فهرست مسائل

پاسخ مسئله‌ی ۱.

الف

در این سناریو، الگویی که می‌تواند کمک کند الگوی مدارشکن (Circuit-Breaker-Pattern) است. این الگو به گونه‌ای طراحی شده است که با شناسایی خطاهای مکرر یا تاخیرهای طولانی در پاسخ‌دهی، ارتباط با سرویس معیوب را به طور موقت قطع کند و از ارسال درخواست‌های جدید به سرویس معیوب جلوگیری نماید.

مزایای الگوی مدارشکن:

- جلوگیری از مشکلات آبشاری: با قطع موقت ارتباط، بار اضافی از روی سرویس معیوب برداشته شده و از گسترش مشکلات به سایر سرویس‌ها جلوگیری می‌شود.
- افزایش تاب‌آوری سیستم: به دلیل کاهش وابستگی مستقیم به سرویس معیوب، سرویس‌های دیگر می‌توانند به عملکرد خود ادامه دهند.
- بازیابی تدریجی: مدارشکن به صورت دوره‌ای وضعیت سرویس معیوب را بررسی می‌کند و در صورت بهبود، ارتباط مجدد را امکان‌پذیر می‌سازد.
- مدیریت بهتر منابع: با جلوگیری از ارسال درخواست‌های بی‌فایده به سرویس معیوب، منابع سیستم بهینه‌تر استفاده می‌شود.

الگوی مدارشکن می‌تواند به عنوان یک بخش مهم از طراحی معماری میکروسرویس به کار گرفته شود تا تاب‌آوری و پایداری سیستم را افزایش دهد.

ب

۱. **استفاده از داده‌های کش:** اطلاعات موجودی کالاها را در یک سیستم کش قابل اطمینان ذخیره کنید. در صورت عدم دسترسی به سرویس «مدیریت موجودی»، از این داده‌ها برای ارائه پاسخ به کاربران استفاده شود.

۲. **پاسخ‌های Fallback :** هنگام خرابی سرویس، پاسخ‌های پیش‌فرضی برای کاربران ارائه شود؛ مثلاً نمایش پیامی مانند «اطلاعات موجودی در حال حاضر در دسترس نیست، لطفاً بعداً تلاش کنید».

۳. **به‌روزرسانی غیر هم‌زمان موجودی:** در صورتی که امکان بررسی موجودی در لحظه وجود ندارد، سفارش را ثبت کرده و عملیات بررسی موجودی و تأیید نهایی پرداخت را به صورت غیرهم‌زمان مدیریت کنید.

۴. **نمایش پیغام شفاف به کاربران:** به کاربران اعلام کنید که سیستم با اختلال مواجه است و آن‌ها را از وضعیت به‌روز نگه دارید. این شفافیت به کاهش نارضایتی کاربران کمک می‌کند.

۵. **مانیتورینگ و اطلاع‌رسانی:** با استفاده از ابزارهای مانیتورینگ، وضعیت سرویس‌ها را نظارت کنید و در صورت خرابی یا تأخیر، به تیم‌های مربوطه اطلاع دهید تا مشکلات سریع‌تر رفع شوند.

۶. **طراحی تجربه‌ی کاری جایگزین:** در صورت خرابی سرویس، به کاربران امکان مشاهده‌ی سایر اطلاعات یا محصولات مشابه را بدهید تا آن‌ها همچنان بتوانند تجربه‌ی مثبتی از سیستم داشته باشند.

۷. **سیستم انتظار و رزرو:** در صورت عدم دسترسی به اطلاعات موجودی، امکان رزرو کالا برای کاربر فراهم شود و پس از رفع مشکل، وضعیت سفارش به‌روز شود.

این اقدامات می‌تواند تجربه‌ی کاربران را حتی در صورت وقوع خرابی حفظ کند و از کاهش رضایت یا اعتماد آن‌ها به سیستم جلوگیری نماید.

