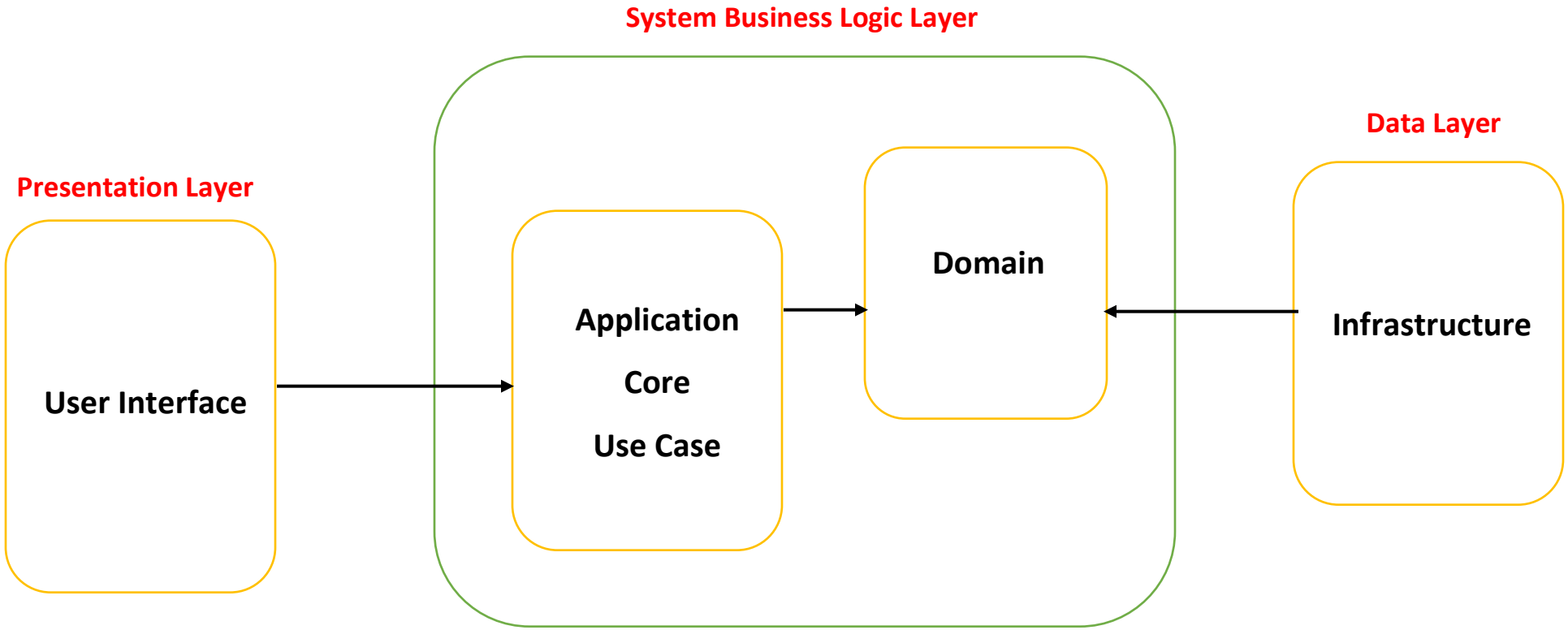
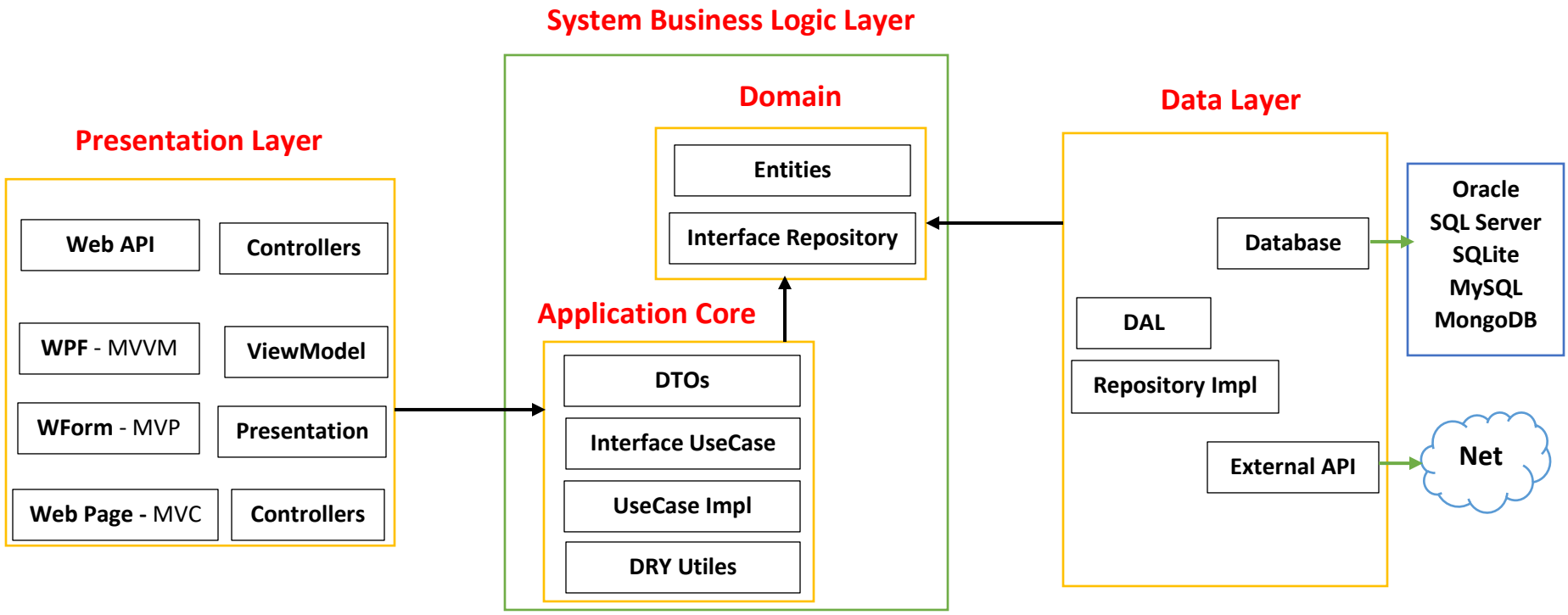
الهيكلية النظيفة (Clean Architecture) للطبقات وأجزاء النظام البرمجي القابلة للصيانة وإعادة الإستخدام.
إعداد وتلخيص المهندسين :- م/ بشير حزام – م/ وائل الشميري – م/ هشام شرف؛ والمعتمدة في مشاريعهم العملية.
من المعلوم ان الهيكلية النظيفة تتكون من اربع طبقات مهمة (Presentation Layer - Application Layer - Domain Layer - Infrastructure Layer) وهكذا.



