Okuma: Ürün Modeli Hakkında

Final Proje Genel Bakış bölümünde detaylandırıldığı gibi, bir e-Ticaret Uygulaması için Ürün Kataloğu arka ucu için bir mikro hizmet oluşturmalısınız.

Ürün Modeli, çeşitli özellikleri, içe aktarılan modüller, Ürün Sınıfı ve kullanılan farklı yöntemleri anlamak ve analiz etmek için service/models.py dosyasına bakalım.

Tahmini okuma süresi: 10 dakika

Özellikler

Bu modelde kullanılan farklı özellikler şunlardır:

1. id - ürünün kimliği
2. name - ürünün adı
3. description - ürünün açıklaması
4. price - ürünün fiyatı
5. available - ürünün mevcut olup olmadığı
6. category - ürünün ait olduğu kategori

Import İfadeleri

1. **import logging** - Python'da `import logging` ifadesini kullanarak yerleşik logging modülünü içe aktarırsınız. Logging modülünü içe aktardıktan sonra, DEBUG, INFO, WARNING, ERROR ve CRITICAL gibi çeşitli önem seviyelerinde mesajları günlüğe kaydetmek için fonksiyonlarını ve sınıflarını kullanabilirsiniz.

2. **from enum import Enum** - Python'da `from enum import Enum` ifadesini kullanarak enum modülünden Enum sınıfını içe aktarırsınız. Enum'lar, Python'da adlandırılmış değerler kümesini tanımlamanın bir yoludur. Enum'larınızı tanımlayarak, sabit bir değer kümesini temsil eden sembolik adlar oluşturabilirsiniz.

3. **from decimal import Decimal** - Python'da `from decimal import Decimal` ifadesini kullanarak decimal modülünden Decimal sınıfını içe aktarırsınız. Decimal sınıfı, ondalık kayan nokta aritmetiğini destekler ve özellikle hassas ondalık hesaplamalar yaparken kullanışlıdır.

4. **from flask import Flask** - Python'da `from flask import Flask` ifadesini kullanarak flask modülünden Flask sınıfını içe aktarırsınız. Flask, web uygulamaları oluşturmanıza olanak tanıyan popüler bir Python web çerçevesidir. Flask sınıfı, Flask çerçevesinin temel bileşenidir ve bir Flask uygulama örneği oluşturmakla sorumludur.

5. **from flask_sqlalchemy import SQLAlchemy** - Python'da `from flask_sqlalchemy import SQLAlchemy` ifadesini kullanarak flask_sqlalchemy modülünden SQLAlchemy sınıfını içe aktarırsınız. Flask SQLAlchemy, popüler bir Nesne-İlişkisel Eşleme (ORM) kütüphanesi olan SQLAlchemy ile entegre olan Flask web çerçevesi için bir eklentidir.

Yöntemler & Sınıflar

1. **db = SQLAlchemy()** - Bu ifade, bir veritabanı bağlantısı kurmak ve SQLAlchemy'yi Flask uygulamasıyla entegre etmek için bir Flask uygulama örneği ile ilişkilendirilmiş bir SQLAlchemy nesnesi oluşturur.
   Aşağıdaki kod veritabanını başlatır.

   ```
   def init_db(app):
       """Initialize the SQLAlchemy app"""
       Product.init_db(app)
   ```

2. **class DataValidationError(Exception)**
   Bu ifade, yerleşik Exception sınıfından türeyen `DataValidationError` adlı özel bir istisna sınıfını tanımlar. Özel istisna sınıfları oluşturarak, kodunuzda belirli hata koşullarını veya olağanüstü senaryoları tanımlayabilir ve gerektiğinde bu istisnaları yükseltebilirsiniz. Bu özel istisnayı kullanarak, veri doğrulaması yaptığınızda, veri geçerli değilse uygun bir hata mesajıyla `DataValidationError` istisnasını yükseltir.

3. **Enumeration of Product Categories**
   Aşağıdaki kod parçası, ek kategorilerle birlikte Kategori enumeration'unu tanımlar ve her bir üyeye açık tam sayı değerleri atar. Her bir üye, geçerli bir ürün kategorisini temsil eder.

