1. تفاوت معماری Clean با Onion؟
   1. در clean، فاصله و وابستگی بین use case ها با domain مون به نسبت onion کمتره. در واقع در clean میخواد که تا حد امکان use case ها مستقل از domain باشن اما توی onion برعکسه و میخواد که use case ها رو تا حد امکان با domain یکپارچه کنه.
   2. ساختار onion به نسبت clean ساده تره اما clean انعطاف پذیری بیشتری داره چون دنبال جداسازی لایه ها و مستقل کردنشون هست.
   3. در onion، برای جداسازی technical concern ها از business logic ها صرفا اونا رو به لایه های بالاتر منتقل می کنن اما اینکار در clean با کمک dependency inversion انجام میشه.
   4. هر دو به دنبال separation of concern هستن.
   5. در هر دو جهت وابستگی از بیرون به داخله و لایه های داخلی اجازه ندارن به لایه های بیرونی وابستگی داشته باشن.
2. معماری تمیز چیست؟
   1. نوعی معماری ساخت پروژه ها است که تمرکز آن بر روی لایه بندی اجزای مختلف یک پروژه و کنترل وابستگی یا استقلال بین آن ها است. این معماری از 4 لایه تشکیل شده است که عبارتند از:
      1. Domain: این لایه شامل مدل ها و business logic های پروژه است و درونی ترین لایه می باشد به این معنی که این لایه هیچ وابستگی ای به لایه های دیگر ندارد.
      2. Application: این لایه شامل سرویس ها و business rule ها است. این لایه بعد از لایه Domain قرار گرفته و فقط به لایه Domain وابستگی دارد.
      3. Infrastructure: این لایه مسئول دسترسی به دیتابیس و external dependency ها می باشد. این لایه بعد از Application قرار گرفته و به آن وابستگی دارد.
      4. Presentation: آخرین لایه در این معماری است که وظیفه ارتباط بین پروژه و کلاینت ها را بر عهده دارد. این لایه بیرونی ترین لایه بوده و به لایه Application وابستگی دارد.
   2. دقت کنیم که وابستگی ها از لایه های بیرونی به داخلی بوده و برعکس آن ممکن نیست.
   3. لایه ای مانند Application در صورت نیاز به دیتا از دیتابیس (نیاز به لایه infrastructure که بالاتر است) از طریق dependency inversion نیاز خود را برطرف می کند. یعنی با استفاده از اینترفیس ها، به پیاده سازی های لایه infrastructure دسترسی داشته و قبلا با کمک dependency injection اینترفیس ها به پیاده سازی مربوطه وصل شده اند. در نتیجه در لایه Application می توان صرفا با استفاده از اینترفیس ها به پیاده سازی ها و درنتیجه لایه بالا نیز دسترسی داشت بدون آنکه وابستگی از Application به Infrastructure ایجاد شود.
3. معماری میکروسرویس چیست؟
   1. در این معماری اجزا و فانکشنالیتی های مختلف یک پروژه به سرویس های کوچک و مستقل شکسته می شود که هر کدام مجزا از باقی سرویس ها کار خود را انجام می دهند.
   2. هر سرویس مسئول بخش خاصی از پروژه است و قرار نیست در کار سرویس های دیگر دخالت کند. در ضمن هر سرویس می تواند به صورت جداگانه deploy شود.
   3. سرویس ها برای ارتباط با یکدیگر از روش های مختلفی می توانند استفاده کنند. روش هایی مانند Remote Procedure Call (RPC)، Event streaming و Message broker ها از این دسته هستند.
   4. یک API Gateway وجود دارد که مسئول دریافت درخواست های کلاینت و ارسال آن درخواست به سرویس مورد نظر است. همچنین جواب را از سرویس مربوطه گرفته و به کلاینت ارسال می کند.
   5. میکروسرویس ها برای پروژه های با مقیاس بالا به کار می آید و برای شروع یک پروژه یا پروژه های کوچک پیشنهاد نمی شود چرا که پیچیدگی و هزینه پیاده سازی بالایی دارد اما همین موضوع باعث می شود برای پروژه های با مقیاس بالا انتخاب مناسبی باشد.  
      پینشهاد می شود که پروژه های کوچک با معماری های ساده تری شروع کنند اما از آنجایی که ممکن است زمانی این پروژه به مقیاس بزرگتری برسد، بهتر است که سرویس ها و فانکشنالیتی ها را با اینترفیس های well-defined تعریف و پیاده سازی کنند.
4. Domain driven design (DDD) چیست؟
5. Test driven design (TDD) چیست؟