ج

برای مدیریت و نظارت بهتر بر وضعیت سرویس معیوب و بازگرداندن ارتباط به حالت عادی، می‌توان از راه‌حل‌های زیر استفاده کرد:

۱. مانیتورینگ سلامت سرویس: یک سیستم مانیتورینگ پیشرفته برای نظارت مداوم بر وضعیت سرویس معیوب پیاده‌سازی کنید. ابزارهایی مانند Prometheus و Grafana می‌توانند شاخص‌های کلیدی عملکرد (KPIs) مانند نرخ خطا، زمان پاسخ‌دهی، و نرخ موفقیت درخواست‌ها را پایش کنند. با این روش، هرگونه اختلال سریعاً شناسایی شده و تیم‌های فنی می‌توانند اقدامات لازم را انجام دهند.

۲. استفاده از مکانیزم بازگرداندن تدریجی (Circuit-Breaker-Recovery) : مدارشکن می‌تواند به صورت دوره‌ای وضعیت سرویس معیوب را بررسی کند. به این صورت که پس از یک دوره‌ی انتظار، مدار به حالت نیمه‌باز تغییر می‌کند و تعداد محدودی درخواست آزمایشی به سرویس ارسال می‌شود. اگر پاسخ‌ها موفقیت‌آمیز بودند، مدارشکن به حالت بازگشته و ارتباط سرویس به صورت کامل بازیابی می‌شود. این مکانیزم امکان بازگشت تدریجی و پایدار سرویس را فراهم می‌آورد.

این دو راه‌حل در کنار هم می‌توانند به تشخیص سریع مشکلات، کاهش زمان خرابی، و بازگرداندن ارتباط به حالت عادی کمک کنند.

پاسخ مسئله‌ی ۲.

الف

کیفیت عملکرد (Performance-Quality)

اقدامات ارزیابی:

- **تست بار (Load-Testing)** : شبیه‌سازی شرایطی که هزاران کاربر همزمان از نرم‌افزار استفاده می‌کنند تا توانایی پاسخگویی سیستم سنجیده شود.
- **تست استرس (Stress-Testing)** : بررسی عملکرد نرم‌افزار در شرایطی که بار از حد معمول بالاتر است (تا مرز شکست سیستم) برای شناسایی نقاط ضعف.
- **تست مقیاس‌پذیری (Scalability-Testing)** : ارزیابی توانایی نرم‌افزار برای افزایش یا کاهش منابع برای مدیریت تعداد کاربران بیشتر.
- **مانیتورینگ زمان پاسخ (Response-Time-Monitoring)** : اندازه‌گیری زمان پاسخ‌دهی به درخواست‌های کاربران و شناسایی مواردی که از حد مجاز کندتر هستند.

ابزارها: استفاده از ابزارهایی مانند **JMeter**، **LoadRunner** یا **Apache-Benchmark** برای انجام آزمون‌ها.

کیفیت تطبیق (Compliance-Quality)

اقدامات ارزیابی:

- **تست امنیت (Security-Testing)** : بررسی آسیب‌پذیری‌های امنیتی مانند تزریق SQL ، حملات XSS ، و ضعف‌های احراز هویت.
- **تست تطابق قانونی (Regulatory-Compliance-Testing)** : ارزیابی انطباق با استانداردها و قوانین بانکی مانند GDPR، PCI DSS، و ISO ۲۷۰۰۱.
- **بررسی احراز هویت و دسترسی** : اطمینان از اینکه مکانیسم‌های احراز هویت قوی و سطوح دسترسی مناسب برای کاربران مختلف وجود دارد.
- **بررسی ثبت رویدادها (Audit-Logging)** : اطمینان از اینکه همه رویدادهای حساس مانند ورود به سیستم یا تراکنش‌ها ثبت و نگهداری می‌شوند.

ابزارها: استفاده از ابزارهای تست امنیت مانند **OWASP-ZAP**، **Burp-Suite**، یا **Nessus**. همچنین بهره‌گیری از چک‌لیست‌های قانونی برای تطبیق با قوانین و استانداردها.

تفاوت بین کیفیت عملکرد و کیفیت تطبیق

- **کیفیت عملکرد:** بر توانایی نرم‌افزار برای ارائه خدمات بهینه در شرایط مختلف (مانند بار زیاد) تمرکز دارد.
- **کیفیت تطبیق:** به رعایت استانداردهای امنیتی و قانونی مربوط به داده‌های بانکی و کاربران ارتباط دارد.

