**بسمه تعالی**

پیمانکاران طرح: آرش آقاپور(403521051)، امیررضا طالبی جوجاده(403521474)، سید محسن باکمال(403521087) و حسین نادری حیدری(403522163)

مستندات پروژه سایت فروش لوازم آرایشی-بهداشتی و پیشنهاد روتین پوستی(سایت **FaceBloom):**

***توضیحات اولیه فریم‌ورک و راه‌اندازی آن:***

**قسمت backend این پروژه مبتنی بر زبان python و فریم‌ورک Fast api کدزنی شده است. سادگی و سرعت در کدزنی و ایجاد api و نیز وجود swagger سیار در این فریم‌ورک تیم را مجاب به استفاده از این فریم‌ورک نمود.**

**برای این فریم‌ورک ابتدای امر یک app ت تعریف می‌گردد.(app = FastAPI())همچنین به واسطه فایل database.py برنامه را به دیتابیس وصل می نماییم. لازم به ذکر است که دیتابیس به کار رفته در این پروژه sqlite می باشد. به واسطه کتابخانه sqlalchemy موتور(engine) و base دیتابیس را تعریف نموده و به واسطه تابع get\_db دیتابیس را generate می نماییم.**

**برای setup این پروژه می بایست ابتدا یک محیط مجازی ساخت. ساخت و استفاده آن در ویندوز بدین شرح است:**

**python –m venv env**

**./env/Scripts/Activate.ps1**

**و برای بالا آوردن خود فریم ورک در آدرس /docs 127.0.0.1:8000می بایست از دستور زیر بهره برد:**

**uvicorn AP-Project.main:app –reload**

**لازم به ذکر است که می بایست کتابخانه uvicorn و fastapi را نیز نصب نمود. اما نیاز به نصب جداگانه هرکدام از کتابخانه ها نیست چرا که تمامی کتابخانه های لازم در فایل requirements.txt جمع آوری شده‌اند. برای نصبشان کافی است در محیط مجازی دستور زیر را تایپ کرد:**

**Pip install –r requirements.txt**

**کتابخانه های مورد نیاز عبارت اند از:**

fastapi

uvicorn

sqlalchemy

passlib

bcrypt

pydantic

passlib[bcrypt]

python-jose[cryptography]

requests

redis

**نکته قابل ملاحظه دیگر وجود داکر و داکرایز کردن این پروژه است. سودمندی داکر برای ایجاد کانتینری سبک تر از محیط مجازی و قابل ران کردن آن روی دستگاه های مختلف می باشد. برای لود کردن و ران کردن تصویر داکر می بایست دستورات زیر را استفاده کرد:**

**docker pull AP-Project-app:latest**

**docker run -d -p 8000:8000 --name facebloom-container AP-Project -app:latest**

**که Dockerfile قرار داده شده در پروژه مطابق با آن است :**

FROM python:3.12

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY AP-Project ./AP-Project

CMD ["uvicorn", "AP-Project.main:app", "--host", "0.0.0.0", "--port", "8000"]

**آخرین موردی که چالش در راه اندازی دارد مربوط به Redis است. این مورد بیشتر برای caching به کار می رود. علاوه بر اینکه کتابخانه قابل نصبی دارد می بایست فایل Redis-x64-5.0.14.1.zip را دانلود و استخراج نمود. سپس فایلهای** redis-server.exe و  **redis-cli.exe را در دو ترمینال جداگانه ران می نماییم و در صورت ping اگر پاسخ PONG داده شد به معنای صحت عملکرد است. این مورد با فایل redis\_client.py مطابق شده است:**

import redis

redis\_client = redis.Redis(

*host* = 'localhost',

*port* = 6379,

*db*=0,

*decode\_responses*=True

)

***توضیحات کلیات طرح به صورت تشریحی و غیرفنی:***

* **این پروژه قرار است سایتی را به عنوان پیشنهاد لوازم آرایشی-بهداشتی و پیشنهاد روتین پوستی تخصصی و شخصی سازی شده بر اساس انجام کوییز و تحلیل سلفی بالا بیاورد. که شامل بخش های متعددی است.**
* **سایت شامل دو بخش ورود کاربران می باشد. یک بخش برای کاربران عادی و بخش دیگر برای ادمین که با استفاده از JWT هر api سطح بندی بر اساس سطح دسترسی شده است. کاربر عادی ابتدا می بایست sign\_up بکند و در صورت انجام مورد در گذشته می بایست login بنماید.**
* **قسمت search & recommendations این برنامه شامل دو حالت coldstart و حالت پیشنهاد بر اساس کوییز و تجربیات کاربر می باشد . الگوریتم این بخش شامل ضریب دهی به بخش های مختلف در دو حالت مختلف سرچ و بر اساس عبارت سرچ شده است. که در فایل search.py این الگوریتم پیاده گشته است. همچنین فصل محل زندگی کاربر به عنوان آیتم امتیازی در این الگوریتم برای پیشنهاد نیز رعایت شده است.**
* **بخشی برای خرید وجود دارد که کاربر می بایست ابتدا محصولات مورد نیاز را به سبد خرید بیفزاید. سپس در ادامه خرید ها را تایید و submit کند. محصولات شامل تعداد می باشند که در صورت تایید خرید از تعداد آنها کاسته می گردد.**
* **بخش مهم دیگر این پروژه کوییز و پرسشنامه آن است که شامل 11 سوال برای یافتن اطلاعات پوستی کاربر می باشد. این مورد در کنار امکان آپلود سلفی (که بر اساس API اماده AILAB می باشد.) توانایی ساخت و پیشنهاد روتین را به سایت می دهد. بر اساس پاسخ این دو مورد سایت به کاربر سه روتین "Full Plan", "Hydration Plan", 'Minimalist Plan پیشنهاد می دهد. هر کدام از روتین ها گام هایی برای انجام دارند.**

if plan\_name == "Full Plan":

            base\_routine = ["cleanser", "toner", "serum", "moisturizer", "sunscreen"]

        elif plan\_name == "Hydration Plan":

            base\_routine = ["cleanser", "serum", "moisturizer", "mask/sunscreen"]

        else:

            base\_routine = ["cleanser", "moisturizer", "sunscreen"]

