«به نام خدا»

دانشگاه فردوسی مشهد

دانشکده مهندسی

گروه مهندسی کامپیوتر

فاز اول و دوم پروژه Docker

نگارش

الهه متقین

زهرا سادات ترویج الاسلامی

استاد درس

دکتر شقایق ایزدپناه

مهر ماه 1400

Microservice Architecture:

سبک معماری میکروسرویس رویکردی برای توسعه یک برنامه کاربردی واحد به عنوان مجموعه‌ای از سرویس‌های کوچک است که هر کدام در فرآیند خاص خود اجرا می‌شوند و با مکانیسم‌های سبک، اغلب یک API با منبع HTTP ارتباط برقرار می‌کنند. این سرویس ها بر اساس قابلیت های تجاری ساخته شده اند و به طور مستقل توسط ماشین آلات استقرار کاملاً خودکار قابل استقرار هستند. حداقل مدیریت متمرکز این سرویس ها وجود دارد که ممکن است به زبان های برنامه نویسی مختلف نوشته شده باشند و از فناوری های مختلف ذخیره سازی داده ها استفاده کنند.

برای شروع توضیح سبک میکروسرویس، مقایسه آن با سبک یکپارچه مفید است: یک برنامه یکپارچه که به عنوان یک واحد ساخته شده است. برنامه های کاربردی سازمانی اغلب در سه بخش اصلی ساخته می شوند: یک رابط کاربری سمت سرویس گیرنده (شامل صفحات HTML و جاوا اسکریپت در حال اجرا در مرورگر بر روی ماشین کاربر) یک پایگاه داده (شامل بسیاری از جداول درج شده در یک مدیریت پایگاه داده مشترک و معمولا رابطه ای) و یک برنامه سمت سرور. برنامه سمت سرور درخواست های HTTP را مدیریت می کند، منطق دامنه را اجرا می کند، داده ها را از پایگاه داده بازیابی و به روز می کند، و نماهای HTML را برای ارسال به مرورگر انتخاب و پر می کند. این نرم افزار سمت سرور یکپارچه است - یک فایل اجرایی منطقی واحد. هر گونه تغییر در سیستم شامل ساخت و استقرار نسخه جدیدی از برنامه سمت سرور است.

چنین سرور یکپارچه راهی طبیعی برای نزدیک شدن به ساخت چنین سیستمی است. تمام منطق شما برای رسیدگی به یک درخواست در یک فرآیند اجرا می شود و به شما امکان می دهد از ویژگی های اساسی زبان خود برای تقسیم برنامه به کلاس ها، توابع و namespaces استفاده کنید. با کمی دقت، می‌توانید برنامه را روی لپ‌تاپ یک توسعه‌دهنده اجرا و آزمایش کنید، و از pipeline توسعه استفاده کنید تا مطمئن شوید که تغییرات به درستی آزمایش شده و در مرحله تولید قرار گرفته‌اند. می‌توانید با اجرای بسیاری از نمونه‌ها در پشت یک load-balancer، یکپارچه را به صورت افقی مقیاس کنید.

برنامه های یکپارچه می توانند موفق باشند، اما مردم به طور فزاینده ای از آنها احساس ناامیدی می کنند - به خصوص که برنامه های کاربردی بیشتری در فضای ابری مستقر می شوند. چرخه های تغییر با هم گره خورده اند - تغییری که در بخش کوچکی از برنامه ایجاد شده است، نیاز به بازسازی و استقرار کل یکپارچه دارد. با گذشت زمان، حفظ یک ساختار ماژولار خوب اغلب دشوار است، و حفظ تغییراتی که فقط باید روی یک ماژول در آن ماژول تأثیر بگذارد، سخت‌تر می‌شود. مقیاس بندی به جای بخشی از آن که به منابع بیشتری نیاز دارد، به مقیاس بندی کل برنامه نیاز دارد.

