

بنام خدا



پروژه درس پایگاه های داده پیشرفته

(تحلیل داده های گوگل پلی استور)

تهییه و تنظیم:

علیرضا قلیزاده

استاد ارجمند :

دکتر نعمت بخش

۱- جمع آوری و آماده سازی داده ها	۳
۲- ایجاد پایگاه داده	۸
۲-۱ ایجاد پایگاه داده GOOGLEPLAYDATA	۸
۲-۲ ایجاد جداول	۹
۲-۳ ایجاد ایندکس ها برای بهینه سازی جستجوها	۱۱
۳- بارگذاری داده ها در جداول	۱۵
۴- توسعه بک اند (عملیات CRUD)	۱۹
۵- توسعه فرانت اند (توسعه داشبورد با STREAMLIT)	۳۰
۵-۱ پیکربندی STREAMLIT	۳۰
۵-۲ تابع بارگذاری داده ها	۳۰
۵-۳ بارگذاری داده ها و نمایش زمان اجرا	۳۲
۵-۴ فیلتر های صفحه کناری (SIDE BAR FILTERS)	۳۲
۵-۵ فیلتر کردن DATAFRAME	۳۳
۶- پردازش داده های تاریخ	۳۴
۷- مصور سازی داده ها (CHARTS)	۳۴
۸- بخش عملیات CRUD (از طریق API)	۳۶
۶- تصاویر اپلیکیشن	۴۲
۷- لینک های ویدیویی دمو و گیت هاب	۴۴
۸- منابع	۴۴

۱. جمع آوری و آماده سازی داده‌ها

پس از دانلود دیتاست از سایت Kaggle ، داده‌ها را پاکسازی می‌کنیم و همچنین داده‌های تکراری و نامعتبر را حذف می‌کنیم و فرمت داده‌ها را استاندارد سازی می‌کنیم.

✓ توضیح فایل `clean_data.py`

۱. بارگذاری داده‌ها از فایل CSV

```
df = pd.read_csv('Google-Playstore.csv')
```

در این مرحله از کتابخانه pandas، دیتاست اصلی که در فایل CSV قرار دارد (با نام "Google-Playstore.csv") بارگذاری می‌شود. این عملیات داده‌ها را به صورت یک DataFrame (جدول داده‌ای) در اختیار ما قرار می‌دهد تا بتوانیم مراحل بعدی پاکسازی و پردازش را روی آن انجام دهیم.

۲. نمایش داده‌های اولیه و اطلاعات دیتاست

```
print("Initial Data:")
print(df.head())
print("\nDataset Info:")
print(df.info())
```

df.head(): پنج سطر اول دیتاست نمایش داده می‌شود تا نگاهی کلی به ساختار و محتوای داده‌ها داشته باشیم.

df.info(): اطلاعاتی مانند تعداد ردیف‌ها، نام ستون‌ها، نوع داده‌ها و تعداد مقادیر غیر تهی (non-null) هر ستون را نمایش می‌دهد. این اطلاعات به ما کمک می‌کند تا از وضعیت اولیه دیتاست مطلع شویم.

۳. حذف رکوردهای تکراری

```
df.drop_duplicates(inplace=True)
print("\nNumber of records after dropping duplicates:", df.shape[0])
```

در این مرحله، با استفاده از تابع `drop_duplicates()` رکوردهای تکراری (duplicate rows) از دیتاست حذف می‌شوند. این کار باعث می‌شود داده‌های مانند ما یکتا و بدون تکرار باقی بمانند که به دقت و صحت تحلیل‌های بعدی کمک می‌کند.

۴. بررسی و حذف مقادیر گمشده

```
print("\nMissing values per column:")
print(df.isnull().sum())
df.dropna(inplace=True)
print("\nNumber of records after dropping rows with missing values:", df.shape[0])
```

ابتدا با استفاده از `df.isnull().sum()` تعداد مقادیر گمشده (NaN) در هر ستون چاپ می‌شود. این بررسی به ما کمک می‌کند تا از وجود داده‌های ناقص مطلع شویم. سپس با دستور `df.dropna(inplace=True)` تمامی ردیف‌هایی که دارای هر نوع مقدار گمشده هستند حذف می‌شوند. (در برخی پروژه‌ها ممکن است بخواهید به جای حذف، داده‌های گمشده را با مقدار میانگین یا میانه جایگزین کنید، اما در اینجا به سادگی ردیف‌های ناقص حذف شده‌اند).

۵. پاکسازی ستون "Installs"

```
df['Installs'] = df['Installs'].str.replace('[+,]', '', regex=True).astype(int)
print("\nSample 'Installs' values after cleaning:")
print(df['Installs'].head())
```

ستون Installs معمولاً به صورت رشته (string) ذخیره شده و شامل علامت‌های کاما (,) و علامت جمع (+) می‌باشد.

با استفاده از متده است `str.replace('[+,]', '', regex=True)` تمامی کاراکترهای غیر عددی (کاما و +) از مقدار ستون حذف می‌شوند.

سپس با (integer) `astype(int)` این ستون به نوع عدد صحیح تبدیل می‌شود. در نهایت، چند نمونه از داده‌های پاکسازی شده چاپ می‌شود تا صحت عملیات بررسی گردد.

۶. پاکسازی ستون "Price"

```
if isinstance(df['Price'].iloc[0], str):
    df['Price'] = df['Price'].str.replace('$', '', regex=True).astype(float)
else:
    print("\n'Price' column is already numeric. Skipping string cleaning for Price.")
print("\nSample 'Price' values after cleaning (if applicable):")
print(df['Price'].head())
```

ابتدا بررسی می‌شود که آیا ستون Price به صورت رشته (string) است یا خیر. اگر به صورت رشته است، با استفاده از متده است `str.replace('$', '', regex=True)` علامت دلار (\$) از مقادیر حذف شده و سپس ستون به نوع عددی (float) تبدیل می‌شود.

در غیر این صورت، پیغامی چاپ می‌شود که نشان می‌دهد ستون قیمت قبلاً به صورت عددی (numeric) است و نیازی به پاکسازی رشته‌ای ندارد.

چند نمونه از مقادیر پاکسازی شده چاپ می‌شود.

۷. پاکسازی ستون "Size"

```
def convert_size(size):
    """
    Convert a size string to megabytes (MB).
    - If the size is "Varies with device", return NaN.
    - If the size ends with 'M', remove the 'M' and convert to float.
    - If the size ends with 'k', remove the 'k', convert to float, and convert kilobytes to megabytes.
    """
    if size == "Varies with device":
        return np.nan
    if size[-1] == 'M':
        try:
            return float(size[:-1])
        except:
            return np.nan
    if size[-1] == 'k':
        try:
            return float(size[:-1]) / 1024 # Convert kB to MB
        except:
            return np.nan
    return np.nan

df['Size'] = df['Size'].apply(convert_size)
print("\nSample 'Size' values after conversion:")
print(df['Size'].head())
df['Size'].fillna(df['Size'].median(), inplace=True)
```

یک تابع `convert_size` تعریف شده که وظیفه تبدیل مقادیر ستون `Size` به مگابایت (MB) را بر عهده دارد:

اگر مقدار "Varies with device" باشد، مقدار NaN برگردانده می‌شود.

اگر مقدار به صورت 19M باشد، حرف "M" حذف شده و مقدار عددی باقیمانده به عنوان مگابایت در نظر گرفته می‌شود.

اگر مقدار به صورت 14k باشد، حرف "k" حذف شده و مقدار عددی به مگابایت (تقسیم بر ۱۰۲۴) تبدیل می‌شود.

سپس تابع `apply` روی ستون `Size` اعمال می‌شود.

چند نمونه از مقادیر تبدیل شده چاپ می‌شود.

در نهایت، مقادیر گمشده در این ستون با میانه (`median`) آن ستون جایگزین می‌شوند.

۸. اعتبارسنجی ستون "Rating"

```
df = df[(df['Rating'] >= 0) & (df['Rating'] <= 5)]
print("\nNumber of records after filtering invalid ratings:", df.shape[0])
```

در این مرحله، رکوردهایی که مقدار ستون Rating خارج از بازه ۰ تا ۵ دارند، حذف می‌شوند. این اعتبارسنجی اطمینان حاصل می‌کند که تنها رکوردهای معتبر باقی بمانند.

۹. نمایش خلاصه نهایی داده‌های پاکسازی شده و ذخیره دیتاست پاکسازی شده در فایل CSV

جدید

```
print("\nCleaned Data Sample:")
print(df.head())
print("\nFinal Dataset Info:")
print(df.info())
print("\nDescriptive Statistics:")
print(df.describe())

df.to_csv('cleaned_googleplaystore.csv', index=False)
print("\nCleaned dataset saved to 'cleaned_googleplaystore.csv'.")
```

در این بخش، چند سطر اولیه از دیتاست پاکسازی شده نمایش داده می‌شود. سپس اطلاعات کلی دیتاست (مانند تعداد رکوردها، نوع داده‌ها و تعداد مقادیر تهی) و آمار توصیفی (مانند میانگین، انحراف معیار، کمینه و بیشینه برای هر ستون چاپ Descriptive Statistics) می‌شود.