نلاحظ ان كلا من طبقة (Application, Infrastructure) تعتمد على طبقة (Domain) ولا يوجد اي علاقة او اعتمادية بين طبقة Application و طبقة Infrastructure فكل طبقة مستقلة عن الآخرى، بينما طبقة (Presentation) تعتمد على طبقة (Application) فقط وليس لها علاقة مع طبقة البنية التحتية (Infrastructure) مهما تغيرت فيما بعد مستقبلا.

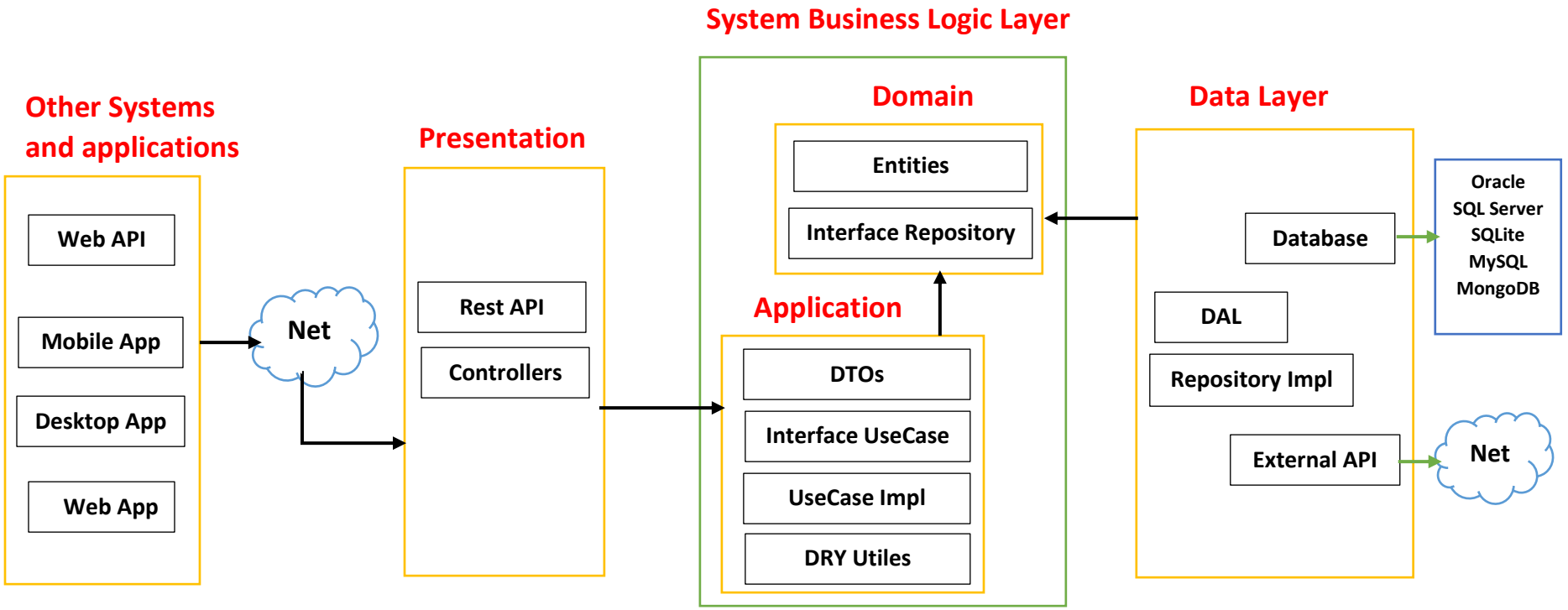
توضيح وتفاصيل أكثر

طبقة (System Business Logic Layer) مستقلة عن نوع قاعدة البيانات ونوع واجهة المستخدم النهائي، ليس لها أي علاقة بنوع مصدر بيانات النظام سوى كانت ملفات او قاعدة بيانات او خدمات وصول خارجية API، ومهما كان نوع قاعدة البيانات ومهما كان نوع التعامل مع قاعدة البيانات سوى إجراءات مخزنة او SQL الاساسية من البرنامج.
لوجيك او منطق النظام ثابت (تتغير في حالة تعديل آلية العمل بها في ارض الواقع) لا يتغير مهما تغييرت مصدر البيانات البنية التحتية (Infrastructure) ومهما تنوعت او تغييرت طبقة المستخدم النهائي.



شكل أخر والذي يتكون من مستويين (two Tiers) التالي :

طبقة (Business Logic) الخاصة بالنظام تكون واحدة ومركزية مهما تعددت او تغييرت طبقة البنية التحتية مزود البيانات (Data) ومهما تعددت طبقة المستخدم (Presentation) موقع ويب او تطبيق موبايل او برنامج Desktop يكون الإتصال و التدفق والوصول الى نفس طبقة (Business Logic) الخاصة بالنظام واي تعديل او تحديث للوجيك النظام يؤثر على كل الطرفيات او كل برامج طبقة المستخدم النهائي.



منطق النظام وفحصه أو التأكد منه وفحص مدخلات المستخدم ايضا تكون كلها في طبقة (Application) ولو تكررت او نسخة الى طبقة (Presentation) ممكن كحماية من جهة الكلاينت Client Side، فالمهم والأهم تكون في طبقة (Application) وهي بمثابة Server Side تحقق من جهة السيرفر (الخادم) لمنطق النظام وهذا يحقق مبدأ الهيكلية النظيفة.