**برای هر قدم توضیحات نحوه استفاده از آن محصول را داده و بر اساس نتایج کوییز و سلفی و همچنین الگوریتم سرچ چند محصول مناسب برای پوست کاربر در هر قدم را پیشنهاد می دهد. مبسوط این توضیحات در فایل add\_product\_to\_routin.py می باشد.**

***توضیحات جداول و تیبل های مهم به کار رفته در دیتابیس پروژه:***

class Users\_for\_sign\_up(*Base*):

    \_\_tablename\_\_ = "Users\_sign\_up"

    user\_id = Column(Integer, *primary\_key*=True, *autoincrement*=True, *index*=True)

    user\_name = Column(String, *unique*=True,*index*=True)

    password = Column(String)

    created\_at = Column(DateTime, *default*=datetime.utcnow)

**جدول برای ورود کاربر که شامل ستون هایی برای id ، نام ، گذرواژه و تاریخ ورود است.**

class Users(*Base*):

    \_\_tablename\_\_ = "Users"

    user\_id = Column(Integer, *unique*=True, *primary\_key*=True)

    password = Column(String)

    skin\_type = Column(SqllEnum(*skin\_type\_allowed*), *nullable*=False)

    concerns = Column(JSON)

    preferences = Column(JSON)

    device\_type = Column(String)

    created\_at = Column(DateTime, *default*=datetime.utcnow)

    budget\_range = Column(Integer)

    browsing = relationship('Browsing\_History', *back\_populates*='user')

    purchasing = relationship('Purchase\_History', *back\_populates*='user2')

    cart\_items = relationship("Cart", *back\_populates*="user")

**از جداول مهم دیتابیس با داشتن نام و گذرواژه و ویزگی های پوستی و در کنار آنها نوع دستگاه و سه اتصال خارجی به جداول Browsing\_History و Purchase\_History و Cart.**

class Admins(*Base*):

    \_\_tablename\_\_ = "Admins"

    user\_id = Column(Integer, *primary\_key*=True, *autoincrement*=True, *index*=True)

    user\_name = Column(Integer, *index*=True)

    password = Column(String, *index*=True)

**جدول ادمین پنل شامل ایدی و نام و گذرواژه.**

class Products(*Base*):

    \_\_tablename\_\_ = "Products"

    product\_id = Column(Integer, *primary\_key*=True, *index*=True, *autoincrement*=True)

    name = Column(String, *index*=True)

    brand = Column(String, *index*=True)

    category = Column(String, *nullable*=False)

    skin\_types = Column(JSON, *nullable*=False)

    concerns\_targeted = Column(JSON, *nullable*=False)

    ingredients = Column(JSON, *nullable*=True)

    price = Column(Numeric(10, 2), *index*=True)

    rating = Column(Float, *index*=True)

    image\_url = Column(String, *nullable*=True, *index*=True)

    tags = Column(JSON, *index*=True)

    count = Column(Integer)

    stock = Column(Integer, *default*=0, *nullable*=False)

    Status = Column(Boolean, *default*=True)

    purchases = relationship("Purchase\_History", *back\_populates*="product")

    cart\_items = relationship("Cart", *back\_populates*="product")

**جدول محصولات شامل نام و برند و ویژگی های پوستی متناسب و قیمت و نمره و تعداد و وضعیت.همچنین شامل دو اتصال خارجی.**

class Cart(*Base*):

    \_\_tablename\_\_ = "Cart"

    id = Column(Integer, *primary\_key*=True, *index*= True)

    user\_id = Column(Integer, ForeignKey("Users.user\_id"))

    product\_id = Column(Integer, ForeignKey("Products.product\_id"))

    quantity = Column(Integer, *nullable*=False)

    user = relationship("Users", *back\_populates*="cart\_items")

    product = relationship("Products", *back\_populates*="cart\_items")

class Purchase\_History(*Base*):

    \_\_tablename\_\_ = 'Purchase\_History'

    id = Column(Integer, *primary\_key*=True, *autoincrement*=True, *index*=True)

    user\_id = Column(Integer, ForeignKey('Users.user\_id'), *index*=True)

    product\_id = Column(Integer, ForeignKey('Products.product\_id'), *index*=True)

    quantity = Column(Integer, *index*=True)

    timestamp = Column(DateTime, *default*=datetime.utcnow, *index*=True)

    user2 = relationship('Users', *back\_populates*='purchasing')

    # browsing2 = relationship('Browsing\_History', back\_populates='purchasing2')

    product = relationship('Products', *back\_populates*='purchases')

**جداول مرتبط با خرید و ثبت سوابق خرید.**

class FinalResult(*Base*):

    \_\_tablename\_\_ = 'final\_result'

    id = Column(Integer, *primary\_key*=True, *index*=True)

    user\_id = Column(Integer, *nullable*=False)

