**HOCHSCHULE HEILBRONN**

**Hochschule für Technik, Wirtschaft und Informatik**

**Studiengang Wirtschaftsinformatik (WIN)**

Projektstudie IT-Systeme

**Beets WebFrontEnd**

**Entwicklungsdokumentation**

vorgelegt bei

Professor. Dr. Detlef Kreuz

von

Strahinja Ivanovic;

Michele Santoro;

Christian Elser;  
Vedad Hamamdzic;

im

WINTERSEMESTER 2015/16

Inhaltsverzeichnis

[Abbildungsverzeichnis III](#_Toc433321830)

[1. WebFrontEnd im Anwendungskontext I](#_Toc433321831)

[1.1. Einordnung des Plug-Ins in den Kontext I](#_Toc433321832)

[1.2. Kommunikation zwischen WfePlugin und Beets I](#_Toc433321833)

[2. Struktur des Plug-Ins II](#_Toc433321834)

[2.1. Methoden und Klassen des Plug-Ins II](#_Toc433321835)

[2.2. Testdokumentation der Methoden/Klassen/Module V](#_Toc433321836)

[2.3. Player Funktionalität der Beetsplugin WebGui V](#_Toc433321837)

# Abbildungsverzeichnis

[Abbildung 1: Kommunikation zwischen Plug-In und Beets](#_Toc431336080) 1

[Abbildung 2: *Beetsplugin Player* 2](#_Toc431336080)

# WebFrontEnd im Anwendungskontext

## Einordnung des Plug-Ins in den Kontext

Das beetswfe (beets **w**eb**f**ront**e**nd) ist ein externes Plug-In für die Applikation Beets. Das Plug-In soll als installierbare Erweiterungen des Hauptprogramms „Beets“ entwickelt werden, und ist somit in funktionaler Weise abhängig von Beets. Somit lässt sich festhalten, dass das Plug-In ohne das eigentliche Beets-Programm nicht lauffähig ist.

## Kommunikation zwischen WfePlugin und Beets

Beets bietet eine standardisierte BeetsPlugin-Klasse, welche zur Kopplung der beiden Applikationen verwendet wurde. Das beetswfe-Plugin erbt also von der BeetsPlugin-Klasse.

Damit das geschriebene Plug-In von Beets als solches identifiziert werden kann, muss dieses in einem einheitlich benannten Ordner „beetsplug“ abgelegt werden. Das Plug-In importiert das Modul „commands“ aus dem Ordner „ui“ von Beets, um die entsprechenden Kommandozeilenaufrufe aus dem Plug-In heraus zu ermöglichen. Solche Kommandozeilenaufrufe erfolgen bislang beim Starten des Plug-Ins (manuell, durch den User), sowie beim Importieren von Dateien.

Des Weiteren kommunizieren Beets und das WfePlugin über die von Beets bereitgestellte Konfigurationsdatei *„config\_default.yaml„*, welche für den Projektkontext angepasst wurde. Das Plug-In nutzt außerdem die Datenbank „library.db“ von Beets, über die Beets-API, und besitzt somit bislang keine eigene.

Den Einzigen direkten Aufruf des Beets-Programms erfolgt beim Importieren der Dateien über das Plug-In. Detailliertere Informationen zum Ablauf des Imports findet man unter dem Menüpunkt [2.1 – Methoden und Klassen des Plug-Ins](#_Methoden_und_Klassen)

Insbesondere veranschaulicht folgende Abbildung die Kommunikation:

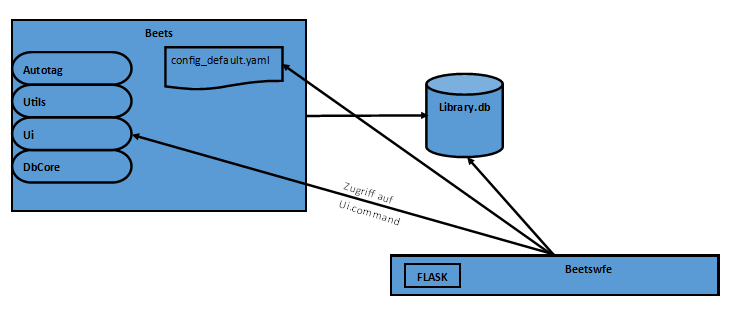


Abbildung 1: Kommunikation zwischen Plug-In und Beets

# Struktur des Plug-Ins

## Methoden und Klassen des Plug-Ins

Das Plug-In lehnt sich architektonisch stark an eine MVC-Struktur an, weißt jedoch darüber hinaus noch weitere Spuren diverser andere Architekturstile auf, wie beispielsweise dem funktionalen Paradigma. Es setzt sich somit aus folgenden Modulen und Domänen zusammen:

* \_\_init\_\_.py
* WebGui.py
* Controllers
  + \_\_init\_\_py
  + Flask\_routing\_controller.py
* Models
  + \_\_init\_\_.py
  + Importer.py
  + Player.py
  + Playlist\_handler.py
  + Responder.py

Die WebGui.py beinhaltet die WfePlugin-Klasse, welche von der Klasse BeetsPlugin erbt.