جدول عن طبقات وأجزاء النظام مرتبة من اول جزء اساسي وحتى اخر جزء:-

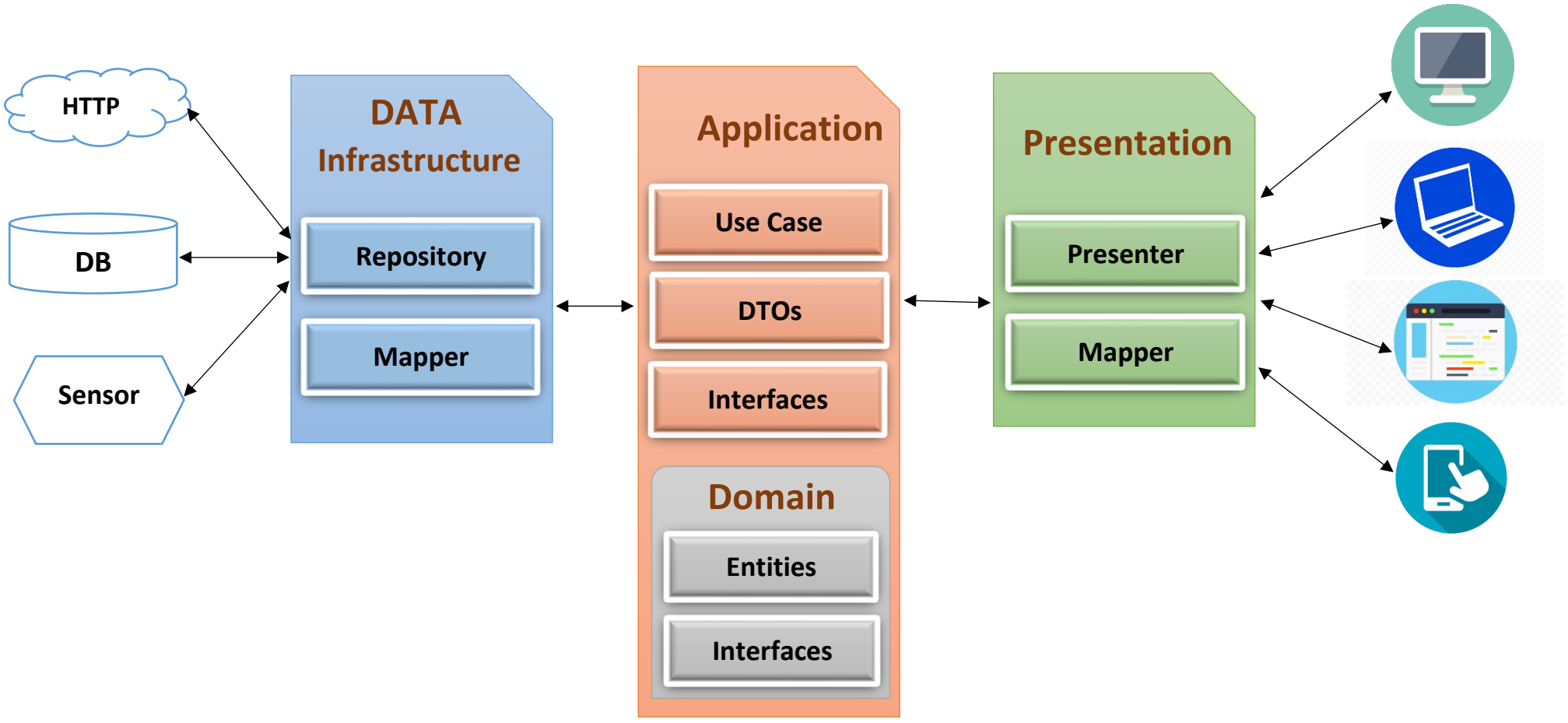
م	اسم الجزء	الأهمية	الإعتمادية	المحتوى	الشرح	نوع المشروع
1	Domain	اساسي	لا يوجد	Entities Interface Repository	طبقة مستقلة عن نوع قاعدة البيانات ونوع واجهة المستخدم النهائي. يستهدف : أساس النظام	مكتبة DLL
2	Core *or* Use Case *or* Application	اساسي	(1,5)	DTOs Interface UC UC Classes Utiles (DRY) UtilesLazy	طبقة مستقلة مع الطبقة التي قبلها عن نوع قاعدة البيانات ونوع واجهة المستخدم النهائي. يستهدف : اساس ومنطق عمل النظام تسمى طبقة Application Core	مكتبة
3	Data *or* Infrastructure	اساسي	(1,5)	DAL Repository Classes Database External Rest API Extensions	طبقة تتعلق بمزود بيانات النظام. يستهدف : مزود البيانات للنظام الخارجية سوى قاعدة بيانات او خدمة وصول بعيدة. وتسمى طبقة Infrastructure البنية التحتية للنظام.	مكتبة
4	UI *or* Presentation	اساسي	(2,5,6)	View or Controllers DRY Report	طبقة تتعلق بواجهة المستخدم النهائي يستهدف : واجهة المستخدم النهائي	Web API Web App Desktop App Mobile App
5	Shared	إضافية	لا يوجد	DRY Enums ValueObjects GeneralConstants	طبقة إضافة ومشتركة لكل الطبقات الأساسية ما عدا الطبقة الأولى. كل الأشياء المشتركة في الأجزاء الأساسية	مكتبة
6	IOC	إضافية	(1,2,3)	Inversion of Control	لإنشاء instance لجميع الكلاسات لطبقات (2,3) بفريم ورك IOC، والتي يدعم بواسطة نمط تصميم حقن التبعية (DI) Dependency Injection	مكتبة
7						

