Document Oriented Middleware using MongoDB

URL	https://elearning.tgm.ac.at/mod/assign/view.php?id=150533

Warehouse-API

Dieses Projekt ist eine Flask-basierte API zur Verwaltung von Lagerhäusern und Produkten. Es verwendet MongoDB zur Datenspeicherung und bietet Endpunkte für CRUD-Operationen an Lagerhäusern und Produkten.

Projekt-Setup

Voraussetzungen

- Python 3.x
- Flask
- Flask-PyMongo
- MongoDB

Code-Übersicht

`app.py`

Datenbankverbindung

```
def get_db():
    if 'db' not in g:
        g.db = PyMongo(app).db
        app.logger.info("Database connected")
    return g.db
```

Diese Funktion stellt eine Verbindung zur MongoDB-Datenbank her und speichert sie im Flask `g` Objekt.

Warehouse-Routes

```
@app.route("/warehouse")
@app.route('/warehouse/<id>', methods=['GET'])
def get_warehouse(id=None):
    db = get_db()
    if not id:
        return list(db.warehouse.find().sort("warehouseID", 1))
    else:
        r = db.warehouse.find_one({"warehouseID": int(id)})
        if not r:
            r =  {"message": "Not found"}
        return r

@app.route("/warehouse", methods=['POST'])
def add_warehouse():
    db = get_db()
    try:
        db.warehouse.insert_one(request.json)
        return {"message": "Warehouse added successfully"}
    except DuplicateKeyError:
```

```
        return {"message": "Warehouse already exists"}
    except OperationFailure:
        return {"message": "Invalid data"}

@app.route("/warehouse/<id>", methods=['DELETE'])
def delete_warehouse(id):
    db = get_db()
    r = db.warehouse.delete_one({"warehouseID": int(id)})
    if r.deleted_count == 0:
        return {"message": "Warehouse not found"}
    return {"message": "Warehouse deleted successfully"}
```

Diese Routen behandeln CRUD-Operationen für Lagerhäuser:

- `GET /warehouse` und `GET /warehouse/<id>;`: Alle Lagerhäuser oder ein spezifisches Lagerhaus per ID abrufen.
- `POST /warehouse`: Ein neues Lagerhaus hinzufügen.
- `DELETE /warehouse/<id>;`: Ein Lagerhaus per ID löschen.

Product-Routes

```
@app.route("/product")
@app.route('/product/<id>', methods=['GET'])
def get_product(id=None):
    db = get_db()
    if not id:
        return list(db.product.find().sort("productID", 1))
    else:
        r = db.product.find_one({"productID": str(id)})
        if not r:
            r =  {"message": "Not found"}
        return r

@app.route("/product", methods=['POST'])
def add_product():
```

```
    db = get_db()
    try:
        db.product.insert_one(request.json)
        return {"message": "Product added successfully"}
    except DuplicateKeyError:
        return {"message": "Product already exists"}
    except OperationFailure:
        return {"message": "Invalid data"}

@app.route("/product/<id>", methods=['DELETE'])
def delete_product(id):
    db = get_db()
    r = db.product.delete_one({"productID": str(id)})
    if r.deleted_count == 0:
        return {"message": "Product not found"}
    return {"message": "Product deleted successfully"}
```

Diese Routen behandeln CRUD-Operationen für Produkte:

- `GET /product` und `GET /product/<id>;`: Alle Produkte oder ein spezifisches Produkt per ID abrufen.
- `POST /product`: Ein neues Produkt hinzufügen.
- `DELETE /product/<id>;`: Ein Produkt per ID löschen.

API-Referenz

Lagerhaus-Endpunkte

- **GET /warehouse**
  - Ruft eine Liste aller Lagerhäuser ab.
  - **Antwort**: Liste von Lagerhäusern.
- **GET /warehouse/<id>**
  - Ruft ein spezifisches Lagerhaus per ID ab.
  - **Antwort**: Lagerhaus-Objekt oder `{"message": "Not found"}`.

- **POST /warehouse**
  - Fügt ein neues Lagerhaus hinzu.
  - **Anfragekörper**: JSON-Objekt, das das Lagerhaus repräsentiert.
  - **Antwort**: `{"message": "Warehouse added successfully"}` oder `{"message": "Warehouse already exists"}`.
- **DELETE /warehouse/<id>**
  - Löscht ein Lagerhaus per ID.
  - **Antwort**: `{"message": "Warehouse deleted successfully"}` oder `{"message": "Warehouse not found"}`.

Produkt-Endpunkte

- **GET /product**
  - Ruft eine Liste aller Produkte ab.
  - **Antwort**: Liste von Produkten.
- **GET /product/<id>**
  - Ruft ein spezifisches Produkt per ID ab.
  - **Antwort**: Produkt-Objekt oder `{"message": "Not found"}`.
- **POST /product**
  - Fügt ein neues Produkt hinzu.
  - **Anfragekörper**: JSON-Objekt, das das Produkt repräsentiert.
  - **Antwort**: `{"message": "Product added successfully"}` oder `{"message": "Product already exists"}`.
- **DELETE /product/<id>**
  - Löscht ein Produkt per ID.
  - **Antwort**: `{"message": "Product deleted successfully"}` oder `{"message": "Product not found"}`.

