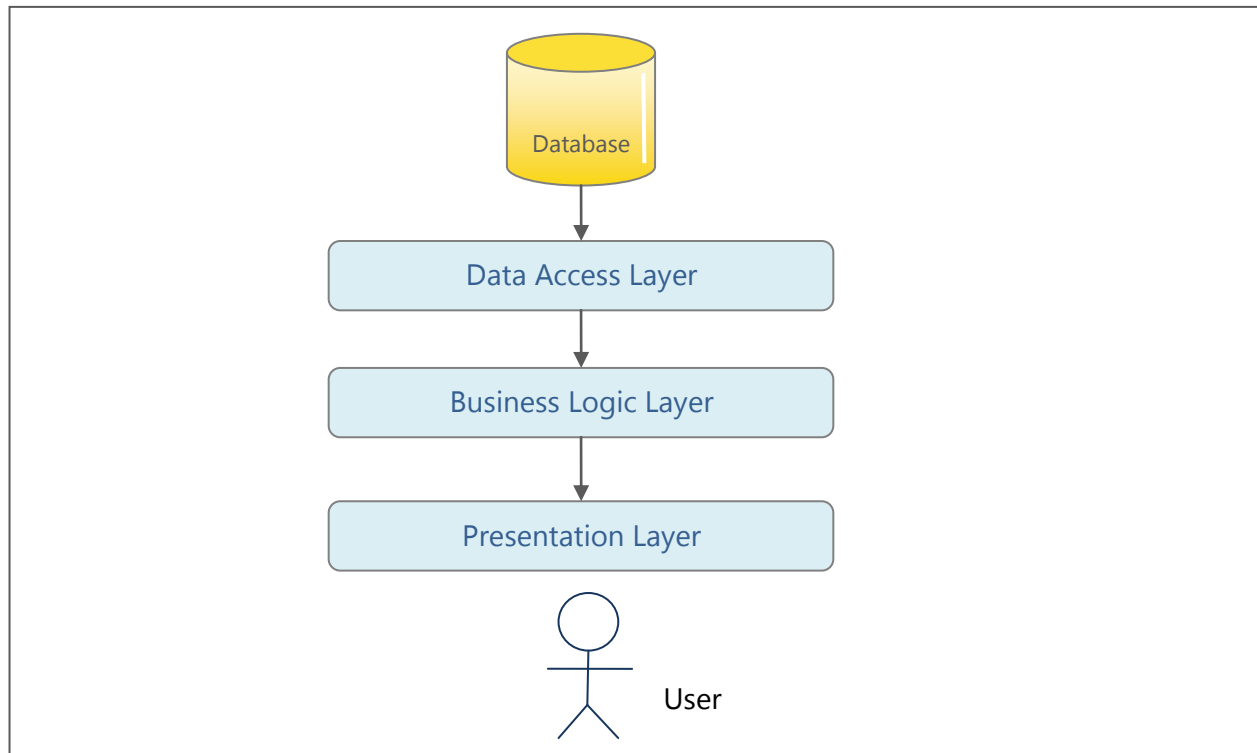


[منهجية العمل في تصميم تطبيقات قواعد البيانات]

الكثير منّا، أو كلنا قد عملنا على تطبيقات قواعد البيانات، إما لأنفسنا أو لجهات معينة، ودائماً ما نواجه العديد من المشاكل في تصميم وتطبيق هذه التطبيقات، إن لم تكن تلك المشاكل في الربط مع قاعدة البيانات، تكون في ربط البيانات مع الواجهات، أو إضافة بيانات من المستخدم إلى قاعدة البيانات، بالمحصلة هناك جملة كبيرة نوعاً ما من المشاكل التي قد تواجهنا في مراحل تصميم هكذا تطبيقات.

لكن السؤال المطروح: ما هو سبب تلك المشاكل و لماذا تنتج وكيف يمكن التغلب عليها؟

في الواقع، أكثر الأخطاء التي تواجه المبرمج هي من صنع نفسه! والسبب يكون دائماً عدم اتباعه لمنهجية عمل مناسبة تؤمن له تصميم أكثر قوة وسهولة في العمل، بل على العكس يقوم مباشرة بفتح بيئة التطوير، ثم عمل بعض النوافذ وإضافة بعض العناصر، والاتصال بقاعدة البيانات ومعالجة العمليات الباقية، كل هذا يتم متراكباً فوق بعضه البعض. ومن خلال هذا التراكم في العمل، تنتج الكثير من الأخطاء، نحن بغنى عنها. لذلك سنقوم في هذه المقالة بشرح بعض المفاهيم الأساسية لعمل هذه التطبيقات بكل قوة وسهولة. الكلام والمفاهيم ينطبق على جميع بيئات التطوير، لكننا سنركز على بيئة ميكروسوفت [NET Platform]، حيث سنستعين ببعض تقنيات هذه البيئة بشكل بسيط، وسنقوم في المقالة القادمة بتطبيق جميع هذه المفاهيم على عدة مشاريع مثل Windows Forms Application، ASP.NET Application و WPF Application. لنلقي نظرة على المخطط التالي الذي يوضح تسلسل الطبقات بدءاً من قاعدة البيانات [الفيزيائية] وانتهاءً بمستخدم النظام. وكما تحدثنا فإن هذه الهرمية يمكن تطبيقها على أي مخدم قواعد بيانات وعلى أي بيئة تطوير أخرى.