طبقة Domain فيها تجريد طبقة البيانات لكن هنا من المفترض ان فيها ايضا تجريد المنطق لكن لا نريد ان تصل طبقة المستخدم النهائي للطبقة Domain كحماية.
طبقة IOC قد تكون طبقة للكل (برنامج مستقل ضمن السلوشن Solution) او كلاس في كل طبقة على حده.

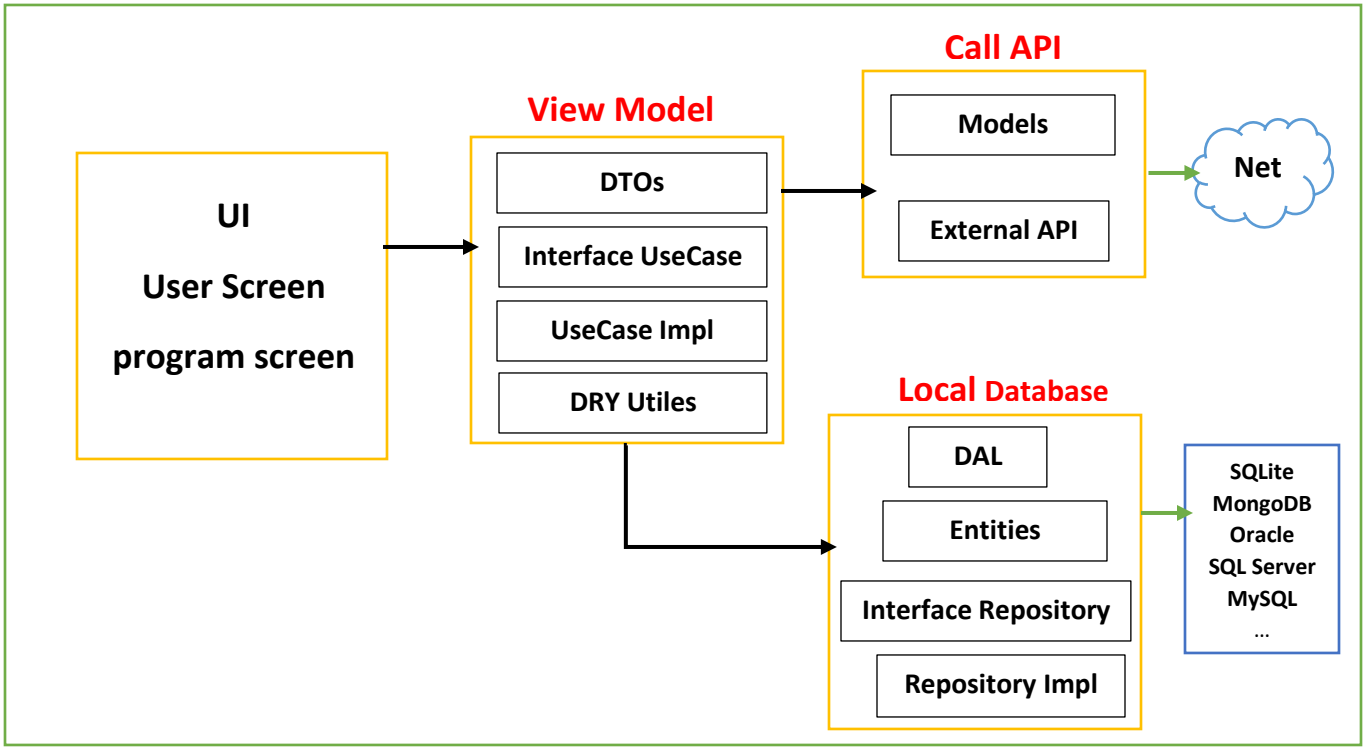
شرح الجزئيات الأساسية في طبقات النظام

الطبقة	الجزئية	المحتوى
Domain	Entities	تمثل بيانات وخصائص كائن محدد سوى كان جدول او عمليات خدمة بعيدة لها نفس البيانات كل جدول له كلاس خاص به يمثل خصائص الجدول. لم يتم تعديلها إلا اذا تغييرت بنية كائن مصدر البيانات كالجدول مثلا.
	Interface Repository	تمثل الوظائف او الدوال الخاصة بكائن محدد سوى كان جدول او عمليات خدمة بعيدة. كل جدول له واجهة Interface خاص به يمثل العمليات التي تنفذ على هذا الجدول. التنفيذ او الكود التابع لها يكون في طبقة Data في جزء Repository
Use Case Application	DTOs	تمثل خصائص عملية كاملة خاصة بمنطق النظام والتي ينفذها المستخدم النهائي يوجد فيها دوال التحويل من كلاس Entities الى نفس هذا الكلاس DTOs والعكس. خصائص هذه الكلاس قد تكون من اكثر من كلاس Entities حسب متطلب العملية. يستفيد منها طبقة المستخدم النهائي لانه لا يوجد معه وصول الى كلاسات Entities التي تمثل كائنات مجال النظام.