این ناامیدی ها منجر به سبک معماری میکروسرویس شده است: ساخت برنامه های کاربردی به عنوان مجموعه سرویس ها. علاوه بر این واقعیت که سرویس‌ها به طور مستقل قابل استقرار و مقیاس‌پذیری هستند، هر سرویس همچنین یک مرز ماژول ثابت را فراهم می‌کند، حتی اجازه می‌دهد تا سرویس‌های مختلف به زبان‌های برنامه‌نویسی مختلف نوشته شوند. آنها همچنین می توانند توسط تیم های مختلف مدیریت شوند.

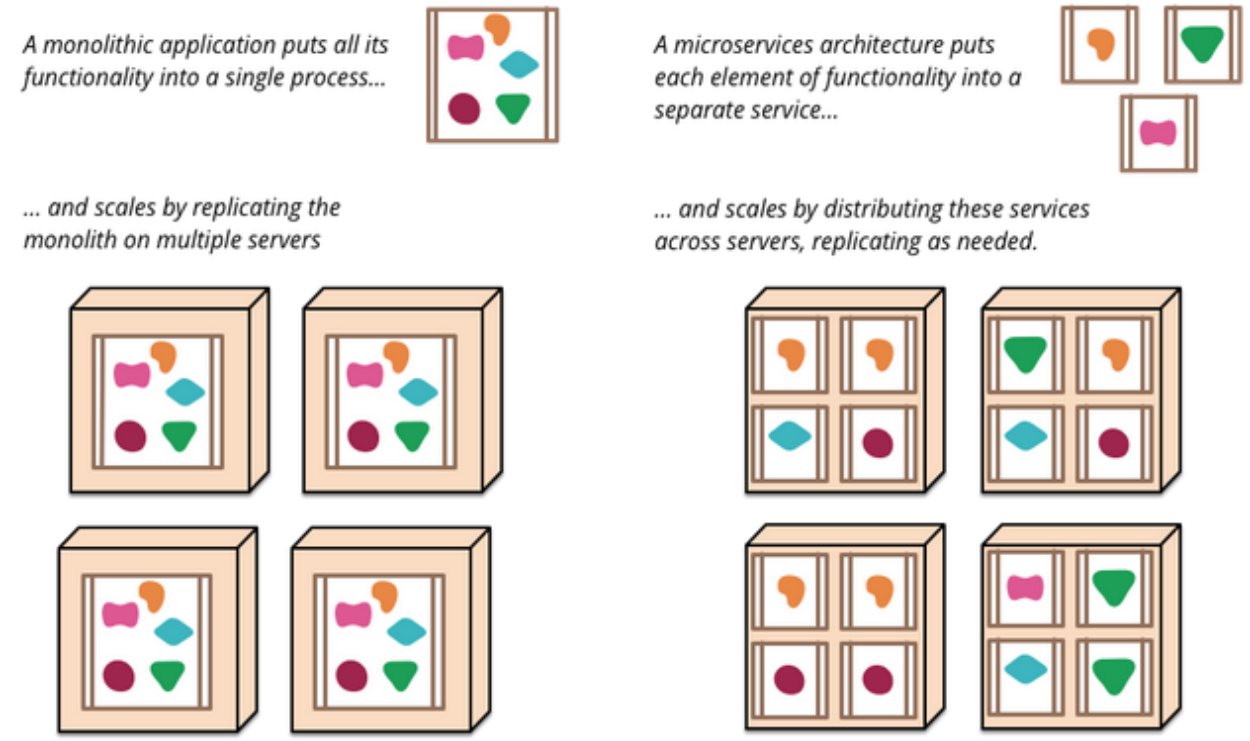


Figure 1: Monoliths and Microservices

*** فاز اول پروژه:**

REST API:

REST API (همچنین به عنوان RESTful API شناخته می شود) یک رابط برنامه نویسی کاربردی (API یا web API) است که با محدودیت های سبک معماری REST مطابقت دارد و امکان تعامل با سرویس های وب RESTful را فراهم می کند. REST مخفف انتقال حالت نمایشی است و توسط دانشمند کامپیوتر روی فیلدینگ ایجاد شده است.