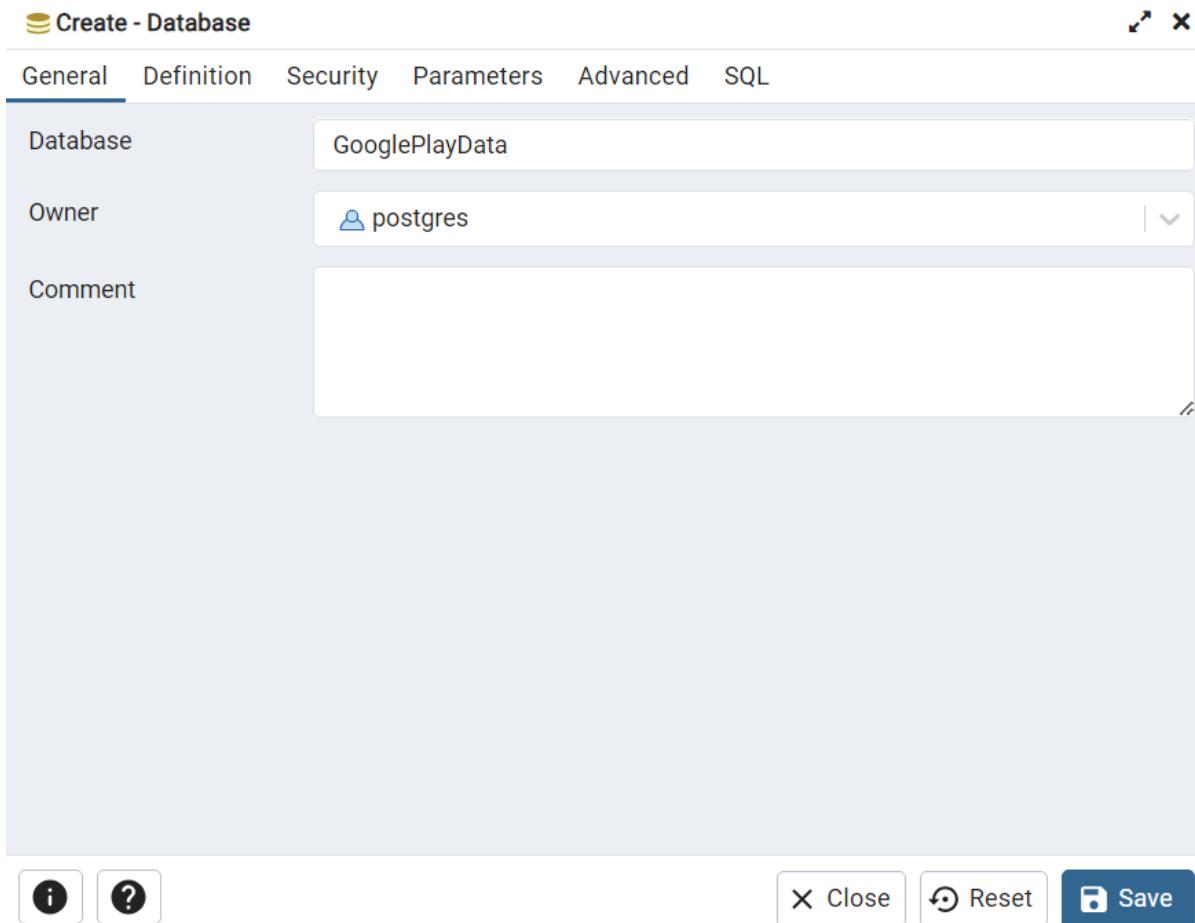
این بررسی‌ها کمک می‌کنند تا از صحت عملیات پاکسازی اطمینان حاصل کنیم. دیتاست پاکسازی شده در یک فایل CSV جدید به نام cleaned_googleplaystore.csv ذخیره می‌شود.

استفاده از index=False تضمین می‌کند که اندیس‌های DataFrame در فایل ذخیره نشوند. این فایل برای استفاده‌های بعدی (تحلیل‌های آماری، مدل‌سازی، و ...) در دسترس خواهد بود.

۲. ایجاد پایگاه داده

پس از پاکسازی داده ها، پایگاه داده خودمون رو ایجاد میکنیم. اسکیمای PostgreSQL رو طراحی کرده و جداول اصلی (developers و categories و applications) رو ایجاد میکنیم و در نهایت برای بهینه سازی جستجو ایندکس گذاری میکنیم.

۱. ایجاد پایگاه داده GooglePlayData



۲. ایجاد جداول

❖ جدول دسته بندی ها categories

```
CREATE TABLE IF NOT EXISTS categories (
    category VARCHAR(100) PRIMARY KEY -- Category name as primary key (unique)
);
```

هدف:

این جدول فقط شامل یک ستون به نام category است که نام دسته بندی ها را نگهداری می کند.

ویژگی ها:

VARCHAR(100): تعیین می کند که نام دسته بندی تا ۱۰۰ کاراکتر می تواند باشد.
PRIMARY KEY: تضمین می کند که هر دسته بندی یکتا باشد و مقدار تکراری ثبت نشود.

❖ جدول توسعه دهنده ها (developers)

```
CREATE TABLE IF NOT EXISTS developers (
    developer_id VARCHAR(255) PRIMARY KEY, -- Developer identifier (assumed unique)
    developer_website VARCHAR(255), -- Developer website URL
    developer_email VARCHAR(255) -- Developer email address
);
```

هدف:

این جدول اطلاعات مربوط به توسعه دهنده ها (مانند شناسه توسعه دهنده، وب سایت و ایمیل) را ذخیره می کند.

ویژگی ها:

developer_id: به عنوان کلید اصلی تعریف شده و باید یکتا باشد.
اندازه ستون ها (۲۵۵ کاراکتر) برای اطمینان از پذیرش مقادیر طولانی تر استفاده شده است.

❖ جدول اپلیکیشن ها (applications)

```
CREATE TABLE IF NOT EXISTS applications (
    app_id VARCHAR(2000) PRIMARY KEY,          -- Application identifier
    app_name TEXT,                            -- Application name
    category VARCHAR(100),                   -- Category name (foreign key to categories.category)
    rating FLOAT,                            -- App rating (e.g., 4.5)
    rating_count BIGINT,                     -- Number of ratings
    installs BIGINT,                         -- Number of installs
    minimum_installs BIGINT,                -- Minimum installs value
    maximum_installs BIGINT,                -- Maximum installs value
    free BOOLEAN,                           -- Indicates if the app is free
    price FLOAT,                            -- App price (if not free)
    currency VARCHAR(50),                  -- Currency code (e.g., USD)
    size FLOAT,                            -- App size in MB
    minimum_android VARCHAR(50),           -- Minimum required Android version
    developer_id VARCHAR(255),              -- Developer identifier (foreign key to developers.developer_id)
    released DATE,                          -- Release date of the app
    last_updated DATE,                     -- Last updated date
    content_rating VARCHAR(50),            -- Content rating (e.g., "Everyone", "Teen")
    privacy_policy TEXT,                  -- URL of the privacy policy
    ad_supported BOOLEAN,                 -- Indicates if the app is ad-supported
    in_app_purchases BOOLEAN,             -- Indicates if the app offers in-app purchases
    editors_choice BOOLEAN,               -- Indicates if the app is an editor's choice
    scraped_time TIMESTAMP,              -- Time when the data was scraped
    CONSTRAINT fk_category FOREIGN KEY (category) REFERENCES categories(category),
    CONSTRAINT fk_developer FOREIGN KEY (developer_id) REFERENCES developers(developer_id)
);
```

هدف:

این جدول شامل تمامی اطلاعات مربوط به اپلیکیشن‌ها است. ستون‌های مختلفی از جمله شناسه اپلیکیشن، نام، دسته‌بندی، امتیاز، تعداد امتیازها، تعداد نصب‌ها، حداقل و حداکثر نصب، وضعیت رایگان بودن، قیمت، واحد پول، اندازه، نسخه اندروید، شناسه توسعه‌دهنده، تاریخ انتشار، تاریخ آخرین به‌روزرسانی، درجه‌بندی محتوا، آدرس حریم خصوصی، وضعیت پشتیبانی از تبلیغات، خریدهای درون برنامه‌ای، انتخاب سردبیر و زمان جمع‌آوری داده‌ها وجود دارد.

ویژگی‌های مهم:

:FOREIGN KEY

ستون **category** به جدول **categories** متصل است (با استفاده از کلید خارجی).
ستون **developer_id** به جدول **developers** متصل است.

نوع داده‌ها:

برای مثال، ستون **app_id** با **VARCHAR(2000)** تعریف شده تا بتواند مقادیر طولانی را ذخیره کند.

ستون **app_name** از نوع **TEXT** است تا محدودیتی در اندازه نداشته باشد.

ستون‌های عددی مانند rating_count, installs, minimum_installs و maximum_installs از نوع BIGINT استفاده می‌شوند.

۳. ایجاد ایندکس‌ها برای بهینه‌سازی جستجوها

ایндکس‌ها به پایگاه داده کمک می‌کنند تا بدون اسکن کل جدول (full table scan) داده‌های مرتبط را سریع‌تر پیدا کند. با استفاده از covering indexes (با INCLUDE) می‌توانیم ستون‌های اضافی را به ایندکس اضافه کنیم تا پرس‌وچهایی که نیاز به آن‌ها دارند به‌طور کامل از ایندکس پاسخ داده شوند.

❖ ایندکس روی ستون category :

```
CREATE INDEX IF NOT EXISTS idx_applications_category  
ON applications(category);
```

هدف:

این ایندکس سرعت جستجو و فیلتر کردن بر اساس دسته‌بندی (category) را افزایش می‌دهد.

❖ ایندکس ترکیبی روی ستون‌های category و free به همراه INCLUDE :

```
CREATE INDEX IF NOT EXISTS idx_applications_category_free  
ON applications(category, free)  
INCLUDE (rating, app_name, price, installs);
```

هدف:

این ایندکس ترکیبی برای فیلتر کردن سریع اپلیکیشن‌های رایگان یا پرداختی در یک دسته‌بندی خاص طراحی شده است.

INCLUDE clause

ستون‌های اضافی مانند rating, app_name, price, installs در ایندکس گنجانده شده‌اند تا پرس‌وچهایی که از این ستون‌ها استفاده می‌کنند بتوانند بدون مراجعه به جدول اصلی (covering index) اجرا شوند. این امر باعث بهبود عملکرد می‌شود.

❖ ایندکس روی ستون **:rating**

```
CREATE INDEX IF NOT EXISTS idx_applications_rating  
ON applications(rating);
```

هدف:

این ایندکس برای تسریع فیلتر یا مرتب‌سازی بر اساس امتیاز (rating) اپلیکیشن‌ها استفاده می‌شود.

❖ ایندکس روی ستون **:content_rating**

```
CREATE INDEX IF NOT EXISTS idx_applications_content_rating  
ON applications(content_rating);
```

هدف:

این ایندکس برای بهبود سرعت فیلتر کردن بر اساس درجه‌بندی محتوا (content_rating) مورد استفاده قرار می‌گیرد.

❖ ایندکس روی ستون **:app_name**

```
CREATE INDEX IF NOT EXISTS idx_applications_app_name  
ON applications(app_name);
```

هدف:

این ایندکس عملکرد جستجوهای متنی بر روی نام اپلیکیشن‌ها را افزایش می‌دهد.

❖ ایندکس روی ستون **:released**

```
CREATE INDEX IF NOT EXISTS idx_applications_released  
ON applications(released);
```

هدف:

این ایندکس سرعت فیلتر کردن و گروهبندی اپلیکیشن‌ها بر اساس تاریخ انتشار را بهبود می‌بخشد.

❖ ایندکس روی ستون **:last_updated**

```
CREATE INDEX IF NOT EXISTS idx_applications_last_updated  
ON applications(last_updated);
```

هدف:

این ایندکس برای تسريع فیلتر کردن یا گروهبندی بر اساس تاریخ آخرین بهروزرسانی استفاده می‌شود.

