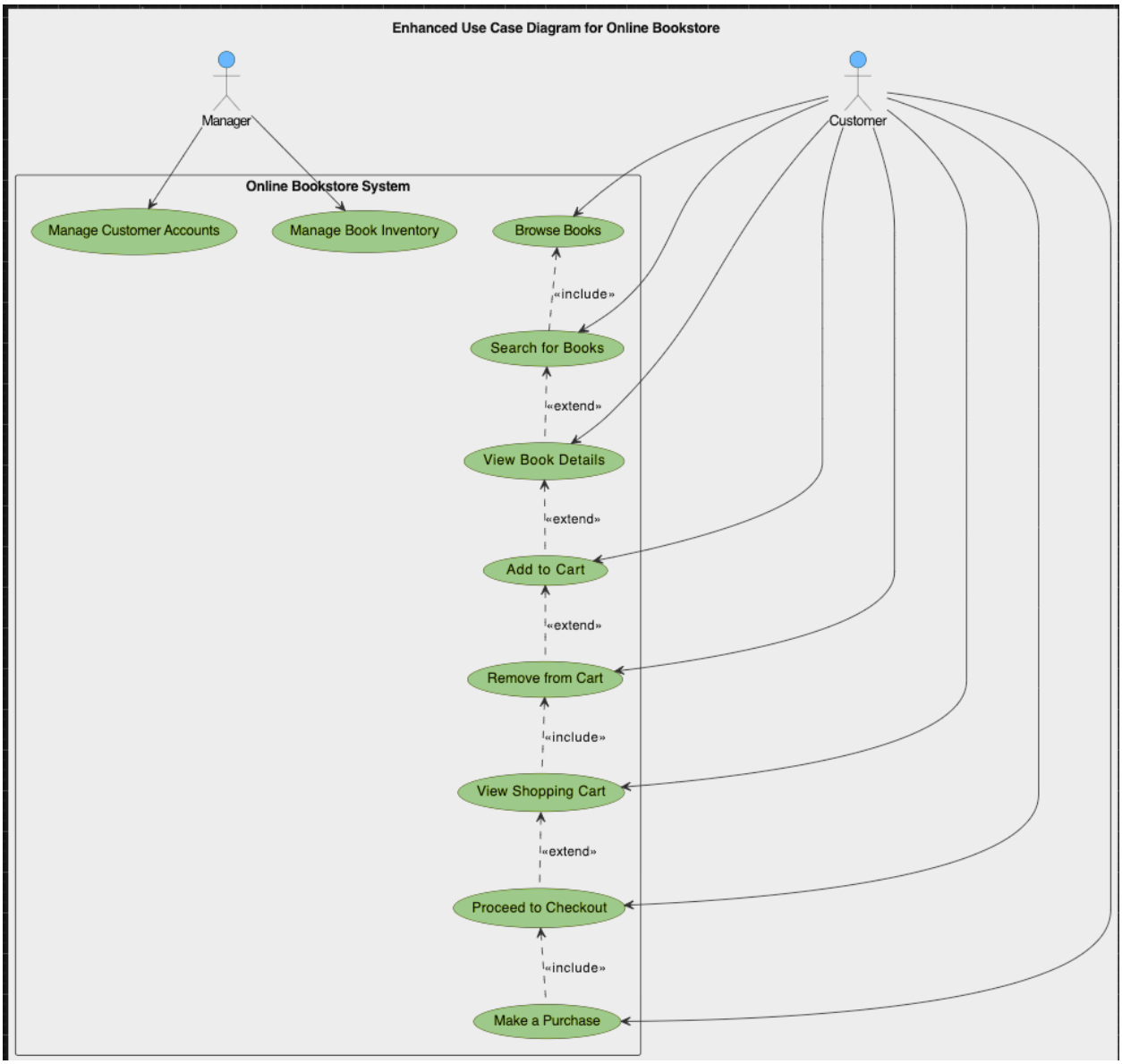
تمرین دوم مهندسی نرم افزار

عرفان رفیعی اسکویی – پوریا زمان وزیری – نیما لطف اللهیان – دریا تقوا

سوال اول)