API مجموعه ای از تعاریف و پروتکل ها برای ساخت و یکپارچه سازی نرم افزارهای کاربردی است. گاهی اوقات به عنوان قراردادی بین یک ارائه‌دهنده اطلاعات و یک کاربر اطلاعات نامیده می‌شود - که محتوای مورد نیاز مصرف‌کننده (تماس) و محتوای مورد نیاز تولیدکننده (پاسخ) را ایجاد می‌کند. به عنوان مثال، طراحی API برای یک سرویس آب و هوا می‌تواند مشخص کند که کاربر یک کد پستی ارائه می‌کند و تولیدکننده با یک پاسخ 2 قسمتی پاسخ می‌دهد، اولی دما بالا و دومی پایین است.

شما می توانید یک API را به عنوان یک واسطه بین کاربران یا مشتریان و منابع یا سرویس های وب که می خواهند دریافت کنند، در نظر بگیرید. همچنین راهی برای یک سازمان برای به اشتراک گذاشتن منابع و اطلاعات با حفظ امنیت، کنترل و احراز هویت - تعیین اینکه چه کسی به چه چیزی دسترسی دارد.

REST مجموعه ای از محدودیت های معماری است، نه یک پروتکل یا یک استاندارد. توسعه دهندگان API می توانند REST را به روش های مختلفی پیاده سازی کنند.

هنگامی که درخواست مشتری از طریق یک RESTful API انجام می شود، نمایشی از وضعیت منبع را به درخواست کننده یا نقطه پایانی منتقل می کند. این اطلاعات، یا نمایش، در یکی از چندین فرمت از طریق HTTP ارائه می‌شود: JSON (نشان‌گذاری شی جاوا اسکریپ)، HTML، XLT، Python، PHP یا متن ساده. JSON عموماً محبوب‌ترین فرمت فایلی است که می‌توان از آن استفاده کرد، زیرا هم برای انسان و هم برای ماشین‌ها قابل خواندن است.

نکته دیگری که باید در نظر داشت: header ها و پارامترها در روش‌های HTTP درخواست HTTP API RESTful نیز مهم هستند، زیرا حاوی اطلاعات شناسه مهمی در مورد فراداده درخواست، مجوز، شناسه منبع یکسان (URI)، حافظه پنهان، کوکی‌ها و بیشتر. سرصفحه‌های درخواست و سرفصل‌های پاسخ وجود دارد که هر کدام اطلاعات اتصال HTTP و کدهای وضعیت خاص خود را دارند.

برای اینکه یک API RESTful در نظر گرفته شود، باید با این معیارها مطابقت داشته باشد:

- یک معماری سرویس گیرنده-سرور متشکل از کلاینت ها، سرورها و منابع، با درخواست ها از طریق HTTP مدیریت می شود.
- ارتباط مشتری-سرور بدون تابعیت، به این معنی که هیچ اطلاعات مشتری بین درخواست‌های دریافت ذخیره نمی‌شود و هر درخواست جداگانه و غیر مرتبط است.
- داده های قابل ذخیره سازی که تعاملات مشتری و سرور را ساده می کند.
- یک رابط یکنواخت بین اجزاء به طوری که اطلاعات به شکل استاندارد منتقل می شود. این مستلزم آن است که:
  - منابع درخواستی قابل شناسایی و جدا از نمایندگی های ارسال شده به مشتری هستند.
  - منابع را می توان توسط مشتری از طریق نمایشی که دریافت می کند دستکاری کرد زیرا نمایش حاوی اطلاعات کافی برای انجام این کار است.
  - پیام‌های self-descriptive که به مشتری بازگردانده می‌شوند، اطلاعات کافی برای توصیف نحوه پردازش مشتری دارند.
  - hypermedia /hypertext در دسترس است، به این معنی که پس از دسترسی به یک منبع، مشتری باید بتواند از پیوندها برای یافتن سایر اقدامات موجود در حال حاضر استفاده کند.

