تمرین سوم درس مهندسی نرم‌افزار

گروه پنجم

اعضای گروه:

امیرحسین فراهانی ۹۷۱۰۶۱۵۴

سارا آذرنوش ۹۸۱۷۰۶۶۸

پیمان حاجی محمد ۹۸۱۷۰۷۷۶

محمدعلی حسین نژاد ۹۸۱۷۰۷۸۷

رویا قوامی ۹۸۱۷۱۰۳۱

۱. الف) به صورت کلی وقتی درمورد معماری حرف میزنیم ساختار فیزیکی اجسام در ذهن میایند. ولی درمورد معماری نرم افزار تعریف زیر را میتوان ارائه داد:

معماری نرم‌افزار، ساختار یا ساختارهای سیستم هست که شامل اجزای مختلف نرم‌افزار، ویژگی‌های قابل مشاهده خارجی آن اجزا و روابط بین آن‌ها میشود.

ب) در مرحله design نرم افزار، بعد از طراحی Data/Class نوبت به مرحله طراحی معماری نرم‌افزار میرسد. و بعد از طراحی معماری سیستم، نوبت به مرحله طراحی interface ها میرسد.

پ) داریم:

- معماری نرم‌افزار نمایشی از سیستم‌نهایی را ارائه میدهد که ارتباط بین ذی‌نفعان را راحت تر میکند.
- معماری، تصمیمات اولیه طراحی را برجسته میکند که تاثیر زیادی روی تمام کارها خواهند داشت.
- درواقع معماری مدلی نسبتا کوچک و قابل درک از ساختار سیستم و نحوه کار اجزای آن ارائه میدهد.
- برای بررسی effectiveness طراحی‌های نرم افزار میتواند استفاده شود. ۵) ریسک ساخت نرم‌افزار را کم‌تر میکند.

ت) این تصویر سطوح مختلف abstraction بدن انسان را نمایش میدهد. چپ ترین تصویر صرفا طراحی اسکلت بدن را نمایش میدهد. راست‌ترین تصویر ارتباط بین بخش های مختلف را نشان میدهد. درواقع

این دو تصویر مشابه Structural properties در معماری نرم‌افزار میباشند. تصویر دوم از راست به نوعی کامپونت‌های مختلف سیستم را نشان میدهد. و تصویر دوم از راست درواقع هم ارز Extra Functional Properties در معماری میباشد.

۲. الف) طبق منبع درس تعریف pattern: یک الگو یک راه حل اثبات شده برای یک مشکل تکرار شونده را در یک زمینه خاص میباشد. درواقع یک الگوی‌طراحی، یک ساختار طراحی هست که یک مشکل طراحی خاص را حل میکند. همینطور هر الگو از سه جزء تشکیل شده: problem, solution, context.

ب) هرکدام از این موارد را دو به دو باهم مقایسه میکنیم.

- مقایسه Pattern و Principle: هردو از این نظر که قابل استفاده در انواع مسائل هستند و صرفا مخصوص زبان‌خاصی نیستند شباهت دارند. ولی Patterها یک راه‌حل برای یک مشکل خاص هستند، ولی Principleها باید در همه راه‌حل‌ها و پیاده‌سازی ها فارغ از نوع مسئله پیاده‌سازی شوند.
- مقایسه Pattern و Idiom: درواقع هردوی این موارد برای مشکل مشخصی راه‌حل بهینه‌ای ارائه میدهند با این تفاوت که Patternها فارغ از زبان‌های برنامه نویسی و برای مسائل مختلف میباشند و ربطی به زبان‌برنامه نویسی ندارند. ولی از طرفی Idiomهای زبان برنامه نویسی، مخصوص یک زبان خاص هستند که توسعه‌دهنده های آن زبان از آن استفاده میکنند. در نتیجه قابل استفاده در همه زبان‌های برنامه نویسی نیستند.
- مقایسه Idiom و Principle: طبق دو مقایسه قبلی میتوان نتیجه گیری کرد تقریبا شباهت این دو خیلی کم هست. چرا که Principle سطح انتزاع بالاتری نسبت به Idiomدارد. و از طرفی Idiomبرای مشکل خاصی هست ولی Principle برای همه پیاده‌سازی ها باید استفاده شود.

۳.

