Wissensgraph-Schema für den Entsorgungsbot
Campus FRA-UAS & Stadt Frankfurt

Version: 4.0 (Final, Core + Professional-Layer)
Projekt: Campus Waste Management Assistant (Graph-RAG)
Zielsystem: Neo4j Property Graph
Autor: ChatGPT (in Zusammenarbeit mit Arvand)

Hinweis: Dieses Dokument ist bewusst ausführlich und enthält erklärende Hinweise (markiert mit 🧠), die du bei Bedarf vor der Abgabe an den Prof entfernen kannst.

1. Einleitung und Motivation

1.1 Problemstellung

Die Entsorgung von Abfällen im universitären Kontext (z. B. FRA-UAS) ist komplex. Sie ist nicht nur die Frage „Welche Tonne?“ sondern hängt von mehreren Faktoren ab:
• dynamischen Zuständen (z. B. flüssig vs. fest, leer vs. gefüllt),
• rechtlichen Klassifizierungen (Abfallsatzung, AVV-Codes, gefährliche Abfälle),
• lokalen Besonderheiten (Campus-Gebäude, Laborbereiche, Sammelstellen),
• und praktischen Handlungen (z. B. „Akku entnehmen“, „Verpackung ausspülen“).

Klassische relationale Datenmodelle oder reine Vektor-Suche (Embedding-only) können diese Wenn-Dann-Logik und die normative Begründung nur unzureichend abbilden. Gerade Fälle wie „Farbeimer – flüssig vs. fest“ oder „elektrische Zahnbürste – Campus vs. Stadt“ erfordern explizite Modellierung von Bedingungen, Regeln und Kontext.

1.2 Lösungsansatz: Neo4j Property Graph als Wissensgraph

Als Lösungsansatz wird ein semantischer Wissensgraph in Form eines Neo4j-Property-Graphen aufgebaut. Der Graph trennt konsequent zwischen:
1. Domänen-Wissen (Begriffe wie Abfallarten, Materialien, rechtliche Klassen),
2. Regel-Logik (Entsorgungsregeln, Bedingungen, Handlungsanweisungen, Rechtsgrundlagen),
3. Infrastruktur (Abfallströme, Container, Gebäude, Räume, externe Einrichtungen).

Dieser Graph dient als strukturierte Wissensbasis für einen Entsorgungsbot, der über ein LLM (z. B. als Graph-RAG-Agent) auf die Daten zugreift, Rückfragen stellt und begründete Antworten liefert.

🧠 Hinweis – kann später gelöscht werden:
Du kannst diesen Abschnitt gut nutzen, um deinem Prof zu erklären, warum ein Property-Graph nötig ist – nämlich wegen Entscheidungslogik + Compliance + Campus-Kontext, die man in einer flachen Vektor-Suche nicht sauber abbilden kann.

2. Datenquellen und Integration

Quelle	Format	Beitrag zum Graphen
Abfall-ABC (FES / Group_3 / Campus-Listen)	CSV	Abfallarten (WasteItem), Synonyme, logische Entsorgungswege, Freitext-Hinweise.
AVV-Katalog (Group_2)	CSV	Offizielle AVV-Codes, Beschreibungen, Information zu gefährlichen Abfällen (Sternchen-Codes).
Abfallsatzung Stadt Frankfurt	PDF (unstrukturiert)	Rechtliche Grundlagen, Definitionen von Abfallarten, Pflichten, Gebühren, Verbote.
Campus-Masterdaten FRA-UAS	CSV / Pläne / manuelle Pflege	Gebäude, Räume/Bereiche, Container-Inventar und ggf. campusinterne Entsorgungsstellen.
Material- und Produktwissen	manuelle Tabellen / Herstellerinfos / LLM-Extraktion	Zuordnung von WasteItems zu Materialien (Material-Knoten und MADE_OF-Beziehungen); Professional-Layer, der schrittweise ausgebaut wird.

🧠 Hinweis – kann später gelöscht werden:
Für eine wissenschaftliche Arbeit kannst du hier noch Literaturverweise ergänzen. Wichtig ist, dass klar wird, woher welche Information im Graph stammt (Data Lineage).