- یک سیستم لایه ای که هر نوع سرور را سازماندهی می کند (کسانی که مسئول امنیت، تعادل بار و غیره هستند) شامل بازیابی اطلاعات درخواستی در سلسله مراتبی است که برای مشتری نامرئی است.

REST مجموعه‌ای از دستورالعمل‌ها است که می‌تواند در صورت نیاز پیاده‌سازی شود و APIهای REST را سریع‌تر و سبک‌تر کند، با مقیاس‌پذیری افزایش‌یافته - برای توسعه اینترنت اشیا (IoT) و برنامه‌های موبایل ایده‌آل است.

توضیحات کد:

در این پروژه از ما خواسته شده بود با توجه به دیتاست تاریخچه فروش بازی ها بر روی پلتفرم های متفاوت سرویس مرکزی شامل چند API را پیاده سازی کنیم.

برای این پروژه سه اپ RestAPI و Authorization و DataAnalysis ساختیم که برای هرکدام API های لازم را در views.py نوشته و در urls.py آنها را فراخوانی کرده. در نهایت هم فایل اصلی در urls.py لینک های برنامه ها را قرار دادیم. همچنین برای آغاز کار در برنامه RestAPI در models.py جدول دیتاست بازی ها را لود کرده و همچنین در Authorization جدول کاربران را ساخته ایم.

ساختار کلی API ها به این صورت است که هنگامی که کلاینت یک درخواست را می فرستد طبق url مورد نظر(مجموعه ای از درخواست ها را توسط postman بصورت یک collection ذخیره کرده ایم.)، پارامتر هایی مانند سال انتشار، پلتفرم و ... را وارد می کند. کاربر ها توسط ماژول APIView در فریم ورک rest احراز هویت شده و در صورت دسترسی، خروجی درخواست آنها با استفاده از تابع filter() که بازی ها را بر اساس پارامتر فرستاده شده فیلتر می کند و سپس توسط serializer ها خروجی درخواست را قابل خواندن برای انسان و تبدیل به فایل json. می کند نمایش داده می شود.

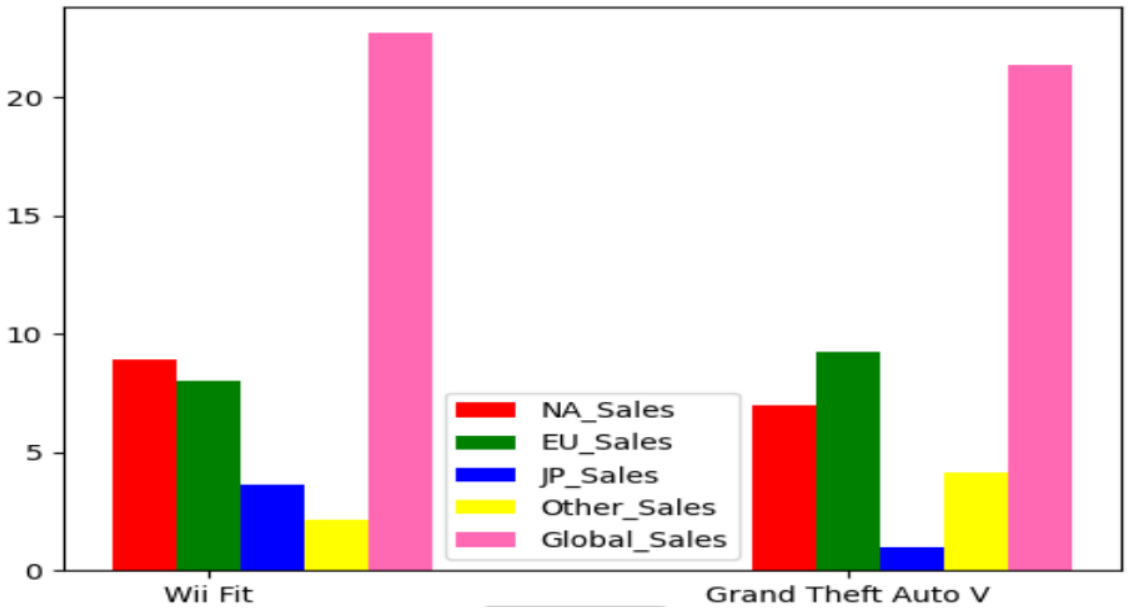
1. بازیابی اطلاعات بازی بر اساس رتبه: در httpRequest کاربر rank بازی موردنظر را وارد کرده و و اطلاعات آن بازی پس از فیلتر بر اساس رتبه بازی به صورت json. نمایش داده می شود.
2. بازیابی اطلاعات بازی بر اساس اسم کامل یا بخشی از اسم آن: در httpRequest کاربر نام کامل بازی یا قسمتی از آن را وارد کرده و اطلاعات تمام بازی های شامل این نام به صورت json. نمایش داده می شود.
3. N بازی برتر بر اساس هر پلتفرم: در httpRequest کاربر نام پلتفرم و یک عدد وارد کرده و اطلاعات N بازی برتر آن پلتفرم پس از فیلتر توسط نام پلتفرم (بر اساس رتبه) به صورت json. نمایش داده می شود.
4. N بازی برتر بر اساس سال: در httpRequest کاربر سال موردنظر و یک عدد وارد کرده و اطلاعات N بازی برتر آن سال پس از فیلتر توسط سال (بر اساس رتبه) به صورت json. نمایش داده می شود.
5. N بازی برتر بر اساس دسته بندی: در httpRequest کاربر ژانر موردنظر و یک عدد وارد کرده و اطلاعات N بازی برتر آن ژانر پس از فیلتر توسط نام ژانر (بر اساس رتبه) به صورت json. نمایش داده می شود.

