باسمه تعالی

دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

فاز نهایی پروژه

نام و نام خانوادگی نویسندگان:

حوری دهش

رضوان عابدین ورامین

ساحل مهدوی‌نژاد

بهمن ۱۴۰۲

۱. از چه فرمتی برای مدیریت داده ها استفاده می‌کنید. دلیل خود را بیان و تحلیل کنید. (برای دلایل خود نیازمندی های Nonfunctional را هم در نظر بگیرید.)

فرمت انتخابی ما Object relational DBMS است. در زیر دلایلی برای درستی انتخاب خود ارائه کردیم:

- کارایی و کارامدی سیستم : Object relational DBMS داده های پیچیده را به خوبی مدیریت می کنند و سازماندهی و بازیابی اطلاعات را آسان تر و سریع تر می کنند.
- امنیت: آنها داده ها را با استفاده از ویژگی هایی مانند رمزگذاری برای کنترل اینکه چه کسی می تواند به چه چیزی دسترسی داشته باشد، ایمن نگه می دارد.
- Portability: آن ها مانند صحبت کردن به یک زبان مشترک (SQL) هستند که انتقال داده ها و برنامه ها را بین سیستم های مختلف کامپیوتری آسان می کنند.
- قابلیت مقیاس پذیری: آن ها می‌توانند افزایش حجم داده ها را بدون کاهش قابل توجه سرعت کنترل کنند، و اطمینان حاصل کنند که سیستم شما سریع می‌ماند.
- دسترس پذیری: آنها در فعال ماندن و کار کردن خوب هستند، حتی اگر برخی از قسمت‌های سیستم مشکل داشته باشند، بنابراین همیشه می‌توان در صورت نیاز به داده‌های خود دسترسی داشت.
- و در آخر چون از object orientation هم تا حدی پشتیبانی میکنند این امکان را به ما میدهند تا از مزیت های برنامه نویسی شی گرایی نیز بهره مند شویم.

دلیل رد سایر فرمت ها:

Sequential and Random Access Files :

مهم ترین چالش فایل ها در رابطه با مقیاس پذیری است، به ویژه در مواجه با سیستم های بزرگ و پیچیده و با افزایش حجم داده ها، هرچند فایل ها در زمینه امنیت نیز آنچنان براورده کننده نیاز ما نیستند، همچنین در صورت ایجاد مشکل ریکاوردی داده از آن ها زمان بر و سخت است.

Relational Databases :

با وجود این که تقریبا همه ی نیاز های non functional را برآورده میکنند اما چون object orientation را ساپورت نمیکنند این گزینه نیز رد میشود. همچنین این نوع فقط داده های simple را پشتیبانی میکند.

Object-Oriented Databases : به دلیل داشتن تکنولوژی رو به رشد و سخت بودن به دست آوردن مهارت های آن، این گزینه نیز رد میشود.

NoSQL Data Stores : به دلیل داشتن تکنولوژی رو به رشد و سخت بودن به دست آوردن مهارت های آن، این گزینه نیز رد میشود، همچنین به دلیل عدم پشتیبانی از مکانیزن locking در زمینه به روز رسانی داده و این که ممکن است مقادیر متفاوتی برای یک شی در قسمت های مختلف وجود داشته باشد این گزینه نیز رد میشود.

۲. بررسی کنید آیا Class Diagram ارائه شده توسط گروه شما در فاز Structural Modelling نیازمند بهینه سازی می‌باشد. برای پاسخ خود دلیل بیاورید (مهمترین کار این است که خصوصیات Coupling و Cohesion را در کلاس‌های طراحی شده بررسی کنید.)

در حالت کلی کلاس دیاگرامی که اکنون داریم نیازمند بهینه سازی نیست بخاطر دلایل زیر:

Cohesion:

کلاس های شخص، سفارشات، غذا، نظرات و امتیازات از Cohesion بالایی برخوردارند چون تمام کلاس یک کار واحد انجام می دهند ولی کلاس های کاربر و فروشنده Cohesion متوسطی دارند و کلاس پشتیبان تلفنی هم از Cohesion کمی برخوردار است که در قسمت های زیر هر کدام توضیح داده شده است:

۱. شخص:

این کلاس اطلاعات مربوط به یک شخص از جمله جزئیات شخصی و عملکردهای مربوط به حساب را در خود گنجانده است یا به عبارت دیگر مدیریت حساب را انجام میدهد پس از Cohesion بالایی برخوردار است.

۲. کاربر:

این کلاس از Cohesion متوسطی برخوردار است بخاطر تابع هایی مانند بازیابی رمز عبور، فراموشی رمز عبور و ارسال رمز تصادفی که در کلاس فروشنده هم وجود دارد ولی اگر بخوایم Cohesion این کلاس را بالا ببریم باید این سه تابع را از این کلاس به کلاس دیگری منتقل بکنیم که فروشنده هم بتواند از آن ارث بری بکند ولی این موضوع باعث میشود که کلاس کاربر تنها یک تابع دیگر به اسم نمایش درباره ما داشته باشد که این اتفاق باعث کم شدن Cohesion این کلاس میشود پس این راه حل نمی تواند باعث افزایش Cohesion این کلاس شود حتی اگر آن سه تابع هم به کلاس شخص منتقل بکنیم بازم همین اتفاق خواهد افتاد و علاوه بر این کلاس پشتیبان تلفنی هم الان دارد از این سه تابع ارث بری می کند توی کلاس شخص که این موضوع درست نمی باشد پس ناچاریم همین سه تابع را درون این کلاس نگه داریم.