   ```
   class Category(Enum):
       """Enumeration of valid Product Categories"""
       UNKNOWN = 0
       CLOTHS = 1
       FOOD = 2
       HOUSEWARES = 3
       AUTOMOTIVE = 4
       TOOLS = 5
   ```

4. **Ürün Sınıfı**
   Aşağıdaki kod parçası, db-model sınıfını temel alarak bir Ürün model sınıfı tanımlar. Ürün modeli, veritabanındaki bir tabloyu temsil eder ve her tablo sütununu bir sınıf niteliği olarak tanımlarsınız.
   Ürün modelindeki sütun tanımları aşağıda detaylandırılmıştır:

- id: Ürün tablosunun birincil anahtarını temsil eden bir tam sayı sütunu.
- name: Ürünün adını temsil eden, maksimum 100 karakter uzunluğunda bir dize sütunu. `nullable=False`, veritabanındaki karşılık gelen sütunun NULL değerine sahip olamayacağını belirtir.
- description: Ürünün açıklamasını temsil eden, maksimum 250 karakter uzunluğunda bir dize sütunu. `nullable=False`, veritabanındaki karşılık gelen sütunun NULL değerine sahip olamayacağını belirtir.
- price: Ürünün fiyatını temsil eden bir sayısal sütun. Kesin ondalık hesaplamalar için `db.Numeric` tipini kullanırsınız. `nullable=False`, veritabanındaki karşılık gelen sütunun NULL değerine sahip olamayacağını belirtir.
- available: Ürünün kullanılabilirliğini temsil eden bir boolean sütun. Varsayılan değeri True'dur. `nullable=False`, veritabanındaki karşılık gelen sütunun NULL değerine sahip olamayacağını belirtir.
- category: Ürünün kategorisini temsil eden bir enum sütunu. Daha önce tanımlanan Category enum'unu kullanır. Enum değerlerini karşılık gelen veritabanı değerlerine eşlemek için `db.Enum` tipini kullanırsınız. `server_default` argümanı, sütun için varsayılan bir değer sağlayan Category.UNKNOWN.name olarak ayarlanmıştır.

Ürün modelini tanımlayarak, SQLAlchemy'nin ORM özelliklerini kullanarak veritabanındaki karşılık gelen tabloyla etkileşimde bulunabilirsiniz. Bu etkileşim, Ürün nesneleri üzerinde CRUD (Oluşturma, Okuma, Güncelleme, Silme) işlemleri gerçekleştirmenizi ve tanımlı sütunları kullanarak veritabanını sorgulamanızı sağlar.

```
class Product(db.Model):
    """
    Class that represents a Product
    This version uses a relational database for a hidden persistence
    from us by SQLAlchemy's object-relational mappings (ORM)
    """
    ##################################################
    # Table Schema
    ##################################################
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    description = db.Column(db.String(250), nullable=False)
    price = db.Column(db.Numeric, nullable=False)
    available = db.Column(db.Boolean(), nullable=False, default=True)
    category = db.Column(
        db.Enum(Category), nullable=False, server_default=(Category.UNKNOWN.name)
    )
```

Örnek Metotlar

1. `__repr__()` metodu

Aşağıdaki kod parçası, bir Ürün nesnesinin string temsilini döndürmek için `__repr__()` metodunu tanımladığınız Ürün modelini göstermektedir. Döndürülen string, ürünün adını ve kimliğini içerir.

```
def __repr__(self):
        return f"<Product {self.name} id=[{self.id}]>"
```

2. `create()` yöntemi
   Aşağıdaki kod parçacığı, veritabanında yeni bir Product nesnesi oluşturmakla sorumlu olan Product sınıfındaki `create()` yöntemini göstermektedir.

```
def create(self):
        """
        Creates a Product to the database
        """
        logger.info("Creating %s", self.name)
        # id must be none to generate next primary key
        self.id = None  # pylint: disable=invalid-name
        db.session.add(self)
        db.session.commit()
```

3. `update()` yöntemi

Aşağıdaki kod parçası, veritabanındaki mevcut bir Ürün nesnesini güncellemekten sorumlu Product sınıfındaki `update()` yöntemini göstermektedir.

```
def update(self):
        """
        Updates a Product to the database
        """
        logger.info("Saving %s", self.name)
        if not self.id:
            raise DataValidationError("Update called with empty ID field")
        db.session.commit()
```

Not: Product nesnesinin id niteliği üzerinde bir kontrol yapılmaktadır. Eğer id boşsa, boş bir ID alanına sahip bir ürün üzerinde güncelleme yapıldığını belirten bir DataValidationError hatası fırlatılır.

