295 - Backend

Auftrag Fallstudie 1 – SetUp

|  |  |
| --- | --- |
| Richtzeit  2h | Sozialform  Einzelarbeit |

## Vorbereitung

In diesem Auftrag setzt du eine API für Todo-Items um. Dabei können Clients die API verwenden, um sich anzumelden, neue Items zu erfassen und diese mit Tags zu gruppieren. Im Laufe dieser Fallstudie wirst du den Code auf Basis des Spring Boot Frameworks erarbeiten. Nutze die Präsis bei Unklarheiten und nimm dir Zeit die Mechanismen in Spring zu verstehen.

## Setup

Kopiere das Projekt aus der Ordnerstruktur lokal auf deinen Computer.  
Öffne das Projekt mit deiner IDE und importiere das Gradle Projekt.

## Datenbank vorbereiten

Die API basiert auf folgender Datenstruktur. Schau dir das ERM an und mach dich mit der Struktur vertraut.

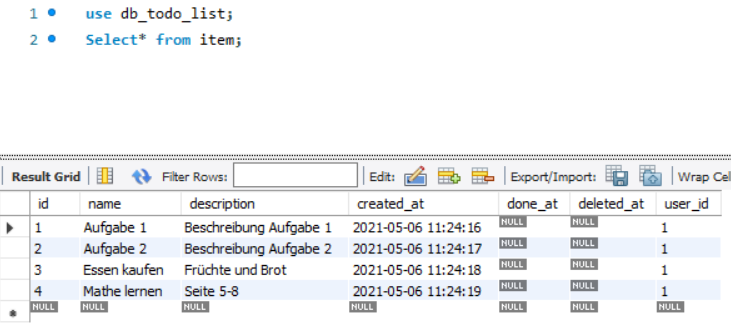
Unsere App soll zur Verwaltung von Todo-Items dienen. Zur Grupierung und Priorisierung sollen diesen Tags wie «wichtig», «unwichtig» oder «dringend» vergeben werden können.

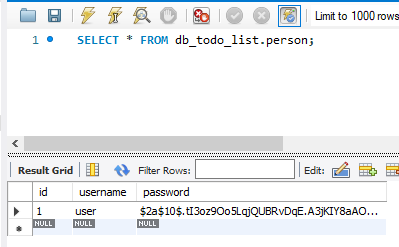
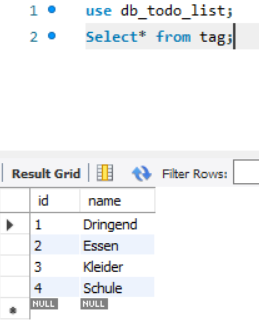
Diagram

Description automatically generated

## Inhalte einfügen

Damit wir nicht mit imaginären Daten arbeiten und testen müssen, haben wir ein Script mit Beispieldaten vorbereitet. Führe die Inhalte des Files: create-database-with-data.sql in der Mysql-Workbench aus. Verifiziere, dass alles richtig eingetragen wurde, mit folgenden Befehlen:



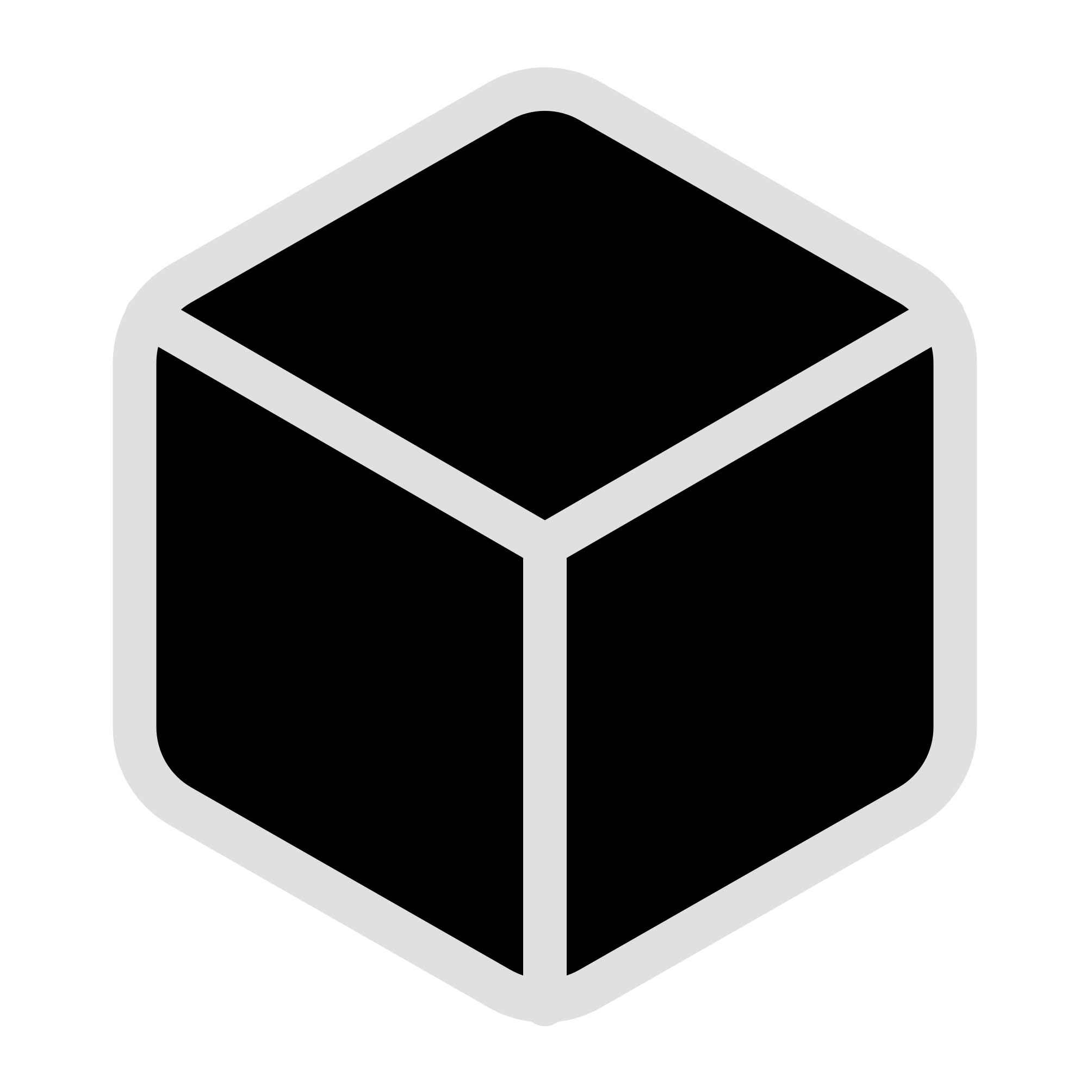


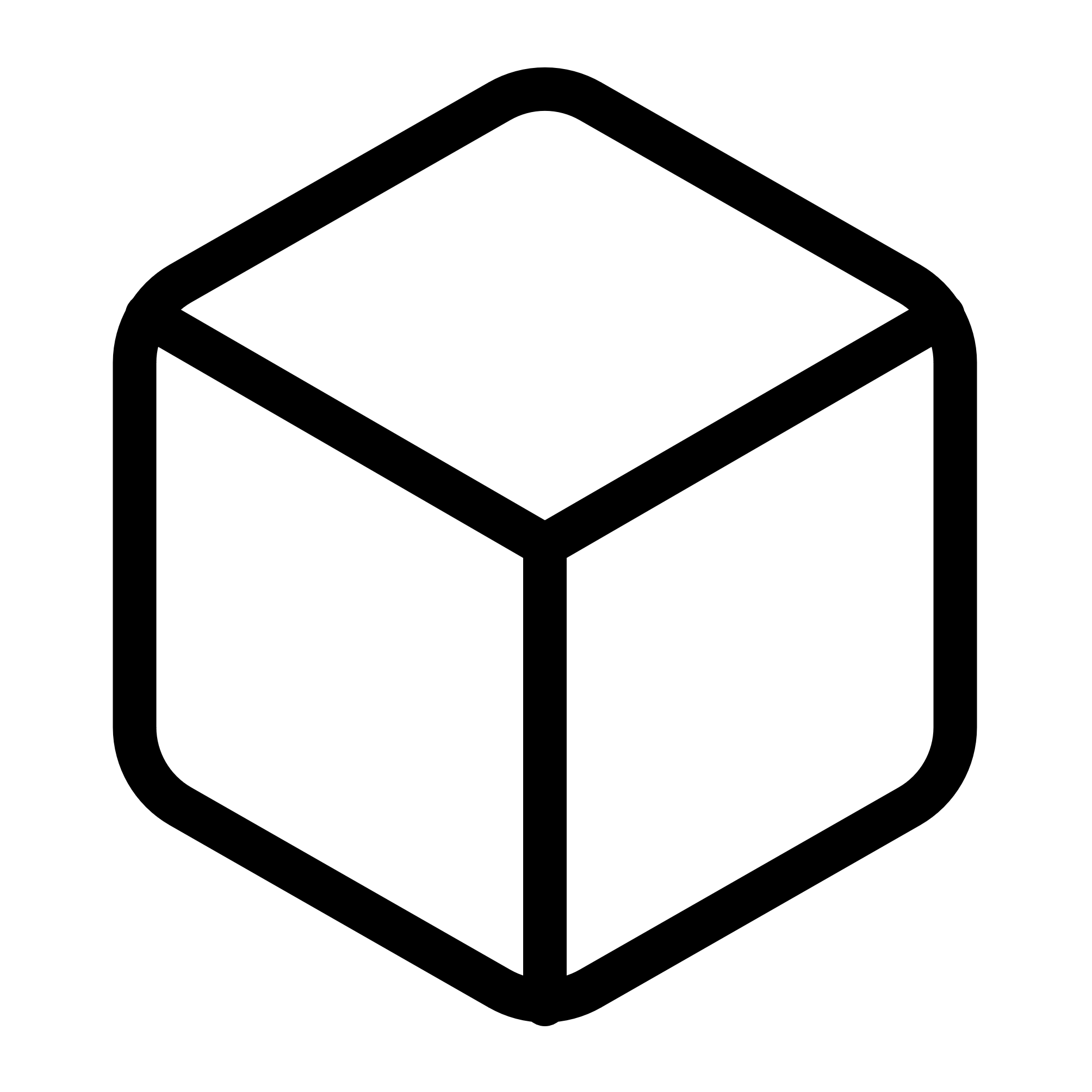
## Backend mit Datenbank verknüpfen

Nun möchten wir diese Datenbank in unserem Backend-Projekt verwenden. Dies erreichen wir mit dem application.properties File im Ordner Resources in unserem Projekt. Folgende Zeilen müssen darinstehen:

#Database  
spring.jpa.hibernate.ddl-auto=none  
spring.datasource.url=jdbc:mysql://${MYSQL\_HOST:localhost}:3306/db\_todo\_list?serverTimezone=UTC  
spring.datasource.username=db\_todo\_list  
spring.datasource.password=Pass4CRUD2020  
  
spring.jpa.properties.hibernate.globally\_quoted\_identifiers=true

## Projektstruktur

Java-Klassen sind in Packages gruppiert. Das Ziel ist hohe Kohäsion und lose Koppelung. Was bedeutet das? Kohäsion ist der Fachbegriff, wie stark Klassen innerhalb eines Packages zusammengehören. Koppelung ist der Fachbegriff, wie stark Packages mit anderen Packages verknüpft sind. Wir wollen also einfach gesagt alles, was zueinander gehört, in ein Package tun, wobei einzelne Packages selbst nur wenig Abhängigkeiten zu anderen Packages haben sollen.

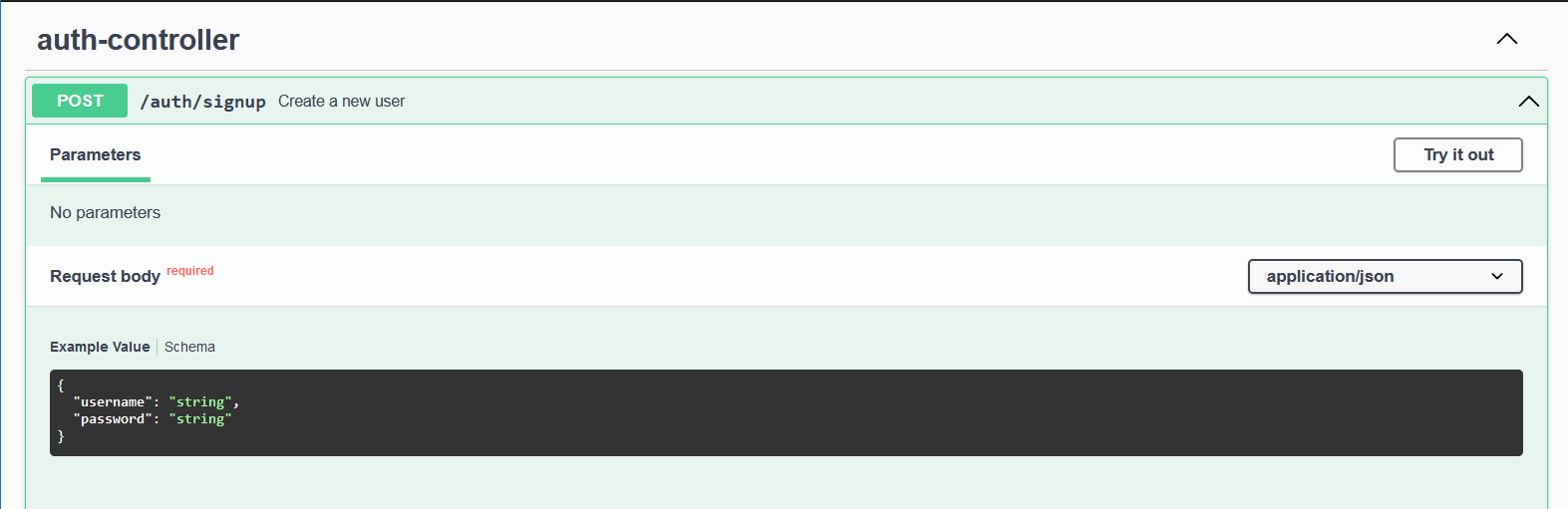
Öffne alle Packages und mach dich als erstes mit der Projektstruktur vertraut. Was wird wo gemacht? Füll die folgende Tabelle aus.