۳. فروشنده:

این کلاس هم Cohesion متوسطی دارد بخاطر همون علتی که برای کلاس کاربر گفتیم.

۴. پشتیبان تلفنی:

این کلاس از Cohesion کمی برخوردار است چون تنها یک عملیات انجام میدهد ولی راه حلی برای این موضوع وجود ندارد بخاطر اینکه اگر این کلاس را بخوایم با کلاس کاربر یکی بکنیم و از آنجایی که در کلاس کاربر تابع هایی است که کلاس پشتیبان نیاز ندارد پس ناچاریم این کلاس را با Cohesion کم داشته باشیم.

۵. غذا:

این کلاس جنبه های مختلف مدیریت مواد غذایی، موجودی، و اقدامات روی اقلام غذایی را پوشش می دهد پس از Cohesion بالایی برخوردار است.

۶. سفارشات:

این کلاس کارهای مربوط به سفارشات مانند کاهش یا افزایش تعداد غذا، وضعیت سفارش، پرداخت و مواردی از این دسته را انجام میدهد پس از Cohesion بالایی برخوردار است.

۷. نظرات و امتیازات:

Cohesion بالایی دارد چون بازخورد، رتبه بندی و نظرات مربوط به کاربران و جواب های فروشندگان را مدیریت می کند.

Coupling:

کلاس های غذا، سفارشات، نظرات و امتیازات Coupling کمی دارند و کلاس های شخص، کاربر، فروشنده و پشتیبان تلفنی هم Coupling زیادی دارند.

۱. شخص - کاربر - فروشنده - پشتیبان تلفنی:

این سه کلاس از کلاس شخص، ارث بری می‌کنند پس به کلاس شخص وابسته اند و هر تغییری در کلاس شخص ممکن است تاثیری بر این سه کلاس داشته باشد پس ارتباط بین این ۴ کلاس بخاطر ارث بری زیاد است در نتیجه Coupling بین این کلاس ها زیاد است برای اینکه بتوانیم این Coupling را کاهش دهیم می‌توانیم در هنگام کد زدن از اصول SOLID استفاده کنیم و همینطور می‌توانیم داخل کلاس ها هم Encapsulate بکنیم یعنی مواردی که ممکن است مستقیما از یک کلاس به کلاس دیگر دسترسی پیدا کنند را در داخل کلاس ها Encapsulate میکنیم مثلا در داخل کلاس فروشنده اگر اطلاعات فروشنده نیاز به تغییر ندارند، می‌توانیم آنها را private کنیم.

۲. غذا:

این کلاس Coupling کمی دارد چون فقط به کلاس های فروشنده و سفارشات وابسته است و از کلاس فروشنده فقط به تابع ایجاد کد برای فروشنده نیاز دارد و از کلاس سفارشات هم به تابع های کاهش یا افزایش تعداد غذا و حذف غذا وابسته است و تغییر در هر کدام از این توابع به تغییرات زیادی در کلاس سفارشات منجر نمیشود.

۳. سفارشات:

این کلاس Coupling کمی دارد چون فقط به کلاس های فروشنده و غذا وابسته است و از کلاس فروشنده فقط به تابع ایجاد کد برای فروشنده نیاز دارد و از کلاس غذا هم به تابع ایجاد کد غذا و چندین تابع دیگر نیاز دارد ولی اگر هر کدام از این تابع ها تغییر بکنند نیاز به تغییرات زیادی در کلاس سفارشات نیست.

۴. نظرات و امتیازات:

این کلاس هم Coupling کمی دارد چون به کلاس های فروشنده و سفارشات وابسته است و از کلاس فروشنده فقط به تابع ایجاد کد برای فروشنده نیاز دارد و از کلاس سفارشات به تابع ایجاد کد سفارش نیاز دارد و تغییر در هر کدام از این توابع به تغییرات زیادی در کلاس نظرات و امتیازات منجر نمیشود.

۶. به نظر شما از چه نوع معماری بایستی برای طراحی سیستم خود استفاده کنید. دلیل خود را بیان و تحلیل کنید. (برای دلایل خود نیازمندی های Nonfunctional را هم در نظر بگیرید.)

معماری Three-Tiered Client–Server برای طراحی سیستم PlatePicks مناسب میباشد، با توجه به Nonfunctional و دلایل ذیل:

- کارایی و اثربخشی:

معماری سه لایه امکان پردازش توزیع شده در چندین لایه را فراهم می کند. تقسیم مسئولیت‌ها، کارایی و اثربخشی سیستم برای پردازش همزمان درخواست‌های چند کاربر را افزایش می‌دهد و تداخل و تاخیر را به حداقل می‌رساند. این معماری همچنین از مقیاس‌پذیری پشتیبانی می‌کند و سیستم را قادر می‌سازد تا با افزایش تقاضای کاربر رسیدگی کند و سفارش‌ها را به طور موثر پردازش کند.

