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 appت 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 127.0.0.1:8000 /docsمی 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.pyمطابق 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 APIاماده AILABمی 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 Enum استفاده شده.

```python
class SkinTypeAllowed(str, Enum):
    oily = 'oily'
    dry = 'dry'
    sensitive = 'sensitive'
    combination = 'combination'


class concersAllowed(str, Enum):
    acne = 'acne'
    dullness = 'dullness'

class CategoryAllowed(str, Enum):
    cleanser = 'cleanser'
    serum = 'serum'
    moisturizer = 'moisturizer'


class InteractionTypeAllowed(str, Enum):
    view = 'view'
    like = 'like'
    wishlist = 'wishlist'
    cart = 'cart'
```

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)

Q1Options تا 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

```python
class UserCreate(BaseModel):
    user_name: str
    password: str

class LoginRequest(BaseModel):
    user_id: int
    password: str

class Admin(BaseModel):
    user_id: int
    password: str

class Token(BaseModel):
    access_token: str
    token_type: str
```

بخش مدل های محصولات :

ProductCreate

مدل برای ثبت محصول جدید:

مشخصات پای name, brand, category

محدودیت پوستskin_types

مشکلات هدفمندconcerns_targeted

ترکیباتingredients

قیمت، موجودی و امتیازprice, rating, count

Product_out1

خروجی API برای یک محصول:

شامل شناسه محصول product_id

اطلاعات کامل محصولimage_url, tags, response

Product_out2

خروجی API برای چند محصول به صورت لیستی(items)

```python
class ProductCreate(BaseModel):
    name: str
    brand: str
    category: str
    skin_types: List[str]
    concerns_targeted: List[str]
    ingredients: List[str]
    price: int
    rating: float
    count: int

class Product_out1(BaseModel):
    product_id: int
    name: str
    brand: str
    category: str
    skin_type: List[str]
    concerns_targeted: List[str]
    ingredients: List[str]
    price: int
    rating: float
    image_url: str
    tags: List[str]
    count: int
    response: str


class Product_out2(BaseModel):
    items: List[Product_out1]
```

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 ازEnum های مربوط

QuizResult
خروجی کوییز شامل نتایج تحلیل پوست و ترجیحات کاربر
فیلدهاuser_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:

```
class QuizResult(BaseModel):
    # quiz_id :int
    user_id :int
    skin_type :SkinTypeAllowed
    concerns :List[str]
    preferences :List[str]
    timestamp :datetime

class QuizInput(BaseModel):
    user_id: int = Field(..., example=1)
    skin_type: str = Field(..., example="oily")
    concerns: List[str] = Field(default_factory=list, example=["acne", "wrinkles"])
    preferences: List[str] = Field(default_factory=list, example=["vegan", "fragrance_free"])
    budget_range: List[float] = Field(default_factory=lambda: [0, 999], example=[100, 500])
```

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 globalهست، بر اساس آیدی آن نام کاربری مقدار دهی می کنیم.(به واسطه تابع find_user_id). همچنین با استفاده از با tokenگذاری و jwtسطح 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 cart خوانده 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 طبق 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

```
season_weights = {
    "summer": {"sunscreen": 3, "moisturizer": 1, "serum": 1},
    "winter": {"moisturizer": 3, "lip_balm": 2, "sunscreen": 0.5},
    "spring": {"serum": 2, "cleanser": 1.5},
    "autumn": {"moisturizer": 2, "cleanser": 1}
}
```

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

```
def search_in_database(user_in_code, searched_item):
    products_scores = {}
    parameters_score = {'concerns': 85, 'category': 75, 'views':63,'searched_box': 301,
                        'rating': 22, 'name': 283, 'brand': 281,
                        'purchase': 41,'searched': 13, 'close': 31,
                        'most_view': 53, 'skin_type': 90, 'user_skin_type': 43, 'user_concerns': 47,
                        'searched_skin_type': 297, 'searched_concerns': 294, 'searched_category': 292, 'budget_range': 51}
```