|  |  |
| --- | --- |
| Verantwortlichkeit | Klasse(n)/Package |
| Starten der Applikation und Main-Methode | TodoApplication.java |
| OpenAPI-Konfiguration | Todolist.openApiConfiguration.java |
| Konstanten für die Security-Konfiguration | Todolist.Security.SecurityConstants |
| Endpoint, um Personen zu verwalten | Todolist.person.personContoller? |
| Schemas für die Datenübertragung | \*DTO.java |
| Alles was die Applikationssicherheit betrifft | Todolist.Security |
| Alles was Personen betrifft | Todolist.person |
| Alles was items Betrifft | todolist.item |
| Alles was Tags betrifft | Todolist.tag |
| Security für Authentifikation | AuthController.java |
| Mapper für Person?? | PersonMapper.java |
|  | Item.java |

## Signup

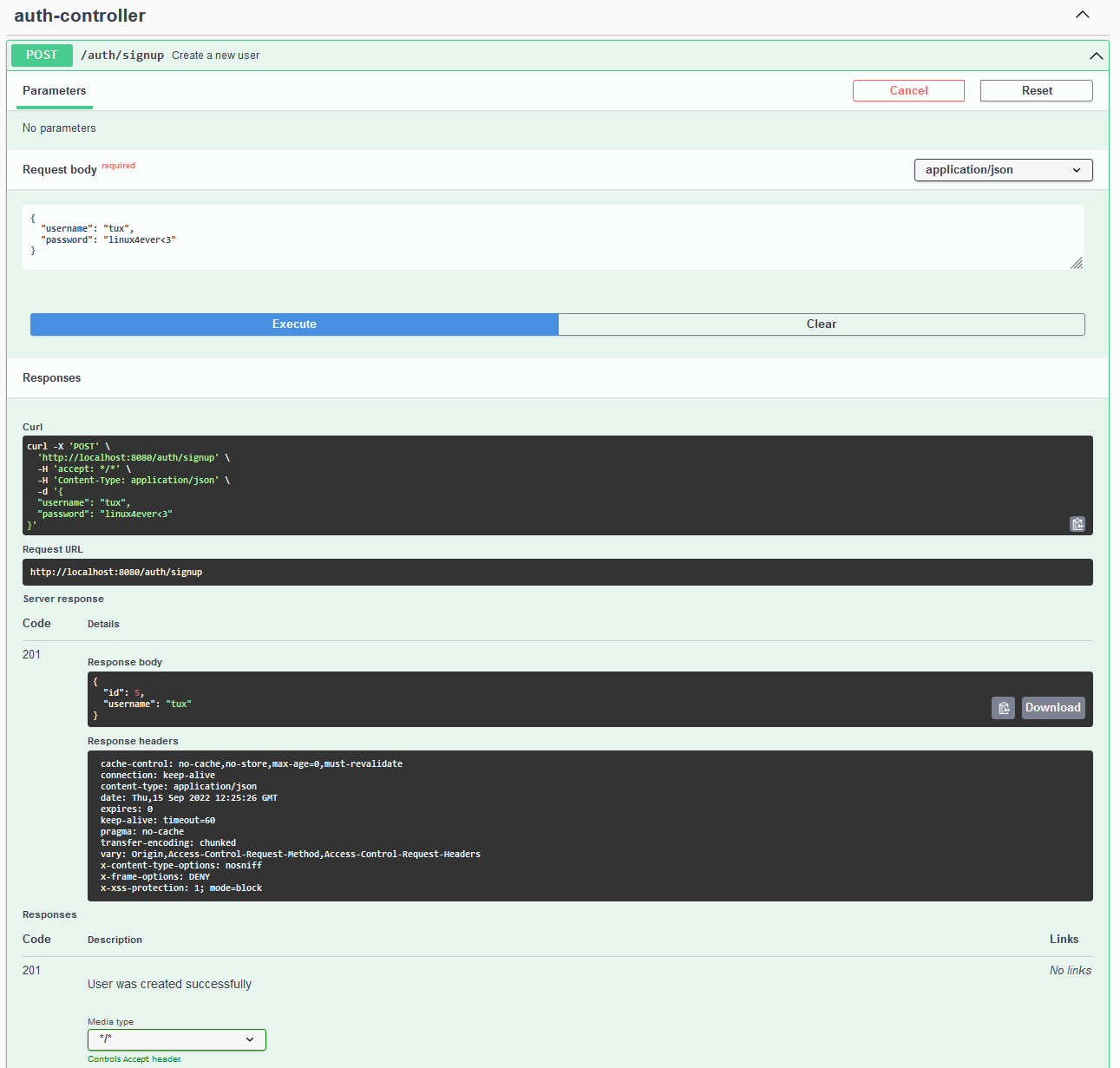
Das Projekt solltest du starten können. Starte das Projekt mit gradle bootRun. In der Konsole solltest du den Startvorgang beobachten können. Die OpenAPI-Konfiguration und die springdoc-Dependency sorgen dafür, dass ein SwaggerUI zur Verfügung gestellt wird. Dies ist eine interaktive Doku, auf welcher man auch schnell etwas ausprobieren kann. Den Link dazu findest du im Readme.

Scrolle zum auth-controller, drücke beim /auth/signup auf «Try it out» um einen User zu erstellen.