4. `delete()` yöntemi

Aşağıdaki kod parçası, bir Product nesnesini veritabanından silmekten sorumlu olan Product sınıfındaki `delete()` yöntemini göstermektedir.

```
def delete(self):
        """Removes a Product from the data store"""
        logger.info("Deleting %s", self.name)
        db.session.delete(self)
        db.session.commit()
```

5. `serialize() method`
   Aşağıdaki kod parçası, bir Product nesnesini sözlük temsilinde dönüştüren Product sınıfındaki `serialize()` yöntemini göstermektedir; bu, JSON serileştirmesi, veri aktarımı veya API yanıtları gibi çeşitli amaçlar için faydalı olabilir.

```
 def serialize(self) -> dict:
        """Serializes a Product into a dictionary"""
        return {
            "id": self.id,
            "name": self.name,
            "description": self.description,
            "price": str(self.price),
            "available": self.available,
            "category": self.category.name  # convert enum to string
        }
```

6. `deserialize()` yöntemi

Aşağıdaki kod parçası, bir Product nesnesini bir sözlük temsili ile verilerle doldurmanıza olanak tanıyan Product sınıfındaki `deserialize()` yöntemini göstermektedir. Bu ayrıştırma süreci, yapılandırılmış verileri ilgili özelliklere sahip bir nesneye dönüştürür.

```
def deserialize(self, data: dict):
        """
        Deserializes a Product from a dictionary
        Args:
            data (dict): A dictionary containing the Product data
        """
        try:
            self.name = data["name"]
            self.description = data["description"]
            self.price = Decimal(data["price"])
            if isinstance(data["available"], bool):
                self.available = data["available"]
            else:
                raise DataValidationError(
                    "Invalid type for boolean [available]: "
                    + str(type(data["available"]))
                )
            self.category = getattr(Category, data["category"])  # create enum from string
        except AttributeError as error:
            raise DataValidationError("Invalid attribute: " + error.args[0]) from error
        except KeyError as error:
            raise DataValidationError("Invalid product: missing " + error.args[0]) from error
        except TypeError as error:
            raise DataValidationError(
                "Invalid product: body of request contained bad or no data " + str(error)
            ) from error
        return self
```

Not: Çeşitli istisnalar (AttributeError, KeyError, TypeError) yakalanır ve uygun hata mesajlarıyla DataValidationError istisnaları olarak yeniden fırlatılır. Bu istisnalar, sağlanan veri sözlüğünün gerekli anahtarları eksik olduğu veya geçersiz veri içerdiği senaryoları yönetir.

Sınıf Yöntemleri

1. `init_db() yöntemi`

Aşağıdaki kod parçası, veritabanı oturumunu başlatmak ve gerekli SQLAlchemy tablolarını oluşturmakla sorumlu olan Product sınıfında `init_db` adında bir sınıf yöntemi içermektedir.

```
@classmethod
   def init_db(cls, app: Flask):
       """Initializes the database session
       :param app: the Flask app
       :type data: Flask
       """
       logger.info("Initializing database")
       # This is where we initialize SQLAlchemy from the Flask app
       db.init_app(app)
       app.app_context().push()
       db.create_all()  # make our sqlalchemy tables
```

2. `all() yöntemi`

Aşağıdaki kod parçası, veritabanından tüm Ürün nesnelerini getiren Ürün sınıfında `all()` adında bir sınıf yöntemi içermektedir.

```
@classmethod
   def all(cls) -> list:
       """Returns all of the Products in the database"""
       logger.info("Processing all Products")
       return cls.query.all()
```

3. `find() method`
   Aşağıdaki kod parçası, veritabanında bir Ürünü ID'sine göre bulmakla sorumlu olan Product sınıfında `find()` adında bir sınıf yöntemi içermektedir.

```
@classmethod
    def find(cls, product_id: int):
        """Finds a Product by it's ID
        :param product_id: the id of the Product to find
        :type product_id: int
        :return: an instance with the product_id, or None if not found
        :rtype: Product
        """
        logger.info("Processing lookup for id %s ...", product_id)
        return cls.query.get(product_id)
```

Not: Belirtilen ID ile bir Ürün nesnesi veritabanında bulunursa, bu nesne döndürülür. Aksi takdirde, None döndürülür.

