Pitshop Documentation

Jannik Schmidt, Simon Jungherz, Frederick Wichert,
Jonas Milkovits, Laurenz Kammeyer

18.03.2022

Einleitung

Dieses Dokument beschreibt die Funktionen und Datenstrukturen der Pitshop-Software. Sein Ziel ist es Entwicklern und Entwicklerinnen ein fundiertes Wissen über die innere Funktion der Pitshop-Software zu vermitteln, damit diese eigenständig und effizient Änderungen am Quellcode vornehmen zu können. Es dient somit für Instandhaltungsarbeiten als auch für die Entwicklung neuer Funktionen. Ein Verständnis von Django wird dabei vorausgesetzt, damit Sachverhalte bündig beschrieben werden können. Soweit möglich wird versucht Komponenten isoliert zu betrachten, damit sich Leser*innen nur mit den Abschnitten befassen müssen an denen sie arbeiten möchten. Für größere Modifikationen wird aber empfohlen das ganze Dokument zu lesen. Technisch weniger versierte Nutzer*innen sollten sich auf das Benutzer*innenhandbuch beziehen.

Überblick

Die Pitshop-Software ist eine webbasierte Abgabeplattform für die Werkstatt des Architekturbereichs der Technischen Universität Darmstadt. Das Backend wird mit Django implementiert. Das Frontend besteht aus HTML, CSS zusammen mit Bootstrap und JavaScript. Als Datenbank wird PostgreSQL verwendet. Die Applikation läuft innerhalb eines Docker-Containers.

Die Pitshop-Software ist nach dem Model-View-Controller (MVC) Paradigma entwickelt. Jede dieser Komponenten wird von einer eigenen "Django-app" umgesetzt.

0.1 View

Alles in Bezug zur Darstellung, befindet sich in der App "client" **/pitshop/client/**. Dort befinden sich alle Django-Templates, sowie alle CSS und JavaScript-Dateien. Um das Design und Verhalten des Webinterfaces anzupassen, ist hier der richtige Ort. Alle Templates befinden sich in einem Subordner des Ordners **/templates**. Jeder Subordner kapselt dabei einen Aspekt der Nutzererfahrung. Die Namen sollten größtenteils selbsterklärend sein, um ein Template jedoch eindeutig mit einer Webansicht zu verknüpfen, kann die Pfadhierarchie, ausgehend von **/pitshop/pitshop/urls.py**, nachvollzogen werden. Die zugehörigen CSS und JS Dateien befinden sich dabei im Ordner **/client** in einem zum Subordner, der so benannt ist wie der des zugehörigen Templates.

0.2 Model

Die Struktur der Datenbank befindet sich im Ordner **/pitshop/models/models.py**. Hier werden alle Datenbank-Tabellen und -Spalten definiert. Die meisten Klassen haben außerdem zusätzliche Property Funktionen, um ihre Funktionalität zu erweitern und Nutzbarkeit zu verbessern. Alles in Bezug zum Aufbau der Datenbank kann hier geändert werden. Die Datenbank selber befindet sich jedoch im Basisordner **/pitshop/**.

0.3 Controller

Die Backendlogik befindet sich in der App **/pitshop/api**. Hier werden alle Funktionen zur Datenbank-Abfrage und -Änderung implementiert. Auch die Kontrolllogik befindet sich hier. Die einzelnen Backendfunktionen sind in der Datei **/api/views.py** zu finden. Wenn angepasst werden soll wie ein Nutzer mit der Datenbank interagiert, ist hier ein guter Anfang Wie eine Funtkion aufgerufen werden kann, bestimmt die Pfad-Datei **/pitshop/api/urls.py**.

1 View

Wie schon im Überblick beschrieben, befindet sich die View in der App **/pitshop/client/**. Im Ordner **/templates** befinden sich dabei alle Django-Templates. Wir benutzen die Template-

Funktionalität von Django hauptsächlich dafür, um die Pfade von Ressourcen und angeueigte Werte zu verwalten. Aus diesen Templates wird dann die Webseite dynamisch zusammengebaut. CSS und JavaScript Dateien befinden sich dabei in dem **/static** Ordner. Anderweitige Ressourcen, wie Bilder oder Icons, befinden sich außerhalb der Client-App im Ordner **/pitshop/global_static**.