3. Designprinzipien und Schichtenmodell

Das Schema folgt einigen grundlegenden Designprinzipien:

• Schichten-Trennung:
– Begriffs-Ebene (Concept Layer): Was ist der Gegenstand? Welches Material? Welche rechtliche Klasse?
– Regel-Ebene (Rule Layer): Unter welchen Bedingungen gilt welcher Entsorgungsweg? Welche Handlungen sind nötig? Welche Rechtsgrundlage gibt es?
– Infrastruktur-Ebene (Infrastructure Layer): Wo und in welcher Form kann die Entsorgung physisch erfolgen?

• Core- vs. Professional-Layer:
  – Core: Minimal notwendige Knoten und Kanten für einen funktionsfähigen Bot.
  – Professional: Erweiterungen für Materiallogik (Circular Economy), detaillierte Handlungsanweisungen, Campus-Gebäude/Raum-Hierarchie und Hazard-Handling.

• LLM-Freundlichkeit:
  – Wenige, semantisch klare Relationstypen (z. B. HAS_RULE, REQUIRES_CHECK, MANDATES, LEADS_TO, MADE_OF).
  – Kurze, eindeutige Definitionen der Knotentypen, damit ein LLM sie im Prompt gut nutzen kann.

🧠 Hinweis – kann später gelöscht werden:
Du kannst das als Drei-Schichten-Grafik zeichnen (Begriffe – Regeln – Infrastruktur) und daneben Core/Professional markieren.

4. Entitäten (Knoten) im Detail

Im Folgenden werden die wichtigsten Knotentypen des Schemas mit Properties und Beziehungen beschrieben.

4.1 WasteItem – Abfallart / Gegenstand

WasteItem ist das zentrale Einstiegselement im Graphen. Es repräsentiert die Abfallart bzw. den Gegenstand, wie er in CSV-Dateien oder in der Nutzerfrage auftaucht (z. B. „Farbeimer“, „elektrische Zahnbürste“, „Bananenschale“).

Typische Properties:

• uid – technische eindeutige ID (z. B. UUID).

• name – Hauptbezeichnung der Abfallart (Unique Constraint sinnvoll).

• synonyms – Liste von Synonymen (z. B. ["Farbdose", "Lackdose"]).

• description – optionale Kurzbeschreibung.

• source – Herkunft, z. B. „FES_AbfallABC“, „Campus_Internal“.

Wichtige Beziehungen:

• (WasteItem)-[:HAS_RULE]->(DisposalRule).

• (WasteItem)-[:INSTANCE_OF]->(WasteLegalClass).

• (WasteItem)-[:CLASSIFIED_AS]->(AVVCode).

• (WasteItem)-[:HAS_HAZARD]->(HazardType).

• (Professional) (WasteItem)-[:MADE_OF]->(Material).

🧠 Hinweis – kann später gelöscht werden:
Nicht jedes WasteItem hat am Anfang alle Beziehungen. Für einfache Bioabfälle reicht oft HAS_RULE + INSTANCE_OF.

4.2 Material – Werkstoff (Professional-Layer)

Material beschreibt den Werkstoff, aus dem ein WasteItem besteht (z. B. Glas, Polypropylen (PP), PET, Aluminium, Papier, Lithium-Akku). Das ist wichtig für Circular Economy und generische Empfehlungen („Wenn ich nur das Material weiß“).

Typische Properties:

• uid – technische ID.

• name – z. B. „Glas“, „Polypropylen (PP)“, „Papier“, „Lithium-Ionen-Akku“.

• description – optionale Beschreibung.

• recyclability_class – grobe Recycelbarkeitskategorie (optional).

Beziehungen:

• (WasteItem)-[:MADE_OF]->(Material).

• (Material)-[:DEFAULT_DISPOSAL]->(WasteStream) – Standard-Entsorgungsweg für dieses Material.

• (Material)-[:HAS_HAZARD]->(HazardType) – z. B. Batterie-Material → gefährlicher Abfall.

🧠 Hinweis – kann später gelöscht werden:
In der Praxis entstehen Material-Daten aus einer Kombination von Herstellerinfos, vorhandenen Tabellen und LLM-gestützter Extraktion (z. B. aus Produktnamen wie „PP-Becher“, „PET-Flasche“). Dieser Teil ist bewusst als Professional-Layer angelegt und kann iterativ ausgebaut werden.

4.3 AVVCode – Europäischer Abfallschlüssel

AVVCode repräsentiert die offizielle Einstufung nach der Abfallverzeichnis-Verordnung (AVV). Codes mit Sternchen („*“) kennzeichnen gefährliche Abfälle.