6. 5 بازی پرفروش در یک سال مشخص برای یک پلتفرم تعیین شده: در httpRequest کاربر نام پلتفرم و یک سال وارد کرده و اطلاعات 5 بازی برتر آن سال و پلتفرم پس از فیلتر توسط نام این دو پارامتر (بر اساس فروش جهانی به صورت نزولی) به صورت json. نمایش داده می شود.
7. بازی هایی که فروش اروپایی آنها بیشتر از آمریکای شمالی بوده است: در httpRequest کاربر چیزی وارد نمی کند و یک لیست برای اضافه کردن بازی های با فروش اروپایی بیشتر از آمریکای شمالی ایجاد کرده که در خروجی اطلاعات این بازی ها به صورت json. نمایش داده می شود.

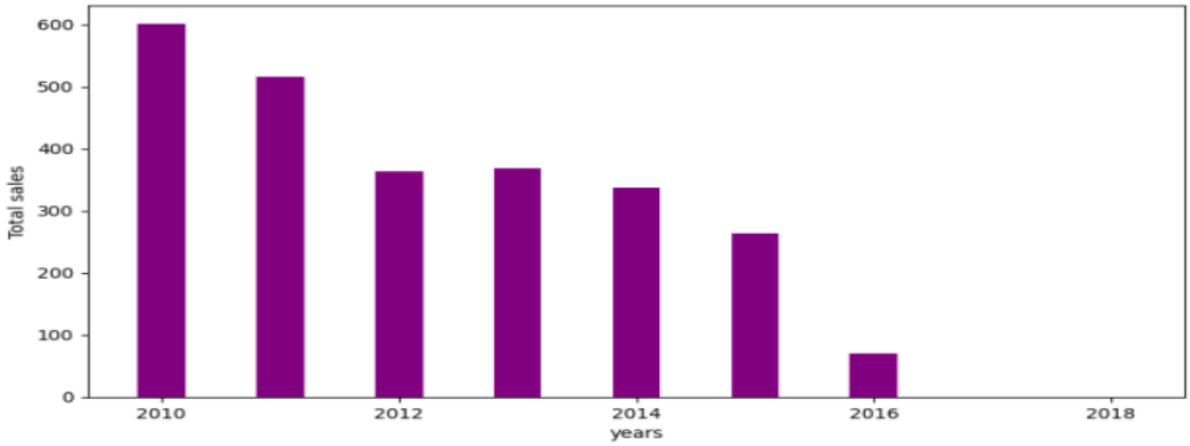
سرویس بعدی، سرویس تحلیل داده ها می باشد که در اپ DataAnalysis، API های موردنظر را به صورت نمودار در فرمت png. در پوشه /results ذخیره می شوند.

