8/2/2025

تقرير لجنة الكود

صيدلية مركز البحوث

إعداد: كرستينا خيمي

بإشراف: م. محمود إلياس

الفصل الأول

التعريف بالمشروع

يتضمن هذا الفصل التعريف بالمشروع ومتطلباته.

1.1 مقدمة

تعتبر صيدلية مركز البحوث العلمية من المراكز الخدمية الرئيسية في المؤسسة، تقتصر مهمتها الأساسية على صرف الأدوية للموظفين وأفراد عائلاتهم المباشرين، معتمدة بذلك على نظام تغطية مالية داخلية. في ظل التطور السريع في مجال هندسة البرمجيات وظهور أطر عمل حديثة توفر أداءً عالياً وتجربة مستخدم أفضل، أصبح النظام الحالي المستخدم في الصيدلية قديماً ومحدود القدرات، مع توقف الدعم الفني له منذ فترة طويلة. من هذا المنطلق، جاء المشروع كحل بديل لإعادة بناء وتطوير نظام الصيدلية بشكل كامل. يهدف المشروع إلى تقديم نظام ويب متكامل وعصري، وذلك بهدف أتمتة العمليات اليومية، وتحسين دقة إدارة المخزون، وتقديم واجهات سهلة الاستخدام تعزز من كفاءة العمل.

2.1 الهدف من المشروع

إن الهدف الأساسي من المشروع هو إنشاء نظام برمجي متكامل وموثوق لأتمتة العمليات اليومية في صيدلية مركز البحوث، مما يؤدي إلى تحسين الكفاءة والدقة وتقليل الأخطاء اليدوية. يسعى النظام لتحقيق ذلك من خلال مركزية إدارة الأدوية عبر توفير قاعدة بيانات شاملة يمكن إدارتها بسهولة. كما يركز على أتمتة إدارة المخزون، حيث يتم تتبع الكميات بشكل دقيق ولحظي وإدارة دفعات الأدوية بناءً على تاريخ انتهاء الصلاحية لضمان تطبيق مبدأ (FIFO) بالإضافة إلى ذلك، يهدف النظام إلى تحسين عملية صرف الوصفات الطبية عن طريق تبسيطها وتسريعها، مع ضمان التطبيق الدقيق لقواعد التغطية المالية الداخلية. ولدعم اتخاذ القرار، سيوفر النظام القدرة على إنشاء تقارير تحليلية إدارية ومالية، وأخيراً، سيحقق التكامل مع أنظمة المنظمة الأخرى من خلال توفير واجهات برمجية (APIs) آمنة.

3.1 المتطلبات الوظيفية

1. يجب أن يسمح النظام للمسؤولين بإنشاء حسابات مستخدمين.
2. يجب أن يسمح النظام للمسؤولين بتعديل معلومات حسابات المستخدمين.
3. يجب أن يفرض النظام المصادقة عبر اسم المستخدم وكلمة المرور
4. يجب أن يطبق النظام آليات للتحكم بصلاحيات الوصول بناءً على أدوار المستخدمين. (RBAC)
5. يجب أن يسمح النظام للصيادلة بإضافة سجلات أدوية جديدة تشمل الاسم، الشركة المصنعة، السعر، التصنيف، العيار، الوصف، التركيب، الشكل الدوائي، الحجم (عدد الكبسولات، حجم عبوة الشراب..)، تاريخ التصنيع وتاريخ انتهاء الصلاحية
6. يجب أن يسمح النظام بالتعديل على خصائص الأدوية الموجودة (مثل الاسم، الشركة المصنعة، السعر، التصنيف، الجرعة، الوصف).
7. يجب أن يسمح النظام بإضافة معلومة عن المستوى الأدنى المسموح لكل دواء.
8. يجب أن يسمح النظام بحذف سجلات الأدوية إن لم يتم تنفيذ عمليات عليها.
9. يجب أن يسمح النظام بتسجيل شحنات الأدوية الواردة بما في ذلك معلومات عن الأدوية في الشحنة وكمياتها والمورد.
10. يجب أن يقوم النظام بتحديث كميات المخزون تلقائيًا بناءً على الشحنات الجديدة.
11. يجب أن يقوم النظام بتحديث المخزون عندما يتم صرف الأدوية.
12. يجب أن يمنع النظام صرف الأدوية إذا كانت غير متوفرة في المخزون.
13. يجب أن يجب أن يقوم النظام بتوليد تقارير مالية تلخص المبيعات ضمن مدة زمنية محدّدة.
14. يقوم النظام بتوليد تقارير إدارية للمخزون في الصيدلية.