Du solltest eine Eingabemaske mit Beispieldaten sehen. Gib einen Usernamen und ein Passwort ein und drücke auf Execute.

Schauen wir uns an, was hier alles sichtbar ist:



8

7

6

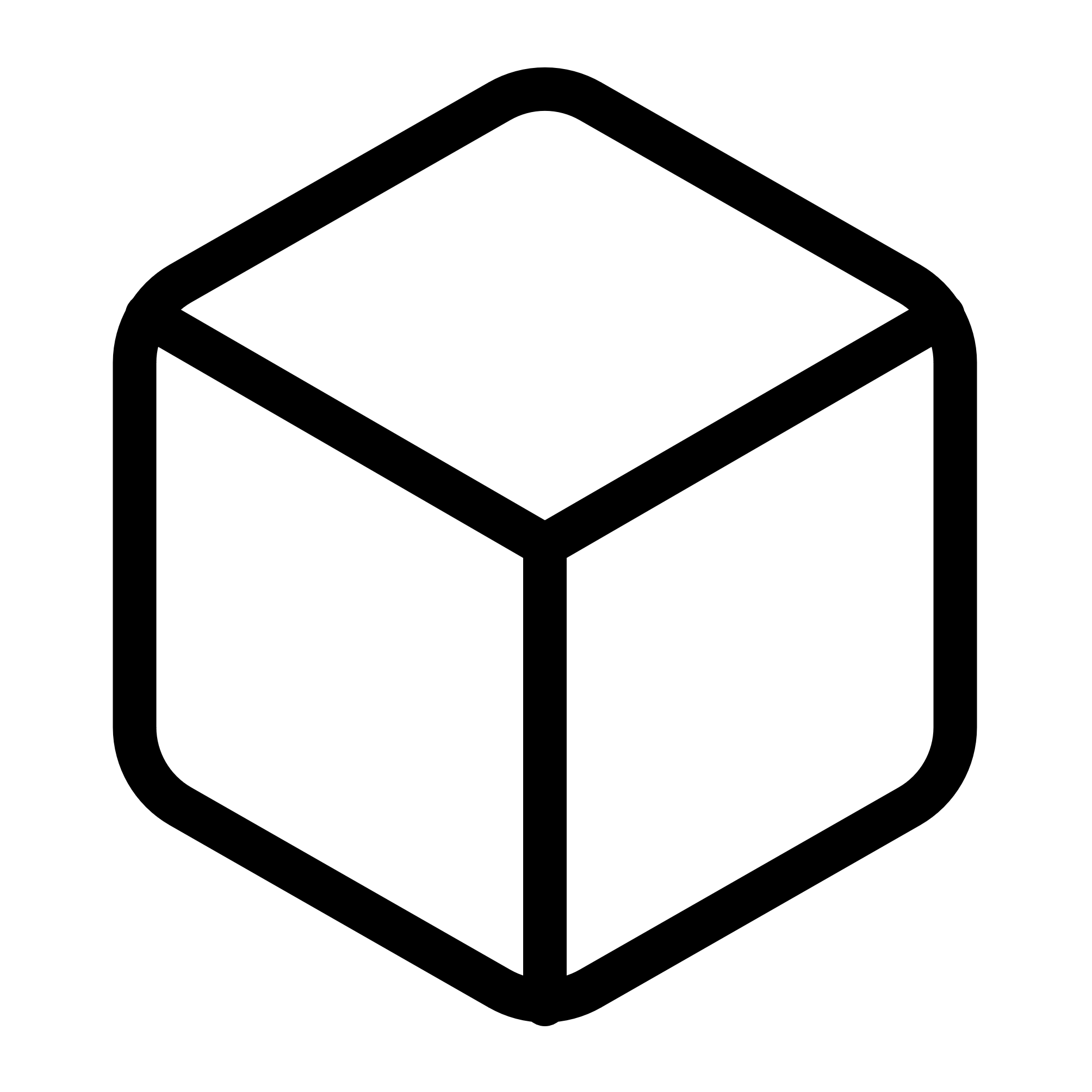
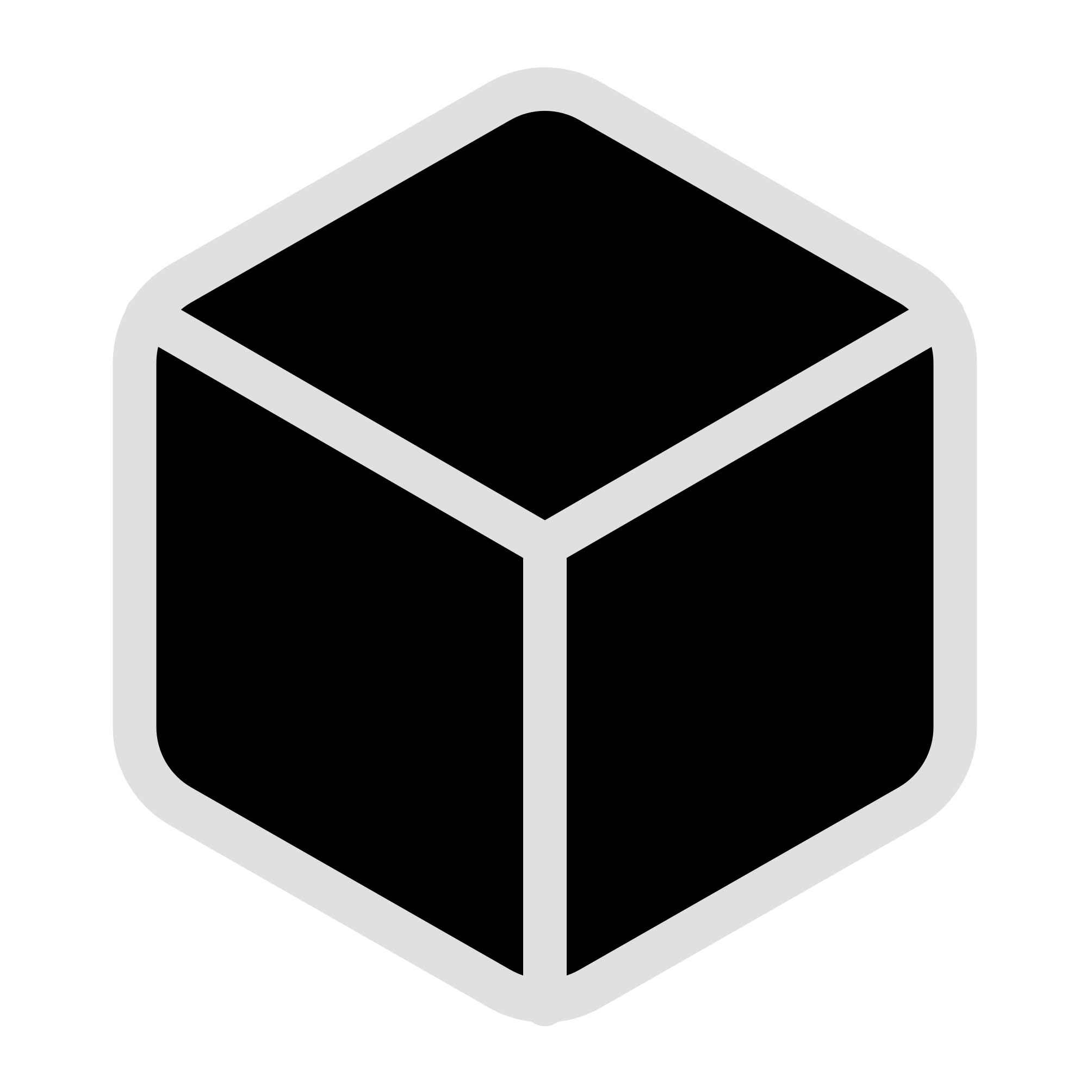
