

# **Implementierung MatFlow Workflowanwendung für Machine Learning Experimente**

Florian Küfner, Soeren Raymond, Alessandro Santospirito,  
Lukas Wilhelm, Nils Wolters

Januar 2021

## Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Aufteilung</b>	<b>4</b>
2.1	Geplante Aufteilung . . . . .	4
2.2	Endgültiger Stand nach Implementierung . . . . .	4
<b>3</b>	<b>Entwurfsumentscheidungen</b>	<b>6</b>
3.1	JSON Converter . . . . .	6
3.2	JSON Status Code . . . . .	7
3.3	Authentifizierung . . . . .	7
<b>4</b>	<b>Anhang</b>	<b>8</b>
4.1	Detailliertere Aufteilung . . . . .	8

# 1 Einleitung

Lorem ispum

## 2 Aufteilung

### 2.1 Geplante Aufteilung

Im Folgenden ist eine Grafik zu sehen, die einen Überblick über die Implementierung des MatFlow Systems gibt. Eine detailliertere Ansicht über die Aufgabenteile findet man im Anhang.

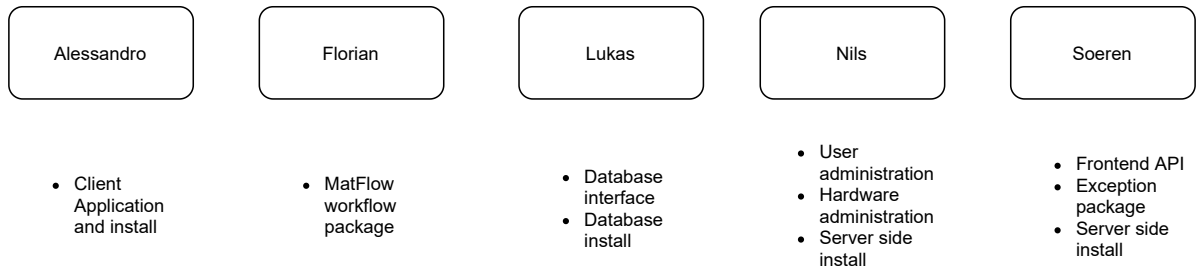


Abbildung 1: Aufteilung

### 2.2 Endgültiger Stand nach Implementierung

Frontend

- **Jede** der anzuwählenden Seiten ist implementiert
- Alle benötigten Datenklassen sind voll funktionsfähig implementiert

Frontend API

- **Jede** der zur Verfügung stehenden Methoden ist bis auf die Authentifizierung voll funktionsfähig implementiert
- Alle Unit Tests sind geschrieben

User Administration

- **Jede** der zur Verfügung stehenden Methoden ist voll funktionsfähig implementiert
- Alle Unit Tests sind geschrieben

Hardware Administration

- **Jede** der zur Verfügung stehenden Methoden ist voll funktionsfähig implementiert
- Alle Unit Tests sind geschrieben

Workflow Package

- **Jede** der zur Verfügung stehenden Methoden ist voll funktionsfähig implementiert
- Alle Unit Tests sind geschrieben

Exception package

- **Jede** der zur Verfügung stehenden Methoden ist voll funktionsfähig implementiert

Database package

- Der Großteil der zur Verfügung stehenden Methoden ist voll funktionsfähig implementiert
- Alle Unit Tests sind geschrieben

Zu implementieren bleibt noch

- Die request Klasse BackendCommunicator und Unit Tests im Frontend
- Die WorkflowData Schnittstelle im Database package
- Der alternierende Operator für airflow
- Die containerweise Ausführung der Serverapplikation im Backend (Nomad Job File)