ب

۱. آزمون‌های کیفیت عملکرد (Performance-Quality-Tests)

- **تست بار (Load-Testing)** : بررسی عملکرد سیستم تحت بار عادی و پیش‌بینی‌شده برای اطمینان از پاسخ‌دهی مناسب.
- **تست استرس (Stress-Testing)** : اعمال بار بیشتر از حد معمول برای شناسایی نقاط ضعف سیستم و اندازه‌گیری میزان تحمل آن.
- **تست مقیاس‌پذیری (Scalability-Testing)** : ارزیابی توانایی سیستم در افزایش منابع (مانند سرورها) برای مدیریت کاربران بیشتر.
- **تست زمان پاسخ :(Response-Time-Testing)** بررسی مدت زمانی که سیستم برای پاسخ به درخواست‌ها صرف می‌کند.
- **تست ظرفیت (Capacity-Testing)** : تعیین حداکثر تعداد کاربران یا حجم تراکنش‌هایی که سیستم می‌تواند پشتیبانی کند.

۲. آزمون‌های کیفیت تطبیق (Compliance-Quality-Tests)

- **تست امنیت (Security-Testing)** : شناسایی آسیب‌پذیری‌ها مانند تزریق SQL، حملات XSS و نقص‌های احراز هویت.
- **تست تطابق قانونی (Regulatory-Compliance-Testing)** : اطمینان از رعایت استانداردها و مقررات مانند **GDPR**، **DSS PCI**، و **ISO۲۷۰۰۱**.
- **تست نفوذ (Penetration-Testing)** : شبیه‌سازی حملات برای ارزیابی میزان مقاومت سیستم در برابر تهدیدهای امنیتی.
- **بررسی رمزنگاری (Encryption-Testing)** : ارزیابی الگوریتم‌ها و مکانیسم‌های رمزنگاری برای حفاظت از داده‌ها.
- **تست ثبت وقایع (Audit-Log-Testing)** : اطمینان از ثبت دقیق و کامل فعالیت‌های کاربران و تراکنش‌های حساس.

ابزارهای مرتبط:

- برای **کیفیت عملکرد:** ابزارهایی مانند **JMeter**، **LoadRunner**، و **Apache-Benchmark**.
- برای **کیفیت تطبیق:** ابزارهایی مانند **OWASP-ZAP**، **Suite Burp**، و **Nessus**.

ج

- **کیفیت عملکرد (Performance-Quality) :** کیفیت عملکرد به توانایی نرم‌افزار در ارائه خدمات بهینه و کارآمد تحت شرایط مختلف، از جمله بار زیاد، تمرکز دارد. این نوع کیفیت با عوامل زیر ارزیابی می‌شود:
  - سرعت پاسخ‌دهی (Response-Time)
  - مقیاس‌پذیری (Scalability)
  - تحمل بار زیاد (Load-Tolerance)
  - پایداری در شرایط بحرانی (System-Stability)

  عدم توجه به این کیفیت می‌تواند منجر به کندی یا از کار افتادن سیستم شود.
- **کیفیت تطبیق (Compliance-Quality) :** کیفیت تطبیق به انطباق نرم‌افزار با استانداردهای امنیتی، قانونی، و صنعتی مرتبط است. این نوع کیفیت با عوامل زیر ارزیابی می‌شود:
  - رعایت قوانین و مقررات
  - امنیت داده‌ها و حفاظت از حریم خصوصی کاربران
  - ثبت و گزارش‌گیری دقیق وقایع (Audit-Logging)
  - مدیریت صحیح دسترسی‌ها (Access-Management)

  عدم رعایت این کیفیت می‌تواند منجر به جرایم قانونی، نقض حریم خصوصی، و کاهش اعتماد کاربران شود.

خلاصه تفاوت:

- **کیفیت عملکرد** به بهره‌وری و کارایی سیستم در پاسخ به کاربران تمرکز دارد.
- **کیفیت تطبیق** به رعایت استانداردها و تضمین امنیت و قانونی بودن نرم‌افزار ارتباط دارد.

