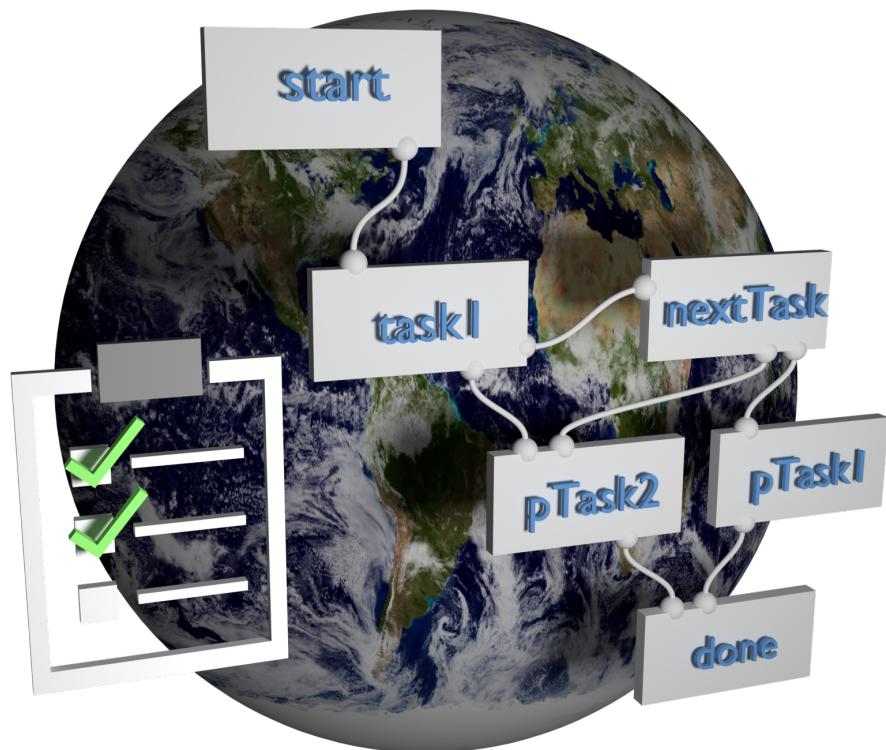


Entwurf

Workflow System für eine virtuelle Forschungsumgebung für Geodaten

Denis Gensh, Marcel Herm, David Krah,
Eduard Kukuy, Daniel Milbaier, Richard Rudolph



28. November 2017
Karlsruher Institut für Technologie, SCC

Inhaltsverzeichnis

1 Einleitung	1
1.1 Entwurfsziele	1
2 Architektur	2
2.1 Aufteilung in die Pakete	2
2.1.1 Client	2
2.1.2 Server	3
2.1.3 WPS-Server	3
3 Klassen und Pakete	4
3.1 Server Klassen	4
3.1.1 edu.kit.scc.pseworkflow.views	4
3.1.2 edu.kit.scc.pseworkflow.cron	7
3.1.3 edu.kit.scc.pseworkflow.models	9
3.2 Client Klassen	16
3.2.1 services	16
3.2.2 models	19
3.2.3 components	24
4 Sequenzdiagramme	33
5 Datenbank	39
5.1 Tabellen	39
5.1.1 Session	39
5.1.2 WPSProvider	40
5.1.3 Edge	40
5.1.4 Process	40
5.1.5 WPS	40
5.1.6 Task	40
5.1.7 Workflow	40
5.1.8 InputOutput	40
5.1.9 Artefact	40
6 URLs	41
6.1 REST API	41
6.2 URL Commands	42
6.3 URLs mit HTML Response	42
7 Entwurfsmuster	43
7.1 Strategy	43
7.2 Object Pool	44
7.3 Observer	44
8 Gantt	45
8.1 Einleitung	45

Inhaltsverzeichnis

8.2	Beschreibung	45
8.3	Diagramm	45
9	Anhang	48
9.1	Klassendiagramm	48

1 Einleitung

Dieses Entwurfsdokument dient der Beschreibung der Architektur unserer Webanwendung WPSflow. Neben den Paket-, Klassen- und Methodenbeschreibungen enthält das vorliegende Dokument verschiedene Arten von **UML** Diagrammen die einzelne Abläufe wie das Bearbeiten eines Workflows oder die Kommunikation zwischen der Anwendung und den -Servern anschaulich darstellen.

Das Komponentendiagramm zeigt die Struktur des modellierten Systems und die Schnittstellen der Komponenten.

Das Entity Relationship (ER) Diagramm zeigt, wie die Daten in einer relationalen Datenbank verwaltet werden.

Die Sequenzdiagramme zeigen den zeitlichen Ablauf typischer Vorgänge.

Im Anhang finden Sie ein großformatiges Klassendiagramm, wobei die Klassen der Übersichtlichkeit halber farblich markiert sind.

1.1 Entwurfsziele

In der Entwurfsphase wurde auf die folgende Aspekte besonders viel Wert gelegt:

- Erweiterbarkeit

Das WPSflow Workflow-System soll zum Zwecke einer zukünftigen Weiterentwicklung erweiterbar sein

- Veränderbarkeit

Es soll möglich sein einzelne Module des Workflow Systems einfach zu verändern

- Nutzerfreundlichkeit

Die grafische Benutzeroberfläche (im folgenden GUI genannt) soll möglichst intuitiv sein. Bei auftretenden Fehlern oder ungültigen Eingaben soll dem Benutzer ein entsprechender Hinweis angezeigt und die Möglichkeit einer erneuten Eingabe angeboten werden.

2 Architektur

2.1 Aufteilung in die Pakete

Das System besteht aus einem **Client**- und **Server**-Teil, die miteinander mittels HTTP Requests kommunizieren. (Abbildung 2.1)

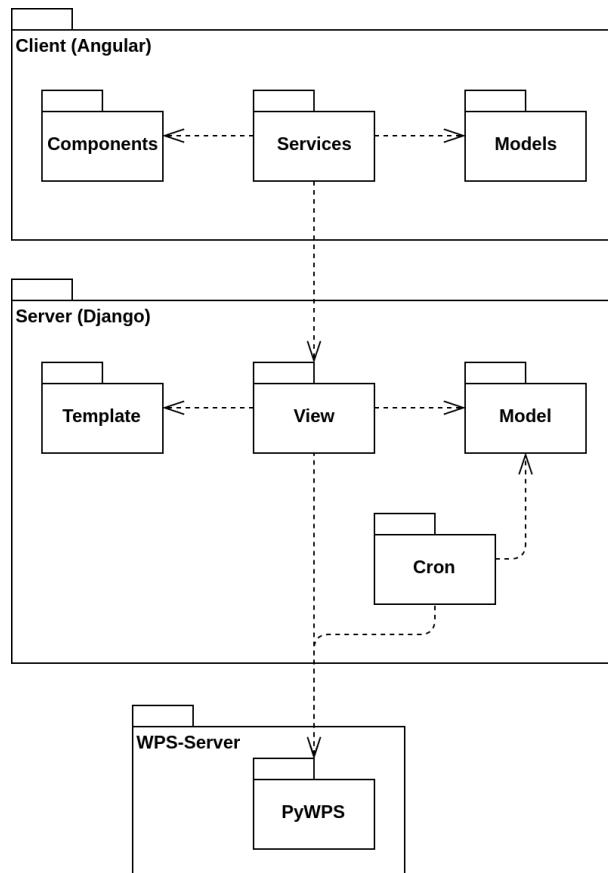


Abbildung 2.1: Paketdiagramm

2.1.1 Client

Der Client, der im Browser des Nutzers läuft, ist auf dem JavaScript-Framework Angular aufgebaut. Es wurde aus folgenden Gründen ausgewählt:

- Idee: Angular ist ein Framework das für Anwendungen verwendet wird die meist direkt im Browser laufen, was bei WPSflow genau der Fall ist
- Angular bietet für WPSflow wichtige Funktionalitäten an. Beispielsweise einen HTTPClient
- Eine hohe Qualität die durch eine große Community gewährleistet wird
- Es gibt eine Vielzahl von Erweiterungen

2.1. AUFTeilung in die Pakete

- Die interne Benutzung von TypeScript, welche es ermöglicht typisierten Code zu schreiben und dadurch einige Fehler zu vermeiden und die Produktivität der Implementierung zu steigern

Angular-Apps sind nach dem **Model View Controller**-Prinzip (MVC) aufgebaut, wobei man für Views den Begriff Components verwendet und für Controller den Begriff Services. Models repräsentieren die Datenstrukturen. Components sind für das Anzeigen (auch Rendering genannt) zuständig und Services schicken Anfragen an den Server und empfangen die Antworten.

2.1.2 Server

Im dem Backend wird das Python-Framework **Django** verwendet. Es benutzt einen modifizierten MVC-Ansatz, der deshalb andere Bezeichnungen verwendet Model - Template - View, kurz MTV.

- Model - Definiert Datenstrukturen und dient dem Zugriff auf die Datenbank
- View - Analog zum Controller im MVC
 - Stellt REST-Schnittstelle zur Verfügung, die vom Client benutzt wird. Dieser Ansatz ermöglicht eine Wiederverwendbarkeit der API für zukünftige Apps - zum Beispiel Smartphone Apps
 - Bearbeitet Abfragen vom Client und ruft Klassen aus dem Model-Paket auf um Daten zu holen oder zu schreiben
- Template - Analog zu Views im MVC, enthält die HTML-Datei der Seiten
- **Cron** - wird in regelmäßigen Zeitabständen aufgerufen und fragt den Status des momentan ausgeführten Tasks beim WPS-Servern an. Falls ein Task erfolgreich beendet wurde, schickt der Cron ggf. den nächsten Task an den WPS-Server

2.1.3 WPS-Server

Ein oder mehrere Server, die für die eigentlichen Berechnungen zuständig sind, werden extern zur Verfügung gestellt und in das System eingetragen. Der Server kommuniziert mit den externen Servern über das WPS Protokoll.

3 Klassen und Pakete

3.1 Server Klassen

3.1.1 edu.kit.scc.pseworkflow.views

WorkflowView

Beschreibt die Workflow-Schnittstelle für die REST-API.

Attribute:

Name	Datentyp	Beschreibung
model	Model	Model, die von Django automatisch für dieses View geladen wird

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
index	HTTPResponse	request: HTTPRequest	Gibt die HTML-Datei aus der Template-Datei zurück. Diese enthält eine Verlinkung auf Code, der auf den Client geladen und dort ausgeführt wird
get	HTTPResponse	request: HTTPRequest id: int	Sucht in der Datenbank nach dem Workflow mit der übergebenen ID und gibt ihn im JSON-Format zurück
create	HTTPResponse	request: HTTPRequest	Erzeugt einen neuen Workflow und gibt seine ID zurück
update	HTTPResponse	request: HTTPRequest id: int	Speichert die Daten in die Datenbank; Überschreibt die Standard Methode. In dieser Methode wird die Workflow enthaltende JSON Datei geparsst und danach werden einzelne Elemente eines Workflows separat gespeichert.
delete	HTTPResponse	request: HTTPRequest id: int	Löscht den Workflow mit der übergebenen ID
start	HTTPResponse	request: HTTPRequest id: int	Führt den Workflow mit der übergebenen ID aus
stop	HTTPResponse	request: HTTPRequest id: int	Stoppt die Ausführung des Workflows mit der übergebenen ID

3.1. SERVER KLASSEN

EditorView

Beschreibt die Seite in der der Nutzer seine Workflows bearbeiten kann.