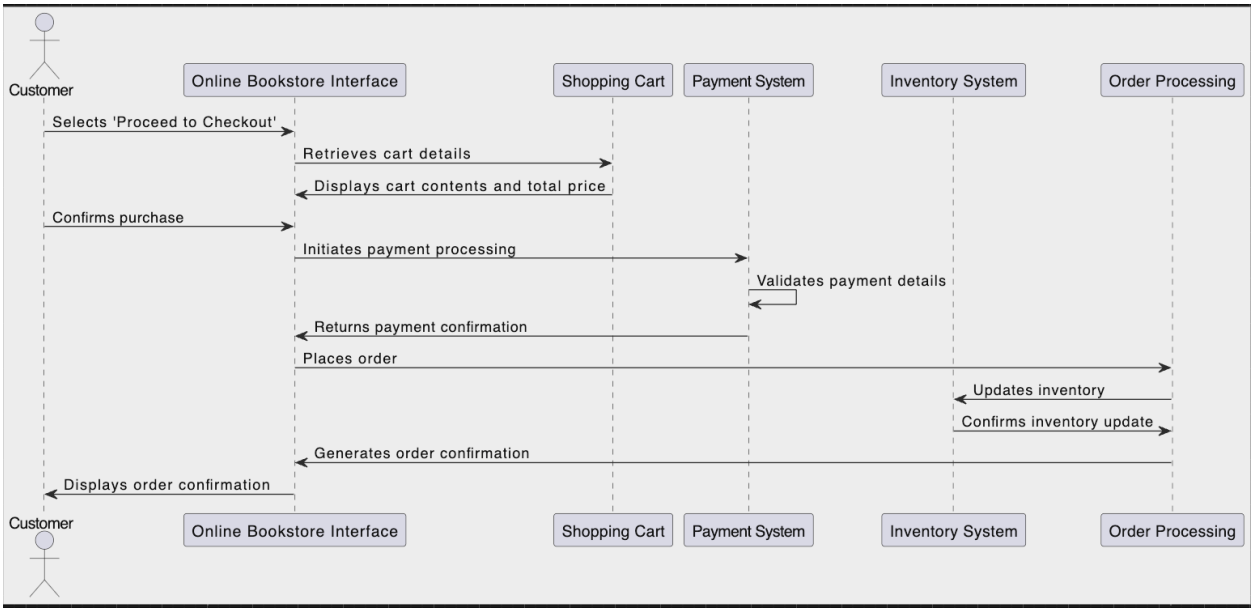
الف)



ب)

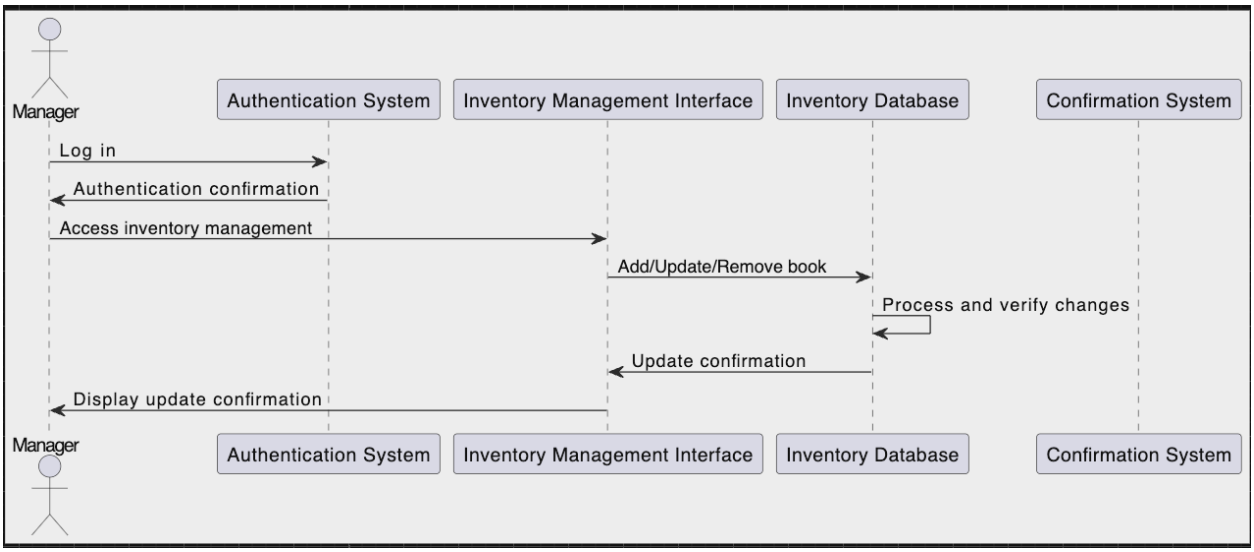
سناریو اول :