مانند سرویس مرکزی، ساختار کلی API ها به این صورت است که هنگامی که کلاینت یک درخواست را می فرستد طبق url مورد نظر(مجموعه ای از درخواست ها را توسط postman بصورت یک collection ذخیره کرده ایم.)، پارامتر هایی مانند سال انتشار، پلتفرم و ... را وارد می کند. کاربر ها توسط ماژول APIView در فریم ورک rest احراز هویت شده و در صورت دسترسی، درخواست آنها با استفاده از تابع filter() که بازی ها را بر اساس پارامتر فرستاده شده فیلتر می کند و سپس توسط serializer ها خروجی درخواست را قابل خواندن برای انسان و تبدیل به فایل json. می کند و این فایل json. توسط کتابخانه های numpy و matplotlib به صورت نمودار هایی انواع مختلف با محور های موردنظر تبدیل شده و در فرمت png. ذخیره شده.

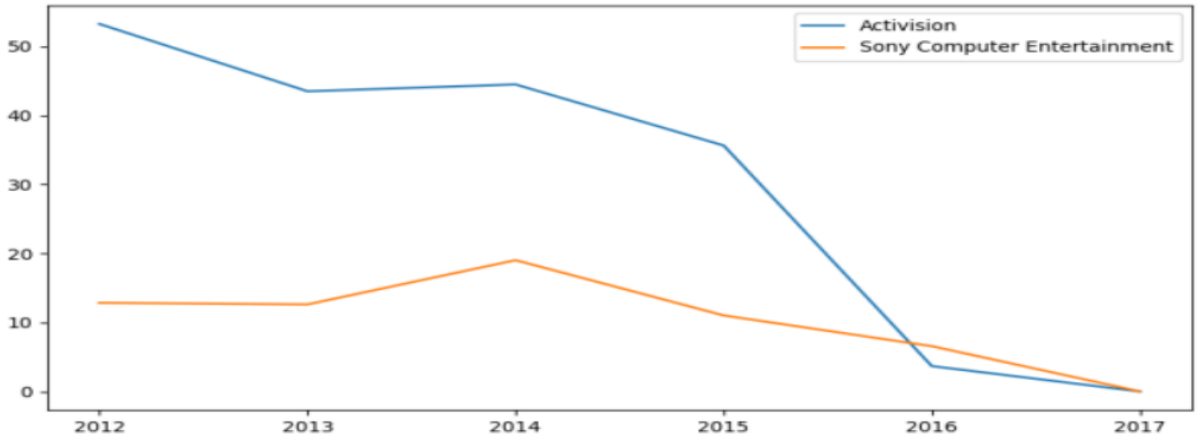
1. مقایسه فروش دو بازی بر روی یک نمودار: در httpRequest کاربر نام کامل دو بازی را وارد کرده و فایل json. حاصل از فیلتر کردن بر اساس نام این دو بازی که حاوی اطلاعات این دو بازی است را با نمودار میله ای که هر رنگ آن نشان دهنده فروش یک قسمت از جهان برای دو بازی است را نمایش داده می شود.



2. مقایسه مجموع فروش هر سال با دریافت یک بازه زمانی: در httpRequest کاربر یک بازه زمانی با سال شروع و پایان وارد کرده و لیست totals را با مقادیر مجموع فروش جهانی تمام بازی های هر سال در آن بازه زمانی را append کرده و سپس خروجی را به صورت نمودار میله ای با محورهای سال ها و totals رسم میکنیم.