Attribute:

Name	Datentyp	Beschreibung
template_name	string	Name der Template-Datei

SettingsView

Beschreibt die Seite in der der Nutzer WPS Server eintragen und löschen kann sowie verfügbare Prozesse davon in das System synchronisieren.

Attribute:

Name	Datentyp	Beschreibung
template_name	string	Name der Template-Datei

WorkflowsView

Beschreibt die Seite in der der Nutzer eine Liste seiner Workflows sieht.

Attribute:

Name	Datentyp	Beschreibung
template_name	string	Name der Template Datei

UserView

Bietet Schnittstelle, durch die der Client abfragen kann, ob der Besucher eingeloggt ist.

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
index	HTTPResponse	request: XMLHttpRequest	Gibt die Accountdaten des eingeloggten Users oder eine Meldung, dass User nicht eingeloggt ist zurück

3.1. SERVER KLASSEN

ProcessView

Bietet Schnittstelle, durch die der Client die Prozesse abfragen und verwalten kann.

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
index	HTTPResponse	request: HTTPRequest	Gibt eine Liste aller verfügbaren Processes zurück
get	HTTPResponse	request: HTTPRequest id: int	Gibt Details des Processes mit übergegebenem ID zurück
create	HTTPResponse	request: HTTPRequest	Erstellt einen neuen Process und gibt seine ID in der Response zurück
update	HTTPResponse	request: HTTPRequest id: int	Speichert die in der Request übergegebenen Daten im Process mit der übergegebenen ID
delete	HTTPResponse	request: HTTPRequest id: int	Löscht den Process mit der übergegebenen ID

3.1. SERVER KLASSEN

WPSView

Bietet Schnittstelle, durch die der Client die WPS Server abfragen und verwalten kann.

f Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
index	HTTPResponse	request: HTTPRequest	Gibt eine Liste aller verfügbarer WPS Server zurück
get	HTTPResponse	request: HTTPRequest id: int	Gibt Details des WPS mit der übergegebenen ID zurück
create	HTTPResponse	request: HTTPRequest	Erstellt einen neuen WPS Server und gibt seine ID in der Response zurück
update	HTTPResponse	request: HTTPRequest id: int	Speichert die in der Request übergegebenen Daten im WPS Server mit der übergegebenen ID
delete	HTTPResponse	request: HTTPRequest id: int	Löscht den WPS Server mit der übergegebenen ID
refresh	HTTPResponse	request: HTTPRequest id: int	Lädt eine Liste aller verfügbarer Processes von dem WPS Server mit der übergegebenen ID und aktualisiert Processes in unserer Datenbank, wenn Änderungen vorliegen. Das erfolgt so: Liste der Prozessen vom Server wird mit dieser aus unserer Datenbank verglichen

3.1.2 edu.kit.scc.pseworkflow.cron

WorkflowJob

Hier wird die externe Bibliothek django-cron verwendet, und diese Klasse überschreibt die Klasse CronJobBase.

Die Klasse erledigt Aufgaben, die ständig im Hintergrund laufen und nicht vom Benutzer gesteuert werden. Methoden in dieser Klasse werden in regelmäßigen Zeitabständen von **Cron** aufgerufen (z.B. einmal pro Minute). Dazu gehören:

- Für jeden Laufenden Task (status = RUNNING) sein Status bei zugehörigem WPS-Server abfragen und, falls er geändert wurde, Änderungen in die Datenbank schreiben
- Alle noch nicht ausgeführte Tasks durchgehen und an den zugehörigen WPS Server schicken, falls er frei ist

An der Stelle lohnt es sich zu erwähnen, welche Bedeutung die Statusen der Tasks für Aufführung der weiteren Tasks haben:

3.1. SERVER KLASSEN

- READY - um an den WPS-Server abgeschickt werden zu dürfen, müssen die Tasks diesen Status haben. Sie kriegen den, falls
 - sie am Anfang des Workflows stehen und keine Tasks vor sich haben, oder
 - wenn vorheriger Task auf FINISHED gesetzt wird. Wenn es mehrere vorherige Tasks gibt, passiert es dann, wenn letzte von allen zu FINISHED wird
- WAITING - der Task wartet auf Ergebnisse von vorherigen Tasks
- RUNNING - der Task wird jetzt ausgeführt auf einem WPS-Server
- FINISHED - der Task wurde erfolgreich auf einem WPS-Server beendet. Ggf. nächste Tasks können vom Status WAITING auf READY gesetzt werden
- FAILED - Alle Tasks, die Output von FAILED Task als Input bekommen, werden nicht ausgeführt
- DEPRECATED - Wenn der Workflow mindestens einen Task mit dem Status DEPRECATED enthält, wird seine Ausführung gestoppt, d. h. keine weitere Tasks werden an den WPS-Server geschickt

Attribute:

Name	Datentyp	Beschreibung
schedule	Schedule	Klasse von django-cron Bibliothek, beschreibt, wann der Job ausgeführt werden muss
code	string	Code dieses Cron-Jobs

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
do	void	-	Überschreibt die Metode in der Parentklasse. Hier passiert alles, was in der Klassenbeschreibung steht

3.1. SERVER KLASSEN

3.1.3 edu.kit.scc.pseworkflow.models

Workflow

Diese Klasse beschreibt einen **Workflow**. Ein Workflow enthält einen oder mehrere **Tasks**. Diese Klasse erbt von der Django Klasse namens „Model“.

Attribute:

Name	Datentyp	Beschreibung
id	int	Primärschlüssel in der Datenbank. Wird von Django automatisch generiert
name	string	Anzeigetitel und Name des Workflows
description	string	Beschreibungstext des Workflows
percent_done	int	Fortschritt des Workflows in Prozent
created_at	Date	Erzeugungsdatum des Workflows
updated_at	Date	Datum der letzten Veränderung
creator	User	Benutzer der den Workflow erstellt hat
last_modifier	User	Benutzer der zuletzt den Workflow verändert hat

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
save	void	keine	Speichert die Daten in die Datenbank; Überschreibt die Standard Methode. In dieser Methode wird die Workflow enthaltende JSON Datei geparsst und danach werden einzelne Elemente eines Workflows separat gespeichert.

Session

Diese Klasse speichert die letzten Workflow den ein Nutzer bearbeitet hat.

Attribute:

Name	Datentyp	Beschreibung
id	int	Die ID der Session
user	User	Der zugehörige User
last_workflow	Workflow	Der Workflow der in der Session erstellt wurde

3.1. SERVER KLASSEN

Task

Beschreibung eines einzelnen Tasks innerhalb eines Workflows, die in der Benutzeroberfläche als Knoten des Graphes dargestellt sind. Task ist eine Instanz eines WPS Prozesses, der von dem WPS Server zur Verfügung gestellt wird. Es kann mehrere Tasks geben, die später als derselbe WPS Prozess ausgeführt werden.

Diese Klasse erbt von der Django Klasse namens „Model“.

Attribute:

Name	Datentyp	Beschreibung
id	int	Primärschlüssel in der Datenbank. Wird von Django automatisch generiert
workflow	Workflow	Workflow, dessen Tasks die Kante verbindet
process	Process	Der zugehörende WPS Prozess, der von dem WPS Server zur Verfügung gestellt wird.
x	int	X Koordinate des Tasks in der Benutzeroberfläche. Damit werden Tasks immer genau an der Stelle im Editor angezeigt, wo sie gespeichert wurden
y	int	Y Koordinate des Tasks in der Benutzeroberfläche
status	Status	Ausführungsstatus des Tasks. Siehe workflow.Status Klasse
title	string	Titel des Tasks
abstract	string	Kurze Beschreibung des Tasks
status_url	string	Url des Tasks auf dem WPS Server. Dient der regelmäßigen Aktualisierung des Ausführungsstatus
started_at	Date	Datum und die Uhrzeit des Ausführungstarts

3.1. SERVER KLASSEN

Status (Enum)

Beschreibt die möglichen Ausführungsstati von Tasks.

Name	Beschreibung
READY	Der Task ist bereit ausgeführt zu werden
WAITING	In diesem Zustand wartet ein Task auf den Input eines noch nicht beendeten Tasks
RUNNING	Der Task ist momentan in Bearbeitung
FINISHED	Der Task wurde erfolgreich ausgeführt
FAILED	Bei der Ausführung des Tasks ist ein Fehler aufgetreten
DEPRECATED	Der Task gehört zum Process, der von dem WPS-Server gelöscht wurde, und sollte deswegen aus dem Workflow rausgenommen bzw. ersetzt werden

Edge

Diese Klasse beschreibt eine Kante, die in der Benutzeroberfläche zwei Taskknoten miteinander verbindet.

newline Diese Klasse erbt von der Django Klasse namens „Model“.

Attribute:

Name	Rückgabetyp	Beschreibung
id	int	Primärschlüssel in der Datenbank. Wird von Django automatisch generiert
workflow	Workflow	Workflow, dessen Tasks die Kante verbindet
from_task	Task	Ausgangsknoten
to_task	Task	Eingangsknoten
input	Input	Die Eingangsdaten eines Tasks
output	Output	Das Ergebnis eines Tasks

3.1. SERVER KLASSEN

Process

Diese Klasse beschreibt den WPS Prozess, der auf dem WPS Server als Code gespeichert ist. Alle Daten werden von dem WPS Server als XML Datei mittels HTTP Response (Describe Processes) empfangen und danach auf dem Django Server erstmal geparsert und danach in der Datenbank gespeichert.

Diese Klasse erbt von der Django Klasse namens „Model“.

Attribute:

Name	Rückgabetyp	Beschreibung
id	int	Primärschlüssel in der Datenbank. Wird von Django automatisch generiert
identifier	String	Eindeutige Kennung des WPS Prozesses
wps	WPS	Beschreibt den WPS Server, der den WPS Prozess zur Verfügung stellt
title	string	Titel des WPS Prozesses
abstract	string	Beschreibung des WPS Prozesses

WPS

Diese Klasse beschreibt den WPS Server, der die WPS Prozesse zur Verfügung stellt.

Diese Klasse erbt von der Django Klasse namens „Model“.

Attribute:

Name	Rückgabetyp	Beschreibung
id	int	Primärschlüssel in der Datenbank. Wird von Django automatisch generiert
wps_provider	WPSProvider	Der Besitzer des WPS Servers
title	string	Name des WPS Servers
abstract	string	Beschreibung des WPS Servers
capabilities_url	string	URL des WPS Servers für die GetCapabilities Operation
describe_url	string	URL des WPS Servers für die DescribeProcess Operation
execute_url	string	URL des WPS Servers für die Execute Operation

3.1. SERVER KLASSEN

WPSPublisher

Beschreibt den Besitzer des WPS Servers. Obwohl man mehrere optionale Attribute auf dem WPS Server eingeben kann, werden nur die wichtigsten davon in der Datenbank gespeichert.

Diese Klasse erbt von der Django Klasse namens „Model“.