Typische Properties:

• code – z. B. „20 01 27*“ (Unique Constraint).

• description – offizielle Beschreibung.

• hazardous – bool (true, wenn Code ein Sternchen enthält oder als gefährlich markiert ist).

• risk_level – integer (z. B. 0–3) als interne Alarmstufe für den Bot.

Beziehungen:

• (WasteItem)-[:CLASSIFIED_AS]->(AVVCode).

• (optional) (AVVCode)-[:INSTANCE_OF]->(WasteLegalClass).

• (optional) (AVVCode)-[:IMPLIES_HAZARD]->(HazardType).

🧠 Hinweis – kann später gelöscht werden:
Ein WasteItem kann in der Praxis mit mehreren AVV-Codes verknüpft sein (z. B. wasserbasierte vs. lösemittelhaltige Farbe). Die Auflösung, welcher Code im Einzelfall zutrifft, erfolgt dann über weitere Bedingungen/Regeln. Typischerweise strebt man 1 „Standard"-AVV-Code pro Item an, das Modell lässt aber 0..n zu.

4.4 WasteLegalClass – Rechtliche Abfallklassen

WasteLegalClass modelliert die Abfallarten, wie sie in der Abfallsatzung definiert werden (z. B. Bioabfall, Altpapier, Sperrmüll, Elektro- und Elektronikgeräte, gefährliche Abfälle).

Typische Properties:

• uid – technische ID.

• name – Klassenname (z. B. „Bioabfall").

• definition – Kurzdefinition aus der Satzung.

Beziehungen (damit WasteLegalClass kein „Waisenkind" ist):

• (WasteItem)-[:INSTANCE_OF]->(WasteLegalClass) – ordnet konkrete Abfallarten ihrer rechtlichen Klasse zu.

• (WasteStream)-[:INSTANCE_OF]->(WasteLegalClass) – ordnet Abfallströme derselben Klasse zu.

🧠 Hinweis – kann später gelöscht werden:
INSTANCE_OF ist hier bewusst gewählt, weil sowohl Item als auch Stream als „Instanzen" der rechtlichen Klasse gesehen werden. Alternativ könnte man für Streams auch ein Label wie BELONGS_TO nutzen – fachlich ist das aber sehr ähnlich.

4.5 HazardType – Gefährdungsarten

HazardType beschreibt Gefährdungseigenschaften (z. B. „gefährlicher Abfall", „entzündlich", „ätzend", „batteriehaltig").

Typische Properties:

• uid – technische ID.

• name – z. B. „gefährlicher Abfall", „entzündliche Flüssigkeit".

• description – optionale Erläuterung.

Beziehungen:

• (WasteItem)-[:HAS_HAZARD]->(HazardType).

• (Material)-[:HAS_HAZARD]->(HazardType).

• (WasteStream)-[:HANDLES_HAZARD]->(HazardType).

4.6 OriginType – Herkunft des Abfalls

OriginType kodiert den Entstehungskontext des Abfalls. Typische Werte sind z. B. „private_haushalt“, „gewerbe“, „hochschule“, „klinik“.

Beziehung:

• (DisposalRule)-[:HAS_ORIGIN]->(OriginType).

4.7 DisposalRule – Entsorgungsregel (Wenn-Dann)

DisposalRule ist der zentrale Regelknoten. Er verknüpft ein WasteItem mit einem WasteStream, ggf. unter Bedingungen und mit Handlungsanweisungen und Rechtsgrundlagen.

Typische Properties:

• uid – technische ID.

• label – Kurzbeschreibung (z. B. „Farbeimer – flüssig/fest – Stadtregel“).

• scope – Geltungsbereich, z. B. „city“, „campus“, „generic“.

• priority – numerische Priorität (Campus-Regeln können Stadt-Regeln übersteuern).

• notes – optionale Freitextnotizen.

Beziehungen:

• (WasteItem)-[:HAS_RULE]->(DisposalRule).

• (DisposalRule)-[:REQUIRES_CHECK]->(Condition).

• (DisposalRule)-[:LEADS_TO]->(WasteStream).

• (DisposalRule)-[:HAS_ORIGIN]->(OriginType).

• (DisposalRule)-[:BASED_ON]->(LegalProvision).

