عنوان: **معماری لایه بندی نرم افزار #1**
نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۸ ۱۸:۵
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, SoC, Separation of Concerns, ASP.Net, Domain Driven Design, DDD, SOLID Principals, C#, MVC, WPF, N-Layer Architecture

طراحی یک معماری خوب و مناسب یکی از عوامل مهم تولید یک برنامه کاربردی موفق می‌باشد. بنابراین انتخاب یک ساختار مناسب به منظور تولید برنامه کاربردی بسیار مهم و تا حدودی نیز سخت است. در اینجا یاد خواهیم گرفت که چگونه یک طراحی مناسب را انتخاب نماییم. همچنین روش‌های مختلف تولید برنامه‌های کاربردی را که مطمئنا شما هم از برخی از این روشها استفاده نمودید را بررسی می‌نماییم و مزایا و معایب آن را نیز به چالش می‌کشیم.

ضد الگو (Antipattern) – رابط کاربری هوشمند (Smart UI)

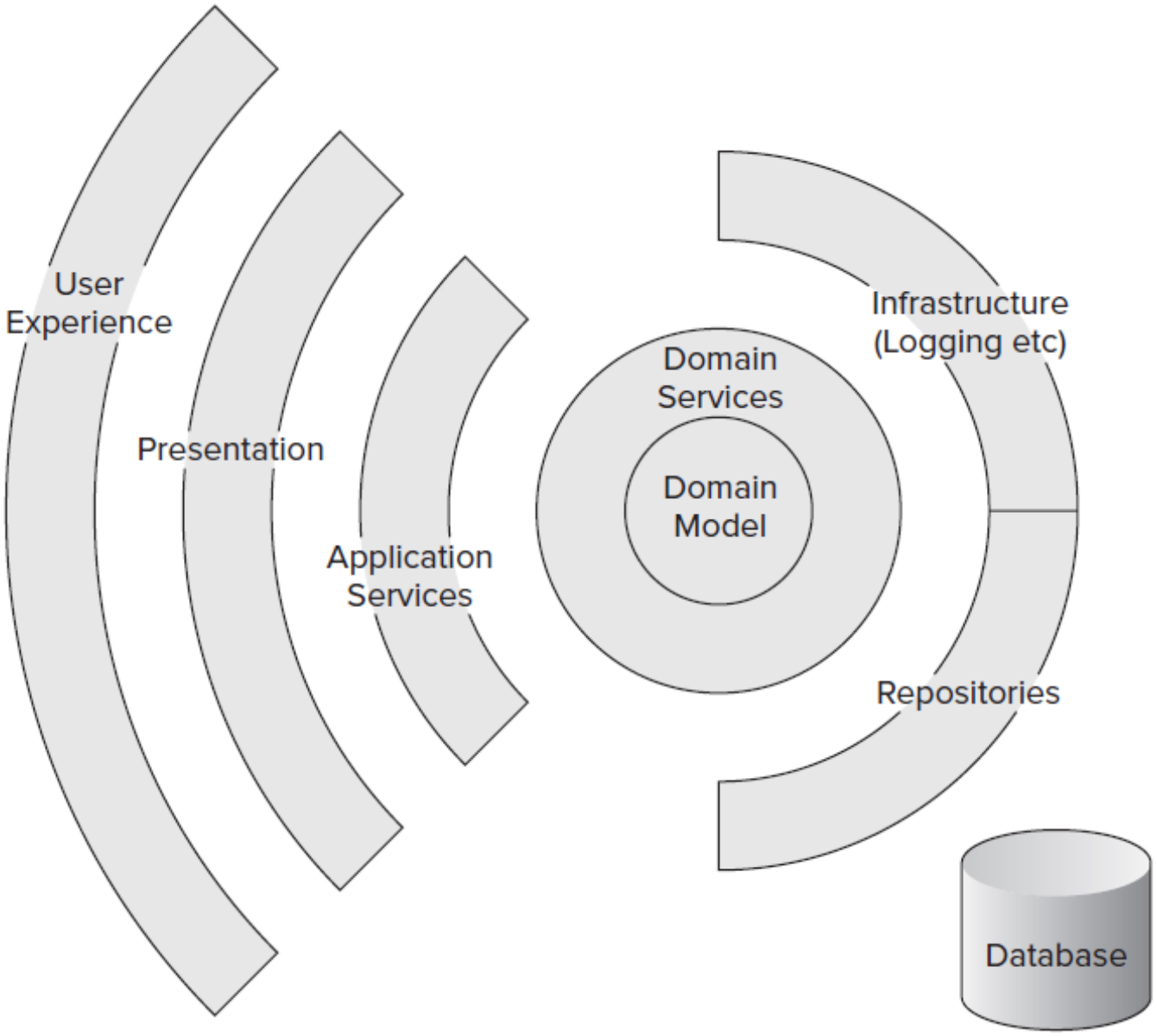
با استفاده از Visual Studio یا به اختصار VS ، می‌توانید برنامه‌های کاربردی را به راحتی تولید نمایید. طراحی رابط کاربری به آسانی عمل کشیدن و رها کردن (Drag & Drop) کنترل‌ها بر روی رابط کاربری قابل انجام است. همچنین در پشت رابط کاربری (Code Behind) تمامی عملیات مربوط به مدیریت رویدادها، دسترسی به داده ها، منطق تجاری و سایر نیازهای برنامه کاربردی، کد نویسی خواهند شد. مشکل این نوع کدنویسی بدین شرح است که تمامی نیازهای برنامه در پشت رابط کاربری قرار می‌گیرند و موجب تولید کدهای تکراری، غیر قابل تست، پیچیدگی کدنویسی و کاهش قابلیت استفاده مجدد از کد می‌گردد.