Attribute:

Name	Rückgabetyp	Beschreibung
id	int	Primärschlüssel in der Datenbank. Wird von Django automatisch generiert
provider_name	string	Name des WPS Server Providers
provider_site	string	Internetadresse des WPS Providers
individual_name	string	Name der Person
position_name	string	Stelle der Person

«abstract» InputOutput

Beschreibt die Inputs und Outputs eines WPS Prozesses. Die Klasse ist abstrakt, dass heißt, dass es keine Tabelle namens InputOutput von Django erzeugt wird.

Diese Klasse erbt von der Django Klasse namens „Model“.

Attribute:

Name	Rückgabetyp	Beschreibung
id	int	Primärschlüssel in der Datenbank. Wird von Django automatisch generiert
process	Process	WPS Prozess, zu dem die Inputs bzw. Outputs gehören
role	Role	Stellt fest, ob die Instanz ein Input oder Output ist
identifier	string	Eindeutige Kennung des Inputs oder Outputs
title	string	Titel des Inputs oder Outputs
abstract	string	Kurze Beschreibung des Inputs oder Outputs
datatype	Datatype	Bestimmt den Datentyp des Inputs oder Outputs. Siehe
min_occurs	int	Minimales Vorkommen eines Inputs oder Outputs
max_occurs	int	Maximales Vorkommen eines Inputs oder Outputs

3.1. SERVER KLASSEN

Input

Beschreibt den Input (Eingangsdaten) des WPS Prozesses.

Diese Klasse erbt die Attribute von der abstrakten Klasse InputOutput (siehe oben).

Ouput

Beschreibt den Output (Endergebnis) des WPS Prozesses.

Diese Klasse erbt die Attribute von der abstrakten Klasse InputOutput (siehe oben).

Datatype (Enum)

Beschreibt die möglichen Datentypen des WPS Prozesses.

Attribute:

Name	Beschreibung
LITERAL	Literal Data ist ein String, normalerweise kurz. Es wird verwendet um numerische oder textuelle Parameter zu übergeben.
COMPLEX	Complex Data sind normalerweise die Raster- oder Vektorgrafiken, es können aber auch beliebige dateibasierte Daten sein
BOUNDING_BOX	Koordinatenpaare für 2D oder 3D Räume

Role (Enum)

Beschreibt die möglichen Werte des Attributs role der abstrakten InputOutput Klasse.

Attribute:

Name	Beschreibung
INPUT	Eingabedaten eines WPS Prozesses
OUTPUT	Endergebnis eines WPS Prozesses

3.1. SERVER KLASSEN

«abstract» Artefact

Beschreibt die eigentlichen Eingangsdaten oder die Endergebnisse eines Tasks. Klasse ist abstrakt, das heißt, dass es keine Tabelle in der Datenbank namens „Artefact“ erzeugt wird. Diese Klasse erbt von der Django Klasse namens „Model“.

Attribute:

Name	Rückgabetyp	Beschreibung
id	int	Primärschlüssel in der Datenbank. Wird von Django automatisch generiert
task	Task	Der Task, zu dem die Inputs oder Outputs gehören
parameter_id	int	Identifikator des Parameters
role	Role	Stellt fest, ob die Instanz ein Input oder Output ist
format	string	Format des Inputs oder Outputs
data	string	Der Wert dieses Attributs hängt von dem Datentyp ab. Falls der Datentyp Literal Data ist wird ein String mit dem Ergebnis gespeichert. Falls der Datentyp Complex Data oder Bounding Box Data ist, also falls das Ergebnis zu groß ist, um es in der Datenbank zu speichern, wird in diesem Attribut eine URL gespeichert die zum Ergebnis auf dem WPS Server führt
created_at	datetime	Erstellungsdatum des Inputs/Outputs
updated_at	datetime	Datum und die Uhrzeit der letzten Änderung

InputArtifact

Beschreibt den die Eingangsdaten des Tasks.

Diese Klasse erbt alle Attribute von der abstrakten Klasse Artefact (siehe oben).

OutputArtifact

Beschreibt das Endergebnis des Tasks.

Diese Klasse erbt alle Attribute von der abstrakten Klasse Artefact (siehe oben).

3.2. CLIENT KLASSEN

3.2 Client Klassen

3.2.1 services

AuthService

Beschreibt den Authentifizierungsservice für Nutzer. Die Klasse ist zuständig für login und logout.

Attribute:

Name	Datentyp	Beschreibung
auth	Observable<User>	Ein Observable mit dem zugehörigen Nutzer

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	HTTPClient	Konstruktor
logout	Promise<boolean>	-	Nutzer-logout
login	Promise<boolean>	email: string password: string	Nutzer-login

WPSService

Beschreibt die Schnittstelle für den Zugriff auf WPS-Services.

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	HTTPClient	Konstruktor
all	Observable<WPS[]>	-	Ein Observable mit allen WPS Services
get	Observable<WPS>	id: number	Gibt ein Observable mit dem WPS Service mit der übergebenen ID zurück
create	Observable<WPS>	process: Process	Erzeugt ein Observable mit einem WPS Service mit dem übergebenen Prozess
remove	Promise<boolean>	id: number	Entfernt den WPS Service mit der übergebenen ID
update	Observable<WPS>	id: number Partial<WPS>	Aktualisiert den WPS Service mit der übergebenen ID mit dem zweiten übergebenen Parameter

3.2. CLIENT KLASSEN

WorkflowService

Beschreibt die Schnittstelle für den Zugriff auf Workflows.

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	HTTPClient	Konstruktor
all	Observable<Workflow[]>	-	Ein Observable mit allen Workflows
get	Observable<Workflow>	id: number	Gibt ein Observable mit dem Workflow mit der übergebenen ID zurück
create	Observable<Workflow>	workflow: Workflow	Erzeugt ein Observable mit dem übergebenen Workflow
remove	Promise<boolean>	id: number	Entfernt den Workflow mit der übergebenen ID
update	Observable<Workflow>	id: number Partial<Workflow>	Aktualisiert das Observable mit dem Workflow mit der übergebenen ID mit dem zweiten übergebenen Parameter
validate	boolean	workflow: Workflow	Überprüft die Syntax des übergebenen Workflows und gibt zurück ob er korrekt ist oder nicht
start	void	id: number	Startet den Workflow mit der übergebenen ID
stop	void	id: number	Stoppt den Workflow mit der übergebenen ID

3.2. CLIENT KLASSEN

ProcessService

Beschreibt die Schnittstelle für den Zugriff auf Prozesse.

Attribute:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	HTTPClient	Konstruktor
all	Observable<Process[]>	-	Gibt ein Observable mit allen Prozessen zurück
get	Observable<Process>	id: number	Gibt ein Observable mit dem Prozess mit der übergebenen ID zurück
create	Observable<Process>	process: Process	Erzeugt ein Observable mit dem übergebenen Prozess
remove	Promise<boolean>	id: number	Entfernt den Prozess mit der übergebenen ID
update	Observable<Process>	id: number Partial<Process>	Aktualisiert das Observable mit dem Prozess mit der übergebenen ID mit dem zweiten übergebenen Parameter

3.2. CLIENT KLASSEN

3.2.2 models

Workflow (Interface)

Dieses Interface beschreibt die Struktur der Workflows.

Attribute:

Name	Datentyp	Beschreibung
edges	WorkflowEdge[]	Die Verbindungen zwischen Tasks im Workflow
tasks	Task[]	Die Tasks im Workflow
creator_id	number	Die ID des Workflow-Erstellers
shared	boolean	Diese Variable speichert ob der Workflow öffentlich oder privat ist
name	string	Der Name des Workflows
id	number	Die ID des Workflows
created_at	number	Das Datum an dem der Workflow erstellt wurde
updated_at	number	Das Datum an dem der Workflow das letzte mal aktualisiert wurde

Edge (Interface)

Dieses Interface beschreibt die Struktur der Kanten.

Attribute:

Name	Datentyp	Beschreibung
id	number	Die ID der Kante
a	Task	Der Task am Anfang der Kante
b	Task	Der Task am Ende der Kante
input_id	number	Die ID des Input-Punktes mit dem die Kante verbunden ist
output_id	number	Die ID des Output-Punktes mit dem die Kante verbunden ist

3.2. CLIENT KLASSEN

Task (Interface)

Dieses Interface beschreibt die Struktur der Tasks die im Workflow die Knoten darstellen.

Attribute:

Name	Datentyp	Beschreibung
id	number	Die ID des Tasks
y	number	Die y-Koordinate des Tasks
x	number	Die x-Koordinate des Tasks
status	TaskState	Der Status des Tasks
input_artefacts	Artifact <'input'>	Die Input-Parameter des Tasks
process_id	number	Die ID des Prozesses der im Task ausgeführt wird
output_artefacts	Artifact <'output'>	Die Output-Parameter des Tasks
created_at	number	Das Datum an dem der Task erstellt wurde
updated_at	number	Das Datum an dem der Task das letzte mal aktualisiert wurde

Artefact<T> (Interface)

Dieses Interface beschreibt die Input/Output Daten. Die Klasse hat den generischen Typ T als role Attribut. Der Typ kann entweder Input oder Output sein.

Attribute:

Name	Datentyp	Beschreibung
parameter_id	number	Die ID des Input- oder Output-Knotens zu der die Artefact-Instanz gehört
task_id	number	Die ID des Tasks zu dem die Artefact-Instanz gehört
workflow_id	number	Die ID des Workflows zu dem die Artefact-Instanz gehört
role	T	Die role gibt an ob die Artefact-Instanz als Input oder als Output verwendet wird
format	string	Gibt das Format der Artefact-Instanz an
data	string	Übergibt die Daten die in der Artefact-Instanz verwendet werden
created_at	number	Das Datum an dem die Artefact-Instanz erstellt wurde
updated_at	number	Das Datum an dem die Artefact-Instanz das letzte mal aktualisiert wurde

3.2. CLIENT KLASSEN

TaskState (Enum)

Beschreibt die möglichen Werte des Attributs status der Task Klasse.

Name	Beschreibung
READY	Der Task ist bereit ausgeführt zu werden
WAITING	In diesem Zustand wartet ein Task auf den Input eines noch nicht beendeten Tasks
RUNNING	Der Task ist momentan in bearbeitung
FINISHED	Der Task wurde erfolgreich ausgeführt
FAILED	Bei der Ausführung des Tasks ist ein Fehler aufgetreten
DEPRECATED	Der Task gehört zum Process, der von dem WPS-Server gelöscht wurde, und sollte deswegen aus dem Workflow rausgenommen bzw. ersetzt werden

Process (Interface)

Dieses Interface beschreibt die Prozesse die jeweils in einem Task enthalten sind.

Attribute:

Name	Datentyp	Beschreibung
id	number	Die ID des Prozesses
title	string	Der Titel des Prozesses
abstract	string	Eine kurze Beschreibung des Prozesses
identifier	string	Die ID des Prozesses die vom WPS-Server festgelegt wird
inputs	ProcessParameter<'input'>[]	Der Typ der Input-Parameter des Prozesses
outputs	ProcessParameter<'output'>[]	Der Typ der Output-Parameter des Prozesses
wps_id	number	Die ID des WPS-Servers der den Prozess bereit stellt
created_timestamp	number	Das Datum an dem der Prozess erstellt wurde
updated_timestamp	number	Das Datum an dem der Prozess das letzte mal aktualisiert wurde

3.2. CLIENT KLASSEN

ProcessParameter<T> (Interface)

Dieses generische Interface mit Typ T beschreibt die jeweiligen Input/Output Parameter des Prozesses. Die Daten werden dabei in einem für WPS gültigen Format angegeben.