- مشتری کتاب‌ها را انتخاب می‌کند و به سبد خرید اضافه می‌کند.
- مشتری محتوای سبد خرید را مشاهده می‌کند.
- مشتری فرایند تسویه حساب را آغاز می‌کند.
- سیستم هزینه کل را محاسبه می‌کند، از جمله مالیات و هزینه ارسال.
- مشتری اطلاعات پرداخت و ارسال را وارد می‌کند.
- سرویس پرداخت پرداخت را پردازش می‌کند.
- سیستم پرداخت را تایید کرده و سفارش را ثبت می‌کند.
- سیستم موجودی کتاب‌ها را به‌روزرسانی می‌کند.
- مشتری تاییدیه خرید را دریافت می‌کند.



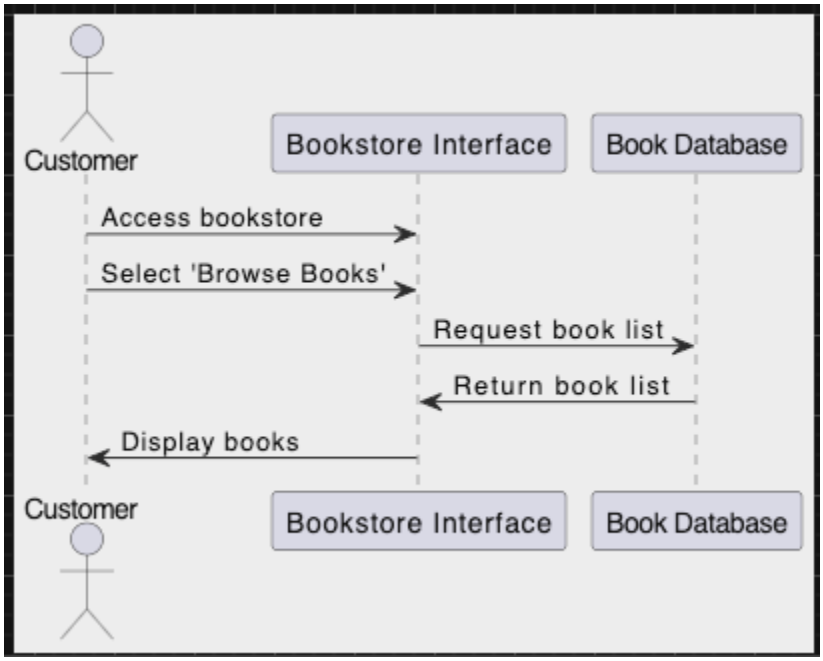
سناریو دوم:

- منیجر وارد سیستم می‌شود.
- منیجر به مدیریت موجودی کتاب‌ها وارد میشود.
- منیجر یکی از اقدامات زیر را انجام می‌دهد:
  - اضافه کردن کتاب جدید به موجودی.
  - به‌روزرسانی جزئیات یک کتاب موجود (مانند قیمت، موجودی، یا اطلاعات کتاب).
  - حذف کتاب از موجودی.
- سیستم تغییرات را تأیید و پردازش می‌کند.
- سیستم پایگاه داده موجودی را به‌روزرسانی می‌کند.
- منیجر تأییدیه به‌روزرسانی را دریافت می‌کند.