به این روش کد نویسی Smart UI می‌گویند که موجب تسهیل تولید برنامه‌های کاربردی می‌گردد. اما یکی از مشکلات عمده‌ی این روش، کاهش قابلیت نگهداری و پشتیبانی و عمر کوتاه برنامه‌های کاربردی می‌باشد که در برنامه‌های بزرگ به خوبی این مشکلات را حس خواهید کرد.

از آنجایی که تمامی برنامه نویسان مبتدی و تازه کار، از جمله من و شما در روزهای اول برنامه نویسی، به همین روش کدنویسی می‌کردیم، لزومی به ارائه مثال در رابطه با این نوع کدنویسی نمی‌بینم.

تفکیک و جدا سازی اجزای برنامه کاربردی (Separating Your Concern)

راه حل رفع مشکل Smart UI ، لایه بندی یا تفکیک اجزای برنامه از یکدیگر می‌باشد. لایه بندی برنامه می‌تواند به شکل‌های مختلفی صورت بگیرد. این کار می‌تواند توسط تفکیک کدها از طریق فضای نام (Namespace) ، پوشه بندی فایلهای حاوی کد و یا جداسازی کدها در پروژه‌های متفاوت انجام شود. در شکل زیر نمونه ای از معماری لایه بندی را برای یک برنامه کاربردی بزرگ می‌بینید.



به منظور پیاده سازی یک برنامه کاربردی لایه بندی شده و تفکیک اجزای برنامه از یکدیگر، مثالی را پیاده سازی خواهیم کرد. ممکن است در این مثال با مسائل جدید و شیوه‌های پیاده سازی جدیدی مواجه شوید که این نوع پیاده سازی برای شما قابل درک نباشد. اگر کمی صبر پیشه نمایید و این مجموعه‌ی آموزشی را پیگیری کنید، تمامی مسائل نامانوس با جزئیات بیان خواهند شد و درک آن برای شما ساده خواهد گشت. قبل از شروع این موضوع را هم به عرض برسانم که علت اصلی این نوع پیاده سازی، انعطاف پذیری بالای برنامه کاربردی، پشتیبانی و نگهداری آسان، قابلیت تست پذیری با استفاده از ابزارهای تست، پیاده سازی پروژه بصورت تیمی و تقسیم بخشهای مختلف برنامه بین اعضای تیم و سایر مزایای فوق العاده آن می‌باشد.