Attribute:

Name	Datentyp	Beschreibung
id	number	Die ID des Prozessparameters
role	T	Die role gibt an, ob der Parameter als Input oder als Output verwendet wird
parameter_type	ProcessParameterType	Gibt an von welchem Typ der Parameter ist
title	string	Der Titel des Prozessparameters
abstract	string	Eine kurze Beschreibung des Prozessparameters
min_occurs	number	Gibt an wie oft dieser Prozessparameter mindestens vorkommen muss
max_occurs	number	Gibt an wie oft dieser Prozessparameter maximal vorkommen darf

ProcessParamenterType (Enum)

Diese Aufzählung beschreibt die möglichen Werte die die parameter_type Variable der ProcessParameter<T>-Klasse haben kann.

Name	Beschreibung
COMPLEX	Oft Rasterbilddateien oder Vektorgrafiken
LITERAL	Der Parameter besteht aus Zeichen
BOUNDING_BOX	Sind in OGC OWS Common definiert als zwei Koordinatenpaare

WPS (Interface)

Dieses Interface beschreibt jeweils einen WPS-Server.

Attribute:

Name	Datentyp	Beschreibung
id	number	Die ID des WPS-Servers
provider	WPSProvider	Der Betreiber des WPS-Servers
abstract	string	Eine kurze Beschreibung des WPS-Servers
title	string	Der Titel des WPS-Servers

3.2. CLIENT KLASSEN

WPSProvider (Interface)

Dieses Interface beschreibt jeweils einen WPS-Anbieter zu einem WPS-Server. Es wird im WPS Interface als Attribut provider verwendet.

Attribute:

Name	Datentyp	Beschreibung
id	number	Die ID des Anbieters des WPS-Servers
name	string	Der Name Anbieters des WPS-Servers
url	string	Die URL zum Anbieter des WPS-Servers

User (Interface)

Dieses Interface beschreibt den Aufbau einer Nutzer-Instanz.

Name	Datentyp	Beschreibung
id	number	Die ID des Nutzers
created_at	number	Das Datum an dem die Nutzer-Instanz erstellt wurde
updated_at	number	Das Datum an dem die Nutzer-Instanz das letzte mal aktualisiert wurde
last_workflow_id	number	Die ID des Workflows den der Nutzer zuletzt im Editor geladen hatte
email	string	Die E-Mail Adresse des Nutzers
last_name	string	Der Nachname des Nutzers
first_name	string	Der Vorname des Nutzers
username	string	Der Nutzernname des Nutzers
group	UserGroup	Die UserGroup des Nutzers

UserGroup (Enum)

Diese Aufzählung beschreibt mögliche Werte für die Benutzergruppe einer Nutzer-Instanz.

Name	Beschreibung
REGULAR	Ein normaler Nutzer
ADMIN	Ein Nutzer mit Admin-Rechten

3.2. CLIENT KLASSEN

3.2.3 components

OnInit (Interface)

Dieses Interface wird von den Komponentenklassen implementiert und bietet eine Standardschnittstelle für die Initialisierung.

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
ngOnInit	void	-	Initialisierung der Komponente

AppComponent

Beschreibt die Seite der Anwendung. Dafür wird immer geprüft ob der Nutzer, der auf die Komponente zugreift, auch eingeloggt ist.

Attribute:

Name	Datentyp	Beschreibung
authService	AuthService	Der Authentifikationsservice über den der Nutzer auf die Anwendung zugreift

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	au :AuthService	Konstruktor
ngOnInit	void	-	Initialisierung der Komponente

3.2. CLIENT KLASSEN

EditorPageComponent

Beschreibt die Komponenten der Editor-Seite die nicht durch Workflow-Komponenten dargestellt werden.

Attribute:

Name	Datentyp	Beschreibung
editorComponent	EditorPageComponent	Referenz zur Editor-Komponente
processListComponent	ProcessListComponent	Referenz zur ProcessList-Komponente
showProcessList	boolean	Ob die Prozess Liste angezeigt wird
snapshots	Workflow[]	Eine Liste geänderten Workflows für die UNDO Funktionalität
onStart	EventEmitter<Workflow>	Emittet beim Klick auf den Startknopf
onStop	EventEmitter<Workflow>	Emittet beim Klick auf den Stopknopf
onFinish	EventEmitter<Workflow>	Emittet beim Fertigstellen der Ausführung
onUndo	EventEmitter<[Workflow, Workflow]>	Emittet beim Klick auf den UNDO-Knopf

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	ws: WorkflowService ps: ProcessService	Konstruktor
ngOnInit	void	-	Initialisierung der Komponente
load	void	workflow: Workflow	Lädt den übergebenen Workflow
save	void	-	Speichert den übergebenen Workflow
start	void	-	Startet den übergebenen Workflow
stop	void	-	Stoppt den übergebenen Workflow
undo	void	steps = 1	Macht die angegebene Zahl an letzten Aktionen rückgängig

3.2. CLIENT KLASSEN

WorkflowPageComponent

Beschreibt die Darstellung der Workflows-Seite in der der Nutzer seine Workflows betrachten kann.

Attribute:

Name	Datentyp	Beschreibung
workflows	Observable<Workflow[]>	Aktuelle Workflows
selected	number	Die Anzahl an aktuell markierten Elementen auf der Workflow-Seite

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	ws: WorkflowService	Konstruktor
ngOnInit	void	-	Initialisierung der Komponente
rename	void	id: number name: string	Ändert den Namen des Elements auf der Workflow-Seite mit der übergebenen ID in den übergebenen Namen
delete	void	id: number	Löscht das Workflow Element mit der übergebenen ID
run	void	id: number	Führt den Workflow mit der übergebenen ID aus
edit	void	id: number	Lädt den Workflow mit der übergebenen ID im Workflow-Editor
select	void	id: number	Markiert den Element auf der Workflow-Seite mit der übergebenen ID

3.2. CLIENT KLASSEN

AdminSettingsPage

Beschreibt die Seite auf der der Admin seine Einstellungen ändern kann.

Attribute:

Name	Datentyp	Beschreibung
wps	Observable<WPS[]>	Ein Observable mit den aktuellsten WPS Objekte
selected	number	Die Anzahl an aktuell markierten Elementen auf der Einstellungsseite

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	ws: WPSService	Konstruktor
ngOnInit	void	-	Initialisierung der Komponente
refreshAllWPS	void	-	Lädt alle WPS-Server und deren Services neu
addWPS	void	id: number	Fügt den WPS-Server mit der übergebenen ID hinzu
removeWPS	void	id: number	Entfernt den WPS-Server mit der übergebenen ID

EditorComponent

Beschreibt den Teil des Editors in dem der Workflow bearbeitet wird. EditorComponent kann auch eine Instanz von EditorPageComponent sein.

Attribute:

Name	Datentyp	Beschreibung
workflow	Workflow	Der Workflow der im aktuellen Editor geladen ist
background	ElementRef	Der Hintergrund der aktuellen Editor-Seite
selectedProcessIndex	number	Der in der Liste ausgewählte Prozess

3.2. CLIENT KLASSEN

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	el: ElementRef	Konstruktor
ngOnInit	void	-	Initialisierung der Komponente
getEdgeSVG	string	edge: Edge	SVG String Darstellung der eines Edges
dragOver	boolean	event: DragEvent	Realisiert, dass Prozesse per Drag hinzugefügt werden können
dragStart	boolean	index: number event: MouseEvent	Start des Drag and Drops
dragMove	boolean	event: MouseEvent	Wird durchgehend während des Drag and Drops ausgeführt
dragEnd	boolean	event: MouseEvent	Ende des Drag and Drops

ProcessComponent

Beschreibt jeweils einen Prozess der auf der Editor-Seite als Knoten eines Workflows platziert werden kann und in der ProcessListComponent angezeigt wird.

Attribute:

Name	Datentyp	Beschreibung
process	Process	Der zu repräsentierende Prozess
draggable	boolean	Ob ein Prozess per Drag and Drop verschoben werden kann.

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	-	Konstruktor
ngOnInit	void	-	Initialisierung der Komponente
getParameterColor	string	type: ProcessParameterType	Gibt eine Hintergrundfarbe je nach Input/Output Types zurück
dragStart	boolean	event: DragEvent	Wird beim Starten des Drag and Drops ausgeführt
openDetailDialog	void	-	Öffnen einer Detailansicht des Prozesses

3.2. CLIENT KLASSEN

TaskComponent

Beschreibt jeweils einen Task der auf der Editor-Seite als Knoten eines Workflows platziert wurde.

Attribute:

Name	Datentyp	Beschreibung
task	Task	der zu repräsentierende Task
onClickInput	EventEmitter<ProcessParameter<'input'>>	Emittet beim Klick auf ein Input
onClickOutput	EventEmitter<ProcessParameter<'output'>>	Emittet beim Klick auf ein Output

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	-	Konstruktor
ngOnInit	void	-	Initialisierung der Komponente
getParameterColor	string	type: ProcessParameterType	Gibt eine Hintergrundfarbe je nach Input/Output Types zurück
getStatusColor	string	type: TaskState	Gibt die Hintergrundfarbe für ein gewissen Task Zustand
openDetailDialog	void	-	Öffnet ein Dialogfenster mit einer Detailansicht des Tasks
getInputPosition	[number, number]	id: number	Gibt die X, Y Position der Input Artefakte
getOutputPosition	[number, number]	id: number	Gibt die X, Y Position der Output Artefakte
clickInput	void	-	Wird beim Klick auf ein Output Artefakt ausgeführt
clickOutput	void	-	Wird beim Klick auf ein Input Artefakt ausgeführt

3.2. CLIENT KLASSEN

ProcessListComponent

Beschreibt die Komponente im Editor die die Liste der Prozesse enthält.

Attribute:

Name	Datentyp	Beschreibung
processes	Process[]	Liste aller Prozesse

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	-	Konstruktor
ngOnInit	void	-	Initialisierung der Komponente

ArtefactDialogComponent

Beschreibt den Teil des Editors in dem die Inputs/Outputs stehen.

Attribute:

Name	Datentyp	Beschreibung
task	Task	Ausgewählter Task
role	string	Input oder Output Artefakt
id	number	ID des Artefakts
representation	ArtefactRepresentationStrategy	Die ArtefactRepresentationStrategy des Komponente

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
constructor	-	role: string id: number	Konstruktor
ngOnInit	void	-	Initialisierung der Komponente

ArtefactRepresentationStrategy (Interface)

Beschreibt eine Schnittstelle zur implementierung der Darstellung verschiedener Elemente.

Attribute:

Name	Datentyp	Beschreibung
format	string	Das zu erkennende Format für das Artefakt
role	string	Input oder Output Artefakt

3.2. CLIENT KLASSEN

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
represent	TemplateRef	-	Gibt eine Repräsentation des Artefaktes zurück

ErrorRepresentation

Beschreibt die Darstellung von Fehlern.