    # selfie\_result = Column(JSON, nullable=True)

    # quiz\_result = Column(JSON, nullable=False)

    # final\_result = Column(JSON, nullable=False)

    skin\_type = Column(SqllEnum(*skin\_type\_allowed*), *nullable*=False)

    concerns = Column(JSON)

    preferences = Column(JSON)

**جدول اصلی برای ذخیره سازی نتایج کوییز.البته جدول دیگری به نام quiz\_result نیز وجود دارد ولیکن از آن استفاده خاصی نشده و دلیل وجود آن برای ذخیره جداگانه نتایج کوییز در صورت لزوم در ادامه کار است.**

class RoutinePlan(*Base*):

    \_\_tablename\_\_ = "routine\_plans"

    id = Column(Integer, *primary\_key*=True, *index*=True)

    user\_id = Column(Integer, *nullable*=True)

    plan\_name = Column(String, *nullable*=False)

    created\_at = Column(DateTime, *default*=datetime.utcnow)

    steps = relationship("RoutineStep", *back\_populates*="routine", *cascade*="all, delete-orphan")

class RoutineStep(*Base*):

    \_\_tablename\_\_ = "routine\_steps"

    id = Column(Integer, *primary\_key*=True, *index*=True)

    routine\_id = Column(Integer, ForeignKey("routine\_plans.id"))

    step\_number = Column(Integer, *nullable*=False)

    description = Column(String, *nullable*=False)

    product\_id = Column(Integer, ForeignKey("Products.product\_id"), *nullable*=True)

    product\_name = Column(String)

    price = Column(Float, *nullable*=True)

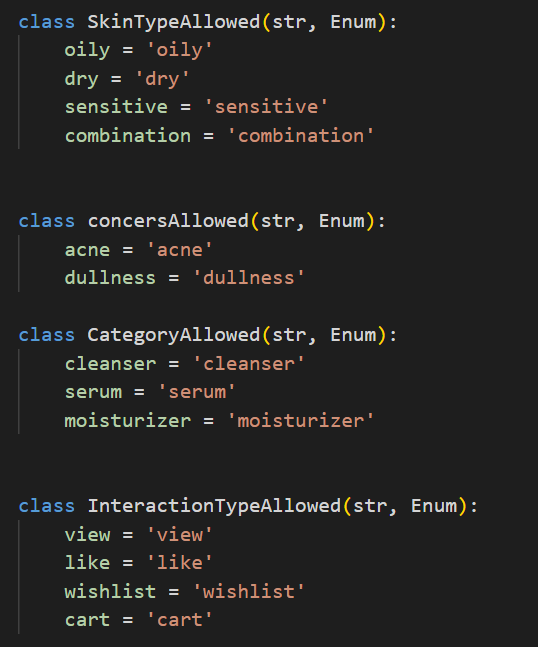
    routine = relationship("RoutinePlan", *back\_populates*="steps")

**جداول مرتبط با روتین که در آنها روتین ساخته شده برای کاربر ذخیره می گردد. شامل آیدی و گام های روتین و توضیحات هر گام و محصولات متناسب با آن قدم و نیز قیمت.**