• (DisposalRule {scope:'campus'})-[:OVERRIDES]->(DisposalRule {scope:'city'}).

• (Professional) (DisposalRule)-[:MANDATES]->(Instruction).

4.8 Condition & ConditionValue – Entscheidungslogik

Conditions modellieren Entscheidungsfragen („Ist die Farbe flüssig?“), ConditionValues die Antwortoptionen („flüssig“, „fest“). Um nicht nur flache IF-ELSE-Strukturen, sondern echte Entscheidungsbäume zuzulassen, können Antworten entweder direkt einen Abfallstrom bestimmen oder zu einer weiteren Frage führen.

Condition – typische Properties:

• key – maschinenlesbarer Name (z. B. „aggregatzustand“, „is_electric“).

• questionText – konkrete Frage an den Nutzer.

• type – z. B. „boolean“, „enum“, „range“.

ConditionValue – typische Properties:

• value – z. B. „flüssig“, „fest“, „ja“, „nein“.

• (optional) llmHint – Hilfstext oder Synonyme (z. B. „eingetrocknet“ ≈ „fest“).

Beziehungen:

• (DisposalRule)-[:REQUIRES_CHECK]->(Condition).

• (Condition)-[:HAS_OPTION]->(ConditionValue).

• (ConditionValue)-[:IMPLIES_STREAM]->(WasteStream) – Antwort führt direkt zu einem Abfallstrom (einfacher Fall).

• (optional) (ConditionValue)-[:NEXT_CHECK]->(Condition) – Antwort löst eine weitere Frage aus (mehrstufiger Entscheidungsbaum).

• (optional, alternativ) (ConditionValue)-[:IMPLIES_RULE]->(DisposalRule) – komplexere Pfade mit eigener Regelmenge.

🧠 Hinweis – kann später gelöscht werden:
Für deinen Use Case reicht meist eine Kombination aus IMPLIES_STREAM (letzter Schritt) und NEXT_CHECK (für mehrstufige Bäume):
Frage 1 → Antwort → NEXT_CHECK (Frage 2) → Antwort → IMPLIES_STREAM.

4.9 Instruction – Handlungsanweisungen (Professional-Layer)

Instruction modelliert Handlungen, die zusätzlich zur Entscheidung nötig sind, z. B. „Verpackung ausspülen“, „Deckel trennen“, „Akku entnehmen“, „Nicht in den Ausguss kippen“.

Typische Properties:

• uid – technische ID.

• text – Beschreibung der Anweisung.

• (optional) category – z. B. „trennen“, „reinigen“, „sicherheit“.

Beziehung:

• (DisposalRule)-[:MANDATES]->(Instruction).

Wichtig: Instruction ist keine Bedingung (kein IF/ELSE), sondern ein zusätzlicher Befehl.

4.10 WasteStream – Abfallstrom / Fraktion

WasteStream repräsentiert den logischen Entsorgungsweg (Fraktion), z. B. Restabfall, Bioabfall, Glas, E-Schrott, Sonderabfall, Sperrmüll.

Typische Properties:

• uid – technische ID.

• name – z. B. „Restabfall“, „Bioabfall“, „Glas“, „Sonderabfall (Schadstoffmobil)“.

• collectionMode – z. B. „hol“ (Holsystem), „bring“ (Bringsystem), „service“ (Abholservice).

• scope – z. B. „city“, „campus“.

Beziehungen:

• (DisposalRule)-[:LEADS_TO]->(WasteStream).

• (ConditionValue)-[:IMPLIES_STREAM]->(WasteStream).

• (WasteStream)-[:INSTANCE_OF]->(WasteLegalClass).

• (WasteStream)-[:AVAILABLE_AT]->(Facility).

• (WasteStream)-[:ALLOWED_IN]->(Container).

• (WasteStream)-[:OPERATED_BY]->(Organization).

• (Professional) (WasteStream)-[:ALLOWS_AVV]->(AVVCode) – welche AVV-Codes von diesem Stream rechtlich akzeptiert werden.

🧠 Hinweis – kann später gelöscht werden:
Damit kannst du nach einer Entscheidung prüfen, ob der ausgewählte Stream zum AVV-Code des Items passt. Wenn kein ALLOWS_AVV-Pfad existiert, hast du einen Validierungsfehler im Regelwerk.