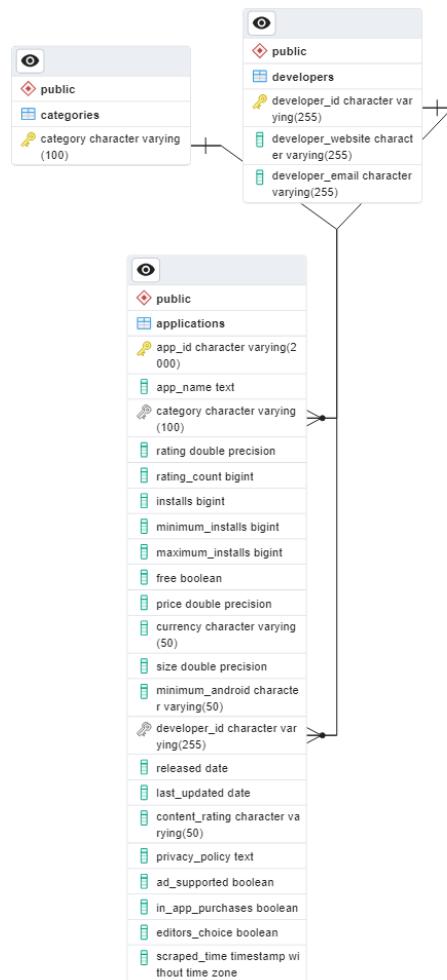
❖ ایندکس روی ستون **:developer_id**

```
CREATE INDEX IF NOT EXISTS idx_applications_developer_id  
ON applications(developer_id);
```

هدف:

این ایندکس عملکرد عملیات join (اتصال) بین جدول **applications** و **developers** را بهبود می‌بخشد.

❖ جداول ERD :



۳. بارگذاری داده‌ها در جداول (توضیح فایل load_data.py)

این اسکریپت پایتون برای پاکسازی، آماده‌سازی و وارد کردن داده‌های دیتابیس اپلیکیشن‌های گوگل پلی (Google Playstore) استفاده می‌شود. در این کد از کتابخانه‌های pandas برای پردازش داده و SQLAlchemy برای اتصال و انتقال داده‌ها به پایگاه داده PostgreSQL استفاده شده است. هدف از این کد، آماده‌سازی داده‌ها برای تحلیل‌های بعدی و ذخیره‌سازی آن‌ها در جداول مجزا (categories, applications, developers) می‌باشد.

۱. ایجاد اتصال به پایگاه داده با استفاده از SQLAlchemy

```
import pandas as pd
from sqlalchemy import create_engine

# 1. Create a SQLAlchemy engine to connect to the PostgreSQL database.
# Replace 'username', 'password', 'localhost', '5432' and 'GooglePlayData'
# with your PostgreSQL credentials and database details.
engine = create_engine('postgresql://Alireza1679@127.0.0.1:5432/GooglePlayData')
```

ابتدا کتابخانه‌های مورد نیاز (sqlalchemy و pandas) وارد شده‌اند. با استفاده از تابع create_engine یک اتصال (engine) به پایگاه داده PostgreSQL ایجاد می‌شود. در رشته اتصال (connection string) اطلاعات مربوط به نام کاربری (postgres)، رمز عبور (GooglePlayData)، آدرس سرور (127.0.0.1)، پورت (5432) و نام پایگاه داده (Alireza1679) درج شده است.

این engine به ما اجازه می‌دهد تا از طریق SQLAlchemy عملیات خواندن و نوشتن داده‌ها را در پایگاه داده انجام دهیم.

۲. بارگذاری داده‌ها از فایل CSV پاکسازی شده

```
# 2. Load the cleaned CSV file into a Pandas DataFrame.
df = pd.read_csv('cleaned_googleplaystore.csv')
```

فایل CSV توسط تابع pd.read_csv() خوانده می‌شود. داده‌ها به صورت یک DataFrame در متغیر df ذخیره می‌شوند تا بتوانیم بر روی آن‌ها پردازش‌های بعدی را انجام دهیم.

۳. تبدیل ستون‌های تاریخ به فرمت datetime

```
# 3. Convert date/time columns to datetime objects.  
df['Released'] = pd.to_datetime(df['Released'], errors='coerce')  
df['Last Updated'] = pd.to_datetime(df['Last Updated'], errors='coerce')  
df['Scraped Time'] = pd.to_datetime(df['Scraped Time'], errors='coerce')
```

ستون‌های Released، Last Updated و Scraped Time که حاوی تاریخ‌ها هستند، به فرمت datetime تبدیل می‌شوند.

استفاده از errors='coerce' باعث می‌شود در صورت عدم توانایی تبدیل مقدار به تاریخ، مقدار NaN استفاده از (Not a Time) به جای آن قرار گیرد.

این تبدیل به ما کمک می‌کند تا بتوانیم در مراحل بعدی از ویژگی‌های زمانی (مانند سال انتشار یا بهروزرسانی) استفاده کنیم.

۴. وارد کردن داده‌ها به جدول categories

```
# 4. Insert data into the 'categories' table.  
#   The 'categories' table contains only the 'category' column.  
categories_df = df[['Category']].drop_duplicates().rename(columns={'Category': 'category'})  
categories_df.to_sql('categories', engine, if_exists='append', index=False)  
print("Categories inserted successfully.")
```

از DataFrame اصلی، تنها ستون Category استخراج می‌شود. با استفاده از drop_duplicates() رکوردهای تکراری حذف می‌شوند تا هر دسته‌بندی تنها یکبار در جدول ثبت شود.

سپس با تابع rename() نام ستون به category تغییر داده می‌شود تا با ساختار جدول مطابقت داشته باشد.

در نهایت، با استفاده از متدهای آماده شده به جدول categories وارد می‌شوند. پارامتر if_exists='append' به این معناست که اگر جدول موجود بود، داده‌ها به آن اضافه شوند. پارامتر index=False باعث می‌شود که ایندکس‌های DataFrame به عنوان ستونی جداگانه در جدول وارد نشوند. در آخر نیز پیامی مبنی بر موفقیت‌آمیز بودن عملیات چاپ می‌شود.

۵. وارد کردن داده‌ها به جدول developers

```
# 5. Insert data into the 'developers' table.  
#   We extract the relevant columns and drop duplicates based on 'developer_id'  
#   to avoid unique constraint violations.  
developers_df = df[['Developer Id', 'Developer Website', 'Developer Email']].rename(  
    columns={  
        'Developer Id': 'developer_id',  
        'Developer Website': 'developer_website',  
        'Developer Email': 'developer_email'  
    }  
)  
# Remove duplicate developers based on the 'developer_id' column  
developers_df = developers_df.drop_duplicates(subset=['developer_id'])  
developers_df.to_sql('developers', engine, if_exists='append', index=False)  
print("Developers inserted successfully.")
```

از دیتاست اصلی، ستون‌های مرتبط با توسعه‌دهندگان شامل Developer Id، Developer Email و Website استخراج می‌شود. با استفاده از متدهای `rename`، `developer_id`، `developer_email` و `developer_website` نام ستون‌ها به `developer_id`، `developer_email` و `developer_website` تغییر داده می‌شود تا با ساختار جدول مطابقت داشته باشد. سپس با استفاده از `drop_duplicates(subset=['developer_id'])` رکوردهای تکراری (براساس شناسه توسعه‌دهنده) حذف می‌شوند. در نهایت داده‌ها به جدول `developers` با استفاده از `to_sql` اضافه می‌شوند.

۶. وارد کردن داده‌ها به جدول applications

```
# 6. Insert data into the 'applications' table.  
#   We select and rename columns to match the schema of the applications table.  
applications_df = df[[  
    'App Id', 'App Name', 'Category', 'Rating', 'Rating Count', 'Installs',  
    'Minimum Installs', 'Maximum Installs', 'Free', 'Price', 'Currency', 'Size',  
    'Minimum Android', 'Developer Id', 'Released', 'Last Updated', 'Content Rating',  
    'Privacy Policy', 'Ad Supported', 'In App Purchases', 'Editors Choice', 'Scraped Time'  
]].rename(columns={  
    'App Id': 'app_id',  
    'App Name': 'app_name',  
    'Category': 'category',  
    'Rating': 'rating',  
    'Rating Count': 'rating_count',  
    'Installs': 'installs',  
    'Minimum Installs': 'minimum_installs',  
    'Maximum Installs': 'maximum_installs',  
    'Free': 'free',  
    'Price': 'price',  
    'Currency': 'currency',  
    'Size': 'size',  
    'Minimum Android': 'minimum_android',  
    'Developer Id': 'developer_id',  
    'Released': 'released',  
    'Last Updated': 'last_updated',  
    'Content Rating': 'content_rating',  
    'Privacy Policy': 'privacy_policy',  
    'Ad Supported': 'ad_supported',  
    'In App Purchases': 'in_app_purchases',  
    'Editors Choice': 'editors_choice',  
    'Scraped Time': 'scraped_time'  
})  
applications_df.to_sql('applications', engine, if_exists='append', index=False)  
print("Applications inserted successfully.")
```

ابتدا ستون‌های مرتبط با اطلاعات اپلیکیشن‌ها از دیتاست اصلی استخراج می‌شوند. این ستون‌ها شامل شناسه اپلیکیشن (App Id)، نام (App Name)، دسته‌بندی (Category)، امتیاز (Rating)، تعداد امتیاز (Rating Count)، تعداد نصب (Installs)، حداقل و حداکثر نصب (Maximum Installs و Minimum Installs) و رایگان بودن (Free)، قیمت (Price)، واحد پول (Currency)، اندازه (Size)، حداقل نسخه اندروید (Minimum Android)، شناسه توسعه‌دهنده (Developer Id)، تاریخ انتشار (Released)، تاریخ آخرین بهروزرسانی (Last Updated)، درجه‌بندی محتوا (Content Rating)، سیاست حفظ حریم خصوصی (Privacy Policy)، آگهی پشتیبانی شده (Ad Supported)، خریدهای درون برنامه‌ای (In App Purchases)، انتخاب سردبیر (Editors Choice) و زمان جمع‌آوری داده (Scraped Time) می‌باشد.

با استفاده از تابع `rename()` نام ستون‌ها به اسمی مطابق با ساختار جدول در پایگاه داده تغییر داده می‌شود.

سپس با استفاده از `to_sql()` داده‌های DataFrame به جدول applications اضافه می‌شوند.

در پایان، پیام "Applications inserted successfully" چاپ می شود تا از موفقیت عملیات مطلع شویم.

۴. توسعه بک اند (عملیات CRUD) (توضیح فایل API.py)

این کد شامل پیکربندی پایگاه داده، تعریف مدل های ORM (برای جداول categories و developers)، تعریف اسکیمه های Pydantic (applications)، تعیین جهت اعتبار سنجی ورودی/خروجی و ایجاد endpoint برای هر یک از این موجودیت ها می باشد.