Außerdem werden in den Templates die Navbar und der Footer eingefügt. Diese sollen auf allen Seiten vorhanden sein und werde deshalb in allen Templates eingebunden. Innerhalb der Navbar wir auch eine Util.js Datei importiert, die grundlegende Funktionalität bereitstellt, die auf quasi jeder Seite gebraucht wird. Sie enthällt beispielsweise eine Funtkion, um mit der Pitshop Api zu kommunizieren, die auch automatisch den Cookie ausliest und mitsendet.

Ein Großteil des eigentlichen HTMLs befindet sich dabei garnicht in den Templates, sondern wird, auf eine GET-Request der zugehörigen JS-Datei hin, innerhalb der **/client/views.py** dynamisch generiert und zurückgegeben. Auf diese Weise können wir viele Komplexitäten im JavaScript vermeiden und stellen sicher, dass die angezeigten Daten immer mit den Daten der Datenbank übereinstimmen. Dabei enthalten die Templates die Grundstruktur der Seite, die dann dynamisch erweitert wird, bis alle Daten geladen wurden.
Die Funktionalität für größere Einzelaufgaben ist dabei jeweils in ihre eigenen JS-Dateien ausgelagert. Wo genau diese Grenze gezogen wird, liegt größtenteils im Auge des Entwickelnden, die Dateinamen sollten aber eindeutig aussagen welcher Aspekt von ihr implementiert wird. Die Datei, die nach der Seite selber benannt ist, behandelt dabei die grundlegende Verwaltung der Seite, wie Animationen, Dropdowns und das Aufrufen der anderen Funktionen.
Da die meisten Methoden in **/views.py** nur die Aufgabe haben die Dictionary-Werte für den Renderaufruf ihrer Templates zu erzeugen, werden diese hier übergangen. Der Großteil ihrer Komplexität liegt in den Templates, die auch die gesamte Struktur ihrer Rückgabe enthalten. Die wenigen Ausnahmen jedoch sollen hier kurz angesprochen werden.

1.1 get-file()

Vertrag

Parameter: HttpRequest: request, suborder_type: str, id: int
Return: Union[HttpResponse, FileResponse]

Gibt eine FileResponse zurück, die die Datei mit der Bestellung mit der ID 'id' kapselt.

1.2 controlling_filter()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

idInput: Zeige nur Bestellungen mit der ID 'idInput'.

nameInput: Zeige nur Bestellungen deren Kundennamen 'nameInput' enthällt.

dateOrderedFromInput: Zeige nur Bestellungen, die am 'dateOrderedFromInput' oder später bestellt wurden.

dateOrderedUntilInput: Zeige nur Bestellungen an, die am 'dateOrderedUntilInput' oder früher bestellt wurden.

billedInput $\in$ **["not-billed", "billed", "all"]:** Filtern nach dem Abgabestatus der Bestellungen.
"not-billed" $\Rightarrow$ `state` = `OrderState`.FINISHED
"billed" $\Rightarrow$ `state` = `OrderState`.BILLED
"all" $\Rightarrow$ `state` = `OrderState`.FINISHED oder `state` = `OrderState`.BILLED

amountFromInput: Zeige nur Bestellungen, deren Kosten höher als 'amountFromInput' sind.

amountUntilInput: Zeige nur Bestellungen, deren Kosten niedriger als'amountFromInput' sind.

Switches:

sort: Wenn gesetzt, sortiert die Bestellungen nach IDs.

amount: Wenn gesetzt, sortiert die Bestellungen nach Preis.

Diese Methode setzt die Filterfunktionalität für die Abrechnungsseite um. Je nach den gesetzten Switches wird die Liste der Bestellungen zusätzlich nach den gesetzten Filterkriterien sortiert.