	Interface UC	تمثل الوظائف وعمليات النظام الحقيقية سوى كانت عملية واحدة او أكثر من عملية متصلة مع بعضها البعض. التنفيذ او الكود التابع لها يكون في الجزئية التالية بنفس هذه الطبقة. يستفيد منها طبقة المستخدم النهائي للوصول الى تنفيذ منطق ولوجيك النظام.
	Use Case Impl	هي تنفيذ للدوال الموجودة في جزئية Interface UC بنفس هذه الطبقة. يستقبل الطلبات من طبقة المستخدم النهائي بواسطة واجهة Interface ثم يعالجها و يرجع النتيجة. قد يرث الكلاس الواحد من واجهة Interface واحدة او اكثر من واجهة بنفس الوقت وحسب الحاجة. يحول مدخلات المستخدم النهائي التي من نوع DTOs الى نوع Entities للتمريرها الى طبقة Infrastructure. يتصل بطبقة البنية التحتية بواسطة واجهة Interface الموجودة في جزئية Interface Repository في طبقة Domain.
Infrastructure Data	DAL	عبارة عن كلاس وظيفته الرئيسية الوصول الى مصدر مزود البيانات سوى كان المصدر قاعدة بيانات (Oracle, Sql Server, MongoDB) او خدمة وصول بعيدة (.API, Web Service, etc) . ثم تنفيذ العمليات (CRUD) على هذا المصدر.
	Repository Impl	عبارة عن تنفيذ للعمليات كائن محدد سوى كان جدول او مجموعة من الوظائف البعيدة. كل جدول له كلاس خاص بكل العمليات التى تنفذ على هذا الجدول (استرجاع، اضافة، تعديل، حذف، ...الخ). هى تنفيذ للدوال الموجودة في جزئية Interface Repository في طبقة Domain.