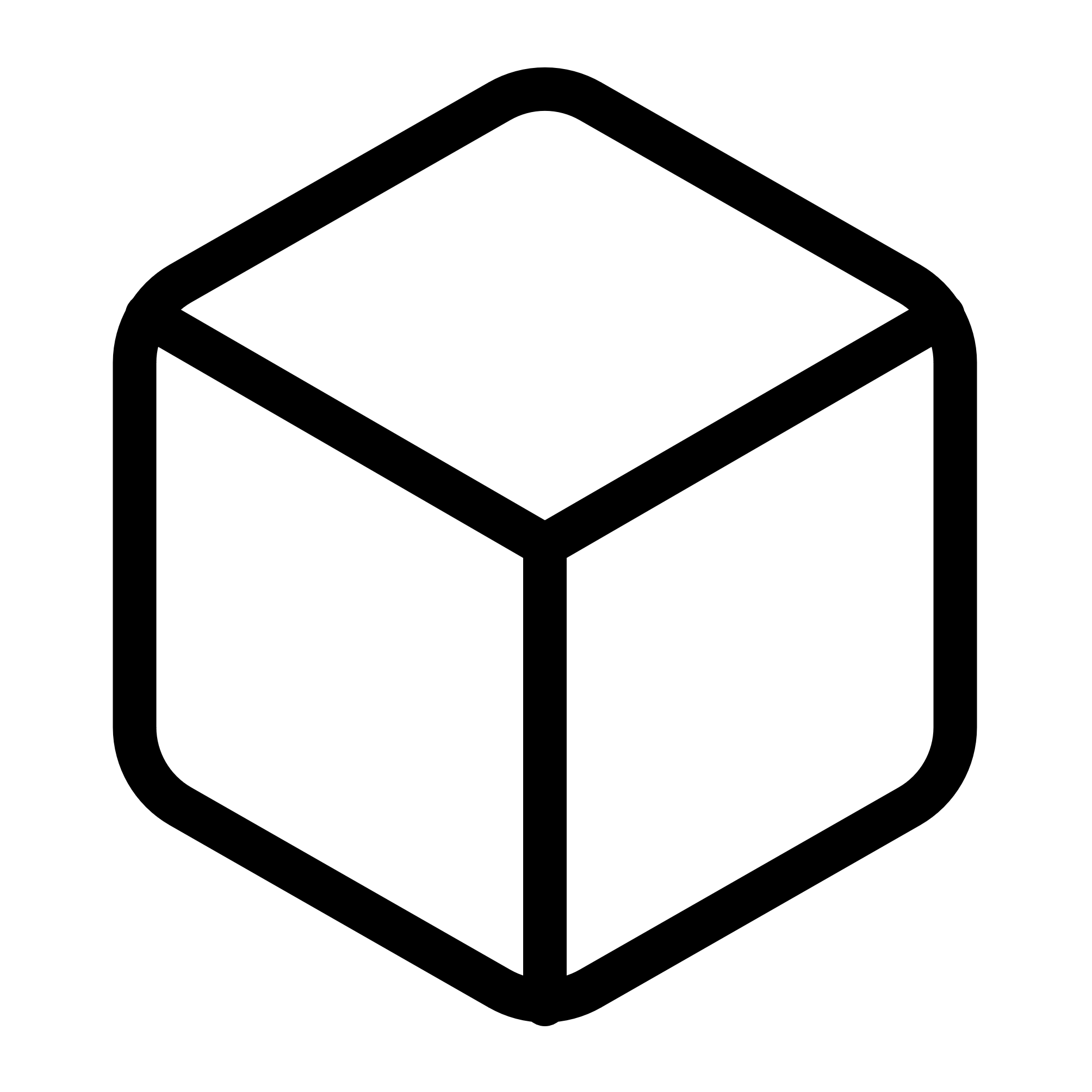
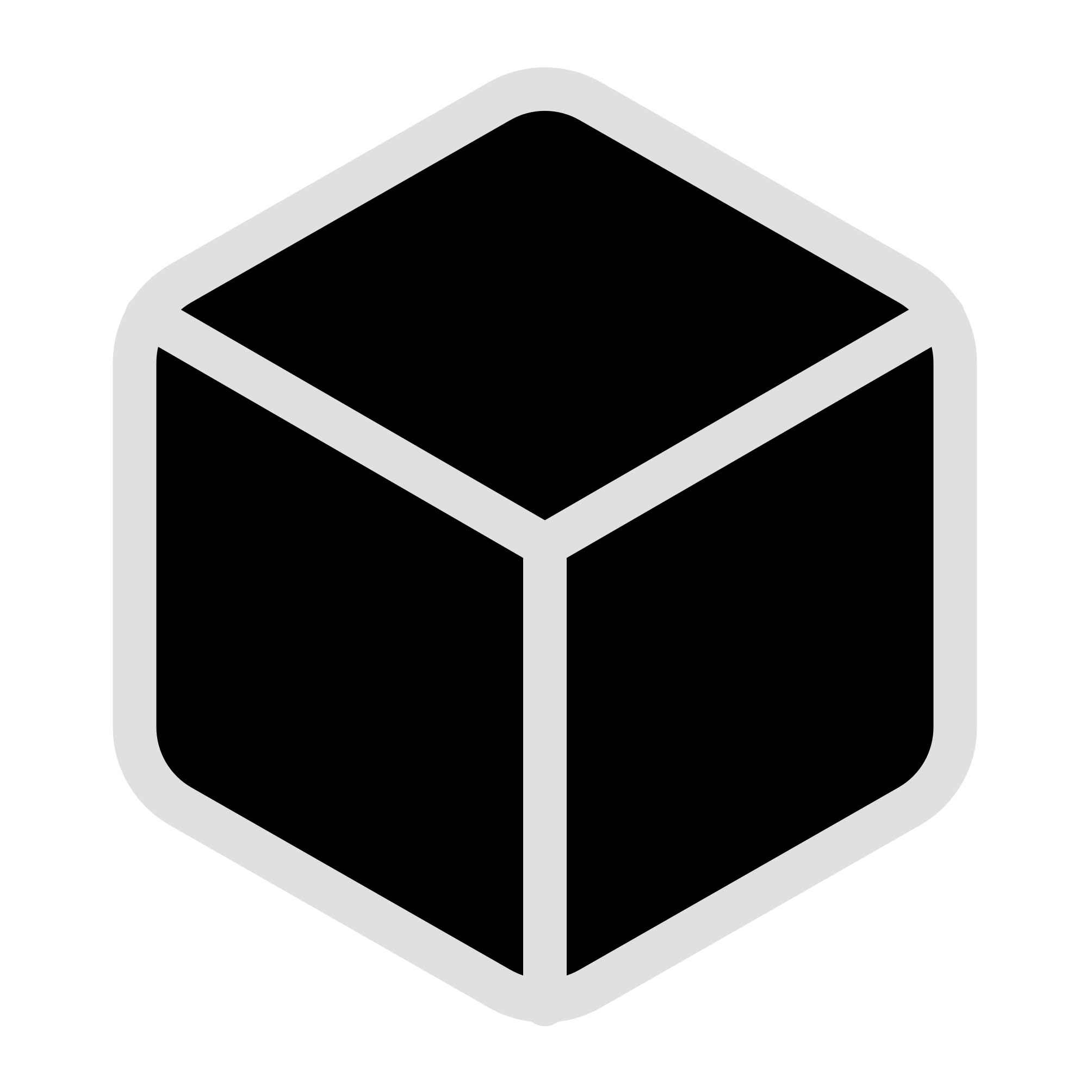
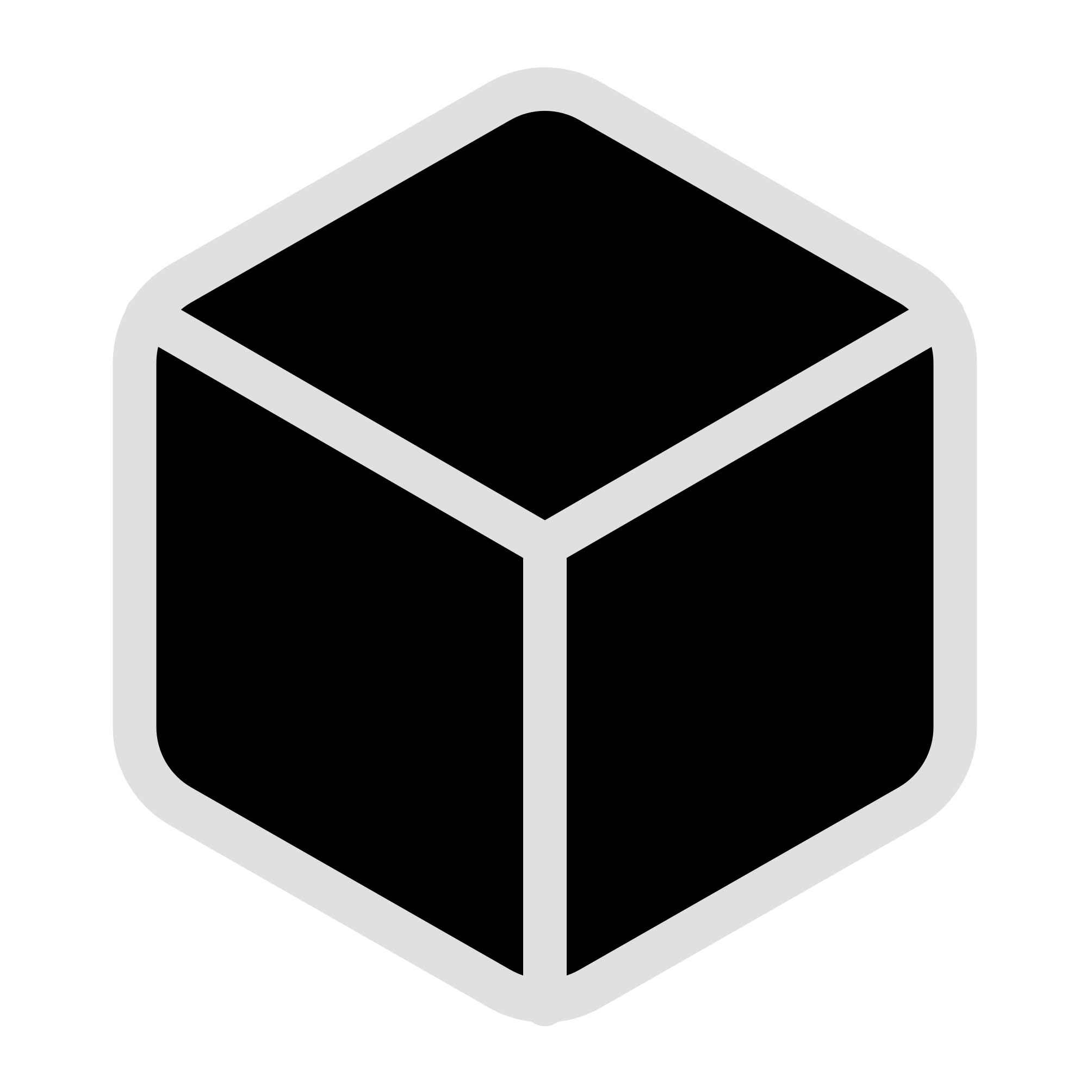