۱. پیکربندی پایگاه داده

```
DATABASE_URL = "postgresql+psycopg2://postgres:Alireza1679@127.0.0.1/GooglePlayData"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

: رشته اتصال به پایگاه داده PostgreSQL است که شامل نام کاربری (postgres)، رمز عبور (Alireza1679)، آدرس سرور (127.0.0.1) و نام پایگاه داده (GooglePlayData) می باشد.

: با استفاده از این تابع یک engine برای اتصال به پایگاه داده ایجاد می شود.
: کلاس SessionLocal تنظیم شده است که به هر درخواست یک نشست (session) جدآگانه برای ارتباط با پایگاه داده اختصاص می دهد.

: این تابع پایه ای برای تعریف مدل های ORM ما فراهم می کند. تمام کلاس های declarative_base ما از این کلاس Base به ارث می برنند.

۲. تعریف مدل‌های ORM

category مدل ✦

```
class Category(Base):
    """
    ORM model for the 'categories' table.
    This table contains a single column 'category' which is used as the primary key.
    """
    __tablename__ = "categories"
    category = Column(String, primary_key=True, index=True)
```

این مدل نمایانگر جدول categories است. تنها ستونی که وجود دارد، category می‌باشد که هم کلید اصلی (PRIMARY KEY) است و هم ایندکس‌گذاری شده است تا جستجو سریع‌تر انجام شود. استفاده از این جدول باعث می‌شود دسته‌بندی‌های اپلیکیشن‌ها به صورت یکتا ذخیره شوند.

Developer مدل ✦

```
class Developer(Base):
    """
    ORM model for the 'developers' table.
    """
    __tablename__ = "developers"
    developer_id = Column(String, primary_key=True, index=True)
    developer_website = Column(String, nullable=True)
    developer_email = Column(String, nullable=True)
```

این مدل نمایانگر جدول developers است. ستون developer_id به عنوان کلید اصلی (PRIMARY KEY) تعریف شده و برای شناسایی یکتا توسعه‌دهندگان استفاده می‌شود. ستون‌های developer_email و developer_website نیز اطلاعات مربوط به وبسایت و ایمیل توسعه‌دهنده را نگهداری می‌کنند.

❖ Application مدل

```
class Application(Base):
    """
    ORM model for the 'applications' table.
    Note: The attribute "privacy_policy_url" is mapped to the actual column "privacy_policy" in the database.
    """
    __tablename__ = "applications"
    app_id = Column(String, primary_key=True, index=True)
    app_name = Column(String, nullable=False)
    category = Column(String, nullable=False) # References Category table
    rating = Column(Float, nullable=True)
    rating_count = Column(Integer, nullable=True)
    installs = Column(Integer, nullable=True)
    free = Column(Boolean, default=True)
    price = Column(Numeric(10, 2), nullable=True)
    currency = Column(String, nullable=True)
    size = Column(Float, nullable=True)
    minimum_installs = Column(Integer, nullable=True)
    maximum_installs = Column(Integer, nullable=True)
    minimum_android = Column(String, nullable=True)
    developer_id = Column(String, nullable=False) # References Developer table
    released = Column(Date, nullable=True)
    last_updated = Column(Date, nullable=True)
    content_rating = Column(String, nullable=True)
    # Mapping "privacy_policy_url" to actual column "privacy_policy"
    privacy_policy_url = Column("privacy_policy", String, nullable=True)
    ad_supported = Column(Boolean, default=False)
    in_app_purchases = Column(Boolean, default=False)
    editors_choice = Column(Boolean, default=False)
    scraped_time = Column(DateTime, nullable=True)
```

این مدل نمایانگر جدول applications است که تمامی اطلاعات مربوط به اپلیکیشن‌ها را ذخیره می‌کند.

ستون‌های free ,installs ,rating_count ,rating ,category ,app_name ,app_id ,maximum_installs ،minimum_installs ،size ،currency ،price ،content_rating ،last_updated ،released ،developer_id ،minimum_android در این scraped_time و editors_choice ،in_app_purchases ،ad_supported جدول وجود دارند.

توجه کنید که ستون privacy_policy_url در مدل با استفاده از متده است نگاشت به ستون واقعی privacy_policy در پایگاه داده نگاشت داده شده است.

ستون category به عنوان کلید خارجی به جدول categories و ستون developer_id به عنوان کلید خارجی به جدول developers ارجاع می‌دهد. در انتهای، با دستور:

```
Base.metadata.create_all(bind=engine)
```

تمامی جداول (در صورت عدم وجود) ایجاد می‌شوند.

۳. تعریف اسکیماهای Pydantic ❖ اسکیماهای مربوط به Category

```
class CategoryBase(BaseModel):
    category: str

class CategoryCreate(CategoryBase):
    """
    Schema for creating a new category.
    """
    pass

class CategoryUpdate(BaseModel):
    """
    Schema for updating a category.
    Since the primary key is the only field, caution is needed when updating.
    """
    category: Optional[str] = None

class CategoryOut(CategoryBase):
    """
    Schema for returning category data.
    """
    class Config:
        orm_mode = True
```

- : مدل پایه که تنها شامل فیلد category می‌شود.
- : برای ایجاد دسته‌بندی جدید، از مدل پایه استفاده می‌کند.
- : جهت بهروزرسانی دسته‌بندی؛ از آنجا که تنها فیلد موجود همان کلید اصلی است، باید با احتیاط بهروزرسانی شود.
- : مدل خروجی جهت بازگرداندن اطلاعات دسته‌بندی؛ اجازه می‌دهد که اشیاء ORM به عنوان داده‌های خروجی به کار روند.

❖ اسکیماهای مربوط به Developer

```
class DeveloperBase(BaseModel):
    developer_website: Optional[str] = None
    developer_email: Optional[str] = None

class DeveloperCreate(DeveloperBase):
    developer_id: str

class DeveloperUpdate(BaseModel):
    """
    Schema for updating a developer record.
    All fields are optional for partial updates.
    """
    developer_website: Optional[str] = None
    developer_email: Optional[str] = None

class DeveloperOut(DeveloperBase):
    developer_id: str
    class Config:
        orm_mode = True
```

: شامل فیلدهای عمومی مانند وبسایت و ایمیل توسعه‌دهنده.
DeveloperBase
: برای ایجاد یک توسعه‌دهنده جدید، علاوه بر فیلدهای پایه، فیلد
DeveloperCreate
نیز ضروری است. **developer_id**
: برای بهروزرسانی جزئی (partial updates)، تمامی فیلدها به صورت
DeveloperUpdate
اختیاری تعریف شده‌اند.
: مدل خروجی برای توسعه‌دهنده؛ با فعال بودن **orm_mode**، سازگاری با اشیاء
DeveloperOut
فراهم می‌شود.

❖ اسکیماهای مربوط به Application

```
class ApplicationBase(BaseModel):
    app_name: str
    category: str
    rating: Optional[float] = None
    rating_count: Optional[int] = None
    installs: Optional[int] = None
    free: bool = True
    price: Optional[float] = None
    currency: Optional[str] = None
    size: Optional[float] = None
    minimum_installs: Optional[int] = None
    maximum_installs: Optional[int] = None
    minimum_android: Optional[str] = None
    developer_id: str
    released: Optional[datetime.date] = None
    last_updated: Optional[datetime.date] = None
    content_rating: Optional[str] = None
    privacy_policy_url: Optional[str] = None
    ad_supported: bool = False
    in_app_purchases: bool = False
    editors_choice: bool = False
    scraped_time: Optional[datetime.datetime] = None

class ApplicationCreate(ApplicationBase):
    app_id: str

class ApplicationUpdate(BaseModel):
    app_name: Optional[str] = None
    category: Optional[str] = None
    rating: Optional[float] = None
    rating_count: Optional[int] = None
    installs: Optional[int] = None
    free: Optional[bool] = None
    price: Optional[float] = None
    currency: Optional[str] = None
    size: Optional[float] = None
    minimum_installs: Optional[int] = None
    maximum_installs: Optional[int] = None
    minimum_android: Optional[str] = None
    developer_id: Optional[str] = None
    released: Optional[datetime.date] = None
    last_updated: Optional[datetime.date] = None
    content_rating: Optional[str] = None
    privacy_policy_url: Optional[str] = None
    ad_supported: Optional[bool] = None
    in_app_purchases: Optional[bool] = None
    editors_choice: Optional[bool] = None
    scraped_time: Optional[datetime.datetime] = None

class ApplicationOut(ApplicationBase):
    app_id: str
    class Config:
        orm_mode = True
```

: **ApplicationBase** شامل تمام فیلدهای عمومی اپلیکیشن مانند نام، دسته‌بندی، امتیاز، تعداد امتیاز، تعداد نصب، اطلاعات قیمت، اندازه، حداقل و حداکثر نصب، نسخه اندروید، شناسه توسعه‌دهنده، تاریخ‌ها، درجه‌بندی محتوا، آدرس سیاست حفظ حریم خصوصی، و سایر ویژگی‌ها.
: **ApplicationCreate** برای ایجاد یک اپلیکیشن جدید، علاوه بر فیلدهای پایه، `app_id` نیز **الرامی** است.

: **ApplicationUpdate** جهت به روزرسانی اپلیکیشن؛ تمامی فیلدها به صورت اختیاری تعریف شده‌اند تا امکان به روزرسانی جزئی فراهم شود.

: **ApplicationOut** مدل خروجی برای اپلیکیشن، که شامل `app_id` نیز می‌شود. با تنظیم `orm_mode = True`، اشیاء ORM به صورت خودکار به این اسکیما تبدیل می‌شوند.

۴. ایجاد FastAPI Application و تعریف Dependency

```
app = FastAPI(title="Google Play Applications API")

def get_db():
    """
    Creates a new database session for each request and closes it after the request.
    """
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

FastAPI Application Instance

با **FastAPI()** یک شیء اپلیکیشن ایجاد می‌کنیم که تمامی `endpoint`‌های API بر روی آن تعریف می‌شوند.

Dependency get_db()

این تابع یک نشست (`session`) پایگاه داده برای هر درخواست ایجاد می‌کند و پس از پایان درخواست، آن را می‌بندد. از این طریق می‌توانیم از یک `dependency` مشترک برای دسترسی به پایگاه داده در تمامی `endpoint`‌ها استفاده کنیم.

۵. تعریف های Endpoint

ریشه Endpoint ✧

```
@app.get("/")
def read_root():
    """
    Root endpoint that returns a welcome message and directs users to the API docs.
    """
    return {"message": "Welcome to the Google Play Applications API. Visit /docs for API documentation."}
```

این endpoint یک پیام خوشآمدگویی برمی‌گرداند و کاربران را به مستندات API (که در `/docs`) راهنمایی می‌کند.