جدول يوضح محتويات الجزئيات الاساسية في الهيكلية النظيفة.


HTTP
DB
Sensor
DATA
Infrastructure
Repository
Mapper
Application
Use Case
DTOs
Interfaces
Domain
Entities
Interfaces
Presentation
Presenter
Mapper

هيكلية جزئية برنامج المستخدم النهائي (Presentation) التي تتصل بخدمة بعيدة وليس فيها كود منطق النظام لانه في الخدمة البعيدة API.
في حالة جزئية المستخدم تتصل بخدمة بعيدة او أكثر وقد يكون لها قاعدة بيانات محلية تحفظ البيانات لكي تتصفحها بدون نت او ليس لها قاعدة بيانات محلية تكون الهيكلية لها بالشكل التالي :-



المعمارية النظيفة Clean Architecture
فكرة الهيكلية النظيفة ان كود منطق (لوجيك) النظام يكون في طبقة واحدة وبدون تكرار بالإضافة الى التحققات والشروط ايضا ويكون مجرد ومستقل على الطبقات الخارجية سوى طبقة البنية التحتية او طبقة المستخدم النهائي ولا يوجد فيها كود يختص بنوع طبقة خارجية محددة بل يكون كود منطق النظام عام وتتناسب معه جميع الطبقات الخارجية مهما كان نوعها.

يجلب لنا هذا النهج العديد من المزايا إلى جانب بساطة قراءة الهيكل ويصبح تطبيقنا لهذة المعمارية (المعمارية النظيفة) :-
1- إستقلالية كود منطق (لوجيك) النظام في طبقة وجزئية واحدة مجردة ومستقلة ولا يوجد تكرار للكود منطق النظام في طبقات وجزئيات متفرقة.
2- مستقلة عن واجهة وتطبيقات المستخدم النهائي.
3- مستقلة عن خدمات البنية التحتية وعن قاعدة البيانات ونوعها.
4- سهولة التوسعة للنظام والصيانة والاختبار لوحدات النظام وقابل للإعادة الإستخدام.

للتحقيق الغرض الكامل لهذة الهيكلية يجب تطبيق وإستخدام جميع مبادئ ونماذج تصميم البرمجيات (Design Principles & Design Patterns) الحديثة والناجحة والمجربة من قبل مهندسي البرمجيات وأثبتت نجاحها في حقل هندسة البرمجيات مثل (DRY, SOLID, Repository Pattern, Design Patterns of 23 for GOF) وغيرها من المبادئ والنماذج الحديثة والتي قد تطرأ علينا في يوما ما وهي ناجحة.

معايير تطوير نظام ناجح

لتطوير نظام ناجح نعتمد على الأتي :-
1- المعمارية (Architecture) المختارة لبناء النظام : اختيار المعمارية هو الخطوة الأولى في تصميم النظام او التطبيق بناءً على المتطلبات؛ مثال: MVC، WEBAPI، Clean Architecture، MVVM، MVP وغيرها.
2- مبادئ التصميم (Design Principles) : يجب أن تتبع عملية تطوير التطبيق مبادئ التصميم مثل مبادئ solid و DRY وغيرها.
3- أنماط التصميم (Design Patterns) : نحتاج إلى اختيار أنماط التصميم الصحيحة والمناسبة لبناء البرنامج.

الافضل يكون معنا طبقتين منفصلة او مستويين two tier :-
1- الطبقة الاولى منطق العمل خاصة بمنطق ولوجيك النظام بشكل عام وليس لها علاقة مع الطبقة الثانية مهما كان نوع الطبقة الثانية موقع ويب او تطبيق موبايل او برنامج سطح مكتب.
2- الطبقة الثانية طبقة المستخدم النهائي تطبيق المستخدم سوى كان موقع ويب او تطبيق موبايل او برنامج سطح مكتب.