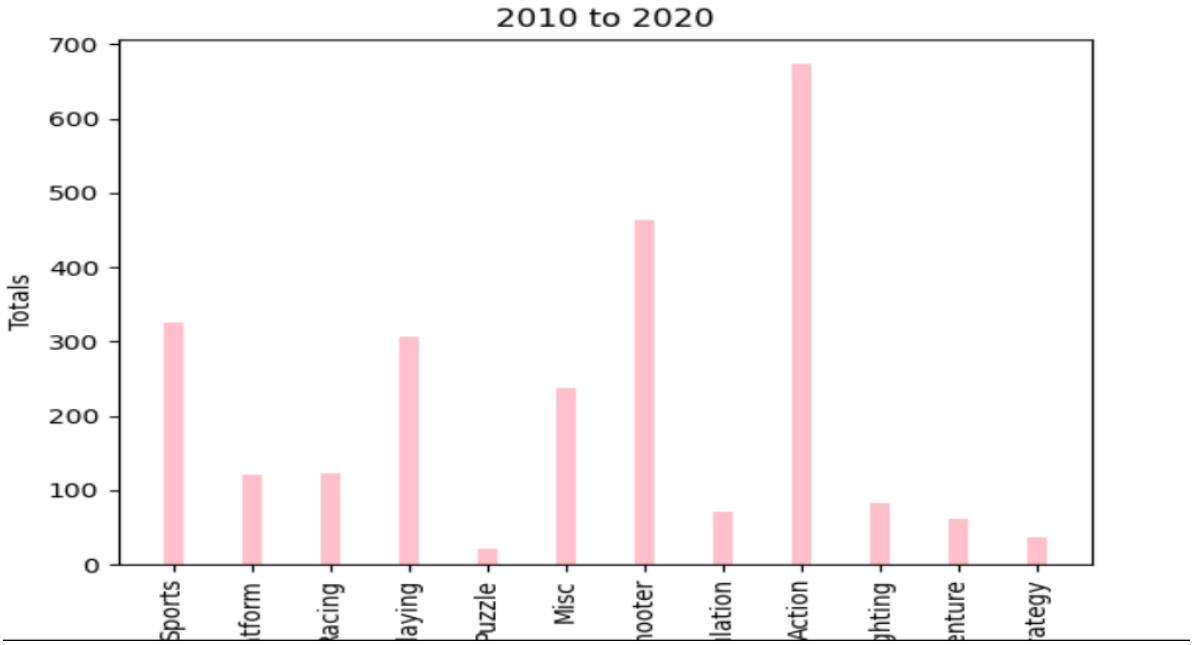


3. مقایسه فروش دو تولیدکننده در یک بازه زمانی مشخص: در httpRequest کاربر یک بازه زمانی با سال شروع و پایان و همچنین دو publisher وارد کرده و لیست totals1 و totals2 را با مقادیر مجموع فروش جهانی تمام بازی هایpublisher1 و publisher2 در هر سال در آن بازه زمانی را append کرده و سپس خروجی توسط دو نمودار با محورهای سال ها و total ها نمایش داده می شود.



4. مقایسه فروش دسته بندی های مختلف در یک بازه زمانی مشخص: در httpRequest کاربر یک بازه زمانی با سال شروع و پایان وارد کرده و لیست totals را با مقادیر مجموع فروش جهانی تمام بازی های یک ژانر در هر سال در آن بازه زمانی را append کرده و لیست x شامل مجموع فروش جهانی بازی های یک ژانر در این بازه است که

در نهایت نمودار میله ای با محورهای ژانر ها و مقدار فروش جهانی آن ها نمایش داده می شود.



**** فاز دوم پروژه:**

Docker:

Docker یک پلت فرم باز برای توسعه، shipping و اجرای برنامه ها است. Docker شما را قادر می سازد تا برنامه های کاربردی خود را از زیرساخت(insfrastructure) خود جدا کنید تا بتوانید نرم افزار را به سرعت تحویل دهید. با Docker، می توانید زیرساخت خود را به همان روشی که برنامه های خود را مدیریت می کنید، مدیریت کنید. با استفاده از روش‌های Docker برای ارسال، آزمایش و استقرار سریع کد، می‌توانید تأخیر بین نوشتن کد و اجرای آن را به میزان قابل توجهی کاهش دهید.

