plugin weitergegeben werden als JSON-Object
Response: Das Ergebnis der Plugin-Ausführung als JSON-Object.

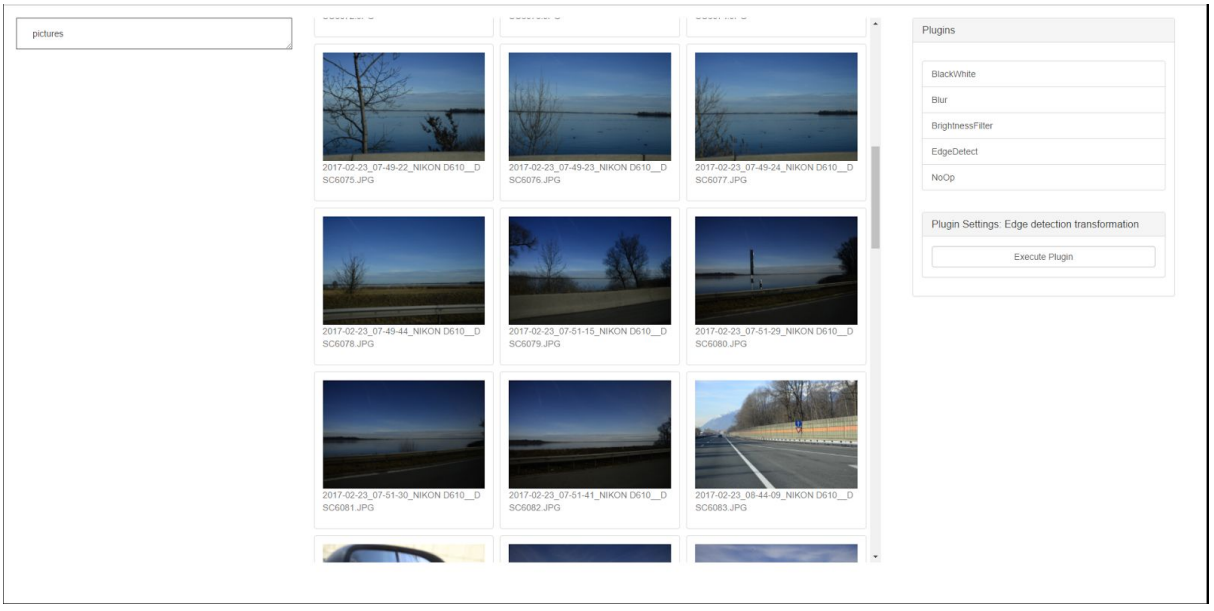
Die Daten in den JSON-Objekten im Request- und Responsebody unterliegen grundsätzlich keiner Einschränkung. Allerdings müssen die Plugins gewisse Konventionen einhalten, wenn die Rückgabewerte vom Frontend verarbeitet wollen. Display-Plugins sollten daher ein "display"-Property(Boolean) setzen, alle Plugins die ein Ergebnis produziert und gespeichert haben (zB Transform-Plugins) sollten via "newPath" (String) dem Aufrufer den Speicherort der neu erstellten Datei anzeigen.

Python-Plugins

Python-Plugins müssen nur die zwei Actions, “info” und “Execute” als Funktionen implementieren, um funktionieren zu können. Beide Funktion erhalten jeweils noch die vom Aufrufer übermittelten Optionen.
Die Python Plugins werden von einem PluginLoader geladen. Dieser lädt die Skripts und ruft die zur jeweiligen Action passende Funktion auf.
Bei der Implementierung der Plugins existieren keine Beschränkungen. Es können auch beliebige Fremdbibliotheken eingebunden werden, falls sie auf dem System verfügbar sind.
Zum Aufruf des in Python geschriebenen PluginLoaders aus dem Node-Backend wird die frei verfügbare Bibliothek “Python-Shell” verwendet.