Attribute:

Name	Datentyp	Beschreibung
format	string	Das zu erkennende Format der Repräsentation
role	string	Input oder Output

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
represent	TemplateRef	-	Gibt ein Template für die Fehler-Repräsentation zurück

JSONRepresentation

Beschreibt die Darstellung von Daten im JSON-Format.

Attribute:

Name	Datentyp	Beschreibung
format	string	Das zu erkennende Format für das Artefakt
role	string	Input oder Output Artefakt

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
represent	TemplateRef	-	Gibt ein Template für die JSON-Repräsentation zurück

XMLRepresentation

Beschreibt die Darstellung von Daten im XML-Format.

Attribute:

Name	Datentyp	Beschreibung
format	string	Das zu erkennende Format für das Artefakt
role	string	Input oder Output Artefakt

3.2. CLIENT KLASSEN

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
represent	TemplateRef	-	Gibt ein Template für die XML-Repräsentation zurück

ImageRepresentation

Beschreibt die Darstellung von Daten im Bild-Format.

Attribute:

Name	Datentyp	Beschreibung
format	string	Das zu erkennende Format für das Artefakt
role	string	Input oder Output Artefakt

Methoden:

Name	Rückgabetyp	Parameter	Beschreibung
represent	TemplateRef	-	Gibt ein Template für die Bild-Repräsentation zurück

4 Sequenzdiagramme

Um die verschiedenen möglichen Abläufe zu verdeutlichen wurden einige Sequenzdiagramme erstellt, die die Funktionen der Anwendung einfach und effektiv erklären.

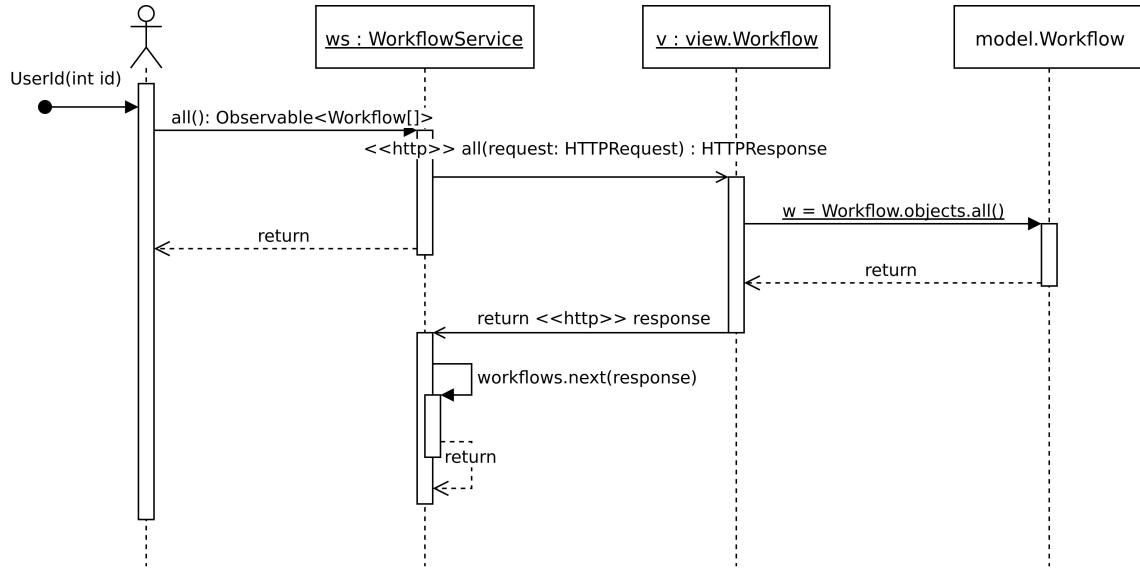


Abbildung 4.1: Get All Workflows

- Der Benutzer ist mit seiner eindeutigen User ID angemeldet und will alle seine Workflows sehen (Abbildung 4.1). Hierfür wird clientseitig eine Liste an Observables in einer Instanz von WorkflowService angelegt. Dann wird ein HTTP Request an die zugehörige serverseitige Instanz von view.Workflow geschickt. Diese frägt bei der Datenbank in model.Workflow die Daten an und schickt sie per HTTP Response zurück an die vorherige Instanz von WorkflowService. Hier wird anschließend die Referenz auf die Observables aktualisiert, wodurch der Benutzer jetzt alle seine Workflows angezeigt bekommt.

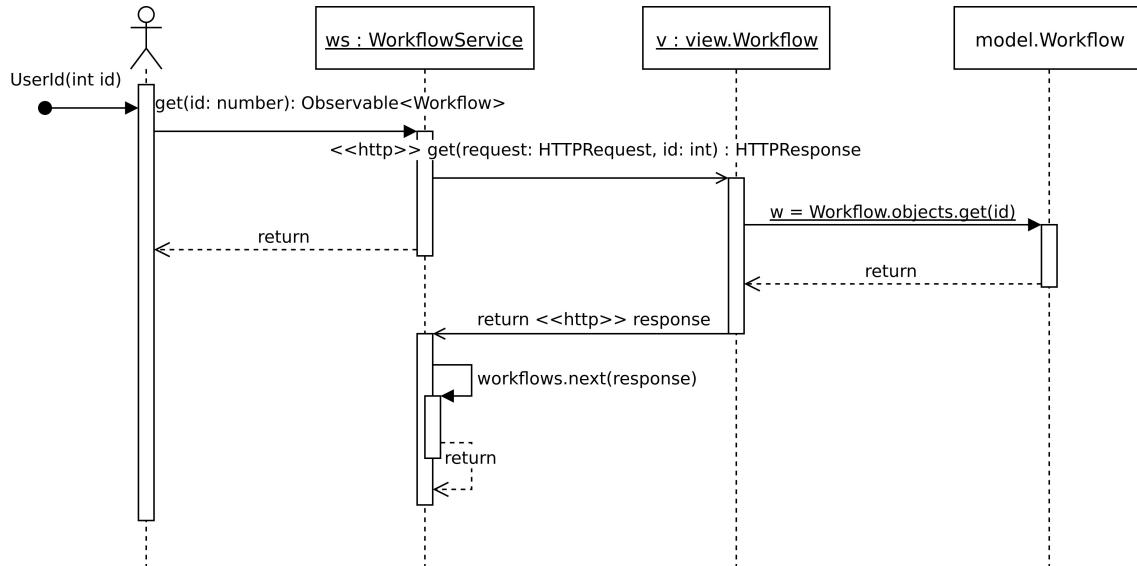


Abbildung 4.2: Get Workflow

- Bei Abbildung 4.2 verläuft das analog wie bei Abbildung 4.1, mit der Ausnahme, dass hier entsprechend immer nur ein Workflow anhand seiner ID geladen wird. Hier wird u.a. ebenfalls der aktuelle Ausführungsstatus des Workflows geladen.

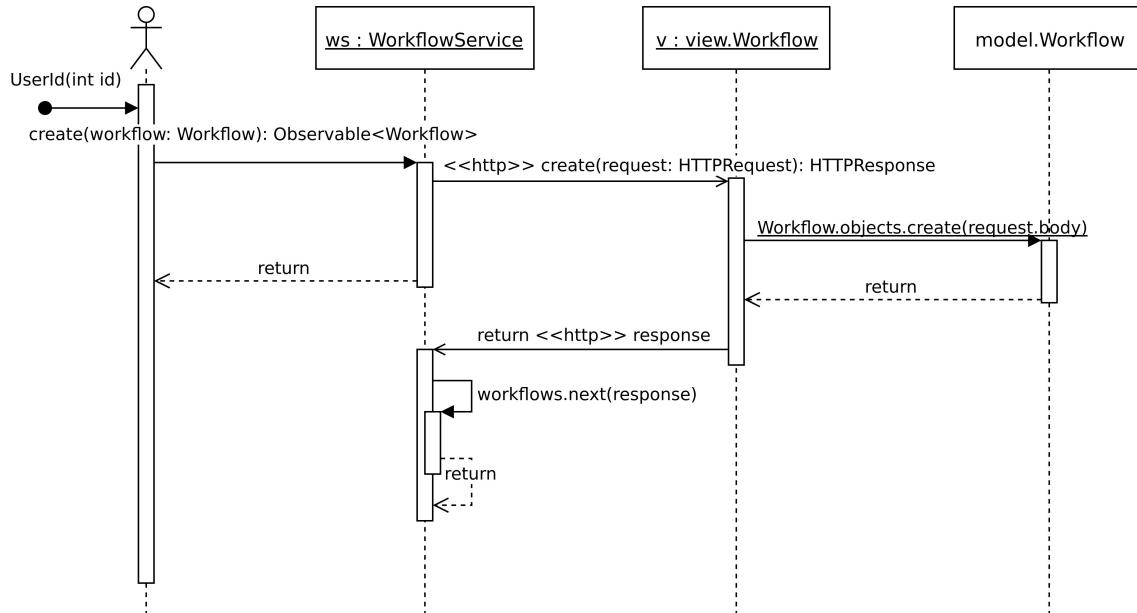


Abbildung 4.3: Save Workflow / Create new Workflow

- Der Benutzer ist bereits mit seiner eindeutigen User ID eingeloggt und hat einen Workflow geladen. Der Benutzer will den Workflow als neuen Workflow speichern (Abbildung 4.3). Er drückt auf "speichern", hier wird er aus dem Editor auf WorkflowService

mittels der `create(workflow)` Methode geleitet. Diese sendet einen HTTP Request an `view.Workflow`. Hier wird dann serverseitig ein neuer Workflow in der Datenbank angelegt. Danach gibt es einen HTTP Response zurück zum `WorkflowService`. Dies geht analog um einen neuen leeren Workflow zu kreieren, nur das hierbei nicht ein bereits vorhandener Workflow gespeichert oder gar geladen sein muss.

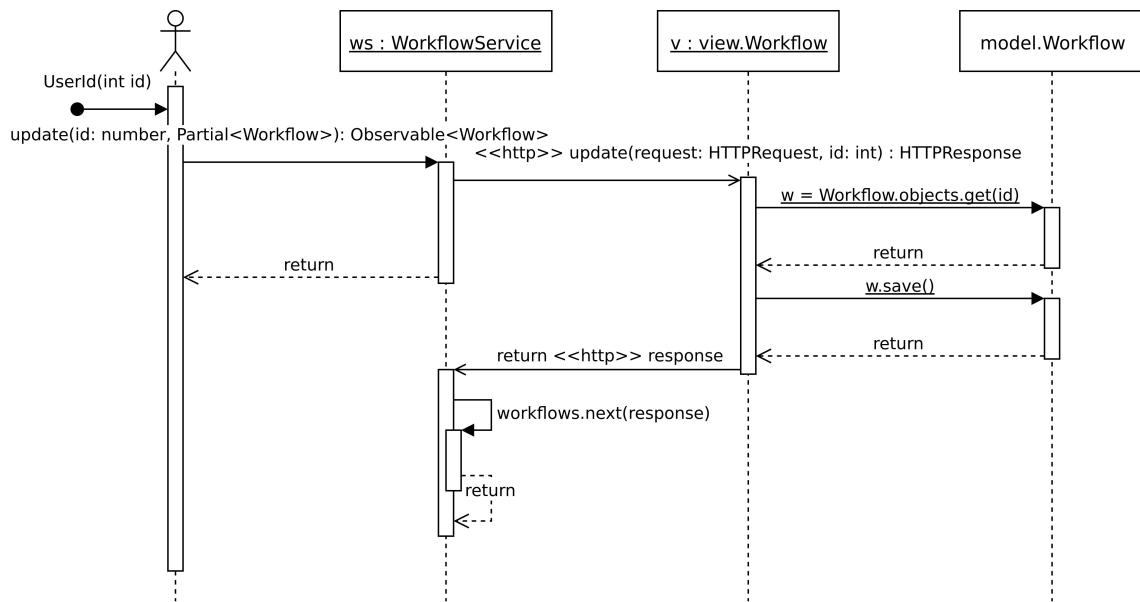


Abbildung 4.4: Edit Workflow

- Der Nutzer ist bereits mit seiner eindeutigen User ID eingeloggt und hat einen Workflow geladen. Er beschließt den geladenen Workflow zu bearbeiten (Abbildung 4.4). Die Änderung wird anschließend über `WorkflowService` wieder als HTTP Request an die Klasse `view.Workflow` geschickt welche dann die Änderung auf die Datenbank (`model.Workflow`) schreibt und anschließend eine HTTP Response zurück an den `WorkflowService` schickt.