Fragen

- Nennen Sie 4 Vorteile eines NoSQL Repository im Gegensatz zu einem relationalen DBMS
  - Flexibles Datenmodell
    - NoSQL Datenbanken erlauben dynamische und unstrukturierte Daten ohne festes Schema
  - Horizontale Skalierbarkeit
    - Einfaches Verteilen der Daten auf mehrere Server
  - Hohe Performance
    - Schneller Datenzugriff durch denormalisierte Datenstrukturen
  - Bessere Verfügbarkeit
    - Durch verteilte Systeme
- Nennen Sie 4 Nachteile eines NoSQL Repository im Gegensatz zu einem relationalen DBMS
  - Eingeschränkte Konsistenz
    - ACID-Eigenschaften werden nicht immer garantiert, was zu Inkonsistenzen führen kann
  - Komplexe Abfragen schwieriger
    - Keine standardisierte Sprache wie SQL
  - Begrenzte Transaktionsunterstützung
    - Atomare Operationen über mehrere Dokumente nicht möglich
  - Redundanz der Daten
    - Durch Denormalisieren werden Daten oft mehrfach gespeichert
- Welche Schwierigkeiten ergeben sich bei der Zusammenführung der Daten?
  - Doppelte Daten, weil keine Relationen
  - Flexibles Datenmodell kann `null` Werte ergeben
- Welche Arten von NoSQL Datenbanken gibt es? Nennen Sie einen Vertreter für jede Art?

    - Key–Value
        - Redis
    - Spaltenorientiert
        - Apache Cassandra
    - Dokumentenorientiert
        - MongoDB
    - Graphbasiert
        - InfoGrid
- Beschreiben Sie die Abkürzungen CA, CP und AP in Bezug auf das CAP Theorem
    - Consistency
    - Availabillity
    - Partition tolerance
    - Ein Datenbanksystem kann nur 2 der 3 Attribute haben
- Mit welchem Befehl können Sie den Lagerstand eines Produktes aller Lagerstandorte anzeigen.
    - Mit einer Aggregation `db.warehouse.aggreate({});`
- Mit welchem Befehl können Sie den Lagerstand eines Produktes eines bestimmten Lagerstandortes anzeigen.
    - `db.warehouse.find({});`

Mongosh

Read

> `db.warehouse.find({key: value})` ohne key: value um alle ergebnisse zu bekommen

Write

```
db.warehouse.updateOne(
   { "warehouseID": 1 },
   { $push:
      { "warehouseData.0.productData": {
         "productID": "99-123456",
         "productName": "Neues Produkt",
         "productQuantity": 1000
      }}
   }
```

Pusht ein neues Objekt in eine collection

Update

```
db.warehouse.updateOne(
  { "warehouseID": 1, "warehouseData.0.productData.productID": "1" },
  { $set:
     { "warehouseData.0.productData.$[elem].productQuantity": 3000 }
  },
  { arrayFilters: [{ "elem.productID": "1" }] }
)
```

DeleteOne

```
db.warehouse.updateOne(
  { "warehouseID": 1 },
  { $pull: { "warehouseData.0.productData": { "productID": "1" } } }
)
```

DeleteAll

```
db.warehouse.deleteOne(
  { "warehouseID": 1 }
```

```
)
```

Fragestellungen

1. Wie ist der lagerbestand eines Produkts in allen Lagern?

```
db.warehouse.aggregate([
  { $unwind: "$warehouseData" },
  { $unwind: "$warehouseData.productData" },
  { $match: { "warehouseData.productData.productID": "1" } },
  { $group: { _id: "$warehouseData.productData.productID",
                    totalQuantity:
                       { $sum: "$warehouseData.productData.productQuantity" }
])
```

2. Welche Produkte haben in einem Lager eine Stückzahl unter 1000?

```
db.warehouse.aggregate([
  { $unwind: "$warehouseData" },
  { $unwind: "$warehouseData.productData" },
  { $match: { "warehouseID": 1, "warehouseData.productData.productQuantit
  { $project:
     { _id: 0,
        productID: "$warehouseData.productData.productID",
        productName: "$warehouseData.productData.productName",
        productQuantity: "$warehouseData.productData.productQuantity" } }
])
```

3. Welche Lager haben ein Produkt lagernd?

```
db.warehouse.find(
   { "warehouseData.productData.productID": "1" },
   { "warehouseName": 1, "warehouseCity": 1, _id: 0 }).pretty()
```

Quellen

https://www.mongodb.com/resources/products/compatibilities/setting-up-flask-with-mongodb

https://en.wikipedia.org/wiki/CAP_theorem