4.1المتطلبات غير الوظيفية

1- يجب أن يكون النظام آمناً، حيث يسمح فقط للمستخدمين المسجّلين باستخدامه.

2- يجب أن يكون التطبيق متاحاً بشكل دائم

3- يجب أن يكون زمن الاستجابة سريعاً، لا يتجاوز ال 10 ثوان.

4- يجب على النظام أن يكون مصمماً بطريقة قابلة للتحديث والتطوير المستمر، بالإضافة إلى سهولة الصيانة في حال ظهور خطأ أو خلل في الأداء

5- يجب أن يوفر النظام واجهات مرنة، سهلة الفهم الاستخدام وجيدة المظهر.

الفصل الثاني

الدراسة النظرية

يوضّح هذا الفصل مجموعة من المفاهيم النظرية المستخدمة ضمن العمل المقدم.

الفصل الثالث

الدراسة المرجعية

يعرض هذا الفصل الأبحاث والبيانات المرتبطة بالعمل المقدم.

الفصل الرابع

الدراسة التحليلية

يعرض هذا الفصل عمليّة تحليل النظام ودراسة متطلباته.

الفصل السادس

تصميم النظام

يعرض هذا الفصل القرارات التصميمية التي بني من خلالها النظام.

1.5فصل الواجهة الأمامية عن الخلفية (Decoupled Frontend/Backend)

تم اعتماد بنية برمجية حديثة ومنظمة لبناء النظام، حيث يتكون من تطبيقين منفصلين يعملان معاً بشكل متكامل: تطبيق خلفي (Backend) مسؤول عن منطق العمل وإدارة البيانات، وتطبيق أمامي (Frontend) مسؤول عن عرض واجهات المستخدم. تقدم هذه البنية مجموعة من الفوائد الأساسية التي تضمن جودة واستدامة المشروع

- **فصل الاهتمامات (Separation of Concerns)**

إن الفصل الواضح بين الواجهة الخلفية والواجهة الأمامية، بالإضافة إلى الفصل الداخلي لمكونات الواجهة الخلفية، يجعل كل جزء من النظام مسؤولاً عن مهمة محددة. هذا يقلل من التعقيد ويجعل تطوير وفهم كل جزء أكثر بساطة.

- **قابلية الصيانة والتوسع (Maintainability & Scalability)** تسمح البنية المتبعة بإصدار نسخة جديدة من الواجهة الأمامية أو الخلفية بشكل مستقل، سواء لإصلاح خطأ أو إضافة ميزة، دون الحاجة لإعادة بناء النظام بأكمله. على سبيل المثال، يمكن تحديث واجهات المستخدم بالكامل دون المساس بمنطق العمل في الواجهة الخلفية. كما يمكن توسيع كل جزء على حدة لمواجهة أي زيادة في الطلب مستقبلاً

- **التنوع التقني** :يمكّننا هذا الفصل من اختيار التقنية الأنسب لكل جزء. تم استخدام ASP.NET Core للواجهة الخلفية لفعاليتها في بناء APIs آمنة وعالية الأداء، وقدرتها على التكامل مع نظام الهوية (Identity) لإدارة المستخدمين والصلاحيات. بينما تم استخدام Reactمع TypeScriptللواجهة الأمامية لمرونتها بناء واجهات مستخدم تفاعلية وحديثة وقابليتها في إعادة الاستخدام من خلال المكونات (Components).

- **سهولة الاختبار** :يمكن اختبار منطق العمل وقواعد البيانات في الواجهة الخلفية بشكل مستقل تماماً عن واجهات المستخدم، مما يسرّع من دورة التطوير ويضمن جودة الكود.

2.5المعمارية العامة للتطبيق (Application Architecture)

من المهم قبل البدء بتصميم تطبيق معين معرفة البنية المعمارية التي سيتّبعها النظام. يعد استخدام بنية معمارية مناسبة للتطبيق ونطاق العمل الذي يعكسه عنصراً حاسماً في نجاع المشروع. حيث من خلال تبني معمارية واضحة ومتينة، يمكن للتطبيق تلبية احتياجات المستخدمين بكفاءة وفاعلية وضمان تقديم تجربه مستخدم سلسة ومتكاملة. بالإضافة إلى تحسين قابلية التوسع والصيانة.