4.11 Infrastruktur: Container, Building, Room, Facility (Location-Layer)

Die Infrastruktur-Ebene verortet die Entsorgung in der realen Welt. Wichtig ist dabei eine saubere Hierarchie, damit klar ist, wo ein konkreter Behälter steht.

Container:

• Properties: uid, type (z. B. „Restabfall_120L“, „Biotonne_240L“, „Batteriebox“), capacity_liters (Float, Einheit: Liter), isUnderground (bool), isCampusSpecific (bool).

• Beziehungen:

– (Container)-[:ALLOWS_STREAM]->(WasteStream).

– (Container)-[:LOCATED_AT]->(Room) oder (Container)-[:LOCATED_AT]->(Facility). Container hängen damit immer an einem konkreten „Leaf“-Ort (Raum oder externe Einrichtung/Campus-Stellplatz), nie direkt am Gebäude.

Building (Campus-Gebäude):

• Properties: uid, name (z. B. „Gebäude 1“), shortCode.

• Beziehung: (Building)-[:CONTAINS]->(Room). Building ist ein hierarchischer Container, aber kein konkreter Entsorgungsort.

Room / Area (Räume/Bereiche):

• Properties: uid, name (z. B. „Hof“, „Foyer EG“, „Labor 1.23“).

• Beziehungen: (Building)-[:CONTAINS]->(Room); (Room)-[:CONTAINS]->(Container). Room ist auf dem Campus in der Regel der „Leaf“-Knoten für LOCATED_AT.

Facility (Stadt / externe oder gebäudeunabhängige Einrichtungen):

• Properties: uid, name (z. B. „Wertstoffhof Nord“, „Glascontainer Ecke X/Y“, „Campus-Containerplatz Gebäude 1 Hof“), type, address, openingHours, scope.

• Beziehungen: (Facility)-[:OPERATED_BY]->(Organization); (WasteStream)-[:AVAILABLE_AT]->(Facility). Facility umfasst sowohl klassische städtische Sammelstellen (Wertstoffhof, Schadstoffmobil-Standort) als auch campusinterne Container-Stellplätze, die nicht sinnvoll als Raum innerhalb eines Gebäudes modelliert werden können (z. B. Containerinsel auf dem Parkplatz).

Optionale Superklasse Location:

• In Neo4j können Building, Room und Facility zusätzlich ein gemeinsames Label „Location“ erhalten, um generische Standortsuchen zu vereinfachen. Die Hierarchie wird dann über CONTAINS-Kanten abgebildet, LOCATED_AT verweist immer auf einen „konkreten“ Location-Knoten (Room oder Facility).

4.12 LegalProvision & Organization – Rechtsquellen und Betreiber

LegalProvision repräsentiert Norm-/Paragraphen-Ausschnitte der Abfallsatzung oder anderer Rechtsquellen.

Typische Properties:

• uid – technische ID.

• lawName – z. B. „Abfallsatzung Stadt Frankfurt am Main“.

• paragraph – z. B. „§3 Abs. 2“.

• textExcerpt – relevanter Ausschnitt.

• url – Link zur Online-Fassung (optional).

Beziehung: (DisposalRule)-[:BASED_ON]->(LegalProvision).

Organization beschreibt Betreiber/Verantwortliche (z. B. Stadt Frankfurt, FES, FRA-UAS).

Beziehungen: (Facility)-[:OPERATED_BY]->(Organization); (WasteStream)-[:OPERATED_BY]->(Organization).

5. Beziehungen (Relationen) mit Kardinalitäten

Die Tabelle fasst die wichtigsten Relationstypen mit typischen Kardinalitäten und Semantik zusammen. In Neo4j sind die Kardinalitäten technisch nicht erzwungen, dienen aber der Modell- und Implementationsklarheit.

Beziehungstyp	Von → Nach	Typische Kardinalität	Semantik
HAS_RULE	WasteItem → DisposalRule	1..n pro WasteItem	Mehrere Regeln (z. B. Stadt- und Campus-Regeln).
INSTANCE_OF	WasteItem/WasteStream → WasteLegalClass	1 pro Element	Rechtliche Abfallklasse.
CLASSIFIED_AS	WasteItem → AVVCode	0..n pro WasteItem	Rechtliche Einstufung nach AVV; mehrere Codes möglich (z. B. je nach Zusammensetzung).
MADE_OF	WasteItem → Material	0..n pro WasteItem	Materialkomponenten eines Gegenstands.
HAS_HAZARD	WasteItem/Material → HazardType	0..n	Gefährdungseigenschaften.
HANDLES_HAZARD	WasteStream → HazardType	0..n	Welche Gefährdungsarten der Stream aufnehmen darf.

