تصمیمات طراحی

سامان اسلامی نظری (810199375) - پاشا براهیمی (810199385) - میثاق محقق (810199484)

در نگاه اول کلاس‌ها و توابع متعددی به چشم می‌آیند که حاکی از انسجام بالای هر کلاس در پیاده‌سازی وظایف مختلف است. هر کلاس تنها تعداد محدودی از وظایف را پیاده‌سازی می‌کند؛ در این طراحی از کلاس‌های بسیار بزرگی که وظایف و اعمال مختلفی را انجام می‌دهند استفاده نشده است.
بنابر اصل کنترلر، در ابتدا کلاس PackageRequestHandler مورد استفاده قرار گرفته که وظیفه هماهنگی و کنترل وظایف کلاس‌هایی را دارد که این مورد کاربرد را پیاده‌سازی می‌کنند. در کنار این کلاس نیز، PackageValidator مشاهده می‌شود؛ این کلاس از اصل فاعل متخصص پیروی می‌کند. کلاس PackageValidator، دارای اطلاعات کافی از بسته مورد نظر بوده و به همین دلیل، وظیفه تایید کردن آن را نیز به همین کلاس سپردیم.
در ادامه به دو کلاس TherapeuticalPackage و Document می‌رسیم؛ این دو کلاس وظیفه کنترل یا محاسبات بخشی از سیستم را بر عهده دارند. وظیفه این دو تنها در نقش یک ساختار داده معنی می‌یابد. نیاز مبرم به این دو کلاس هنگامی احساس می‌شود که به فیلدهای داخل آنان و پیچیدگی ساختار این دو نگاهی بیندازیم. در صورتی که از داده ساختاری مشابه استفاده نمی‌کردیم، ناچارا مجبور به ذخیره این فیلدها به صورت Atomic در باقی کلاس‌ها می‌شدیم که موجب کثیفی کد و پیچیدگی آن می‌شد. واضح و مبرهن است که استفاده از این دو کلاس در نهایت به ساختار تمیزتر و زیباتر سیستم کمک شایانی می‌کند. بدون توضیح دیگری به توجیه بخش‌های باقی‌مانده می‌پردازیم.
به کلاس FillFormHandler می‌رسیم. همانطور که از اسمش پیداست، این یک کلاس کنترلر است. این کلاس وظیفه گرد هم آوردن کلاس‌های Document، Reservation، TherapeuticalPackageService و Patient را دارد. با استفاده از این کلاس، بخش مهمی از این موردکاربرد را که در بخش قراردادها نیز آورده شده، پیاده‌سازی می‌شود.
کمی آن طرف‌تر از کلاس‌های مذکور، به دو کلاس مهم دیگر می‌رسیم؛ TherapeuticalProcessManager و ProcessVerifier. هر دو این کلاس‌ها از اصل فاعل متخصص پیروی می‌کنند. کلاس اول با استفاده از دو ساختار داده TherapeuticalProcess و OrganizationUser بخشی از منطق مورد کاربرد را که اضافه کردن یک روند درمانی است، پیاده‌سازی می‌کند. کلاس دوم نیز صرفا وظیفه تایید یک روند درمانی را پیاده‌سازی می‌کند.

در ادامه به دو کلاس User و Service برمی‌خوریم. این دو کلاس الگوی بسیار مهم سردر را پیاده‌سازی کرده‌اند. از طریق این دو کلاس، کلاس‌های دیگری که از آن‌ها ارث‌بری کرده‌اند، به صورت abstract در دسترس خواهند بود. کلاس‌های OrganizationUser و Patient از کلاس User و کلاس TherapeuticalPackageService از Service ارث‌بری می‌کنند. دقت شود که در کلاس Service تنها دسترسی abstract به یک کلاس فراهم شده؛ شاید این موضوع به ذهن خطور کند که چرا همان کلاس را جایگزین کلاس پدر کنیم؟ با اینکه این کار امکان‌پذیر است (و امکان‌پذیری نشان‌دهنده رعایت اصول درست شی‌گرایی در طراحی این سیستم است)، برای رعایت گسترش‌پذیری سیستم، در صورت اضافه شدن موجودیت‌های دیگر، این الگو حفظ شده است.