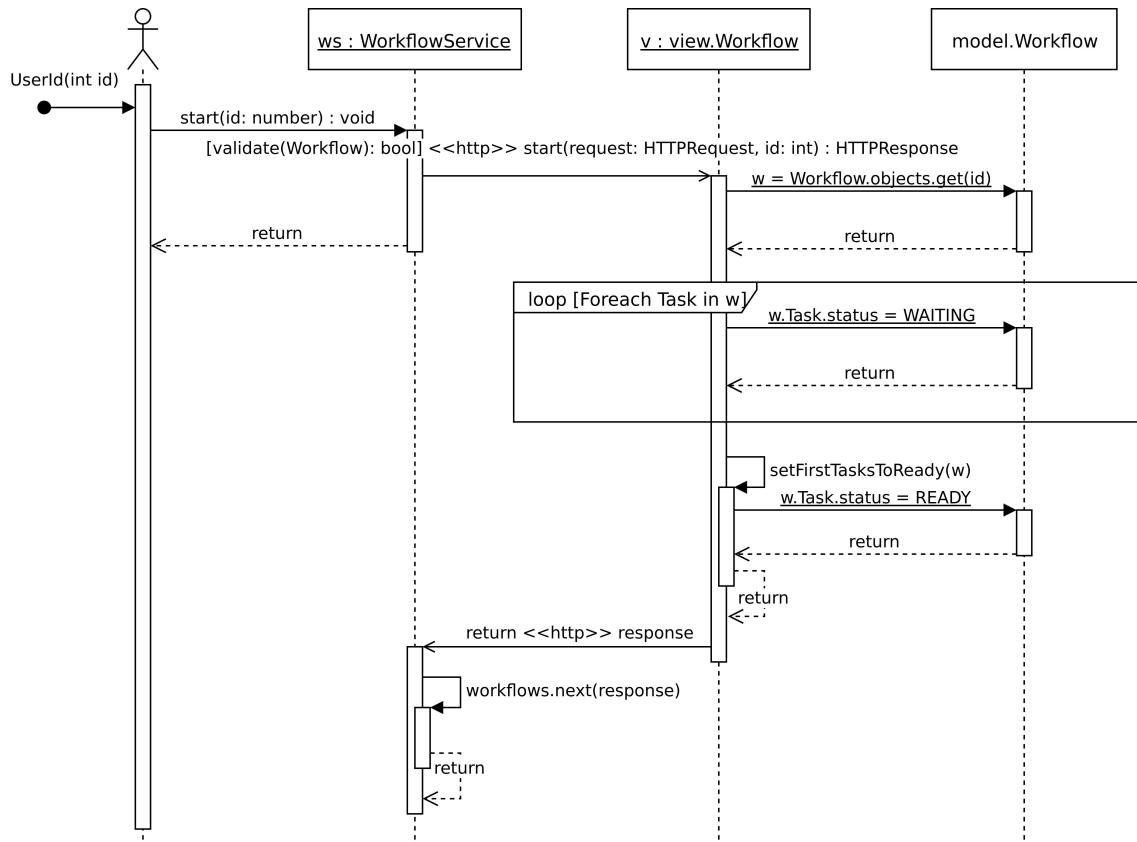


Abbildung 4.5: Execute Workflow Clientside

- Der Nutzer ist auch hier bereits angemeldet und hat einen Workflow geladen. Er beschließt den geladenen Workflow auszuführen (Abbildung 4.5). Hierfür wird an den WorkflowService `start(id: int)` geschickt, welcher mittels HTTP Request an `view.Workflow` weitergeleitet wird, vorausgesetzt, dass die Syntax korrekt ist. Serverseitig (`view.Workflow`) wird dann die aktuelle Version des Workflows aus der Datenbank geladen und der Status aller Tasks des Workflows auf "WAITING" gesetzt. Danach werden die ersten, bzw. der erste Task ermittelt und dieser auf "READY" gesetzt, damit diese(r) dann automatisch von Cron gestartet werden / wird.

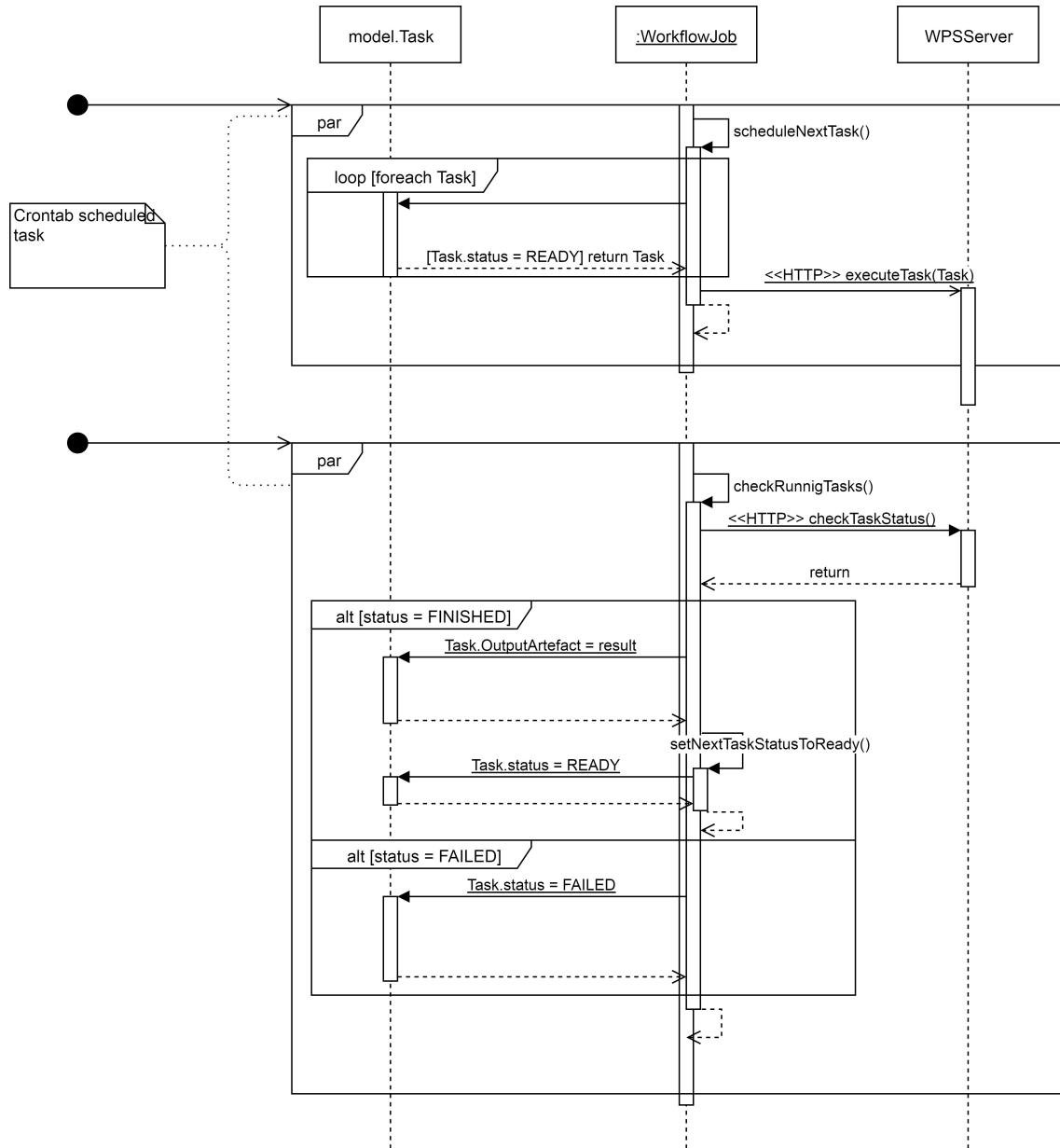


Abbildung 4.6: Execute Workflow Serverside

- Serverseitig werden mittels Crontab in WorkflowJob zwei Jobs parallel ausgeführt (Abbildung 4.6). Der Job `scheduleNextTask()` lädt stets den nächsten Task, welcher "READY" ist von der Datenbank, und führt den Task asynchron auf dem entsprechenden WPS Server aus.

Der Job `checkRunningTasks()` prüft regelmäßig alle Tasks, welche derzeit ausgeführt werden, ob sie bereits fertig sind. Wenn ein Task fertig ist werden die Ergebnisse in die Datenbank geladen und anschließend werden die darauf folgenden Tasks überprüft, ob sie alle Voraussetzungen erfüllen und ausgeführt werden können. Wenn dem so ist, wird deren Status von "WAITING" auf "READY" gesetzt, was dazu führt, dass der Job `scheduleNext()` sie ausführt. Wenn ein Task in einem Workflow fehlschlägt wir

der Status des gesamten Workflows auf “FAILED“ gesetzt, und die weiteren Tasks werden nicht ausgeführt.

5 Datenbank

In diesem Abschnitt wird beschrieben, wie die Daten in einer relationaler Datenbank aussehen werden. Die Umwandlung aus den Django Modellen in die relationalen Tabellen wird von dem ORM (Object Relational Mapping) Modul realisiert, das von Django bereitgestellt wird. Django unterstützt mehrere moderne Datenbanksysteme, was bedeutet, dass die Datenbank austauschbar ist. Im folgenden (Abbildung 5.1) sind die Entitäten und deren Beziehungen dargestellt.