1- Visual Studio را باز کنید و یک Solution خالی با نام SoCPatterns.Layered ایجاد نمایید.

· جهت ایجاد Solution خالی، پس از انتخاب New Project ، از سمت چپ گزینه Other Project Types و سپس Visual Studio Solutions را انتخاب نمایید. از سمت راست گزینه Blank Solution را انتخاب کنید.

2- بر روی Solution کلیک راست نموده و از گزینه New Project > Add یک پروژه Class Library با نام SoCPatterns.Layered.Repository ایجاد کنید.

-3 با استفاده از روش فوق سه پروژه Class Library دیگر با نامهای زیر را به Solution اضافه کنید:

SoCPatterns.Layered.Model

SoCPatterns.Layered.Service

SoCPatterns.Layered.Presentation

-4 با توجه به نیاز خود یک پروژه دیگر را باید به Solution اضافه نمایید. نوع و نام پروژه در زیر لیست شده است که شما باید با توجه به نیاز خود یکی از پروژه‌های موجود در لیست را به Solution اضافه کنید.

(Windows Forms Application (SoCPatterns.Layered.WinUI

(WPF Application (SoCPatterns.Layered.WpfUI

(ASP.NET Empty Web Application (SoCPatterns.Layered.WebUI

(ASP.NET MVC 4 Web Application (SoCPatterns.Layered.MvcUI

-5 بر روی پروژه SoCPatterns.Layered.Repository کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌ی SoCPatterns.Layered.Model ارجاع دهید.

-6 بر روی پروژه SoCPatterns.Layered.Service کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌های SoCPatterns.Layered.Model و SoCPatterns.Layered.Repository ارجاع دهید.

-7 بر روی پروژه SoCPatterns.Layered.Presentation کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌های SoCPatterns.Layered.Model و SoCPatterns.Layered.Service ارجاع دهید.

-8 بر روی پروژه‌ی UI خود به عنوان مثال SoCPatterns.Layered.WebUI کلیک راست نمایید و با انتخاب گزینه Add Reference به پروژه‌های SoCPatterns.Layered.Service ، SoCPatterns.Layered.Repository ، SoCPatterns.Layered.Model و SoCPatterns.Layered.Presentation ارجاع دهید.

-9 بر روی پروژه‌ی UI خود به عنوان مثال SoCPatterns.Layered.WebUI کلیک راست نمایید و با انتخاب گزینه Set as StartUp Project پروژه‌ی اجرایی را مشخص کنید.

-10 بر روی Solution کلیک راست نمایید و با انتخاب گزینه Add > New Solution Folder پوشه‌های زیر را اضافه نموده و پروژه‌های مرتبط را با عمل Drag & Drop در داخل پوشه‌ی مورد نظر قرار دهید.

1. UI
§ SoCPatterns.Layered.WebUI

2. Presentation Layer
§ SoCPatterns.Layered.Presentation

3. Service Layer
§ SoCPatterns.Layered.Service

4. Domain Layer

§ SoCPatterns.Layered.Model

5. Data Layer

§ SoCPatterns.Layered.Repository

توجه داشته باشید که پوشه بندی برای مرتب سازی لایه‌ها و دسترسی راحت‌تر به آنها می‌باشد.

پیاده سازی ساختار لایه بندی برنامه به صورت کامل انجام شد. حال به پیاده سازی کدهای مربوط به هر یک از لایه‌ها و بخش‌ها می‌پردازیم و از لایه Domain شروع خواهیم کرد.