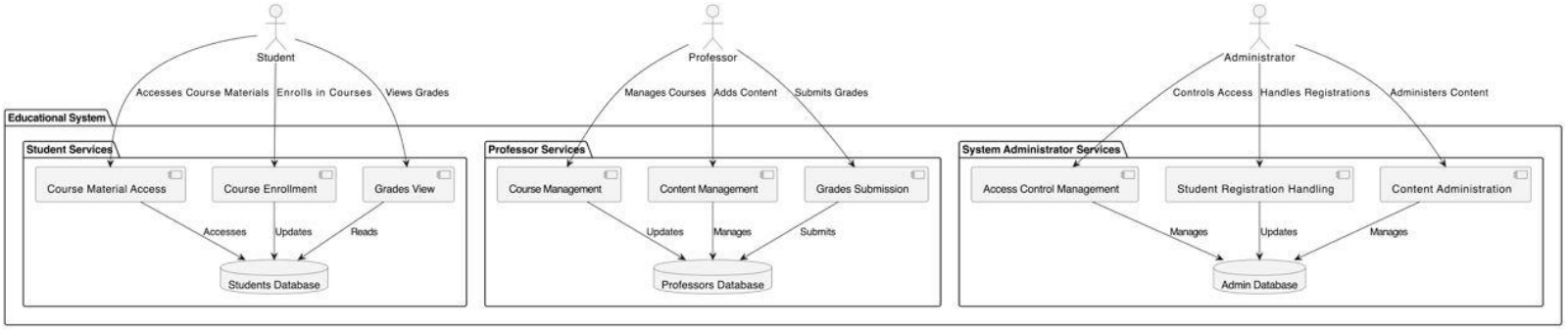
سناریو سوم:

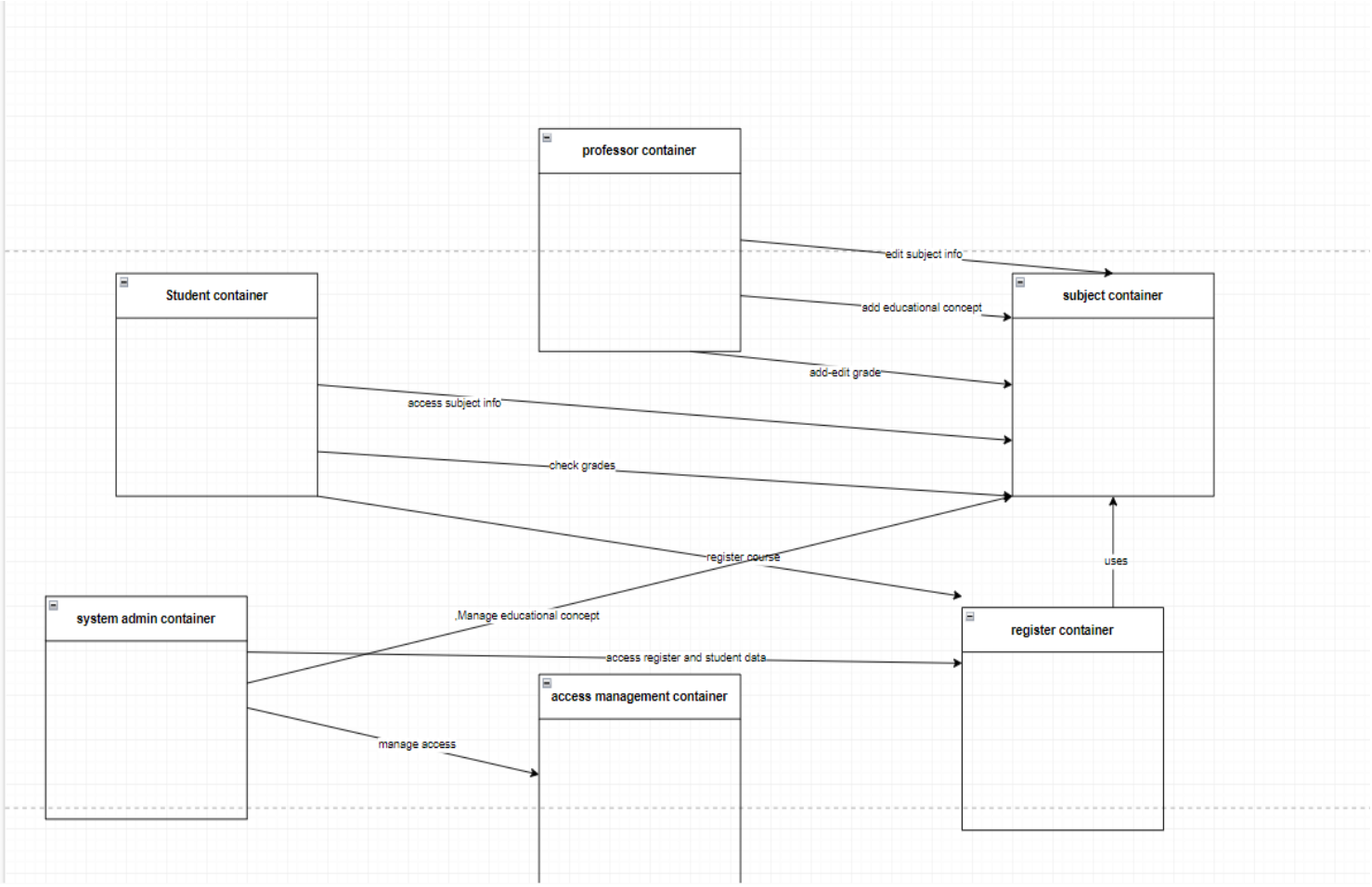
- مشتری به فروشگاه کتاب آنلاین دسترسی پیدا می‌کند.
- مشتری گزینه 'مرور کتاب‌ها' را انتخاب می‌کند.
- سیستم لیستی از کتاب‌های موجود را نمایش می‌دهد.