1.3 filtered_orders()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

status: Iterable. Zeige nur Bestellungen deren Status teil der Liste ist. Darf nur Werte aus der Klasse 'OrderState' in **/pitshop/model/models.py** enthalten.

dateFrom: Zeige nur Bestellungen, die am 'dateFrom' oder später bestellt wurden.

dateUntil: Zeige nur Bestellungen an, die am 'dateUntil' oder früher bestellt wurden.

id: Zeige nur Bestellungen mit der ID 'id'.

name: Zeige nur Bestellungen deren Kundennamen 'name' enthällt.

sort: Switch. Wenn gesetzt, sortiert die Bestellungen nach IDs.

Return: HttpResponse

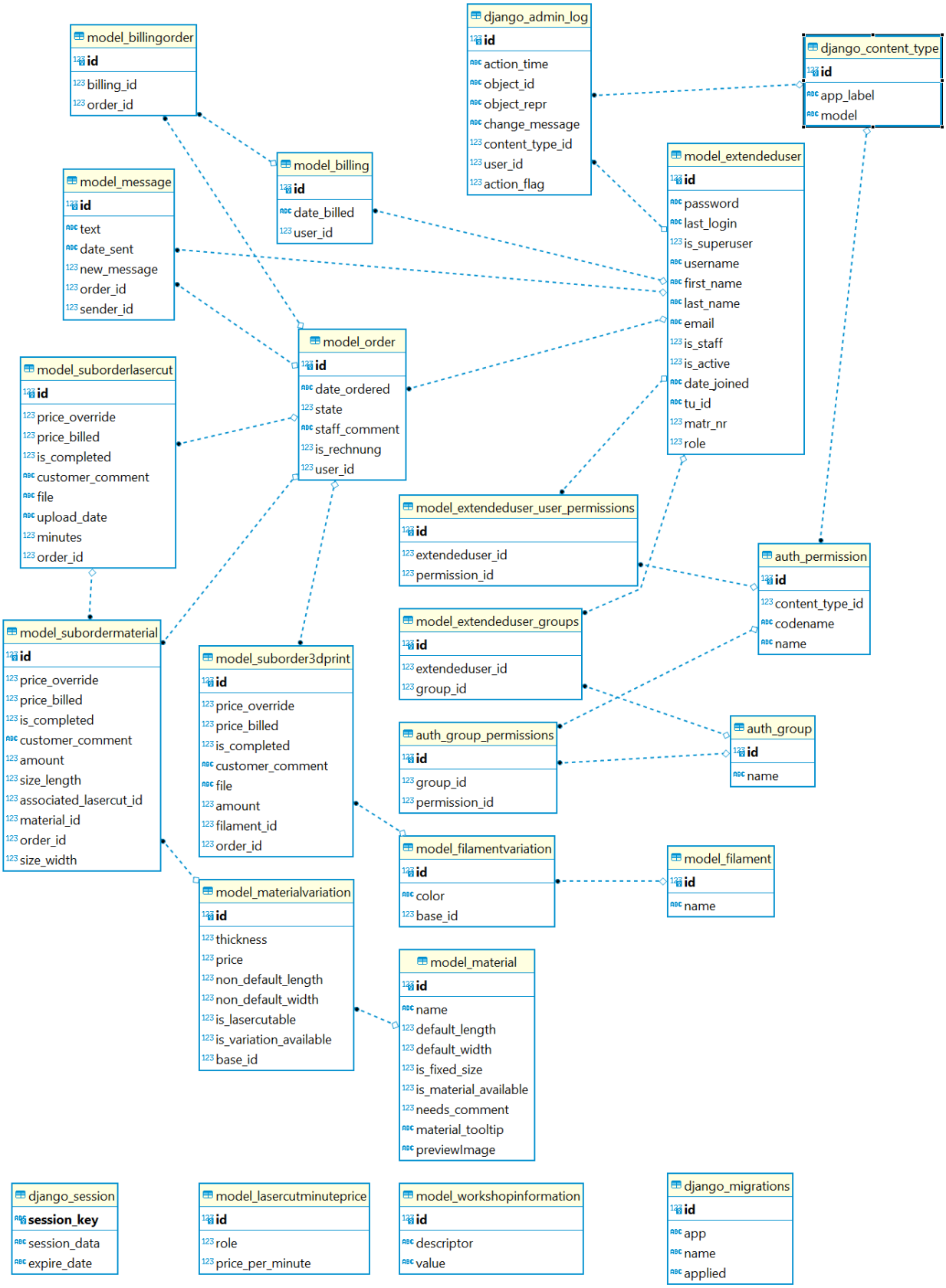
Diese Methode setzt die Filterfunktionalität für die Bestellungsübersicht um. Je nach den gesetzten Switches wird die Liste der Bestellungen zusätzlich nach den IDs sortiert.