Categories برای CRUD های Endpoint ✧

```
@app.post("/categories/", response_model=CategoryOut)
def create_category(category: CategoryCreate, db: Session = Depends(get_db)):
    """
    Create a new category.

    - **category**: Category data provided in the request body.
    - **db**: Database session dependency.
    """
    db_cat = db.query(Category).filter(Category.category == category.category).first()
    if db_cat:
        raise HTTPException(status_code=400, detail="Category already exists")
    new_cat = Category(**category.dict())
    db.add(new_cat)
    db.commit()
    db.refresh(new_cat)
    return new_cat

@app.get("/categories/", response_model=List[CategoryOut])
def read_categories(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    """
    Retrieve a list of categories with pagination.

    - **skip**: Number of records to skip.
    - **limit**: Maximum number of records to return.
    - **db**: Database session dependency.
    """
    cats = db.query(Category).offset(skip).limit(limit).all()
    return cats

@app.get("/categories/{category_id}", response_model=CategoryOut)
def read_category(category_id: str, db: Session = Depends(get_db)):
    """
    Retrieve a specific category by its identifier.

    - **category_id**: The unique category identifier (category name).
    - **db**: Database session dependency.
    """
    db_cat = db.query(Category).filter(Category.category == category_id).first()
    if not db_cat:
        raise HTTPException(status_code=404, detail="Category not found")
    return db_cat
```

```

@app.put("/categories/{category_id}", response_model=CategoryOut)
def update_category(category_id: str, category_update: CategoryUpdate, db: Session = Depends(get_db)):
    """
    Update an existing category.

    - **category_id**: The unique identifier of the category to update.
    - **category_update**: Data for updating the category.
    - **db**: Database session dependency.

    Note: Since the primary key is the category name, updating it may affect related foreign key constraints.
    """
    db_cat = db.query(Category).filter(Category.category == category_id).first()
    if not db_cat:
        raise HTTPException(status_code=404, detail="Category not found")

    update_data = category_update.dict(exclude_unset=True)
    if "category" in update_data and update_data["category"]:
        db_cat.category = update_data["category"]
    db.commit()
    db.refresh(db_cat)
    return db_cat

@app.delete("/categories/{category_id}")
def delete_category(category_id: str, db: Session = Depends(get_db)):
    """
    Delete a category from the database.

    - **category_id**: The unique identifier of the category to delete.
    - **db**: Database session dependency.

    Note: Deleting a category that is referenced by applications may result in a foreign key violation.
    """
    db_cat = db.query(Category).filter(Category.category == category_id).first()
    if not db_cat:
        raise HTTPException(status_code=404, detail="Category not found")
    db.delete(db_cat)
    db.commit()
    return {"detail": "Category deleted successfully"}

```

یک دسته‌بندی جدید ایجاد می‌کند. ابتدا بررسی می‌کند که دسته‌بندی با همان نام قبلاً وجود ندارد و سپس دسته‌بندی جدید را در جدول ذخیره می‌کند. (limit و skip) لیستی از دسته‌بندی‌ها را بر اساس پارامترهای صفحه‌بندی (Create_category()) بازمی‌گرداند.

یک دسته‌بندی خاص (با استفاده از نام دسته‌بندی به عنوان شناسه) را بازمی‌گرداند.

با توجه به اینکه کلید اصلی جدول categories همان نام دسته‌بندی است، به روزرسانی آن نیازمند دقت است. (Update_category()) در صورت وجود دسته‌بندی، اطلاعات به روزرسانی شده را ذخیره می‌کند.

دسته‌بندی مشخص شده را حذف می‌کند. نکته اینکه حذف دسته‌بندی‌هایی که توسط اپلیکیشن‌ها استفاده شده‌اند ممکن است منجر به خطای کلید خارجی (Delete_category()) شود.

Developers CRUDe Endpoint ✦

```
@app.post("/developers/", response_model=DeveloperOut)
def create_developer(developer: DeveloperCreate, db: Session = Depends(get_db)):
    """
    Create a new developer record.
    """
    db_dev = db.query(Developer).filter(Developer.developer_id == developer.developer_id).first()
    if db_dev:
        raise HTTPException(status_code=400, detail="Developer already exists")
    new_dev = Developer(**developer.dict())
    db.add(new_dev)
    db.commit()
    db.refresh(new_dev)
    return new_dev

@app.get("/developers/{developer_id}", response_model=DeveloperOut)
def read_developer(developer_id: str, db: Session = Depends(get_db)):
    """
    Retrieve a developer by developer_id.
    """
    db_dev = db.query(Developer).filter(Developer.developer_id == developer_id).first()
    if not db_dev:
        raise HTTPException(status_code=404, detail="Developer not found")
    return db_dev

@app.put("/developers/{developer_id}", response_model=DeveloperOut)
def update_developer(developer_id: str, developer_update: DeveloperUpdate, db: Session = Depends(get_db)):
    """
    Update an existing developer record.
    - **developer_id**: The unique identifier of the developer to update.
    - **developer_update**: Data for updating the developer.
    - **db**: Database session dependency.
    """
    db_dev = db.query(Developer).filter(Developer.developer_id == developer_id).first()
    if not db_dev:
        raise HTTPException(status_code=404, detail="Developer not found")
    update_data = developer_update.dict(exclude_unset=True)
    for key, value in update_data.items():
        setattr(db_dev, key, value)
    db.commit()
    db.refresh(db_dev)
    return db_dev

@app.delete("/developers/{developer_id}")
def delete_developer(developer_id: str, db: Session = Depends(get_db)):
    """
    Delete a developer record.
    - **developer_id**: The unique identifier of the developer to delete.
    - **db**: Database session dependency.
    """
    db_dev = db.query(Developer).filter(Developer.developer_id == developer_id).first()
    if not db_dev:
        raise HTTPException(status_code=404, detail="Developer not found")
    db.delete(db_dev)
    db.commit()
    return {"detail": "Developer deleted successfully"}
```

یک رکورد جدید در جدول developers ایجاد می‌کند. در صورت وجود رکورد با همان شناسه، خطای ۴۰۰ برگردانده می‌شود.
یک توسعه‌دهنده را بر اساس شناسه آن بازمی‌گرداند.

اجازه به روزرسانی جزئی (partial update) برای یک توسعه‌دهنده را می‌دهد. در صورت عدم وجود توسعه‌دهنده خطای ۴۰۴ برگردانده می‌شود. رکورد توسعه‌دهنده مشخص شده را حذف می‌کند.

Applications برای CRUD های Endpoint ❖

```

@app.post("/applications/", response_model=ApplicationOut)
def create_application(app_data: ApplicationCreate, db: Session = Depends(get_db)):
    """
    Create a new application record.
    Note: The referenced developer must already exist.
    """
    db_app = db.query(Application).filter(Application.app_id == app_data.app_id).first()
    if db_app:
        raise HTTPException(status_code=400, detail="Application already exists")

    # Ensure that the referenced developer exists.
    db_dev = db.query(Developer).filter(Developer.developer_id == app_data.developer_id).first()
    if not db_dev:
        raise HTTPException(status_code=400,
                            detail=f"Developer '{app_data.developer_id}' does not exist. Please create the developer record first.")

    new_app = Application(**app_data.dict())
    db.add(new_app)
    db.commit()
    db.refresh(new_app)
    return new_app

@app.get("/applications/", response_model=List[ApplicationOut])
def read_applications(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    """
    Retrieve a list of applications with pagination.
    - **skip**: Number of records to skip.
    - **limit**: Maximum number of records to return.
    - **db**: Database session dependency.
    """
    apps = db.query(Application).offset(skip).limit(limit).all()
    return apps

@app.get("/applications/{app_id}", response_model=ApplicationOut)
def read_application(app_id: str, db: Session = Depends(get_db)):
    """
    Retrieve a specific application by its app_id.
    - **app_id**: Unique identifier of the application.
    - **db**: Database session dependency.
    """
    db_app = db.query(Application).filter(Application.app_id == app_id).first()
    if db_app is None:
        raise HTTPException(status_code=404, detail="Application not found")
    return db_app

@app.put("/applications/{app_id}", response_model=ApplicationOut)
def update_application(app_id: str, app_update: ApplicationUpdate, db: Session = Depends(get_db)):
    """
    Update an existing application record.
    - **app_id**: Unique identifier of the application to update.
    - **app_update**: Data to update (partial updates allowed).
    - **db**: Database session dependency.
    """
    db_app = db.query(Application).filter(Application.app_id == app_id).first()
    if db_app is None:
        raise HTTPException(status_code=404, detail="Application not found")

    update_data = app_update.dict(exclude_unset=True)
    # If updating developer_id, ensure the new developer exists.
    if "developer_id" in update_data:
        db_dev = db.query(Developer).filter(Developer.developer_id == update_data["developer_id"]).first()
        if not db_dev:
            raise HTTPException(status_code=400, detail=f"Developer '{update_data['developer_id']}' does not exist.")

    for key, value in update_data.items():
        setattr(db_app, key, value)
    db.commit()
    db.refresh(db_app)
    return db_app

@app.delete("/applications/{app_id}")
def delete_application(app_id: str, db: Session = Depends(get_db)):
    """
    Delete an application record.
    - **app_id**: Unique identifier of the application to delete.
    - **db**: Database session dependency.
    """
    db_app = db.query(Application).filter(Application.app_id == app_id).first()
    if db_app is None:
        raise HTTPException(status_code=404, detail="Application not found")
    db.delete(db_app)
    db.commit()
    return {"detail": "Application deleted successfully"}

```

یک اپلیکیشن جدید ایجاد می‌کند. قبل از ایجاد رکورد جدید، بررسی می‌شود که آیا اپلیکیشن با همان `app_id` وجود دارد یا خیر. همچنین وجود توسعه‌دهنده‌ای که با مشخص شده وجود دارد یا نه؛ در غیر این صورت، خطای مناسب برگردانده می‌شود.

(`pagination`) لیستی از اپلیکیشن‌ها را با پشتیبانی از صفحه‌بندی (`Read_applications()`) بازمی‌گرداند.

یک اپلیکیشن را بر اساس شناسه آن (`app_id`) بازمی‌گرداند. در صورت عدم وجود رکورد، خطای ۴۰۴ ارسال می‌شود.

(`Update_application()`) امکان بهروزرسانی جزئی رکورد اپلیکیشن را فراهم می‌کند. در صورت درخواست تغییر شناسه توسعه‌دهنده، ابتدا وجود توسعه‌دهنده جدید بررسی می‌شود.

(`Delete_application()`) رکورد اپلیکیشن مورد نظر را حذف می‌کند و در صورت عدم یافتن رکورد، خطای ۴۰۴ ارسال می‌کند.

۶. اجرای API

```
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

با اجرای این بخش (به عنوان مثال با دستور `python API.py` یا استفاده از `uvicorn` با گزینه `--reload`، API را روی آدرس `http://127.0.0.1:8000` راهاندازی می‌شود).

Endpoint ریشه (/) پیام خوش‌آمدگویی ارسال می‌کند. از طریق آدرس `docs` می‌توانیم مستندات تعاملی (Swagger UI) API را مشاهده کنیم.

۵. توسعه فرانت اند (توسعه داشبورد با Streamlit) (توضیح فایل dashboard.py)

۱. پیکربندی Streamlit

```
st.set_page_config(page_title="Google Play Data Dashboard", layout="wide")
```

تنظیمات اولیه صفحه داشبورد Streamlit انجام می‌شود؛ عنوان صفحه تنظیم شده و طرح‌بندی صفحه به صورت "wide" (عرضی) انتخاب می‌شود.

۲. تابع بارگذاری داده‌ها (load_data)

```
@st.cache_data
def load_data():
    # Define PostgreSQL connection parameters
    host = '127.0.0.1'
    dbname = 'GooglePlayData'
    user = 'postgres'
    password = 'Alireza1679'

    # Connection string for PostgreSQL
    connection_str = f'postgresql+psycopg2://{user}:{password}@{host}/{dbname}'

    # Create a SQLAlchemy engine
    engine = create_engine(connection_str)
```

تابع `load_data` با دکوریتور `@st.cache_data` تزیین شده است تا داده‌ها تنها یکبار خوانده شوند و در دفعات بعدی از کش استفاده گردد.

در این قسمت، اطلاعات اتصال به پایگاه داده PostgreSQL (شامل آدرس، نام پایگاه داده، نام کاربری و رمز عبور) تنظیم می‌شود.

یک `engine` از طریق SQLAlchemy ایجاد می‌شود تا بتوان از آن برای اجرای کوئری‌ها استفاده کرد.