سوال دوم)

نمودار سطح container به شرح زیر است:





هر کانتینر با کانتینر بعدی بصورت مستقیم داده تبادل میکنند و در دیتابیس مخصوص آن کانتینر(که جزیی از آن کانتینر است) ذخیره میشود.

ب)

برتری این معماری نسبت به معماری مونولتیک این است که هر کانتینر بصورت مستقل عمل میکند و اگر در آینده نیازی به سرویسی داشتیم که تعداد دانشجویانش(برای مثال) 4 یا 5 برابر این سیستم بود تنها کافی است از کانتینر student چهار یا پنج instance بگیریم، یا خود کانتینر student را مودیفای کنیم. بدون اینکه در بقیه سیستم خللی ایجاد شود چون عملا ورودی ها و خروجی های ماکروسرویس همان است.

پ)

کانتینر ثبت نام:

فرآیند ثبت نام و دانش آموزان ثبت نام شده را درخود نگه میدارد. بعلاوه این سرویس از سرویس درس برای نگه داری واحدهای مربوط به دروس استفاده میکند.

کانتینر access management :

این کانتینر مدیریت دسترسی ها و permission ها را در خود نگه میدارد که از طریق مدیر سیستم قابل دسترسی است.

کانتینر subject :

این کانتینراطلاعات دروس را نگه داری میکند و توسط استاد مربوطه تغییر میدهد و میتواند توسط کانتینر ثبت نام استفاده شود.

ت)

1. **ارتباطات سنکرون (Synchronous Communication) :** در این نوع ارتباط، یک سرویس به طور مستقیم از سرویس دیگر درخواست اطلاعات می‌کند و منتظر پاسخ می‌ماند. این ارتباط معمولاً از طریق HTTP/HTTPS انجام می‌شود و اغلب از REST API یا گاهی SOAP استفاده می‌شود.
2. **ارتباطات آسنکرون (Asynchronous Communication) :**در این روش، سرویس‌ها بدون انتظار برای پاسخ فوری، پیام‌ها یا رویدادها را ارسال یا دریافت می‌کنند. این ارتباط معمولاً از طریق صف‌های پیام‌رسانی مانندRabbitMQ ، Kafkaیا از طریق سیستم‌های رویدادمحور مانند AWS SNS/SQS انجام می‌شود.
3. **gRPC :** این یک چارچوب ارتباطی است که توسط Google توسعه یافته و برای ارتباط بین سرویس‌ها در زمان واقعی استفاده می‌شود. از پروتکل HTTP/2 برای ارتباط سریع و کارآمد استفاده می‌کند.