2 Model

Die Datenbank-Tabellen und -Spalten werden in der Datei **/pitshop/models/models.py** definiert. Die Struktur der Datenbank beschreibt dabei Nutzer*innen (`ExtendedUser`), denen Bestellungen (`Order`) zugeordnet sind. Diese Stukur wird weiter unterteilt, bis alle Aspekte einer Bestellung abgedeckt sind. So unterteilt sich eine Bestellung weiter in Unterbestellungen usw. Ein `ExtendedUser` erbt dabei von der abstrakten Django-Klasse `AbstractUser`. Eine `Order` ist mit einem `ExtendedUser` über ihr `user_id` Attribut verknüpft. Dabei kann ein `ExtendedUser` maximal eine aufgegebene, aber beliebig viele abgeschlossene `Orders` haben. Eine Bestellung im Warenkorb ist ebenfalls eine `Order`, deren Status aber `None` ist. So kann der Warenkorb mit wenig Aufwand in der Datenbank zwischengespeichert werden, sodass der Nutzer nicht alle Aufträge verliert, sobald er sein Fenster schließt. Alle möglichen Status sind in der Klasse `OrderState` enthalten. Jeder Zustand sagt dabei aus wie weit eine Bestellung schon bearbeitet wurde.
Eine `Order` besteht dabei wieder aus einer Menge assoziierter Lasercut-Suborders (`suborderlasercut`), Material-Suborders(`subordermaterial`) und 3D-Druck-Suborders(`suborder3dprint`). Die Suborders sind dabei über ihre `order_id` Attribute mit der Bestellung verknüpft. Alle Suborder erben außerdem von der abstrakten Klasse `Suborder`, die ein Funktionalität bereitstellt, die alle Suborders erfüllen müssen. Sie enthällt auch ein Feld um die abgerechneten Kosten einer Suborder zu speichern, damit diese nicht von den aktuellen Preisen in der Datenbank abhängen. Die Kinderklassen benötigen so nur noch die für sie spezifischen Attribute.
Die Menge an Materialien, aus der eine Lasercut-Order gefertigt werden soll, wird dabei als

Menge von Material-Subordnern modelliert. Eine Material-Suborder kann dazu zusätzlich mit dem `associated_lasecut_id` mit einer Lasercut-Suborder verbunden werden.
Das Material einer Material-Suborder wird über einen Verweis auf eine Material-Variation (`materialvariation`) definiert. Die Material-Variation Tabelle beschreibt dabei alle Möglichkeiten wie ein konkretes Material von seinen Standardwerten abweichen kann und speichert diese Abweichungen. Die Standardwerte sind wiederum in der Tabelle `material` definiert, zusammen mit Namen, dem zugehörigen Tooltip, einem Vorschaubild und ob ein Kommentar benötigt wird.
Eine 3D-Druck-Suborder ist ähnlich aufgebaut, aber deutlich weniger komplex. Für den 3D-Druck existieren Filamentvariationen (`filamentvariation`), die die möglichen Farben abspeichert in denen ein Filament (`filament`) verfügbar ist.
Eine Bestellung kann auch eine eigene Nachricht umfassen (`message`).
Für Authentifizierung und Sicherheit wird das Django eigene Administratorsystem benutzt, dessen Tabellen automatisch erstellt werden.
Um die Benutzer*innenfreundlichkeit der Klassen zu erhöhen, sind die meisten ausgiebig mit `@property`-Attributen versehen. Diese erfüllen nur sehr einfache und selbsterklärende Aufgaben, die aber das Arbeiten mit der Klasse enorm vereinfachen. Vor dem Schreiben einer eigenen Methode für simple Aufgaben im Bezug zu einer dieser Klassen, ist es wert zuerst einen Blick in die Liste der `@property`-Attribute zu werfen. Die Chancen stehen gut, dass sie schon existiert.