***توضیحات درمورد schemas.py :***

**بخش کلاس های : Enum**

**در این بخش برای اینکه ورودی محدود به مقادیر خاص باشد از Enumاستفاده شده.**



**نوع پوست کاربر (oily, dry, sensitive, combination)**

**مشکلات پوستی (acne, dullness)**

**دسته‌بندی محصولات (cleanser, serum, moisturizer)**

**نوع تعامل کاربر با محصول (view, like, wishlist, cart)**

**Q1Optionsتا Q10Options: گزینه‌های پرسشنامه (Quiz) برای تعیین نوع پوست و عادت‌های کاربر**

**اگر کاربر مقدار غیرمجاز وارد کنه FastAPI خطا میدهد.**

**بخش مدل های کاربر :**

**UserCreate**

**داده‌های مورد نیاز برای ثبت‌نام کاربر**

**فیلدها: user\_name و password.**

**LoginRequest**

**داده‌های لازم برای ورود کاربر**

**فیلدها: user\_id و password.**

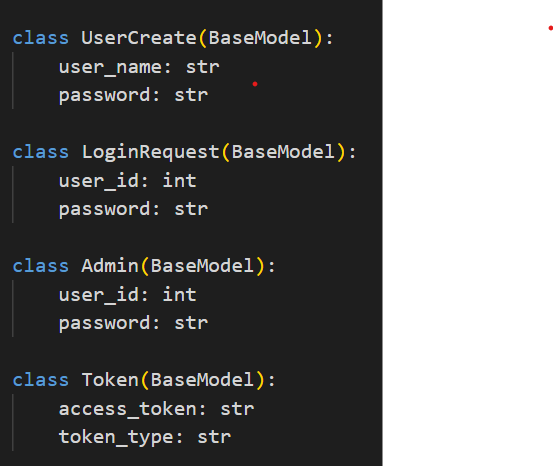
**Admin**

**ورود مدیر سیستم، مشابه LoginRequest.**

**Token**

**خروجی پس از احراز هویت**

**شامل access\_token و token\_type**

****

**بخش مدل های محصولات :**

**ProductCreate**

**مدل برای ثبت محصول جدید:**

**مشخصات پای name, brand, category**

**محدودیت پوستskin\_types**

**مشکلات هدفمند concerns\_targeted**

**ترکیبات ingredients**

**قیمت، موجودی و امتیاز price, rating, count**

**Product\_out1**

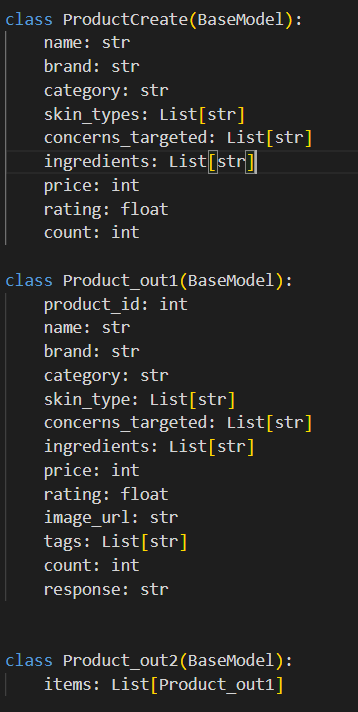
**خروجی API برای یک محصول:**

**شامل شناسه محصول product\_id**

**اطلاعات کامل محصول image\_url, tags, response**

**Product\_out2**

**خروجی API برای چند محصول به صورت لیستی (items)**

******

**بخش مدل های : History**

**BrowsingHistoryCreate**

**ثبت تعامل کاربر با محصول:**

**user\_id, product\_id, timestamp**

**interaction\_typeنوع تعامل (مشاهده، لایک و …)**

**PurchaseHistoryCreate**

**ثبت خرید کاربر:**

**user\_id, product\_id, quantity**

**timestamp زمان خرید، پیش‌فرض زمان فعلی**

**Purchase\_input**

**دریافت خرید کاربر (API input)**

**فیلدها product\_id و quantity**

**ContextualSignalCreate**

**سیگنال‌های زمینه‌ای برای پیشنهاد محصولات هوشمند:**

**user\_id, timestamp**

**بخش مدل های جستجو و کوییز:**

**Search :**

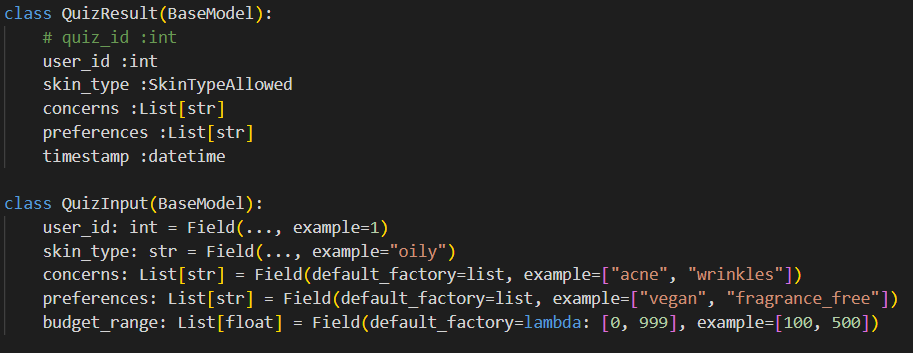
**دریافت متن از کاربر**

**فیلد : Search**

**QuizQuestions :**

**دریافت پاسخ‌های کاربر به کوییز  
فیلدها: user\_id و پاسخ‌های q1 تا q10 از Enumهای مربوط**

**QuizResult  
خروجی کوییز شامل نتایج تحلیل پوست و ترجیحات کاربر  
فیلدها user\_id, skin\_type, concerns, preferences, timestamp**

**QuizInput**  
**داده‌های اولیه کوییز برای پیشنهاد محصولات  
فیلدها skin\_type, concerns, preferences, budget\_range :   
  
که الان کلاس های QuizResult و Quizinput را مشاهده میکنید:  
  
**

**بخش مدل های روتین:**

**StepOut  
مدل یک مرحله از روتین  
فیلدها step\_name, product\_id, product\_name, instructions :**

**RoutineOut  
مدل خروجی کل روتین پوستی  
فیلدها Plan\_name :و step که شامل لیست StepOut است**

***توضیحات API های قرار داده شده در پروژه:***

**بخش** **fill database:**

**در این بخش با تابع add\_product\_to\_databases 1000 دیتاست و محصول رندوم ایجاد می‌شوند. انتخاب بخش های رندوم هر ستون به صورت زیر است:**

*name*=f'product {i}',

*brand*=random.choice(['iran', 'turkey', 'america', 'german', 'france']),

*category*=random.choice(['cleanser', 'serum',  'moisturizer']),

*skin\_types*= list(set([random.choice(['oily', 'dry', 'sensitive', 'combination']) for i in range(len\_of\_skintype)])),

*concerns\_targeted*=list(set([random.choice(['acne', 'combination', 'dullness']) for i in range(len\_of\_concens)])),

*ingredients* = ['string'],

*price*=random.randrange(1000, 10000),

*rating* = round(random.uniform(3.6, 5), 2),

*image\_url* = 'string',

*tags* = ['string'],

*count* = random.randint(10, 10000)

**بخش Account:**

**این دو بخش خود شامل دو زیر بخش است:**

* **زیربخش sign\_up:**

**در این زیر بخش با استفاده از تابع create\_user نام کاربری و گذرواژه را از کاربر می گیریم. با دستورات زیر گذرواژه را هش و رمزنگاری می کنیم:**

from passlib.context import CryptContext

pwd\_context = CryptContext(*schemes*=["bcrypt"], *deprecated*="auto")

user\_data["password"] = pwd\_context.hash(*user*.password)

**سپس با استفاده از دستورات زیر آنها را در جدول ذخیره می کنیم:**

*db*.add(new\_user)

*db*.commit()

*db*.refresh(new\_user)

**در ادامه user\_in\_code را که متغیر global هست، بر اساس آیدی آن نام کاربری مقدار دهی می کنیم.(به واسطه تابع find\_user\_id). همچنین با استفاده از با token گذاری و jwt سطح دسترسی کاربر را normal قرار می دهیم.**