انتخاب پروتکل ارتباطی بستگی به نیازهای خاص هر سیستم دارد. مثلا اگر پاسخ سریع و تعاملی مورد نیاز است، HTTP/HTTPS با REST یا gRPC توصیه می‌شود. برای سناریوهایی که نیاز به مقیاس‌پذیری بالا و تحمل خطا دارند، ارتباطات آسنکرون مانند کار با صف‌های پیام‌رسانی یا سیستم‌های رویدادمحور مناسب‌تر هستند.

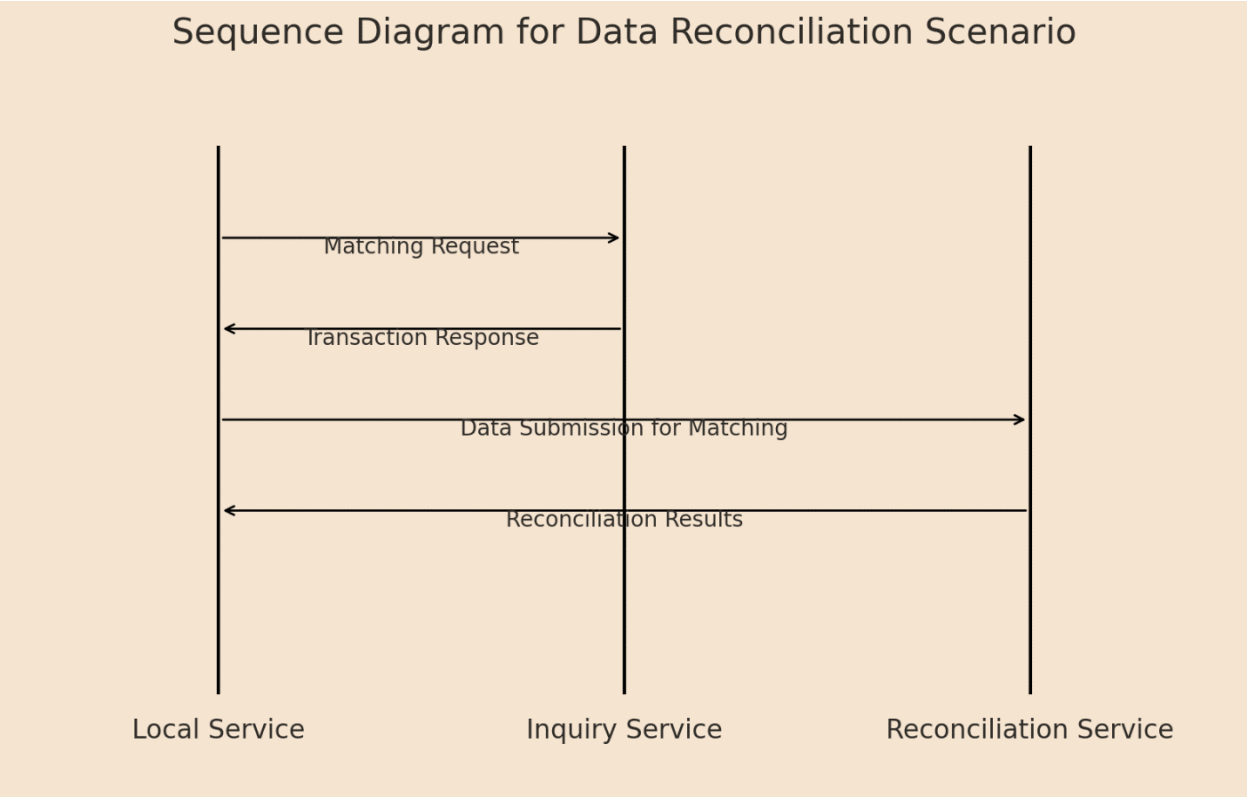
سوال سوم)

الف) برای ترسیم معماری ارتباطی بین سرویس محلی، سرویس inquiry و سرویس مغایرت‌گیری، ما می‌توانیم از یک نمودار UML (زبان مدل‌سازی یکپارچه) استفاده کنیم. نمودار UML مناسب برای این سناریو یک نمودار (Sequence Diagram) است که جریان ارتباطی و ترتیب فرآیندها را در طول زمان نشان می‌دهد.