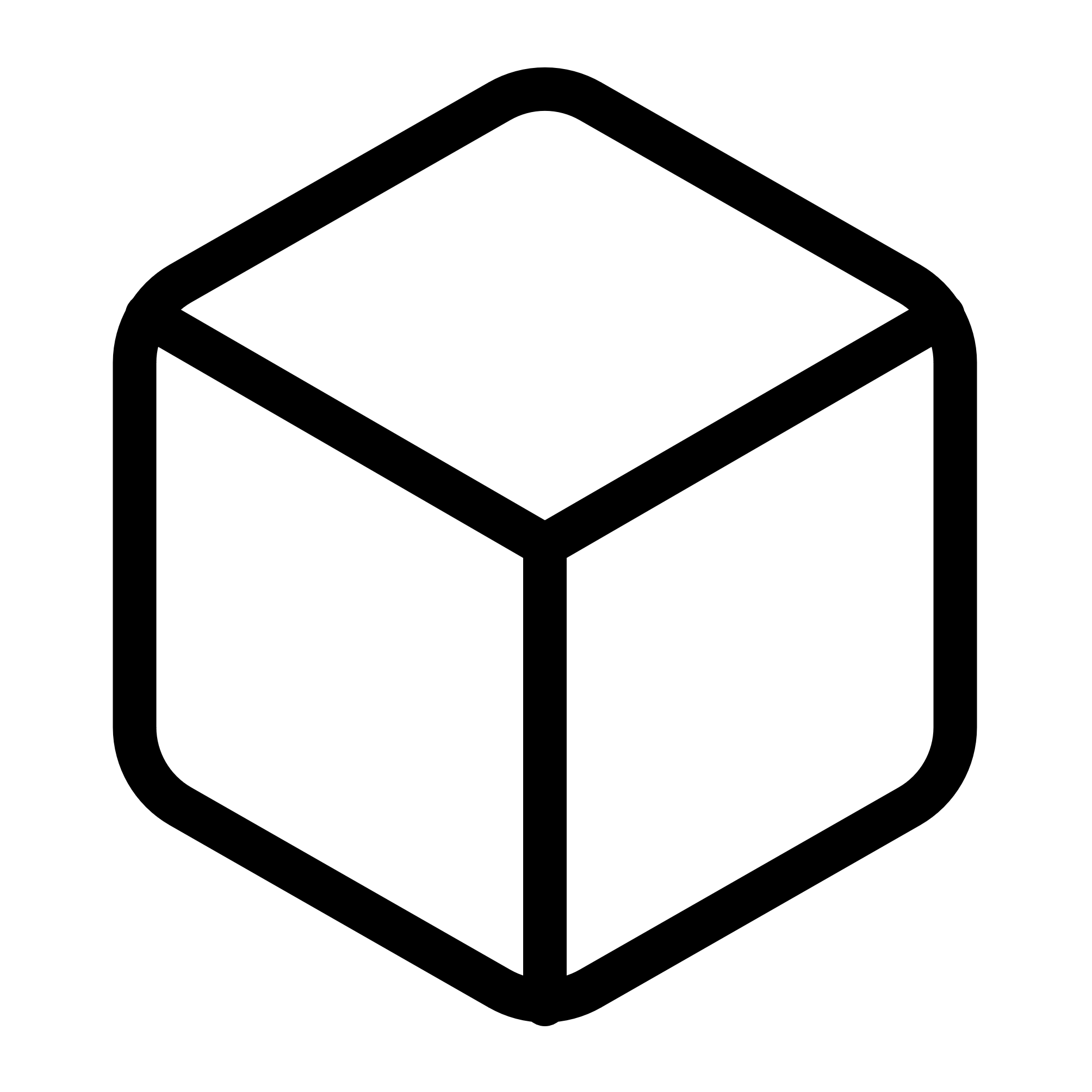
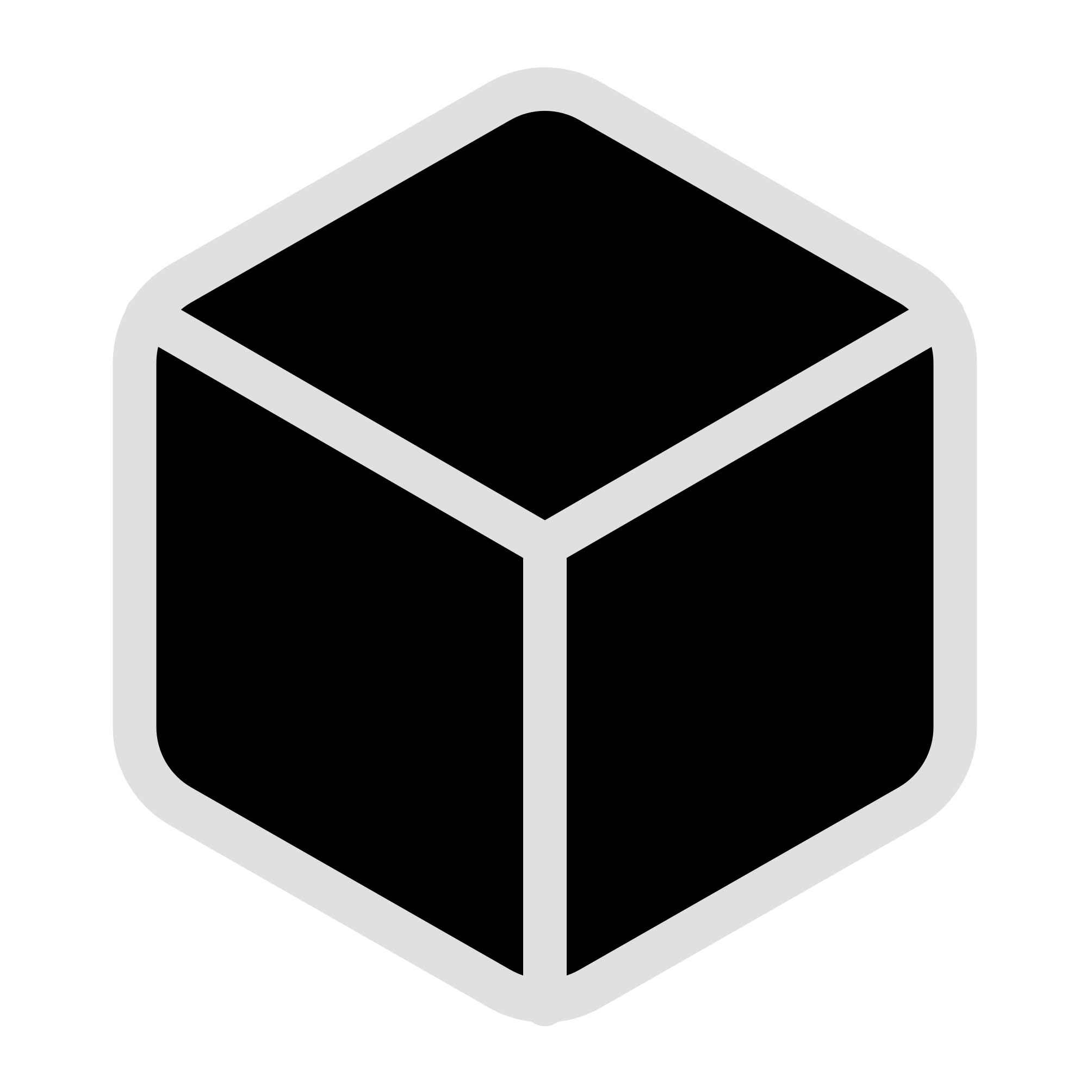
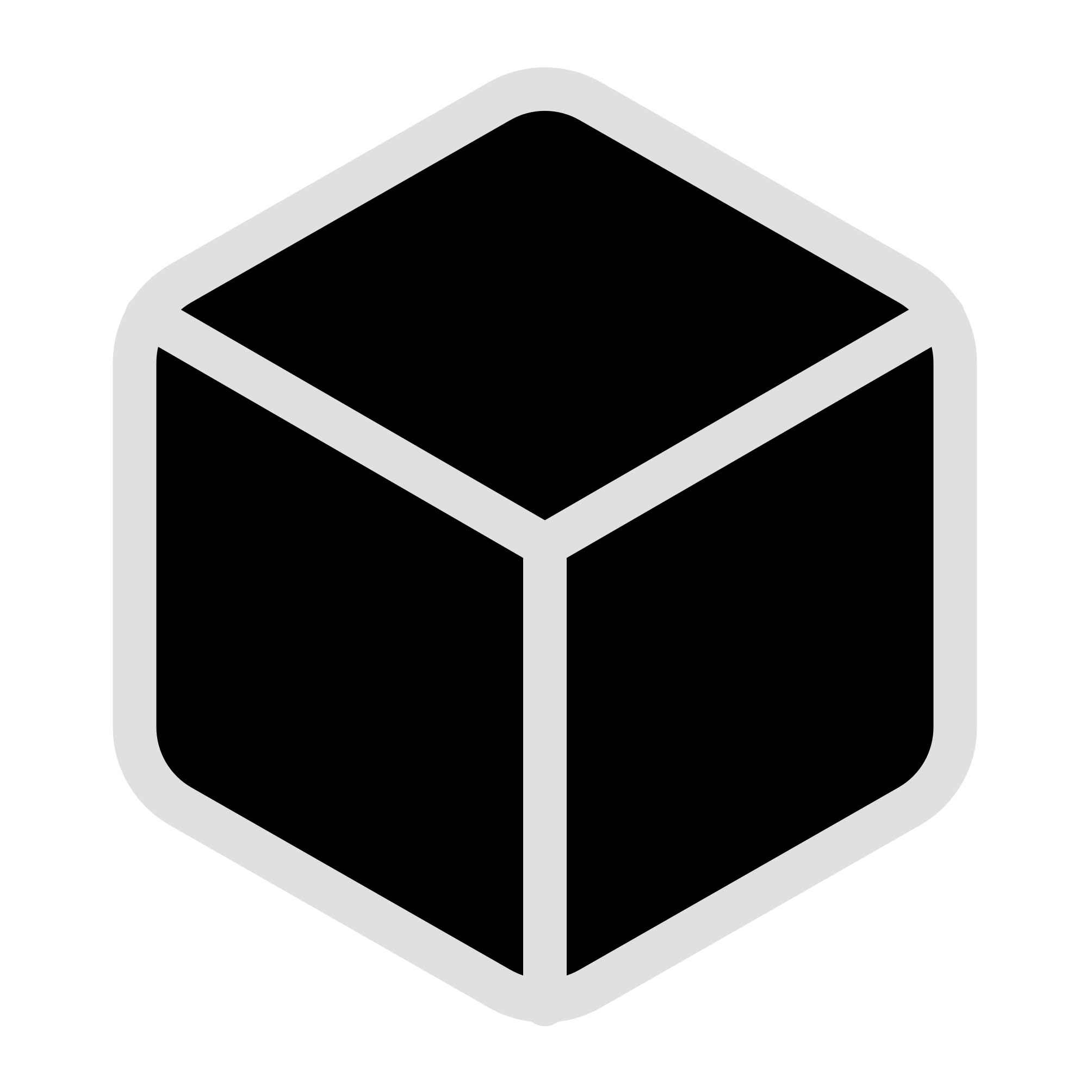
5