از داکر زمانی استفاده می شود که برنامه شما نیاز به تحویل سریع و مداوم، استقرار و scaling در پاسخگویی و همچنین اجرای workload های بیشتر بر روی سخت افزار دارد.

Images

یک image یک الگوی فقط خواندنی با دستورالعمل‌هایی برای ایجاد یک ظرف Docker است. اغلب، یک image بر اساس یک image دیگر، با برخی از سفارشی سازی های اضافی است. برای مثال، ممکن است image بسازید که بر اساس image اوبونتو باشد، اما وب سرور آپاچی و برنامه شما و همچنین جزئیات پیکربندی مورد نیاز برای اجرای برنامه شما را نصب کند.

Containers

کانتینر یک نمونه قابل اجرا از یک image است. می‌توانید با استفاده از Docker API یا CLI یک محفظه ایجاد، شروع، توقف، حرکت یا حذف کنید. می‌توانید یک ظرف را به یک یا چند شبکه متصل کنید، فضای ذخیره‌سازی را به آن متصل کنید، یا حتی یک تصویر جدید بر اساس وضعیت فعلی آن ایجاد کنید.

به طور پیش فرض، یک کانتینر به خوبی از سایر کانتینرها و ماشین میزبان آن جدا شده است. می‌توانید کنترل کنید که شبکه یک کانتینر، ذخیره‌سازی یا سایر زیرسیستم‌های زیرین از کانتینرهای دیگر یا از ماشین میزبان چقدر جدا هستند.

* در فاز دوم پروژه قصد داریم که از طریق Dockerfile و docker-compose، Image Docker تهیه کرده و به صورت یک Container اجرا کنیم. ابتدا باید docker و docker-compose را نصب کرد.

Dockerfile

Docker می تواند با خواندن دستورالعمل های یک Dockerfile تصاویر را به طور خودکار بسازد. Dockerfile یک سند متنی است که شامل تمام دستوراتی است که کاربر می تواند در خط فرمان برای جمع آوری یک تصویر فراخوانی کند. با استفاده از docker build، کاربران می توانند یک automated build ایجاد کنند که چندین دستورالعمل خط فرمان را پشت سر هم اجرا می کند.

در Dockerfile اطلاعاتی شامل:

- ورژن python 3.8 که base image پروژه است.
- environment های PYTHONUNBUFFERED و PYTHONSONTWRITEBYTECODE برابر 1 قرار می گیرند.
- RUN mkdir و WORKDIR همین برنامه و اپی که در آن هستیم.
- COPY کردن فایل requirements.txt برای استفاده از image و بالا آوردن پروژه در مکان های دیگر در مسیر برنامه
- و سپس RUN pip3 install –r requrements.txt تا نصب شود.
- RUN pip3 install gunicorn
- . COPY دایرکتوری را در مسیر برنامه کپی می کنیم.

Docker-Compose

Compose ابزاری برای تعریف و اجرای برنامه های Docker چند کانتینری است. با Compose، از یک فایل YAML برای پیکربندی سرویس های برنامه خود استفاده می کنید. سپس با یک دستور، تمام سرویس ها را از پیکربندی خود ایجاد و راه اندازی می کنید.

در docker-compose اطلاعاتی شامل ورژن django 3 و جزئیات سرویس ها قرار می گیرند. هر برنامه را باید در قسمت سرویس ها نوشت که اطلاعاتی مانند:

- . build شود
- نام container
- port هایی که بهم bind می شوند
- gunicorn –b 0.0.0.0:8000 RestAPI.wsgi روی 0.0.0.0 و پورت 8000 اجرا می شود.
- volume که برنامه را اجرا می کند و در صورت ایجاد تغییر در مکان موردنظر تغییرات اعمال شوند.