الف)

در الگوهای معماری ما به بررسی اسکلت بندی و زیرساخت کلی یک نرم افزار می پردازیم، اما الگوهای طراحی نرم افزار به جزییات بیشتری توجه می کنیم و وارد سطح کد میشویم در طراحی به بررسی وظیفه هر ماژول می پردازیم، اسکوپ کلاس ها را بررسی می کنیم، فانکشنالیتی ها را تعیین می کنیم و مشخص میکنیم ارتباط کلی اجزای نرم افزار با یکدیگر چگونه باشد.

مواردی همچون عملکرد کلی سیستم(performance)، تحمل پذیری در برابر خطاها(fault tolerance)، مقیاس پذیری (scalability)و ضریب اطمینان (reliability) جزو موارد اصلی هستند که باید در معماری به آنها توجه شود. اما در طراحی به مواردی همچون نحوه ارتباط میان ماژولی می پردازیم

از دیگر تفاوت های این دو این است که در الگوی معماری در سطح بالای سیستم محدوده را مشخص می کنیم مانند ساختار اجزای سیستم و اما در الگو های طراحی ما محدوده سطح پایین سیستم را تعریف می کنیم مانند اینکه یک جزء سیستم چگونه پیاده سازی می گردد و توجه داشته باشید ما در الگوی معماری نحوه پیاده سازی را مشخص نمی نماییم.

ب)

معماری میکروسرویس:

این معماری همانطور که از نامش مشخص است کلیت سیستم را به چند سرویس مستقل از هم می شکند که می توانند با هم در ارتباط باشند اما به طور کلی می توانند مستقلا به فعالیت خود ادامه دهند. این معماری این را به ما می دهد که بتوانیم تیم های کاملا مستقل از همی داشته باشیم که روی توسعه نرم افزار کار کنند و در واقع سرعت توسعه و مدیریت آن پیشرفت قابل ملاحظه ای خواهد داشت. همچنین این معماری امکان تشخیص منبع خطا و در ادامه رفع آن را بسیار آسان می کند. این معماری متضمن قابلیت نگهداری هر سیستم است. البته مانند هر معماری دیگری ممکن است آفاتی هم به بار آورد مثل تقسیم بیش از حد سرویس ها. همچنین برای مواردی همچون یک mvp کوچک احتمالا مناسب نیست چون ارتباط میان سرویسی هم هزینه پردازشی بیشتری خواهد داشت هم هزینه توسعه بالاتر (این خود می تواند به زمان بیشتر نیز منجر شود که با روح mvp در تناقض است.)

معماری چند لایه (می تواند هر تعداد لایه داشته باشد اما نوع رایج آن سه لایه است که به توضیح آن می پردازم):

در این معماری سیستم به چند لایه تقسیم می شود که در نوع رایج ان سیستم سه سطح نمایش یا رابط کاربری، اپلیکیشن یا کاربرد و سطح ذخیره سازی را شامل می شود.

لایه نمایش شامل هر چیزی است که کاربر نشان داده می شود در دنیای امروز front-end و همچنین کلاینت های اندرویدی و ... در این لایه قرار می گیرند

لایه کاربرد لایه ای است که تمام پردازش ها و منطق های سیستم در آن قرار دارد

در لایه ذخیره سازی ما به ذخیره و بازیابی اطلاعات (معمولا در دیتابیس) می پردازیم.

این معماری این امکان را به ما می دهد که توسعه و نگهداری هر لایه مجزا از لایه های دیگر باشد و چند تیم توسعه مجزا می توانند روی یک پروژه کار کنند. در نتیجه ما می توانیم تغییراتی در هر لایه به وجود آوریم بدون اینکه عملکرد لایه های دیگر خراب شود

برای مدت ها این معماری معماری اصلی نرم افزارها بود اما امروز جای خود را به معماری های جدیدتری مثل میکرو سرویس داده است. (البته همچنان ممکن برای برخی از کاربردها بهترین حالت باشد) مشکل اصلی که باعث جایگزینی آن شده است مربوط به بزرگ شدن بیش از حد هر لایه است به نحوی که بعد از مدتی عملا شما امکان اضافه کردن قابلیت های جدید و توسعه را از دست میدهید و نگهداری آن هم به علت بسیار بزرگ شدن دشوار می شود حال آن که میکروسرویس بر این مشکل فائق می آید.