Eine detailierte Übersicht über den Aufbau der Datenbank lässt sich auch dem nachfolgenden ER-Diagramm entnehmen. Es zeigt alle Tabellen, ihre nutzbaren Attribute und wie die Tabellen miteinander in Relation stehen.


model_billingorder
id
billing_id
order_id
model_billing
id
date_billed
user_id
django_admin_log
id
action_time
object_id
object_repr
change_message
content_type_id
user_id
action_flag
django_content_type
id
app_label
model
model_message
id
text
date_sent
new_message
order_id
sender_id
model_extendeduser
id
password
last_login
is_superuser
username
first_name
last_name
email
is_staff
is_active
date_joined
tu_id
matr_nr
role
model_order
id
date_ordered
state
staff_comment
is_rechnung
user_id
model_suborderlasercut
id
price_override
price_billed
is_completed
customer_comment
file
upload_date
minutes
order_id
model_extendeduser_user_permissions
id
extendeduser_id
permission_id
auth_permission
id
content_type_id
codename
name
model_extendeduser_groups
id
extendeduser_id
group_id
model_subordermaterial
id
price_override
price_billed
is_completed
customer_comment
amount
size_length
associated_lasercut_id
material_id
order_id
size_width
model_suborder3dprint
id
price_override
price_billed
is_completed
customer_comment
file
amount
filament_id
order_id
auth_group_permissions
id
group_id
permission_id
auth_group
id
name
model_filamentvariation
id
color
base_id
model_filament
id
name
model_materialvariation
id
thickness
price
non_default_length
non_default_width
is_lasercutable
is_variation_available
base_id
model_material
id
name
default_length
default_width
is_fixed_size
is_material_available
needs_comment
material_tooltip
previewImage
django_session
session_key
session_data
expire_date
model_lasercutminuteprice
id
role
price_per_minute
model_workshopinformation
id
descriptor
value
django_migrations
id
app
name
applied

3 Controller

Die Backend-Logik befindet sich in der Datei **/pitshop/api/views**. Hier werden alle Funktionen zur Verwaltung der Datenbank implementiert. Mit Ausnahme der Django-Administration laufen alle Interaktionen zwischen einer/m Benutzer*in und der Datenbank über eine dieser Methoden. Die Logik ist hier sehr modular aufgebaut. Die Methoden sind jeweils für nur eine Funktionalität verantwortlich, wie z.B. das Hinzufügen oder löschen einer Bestellung. Da es keine übergeordnete Struktur gibt, werden die Methoden hier nur kurz erklärt. Eine Liste aller Zustände in denen sich eine Bestellung befinden kann, finden Sie in der Klasse 'OrderState' in **/pitshop/model/models.py** .Die Operationen können nach Bedarf miteinander kombiniert werden, um das gewünschte Resultat zu erreichen. Zu achten ist dabei nur darauf, dass die Methoden gewisse Restriktionen enthalten, um illegale Operationen zu verhindern. Diese Bedingungen sind:

- Der/Die Benutzer*in muss authentifiziert sein, also eingeloggt.
- Ein*e Nutzer*in kann nur eine aktive Bestellung haben. Es kann allerdings eine weitere, noch nicht aufgegbene, Bestellung mit Status `state` = `None` als "im Warenkorb" existieren.
- Nur Mitarbeiter*innen können eine Bestellung bearbeiten, die von einer anderen Person erstellt wurden.
- Ein*e Nutzer*in kann nur eine Bestellung bearbeiten, die noch nicht angefangen wurde (`state` = `OrderState`.SUBMITTED). Ein*e Mitarbeiter*in kann eine Bestellung so lange bearbeiten, bis sie abgerechnet ist (`state` = `OrderState`.BILLED).
- Eine Änderung an einer Bestellung, muss alle Vorgaben des zugrundeliegenden Mediums erfüllen. Eine Bestellung kann nur in einen Zustand versetzt werden, in dem sie auch hätte bestellt werden können. Minimale Materialgrößen und andere Bedingugen müssen also respektiert werden.
- Nur Mitarbeiter*innen können den Zustand einer Order ändern.

3.1 Methoden

3.2 auth()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

username: Gültiger Benutzername des Benutzers, der sich einloggen möchte

password: Gültiger Passwort des Benutzers, der sich einloggen möchte

Return: JsonResponse({})

Nutzt Djangos eigene Authentifizierungsmethoden, um den Benutzer*in einzuloggen, wenn er/sie korrekte Accountdaten bereitstellt. Returnt eine JsonResponse, der eine leere Antwort enthält.

3.3 unauth()

Vertrag

Parameter: HttpRequest request. Es werden keine extra Parameter benötigt.
Return: JsonResponse({})

Diese Methode loggt den Nutzer aus, der die Anfrage gesendet hat. Er muss vorher eingeloggt gewesen sein.

3.4 get_order()

Vertrag

Parameter:

user: ExtendedUser dessen Bestellung gewünscht ist

order_id: ID der Bestellung des users, die gewünscht ist

Return: Tupel(order, ErrorCode)

Gibt die order des user mit der ID order_id zurück. Falls diese nicht existiert wird ein Fehlercode abhängig von der Rolle des Users zurückgegeben.

3.5 add_suborder()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

type $\in$ { 'lasercut', 'material', '3dprint' } Typ der Teilbestellung, die angelegt werden soll.

comment: String mit dem Kommentar zur Teilbestellung

Für Lasercut: (siehe unten)

Für Material:

amount: Integer [$0 <$ amount < 100]

variation_id: ID der Variation des Materials

width: Integer mit Breite in Milimetern

length: Integer mit Länge in Milimetern

Für 3D-Druck:

amount: Integer [$0 <$ amount < 100]

Parameter, die direkt Element der HttpRequest sind und nicht im Body:

Für Lasercut und 3D-Druck:

FILES: MultiValueDict[str "upload-file", uploadedfile.UploadedFile] mit der Datei der Teilbestellung]

Return: JsonResponse({})

Erzeugt eine neue Teilbestellung mit den spezifizierten Parametern. Wenn die Teilbestellung erfolgreich angelegt wurde, wird eine leere JsonResponse zurückgegeben. Falls der/die Benutzer*in keine Bestellung hat, wird eine erstellt.

3.6 add_lasercut_order()

Vertrag

Parameter: HttpRequest request mit folgenden POST-Parametern:

"lc-variation": Liste mit variation_id der Materialien, die für den Lasercut

"lc-cut-width": Liste mit den Breiten der Materialien in Milimetern

"lc-cut-length": Liste mit den Breiten der Materialien in Milimetern

"lc-count": Liste an Integern mit wie oft welches Material verwendet werden soll

Return: Tuple[Optional[ErrorCode], Optional[SubOrderLaserCut]]

Erzeugt eine neue Lasercut-Bestellung mit den spezifizierten Parametern. Wenn die Bestellung erfolgreich angelegt wurde, wird ein Tupel aus 'None' neuen SubOrderLaserCut zurückgegeben. Andernfalls ein Tupel aus einem ErrorCode 'None').