د

۱. عدم رعایت کیفیت عملکرد (Performance-Quality) :

- **کندی سیستم:** زمان پاسخ‌دهی طولانی می‌تواند باعث ایجاد نارضایتی و از دست رفتن اعتماد کاربران شود.
- **قطع ارتباط یا خرابی:** اگر نرم‌افزار تحت بار زیاد از کار بیفتد، کاربران ممکن است قادر به تکمیل تراکنش‌های خود نباشند، که این موضوع باعث کاهش تجربه کاربری می‌شود.
- **عدم پایداری:** خرابی‌های مکرر یا ناپایداری سیستم ممکن است کاربران را به سمت استفاده از سرویس‌های رقیب سوق دهد.

۲. عدم رعایت کیفیت تطبیق (Compliance-Quality) :

- **نقض حریم خصوصی:** افشای داده‌های حساس کاربران مانند اطلاعات بانکی یا شخصی می‌تواند منجر به کاهش اعتماد کاربران به نرم‌افزار شود.
- **نقض قوانین:** عدم رعایت مقررات مانند **GDPR** یا **PCI-DSS** ممکن است به جریمه‌های سنگین و بدنامی برند منجر شود.
- **ناامنی داده‌ها:** کاربران ممکن است احساس کنند که اطلاعات آن‌ها در معرض خطر قرار دارد و از استفاده از نرم‌افزار صرف‌نظر کنند.

نتیجه کلی:

کیفیت عملکرد: مشکلات مرتبط با عملکرد می‌تواند منجر به نارضایتی آنی کاربران و کاهش استفاده از نرم‌افزار شود.
کیفیت تطبیق: عدم رعایت تطبیق می‌تواند اثرات طولانی‌مدت‌تری مانند از دست رفتن اعتبار، مشکلات قانونی، و کاهش اعتماد عمومی ایجاد کند.

پاسخ مسئله‌ی ۳.

الف

۱. ریسک وابستگی به کتابخانه متن‌باز:

- **شرح ریسک:** کتابخانه متن‌باز ممکن است دارای مشکلاتی مانند باگ‌های غیرمنتظره، توقف پشتیبانی توسط توسعه‌دهندگان، یا آسیب‌پذیری‌های امنیتی باشد که بر قابلیت اطمینان و امنیت سامانه تأثیر منفی بگذارد.
- **پتانسیل اثرگذاری:** در صورت وقوع این ریسک، عملکرد سامانه مختل شده یا توسعه و نگهداری پروژه با چالش مواجه خواهد شد.
- **راهبرد کاهش ریسک:**
  - ارزیابی دقیق کتابخانه پیش از انتخاب.
  - استفاده از نسخه‌های پایدار و به‌روزرسانی‌های مداوم.
  - برنامه‌ریزی برای جایگزین‌سازی کتابخانه در صورت بروز مشکلات.

۲. ریسک وابستگی به شرکت خارجی برای پردازش داده‌های آموزشی:

- **شرح ریسک:** شرکت خارجی ممکن است به دلیل مشکلاتی مانند قطعی سرویس، تأخیر در پاسخ‌دهی، تغییر در سیاست‌های ارائه خدمات یا افزایش هزینه‌ها، عملکرد سامانه را تحت تأثیر قرار دهد.
- **پتانسیل اثرگذاری:** این ریسک می‌تواند موجب کاهش کیفیت یا قطع شدن دسترسی به داده‌های آموزشی شود که تأثیر منفی بر دقت سامانه مبتنی بر هوش مصنوعی خواهد داشت.
- **راهبرد کاهش ریسک:**
  - تعریف SLA (توافق‌نامه سطح خدمات) با شرکت خارجی.
  - تهیه نسخه پشتیبان از داده‌های آموزشی.
  - بررسی و ایجاد جایگزین‌های احتمالی برای فراخوانی API .

این دو ریسک باید به دقت مدیریت شوند تا از تأثیرات منفی آن‌ها بر پروژه جلوگیری شود.