## 3 Entwurfsumentscheidungen

### 3.1 JSON Converter

#### 3.1.1 Frontend

bla

#### 3.1.2 API

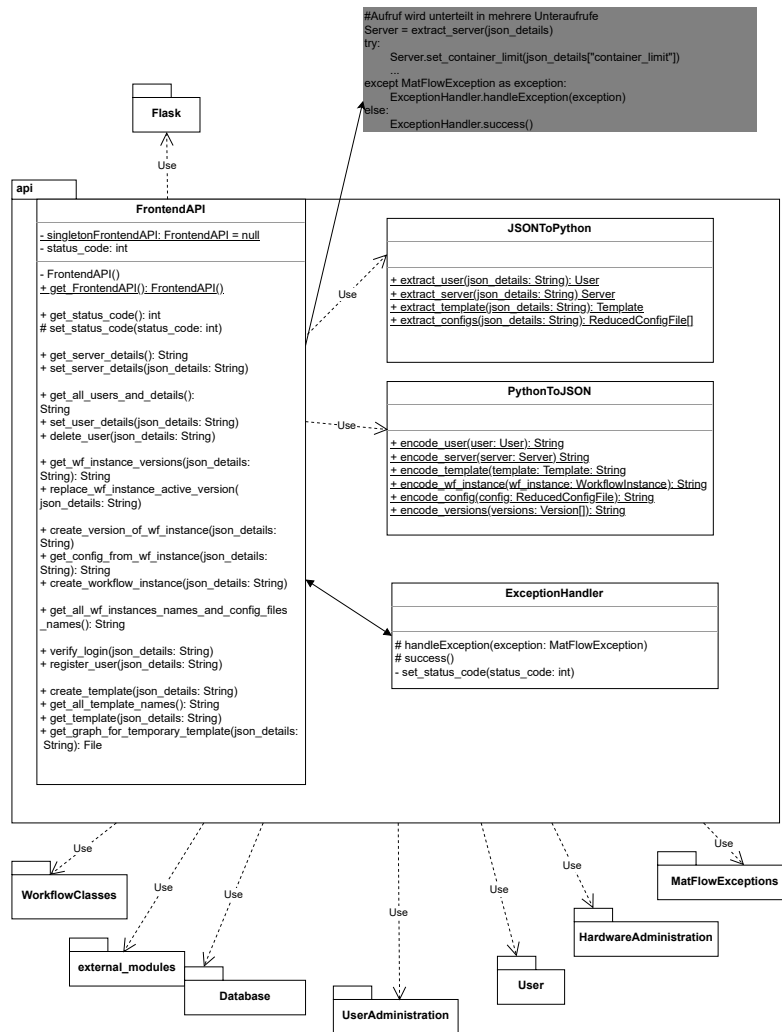


Abbildung 2: API package

Im Backend sind auch analog zum Frontend zwei JSON Converter(jewils von und zu) vorhanden. Diese Entwurfsentscheidung wurde aber verworfen, da sie das Kapselungsprinzip verletzt. Von nun an sind alle in den Converter vorhandenen Methoden direkt als Klassen- (von JSON) oder Objektmethode (zu JSON) direkt in den jeweiligen Klassen implementiert. Als Beispiel: `encode_template(template: Template)` und `extract_template(json_details: String): Template` sind nun als `encode_template(self): str` und classmethod `extract_template: Template` in `Template` vorhanden.

### 3.2 JSON Status Code

Wie man auch in der API sieht, ist die Bereitstellung eines Status Codes bei einem request an die API nicht eindeutig dem Entwurf zu entnehmen, deswegen hier noch einmal eine saubere Erläuterung: Es wird bei einer Anfrage an die API diese Anfrage durchgeführt. Je nachdem ob sie nicht erfolgreich war (`MatFlowException` geworfen, spezieller Status Code vorhanden) oder doch erfolgreich war (success Status Code und eventuell Daten vorhanden) wird in der Klasse `ExceptionHandler` die Antwort in json gebaut und an den Client zurück geschickt. Der Client bekommt also **immer** eine Antwort mit einem json response body, indem sich der status code befindet. Es handelt sich hier also nur um eine Anlehnung an die bereits etablierten HTTP Status Codes.

### 3.3 Authentifizierung

Im Frontend kann man theoretisch unabhängig der User Privilegien mittels URL Manipulation auf alle Seiten zugreifen. Um dies zu verhindern, wird im Frontend ein einzigartiges Cookie mit der Anfrage an die Frontend API geschickt, die dann ??

Man sieht im Folgenden die exportierten Issues aus Gitlab:

Abbildung 3: Aufteilung