Data Access Layer

هذه الطبقة هي المسؤولة عن إنشاء الإتصال مع قاعدة البيانات، وإجراء العمليات عليها من إضافة وتعديل وحذف وجلب للبيانات وهو ما يطلق عليه إسم CRUD [Create Retrieve Update Delete]، والمهم في هذه الطبقة هو فقط التخاطب مع قاعدة البيانات بدون أي أليات تحقق من البيانات أو رسائل مرجعة للمستخدم. فإذا ما قلنا [Data Access Layer]، فإن معناها محدد كإسمها، الوصول للبيانات فقط، وليس التحقق منها أو إجراء أي عمليات أخرى، لأن كل ذلك من مهمة الطبقة الثانية. هكذا يكون لدينا آلية وصول للبيانات مفصولة عن قواعد أو ضوابط التطبيق، ومفصولة عن التقنيات الأخرى، فمثلاً يمكن استخدام هذه الطبقة في تطبيقات ASP.NET أو WPF أو Windows Forms وما إلى ذلك من التقنيات الأخرى.

ما هي طبيعة هذه الطبقة، وكيف يمكننا إنشاءها؟

في الواقع كل لغة برمجة تزودنا بتقنيات معينة لإنشاء هذه الطبقة، وبالطبع يمكننا إنشاؤها يدوياً كل ذلك يعود لطبيعة البيئة المستخدمة، ولطبيعة متطلبات المشروع، ورغبة المستخدم.

بما أننا سنستخدم تقنيات .NET. في أمثلتنا سأقوم بعرض كيفية إنشاءها بواسطة هذه التقنيات، بأسلوب توضيحي فقط، وكما تحدثنا سيأتي تطبيقها بالتفصيل وإستخدامها في مقالات قادمة بإذن الله.

تزدنا NET. بعدة تقنيات للعمل، اساسها هو ADO.NET [Active Data Object.NET] تتفرع منه التقنيات الأخرى.

1. ADO.NET Entity Framework Model

وهي عبارة عن عنصر جديد يتم إضافته للمشروع، ثم يظهر لنا العديد من الخيارات لكي نحدد الإتصال مع قاعدة البيانات، ثم الربط مع الجداول المحددة، وهذ التقنية ستقوم تلقائيا بإنشاء كلاسات للجداول وإذا كان هناك علاقات بين الجداول سيضمنها ضمن الكلاسات أيضاً. عند الانتهاء من التصميم، نكون قد حصلنا على طبقتنا المطلوبة، وبالطبع تحوي هذه الطبقة على كلاس خاص بعمليات CURD تضاف تلقائياً.

2. LINQ to Classes

وهي مشابهة للتقنية السابقة من حيث التصميم والتعامل، لكن الفرق يكمن في التصميم الداخلي لها، وفي طريقة تعاملها مع قاعدة البيانات، حيث تقوم بالتخاطب مع قاعدة البيانات عن طريق ما يسمى ب SqlConnection بينما تقوم التقنية السابقة بالتخاطب معها بواسطة ما يسمى ب EntityClient.

3. Data Set

أيضاً الكثير منا يعرف ما هي الـ DataSet وهي عبارة عن DataAccessLayer بوضع Disconnected عن قاعدة البيانات، وهي أيضاً عنصر يضاف للمشروع ثم نقوم بتصميم الجداول عن طريق إضافة TableDataAdpater، وهناك الكثير من الأدوات تساعدنا في إنشاء الإستعلامات queries لملىء تلك الجداول بالبيانات المطلوبة، بدل أن نعرف كل ذلك بإيدينا، فالطريقة المتبعة، أن نقوم بتعريف عنصر من DataSet في مكان ما في الكود ثم تعريف TableDataAdapter ثم مل DataSet بالجدول المناسب، وفيما لو انتقلنا إلى نافذة أخرى للعمل أو لجز آخر من الكود، فإننا سنضطر لإعادة هذه التعريفات، مما يجعل العمل أكثر صعوبة وممل. لكن بتعريفها كطبقة مستقلة كما تحدثنا، هو الطريقة السليمة والأفضل للعمل.

4. إنشاء الطبقة يدوياً

كأن نعرف كلاس معين، ونعرف ضمنه إعدادات الاتصال مع قاعدة البيانات ودوال التخاطب مع قاعدة البيانات، عمليات CRUD. ولا يخفى أننا بحاجة لإنشاء كلاسات للجداول يدوياً أيضاً، فمن أكثر المشاكل التي يقع فيها المبرمج في إنشاء تطبيقاته، أنه يقوم بإحضار البيانات من قاعدة البيانات بشكل متغيرات فقط، كأن

يحضر حقل الإسم والعنوان فقط، ويخزن كل واحدة ضمن متغير. هنا تكمن المشاكل والصعوبة، بينما لو عرف كلاس لكل جدول يكون التعامل أسهل في جميع الحالات.