سنتحدث في هذا القسم عن المعمارية العامة التي تم اختيارها لتصميم التطبيق والتقنيات المستخدمة في الواجهة الأمامية والخلفية

1.2.5معمارية الواجهة الخلفية: البنية النظيفة(Clean Architecture)

لضمان بناء واجهة خلفية متينة وقابلة للصيانة، تم اعتماد **البنية المعمارية النظيفة** .تقوم هذه المعمارية على تنظيم الكود في طبقات متحدة المركز، مع فرض **قاعدة الاعتمادية** (**The Dependency Rule**) التي تنص على أن جميع الاعتماديات يجب أن تشير نحو الداخل فقط، مما يجعل جوهر النظام (منطق العمل) مستقلاً عن التفاصيل التقنية.

تم تقسيم الواجهة الخلفية إلى الطبقات التالية:

1.1.2.5طبقة النطاق (Domain Layer):

هي قلب النظام وتحتوي على كيانات العمل الأساسية (Entities) التي تمثل مفاهيم الصيدلية مثلMedication ، InsuredPerson، و .Prescriptionكما تحتوي على قواعد العمل الجوهرية التي لا تتغير. هذه الطبقة هي الأكثر استقراراً ولا تعتمد على أي طبقة أخرى، مما يضمن استقلاليتها الكاملة.

2.1.2.5طبقة التطبيق (Application Layer)

تحيط هذه الطبقة بطبقة المجال وتحتوي على منطق العمل الخاص بالتطبيق .(Use Cases) هي التي تنسق التفاعل بين كيانات المجال لتنفيذ المهام التي يجب على التطبيق القيام بها، مثل الخدمة التي تنفذ عملية صرف وصفة طبية.

هذه الطبقة تعتمد على طبقة النطاق (Domain Layer)فقط.

في بنية النظام المتبعة، لا يتم تبادل كيانات المجال (Domain Entities) مباشرة مع الطبقات الخارجية، بل يتم الاعتماد على نمط **كائنات نقل البيانات** .(**Data Transfer Objects – DTOs**) تعمل هذه الكائنات كوسيط أو "عقد بيانات (Data Contract) "بين طبقة التطبيق وطبقة العرض .(API) هذا الفصل المتعمد بين نماذج المجال الداخلية والنماذج الخارجية هو قرار تصميمي أساسي يهدف إلى تحقيق عدة أهداف معمارية هامة. الهدف الأساسي من استخدام DTOs هو إخفاء التعقيد الداخلي لطبقة المجال. أيضاً تعمل الـ DTOs كخط دفاع أول ضد البيانات غير الصالحة. من خلال استخدام سمات التحقق من الصحة (Data Annotations) مباشرة على خصائص الـDTO ، يتم فرض قواعد العمل الأساسية قبل أن تصل البيانات إلى الخدمات في طبقة التطبيق.

3.1.2.5طبقة البنية التحتية:(**Infrastructure Layer**)

هي الطبقة الخارجية التي تحتوي على التنفيذ الفعلي للتفاصيل التقنية. تشمل هذه الطبقة DbContext للتواصل مع قاعدة البيانات باستخدامEntity Framework Core ، وتنفيذ نمط المستودع(Repository Pattern) ، وأي خدمات أخرى تتعامل مع العالم الخارجي.

4.1.2.5طبقة العرض:(**Presentation Layer**)

هي الطبقة الخارجية النهائية التي تمثل نقطة الدخول للنظام. في مشروعنا، هي عبارة عن مشروع ASP.NET Core Web APIالذي يستقبل طلبات HTTP من الواجهة الأمامية ويوجهها إلى طبقة التطبيق

.

2.2.5معمارية الواجهة الأمامية: تطبيق الصفحة الواحدة(**SPA**)

تم بناء الواجهة الأمامية كتطبيق صفحة واحدة باستخدام مكتبة React و TypeScriptتعتمد هذه البنية على **المكونات**(**Component–Based Architecture**) ، حيث يتم تقسيم واجهة المستخدم إلى أجزاء صغيرة ومستقلة وقابلة لإعادة الاستخدام.