* **زیربخش login:**

**با تابع review\_user و به واسطه تابع find\_user\_id و دستورات زیر نام کاربری را با نام موجود در دیتابیس تطابق داده:**

user\_review = *db*.query(models.Users\_for\_sign\_up).filter(models.Users\_for\_sign\_up.user\_name == *user*.user\_name).first()

    admin\_review = *db*.query(models.Admins).filter(models.Admins.user\_name == *user*.user\_name).first()

**و با چک کردن هش پسورد لاگین با هش پسورد ذخیره شده(کلا پسورد به صورت هش ذخیره می شود) گذواژه تطابق داده می شود. طبق دستورات زیر برای ادمین و کاربر عادی به صورت زیر خواهد بود:**

if user\_review and pwd\_context.verify(*user*.password, user\_review.password):

        user\_in\_code = user\_review.user\_id

        token = create\_access\_token({"sub": str(user\_in\_code), "role": "user"})

        return {"access\_token": token, "token\_type": "bearer"}

    if admin\_review and pwd\_context.verify(*user*.password, admin\_review.password):

        token = create\_access\_token({"sub": str(*user*.user\_id), "role": "admin"})

        return {"access\_token": token, "token\_type": "bearer"}

**بخش product :**

**این بخش به سه زیر بخش تقسیم می گردد:**

* **زیر بخش add\_product :**

**در این زیربخش با تابع product\_create توانایی افزودن محصول و ثبت آن در دیتابیس در تیبل products را داریم.**

* **زیر بخش deleting\_product:**

**این تابع با گرفتن product\_id در صورت وجود محصول مورد نظر را حذف می کند و پیام Product deleted successfully را داده وگرنه خطای 404 همراه با پیام not found! می دهد.**

* **زیر بخش all\_product:**

**تابع get\_all\_products بر این اساس که api از نوع get است و نه post ، بنابراین تمامی محصولات را نمایش می دهد. البته در اینجا از redis و caching نیز بهره برده شده است.**

**بخش shop :**

**شامل دو زیربخش است:**

* **زیربخش add\_to\_cart:**

**این تابع وظیفه افزودن محصولات به سبد خرید بر اساس product\_id و quantity را دارد. در صورت نبود همچنین کالایی خطای 404 و پیغام product not found! می دهد. همچنین اگر تعداد خواسته شده بیشتر از تعداد موجود در انبار باشد آنگاه نیز خطای 404 همراه با پیغام Not enough stock available می دهد. در صورت عدم مشکل به تیبل cart برای خرید اضافه می شود.**

* **زیربخش checkout:**

**این تابع نقش تایید نهایی برای خرید کالاها را دارد. خرید ها از تیبل cartخوانده شده و در صورت عدم وجود خطای 400 و cart is empty می خورند. در صورت وجود محصولات یکی یکی چک و خریده می شوند. و در صورت تمام شدن تعداد محصولی پس از این خرید ستون status آن false و آن محصول به اتمام می رسد. پس از اتمام خرید ظبق دستور زیر سابقه خرید محصولات برای شخص ثبت می شود و تیبل cart نیز از آن محصولات بدیهی است که خالی می شود:**

purchase = models.Purchase\_History(

*user\_id*=user\_in\_code,

*product\_id*=item.product\_id,

*quantity*=item.quantity

        )

*db*.add(purchase)

*db*.query(models.Cart).filter(models.Cart.user\_id == user\_in\_code).delete()

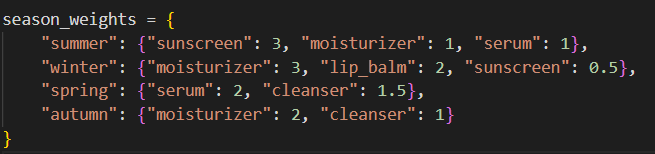
*db*.commit()

**بخش search:**

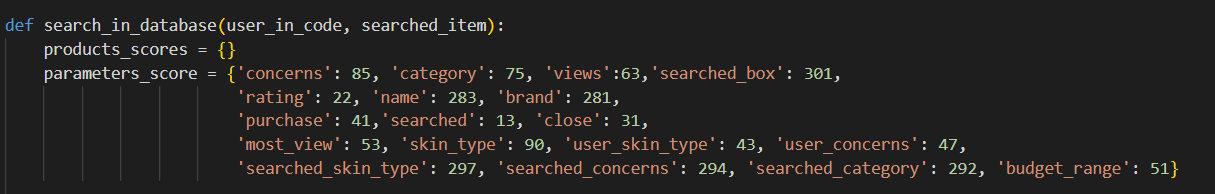
**این بخش سیستم جستجوی محصولات مراقبت پوستی را با توجه به ویژگی‌ها و ترجیحات کاربر پیشنهاد می‌دهد و نتایج را بر اساس امتیاز مرتب می‌کند**