`4. find_by_name() metodu`

Aşağıdaki kod parçası, Ürün sınıfı içinde, eşleşen isme sahip tüm Ürün nesnelerini veritabanından alan `find_by_name()` adlı bir sınıf metodunu içermektedir.

```
@classmethod
   def find_by_name(cls, name: str) -> list:
       """Returns all Products with the given name
       :param name: the name of the Products you want to match
       :type name: str
```

```
        :return: a collection of Products with that name
        :rtype: list
        """
        logger.info("Processing name query for %s ...", name)
        return cls.query.filter(cls.name == name)
```

Not: Bu yöntemi Product sınıfında çağırarak geçerli bir ad sağlarsanız, o adı eşleşen Product nesnelerinin bir koleksiyonunu alırsınız.

5. `find_by_price()` yöntemi
   Aşağıdaki kod parçası, fiyatı eşleşen tüm Product nesnelerini veritabanından almayı sağlayan Product sınıfında `find_by_price()` adlı bir sınıf yöntemini içermektedir.

```
@classmethod
    def find_by_price(cls, price: Decimal) -> list:
        """Returns all Products with the given price
        :param price: the price to search for
        :type name: float
        :return: a collection of Products with that price
        :rtype: list
        """
        logger.info("Processing price query for %s ...", price)
        price_value = price
        if isinstance(price, str):
            price_value = Decimal(price.strip(' "'))
        return cls.query.filter(cls.price == price_value)
```

Not: Bu yöntemi Product sınıfında çağırarak geçerli bir fiyat sağladığınızda, o fiyata uyan Product nesnelerinin bir koleksiyonunu alacaksınız.

6. `find_by_availability()` yöntemi
   Aşağıdaki kod parçası, kullanılabilirliğe göre veritabanından tüm Product nesnelerini getiren Product sınıfında bulunan `find_by_availability()` adlı bir sınıf yöntemini içermektedir.

```
@classmethod
    def find_by_availability(cls, available: bool = True) -> list:
        """Returns all Products by their availability
        :param available: True for available products
        :type available: str
        :return: a collection of available Products
        :rtype: list
        """
        logger.info("Processing available query for %s ...", available)
        return cls.query.filter(cls.available == available)
```

Not: Ürün sınıfında bu yöntemi çağırarak geçerli bir kullanılabilirlik değeri sağlarsanız (sağlanmadığında varsayılan olarak True olur), belirtilen kullanılabilirlikle eşleşen Ürün nesnelerinin bir koleksiyonunu alırsınız.

7. `find_by_category()` yöntemi
   Aşağıdaki kod parçası, Ürün sınıfı içinde `find_by_category()` adında bir sınıf yöntemi içermektedir; bu yöntem, kategorilerine göre veritabanından tüm Ürün nesnelerini alır.

```
@classmethod
    def find_by_category(cls, category: Category = Category.UNKNOWN) -> list:
        """Returns all Products by their Category
        :param category: values are ['MALE', 'FEMALE', 'UNKNOWN']
        :type available: enum
        :return: a collection of available Products
        :rtype: list
        """
        logger.info ("Processing category query for %s ...", category.name)
        return cls.query.filter(cls.category == category)
```

Not: Bu yöntemi Product sınıfında çağırarak geçerli bir kategori değeri sağlarsanız (sağlanmazsa varsayılan olarak Category.UNKNOWN olarak ayarlanır), belirtilen kategoriye uyan Product nesnelerinin bir koleksiyonunu alırsınız.

Yazar(lar)

- Anita Narain