4

3

2

1

1. Methode und URI 
2. Content-Type Request-Header 
3. Request Body 
4. Submit-Button 
5. Request als curl-Befehl (curl ist ein commandline http-client) 
6. Status-Code und Response-Body 
7. Response-Headers 
8. Statische Dokumentation von verschiedenen Antworten 

Stell sicher, das du vom signup 201-Created zurückbekommst. Pass bei alfälligen 4XX Errors den Request an.

## Login

Wir verwenden für API’s gerne Bearer-Tokens. «Bearer» ist englisch und heisst übersetzt «Träger». Zu verstehen ist dies im Sinne von «Berechtigt den Träger dieses Tokens». Wir bekommen also quasi ein Ticket, mit dem wir Requests machen dürfen.

Das Token erhalten wir durch einen POST request an /auth/signin. Was müssen wir mitsenden? Wie sieht die Antwort aus? Finde es im Swagger-UI!

Das effektive Token könnte wie folgt aussehen:  


Dieses müssen wir wie folgt als Request-Header setzen:



Im Swagger-UI können wir dies einfach durch klicken auf das Schloss ganz oben rechts oder bei irgendeinem Request tun, indem wir nur das Token als Value eingeben.

Bei Postman ist es am einfachsten auf Level Collection im Authorization Tab den Type Bearer Token einzustellen und das Token im entsprechenden Feld hinzuzufügen.

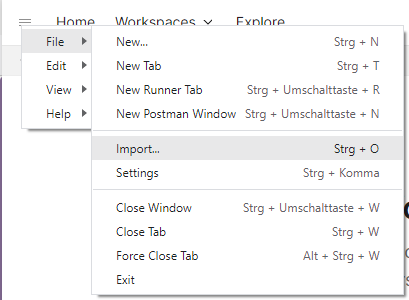
**Achtung!**

Solche Tokens sind, wie Passwörter, geheim zu halten. Sie beinhalten zusätzliche Daten (wie z.B. Berechtiungen) und autorisieren jeden, der das Token hat, bis das Token abläuft.



Wir werden uns zu einem späteren Zeitpunkt noch mit den Details dazu beschäftigen.

## OpenAPI

Swagger-UI ist ein Tool, welches unter der Haube auf dem OpenAPI-Standard basiert. Unter [vielen weiteren Tools](https://openapi.tools/) unterstützt auch Postman diesen Standard. Ganz oben im Swagger-UI ist der Link zum api-docs.json. Du kannst den Link oder das JSON kopieren und im Postman über den Import-Button oder das Menü oben links importieren.

Wenn du noch Zeit hast, spiele etwas mit Postman/Swagger-UI herum.