همچنین باید گفت در این معماری می توان لایه های دیگری را بسته به کاربرد اضافه کرد مانند لایه امنیت و ...

۴.

الف)

رابط کاربری (UI) به معنای طراحی و ارائه اجزای گرافیکی و بصری مانند دکمه‌ها، منوها و صفحات وب است که کاربر با آن‌ها در ارتباط است.

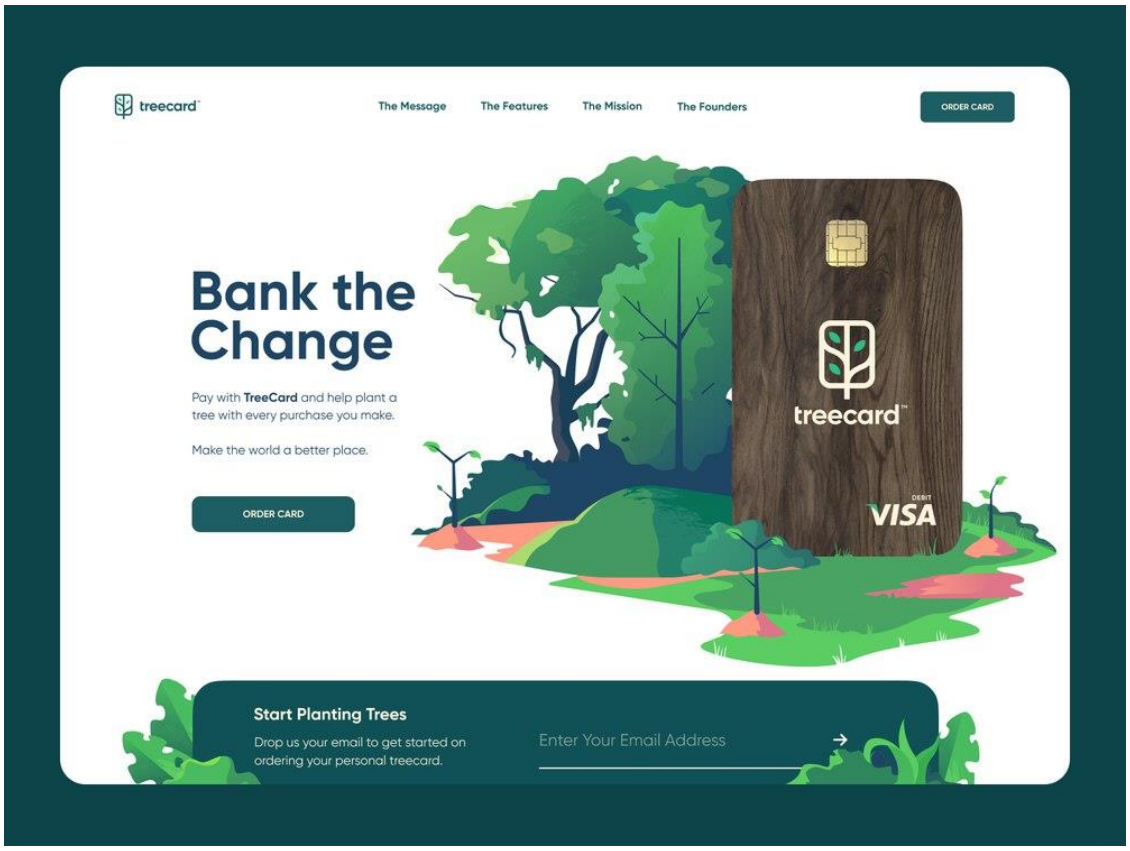
تجربه کاربری (UX) شامل تمام تعاملات کاربر با محصول، از جمله طراحی، فرآیند خرید، نحوه استفاده و رفتار کاربر در محصول است.

UI بخشی از UX است و هر دو برای ایجاد تجربه کاربری بهتر و موثرتر باید با همکاری و هماهنگی صورت گیرد.