نظرات خوانندگان

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۰:۲

مباحثی از این دست بسیار مفید و ضروری است و به شدت استقبال می‌کنم از شروع این سری مقالات. البته پیش‌تر هم مطالبی از این دست در سایت ارائه شده است که امیدوارم این سری مقالات بتونه تا حدی پراکندگی مطالب مربوطه را از بین ببرد. فقط لطف بفرمایید در این سری مقالات مرز بندی مشخصی برای برخی مفاهیم در نظر داشته باشید. به عنوان مثال گاهی در یک مقاله مفهوم Repository معادل مفهوم لایه سرویس در مقاله دیگر است. یا Domain Model مرز مشخصی با View Model داشته باشد.
همچنین بحث‌های خوبی مهندس نصیری عزیز در مورد عدم نیاز به ایجاد Repository در مفهوم متداول در هنگام استفاده از EF داشتند که در رفرنس‌های معتبر دیگری هم مشاهده می‌شود. لطفاً در این مورد نیز بحث بیشتری با مرز بندی مشخص داشته باشید.

نویسنده: حسن
تاریخ: ۱۳۹۱/۱۲/۲۸ ۲۲:۵

آیا صرفا تعریف چند ماژول مختلف برنامه را لایه بندی می‌کند و ضمانتی است بر لایه بندی صحیح، یا اینکه استفاده از الگوهای MVC و MVVM می‌توانند ضمانتی باشند بر جدا سازی حداقل لایه نمایشی برنامه از لایه‌های دیگر، حتی اگر تمام اجزای یک برنامه داخل یک اسمبلی اصلی قرار گرفته باشند؟

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۰:۵۳

این سری مقالات جمع بندی کامل معماری لایه بندی نرم افزار است. پس از پایان مقالات یک پروژه کامل رو با معماری منتخب پیاده سازی میکنم تا تمامی شک و شبهات برطرف بشه. در مورد مرزبندی لایه‌ها هم صحبت می‌کنم و مفهوم هر کدام را دقیقا توضیح میدم.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۰:۵۹

اگر مقاله فوق رو با دقت بخونید متوجه میشید که MVC و MVVM در لایه UI پیاده سازی میشن. البته در MVC لایه Model رو به Domain Model و Repository در برخی مواقع لایه Controller رو در لایه Presentation قرار میدن. در MVVM نیز لایه Model در Domain Model و Repository و لایه View Model نیز در لایه Presentation قرار میگیره. همچنین View Modelها نیز در لایه Service قرار میگیرن.
در مورد ماژول بندی هم اگر در مقاله خونده باشید میتونید لایه‌ها رو از طریق پوشه ها، فضای نام و یا پروژه‌ها از هم جدا کنید

نویسنده: حسن
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۰:۱۴

شما در مطلبتون با ضدالگو شروع کردید و عنوان کردید که روش code behind یک سری مشکلاتی رو داره. سؤال من هم این بود که آیا صرفا تعریف چند ماژول جدید می‌تواند ضمانتی باشد بر رفع مشکل code behind یا اینکه با این ماژول‌ها هم نهایتا همان مشکل قبل پابرجا است یا می‌تواند پابرجا باشد.

ضمن اینکه تعریف شما از لایه دقیقا چی هست؟ به نظر فقط تعریف یک اسمبلی در اینجا لایه نام گرفته.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۱:۵۸

صحبت شما کاملاً صحیح است و صرفاً با ماژولار کردن به معماری چند لایه نمی‌رسیم. اما نویسنده مقاله نیز چنین نگفته و در پایان مقاله بحث پایان **ساختار** چند لایه است و نه پایان پروژه. این قسمت اول این سری مقالات است و قطعاً در هنگام پیاده سازی کدهای هر لایه مباحثی مطرح خواهد شد تا تضمین مفهوم مورد نظر شما باشد.