- امنیت:

معماری سه لایه امکان اجرای اقدامات امنیتی را در هر لایه فراهم می کند. لایه presentation می تواند احراز هویت کاربر و مدیریت session را مدیریت کند و دسترسی ایمن به سیستم را تضمین کند. لایه application می تواند قوانین authorization را اعمال کند و business logic را برای محافظت از اطلاعات حساس کاربر و جزئیات پرداخت پیاده سازی کند. لایه data می تواند encryption، secure database access و مکانیسم های backup را برای محافظت از داده های کاربر پیاده سازی کند. با گنجاندن اقدامات امنیتی در هر لایه، معماری به محافظت در برابر حملات امنیتی مانند حملات DDoS و سرقت داده ها کمک می کند.

- قابلیت استفاده:

لایه presentation معماری سه لایه بر طراحی رابط کاربری و تجربه کاربر تمرکز دارد. این اجازه می دهد تا یک رابط کاربری ساده، منطقی و جذاب ایجاد کرد که تعامل آسان با سیستم را تسهیل می کند.

- قابل حمل بودن:

معماری سه لایه از قابلیت حمل و نقل از طریق استفاده از فناوری های استاندارد توسعه وب (HTML, CSS, JavaScript) در لایه presentation پشتیبانی می کند. با رعایت این استانداردها، سیستم با مرورگرها و پلتفرم های مختلف سازگار می شود. علاوه بر این، پیکربندی وب سرور، که در درجه اول در لایه میانی (لایه application) قرار دارد، امکان راه اندازی و انطباق صحیح سیستم را با محیط های مختلف فراهم می کند و اجرای روان در پلتفرم های مختلف را تضمین می کند.

- مقیاس پذیری:

معماری سه لایه ذاتا گزینه های مقیاس پذیری را فراهم می کند. هر لایه را می توان به طور مستقل، بسته به نیاز سیستم، مقیاس بندی کرد. به عنوان مثال، لایه presentation را می توان با افزودن سرورهای وب بیشتر برای مدیریت افزایش ترافیک کاربر، به صورت افقی تغییر داد. لایه application را می توان با معرفی متعادل کننده های بار و تکنیک های خوشه بندی برای توزیع بار پردازش، مقیاس بندی کرد. لایه data را می توان با استفاده از مکانیسم های همانندسازی پایگاه داده، به اشتراک گذاری یا ذخیره سازی، مقیاس بندی کرد. این انعطاف‌پذیری معماری به سیستم اجازه می‌دهد تا تعداد بیشتری از کاربران، سفارش‌ها، پرداخت‌ها و اطلاعات کاربر را به طور موثر مدیریت کند.

- دسترسی:

معماری سه لایه، هنگامی که با شیوه های توسعه مناسب ترکیب شود، می تواند در دسترس بودن سیستم را تسهیل کرده و اختلالات را به حداقل برساند. با توزیع اجزای سیستم در چندین لایه و اجرای مکانیسم‌های افزونگی و تحمل خطا، معماری پایداری سیستم را افزایش می‌دهد. علاوه بر این، ابزارهای نظارت و مکانیسم‌های رسیدگی به خطا را می‌توان در هر لایه برای شناسایی و رفع سریع اختلالات، کاهش تأثیر منفی بر تجربه کاربر و اطمینان از دسترسی مداوم پیاده‌سازی کرد.

معماری Server-Based، کل پردازش در سمت سرور انجام می شود و سمت سرویس گیرنده به نمایش نتایج محدود می شود. این معماری ممکن است برای این سیستم ایده آل نباشد زیرا فاقد تعامل و به روز رسانی در زمان واقعی است. این سیستم به تعاملات کاربر مانند افزودن اقلام به سبد خرید و ... نیاز دارد که اجرای کارآمد آن با معماری Server-Based چالش‌برانگیزتر است.

معماری Client-Based، تمامی پردازش ها در سمت کلاینت انجام می شود و سرور فقط وظیفه ذخیره و بازیابی داده ها را بر عهده دارد. این معماری ممکن است به دلایل مختلفی برای این سیستم مناسب نباشد. اولاً، به قدرت محاسباتی قابل توجهی در سمت مشتری نیاز دارد، که ممکن است برای همه دستگاه‌های کاربران، به ویژه برای دستگاه‌های تلفن همراه با منابع محدود، امکان‌پذیر نباشد. ثانیا، مدیریت امنیت و یکپارچگی داده ها چالش برانگیزتر می شود زیرا سمت مشتری کنترل بیشتری بر سیستم دارد. در نهایت، اطمینان از سازگاری و همگام سازی بین چندین مشتری می تواند دشوار باشد، به خصوص زمانی که با به روز رسانی های بلادرنگ و پردازش سفارش سروکار داریم.