HAS_ORIGIN	DisposalRule → OriginType	1	Kontext (Haushalt, Campus, etc.).
BASED_ON	DisposalRule → LegalProvision	0..1	Rechtsgrundlage der Regel.
OVERRIDES	DisposalRule(campus) → DisposalRule(city)	0..1	Campus-Regel überschreibt Stadt-Regel.
REQUIRES_CHECK	DisposalRule → Condition	0..n	Löst Nutzer-Rückfragen aus.
HAS_OPTION	Condition → ConditionValue	1..n	Mögliche Antworten auf eine Frage.
NEXT_CHECK	ConditionValue → Condition	0..1	Folgefrage; ermöglicht mehrstufige Entscheidungsbäume.
IMPLIES_STREAM	ConditionValue → WasteStream	0..1	Antwort führt direkt zu einem Abfallstrom (Ende des Pfades).
MANDATES	DisposalRule → Instruction	0..n	Notwendige Handlungen vor der Entsorgung.
LEADS_TO	DisposalRule → WasteStream	0..1	Standard-Entsorgungsweg ohne weitere Bedingungen.
ALLOWS_STREAM	Container → WasteStream	0..n	Welche Streams der Container akzeptiert.
LOCATED_AT	Container → Room/Facility	1	Physische Verortung an einem konkreten Leaf-Standort (Raum oder Facility).
CONTAINS	Building → Room; Room → Container	1..n	Campus-Hierarchie (von Gebäuden zu Räumen zu Containern).
AVAILABLE_AT	WasteStream → Facility	0..n	Wo (Orte) dieser Stream verfügbar ist.
OPERATED_BY	Facility/WasteStream → Organization	1	Betreiber / verantwortliche

			Organisation.
ALLOWS_AVV	WasteStream → AVVCode	0..n	Welche AVV-Codes dieser Stream rechtlich aufnehmen darf (Validierung der Regeln).

6. Daten-Mapping (ETL-Spezifikation)

Die folgende Tabelle beschreibt, wie zentrale Spalten aus den Rohdaten in Graph-Knoten und -Beziehungen überführt werden. Die konkreten Spaltennamen können leicht variieren – wichtig ist die Mapping-Logik.

Quelldatei / Spalte	Ziel im Graphen	Transformationslogik
Group_3 / Abfall-ABC: „Abfallart“	WasteItem.name	Direktes Mapping; zusätzlich uid generieren (UUID).
Group_3: „Synonyme“	WasteItem.synonyms	Kommagetrennten String in Liste umwandeln.
Group_3: „Entsorgungsweg“	WasteStream.name (via DisposalRule.LEADS_TO)	Für einfache Fälle direkter Stream; bei komplexeren Fällen Teil einer Regel.
Group_3: „Hinweise“	Condition / ConditionValue / Instruction	NLP-/LLM-basiertes Parsing: „Wenn/Falls…“ → Conditions (inkl. NEXT_CHECK); Imperative → Instructions.
Group_2: „Abfallbezeichnung“	Matching zu WasteItem.name	String-Match oder fuzzy Matching; dient der Verknüpfung zu AVVCode.
Group_2: „AVV_Schluessel“	AVVCode.code	Direktes Mapping; hazardous aus Sternchen ableiten.
Group_2: „Beschreibung“	AVVCode.description	Direktes Mapping.
Group_2: „gefährlich“/Sternchen	AVVCode.hazardous, AVVCode.risk_level	Bool + Heuristik für Risk-Level (z. B. hazardous=true → risk_level≥1).
Abfallsatzung:	WasteLegalClass.name,	Manuelle/LLM-Extraktion

Abfallarten/Kategorien	WasteLegalClass.definition	aus den relevanten Paragraphen.
Abfallsatzung: Paragraphen	LegalProvision	Auswahl relevanter Paragraphen, Anlegen von LegalProvision-Knoten.
Campus-Daten: Gebäude	Building	Building.uid + Building.name aus Campus-Liste.
Campus-Daten: Räume/Bereiche	Room	Room.uid + Room.name; Building-CONTAINS->Room.
Campus-Daten: Behälter/Fraktionen	Container + ALLOWS_STREAM	Container pro Standort/Fraktion anlegen, ALLOWS_STREAM auf passende WasteStreams setzen.
Material-Quellen (Herstellerinfos, Produktlisten, manuelle Tabellen)	Material + (WasteItem)-[:MADE_OF]->(Material)	Materialknoten anlegen (z. B. „PP“, „PET“, „Glas“); Zuordnung der WasteItems über Domänenwissen oder LLM-gestützte Regeln (z. B. wenn Name „PP-Becher“ enthält).