3.7 add_material_order()

Vertrag

Parameter:

data: dict[str, Any] mit den Parametern der Materialbestellung

order: Order zu der die Materialbestellung hinzugefügt werden soll

Return: Optional[ErrorCode]

Erzeugt eine neue Materialbestellung mit den spezifizierten Parametern und fügt sie der Bestellung 'order' hinzu. Das Dictionary data muss ['amount', 'material', 'comment, 'width', 'length'] enthalten. Diese Werte sind genau so zu belgen wie in `add_suborder()` beschrieben. Wenn die Materialbestellung erfolgreich angelegt wurde, wird "None" zurückgegeben.
Diese Methode ist eine Hilfsmethode von `add_suborder()` und sollte nur in Sonderfällen direkt aufgerufen werden. Für das normale Hinzufügen einer Materialbestellung sollte `add_suborder()` verwendet werden.

3.8 add_3dprint_order()

Vertrag

Parameter:

HttpRequest: HttpRequest mit den Werten der 3D-Druckbestellung als POST-Parameter in Form-data und einem MultiValueDict[str, uploadedfile.UploadedFile] mit der Datei der 3D-Druckbestellung im 'FILES'-Parameter

order: Order zu der die 3D-Druckbestellung hinzugefügt werden soll

Return: Optional[ErrorCode]

Erzeugt eine neue 3D-Druckbestellung mit den spezifizierten Parametern und fügt sie der Bestellung 'order' hinzu. Das Dictionary data muss ['amount', 'comment', 'variation_id'] enthalten. Der Kommentar darf maximal 10.000 Zeichen lang sein und es können nicht mehr als 99 Exemplare bestellt werden. Wenn die 3D-Druckbestellung erfolgreich angelegt wurde, wird "None" zurückgegeben.
Diese Methode ist eine Hilfsmethode von `add_suborder()` und sollte nur in Sonderfällen direkt aufgerufen werden. Für das normale Hinzufügen einer 3D-Druckbestellung sollte `add_suborder()` verwendet werden.

3.9 change_suborder()

Vertrag change_suborder() Teil 1

Parameter: HttpRequest request mit folgenden Body-Parametern:

Pflicht:

type $\in$ { 'lasercut', 'material', '3dprint' } Typ der Teilbestellung, die geändert werden soll.

id: ID der Teilbestellung, die geändert werden soll

Optional:

order_id: ID der Bestellung, deren Teilbestellung geändert werden soll (Wenn nicht angegeben, wird die aktive Bestellung der/des anfragenden User*in verwendet)

newState: Boolean, ob die Teilbestellung auf abgeschlossen gesetzt werden soll

comment: String mit neuem Kommentar zur Teilbestellung

price: Integer mit neuem Preis der Teilbestellung (überschreibt berechneten Preis)

reset_price_override: Boolean. Wenn definiert wird der Preis der Teilbestellung auf den normal berechneten Preis zurückgesetzt (unabhängig vom Wert von reset_price_override)

Für Material:

amount: Integer [$0 <$ amount < 100]

variation_id: ID der Variation des Materials

width: Integer mit Breite in Milimetern

length: Integer mit Länge in Milimetern

Vertrag change_suborder() Teil 2

Für Lasercut:

minutes: Integer mit neuen Minuten der Lasercut-Teilbestellung

Für 3D-Druck:

amount: Integer [$0 < \text{amount} < 100$]

variation_id: Neue Variation des Filaments der 3D-Druckbestellung

Switches:

get_price: Der Preis der Teilbestellung soll Teil der Rückgabe sein ['price']

get_price_str: Der Preistext der Teilbestellung soll Teil der Rückgabe sein ['price_str']

get_order_price: Der Preis der Bestellung soll Teil der Rückgabe sein ['order_price']

get_order_price_str: Der Preistext der Bestellung soll Teil der Rückgabe sein ['order_price_str']

Return: dict[str, Any]

Ändert die Teilbestellung mit der ID 'id' und gibt ein Dictionary mit verschiedenen Preiswerten zurück, wenn die passenden Switches auf 'True' gesetzt sind. Wird keine Bestellung über 'order_id' spezifiziert, wird die aktive Bestellung des/der anfragenden User*in verwendet.
Je nach Typ, können verschiede Werte der Teilbestellung geändert werden. Optionale Werte, die nicht angegeben werden, behalten ihren Wert.