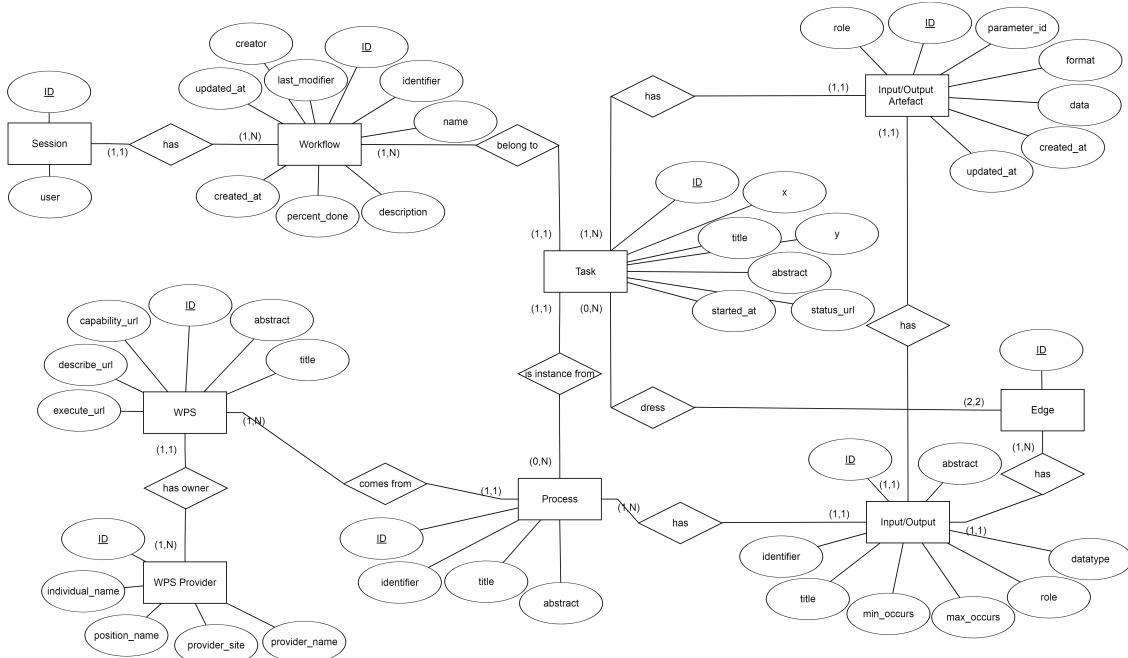


Abbildung 5.1: Entity Relationship Diagram

5.1 Tabellen

Im folgenden finden Sie den Aufbau der Tabellen wie sie in der Datenbank zu finden sein werden.

5.1.1 Session

id	user_id	last_workflow_id

5.1. TABELLEN

5.1.2 WPSProvider

id	provider_name	provider_site	individual_name	position_name

5.1.3 Edge

id	workflow	from_task_id	to_task_id	input_id	output_id

5.1.4 Process

id	wps_id	identifier	title	abstract

5.1.5 WPS

id	service_provider_id	title	capabilities_url	describe_url	execute_url	abstract

5.1.6 Task

id	workflow_id	process_id	x	y	status	title	status_url	started_at	abstract

5.1.7 Workflow

id	name	percent_done	created_at	updated_at	creator_id	last_modifier_id	description

5.1.8 InputOutput

id	process_id	role	identifier	title	datatype	min_occurs	max_occurs	abstract

5.1.9 Artefact

id	task_id	parameter_id	role	format	data	created_at	updated_at

6 URLs

6.1 REST API

Der Server bietet eine **REST** API für die Kommunikation zwischen Client und Server an. Bei REST Anfragen werden dem Client die Daten als JSON geschickt und umgekehrt schickt der Client dem Server JSON Repräsentationen von den speichernden Daten.

Method	URL	Handler	Beschreibung
GET	/user	UserView.index	Rückgabe des eingeloggten Benutzers
GET	/workflow	WorkflowView.index	Rückgabe aller Workflows eines Benutzers
POST	/workflow	WorkflowView.create	Erstellen eines neuen Workflows
GET	/workflow/:id	WorkflowView.get	Rückgabe eines einzelnen Workflows
PATCH	/workflow/:id	WorkflowView.update	Bearbeiten eines Workflows
DELETE	/workflow/:id	WorkflowView.delete	Löschen eines Workflows
GET	/process	ProcessView.index	Rückgabe aller Prozesse
POST	/process	ProcessView.create	Erstellen eines neuen Prozesses
GET	/process/:id	ProcessView.get	Rückgabe eines einzelnen Prozesses
PATCH	/process/:id	ProcessView.update	Bearbeiten eines Prozesses
DELETE	/process/:id	ProcessView.delete	Löschen eines Prozesses
GET	/wps	WPSView.index	Rückgabe aller WPS Services
POST	/wps	WPSView.create	Erstellen eines neuen WPS Services
GET	/wps/:id	WPSView.get	Rückgabe eines einzelnen WPS Services
PATCH	/wps/:id	WPSView.update	Bearbeiten eines WPS Services
DELETE	/wps/:id	WPSView.delete	Löschen eines WPS Services

6.2. URL COMMANDS

6.2 URL Commands

Neben der REST API kann der Server auch bestimmte Commands ausführen. Diese Commands werden ausschließlich per GET Anfragen an den Server geschickt und verarbeitet.

Method	URL	Handler	Beschreibung
GET	/workflow_start/:id	WorkflowView.start	Starten eines Workflows
GET	/workflow_stop/:id	WorkflowView.stop	Stoppen eines Workflows
GET	/wps_refresh	WPSView.refresh	Aktualisieren der gespeicherten WPS Service Daten

6.3 URLs mit HTML Response

Da das Routing vom Client übernommen wird muss der Server bloß sicherstellen, dass die URLs zu den einzelnen Seiten existieren und die index.html liefern.

Method	URL	Handler
GET	/	EditorView.index
GET	/editor	EditorView.index
GET	/workflows	Workflows.index
GET	/settings	Settings.index

7 Entwurfsmuster

WPSflow behilft sich sowohl serverseitig, als auch clientseitig mit diversen Frameworks - hauptsächlich Django für den HTTP Server und Angular für die Web-App. Daher müssen wenige Entwurfsmuster selber geschrieben werden, sondern die vorhandenen Schnittstellen der Frameworks genutzt werden. Dennoch haben sich ein Paar nützliche Entwurfsmuster aus dem Klassendiagramm herauskristallisiert.

7.1 Strategy

Das Format der Artefakte hängt von den Resultaten eines WPS Prozesses ab. Diese Resultate können verschiedene Formen annehmen, wobei wir die Häufigsten abdecken wollen - diese sind XML, JSON und Bilder. Dieses Problem wird mit dem Strategy-Entwurfsmuster (Abbildung 7.1) gelöst. Falls ein Task fehlschlägt, wird dieser durch die Klasse ErrorRepresentation dem Benutzer angezeigt. Nicht behandelte Formate werden je nach Dateigröße als Raw-String angezeigt oder zum Download angeboten.

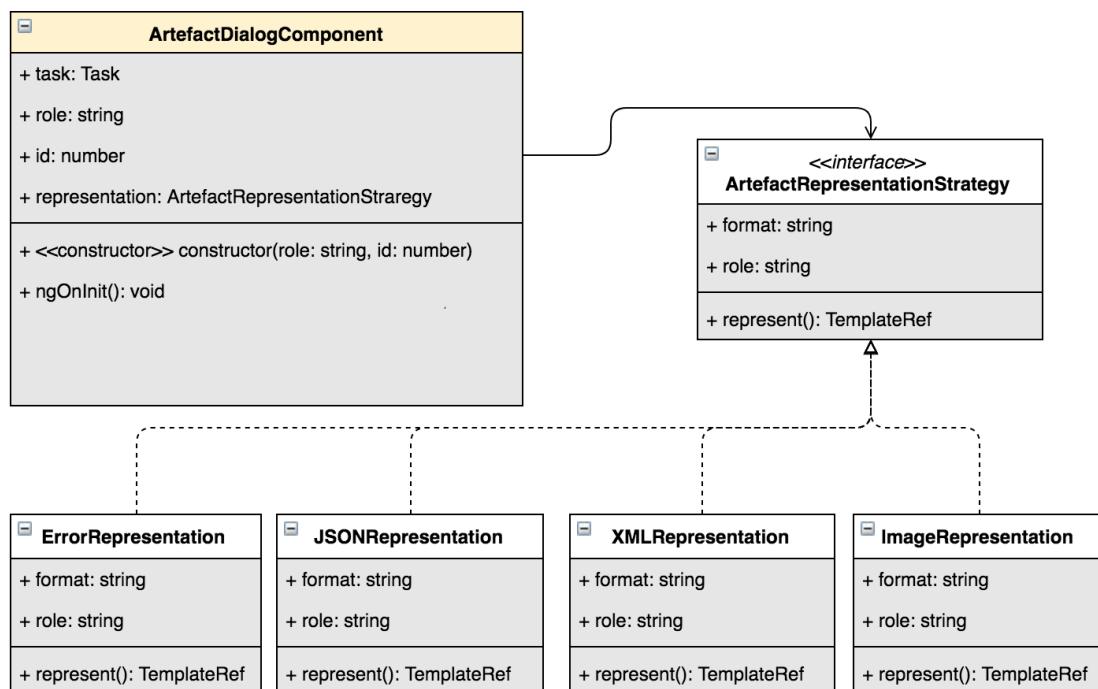


Abbildung 7.1: Verwendung der Strategy-Entwurfsmusters im Frontend

7.2. OBJECT POOL

7.2 Object Pool

Es werden keine HTTP Request von Client-Komponenten geschickt, hierfür sind stattdessen Services (Abbildung 7.2) zuständig. Sie bieten den Komponenten eine Schnittstelle zum Erstellen, Bearbeiten und Löschen von Daten.

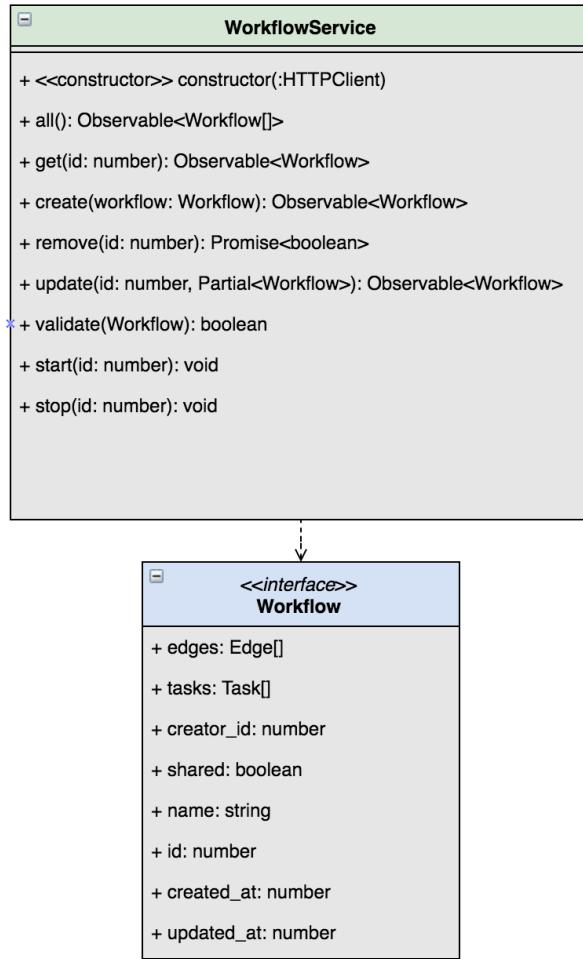


Abbildung 7.2: Beispiel eines Object Pools (hier für Workflows)

7.3 Observer

Der Client hält seine Daten aktuell, indem er in regelmäßigen Abständen Anfragen an den Django Server schickt. Um die Aktualisierung der Daten möglichst angenehm zu halten, werden Observables benutzt. Wir benutzen hierfür eine schon fertige Implementierung von rxjs. Beispiel siehe WorkflowService in Abbildung 7.2

8 Gantt

8.1 Einleitung

Im Folgenden wird die erste Planung der Implementierung umschrieben. Dabei wird lediglich auf die große Struktur eingegangen, genauere Beschreibungen und feinere Aufteilung in kleinere Teilaufgaben erfolgt über Redmine.