Das WfePlugin beinhaltet folgende Methoden:

* **\_\_init\_\_**
  + gehört zur Klasse „WfePlugin“
* **commands**
  + gehört zur Klasse „WfePlugin“
  + wurde von Beets übernommen
  + sorgt dafür, dass Befehle aus der Kommandozeile heraus ausgeführt werden können.
* **func**
  + gehört zur Klasse „WfePlugin“
  + wurde von Beets übernommen
  + zieht sich die benötigten Informationen/Parameter aus der Konfigurationsdatei von Beets „config\_default.yaml„
* **connect\_db** 
  + gehört zum Modul
  + baut die Verbindung zur Datenbank auf

Die Domäne Controllers beinhaltet lediglich 2 Module. Einerseits die \_\_init\_\_.py, und andererseits den tatsächlichen Controller: Flask\_routing\_controller.py. Diese bietet das Grundgerüst für eine MVC-Architektur und wird benötigt um Verlinkungen und Weiterleitungen über die einzelnen architektonischen Schichten der Applikation zu ermöglichen:

Der Flask\_routing\_controller.py beinhaltet folgende Methoden:

* **before\_request**
  + gehört zum Modul
  + wird vor dem eigentlichen Routing (beim Starten des Plugins) aufgerufen um vorab die Gegebenheiten des Controllers zu gewährleisten
* **index**
  + gehört zum Modul
  + leitet beim Starten des Plugins auf die Startseite (index.html) weiter und gibt die Items aus der Library als Response mit.
* **item\_file**
  + gehört zum Modul
  + die URL übergibt die ID der ausgewählten Title und gibt diese der play\_file Funktion weiter
  + verweist über das Anklicken eines Titels, wählt das entsprechende Lied aus und lässt es über eine andere Funktionalität abspielen.
* **import\_music**
  + gehört zum Modul
  + wird zur Weiterleitung auf die import.html verwendet.
* **all\_items**
  + gehört zum Modul
  + gibt alle Items der Startseite zurück
* **create\_playlist**
  + gehört zum Modul
  + erstellt eine Playlist über den Aufruf der add\_playlist – Methode aus dem Model

Der Ordner Models beinhaltet die Methoden die die Geschäftslogik der Applikation abbilden. Diese werden benötigt um der Anwendung Leben einzuhauchen:

Die importer.py beinhaltet folgende Methoden:

* **allowed\_file**
  + gehört zum Modul
  + prüft, ob die hochzuladende Datei von der Applikation unterstützt wird.
  + Es sollen schließlich nur .mp3 , .ogg, .wave – Dateien hochgeladen werden.
* **upload**
  + gehört zum Modul
  + lädt die eigentliche Datei hoch, indem die Methoden **import\_cmd** und **remove\_files** in entsprechender Reihenfolge aufgerufen werden.
  + Speichert die hochzuladenden Dateien in einem temporären Ordner, welcher nach dem Import wieder entleert wird.
* **import\_cmd**
  + gehört zum Modul
  + ruft einen Konsolenbefehl auf, welcher einen Beets-Aufruf (Import-Befehl) startet.
* **remove\_files**
  + gehört zum Modul
  + leert den in der **upload**-Methode definierten Ordner

Die player.py beinhaltet folgende Methoden:

* **play\_file**
  + gehört zum Modul
  + erhält die entsprechenden Informationen aus dem Controller und sendet die Daten an die index.html weiter, damit diese mittels Javascript den Player starten kann.

Die playlist\_handler.py beinhaltet folgende Methoden:

* **add\_playlist**
  + gehört zum Modul
  + erstellt eine Playlist als .m3u Datei im entsprechenden Ordner mit dem Namen „playlist\_folder“
* **list\_playlist**
  + gehört zum Modul
  + lädt alle vorhanden .m3u Dateien aus dem Hauptverzeichnis in die index.html

Der responder.py wurde aus dem bestehenden Web-Plugin von Beets entnommen.

## Testdokumentation der Methoden/Klassen/Module

## Player Funktionalität der Beetsplugin WebGui

Die Player Funktionalität im Beetsplugin ist eine essentielle Funktion im WebGui, dies ermöglicht einige Einstellungen zu tätigen um eine reibungslose Musik Vielfalt zu erlangen, siehe Abb. 2.

Die integrierten Möglichkeiten im Player sind unteranderem:

* Aktueller Songtitel:
  + Wird im oberen Segment Angezeigt.
* Play und Pause Funktionen:
  + Wird durch Standardisierte Bilder, im Linken Segment Angezeit.
* Lautstärkeregelung:
  + Kann durch Verschiebung der leiste eingestellt werden.
* Vorspulen eines Songs in der Progressbar:
  + Kann ebenfalls durch Verschiebung der Leiste Verschoben werden.
* Zeitanzeige:
  + Wird unterhalb der Progressbar Angezeit.

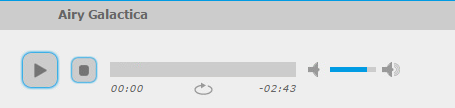


Abbildung Beetsplugin Player