**Layered Pattern**

The layered pattern is probably one of the most well-known software architecture patterns. Many developers use it, without really knowing its name. The idea is to split up your code into “layers”, where each layer has a certain responsibility and provides a service to a higher layer.

There isn’t a predefined number of layers, but these are the ones you see most often:

* Presentation or UI layer
* Application layer
* Business or domain layer
* Persistence or data access layer
* Database layer

الگوی لایه ای

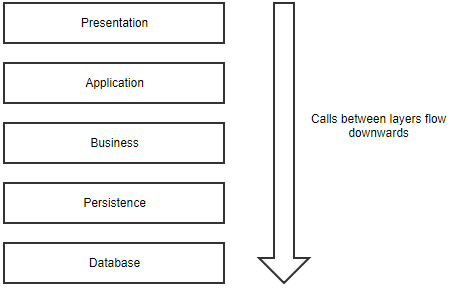
الگوی لایه ای احتمالاً یکی از شناخته شده ترین الگوهای معماری نرم افزار است. بسیاری از توسعه دهندگان بدون اینکه واقعاً نام آن را بدانند از آن استفاده می کنند. ایده این است که کدهای خود را به "لایه ها" تقسیم کنید ، جایی که هر لایه مسئولیت خاصی دارد و به لایه بالاتری سرویس ارائه می دهد.

تعداد مشخصی از لایه ها مشخص نشده است ، اما اینها نمونه هایی هستند که اغلب مشاهده می کنید:

* لایه ارائه یا UI
* سطح کاربردی
* لایه تجاری یا دامنه
* لایه ماندگاری یا دسترسی به داده ها
* لایه پایگاه داده

The idea is that the user initiates a piece of code in the presentation layer by performing some action (e.g. clicking a button). The presentation layer then calls the underlying layer, i.e. the application layer. Then we go into the business layer and finally, the persistence layer stores everything in the database. So higher layers are dependent upon and make calls to the lower layers.

ایده این است که کاربر با انجام برخی اقدامات (به عنوان مثال با کلیک بر روی یک دکمه) یک قطعه کد را در لایه ارائه آماده سازی اولیه کند. لایه ارائه سپس لایه زیرین ، یعنی لایه برنامه را فراخوانی می کند. سپس وارد لایه کسب و کار می شویم و سرانجام ، لایه ماندگاری همه چیز را در پایگاه داده ذخیره می کند. بنابراین لایه های بالاتر به لایه های پایین وابسته شده و تماس برقرار می کنند.

[](https://blog.ndepend.com/wp-content/uploads/layered-1.png)

You will see variations of this, depending on the complexity of the applications. Some applications might omit the application layer, while others add a caching layer. It’s even possible to merge two layers into one. For example, the ActiveRecord pattern combines the business and persistence layers.

بسته به پیچیدگی برنامه ها ، تغییرات مختلف آن را مشاهده خواهید کرد. برخی از برنامه ها ممکن است لایه برنامه را حذف کنند ، در حالی که برخی دیگر یک لایه حافظه پنهان اضافه می کنند. حتی ممکن است دو لایه را در یک هم ادغام کنید. به عنوان مثال ، الگوی ActiveRecord لایه های کسب و کار و پایداری را با هم ترکیب می کند.

### Layer Responsibility

As mentioned, each layer has its own responsibility. The presentation layer contains the graphical design of the application, as well as any code to handle user interaction. You shouldn’t add logic that is not specific to the user interface in this layer.

مسئولیت لایه

همانطور که گفته شد ، هر لایه مسئولیت خاص خود را دارد. لایه ارائه شامل طراحی گرافیکی برنامه و همچنین هر کد برای کنترل تعامل کاربر است. شما نباید منطق خاصی را در رابط کاربری این لایه اضافه کنید.

The business layer is where you put the models and logic that is specific to the business problem you are trying to solve.

The application layer sits between the presentation layer and the business layer. On the one hand, it provides an abstraction so that the presentation layer doesn’t need to know the business layer. In theory, you could change the technology stack of the presentation layer without changing anything else in your application (e.g. change from WinForms to WPF). On the other hand, the application layer provides a place to put certain coordination logic that doesn’t fit in the business or presentation layer.

لایه کسب و کار جایی است که شما مدل ها و منطق خاصی را برای مشکل تجاری که می خواهید حل کنید قرار می دهید.

لایه برنامه بین لایه ارائه و لایه تجاری قرار می گیرد. از یک طرف ، این انتزاع را ایجاد می کند تا لایه ارائه نیازی به لایه کسب و کار نداشته باشد. در تئوری ، شما می توانید پشته فناوری لایه ارائه را بدون تغییر هر چیز دیگری در برنامه خود تغییر دهید (به عنوان مثال تغییر از WinForms به WPF) از طرف دیگر ، لایه برنامه مکانی را برای قرار دادن منطق هماهنگی خاصی فراهم می کند که در لایه مشاغل یا ارائه ارائه نمی شود.

Finally, the persistence layer contains the code to access the database layer. The database layer is the underlying database technology (e.g. SQL Server, MongoDB). The persistence layer is the set of code to manipulate the database: SQL statements, connection details, etc.

سرانجام ، لایه ماندگاری شامل کد برای دسترسی به لایه پایگاه داده است. لایه پایگاه داده زیربنای فناوری پایگاه داده است (به عنوان مثال SQL Server ، MongoDB ) لایه ماندگاری مجموعه ای از کد ها برای دستکاری پایگاه داده است: جملات SQL ، جزئیات اتصال و غیره.

### Advantages

* Most developers are familiar with this pattern.
* It provides an easy way of writing a well-organized and testable application.

مزایا

• اکثر توسعه دهندگان با این الگوی آشنا هستند.

• این یک راه آسان برای نوشتن یک برنامه به خوبی سازمان یافته و قابل آزمایش را فراهم می کند.

### Disadvantages

* It tends to lead to monolithic applications that are hard to split up afterward.
* Developers often find themselves writing a lot of code to pass through the different layers, without adding any value in these layers. If all you are doing is writing a simple CRUD application, the layered pattern might be overkill for you.

معایب

• منجر به برنامه های monolithic می شود که بعداً جداسازی آن سخت است.

• توسعه دهندگان غالباً اینگونه تصور میکنند كه كد زیادی را برای گذر از لایه های مختلف می نویسند ، بدون آنكه هیچ ارزشی در این لایه ها اضافه كنند. اگر تنها کاری که می کنید نوشتن یک برنامه CRUD ساده است ، الگوی لایه ای ممکن است برای شما زیاد باشد.

<https://dzone.com/articles/software-architecture-the-5-patterns-you-need-to-k>