ب)

- Minimalism:

- مشکل در پی حل: برخی کاربران از طراحی‌های پر هیجان و پر رنگ خسته شده‌اند و به دنبال کاهش بار اطلاعاتی در رابط کاربری و ارائه یک تجربه کاربری ساده و خلاصه هستند.
- موارد کاربرد: مناسب برای وب‌سایت‌های شخصی، وبلاگ‌ها و اپلیکیشن‌های ساده
- راه حل پیشنهادی: استفاده از المان‌های ساده و خلاصه، استفاده از فونت‌های ساده و خوانا، استفاده از رنگ‌های کم و کم‌رنگ، استفاده از آیکون‌های ساده و شفاف، حذف اجزای غیرضروری و تمرکز بر روی محتوا به عنوان راهکار اصلی این الگو مطرح می‌شود.



- Skeuomorphism:

- مشکل در پی حل: ارائه یک تجربه کاربری واقع‌گرایانه و شبیه به دنیای واقعی

- موارد کاربرد: مناسب برای اپلیکیشن‌های بازی، محصولات دیجیتالی مثل کتابخانه‌های الکترونیکی و نرم‌افزارهای گرافیکی

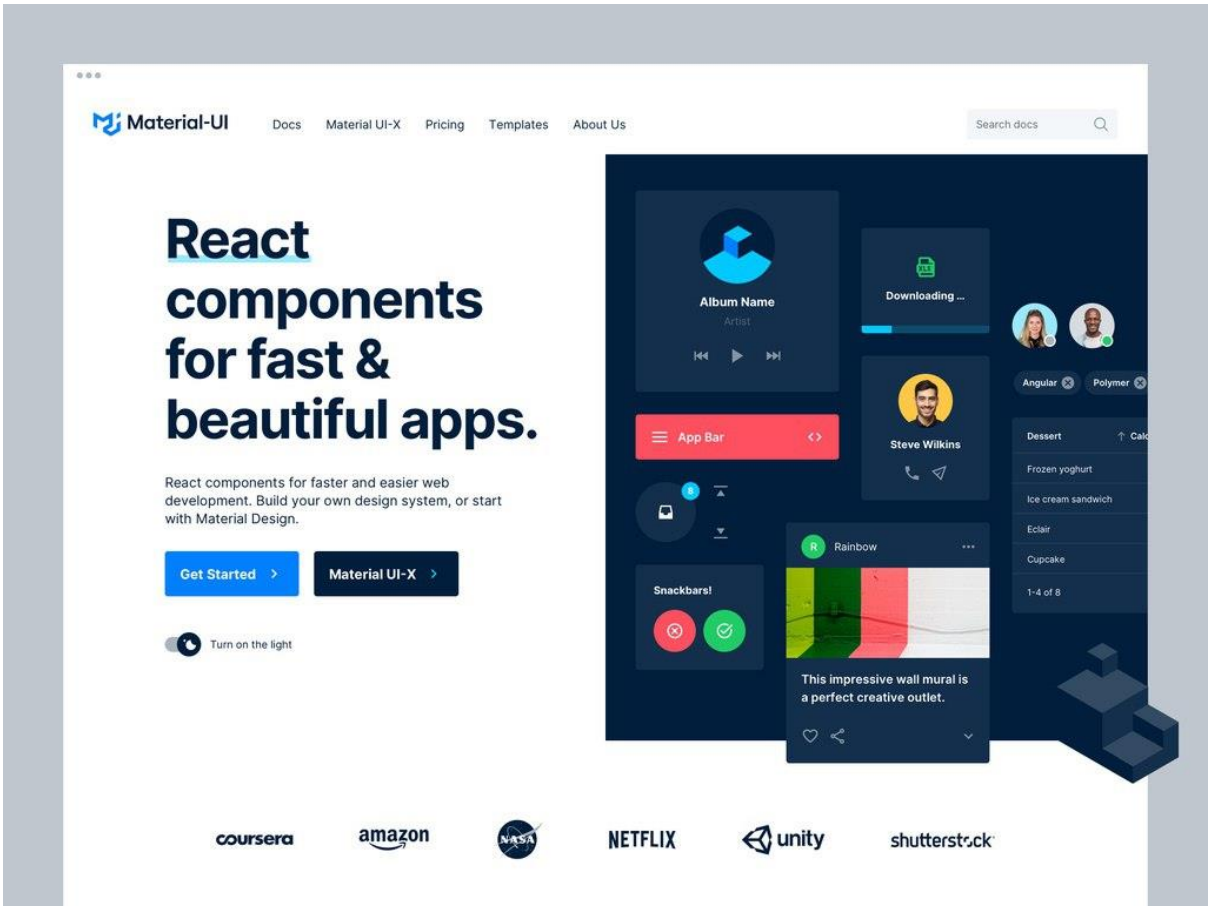
- راه حل پیشنهادی: استفاده از المان‌های گرافیکی با تراکم بالا، استفاده از رنگ‌های طبیعی، استفاده از حرکات واقع‌گرایانه، استفاده از سایه‌ها و نورپردازی