```

query = """
SELECT
    a.app_id,
    a.app_name,
    a.category,
    a.rating,
    a.rating_count,
    a.installs,
    a.free,
    a.price,
    a.currency,
    a.size,
    a.minimum_installs,
    a.maximum_installs,
    a.minimum_android,
    a.developer_id,
    a.released AS release_date,
    a.last_updated,
    a.content_rating,
    a.privacy_policy AS privacy_policy_url,
    a.ad_supported,
    a.in_app_purchases,
    a.editors_choice,
    a.scraped_time,
    c.category AS category_name,
    d.developer_email
FROM applications a
JOIN categories c ON a.category = c.category
JOIN developers d ON a.developer_id = d.developer_id;
"""
df = pd.read_sql(query, engine)
return df

```

یک پرس‌وجوی SQL نوشته شده است که ستون‌های مورد نظر از جدول `applications` به همراه ستون‌های اضافی از جداول `developers` و `categories` (با استفاده از JOIN) انتخاب می‌کند. ستون `privacy_policy` به عنوان alias با `privacy_policy_url` انتخاب شده تا در DataFrame خروجی به این نام شناخته شود. در نهایت، داده‌ها با استفاده از تابع `pd.read_sql()` در یک DataFrame ذخیره و بازگردانده می‌شوند.

۳. بارگذاری داده‌ها و نمایش زمان اجرا

```
load_start = time.time()
df = load_data()
load_time = time.time() - load_start
st.sidebar.write(f"Data loaded in {load_time:.2f} seconds")
```

زمان شروع بارگذاری داده‌ها ثبت می‌شود.

پس از بارگذاری، زمان سپری شده محاسبه شده و در نوار کناری (sidebar) نمایش داده می‌شود.

۴. فیلترهای صفحه کناری (Sidebar Filters)

```
st.sidebar.header("Filter Options")

# Filter by Category
category_filter = st.sidebar.multiselect(
    "Select Categories",
    options=df["category"].unique(),
    default=df["category"].unique()
)

# Filter by Rating Range
rating_filter = st.sidebar.slider(
    "Select Rating Range",
    min_value=0.0,
    max_value=5.0,
    value=(0.0, 5.0)
)

# Filter by Price (Free or Paid)
price_filter = st.sidebar.selectbox(
    "Select Price Range",
    options=["Free", "Paid"]
)

# Filter by Content Rating
content_rating_filter = st.sidebar.multiselect(
    "Select Content Ratings",
    options=df["content_rating"].unique(),
    default=df["content_rating"].unique()
)

# Advanced Search: Search for App Name
search_term = st.sidebar.text_input("Search for App Name", "")
```

بخش فیلترها با یک هدر (Header) در نوار کناری نمایش داده می‌شود.

✓ فیلتر بر اساس دسته‌بندی

با استفاده از `multiselect` کاربر می‌تواند یک یا چند دسته‌بندی را انتخاب کند.

به صورت پیش‌فرض تمام دسته‌بندی‌ها انتخاب شده‌اند.

✓ فیلتر بر اساس محدوده امتیاز

یک اسلایدر جهت انتخاب بازه امتیاز بین ۰ تا ۵ قرار داده شده است.

✓ فیلتر بر اساس وضعیت رایگان بودن (Paid Free یا Paid)

یک منوی کشویی (Selectbox) فراهم شده تا کاربر بتواند بین اپلیکیشن‌های رایگان و پرداختی

انتخاب کند.

✓ فیلتر بر اساس درجه‌بندی محتوا (Content Rating)

با استفاده از این فیلتر، کاربر می‌تواند بر اساس درجه‌بندی محتوا (مثلًا Teen، Everyone و

...) اپلیکیشن‌ها را فیلتر کند.

✓ جستجوی پیشرفته بر اساس نام اپلیکیشن

یک فیلد متنی جهت وارد کردن عبارت جستجو برای نام اپلیکیشن فراهم شده است.

۵. فیلتر کردن DataFrame

```
filtered_df = df[
    (df["category"].isin(category_filter)) &
    (df["rating"] >= rating_filter[0]) &
    (df["rating"] <= rating_filter[1]) &
    (df["free"] == (price_filter == "Free")) &
    (df["content_rating"].isin(content_rating_filter))
]

# Apply advanced search filter if a search term is provided
if search_term:
    filtered_df = filtered_df[filtered_df["app_name"].str.contains(search_term, case=False, na=False)]

st.write(f"Showing {len(filtered_df)} apps after filtering")
# Limit displayed rows to avoid high memory usage
st.dataframe(filtered_df.head(3000))
```

اصلی `df` بر اساس فیلترهای انتخاب‌شده توسط کاربر (دسته‌بندی، امتیاز، وضعیت Free/Paid و درجه‌بندی محتوا) فیلتر می‌شود.

در صورت وجود یک عبارت جستجو، اپلیکیشن‌هایی که نام آن‌ها شامل متن جستجو شده باشد، انتخاب می‌شوند.

نتیجه فیلتر نهایی در `filtered_df` ذخیره می‌شود.
تعداد رکوردهای باقی‌مانده پس از اعمال فیلترها چاپ می‌شود.
تنها ۳۰۰۰ ردیف اول به صورت جدول در داشبورد نمایش داده می‌شود تا از بارگذاری بیش از حد داده جلوگیری شود.

۶. پردازش داده‌های تاریخ

```
filtered_df["release_date"] = pd.to_datetime(filtered_df["release_date"], errors="coerce")
filtered_df["last_updated"] = pd.to_datetime(filtered_df["last_updated"], errors="coerce")
filtered_df["release_year"] = filtered_df["release_date"].dt.year
filtered_df["updated_year"] = filtered_df["last_updated"].dt.year
```

ستون‌های `last_updated` و `release_date` به فرمت `datetime` تبدیل می‌شوند.
سپس سال انتشار و سال بهروزرسانی از این ستون‌ها استخراج شده و به عنوان ستون‌های DataFrame `updated_year` و `release_year` اضافه می‌شوند.
این اطلاعات برای مصورسازی روند زمانی مورد استفاده قرار می‌گیرند.

۷. مصورسازی داده‌ها (Charts)

✓ نمودار روند انتشار اپلیکیشن‌ها (Release Trend)

```
st.subheader("App Release Trend Over the Years")
selected_category_for_trend = st.selectbox(
    "Select a Category for Release Trend",
    options=df["category"].unique(),
    key="release_trend"
)
release_start = time.time()
release_trend = filtered_df[filtered_df["category"] == selected_category_for_trend].groupby("release_year").size().reset_index(name="App Count")
release_time = time.time() - release_start
st.write(f"Release Trend query executed in {release_time:.2f} seconds")
fig_release = px.line(release_trend, x="release_year", y="App Count", title=f"Release Trend for {selected_category_for_trend}")
st.plotly_chart(fig_release)
```

کاربر یک دسته‌بندی از میان گزینه‌های موجود انتخاب می‌کند.
داده‌های مربوط به اپلیکیشن‌های آن دسته‌بندی بر اساس سال انتشار گروه‌بندی شده و تعداد آن‌ها محاسبه می‌شود.
زمان اجرای گروه‌بندی اندازه‌گیری و نمایش داده می‌شود.

سپس نمودار خطی (line chart) جهت نمایش روند انتشار اپلیکیشن‌ها رسم و به نمایش درمی‌آید.

✓ نمودار روند بهروزرسانی اپلیکیشن‌ها (Last Updated Trend)

```
st.subheader("Last Updated Trend Over the Years")
selected_category_for_update = st.selectbox(
    "Select a Category for Last Updated Trend",
    options=df["category"].unique(),
    key="update_trend"
)
update_start = time.time()
update_trend = filtered_df[filtered_df["category"] == selected_category_for_update].groupby("updated_year").size().reset_index(name="App Count")
update_time = time.time() - update_start
st.write(f"Last Updated query executed in {update_time:.2f} seconds")
fig_update = px.line(update_trend, x="updated_year", y="App Count", title=f"Last Updated Trend for {selected_category_for_update}")
st.plotly_chart(fig_update)
```

مشابه نمودار قبل، این قسمت اپلیکیشن‌های یک دسته‌بندی خاص را بر اساس سال بهروزرسانی گروه‌بندی می‌کند.

زمان اجرای این عملیات اندازه‌گیری و سپس نمودار خطی رسم می‌شود.

✓ نمودار میانگین امتیاز بر اساس دسته‌بندی‌ها (Average Rating)

```
st.subheader("Average Rating per Category")
avg_start = time.time()
avg_rating = filtered_df.groupby("category")["rating"].mean().reset_index()
avg_time = time.time() - avg_start
st.write(f"Average Rating query executed in {avg_time:.2f} seconds")
fig_avg_rating = px.bar(avg_rating, x="category", y="rating", title="Average Rating by Category")
st.plotly_chart(fig_avg_rating)
```

داده‌ها بر اساس دسته‌بندی گروه‌بندی شده و میانگین امتیاز هر دسته محاسبه می‌شود. نمودار میله‌ای رسم می‌شود تا میانگین امتیاز هر دسته به‌طور بصری نمایش داده شود.

✓ نمودار توزیع قیمت اپلیکیشن‌های پرداختی (Price Distribution)

```
st.subheader("Price Distribution of Paid Apps")
paid_apps = filtered_df[filtered_df["price"] > 0]
if not paid_apps.empty:
    price_start = time.time()
    fig_price = px.histogram(paid_apps, x="price", nbins=50, title="Price Distribution of Paid Apps")
    price_time = time.time() - price_start
    st.write(f"Price Distribution query executed in {price_time:.2f} seconds")
    fig_price.update_layout(xaxis_title="Price (USD)", yaxis_title="Number of Apps", bargap=0.1)
    st.plotly_chart(fig_price)
else:
    st.write("No paid apps available in the current filter.")
```

ابتدا اپلیکیشن‌های دارای قیمت (غیر صفر) فیلتر می‌شوند.
سپس نمودار هیستوگرام با ۵۰ بین (bins) رسم می‌شود تا توزیع قیمت‌ها نمایش داده شود.
زمان اجرای پرس‌وجو اندازه‌گیری شده و در داشبورد نمایش داده می‌شود.

✓ نمودار مجموع تعداد نصب‌ها به تفکیک دسته‌بندی (Total Installs)