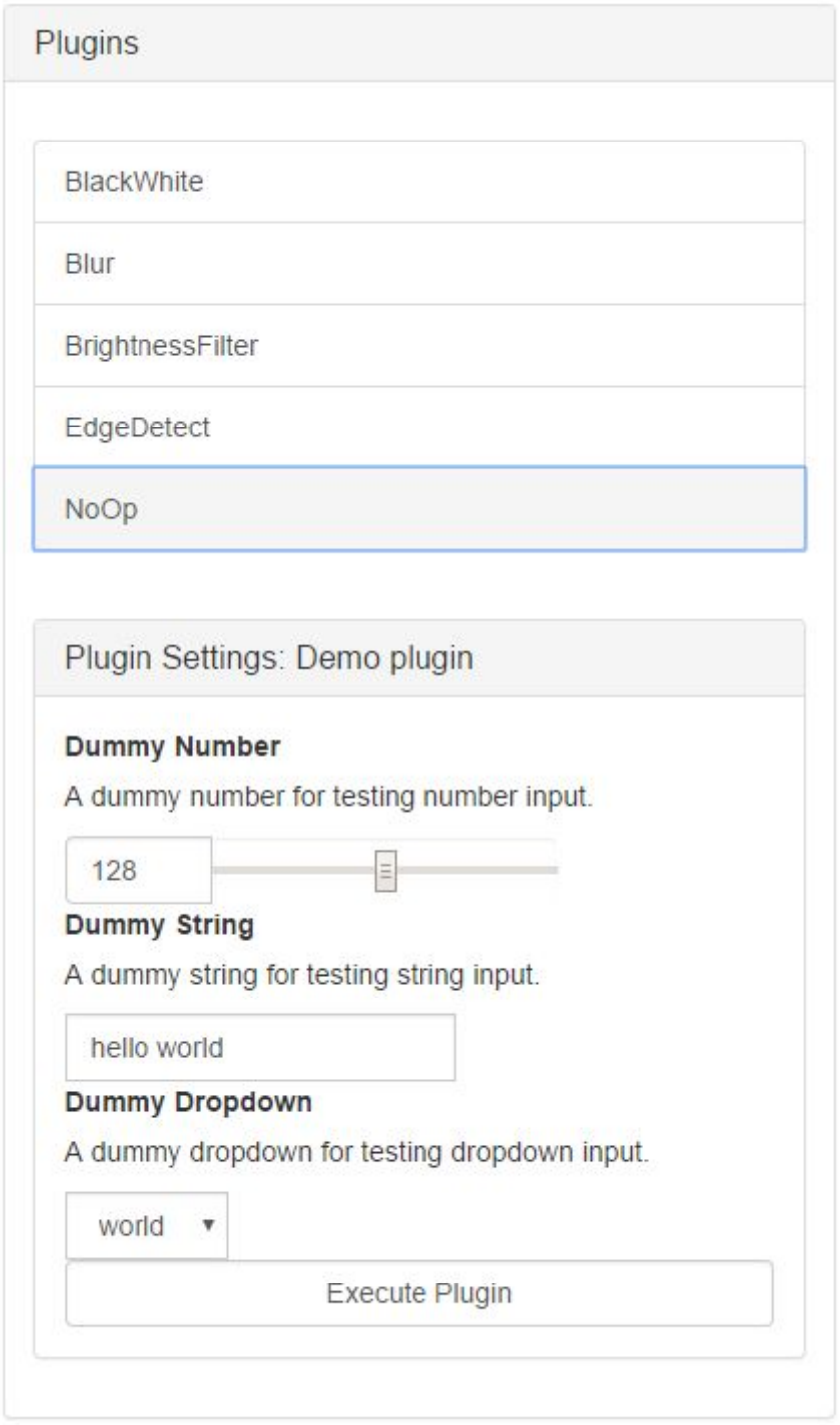
Ausführen eines Plugins

Im rechten Bereich der Benutzeroberfläche befinden sich die Steuerungselemente für die Plugins. Hier werden alle Plugins automatisch gelistet, sofern sie beim Start des Servers als Python Dateien im Plugins Ordner vorhanden sind.



Über die “info“ Action die vom Plugin bereitgestellt wird, wird vom Server Information zum Plugin bereitgestellt. Diese Information wird benutzt um dynamisch die Bedienelemente für die Einstellungen zu erzeugen.

Die Bedienelemente unterstützen momentan “string”, “number” und “dropdown” Werte. Ein Plugin kann weiters Angeben, ob es nur die Einzelverarbeitung oder Batch Processing unterstützt. Je nachdem ruft der Server das Plugin mit unterschiedlichen Eingabedaten auf. Plugins, die Batch Processing unterstützen werden einmal mit allen aktuell selektierten Bildern aufgerufen. Plugins, die kein Batch Processing unterstützen, werden für jedes ausgewählte Bild einzeln aufgerufen.



Sind alle Parameter eines Plugins gesetzt, so kann das gewählte Plugin auf die ausgewählten Bilder angewendet werden.
Die Bilder werden dann vom Plugin gespeichert und es wird ein Pfad zurückgegeben. Momentan ist der Status der Plugin Ausführung noch nicht auf der Benutzeroberfläche sichtbar.

Frontend Plugin Service

Das Frontend Plugin Service wird für die Kommunikation mit dem Server verwendet. Nach dem Single Responsibility Principle (vgl. Bob Martin) ist das Frontend Plugin Service, die einzige Komponente, die mit dem Plugin-Execution und Plugin-Info API kommuniziert. Über Observables (RxJS) werden die Liste der Plugins, sowie das aktuell ausgewählte Plugin asynchron zur Verfügung gestellt.

Jedes mal, wenn ein Plugin in der Liste ausgewählt wird, wird die Information (mögliche Einstellungen, Plugin Name, etc.) vom Server geladen um immer aktuelle Daten zur Verfügung zu stellen.

Sicherheitsrisiken

Da bei der Entwicklung der Plugins keinerlei Grenzen gesetzt werden, könnte natürlich auch Schadcode über ein Plugin ausgeführt werden. Daher ist es wichtig, den Zugriff auf die Anwendung extern zu beschränken.

Quellen/Links

Angular https://angular.io/
JavaScript Framework unterstützt von Google
Express https://expressjs.com/de/
Minimalistische Webframework für Node.js
Node.js https://nodejs.org/en/
Angular CLI https://github.com/angular/angular-cli
Pillow https://python-pillow.org/
Python Library zur Verarbeitung von Bilddateien
Python-Shell https://github.com/extrabacon/python-shell
Library zum Aufruf von Python-Skripts aus Node.js Anwendungen
Angular-tree-component https://github.com/500tech/angular-tree-component