8.2 Beschreibung

Die gesammte Implementierung kann in einzelnen Blöcken gesehen werden, die den Ablauf grob beschreiben. Bevor mit der eigentlichen Implementierung begonnen werden kann, müssen Vorbereitungen getroffen werden wie zum Beispiel das generelle Setup der Django Projektstruktur.

Vorbereitung: Generelles Setup des Projekts, Erstellen des Projekts mit Grundbasis und einpflege in das GitHub Repository und weitere Vorbereitungen. (Woche 0)

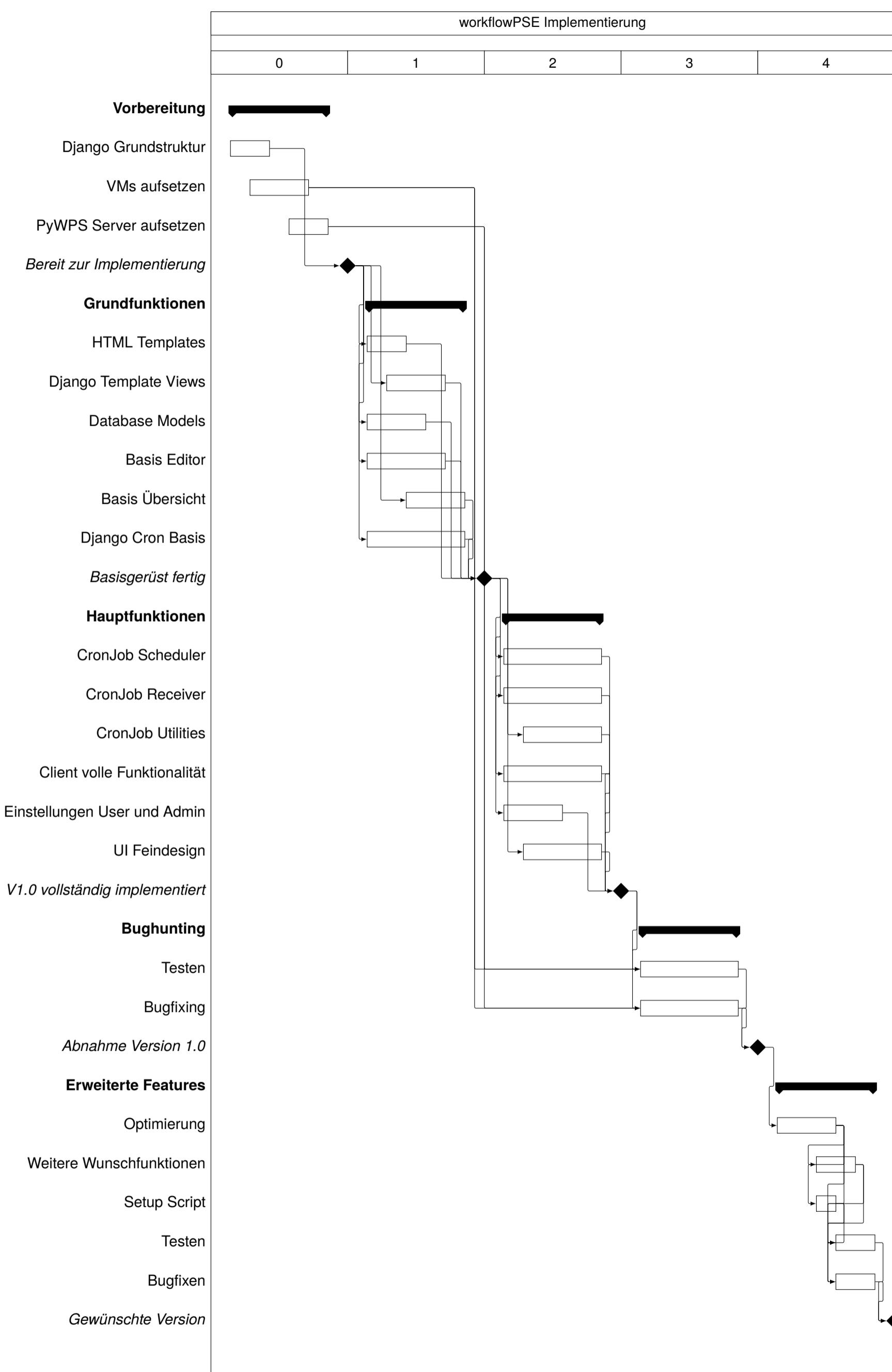
Grundfunktionen: Grundstruktur des Produkts, alle Anlagen und Seiten des Produkts werden erstellt, für alle Funktionen werden leere Stubs gebaut, die Datenbank angelegt und erste Benutzerschnittstellen ohne direktes UI-Layout oder -Design erstellt. (Woche 1)

Hauptfunktionen: Alle vorgesehenen Funktionalitäten sollen in diesem Block fertig implementiert werden, erst jetzt wird ein ansprechendes Front-End aus den bisher rein funktionalen Komponenten gebaut. Am Ende des Blocks soll eine erste Version 1.0 zum Testen bereit sein. Was noch nicht beachtet werden soll sind Bugfixes, es soll zuerst das Grundgerüst stehen, bevor feinere Fehler behoben werden. (Woche 2)

Bughunting: Die erste V1.0 wird auf Herz und Nieren geprüft, Fehler direkt behoben und die Benutzbarkeit und Nutzerfreundlichkeit evaluiert. Kleinere Fehler werden direkt behoben, gröbere Strukturänderungen werden erst im Punkt Optimierung behoben. Derartige Eingriffe sollen aber vermieden werden, um Architektur und Stabilität nicht zu gefährden. Getestet wird erst unter Debug-Einstellungen und zuletzt in einem Produktionsumfeld, Voraussetzung hierfür sind korrekt aufgesetzte Virtuelle Umgebungen und Testserver aus der Vorbereitungsphase. (Woche 3)

Erweiterte Features: Zunächst werden die Ergebnisse der Testphase ausgewertet und das Produkt dementsprechend angepasst und optimiert. Danach werden weitere Wunschfunktionen implementiert ausgiebig getestet und gegebenenfalls angepasst. Ist dies geschehen, kann mit dem letzten Testbericht die Abnahme der gewünschten Produktversion und der Übergang in die Qualitätssicherung erfolgen. (Woche 4)

8.3 Diagramm



Glossar

Client Der Client stellt das Endgerät des Nutzers und die dort ausgeführten Anwendungen dar.

Cron Cron ist ein Programm, das ständig im Hintergrund abläuft und der zeitbasierten Ausführung von Prozessen unter unixartigen Systemen, um wiederkehrende Aufgaben zu automatisieren.

Django Django ist ein auf Python basierendes Open-Source-Webframework das dem **Model View Controller** Schema folgt.

Model View Controller Model View Controller ist ein Architekturmuster, das in der Programmierung für einen flexiblen Entwurf eingesetzt wird um eine Anwendung einfacher an verschiedene Schnittstellen anzupassen. Nur View und Controller müssten dafür dann neu implementiert werden.

REST Ist ein Programmierparadigma für HTTP Server. Hierbei wird eine Standard für die Kommunikation von Ressourcen festgelegt..

Server Der Server stellt die Anwendung für den Client bereit.

Task Ein Schritt in einem Workflow.

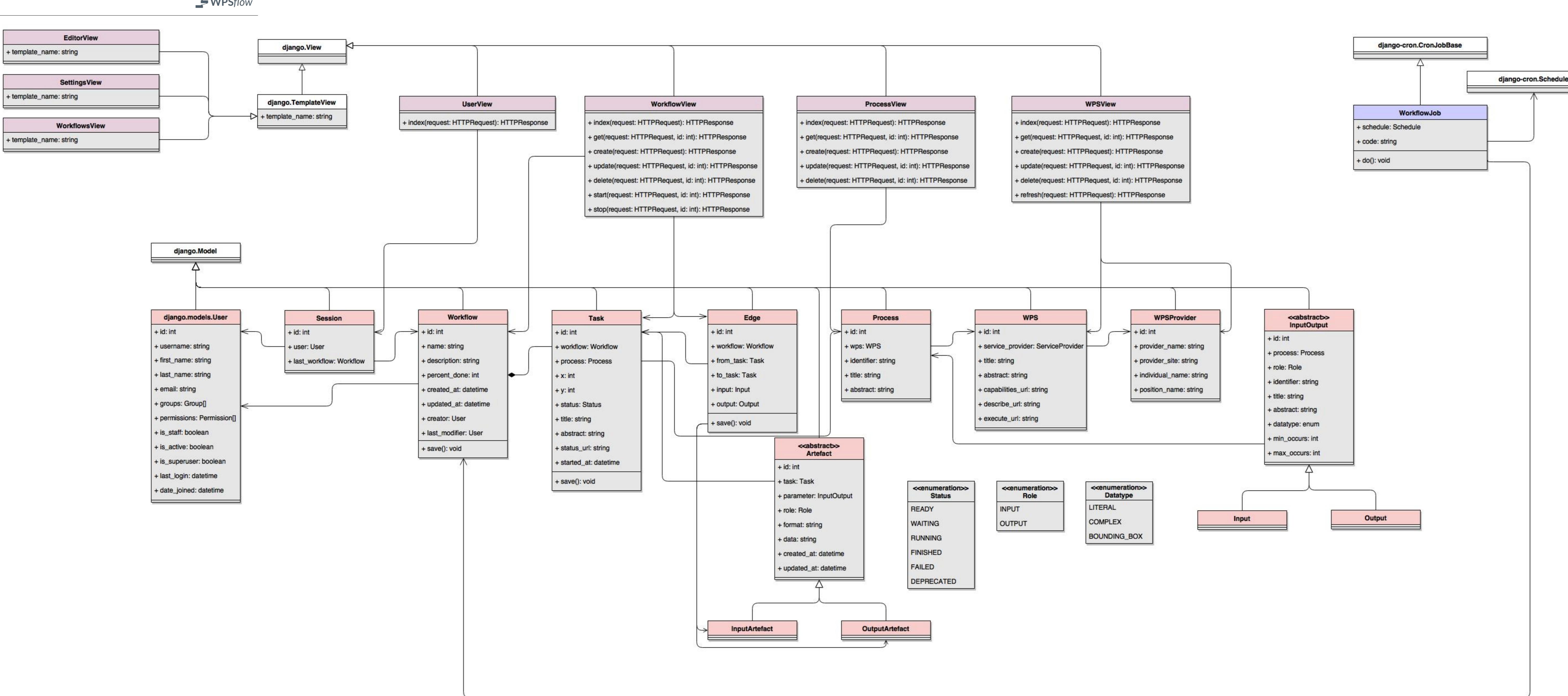
UML Unified Modeling Language.

Workflow Ein strukturierter Arbeitsablauf.

9 Anhang

9.1 Klassendiagramm

Server



WPSflow

Client