```
st.subheader("Total Installs by Category")
installs_start = time.time()
installs_dist = filtered_df.groupby("category")["installs"].sum().reset_index()
installs_time = time.time() - installs_start
st.write(f"Total Installs query executed in {installs_time:.2f} seconds")
fig_installs = px.bar(installs_dist, x="category", y="installs", title="Total Installs by Category")
st.plotly_chart(fig_installs)
```

داده‌ها بر اساس دسته‌بندی گروه‌بندی شده و مجموع تعداد نصب‌ها محاسبه می‌شود.
نمودار میله‌ای رسم شده تا مجموع نصب‌ها در هر دسته‌بندی نمایش داده شود.

۸. بخش عملیات CRUD (از طریق API)

این قسمت داشبورد، از کتابخانه requests برای ارسال درخواست‌های HTTP به API استفاده می‌کند.
عملیات CRUD برای سه موجودیت Applications، Developers و Categories به صورت زیر
پیاده‌سازی شده‌اند:

❖ انتخاب عملیات CRUD

```
crud_option = st.selectbox("Select CRUD Operation",
                           options=["Create Application", "Update Application", "Delete Application", "Get Application",
                                     "Create Developer", "Update Developer", "Delete Developer", "Get Developer",
                                     "Create Category", "Update Category", "Delete Category", "Get Category"])
```

با استفاده از یک selectbox ، کاربر می تواند یکی از عملیات CRUD (ایجاد، بروزرسانی، حذف یا دریافت) را برای هر یک از سه موجودیت انتخاب کند.

Applications برای CRUD ❖

```
if crud_option == 'Create Application':
    with st.form("Create Application"):
        app_name = st.text_input("App Name")
        category = st.text_input("Category")
        rating_count = st.number_input("Rating Count", min_value=0, max_value=100, value=0)
        rating_mean = st.number_input("Rating Mean", min_value=0.0, max_value=5.0, value=0.0)
        price = st.number_input("Price", min_value=0, value=0.0)
        released = st.date_input("Released")
        content_rating = st.text_input("Content Rating")
        privacy_policy_url = st.text_input("Privacy Policy URL")
        android_id = st.text_input("Android ID", optional=True, value="None")
        in_app_purchases = st.text_input("In-App Purchases", optional=True, value="None")
        editors_choice = st.text_input("Editor's Choice", optional=True, value="None")
        submitted = st.form.submit_button("Create Application")

    if submitted:
        payload = {
            "app_id": None,
            "app_name": app_name,
            "category": category,
            "rating_count": rating_count,
            "rating_mean": rating_mean,
            "price": price,
            "content_rating": content_rating,
            "released": released.isoformat() if released else None,
            "privacy_policy_url": privacy_policy_url,
            "android_id": android_id,
            "in_app_purchases": in_app_purchases,
            "editors_choice": editors_choice,
            "last_updated": datetime.now().isoformat()
        }
        response = requests.post(f"{api_base_url}/applications/", json=payload)
        if response.status_code in (200, 201):
            st.success("Creation successful!")
        else:
            st.error(f"Failed to create application: {response.text}")

if crud_option == 'Update Application':
    with st.form("Update Application"):
        app_id = st.text_input("App ID to update")
        app_name = st.text_input("App Name")
        category = st.text_input("Category")
        rating_count = st.number_input("Rating", min_value=0, max_value=100, value=0)
        rating_mean = st.number_input("Rating Mean", min_value=0.0, max_value=5.0, value=0.0)
        installs = st.number_input("Installs", min_value=0, value=0, step=1)
        price = st.number_input("Price", min_value=0, value=0.0)
        currency = st.text_input("Currency")
        minimum_installs = st.number_input("Minimum Installs", min_value=0, value=0)
        maximum_installs = st.number_input("Maximum Installs", min_value=0, value=0, step=1)
        minimum_android_id = st.text_input("Minimum Android ID", optional=True, value="None")
        developer_id = st.text_input("Developer ID", optional=True, value="None")
        last_updated = st.date_input("Last Updated")
        privacy_policy_url = st.text_input("Privacy Policy URL")
        android_id = st.text_input("Android ID", optional=True, value="None")
        in_app_purchases = st.text_input("In-App Purchases", optional=True, value="None")
        editors_choice = st.text_input("Editor's Choice", optional=True, value="None")
        submitted = st.form.submit_button("Update Application")

    if submitted:
        payload = {
            "app_id": app_id,
            "app_name": payload["app_name"] + " New Name",
            "category": payload["category"] + category
        }
        payload["rating_count"] = rating_count
        payload["rating_mean"] = rating_mean
        payload["installs"] = installs
        payload["price"] = price
        payload["currency"] = currency
        payload["minimum_installs"] = minimum_installs
        payload["maximum_installs"] = maximum_installs
        payload["minimum_android_id"] = minimum_android_id
        payload["developer_id"] = developer_id
        payload["last_updated"] = last_updated.isoformat()
        payload["content_rating"] = content_rating
        payload["privacy_policy_url"] = privacy_policy_url
        payload["android_id"] = android_id
        payload["in_app_purchases"] = in_app_purchases
        payload["editors_choice"] = editors_choice
        response = requests.patch(f"{api_base_url}/applications/{app_id}")
        if response.status_code in (200, 201):
            st.success("Update successful!")
        else:
            st.error(f"Failed to update application: {response.text}")

if crud_option == 'Delete Application':
    st.warning("Are you sure to delete?")
    submitted = st.form.submit_button("Delete Application")
    if submitted:
        response = requests.delete(f"{api_base_url}/applications/{app_id}")
        if response.status_code == 200:
            st.success("Delete successful!")
        else:
            st.error(f"Failed to delete application: {response.text}")

if crud_option == 'Get Application Details':
    with st.form("Get Application Details"):
        app_id = st.text_input("App ID to retrieve")
        submitted = st.form.submit_button("Get Application")
        if submitted:
            response = requests.get(f"{api_base_url}/applications/{app_id}")
            if response.status_code == 200:
                st.json(response.json())
            else:
                st.error(f"Failed to retrieve application: {response.text}")
```

ایجاد اپلیکیشن:

فرم "create_app_form" اطلاعاتی مانند شناسه، نام، دسته‌بندی، امتیاز، تعداد امتیاز، تعداد نصب، وضعیت رایگان بودن، قیمت، واحد پول، اندازه، حداقل و حداکثر نصب، نسخه اندروید، شناسه توسعه‌دهنده، تاریخ انتشار، تاریخ آخرین بهروزرسانی، درجه‌بندی محتوا، آدرس سیاست حریم خصوصی، وضعیت تبلیغات، خریدهای درون برنامه‌ای، انتخاب سردبیر و زمان جمع‌آوری داده‌ها را از کاربر دریافت می‌کند.

پس از ارسال فرم، یک درخواست POST به endpoint /applications ارسال می‌شود.

بهروزرسانی اپلیکیشن:

فرم "update_app_form" امکان وارد کردن شناسه اپلیکیشن و فیلدهایی که نیاز به بهروزرسانی دارند را فراهم می‌کند.
درخواست PUT به endpoint /applications/{app_id} ارسال می‌شود.

حذف اپلیکیشن:

فرم "delete_app_form" تنها شناسه اپلیکیشن مورد نظر را می‌گیرد.
درخواست DELETE به endpoint /applications/{app_id} ارسال می‌شود.

دربیافت اطلاعات اپلیکیشن:

فرم "get_app_form" با دریافت شناسه اپلیکیشن، اطلاعات آن را از طریق درخواست GET دریافت و نمایش می‌دهد.

Developers CRUD ✨

```
elif crud_option == "Create Developer":  
    st.markdown("### Create a New Developer")  
    with st.form("create_dev_form"):  
        developer_id = st.text_input("Developer ID")  
        developer_website = st.text_input("Developer Website")  
        developer_email = st.text_input("Developer Email")  
        submitted = st.form_submit_button("Create Developer")  
    if submitted:  
        payload = {  
            "developer_id": developer_id,  
            "developer_website": developer_website,  
            "developer_email": developer_email  
        }  
        response = requests.post(f"{api_base_url}/developers/", json=payload)  
        if response.status_code in (200, 201):  
            st.success("Developer created successfully!")  
        else:  
            st.error(f"Failed to create developer: {response.text}")  
  
elif crud_option == "Update Developer":  
    st.markdown("### Update an Existing Developer")  
    with st.form("update_dev_form"):  
        developer_id = st.text_input("Developer ID to Update")  
        st.markdown("Enter fields to update:")  
        developer_website = st.text_input("Developer Website")  
        developer_email = st.text_input("Developer Email")  
        submitted = st.form_submit_button("Update Developer")  
    if submitted:  
        payload = {}  
        if developer_website: payload["developer_website"] = developer_website  
        if developer_email: payload["developer_email"] = developer_email  
        response = requests.put(f"{api_base_url}/developers/{developer_id}", json=payload)  
        if response.status_code == 200:  
            st.success("Developer updated successfully!")  
        else:  
            st.error(f"Failed to update developer: {response.text}")  
  
elif crud_option == "Delete Developer":  
    st.markdown("### Delete a Developer")  
    with st.form("delete_dev_form"):  
        developer_id = st.text_input("Developer ID to Delete")  
        submitted = st.form_submit_button("Delete Developer")  
    if submitted:  
        response = requests.delete(f"{api_base_url}/developers/{developer_id}")  
        if response.status_code == 200:  
            st.success("Developer deleted successfully!")  
        else:  
            st.error(f"Failed to delete developer: {response.text}")  
  
elif crud_option == "Get Developer":  
    st.markdown("### Get Developer Details")  
    with st.form("get_dev_form"):  
        developer_id = st.text_input("Developer ID to Retrieve")  
        submitted = st.form_submit_button("Get Developer")  
    if submitted:  
        response = requests.get(f"{api_base_url}/developers/{developer_id}")  
        if response.status_code == 200:  
            dev_data = response.json()  
            st.json(dev_data)  
        else:  
            st.error(f"Failed to retrieve developer: {response.text}")
```

ایجاد توسعه‌دهنده:

فرم "create_dev_form" شامل فیلدهای شناسه توسعه‌دهنده، وبسایت و ایمیل می‌باشد.
درخواست POST به endpoint /developers ارسال می‌شود.

به روزرسانی توسعه‌دهنده:

فرم "update_dev_form" امکان وارد کردن شناسه توسعه‌دهنده و فیلدهای جدید را می‌دهد.
درخواست PUT به endpoint /developers/{developer_id} ارسال می‌شود.

حذف توسعه‌دهنده:

فرم "delete_dev_form" شناسه توسعه‌دهنده را دریافت و درخواست DELETE به endpoint /developers/{developer_id} دریافت اطلاعات توسعه‌دهنده:

فرم "get_dev_form" شناسه توسعه‌دهنده را گرفته و با ارسال درخواست GET، اطلاعات آن را نمایش می‌دهد.

Categories برای CRUD ❖