فالاتصال بين الطبقتين يكون بأي طريقة سوى كان بالمكاتب او بخدمة الاتصال البعيدة API وهي الأفضل وهي تكون بجانب الطبقة الأولى.
فكل طبقة لها المعمارية المناسبة لها لذا قد نستخدم في الطبقة الاولى الهيكلية النظيفة او أي هيكلية مناسبة يحددها المطور بناءً على متطلبات النظام، ونستخدم في الطبقة الثانية مثلاً MVC للموقع الويب و MVVM وللتطبيق الموبايل وهكذا حسب ما يتطلب ويحددها المطور.
قد يوجد نموذج تصميم حين تستخدمه انه يسبب كسر نموذج اخر وهذا ممكن لانه يحل مشكلة اكبر.

الفرق بين مصطلح tier و layer

(tier) تتعلق بالمعمارية المادية physical architecture.

(layer) تتعلق بالمعمارية المنطقية logical architecture.

تشير مصطلح (layer) الطبقة إلى أجزاء من البرامج يتم فصلها منطقيًا ، ولكنها تعيش عادةً في نفس العملية والجهاز تكون طبقات الى بعضها البعض بنفس الجهاز والترابط بينها يكون بواسطة اضافة مكتبة للطبقة من الطبقة العليا.

بينما تشير (tier) الطبقة إلى أجزاء من البرامج التي تعيش في عمليات أو مجالات أو آلات وأجهزة مختلفة والتواصل فيما بينها يكون بواسطة الاتصال عن بعد كالشبكة مثلا.

يمكن أن يكون لديك طبقات متعددة (layer) على نفس المستوى (tier) الواحد وهكذا.

الطبقة (tier) تشير إلى الفصل المادي (physical separation)؛ طبقة (layer) تدور حول الفصل المنطقي (logical separation).

الطبقات (layer) المنطقية هي مجرد طريقة لتنظيم التعليمات البرمجية الخاصة بك، فهي التنظيم المنطقي للكود؛ فهي لا تعمل على أجهزة كمبيوتر مختلفة أو في عمليات مختلفة على جهاز كمبيوتر واحد فكلها تعمل وتشتغل معا مع بعضها البعض بنفس الوقت لذا لا تعمل اذا فصلت عن بعضها او حدث خطأ او تعطل للطبقة واحده منها.

فإن المستويات (tier) المادية تتعلق فقط بمكان تشغيل الكود فهي آلة فعلية او خادم، فهي النشر المادي للطبقات (layer).

يوجد نوعين من التعامل مع كلاسات (Use Case) وهي كالتالي :-

- **النوع الأول بناء كلاس (Use Case) لكل واجهة مستخدم نهائية بحيث تحتوي على جميع دوال واجهة المستخدم النهائي المحددة، وهذ النوع غير جيد.**
- **النوع الثاني بناء كلاس (Use Case) لكل وظيفة او دالة بحيث يكون الكلاس له وظيفة واحدة فقط مستقلة او مجموعة من الوظائف تعمل معا في إستدعاء واحد ولا يمكن لاي وظيفة ان تعمل بدون ما تحتاج للوظيفة الآخرى في نفس الكلاس الواحد؛ هذا مشابه كما يتم في الهيكلية النظيفة، وهذا النوع هو الأفضل والجيد ويحقق مبادئ البرمجة الحديثة مثل مبادئ الـ Solid ومبدأ DRY وغيرها؛ وهذا النوع يتطلب طبقة جديدة وسيطة (نستخدم نمط Facade Pattern) بين طبقة (Use Case) وبين طبقة المستخدم (واجهات المستخدم النهائي) حيث هذه الطبقة الجديدة تعتبر طبقة عبور فقط وفيها جميع دوال واحتياجات كل واجهة مستخدم على حده والتي تستدعي اكثر من كلاس من طبقة (Use Case) لكي تكون كلاس واحد يمرر الى طبقة المستخدم (Presentation) مهما كان نوعه تطبيق ويب او ديسك توب او خدمة API وهكذا.**

النهاية.

إعداد المهندسين :-

م/ بشير حزام - م/ وائل الشميري - م/ هشام شرف

للتواصل والدعم والإستشارات :-

00967 733260613 – 00967 773530076 – 00967 773950771