🧠 Hinweis – kann später gelöscht werden:
Hier ist jetzt explizit beschrieben, dass Material-Daten nicht aus dem Nichts kommen, sondern aus Herstellerinfos, manuellen Tabellen oder LLM-Extraktion. Das beantwortet die typische Prof-Frage „Woher wissen Sie das?“.

7. Beispielhafte Reasoning-Flows (End-to-End)

7.1 Bananenschale (einfacher Bioabfall, Campus)

1. Der Nutzer fragt: „Wohin mit einer Bananenschale auf dem Campus?“
2. LLM/NER erkennt WasteItem "Bananenschale".
3. WasteItem "Bananenschale" INSTANCE_OF→WasteLegalClass „Bioabfall“.
4. WasteItem HAS_RULE→DisposalRule_Banana_campus.
5. DisposalRule_Banana_campus hat keine REQUIRES_CHECK-Beziehungen → keine Rückfragen nötig.
6. DisposalRule_Banana_campus LEADS_TO→WasteStream „Bioabfall (Campus)“.
7. WasteStream „Bioabfall (Campus)“ ALLOWED_IN→Container „Biotonne_240L“.
8. Container „Biotonne_240L“ LOCATED_AT→Room „Hof Gebäude 1“.

Bot-Antwort (vereinfacht):
„Bananenschalen gehören in die Biotonne. Die nächste Biotonne findest du im Hof von Gebäude 1.“

7.2 Farbeimer (mehrstufige Entscheidung + AVV-Validierung)

1. Der Nutzer fragt: „Wie entsorge ich einen alten Farbeimer?“
2. WasteItem "Farbeimer" wird erkannt.
3. WasteItem CLASSIFIED_AS→AVVCode „20 01 27*“. Falls weitere Codes möglich wären (z. B. für andere Zusammensetzungen), wären diese ebenfalls verknüpft.
4. WasteItem HAS_RULE→DisposalRule_Farbeimer_city.
5. DisposalRule_Farbeimer_city HAS_ORIGIN→OriginType „private_haushalt“.
6. DisposalRule_Farbeimer_city REQUIRES_CHECK→Condition_1 „ist_farbe“ mit Frage „Handelt es sich um Farbe/Lack?“.
7. Condition_1 HAS_OPTION→Value_1_yes („ja“), Value_1_no („nein“).
8. Value_1_yes NEXT_CHECK→Condition_2 „aggregatzustand“ mit Frage „Ist die Farbe noch flüssig oder bereits eingetrocknet?“.
9. Nutzer antwortet auf Condition_1: „Ja“ → Pfad folgt NEXT_CHECK zu Condition_2.
10. Condition_2 HAS_OPTION→Value_2_flüssig, Value_2_fest.
11. Nutzer antwortet „flüssig“ → Value_2_flüssig IMPLIES_STREAM→WasteStream „Sonderabfall (Schadstoffmobil/Wertstoffhof)“.
12. Validierung: System prüft, ob WasteStream „Sonderabfall (Schadstoffmobil/Wertstoffhof)“ ALLOWS_AVV→AVVCode „20 01 27*“. Nur wenn diese Beziehung existiert, gilt die Regel als konsistent.
13. WasteStream „Sonderabfall (Schadstoffmobil/Wertstoffhof)“ AVAILABLE_AT→Facility „Wertstoffhof Nord“.
14. Zusätzlich: DisposalRule_Farbeimer_city MANDATES→Instruction „Nicht in Ausguss oder Toilette kippen.“.