1.2.2.5طبقة العرض والمكونات:(UI/Components)

تتألف من مجموعة من المكونات التي تشكل واجهة المستخدم، مثل LoginPage وMainLayout ، بالإضافة إلى مكونات واجهة المستخدم الأساسية من مكتبة.shadcn/ui

2.2.2.5طبقة إدارة الحالة(State Management)

يتم استخدام React Context API لإدارة الحالة العامة للتطبيق، وتحديداً حالة المصادقة .(Authentication) يقوم AuthContext بتوفير معلومات المستخدم الذي قام بتسجيل الدخول إلى جميع المكونات دون الحاجة لتمريرها بشكل يدوي.

3.2.2.5طبقة الخدمات:(Services)

لعزل منطق استدعاء الواجهة الخلفية عن المكونات، تم إنشاء طبقة خدمات مثل .authService.ts هذه الطبقة مسؤولة عن إجراء طلبات HTTP باستخدام axios ومعالجة الاستجابات.

3.2.5الربط بين الواجهتين

يتم الاتصال بين الواجهة الأمامية والخلفية عبر بروتوكولHTTPS ، حيث تقوم الواجهة الخلفية بعرض واجهة برمجة تطبيقات (RESTful API) تستقبل الطلبات وتعيد البيانات بصيغة .JSON تم تكوين سياسة **CORS** (**Cross-Origin Resource Sharing**)في الواجهة الخلفية للسماح للواجهة الأمامية (التي تعمل على نطاق مختلف) بالوصول إلى هذه الموارد بشكل آمن.

3.5استراتيجية الرماز أولاً (Code-First)

كأحد القرارات التصميمية في الواجهة الخلفية، تم اعتماد منهجية الرماز أولاً (Code-First) لتطوير قاعدة بيانات النظام، وذلك بالاعتماد على الكامل على إطار العمل (Entity Framework Core). تضع هذه المنهجية الرماز البرمجي في مقدمة عملية التصميم، حيث يتم تعريف نماذج وهياكل البيانات على شكل صفوف، هذه الصفوف تمثل كيانات العمل مثل (Medication) و (Patient).

بعد تعريف هذه النماذج، يتم استخدام أداة (Migrations) المدمجة في (Entity Framework)التي تقوم بتحليل الصفوف وتوليد الشيفرة اللازمة (SQL Scripts) لأنشاء مخطط قاعدة البيانات (Schema) بالكامل أو تعديله ليتطابق مع أي تعديلات تطرأ على الكود. تم اختيار هذه الطريقة لما توفره من مزايا جوهرية؛ فهي تسرّع من وتيرة التطوير بشكل ملحوظ، وتضمن التوافق الدائم بين الكود وقاعدة البيانات، مما يمنع حدوث أخطاء ناتجة عن عدم تطابق الهياكل. كما أنها تسهّل إدارة التغييرات على بنية قاعدة البيانات مع تطور متطلبات المشروع، حيث يتم حفظ كل تغيير في ملف (Migrations)يمكن تتبعه والعودة إليه ضمن نظام التحكم بالإصدارات(Version Control)

4.5الأنماط التصميمية المستخدمة

لتحقيق هذه البنية المعمارية بكفاءة، تم الاعتماد على مجموعة من الأنماط التصميمية التي تقدم حلولاً مجربة للمشاكل المتكررة في تصميم البرمجيات. هذه الأنماط ليست مجرد تفاصيل تنفيذية، بل هي جزء لا يتجزأ من الاستراتيجية التصميمية للنظام.

1.4.5نمط المستودع(Repository Pattern)

تم استخدام هذا النمط لعزل الخدمات في طبقة التطبيق عن .Entity Framework Coreبدلاً من أن تتعامل الخدمات مباشرة مع DbContext، فإنها تتعامل مع واجهة عامة مثل IRepository هذا القرار التصميمي يوفر فائدتين رئيسيتين: أولاً، **فصل الاهتمامات**، حيث يركز منطق العمل على "ماذا" يريد أن يفعل بالبيانات، بينما يركز المستودع على "كيفية" تنفيذ ذلك. ثانياً، **تحسين قابلية الاختبار**، حيث يمكننا بسهولة اختبار الخدمات عن طريق تزويدها بنسخة وهمية من المستودع (Mock Repository) تعمل في الذاكرة، مما يلغي الحاجة إلى الاتصال بقاعدة بيانات حقيقية أثناء الاختبار.