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

```python
@app.post("/search", response_model=schemas.Product_out2, tags=['Search'])
def search_input(search: schemas.Search, db: Session = Depends(get_db)):
    search_result = search_in_database(user_in_code, search.search)
    browse_create = {}
    count1 = 0
    count2 = 0
    submit_list = []
    for items in search_result['items']:
        browse_create['user_id'] = user_in_code
        browse_create['product_id'] = items['product_id']
        browse_create['interaction_type'] = 'view'
        new_browse = models.Browsing_History(**browse_create)
        db.add(new_browse)
        count1 += 1
        if count1 == 3:
            break
    db.commit()
    for items in search_result['items']:
        submit_list.append(items)

    return {'items': submit_list}
```

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 adminمی باشد که با استفاده از jwt و tokenکه توکن آن در فایل token-utils.py ساخته و verifyمی 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 getاست سوالات کوییز را همراه با گزینه های پاسخ نمایش می دهد. این اطلاعات در فایل questions.py قرار داشته و در main.py دوباره importمی 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 returnمی کند. حال اگر عکس سلفی فرستاده نشده باشد final_resultما همان quiz_result بوده و همان در تیبل FinalResult و Usersذخیره 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 داردFull 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 Planو ترکیبات دیگر برای بقیه برنامه‌ها

```
for plan_name in plans:
    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"]
```

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

```
routine_steps = []
plans = ["Full Plan", "Hydration Plan", 'Minimalist Plan']
instructions = {
    "cleanser": "Apply to wet face, massage gently, then rinse thoroughly.",
    "toner": "Apply to clean skin with a cotton pad or hands, let absorb.",
    "serum": "Apply a few drops to face and neck, gently pat until absorbed.",
    "moisturizer": "Apply evenly to face and neck to lock in hydration.",
    "sunscreen": "Apply generously on exposed skin 15 minutes before sun.",
    "mask/sunscreen": "Apply as a mask or thin layer for sun protection."
}
```

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.

```
routine = {}
routine = {'Plan_name': plan_name}
step1 = []
for idx, step in enumerate(base_routine, start=1):
    pro = search_in_database(user_in_code, step)
    if not pro:
        continue
    products = pro['items'][:5]
    p_id = ''
    p_name = ''
    for prod in products:
        p_id += str(prod['product_id'])
        p_name += prod['name']
        p_id += ', '
        p_name += ', '
    per_steps = {'step_name': step, 'product_id': p_id, 'product_name': p_name, 'instructions': instructions[step]}
    step1.append(per_steps)
routine['step'] = step1
step1 = []
routine_steps.append(routine)
```

generate_routine:

این بخش وظیفه تولید برنامه مراقبت پوستی دارد. با درخواست POST به /generate_routine، سه پلن اصلی(Full) ، Hydrationو (Minimalist) ساخته می‌شود که شامل مراحل مشخص، محصولات پیشنهادی و دستورالعمل استفاده هستند. خروجی در قالب مدل RoutineOutبازگردانده می‌شود.

```python
@app.post("/generate_routine", response_model=List[schemas.RoutineOut], tags=['Routine'])
def generate_routine():
    return add_product(user_in_code)
```

- 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 ارائهٔ تجربهٔ 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 : صفحهٔ ثبت‌نام که کاربران جدید می‌توانند از طریق این صفحه حساب کاربری بسازند.
- Login.html : صفحهٔ 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 ارائهٔ دلایلی برای ترجیح‌دادن استفاده از این سایت درمیان سایت‌های مشابه.
- Shop.html : صفحهٔ 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 صفحهٔ 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 ارائهٔ 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.

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

نحوهٔ تعامل با API ها و نکات کلیدی:

- توکن JWT: همهٔ 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 سابقهٔ 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 تجربهٔ کاربری:

- Responsiveness : صفحات سایت به‌گونه‌ای طراحی شده اند که تجربهٔ 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.