ب

ریسک ۱: وابستگی به کتابخانه متن‌باز

- **برنامه کاهش:**
  - بررسی و ارزیابی مداوم کتابخانه متن‌باز، شامل آزمایش و تحلیل امنیتی پیش از استفاده.
  - به‌کارگیری مستندات دقیق از نسخه‌های استفاده‌شده و اطمینان از سازگاری کتابخانه با نیازهای پروژه.
  - شناسایی یک یا دو جایگزین مناسب برای کتابخانه و انجام آزمایش‌های اولیه بر روی آن‌ها.
- **برنامه واکنش:**
  - در صورت بروز مشکل (مانند توقف پشتیبانی)، سریعاً به یکی از کتابخانه‌های جایگزین مهاجرت کنید.
  - تیم توسعه را آماده نگه دارید تا در صورت شناسایی باگ یا آسیب‌پذیری، تغییرات لازم را اعمال کرده یا موقتاً از بخش‌هایی از سامانه که وابسته به کتابخانه است چشم‌پوشی کنند.

ریسک ۲: وابستگی به شرکت خارجی برای پردازش داده‌های آموزشی

- **برنامه کاهش:**

– عقد قرارداد با شرکت خارجی شامل توافق‌نامه سطح خدمات (SLA) که تضمین‌هایی در مورد دسترسی، کیفیت خدمات، و زمان پاسخ ارائه دهد.

– تهیه نسخه‌های پشتیبان از داده‌های آموزشی و ایجاد یک پایگاه داده داخلی برای ذخیره موقت داده‌ها در صورت بروز اختلال.

– ارزیابی و انتخاب یک یا چند ارائه‌دهنده جایگزین برای پردازش داده‌ها.

- **برنامه واکنش:**

– در صورت قطعی یا خرابی خدمات، از داده‌های ذخیره‌شده در سیستم داخلی استفاده کرده و سامانه را موقتاً به حالت پردازش محلی (local-processing) تغییر دهید.

– در صورت تغییر سیاست‌ها یا افزایش غیرمنتظره هزینه‌ها، به ارائه‌دهنده جایگزین مهاجرت کنید یا بخشی از پردازش داده‌ها را به صورت داخلی انجام دهید.

پاسخ مسئله‌ی ۴.

الف

استفاده از اصول انتزاع تنها محدود به سطح کدهای پروژه نیست و می‌تواند در لایه‌های مختلف سیستم نرم‌افزاری، از جمله مدیریت داده‌ها، مورد استفاده قرار گیرد. انتزاع به معنای پنهان‌سازی جزئیات غیرضروری و تمرکز بر مفاهیم یا عملیات اصلی است. در سیستم مدیریت سفارش فودلاین، انتزاع می‌تواند در سطوح زیر اعمال شود:

۱. انتزاع در سطح کد:

- طراحی ماژول‌ها و کلاس‌ها با استفاده از الگوهای طراحی (مانند Facade یا Adapter) برای پنهان‌سازی پیچیدگی و ارائه‌ی رابط‌های ساده.
- جداسازی منطق کسب‌وکار (Business-Logic) از لایه‌های دیگر، مانند رابط کاربری و دسترسی به داده‌ها، به منظور تسهیل در نگهداری و توسعه.
- استفاده از لایه‌های سرویس (Service-Layers) برای انتزاع منطق پیچیده و ارائه‌ی عملکردهای ساده و شفاف به لایه‌های بالاتر.

۲. انتزاع در سطح داده‌ها:

- **طراحی پایگاه داده انتزاعی:** به جای دسترسی مستقیم به جداول پایگاه داده، از مدل‌های داده‌ای و واسط‌هایی (مانند ORM ها نظیر Entity-Framework یا Hibernate) استفاده کنید تا جزئیات نحوه ذخیره‌سازی داده‌ها پنهان شوند.
- **انتزاع در مدل‌سازی داده‌ها:** تعریف موجودیت‌های مفهومی مانند Order، Customer و MenuItem به جای نمایش مستقیم جداول پایگاه داده.
- **پنهان‌سازی فرمت داده‌ها:** ارائه اطلاعات مشتریان یا سفارش‌ها به صورت انتزاعی و استاندارد، بدون نمایش جزئیات ساختار داده‌ها یا فرمت‌های داخلی مانند XML یا JSON .