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۲:۳۸

با تشکر از دوست عزیزم جناب آقای آرمان فرقانی با توضیحی که دادند.
یکی از دلایل این شیوه کد نویسی امکان تست نویسی برای هر یک از لایه‌ها و همچنین استقلال لایه‌ها از هم دیگه هست که هر لایه بتونه بدون وجود لایه‌ی دیگه تست بشه. ماژولار کردنه ممکنه مشکل Smart UI رو حل کنه و ممکنه حل نکنه. بستگی به شیوه کد نویسی داره.

نویسنده: بهروز
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۳:۱۱

وقتی نظرات زیر مطلب شما رو میخونم میفهمم که نیاز به این سری آموزشی که دارید ارائه میدید چقدر زیاد احساس میشه
فقط می‌خواستم بگم بر سر این مبحثی که دارید ارائه می‌دید اختلاف بین علما زیاد است!(حتی در عمل و در شرکت‌های نرم افزاری که تا به حال دیدم چه برسد در سطح آموزش...)
امیدوارم این حساسیت رو در نظر بگیرید و همه ما پس از مطالعه این سری آموزشی به فهم مشترک و یکسانی در مورد مفاهیم موجود برسیم
فکر می‌کنم وجود یک پروژه برای دست یافتن به این هدف هم ضروری باشد
باز هم تشکر

نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۱/۱۲/۲۹ ۱۳:۵۹

من هم وقتی کار بر روی این معماری رو شروع کردم با مشکلات زیادی روبرو بودم و خیلی از مسائل برای من هم نامانوس و غیر قابل هضم بود. ولی بعد از اینکه چند پروژه نرم افزاری رو با این معماری پیاده سازی کردم فهم بیشتری نسبت به اون پیدا کردم و خیلی از مشکلات موجود رو با دقت بالا و با در نظر گرفتن تمامی الگوها رفع کردم. امیدوارم این حس مشترک بوجود بیاد. ولی دلیل اصلی ایجاد تکنولوژی‌ها و معماری‌های جدید اختلاف نظر بین علماست. این اختلاف نظر در اکثر مواقع میتونه مفید باشه.
ممنون دوست عزیز

نویسنده: مسعود مشهدی
تاریخ: ۱۳۹۲/۰۱/۰۴ ۱۸:۳۳

با سلام
بابت مطالبتون سپاسگذارم
همون طور که خودتون گفتید نظرات و شیوه‌های متفاوتی در نوع لایه بندی‌ها وجود داره.
در مقام مقایسه لایه بندی زیر چه وجه اشتراک و تفاوتی با لایه بندی شما داره.

```
Application.Web

Application.Manager


Application.DAL


Application.DTO


Application.Core
```

Application.Common

با تشکر

نویسنده: محمود راستین
تاریخ: ۱۳۹۴/۰۵/۲۵ ۳:۳۰

با سلام
ممنون بابت به اشتراک گذاری این مطلب.

میخواستم بدونم چرا ما باید Presentation Layer رو ایجاد کنیم ؟ شما در Presentation Layer اومدید MVP Pattern رو پیاده سازی کردید ، مثلا اگر من از MVC استفاده کنم مگه هر دوی اونها (یعنی MVP و MVC) یک presentation pattern نیستند. پس چرا باید از هردو استفاده کنیم ؟
فرمودید برای آزمون واحد راحتر هستیم وقتی Presentation Layer رو ایجاد کنیم ، مگر MVC این قابلت رو نداره ؟

نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۵/۲۵ ۸:۳۶

این قسمت اول بود و یک طرح کلی غیر وابسته به فناوری خاصی. در قسمت سوم آن به مثال وب فرم‌ها می‌رسید.

نویسنده: محمود راستین
تاریخ: ۱۳۹۴/۰۵/۲۵ ۱۴:۴۷

من قسمت سوم رو هم خوندم. اونجا هم به MVC اشاره ای نشده. اگر در Web Form از الگوی MVP استفاده بشه منطقی به نظر میرسه ولی سوال من این بود وقتی از MVC استفاده کنیم باز هم باید از MVP استفاده کنیم ؟ آیا این منطقیه ؟