در مدل دامنه، نسخه کامل‌تر کلاس‌ها را می‌توان مشاهده کرد که در آنجا انواع دیگر Service هم داریم که از آن ارث می‌برند.

در پایین نمودار شاهد کلاس‌های مخصوص Payment هستیم. PackageReservationHandler یک کلاس کنترلر است که با ایجاد هماهنگی میان کلاس‌های نشان‌داده شده در نمودار، وظیفه پرداخت پکیج را دارد. کلاس PaymentHandler نیز وظیفه ایجاد پرداخت‌ها و ذخیره آن‌ها را دارد. در نهایت با کلاس ExternalPaymentHandler مواجهیم که وظیفه آن پرداخت و دریافت تایید پرداخت از طریق درگاه‌های خارج از سیستم را دارد.

کلاس بعدی که از اصل فاعل متخصص استفاده می‌کند، کلاس UserExpensesConfirmManager می‌باشد. این کلاس با در اختیار داشتن لیستی از مدارک و هزینه‌ها، تایید مدارک توسط کاربر را پیاده‌سازی می‌کند.

دو کلاس فاعل متخصص دیگر را نیز در گوشه پایین نمودار شاهد هستیم. این دو کلاس عبارتند از UserDocumentManager و AccountingManager. وظیفه این دو کلاس از توابعی که پیاده‌سازی می‌کنند و از منابعی که در اختیار دارند پیداست (دقت شود چه در این بخش و چه در بخش‌های دیگری که صحبتی از منابع در میان است، فاعل متخصص حتما نباید آن‌ها را به صورت فیلدی در خود ذخیره کند؛ بلکه این منابع می‌توانند به صورت آرگومان تابع نیز به تابع یا کلاس مد نظر انتقال داده شوند). اولی وظیفه اضافه کردن مدارک مخارج به پوشه بیمار را دارد و دومی نیز وظیفه دریافت تایید مدارک مخارج توسط بخش حسابداری را دارد.

در بالاتر شاهد دو کلاس فاعل متخصصِ FormVerificationManager و DocumentVerificationUserNotifier هستیم. علاوه بر آن یک کلاس کنترلر ساده نیز به نام EmailServiceHandler داریم که نامش گویای وظیفه‌اش است؛ دقت شود برای اجتناب از توضیح واضحات و پرهیز از نشان دادن مواردی مانند سرویس ایمیل و اجزایش که از حوصله این نمودار خارج است، شکل وابستگی‌های کلاس EmailServiceHandler را نکشیدیم.

در ادامه به کلاس AddDoctorManager برمی‌خوریم. این کلاس وظیفه ساختن یک شی از Doctor را با استفاده از اطلاعات داده شده دارد. این کلاس از اصل سازنده پیروی می‌کند.
کمی بالاتر از کلاس قبلی، به کلاس TherapeuticalPackageManager می‌رسیم که این نیز از اصل سازنده پیروی می‌کند. وظیفه آن ایجاد یک TherapeuticalPackage جدید، با استفاده از اطلاعات ورودی است.
بالاتر از کلاس قبلی نیز به VerifyOrgAccountManager می‌رسیم که نامش گویای وظیفه‌اش است. این کلاس به اطلاعات لازم برای پیاده‌سازی این منطق دسترسی داشته و در نتیجه از اصل فاعل متخصص پیروی می‌کند. در اینجا برای تقسیم وظایف صحیح و افزایش انسجام، شاهد یک ارتباط با OrgAccountVerifier هستیم که بخشی از منطق کلاس دیگر را پیاده‌سازی می‌کند.
در انتها تعدادی کلاس کوچک دیگر مشاهده می‌شود که در انتهای نامشان نشانی از پسوندهایی مانند Manager و یا Handler مشاهده نمی‌کنیم. این کلاس‌ها تماما به عنوان ساختار داده استفاده می‌شوند.