**الگوریتم و نحوه عملکرد :**

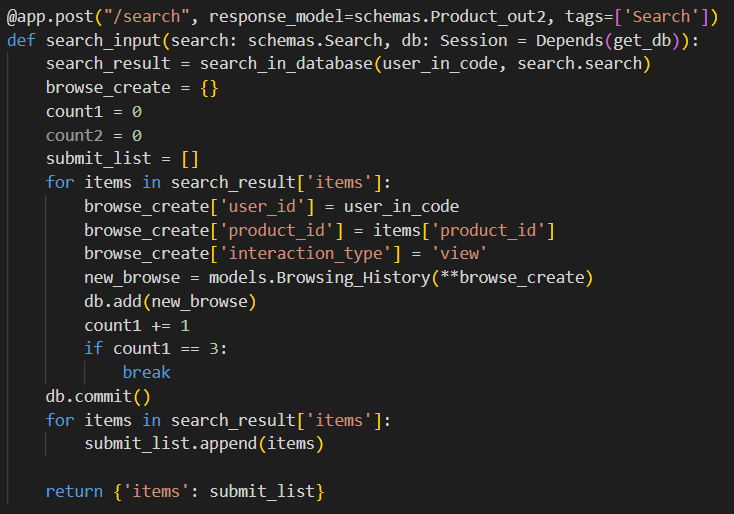
**تشخیص فصل جاری :  
تابع get\_season ماه فعلی را گرفته و فصل را تعیین می‌کند زمستان بهار تابستان پاییز  
این اطلاعات برای وزن‌ دهی محصولات متناسب با فصل استفاده می‌شود season\_weights**

****

**محاسبه امتیاز محصولات :  
یک دیکشنری parameters\_score تعریف شده که هر عامل موثر در امتیازدهی مثل مشاهده قبلی خرید نوع پوست نگرانی‌ها و فصل را وزن‌ دهی می‌کند  
همچنین reasons\_of\_parameters توضیح می‌دهد که هر امتیاز به چه دلیلی به محصول تعلق گرفته است**

****

**جستجوی محصول در پایگاه داده :  
ابتدا بررسی می‌شود که کاربر چیزی با شناسه محصول نام برند دسته‌بندی نوع پوست یا نگرانی‌ها جستجو کرده است  
محصولات مطابق با معیارهای جستجو به products\_scores اضافه می ‌شوند**

****

**این بخش مربوط به جستجوی محصولات در پایگاه داده است و با استفاده از مسیر /search پیاده‌سازی شده است. هدف این قسمت، دریافت یک عبارت جستجو از کاربر، پیدا کردن محصولات مرتبط در دیتابیس و ثبت سابقه‌ی مرور (Browsing History) برای تحلیل رفتار کاربر است.**

**تطبیق با مشخصات کاربر :  
با استفاده از توابع user\_information s\_product concerns\_targeted purchase\_da محصولاتی که با نوع پوست نگرانی‌ها یا خریدهای گذشته کاربر تطابق دارند امتیاز بیشتری می‌ گیرند**

**امتیازدهی اضافی بر اساس بازدیدها و بودجه :  
تعداد بازدید کاربران users\_views و بازدیدهای متداول به امتیاز محصول اضافه می ‌شود  
اگر قیمت محصول نزدیک به بودجه کاربر باشد امتیاز بیشتری دریافت می‌ کند**

**محاسبه امتیاز بر اساس فصل :  
محصولات متناسب با فصل جاری season\_weights افزایش امتیاز می‌ گیرند**

**مرتب‌سازی و انتخاب برترین محصولات :  
محصولات بر اساس مجموع امتیاز مرتب می‌شوند و ده محصول برتر انتخاب می ‌شوند  
هر محصول همچنین دلیل تعلق امتیاز به آن محصول در خروجی ذکر می‌ شود**

**ساخت خروجی نهایی :  
اطلاعات محصولات شامل شناسه نام برند دسته‌بندی نوع پوست نگرانی‌ها مواد تشکیل‌دهنده قیمت امتیاز تصویر و توضیح امتیاز در یک دیکشنری خروجی قرار می‌ گیرد**

**بخش admin:**

**این بخش مشابه ساخت normal user البته با سطح دسترسی admin می باشد که با استفاده از jwt و token که توکن آن در فایل token-utils.py ساخته و verify می گردد. بنابراین بخش هایی برای کاربر عادی حتی پس از لاگین نیز قفل هستند اما برای ادمین آن بخش ها آزادند مانند پاک کردن یا افزودن دیتا و محصول و... .**

**بخش quiz:**

**شامل دو زیربخش است:**

* **زیر بخش questions:**

**که در آن به واسطه تابع quiz\_questions و بدلیل اینکه از نوع get است سوالات کوییز را همراه با گزینه های پاسخ نمایش می دهد. این اطلاعات در فایل questions.py قرار داشته و در main.py دوباره import می شوند.**

* **زیربخش submit:**

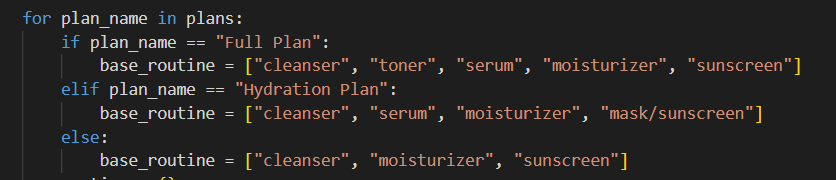
**در تابع submitting\_quiz ابتدای امر پاسخ سوالات و در صورت وجود تصویر سلفی فرد را می گیریم. در صورت عدم ارسال سلفی باید تیک send empty فایل برداشته شود. پس exception handling های مربوط به عکس در تابع به سراغ آنالیز کوییز می رویم . دیکشنری answers را که شامل پاسخ های کاربر است ، به analyze\_quiz که در فایل utils.py قرار دارد ارسال می کنیم و نتیجه را در quiz\_result می ریزیم. تابع analyze\_quiz بر اساس پاسخ های کوییز اطلاعات پیرامون skin\_type, concerns, preferences می یابد. و آنها را return می کند. حال اگر عکس سلفی فرستاده نشده باشد final\_result ما همان quiz\_result بوده و همان در تیبل FinalResult و Usersدخیره می شود وگرنه به تابع merge\_results جواب کوییز و سلفی را می دهیم و آن تابع skin\_type را بر اساس سلفی، concerns را بر اساس مجموع و اجتماع هر دو بخش و در نهایت preferences را بر اساس کوییز می سنجد و پاس می دهد و در نهایت final\_result ذخیره شده و در جداول مربوطه ثبت می گردد.**