في جميع الحالات السابقة نحصل على شيء موحد، عبارة عن كلاس يحوي دوال التعامل مع قاعدة البيانات، بغض النظر عن أي ضوابط أو قواعد يفرضها المشروع علينا. فقط إدخال بيانات معينة أو جلب بيانات معينة ببساطة.

Business Logic Layer

من المعلوم أن كل تطبيق له قواعد خاصة، ومنهجية عمل معينة؛ أي هناك تطبيقات تفرض بعض الشروط على البيانات الداخلة، مثلاً لا تقبل عدد سالب، أو حقل فارغ، وما إلى ذلك من الشروط. هذه الطبقة هي المسؤولة عن إنجاز هذه المهمة، وبالتالي تضمن وصول البيانات الصحيحة إلى قاعدة البيانات.

كيف يتم إنشاء هذه الطبقة؟

يمكن اعتبار هذه الطبقة كلاس خاص، يحوي على Object من نوع الطبقة الأولى، وبالتالي هو المسؤول عن التخابط مع الطبقة الأولى، وهو صلة الوصل أيضاً مع User Interface. ونقوم بتعريف كل الضوابط والشروط والقواعد على البيانات ضمنه، مثلاً لنتخيل ميثود ضمنه تدعى AddUser، تقوم بأخذ بيانات من المستخدم وتحاول حفظها إلى قاعدة البيانات، فتقوم هذه الميثود بأخذ متغيرات مثل Username و Password و ConfirmedPassword. كما أدخلها المستخدم في عناصر الواجهات، فتقوم هذه الميثود بالتحقق من أن User Name يحوي على قيمة وليس فارغ، ثم يقوم بالتحقق فيما إذا كان الإسم محجوز مسبقاً أم لا، ويقوم بالتحقق من توافق كلمتي المرور، بعد التحقق من صحة كل هذه الشروط، يأتي الوقت لطلب ميثود من الطبقة الأولى لأخذ هذه البيانات [الصحيحة] وإرسالها لقاعدة البيانات لحفظها. وهكذا نلاحظ أننا قمنا بتوزيع المهام، فواجهات المستخدم لم تعد تضع الشروط الخاصة بالمشروع، بل هي مجموعة في هذه الطبقة، ومرة أخرى يمكننا استخدام الطبقتين الأولى والثانية في التقنيات الأخرى، كون هناك استقلالية في العمل، وتوزيع في المهام. ولا يوجد ارتباط مع واجهات المستخدم ابداً، فإسم المستخدم مثلاً لا يهمنا في هذه الطبقة هل هو قادم من TextBox ضمن مشروع Windows Forms أو من ASP.NET. في النهاية هو [string]. أما إذا كان هناك أي خلل في الشروط فإننا سنقوم بإفشال عمل هذه الميثود وإرسال رسالة إلى

واجهة المستخدم، أغلب الرسائل وأفضل الطرق يتم عن طريق Exception، مثلاً إذا كان إسم المستخدم محجوز مسبقاً، نقوم مباشرة بعد الشرط بعمل throw لـ Exception بالشكل:

```
throw new Exception("User name is already exist! Please try another name");
```

طبعاً واجهة المستخدم هي المسؤولة عن التقاط هذا الخطأ، وعرضه بطريقة مناسبة للمستخدم.

Presentation Layer

في هذه الطبقة نقوم بتصميم واجهات المستخدم، وضمن هذه الواجهات نقوم بطلب Object من الطبقة الثانية وإرسال البيانات إليها بكل بساطة، ونستطيع ملاحظة الرسائل القادمة من الطبقة الثانية. وهذه الطبقة كما قلنا هي نوع المشروع الذي نقوم بالعمل عليه هل هو تطبيق ويب أو تطبيق سطح مكتب أو تطبيق موبايل، لا يهم الآن ما هو نوع المشروع ما دمنا قمنا بفصل الطبقات، فلو أردنا تغيير المشروع أو إنشاء مشروعين مختلفين [تطبيق ويب وتطبيق سطح مكتب] بنفس السلوك، يتم ذلك بكل سهولة.

تجدر الإشارة هنا إلى أن الطبقة الأولى والثانية بإحتوائهما على آليات التعامل مع البيانات، يمكننا فصلهما في مشروع مستقل أو فصل كل طبقة في مشروع مستقل من نوع [Class Library] مثلاً، وبالتالي يصبح إعادة الاستخدام في منتهى السهولة. أما إذا أردنا جمع الطبقات في مشروع واحد، فالأفضل وضع الطبقة الأولى والثانية كل واحدة في مجلد مستقل، وأن نسمي المجلد الأول [DataAccessLayer] والمجلد الثاني [BusinessLogicLayer]. وبهذا يكون لدينا فصل حقيقي للطبقات.

كل النقاش السابق عبارة عن منهجيات تم وضعها من قبل أناس واجهوا تلك المشاكل، وبحكم خبرتهم وضعوا هذه المنهجيات، ليستفيد منها غيرهم، فعلينا الإستفادة من أخطاء وتجارب الآخرين، لا أن نكررها عدة مرات حتى نصل إلى ما وصلوا إليه سابقاً.