Bot-Antwort (vereinfacht):
„Flüssige Farbe gilt als gefährlicher Abfall (z. B. AVV 20 01 27*). Bitte auf keinen Fall in Ausguss oder Toilette kippen. Gib den Farbeimer beim Schadstoffmobil oder am Wertstoffhof ab (z. B. Wertstoffhof Nord).“

7.3 Elektrische Zahnbürste (Campus-Override)

1. Nutzer: „Wohin mit einer elektrischen Zahnbürste an der Hochschule?“
2. WasteItem "Zahnbürste" wird erkannt.
3. WasteItem HAS_RULE→Rule_Zahnbuerste_city und Rule_Zahnbuerste_campus.
4. Rule_Zahnbuerste_campus scope="campus"; Rule_Zahnbuerste_city scope="city".
5. Rule_Zahnbuerste_campus OVERRIDES→Rule_Zahnbuerste_city.
6. Beide Regeln REQUIRES_CHECK→Condition „is_electric“ mit questionText „Handelt es sich um eine elektrische Zahnbürste?“.
7. Nutzer antwortet „Ja“ → ConditionValue true.
8. ConditionValue true (in der Campus-Regel) IMPLIES_STREAM→WasteStream „E-Schrott

(Campus-Sammelstelle)“.
9. Optional: WasteStream „E-Schrott (Campus-Sammelstelle)“ ALLOWS_AVV→AVVCode für Elektrogeräte.
10. WasteStream „E-Schrott (Campus-Sammelstelle)“ AVAILABLE_AT→Facility „Campus-Sammelstelle Elektrogeräte“.

Bot-Antwort (vereinfacht):
„Elektrische Zahnbürsten gelten als Elektrogeräte. Bitte gib sie an der E-Schrott-Sammelstelle auf dem Campus ab. Die nächste Sammelstelle befindet sich in Gebäude X, Erdgeschoss.“

8. Empfohlene nächste Schritte

1. Core-Schema in Neo4j implementieren:
  • Labels für die Kernknoten anlegen (WasteItem, WasteLegalClass, AVVCode, HazardType, DisposalRule,Condition,ConditionValue,
    OriginType, LegalProvision, WasteStream, Container, Building, Room, Facility, Organization, Material).
  • Sinnvolle Constraints/Indizes setzen (z. B. auf WasteItem.name, AVVCode.code, WasteStream.name).

2. Pilot-Datenimport:
  • Abfall-ABC-Daten (Group_3) für eine Teilmenge typischer Fälle (Restmüll, Bio, Glas, Papier, E-Schrott).
  • AVV-Daten (Group_2) für kritische Abfälle (Farben/Lacke, Chemikalien, Batterien, Laborabfälle).
  • Erste LegalProvision-Knoten aus der Abfallsatzung (z. B. zentrale Pflichten/Verbote).

3. Schrittweise Ausdifferenzierung der Logik:
  • Zuerst kritische und häufig nachgefragte Abfallarten modellieren (Farben, Batterien, Elektrogeräte, Sperrmüll).
  • Hinweise-Spalte in Conditions (If/Else, inkl. NEXT_CHECK) und Instructions (Handlungen) aufspalten.
  • Campus-spezifische DisposalRules mit scope="campus" anlegen und Stadt-Regeln via OVERRIDES übersteuern.

4. Campus-Infrastruktur ergänzen:
  • Gebäude, Räume und Container aus Campus-Masterdaten importieren.
  • Container via ALLOWS_STREAM und LOCATED_AT an WasteStreams und Räume/Fazilitäten anbinden.

5. Material-Layer iterativ aufbauen:
  • Zuerst für wenige, kritische Produktgruppen (Getränkebehälter, Batterien, Laborgefäße) Materials und MADE_OF pflegen.

- LLM-gestützte Vorschläge für Materialien testen und manuell validieren.

6. LLM-/Agent-Integration vorbereiten:
   - Systemprompt formulieren, der die wichtigsten Knotentypen und Relationen kurz beschreibt.
   - Standard-Cypher-Query-Patterns definieren (z. B. „Item → Regeln → Bedingungen → Streams (inkl. NEXT_CHECK) → Container/Location“).
   - Optional: EDC-/Tripel-Extraktion nutzen, um aus weiteren Textquellen (PDFs, Webseiten) den Graph schrittweise zu erweitern.

🧠 Hinweis – kann später gelöscht werden:
Dieser Abschnitt funktioniert gut als „Roadmap“-Folie oder als Ausblick in einer schriftlichen Arbeit.