**بخش routine:**

**این بخش مربوط است به ساخت روتین‌های مراقبت پوستی برای کاربر و با توجه به محصولات موجود بهترین ترکیب مراحل را برای هر برنامه روتین پیشنهاد می‌ دهد**

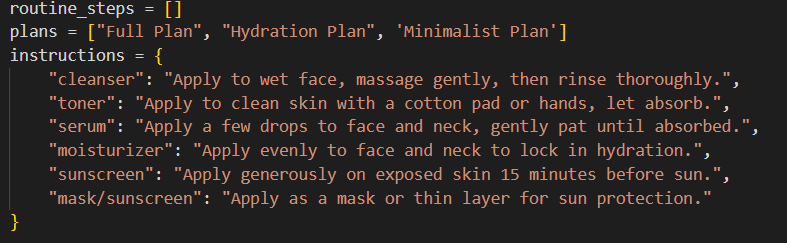
**برنامه‌های روتین :  
سه برنامه اصلی وجود دارد Full Plan Hydration Plan Minimalist Plan**

**تعیین مراحل هر برنامه :  
برای هر برنامه لیستی از مراحل پایه تعریف می‌شود مانند cleanser toner serum moisturizer sunscreen برای Full Plan و ترکیبات دیگر برای بقیه برنامه‌ها**

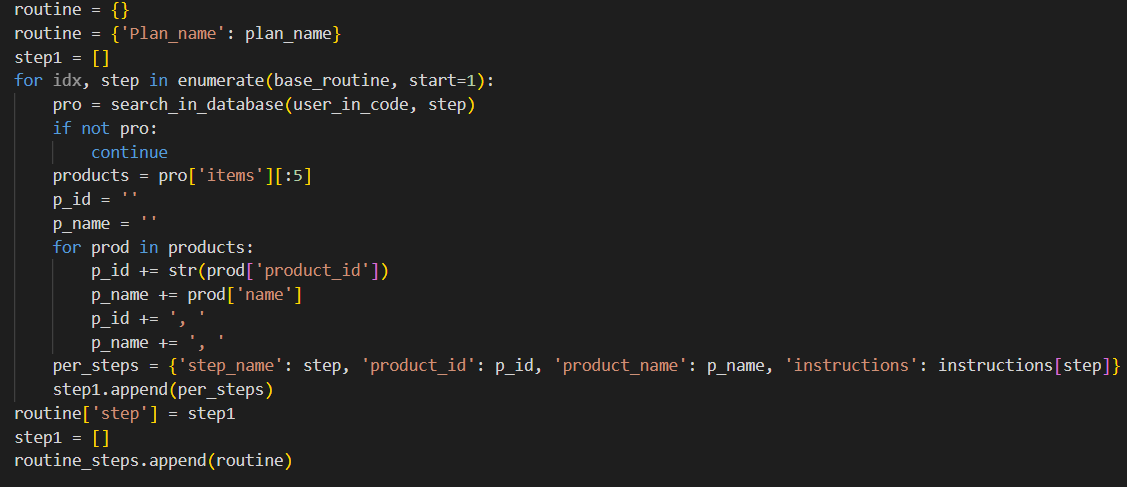
****

**جستجوی محصولات مناسب :  
برای هر مرحله با استفاده از تابع search\_in\_database محصولات مرتبط با نوع مرحله جستجو می ‌شوند  
تا پنج محصول اول برای هر مرحله انتخاب و اطلاعات آنها شامل شناسه و نام جمع‌آوری می‌ شود**

**ساخت مرحله روتین :  
برای هر مرحله یک دیکشنری شامل نام مرحله شناسه محصولات نام محصولات و دستورالعمل استفاده ساخته می‌ شود  
تمام مراحل هر برنامه در یک دیکشنری برنامه قرار می‌گیرد و به لیست نهایی اضافه می ‌شود  
در نهایت لیست کامل روتین‌ها شامل برنامه‌ها و مراحل مربوطه بازگردانده می‌ شود**

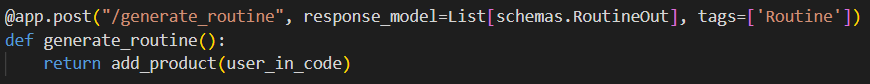
****

**طبق کد برای هر مرحله‌ی روتین (مثل پاک‌کننده یا سرم) محصولات مناسب از دیتابیس جستجو می‌شوند، سپس شناسه، نام و دستورالعمل استفاده آن‌ها ذخیره شده و در قالب یک روتین کامل به کاربر ارائه می‌شود.**

****

**generate\_routine:**

**این بخش وظیفه تولید برنامه مراقبت پوستی دارد. با درخواست POST به /generate\_routine، سه پلن اصلی (Full، Hydration و Minimalist) ساخته می‌شود که شامل مراحل مشخص، محصولات پیشنهادی و دستورالعمل استفاده هستند. خروجی در قالب مدل RoutineOut بازگردانده می‌شود.**

****

* در بخش frontend پروژه، صفحات وب با استفاده از HTML، CSS و JavaScript و تعاملات API با بکند ساخته شده است. هدف از طراحی این بخش ارائۀ تجربۀ کاربری مناسب و صمیمانه با مشتریان جهت خرید محصولات و استفاده از بخش‌های مختلف سایت است.