در این نمودار، ما سه فعالیت اصلی داریم:

1. **Local Service** : این سرویس مسئولیت نگهداری و مدیریت داده‌های محلی و ارسال درخواست‌ها برای تطبیق داده‌ها را بر عهده دارد.
2. **Inquiry Service** : این سرویس توسط بانک ارائه می‌شود و پاسخ‌های مربوط به تراکنش‌های بانکی را ارائه می‌دهد.
3. **Reconciliation Service** : این سرویس وظیفه تطبیق داده‌های دریافتی از سرویسinquiry با داده‌های موجود در سرویس محلی را دارد و در صورت وجود تناقض، این تناقض‌ها را ثبت می‌کند.

در sequence diagram، ارتباط بین این سه سرویس و چگونگی انجام فرآیندها در زمان نشان داده می‌شود. حال نمودار مورد نظر را ترسیم میکنیم :



در Sequence Diagram بالا، فرآیند ارتباطی بین سرویس‌های مختلف نشان داده شده است:

1. **Matching Request** : "Local Service" درخواستی را برای تطبیق داده‌ها به "Inquiry Service" ارسال می‌کند.
2. **Transaction Response** : "Inquiry Service" پاسخ مربوط به تراکنش‌های بانکی را به "Local Service" بازمی‌گرداند.
3. **Data Submission for Matching** : "Local Service" داده‌های دریافتی را همراه با داده‌های محلی خود به "Reconciliation Service" ارسال می‌کند.
4. **Reconciliation Results** : "Reconciliation Service" نتایج تطبیق را به "Local Service" بازمی‌گرداند و در صورت وجود تناقض، این تناقض‌ها در جدول مغایرت ثبت می‌شوند.

این نمودار نمایشی از چگونگی ارتباط و ترتیب عملیات بین سرویس‌های مختلف در سناریوی مغایرت‌گیری داده هارا مشخص می‌کند.

ب)

برای پاسخگویی در مورد معماری و مدل ارتباطی مورد استفاده برای رفع مشکل failed شدن درخواست‌ها، می‌توانیم به چند موضوع کلیدی اشاره کنیم:

معماری ارتباطی:

1. **سرویس‌گرا(Service-Oriented Architecture - SOA) :** با توجه به سناریو، استفاده از معماری سرویس‌گرا می‌تواند مفید باشد. در این مدل، هر سرویس) مانند سرویس محلی، سرویس Inquiry و سرویس مغایرت‌گیری (به صورت مستقل عمل می‌کند و از طریقAPI ها یا پروتکل‌های مبتنی بر وب با یکدیگر ارتباط برقرار می‌کنند.

2. **میکروسرویس‌ها(Microservices) :** این مدل می‌تواند به خصوص برای سیستم‌های بزرگ و پیچیده با حجم بالای تراکنش‌ها مناسب باشد. هر بخش از سیستم به عنوان یک میکروسرویس مستقل عمل می‌کند که امکان انعطاف‌پذیری بیشتر و مدیریت بهتر خطاها را فراهم می‌آورد.