- Material Design:

- مشکل در پی حل: تنوع دستگاه‌ها و سیستم‌عامل‌های مختلف باعث شده است که طراحان نیاز به ارائه یک تجربه کاربری یکپارچه و هماهنگ برای کاربران در تمام پلتفرم‌ها داشته باشند.

- موارد کاربرد: مناسب برای اپلیکیشن‌های تلفن همراه، تبلت و وب

- راه حل پیشنهادی: استفاده از اجزای گرافیکی سه بعدی، استفاده از رنگ‌های شاد و زنده، استفاده از آیکون‌های ساده و خلاصه، استفاده از حرکات نرم و صاف. تمرکز بر روی ارائه اطلاعات به صورت سلسله مراتبی و مطابق با هر پلتفرم به عنوان راهکار اصلی این الگو مطرح می‌شود.

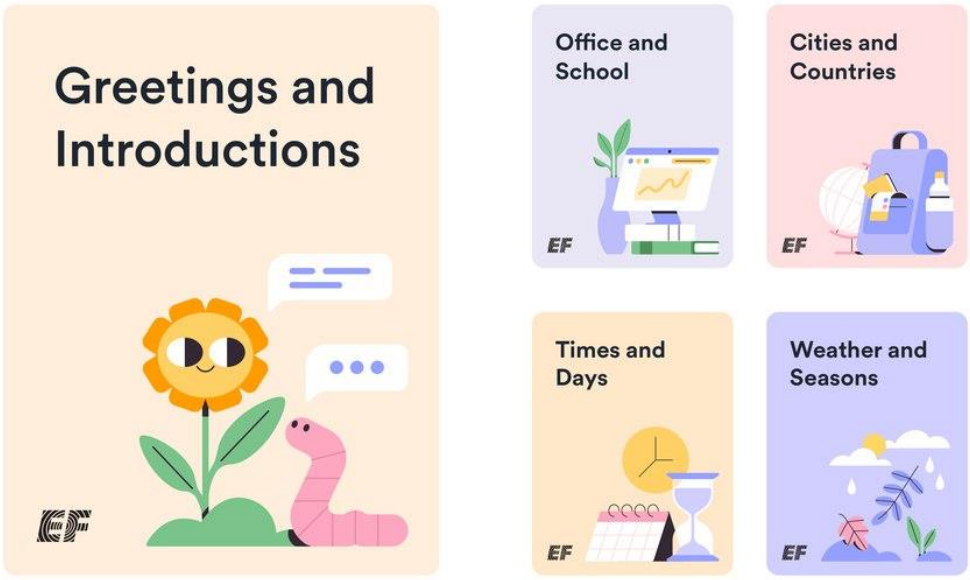


- Flat Design:

- مشکل در پی حل: افزایش تعداد دستگاه‌های مختلف و اندازه صفحه نمایش آن‌ها باعث شده است که طراحان برای ارائه رابط کاربری بهینه و جذاب، نیاز به یک الگوی ساده و کاربرپسند و کاهش تعداد المان‌های غیرضروری در رابط کاربری داشته باشند.

- موارد کاربرد: مناسب برای وب‌سایت‌ها و اپلیکیشن‌هایی که نیاز به طراحی ساده، خلاصه و مینیمال دارند.

- راه حل پیشنهادی: استفاده از رنگ‌های شاد و زنده، استفاده از فونت‌های ساده و خوانا، حذف المان‌های غیرضروری و استفاده از آیکون‌های ساده و تمرکز بر روی محتوا به عنوان راهکار اصلی این الگو مطرح می‌شود.



تفاوت در سه الگوی آخر:

۵.