۳. انتزاع در سطح معماری:

- جداسازی لایه‌های مختلف سیستم (مانند لایه‌های ارائه، منطق کسب‌وکار و داده) به گونه‌ای که تغییرات در یک لایه تأثیری بر لایه‌های دیگر نداشته باشد.
- استفاده از هاAPI برای ارائه خدمات به ماژول‌های داخلی یا سیستم‌های خارجی بدون نمایش جزئیات داخلی.

نتیجه‌گیری: اصل انتزاع نه تنها در کدنویسی، بلکه در طراحی داده‌ها و معماری سیستم نیز قابل اجراست و می‌تواند به ساده‌سازی، پایداری و قابلیت نگهداری پروژه کمک کند.

ب

۱. طراحی ماژولار و میکروسرویس:

- هر بخش از سیستم به صورت یک زیرسیستم مستقل طراحی شود که وظیفه مشخص و محدودی دارد (مانند زیرسیستم مدیریت سفارش‌ها، ردیابی وضعیت سفارش، و مدیریت اطلاعات مشتری).
- زیرسیستم‌ها به صورت مستقل از یکدیگر توسعه، اجرا و استقرار داده شوند.
- ارتباط میان زیرسیستم‌ها از طریق هایAPI استاندارد یا پیام‌محور (Message Queues) انجام شود تا وابستگی‌ها کاهش یابد.

۲. استفاده از اصول SOLID در طراحی:

- اصل Single-Responsibility: هر زیرسیستم فقط یک مسئولیت مشخص داشته باشد (مثلاً زیرسیستم مدیریت سفارش تنها مسئول ذخیره و پردازش سفارش‌ها باشد).
- اصل Interface-Segregation: زیرسیستم‌ها فقط وابسته به رابط‌های ضروری باشند و نیازی به استفاده از جزئیات غیرضروری دیگر سیستم‌ها نداشته باشند.

۳. پیاده‌سازی ارتباط‌های غیرهم‌زمان:

- استفاده از مکانیزم‌های غیرهم‌زمان مانند Kafka یا RabbitMQ برای ارسال و دریافت پیام‌ها بین زیرسیستم‌ها.
- این روش امکان جداسازی زمانی زیرسیستم‌ها و افزایش مقیاس‌پذیری را فراهم می‌کند.

۴. طراحی برای شکست:

- استفاده از الگوهایی مانند مدارشکن (Circuit-Breaker) برای شناسایی و جلوگیری از خطاهای آبشاری در سیستم.
- پشتیبان‌گیری و بازیابی اطلاعات در زیرسیستم‌هایی که با داده‌های حساس کار می‌کنند (مانند اطلاعات مشتریان).

مثال پیاده‌سازی: فرض کنید تیم می‌خواهد سه زیرسیستم اصلی برای مدیریت سفارش، ردیابی وضعیت سفارش، و مدیریت اطلاعات مشتری طراحی کند:

۱. زیرسیستم مدیریت سفارش مسئول ذخیره‌سازی و پردازش سفارش‌های مشتریان است. این زیرسیستم درخواست‌های ثبت سفارش را از طریق یک RESTful-API دریافت می‌کند و اطلاعات را در پایگاه داده ذخیره می‌کند.

۲. زیرسیستم ردیابی وضعیت سفارش از یک سیستم پیام‌محور مانند Kafka استفاده می‌کند تا به‌صورت غیرهم‌زمان وضعیت سفارش را از زیرسیستم‌های دیگر (مانند زیرسیستم ارسال غذا) دریافت کند و آن را به‌روز کند.

۳. زیرسیستم مدیریت اطلاعات مشتری وظیفه مدیریت داده‌های مشتریان را برعهده دارد و تنها از طریق یک رابط API اطلاعات را به سایر زیرسیستم‌ها ارائه می‌دهد، بدون اینکه دسترسی مستقیم به پایگاه داده فراهم باشد.

نتیجه: این رویکرد نه تنها استقلال عملکردی هر زیرسیستم را حفظ می‌کند، بلکه مقیاس‌پذیری و قابلیت اطمینان سیستم را نیز تضمین می‌کند. هر زیرسیستم می‌تواند جداگانه مقیاس‌دهی یا به‌روزرسانی شود بدون اینکه بر عملکرد زیرسیستم‌های دیگر تأثیر بگذارد.