3.10 remove_suborder()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

type: Typ der Teilbestellung, die gelöscht werden soll.

id: ID der Teilbestellung, die gelöscht werden soll

Optional:

order_id: ID der Bestellung, deren Teilbestellung gelöscht werden soll (Wenn nicht angegeben, wird die aktive Bestellung der/des anfragenden User*in verwendet)

Return: JsonResponse({"last_material","last_lasercut","last_3dprint","delete_order"})

Löscht eine Teilbestellung des spezifizierten Typs und der ID 'id' und gibt ein Dictionary zurück, das angibt welcher Typ von Teilbestellung noch in der Bestellung enthalten ist. Wird keine Bestellung über 'order_id' spezifiziert, wird die aktive Bestellung des/der anfragenden User*in verwendet. Wird die letzte Teilbestellung einer Bestellung gelöscht, wird auch die Bestellung gelöscht.

3.11 submit_order()

Vertrag

Parameter: HttpRequest request
Return: JsonResponse({})

Setzt eine vorläufige Bestellung auf 'abgesendet' (`state` = `OrderState`.SUBMITTED). Mit dieser Methode wird die Bestellung im Warenkorb aufgegeben.

3.12 controlling_delete_billing()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

billingIdToDelete: ID der Rechnung, die gelöscht werden soll

Return: JsonResponse({})

Macht das Abrechnen einer Bestellung rückgängig und setzt den Zustand der Bestellung auf 'abgeschlossen' (`state` = `OrderState`.FINISHED). Setzt außerdem das 'price_billed' Attribut auf 'None'.

3.13 controlling_create_new_billing()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

orderIdsToBill: String von IDs von Bestellungen, die abgerechnet werden sollen. Getrennt durch Semikolons.

Return: JsonResponse({})

Erstellt eine Abrechnung mit allen Bestellungen und verknüpft die Rechnung mit den Bestellungen in der BillingOrder Tabelle. Setzt außerdem das 'price_billed' Attribut auf den Preis der Bestellung.

3.14 change_state_of_order()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

orderId: ID der Bestellung, deren Status geändert werden soll

newState: Neuer Zustand der Bestellung

setToInProgressIfSubmitted: Switch. Wenn definiert, wird die Bestellung auf 'in Arbeit' (`state` = `OrderState`.IN_PROGRESS) gesetzt, falls sie vorher 'aufgegeben' (`state` = `OrderState`.SUBMITTED) war

Return: JsonResponse({})

Ändert den Status der Bestellung mit der ID 'orderId' auf den Zustand 'state' und gibt im Erfolgsfall ein leeres JsonObjekt zurück. Wird der Switch 'setToInProgressIfSubmitted' definiert, wird die Bestellung auf 'in Arbeit' (`state` = `OrderState`.IN_PROGRESS) gesetzt, wenn sie 'aufgegeben' (`state` = `OrderState`.SUBMITTED) ist.

3.15 save_staff_comment()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

orderId: ID der Bestellung, deren Mitarbeiter*innenkommentar geändert werden soll

staffComment: String. Neuer Mitarbeiter*innenkommentar

Return: JsonResponse({})

Setzt den Mitarbeiter*innenkommentar der Bestellung mit der ID 'orderId' und gibt im Erfolgsfall ein leeres JsonObjekt zurück.

3.16 save_message()

Vertrag

Parameter: HttpRequest request mit folgenden Body-Parametern:

orderId: ID der Bestellung, zu der eine Nachricht geschrieben wurde

message: String. Neue Nachricht

Return: JsonResponse({})

Hängt eine neue Nachricht an den Nachrichtenverlauf der Bestellung mit der ID 'orderId' an und gibt im Erfolgsfall ein leeres JsonObjekt zurück.

3.17 generate_preview()

Vertrag

Parameter: HttpRequest request mit Parameter MultiValueDict[str "upload-file", uploadedfile.UploadedFile] FILES **Return:** FilerResponse()

Streamed die Datei 'upload-file' für die Browser-Preview.