روش‌های مدیریت خطاها و درخواست‌های failed :

1. **تکرار مجدد درخواست‌ها(Retry Mechanism) :** یکی از روش‌های رایج برای مدیریت درخواست‌هایی که به دلایل مختلف موفق به اجرا نمی‌شوند، استفاده از مکانیزم تکرار مجدد است. در این روش، در صورت شکست یک درخواست، سیستم به صورت خودکار درخواست را پس از مدت زمان مشخصی دوباره ارسال می‌کند.
2. **صف‌بندی درخواست‌ها(Queueing Mechanism) :** استفاده از صف‌ها برای مدیریت درخواست‌ها یک روش موثر دیگر است. در این روش، درخواست‌هایی که به هر دلیلی نمی‌توانند فوراً پردازش شوند، در یک صف قرار می‌گیرند و به ترتیب پردازش می‌شوند. این کار به کاهش فشار بر سیستم و جلوگیری از ازدحام درخواست‌ها کمک می‌کند.
3. **مدیریت خطاها(Error Handling) :** پیاده‌سازی روش‌های پیشرفته مدیریت خطا که شامل شناسایی، ثبت و رسیدگی به خطاهای مختلف است، برای جلوگیری از شکست سیستم در شرایط اضطراری ضروری است.
4. **مانیتورینگ و آلارم‌دهی(Monitoring and Alerting) :** ایجاد سیستم‌های نظارتی برای تشخیص و اطلاع‌رسانی سریع در مورد خطاها و نقص‌ها به تیم‌های مربوطه برای اقدامات سریع و مؤثر.

با این توضیحات، سیستم طراحی شده باید قادر باشد به صورت مؤثر با failed شدن درخواست‌ها مقابله کند و تضمین کند که تمام تراکنش‌ها به درستی پردازش و تطبیق داده می‌شوند.

پ)

برای کاهش سربار ارتباطی بین سرویس محلی و سرویسinquiry ، چندین رویکرد و تکنیک می‌تواند مفید باشد:

1. (Caching) :

- **کشینگ در سمت کلاینت** :ذخیره پاسخ‌های متداول یا تکراری در سرویس محلی می‌تواند به کاهش تعداد درخواست‌های ارسالی به سرویس inquiry کمک کند.
- **کشینگ در سمت سرور** :ذخیره داده‌هایی که به طور مکرر پرس و جو می‌شوند در سرویسinquiry ، می‌تواند سرعت پاسخ‌گویی به درخواست‌ها را افزایش دهد.

2. (Batch Processing) :

- ارسال درخواست‌ها در دسته‌های بزرگتر به جای درخواست‌های فردی می‌تواند به کاهش تعداد ارتباطات کمک کند. بچینگ می‌تواند به ویژه در سناریوهایی که درخواست‌ها می‌توانند به صورت انبوه و غیرزمان‌حساس پردازش شوند مفید باشد.

3. فشرده‌سازی داده‌ها(Data Compression) :

- فشرده‌سازی داده‌های ارسالی و دریافتی بین سرویس محلی و سرویس inquiry می‌تواند حجم داده‌های منتقل شده را کاهش دهد و به کاهش بار شبکه کمک کند.

4. استفاده از پروتکل‌های ارتباطی کارآمد:

- انتخاب پروتکل‌هایی مانند gRPC یا MQTT که برای موارد استفاده با حجم بالا و کارایی بالا طراحی شده‌اند، می‌تواند مفید باشد.

5. توسعه API های کارآمد:

- طراحیAPI ها به گونه‌ای که اجازه می‌دهد درخواست‌های کمتر و با داده‌های مرتبط‌تر ارسال شود. به عنوان مثال، استفاده از GraphQL برای دریافت دقیقاً داده‌های مورد نیاز در هر درخواست.

6. ترتیب‌بندی و اولویت‌بندی درخواست‌ها:

- تعیین اولویت برای درخواست‌های حیاتی و تأخیر در درخواست‌های کم‌اهمیت‌تر می‌تواند به مدیریت بهتر پهنای باند و منابع کمک کند.

7. مانیتورینگ و بهینه‌سازی مداوم:

- نظارت دائمی بر عملکرد سیستم و انجام بهینه‌سازی‌های مداوم بر اساس داده‌های جمع‌آوری شده، برای حفظ کارایی و کاهش سربار ضروری است.

با اجرای این راهکارها، می‌توان اطمینان حاصل کرد که سیستم شما به شکلی کارآمد عمل می‌کند و سربار ارتباطی بین سرویس‌های مختلف به حداقل می‌رسد.