- Singleton pattern: در این الگو تنها یک نمونه از یک کلاس مشخص وجود دارد و امکان ایجاد نمونه جدیدی از آن کلاس وجود ندارد. یک مثال از این الگو در واقعیت می تواند دولت ها باشند که تنها یک نمونه رسمی از آنها در هر کشور وجود دارد.
- Adapter pattern : این الگو زمانی مورد استفاده قرار می گیرد که قصد داریم دو شی که رابط های متفاوتی دارند را بدون تغییر در ساختار کلاس هایشان با یکدیگر در ارتباط

قرار دهیم. مثال در دنیای واقعی استفاده از آداپتور های تبدیل برق ۱۱۰ ولت به ۲۲۰ در کشور های مختلف برای برق رسانی به وسایل برقی می باشد

- Decorator pattern: به کمک این الگو میتوانیم ویژگی های جدیدی به یک کلاس اضافه کنیم بدون آنکه در ساختار آن کلاس تغییری ایجاد کنیم. مثال در دنیای واقعی، هنگامی که در محیط سردی قرار دارید می توانید یک تی شرت گرم بپوشید، سپس اگر بخواهید در برابر سرمای بیشتر نیز دوام بیاورید می توانید یک کاپشن دیگر نیز روی تی شرت خود بپوشید بدین طریق بدون تغییر در تی شرت خود و تنها با wrap کردن توانستید ویژگی جدید گرم تر بودن را به خود اضافه کنید و همچنین هر زمانی که اراده کنید می تواند کاپشن را در بیاورید.
- Observer pattern: در این الگو یک آبجکت publisher وجود دارد که با به وجود آمدن تغییر در وضعیتش، آبجکت های subscriber را از این تغییر با خبر می کند و در واقع subscriber ها در حال observe کردن آبجکت مذکور می باشند. مثال در دنیای واقعی، سابسکرایب کردن در یک کانال یوتیوب می باشد. هنگامی که یک کانال را سابسکرایب می کنید دیگر نیازی به اینکه هر بار برای مطلع شدن از آپلود شدن ویدیو جدید در کانال مد نظر سایت یوتیوب را چک کنید، به صورت خودکار بعد از آپلود شدن ویدیو، پابلیشر به تمامی سابسکرایبر هایش خبر (نوتیفیکیشن) آن را ارسال می کند.
- Strategy pattern: این الگو امکان ایجاد و در کنار هم قرار دادن چندین الگوریتم و استراتژی برای انجام یک تسک را به ما می دهد. بعنوان مثال در واقعیت میتوان به وجود روش های گوناگون برای سفر به یک شهر دیگر اشاره کرد (خودرو شخصی، اتوبوس، قطار و...)
- Proxy pattern: این الگو جهت ایجاد امکان دسترسی به یک شی از طریق شی دیگر استفاده می شود. بعنوان مثال، استفاده از کارت بانکی به جای پول نقد در خرید از فروشگاه
- Command pattern: در این الگو ما می توانیم درخواست ها را به صورت یک آبجکت در آورده و آنها را در صف های انتظار قرار دهیم تا نوبت به سرویس دهی آنها برسد، بعنوان مثال ثبت سفارش مشتری ها در رستوان توسط گارسون و انتقال سفارش به صف انتظار برای پخت و سرو غذا
- Mediator pattern: در این الگو آبجکت ها دو به دو با یکدیگر ارتباط برقرار نمی کنند بلکه یک آبجکت واسط وجود دارد که در نقش واسط بین دیگر آبجکت ها ارتباطات لازم را برقرار می کند. بعنوان مثال نقش برج مراقبت بعنوان واسط برای دادن اطلاعات به هواپیما

ها، هواپیما ها دو به دو با یکدیگر تماس برقرار نمی کنند بلکه با برج مراقبت هماهنگ کرده و برای بلند شدن و نشستن به نوبت بر روی باند فرودگاه هماهنگی های لازم را انجام میدهند