***ساختار frontend پروژه:***

به طور کلی ساختار این بخش شامل دو بخش static و templates می‌باشد. بخش اول شامل صفحات استایل (CSS) و اسکریپت‌های جاوااسکریپت (JavaScript) می‌باشد. بخش دوم نیز شامل صفحات HTML است که ساختارهای متفاوت سایت را تشکیل می‌دهند.

**بخش static:**

در این بخش استایل صفحات مختلف برای بخش‌های متفاوت سایت به آن‌ها داده شده است.

**بخش templates:**

* Base.html : قالب پایه که توسط سایر صفحات استفاده می‌شود. این قالب شامل اجزای عمومی سایت مثل هدر، فوتر و بخش‌های مشترک می‌باشد. مثل اکثر سایت‌های معروف که صفحات متفاوت آن‌ها قالب مشخصی دارند که کاربران آن سایت‌ها را با آن قالب می‌شناسند، ما نیز با هدف ایجاد ذهنیت در ذهن کاربران (مخصوصا افراد با هوش تصویری خوب) چنین بخشی را طراحی کردیم.
* Sign-up.html : صفحۀ ثبت‌نام که کاربران جدید می‌توانند از طریق این صفحه حساب کاربری بسازند.
* Login.html : صفحۀ ورود کاربران که امکان ورود به حساب کاربری را برای کاربران فراهم می‌کند.
* About.html : اطلاعاتی مربوط به سایت؛ داستان ساخت سایت و هدف آن و آشنایی با طراحان سایت، راه ارتباطی با آن‌ها (Github account و Phone number) و هم‌چنین ارائۀ دلایلی برای ترجیح‌دادن استفاده از این سایت درمیان سایت‌های مشابه.
* Shop.html : صفحۀ اصلی فروشگاه که شامل فهرست محصولات و اطلاعات کلی درمورد آن‌ها و هم‌چنین قابلیت جست‌وجو برای پیداکردن محصولات دلخواه می‌باشد. از طریق این صفحه می‌توان به صفحۀ اطلاعات جزئی محصولات نیز راه یافت.
* Shop-single.html : صفحه‌ای که اطلاعات یک محصول شامل اسم محصول، این‌که مناسب چه نوعی از پوست است (Skin Type)، دسته‌بندی محصول (Category)، عوارض (Concerns)، بخشی از بدن که کرم مربوط به آن است (Targeted)، میزان حجم کرم (Count) و وضعیت دردسترس بودن یا نبودن محصول (Status) می‌باشد. در این صفحه دکمه‌ای تحت عنوان Add to Cart با هدف اضافه‌کردن محصول موردنظر به سبد خرید نیز طراحی شده است.
* Quiz.html : این صفحه با هدف طراحی یک آزمون جهت شناسایی اطلاعات پوستی کاربر و ارائۀ روتین پوستی مناسب طراحی شده است.

***تعاملات backend و frontend:***

در صفحات مختلف این پروژه، بخش frontend به‌طور خاص به APIهای بکند تعامل دارد تا اطلاعات کاربر، محصولات، سفارشات و کوییزها را مدیریت کند.

* صفحۀ ثبت‌نام و ورود: در صفحۀ ثبت‌نام، کاربر اطلاعات خود را وارد کرده و پس از ثبت‌نام داده‌ها به API ارسال شده و در دیتابیس ذخیره می‌شود. در صفحۀ ورود نیز کاربر اطلاعات خود را وارد کرده و با صحت‌سنجی درصورت درستی اطلاعات، کاربر به صفحۀ اصلی سایت هدایت می‌شود.
* صفحۀ کوییز: این صفحه برای دریافت اطلاعات پوستی کاربران طراحی شده است. اطلاعات و نتیجۀ کوییزها جمع‌آوری شده و به API ارسال می‌شود.
* صفحۀ محصولات: در این صفحه جست‌وجوهای کاربر به API ارسال می‌شود و سپس نتایج برای کاربر نمایش داده می‌شوند.

***نحوۀ تعامل با API ها و نکات کلیدی:***

* توکن JWT: همۀ تعاملات مهم مثل ورود، ثبت‌نام، خرید، جست‌وجو و... با استفاده از این توکن مدیریت می‌شود.
* جست‌وجو و پیشنهادات: این سیستم براساس داده‌های پوستی کاربر و سابقۀ خرید طراحی شده است. این تعاملات به API ارسال می‌شود و نتیجه در صفحه بارگذاری می‌شود.
* ثبت محصولات به سبد خرید: اضافه‌کردن محصولات به سبد خرید از طریق Cart.js و ارتباط به API مربوط به backend انجام می‌شود. هنگامی که خرید تأیید شود، اطلاعات به پایگاه داده ارسال و ذخیره می‌شوند.

***نکات مهم طراحی و تجربۀ کاربری:***

* Responsiveness : صفحات سایت به‌گونه‌ای طراحی شده اند که تجربۀ کاربری مناسبی در دستگاه‌های مختلف مثل تلفن همراه و رایانه داشته باشند.
* کاربری ساده: صفحات سایت بدون هیچ‌گونه بخش اضافی که باعث سردرگمی کاربر شود طراحی شده است. در تمامی صفحات می‌توان به راحتی به صفحات دیگر منتقل شد و از آن‌ها استفاده کرد.
* طراحی زیبا: استفاده از استایل‌های CSS برای جذابیت در طراحی ساخته شده‌اند و هماهنگی‌های این صفحات باعث ایجاد زیبایی در صفحات شده است.