```
elif crud_option == "Create Category":
    st.markdown("### Create a New Category")
    with st.form("create_cat_form"):
        category = st.text_input("Category")
        submitted = st.form_submit_button("Create Category")
    if submitted:
        payload = {"category": category}
        response = requests.post(f"{api_base_url}/categories/", json=payload)
        if response.status_code in (200, 201):
            st.success("Category created successfully!")
        else:
            st.error(f"Failed to create category: {response.text}")

elif crud_option == "Update Category":
    st.markdown("### Update an Existing Category")
    with st.form("update_cat_form"):
        category_id = st.text_input("Category to Update")
        new_category = st.text_input("New Category Value (if updating)")
        submitted = st.form_submit_button("Update Category")
    if submitted:
        payload = {}
        if new_category:
            payload["category"] = new_category
        response = requests.put(f"{api_base_url}/categories/{category_id}", json=payload)
        if response.status_code == 200:
            st.success("Category updated successfully!")
        else:
            st.error(f"Failed to update category: {response.text}")
```

```

elif crud_option == "Delete Category":
    st.markdown("### Delete a Category")
    with st.form("delete_cat_form"):
        category_id = st.text_input("Category to Delete")
        submitted = st.form_submit_button("Delete Category")
    if submitted:
        response = requests.delete(f"{api_base_url}/categories/{category_id}")
        if response.status_code == 200:
            st.success("Category deleted successfully!")
        else:
            st.error(f"Failed to delete category: {response.text}")

elif crud_option == "Get Category":
    st.markdown("### Get Category Details")
    with st.form("get_cat_form"):
        category_id = st.text_input("Category to Retrieve")
        submitted = st.form_submit_button("Get Category")
    if submitted:
        response = requests.get(f"{api_base_url}/categories/{category_id}")
        if response.status_code == 200:
            cat_data = response.json()
            st.json(cat_data)
        else:
            st.error(f"Failed to retrieve category: {response.text}")

```

ایجاد دسته‌بندی:

فرم "create_cat_form" تنها شامل فیلد دسته‌بندی است.
درخواست POST /categories به ارسال می‌شود.

بهروزرسانی دسته‌بندی:

فرم "update_cat_form" شامل نام دسته‌بندی فعلی و مقدار جدید (در صورت بهروزرسانی) می‌باشد.

درخواست PUT endpoint /categories/{category_id} به ارسال می‌شود.

حذف دسته‌بندی:

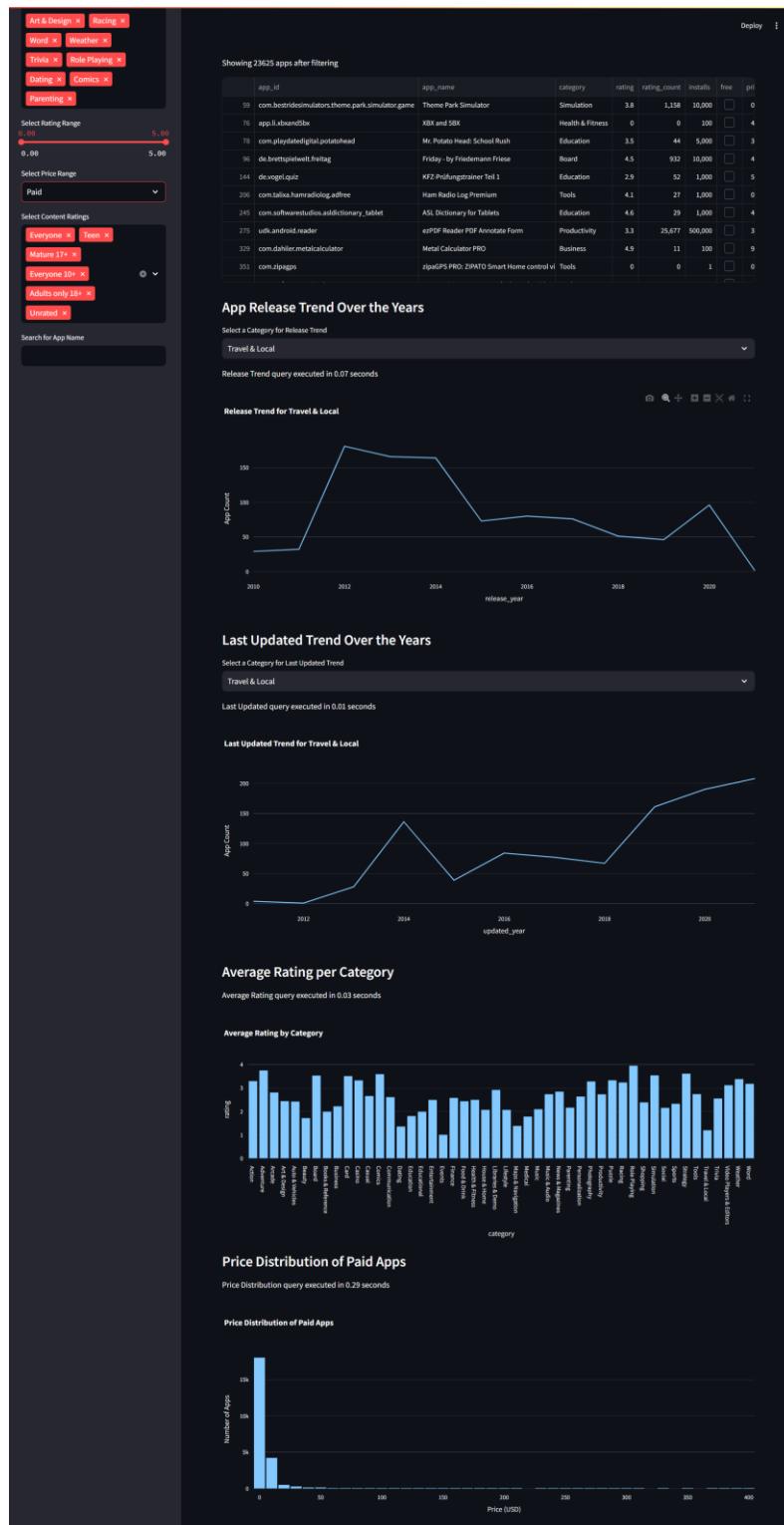
فرم "delete_cat_form" تنها نام دسته‌بندی مورد نظر را دریافت کرده و درخواست DELETE endpoint /categories/{category_id} به ارسال می‌کند.

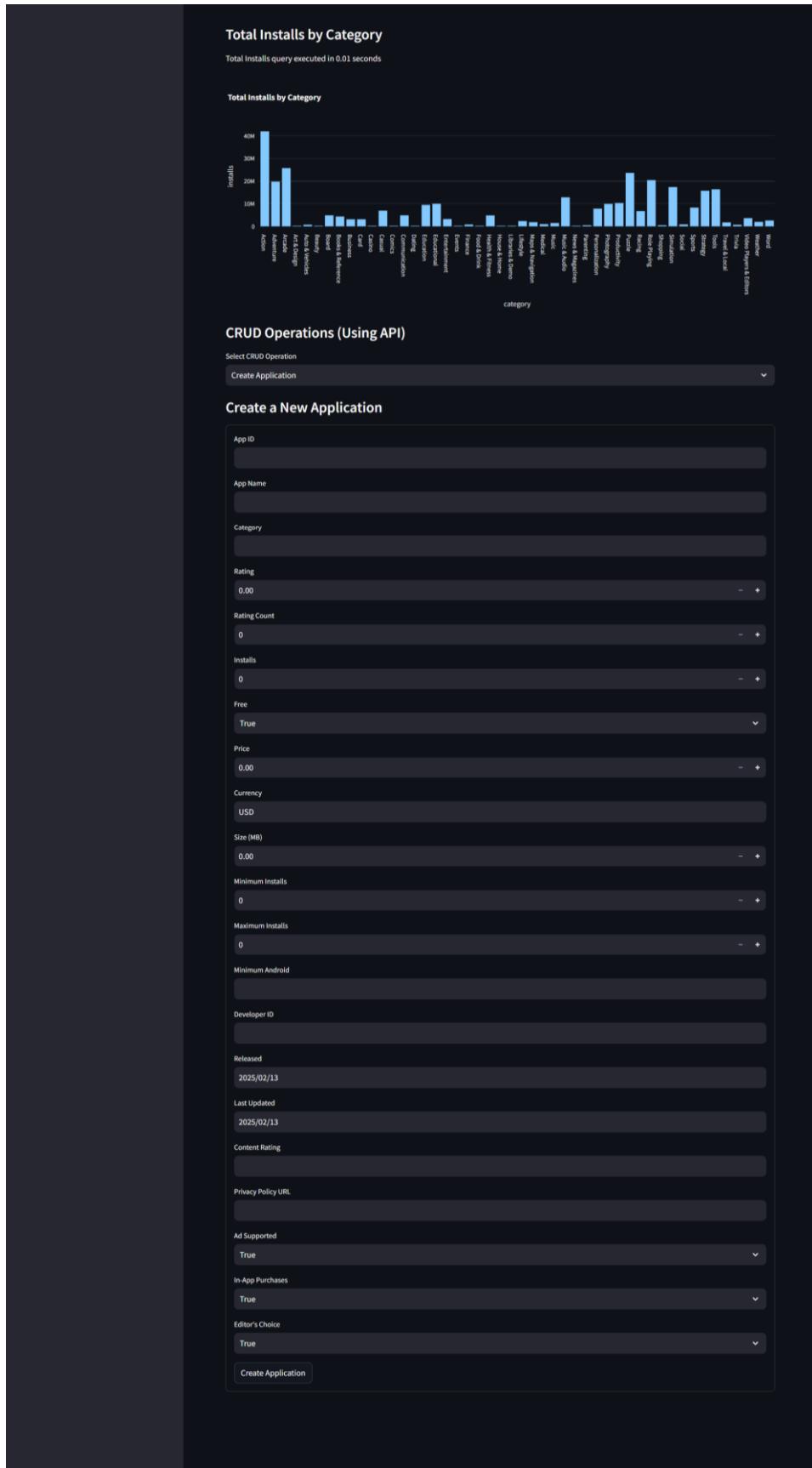
دريافت اطلاعات دسته‌بندی:

فرم "get_cat_form" با دریافت نام دسته‌بندی، اطلاعات آن را از طریق درخواست GET نمایش می‌دهد.

در هر یک از این عملیات‌ها، پس از ارسال درخواست HTTP API، بر اساس کد وضعیت (مانند ۲۰۰، ۴۰۴) پیام موفقیت یا خطا به کاربر نمایش داده می‌شود.

۶. تصاویر اپلیکیشن





۷. لینک های ویدیویی دمو و گیت هاب

➤ لینک ویدیویی دمو

- <https://screenrec.com/share/rI3GBZfg0N>

➤ لینک ریپازیتوری Github

- <https://github.com/alirza-gz/Google-Play-Store-Analytics-Optimization-Platform.git>

۸. منابع

- [PostgreSQL: Documentation](#)
- [Streamlit documentation](#)
- [Our Documentation | Python.org](#)