- Composite pattern: این الگو به ما کمک می کند تا آبجکت ها را ساختاری درخت شکل و سلسله مراتبی شکل دهیم. بعنوان مثال، در سیستم نظامی، درجه بندی ها و گروه بندی ها به صورت سلسله مراتبی هستند و دستورات لازم از ریشه به سمت برگ ها منتقل میشوند و سربازان درجه پایین تر از وظایف و تسک های خود مطلع می شوند
- Visitor pattern: این الگوریتم این امکان را به ما میدهد تا الگوریتم ها را از آبجکت هایی که روی آنها کار می کنند تفکیک کنیم. بعنوان مثال یک کارمند بیمه را تصور کنید که میخواهد مشتری های بیشتر جذب بیمه کند. وی بدین صورت عمل می کند که نسبت به اینکه مشتری چه فردی (آبجکت) است پیشنهادات مختلفی می دهد، بعنوان مثال به یک کارمند پیشنهاد بیمه بازنشستگی میدهد و به یک رئیس بانک بیمه سرقت را پیشنهاد می کند.

۶.

الف)

Seamlessness به معنای ایجاد یک تجربه کاربری بدون درز است. تمام المان‌های رابط کاربری به گونه‌ای طراحی شده‌اند که هماهنگی و یکپارچگی بین آن‌ها وجود دارد و کاربران می‌توانند به راحتی از یک المان به دیگری منتقل شوند بدون اینکه احساس درز بین المان‌ها داشته باشند و به صورت فکر شده و کامل است.

ب)

متدولوژی Seamlessness به مجموعه روش‌ها و فرایندهای طراحی اشاره دارد که برای ایجاد یک تجربه کاربری بدون درز در رابط کاربری استفاده می‌شود.

شامل مراحل طراحی، تحلیل، توسعه و ارزیابی است که هدف آن ایجاد یک تجربه کاربری ساده، قابل فهم و هماهنگ است.

مراحل دقیقتر به صورت زیر است:

۱. محصول رویایی خود را کاوش کنید:

اولین قدمی که باید قبل از شروع فرآیند طراحی بردارید، ایجاد یک طرح کلی از آنچه می خواهید نرم افزار انجام دهد است.

۲. تیم خود را درگیر کنید:

با مدیران و رهبران بخش صحبت کنید. با کاربران صحبت کنید ناامیدی آنها را پیدا کنید و بینش آنها را در مورد این پروژه دریافت کنید. برای اینکه پروژه موفق باشد، به حمایت همه افراد درگیر نیاز دارید.

۳. توسعه دهنده مناسب را پیدا کنید:

این مکان مناسبی برای پس انداز پول نیست. سیستم شما یک سرمایه گذاری برای آینده شرکت شما است. شما می‌خواهید یک توسعه‌دهنده چه خارجی و چه در کارکنان دارای تجربه و ابزار مناسب برای انجام کار حرفه‌ای و به موقع باشد.

۴. اهداف خود را به اشتراک بگذارید:

اطمینان حاصل کنید که کل تیمی که روی پروژه کار می‌کنند (و به‌ویژه توسعه‌دهندگان!) درک متقابلی از اهداف سیستم و چگونگی سود آن برای شرکت دارند.

۵. تست:

در حالت ایده‌آل، توسعه‌دهنده تضمین کیفیت کامل را انجام می‌دهد (اگر این کار را نکردند، آنها را استخدام نکنید - اگر مرحله آزمایشی را در نظر نگرفته باشند، زمان توسعه دو برابر آن چیزی است که آنها نقل می‌کنند).

۷.

- نرم افزار ماژولار شامل بخش های مجزا و مستقل است که به صورت ماژول های جداگانه طراحی شده اند. هر ماژول مسئولیت خاص خود را دارد و می تواند به صورت جداگانه توسعه و به سیستم اضافه شود. در این نوع نرم افزار، تغییر یک ماژول، تأثیری بر سایر بخش های سیستم ندارد و قابلیت توسعه و ارتقاء آن بسیار بالاست.

  اما نرم افزار یک پارچه شامل بخش های مختلف است که به صورت یکپارچه با هم کار می کنند و تغییر در یک بخش، تأثیری بر سایر بخش های سیستم خواهد داشت. این نوع نرم افزار برای پروژه های کوچک و ساده مناسب است و قابلیت توسعه و ارتقاء آن محدود است.
- در زمانی که نیازمند پیاده سازی سریع پروژه هستیم و یا نرم افزار نیازمند توسعه و ارتقای زیادی نمی باشد، طراحی یکپارچه به دلیل سرعت بالا و پیچیدگی کمتر در پیاده سازی مناسب تر از ماژولار است.