2.4.5نمط وحدة العمل(Unit of Work Pattern)

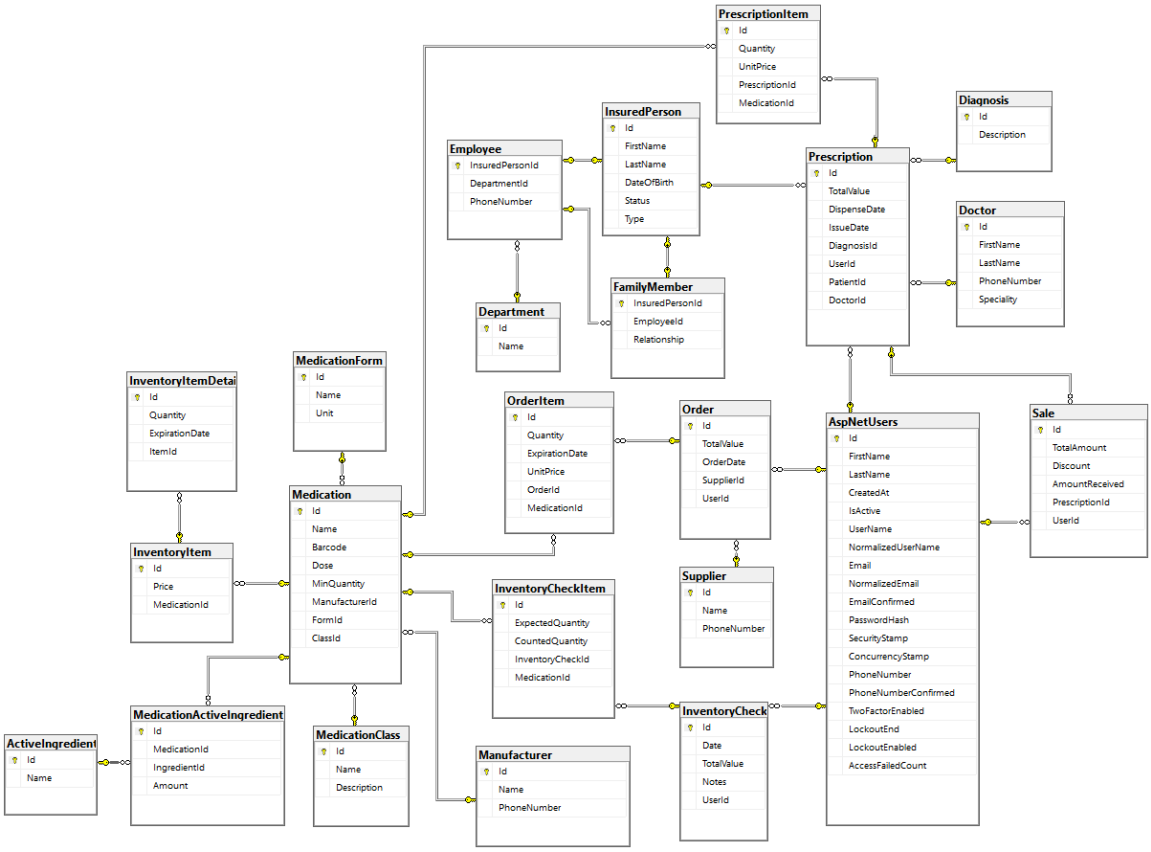
يعتبر هذا النمط حيوياً لعمليات مثل صرف وصفة طبية، والتي تتطلب تعديل عدة جداول في قاعدة البيانات. يقوم UnitOfWorkبتجميع كل هذه التغييرات في معاملة واحدة. عند استدعاء SaveChangesAsync()، يتم تنفيذ جميع العمليات ضمن معاملة قاعدة بيانات واحدة. إذا حدث أي خطأ في أي خطوة (مثل عدم كفاية المخزون)، يتم التراجع عن جميع التغييرات السابقة، مما يضمن **تكامل البيانات** (**Data Integrity**) ويمنع ترك قاعدة البيانات في حالة غير متسقة.

3.4.5نمط حقن الاعتماديات(Dependency Injection – DI)

يُستخدم هذا النمط بشكل أساسي في ASP.NET Core لإدارة دورة حياة الخدمات والمستودعات. هذا يقلل من الترابط بين المكونات(Decoupling) ، حيث يعتمد المتحكم على الواجهة وليس على تنفيذها الفعلي هذا يجعل النظام أكثر مرونة وقابلية للصيانة، حيث يمكن استبدال تنفيذ خدمة معينة بآخر في مكان واحد فقط –البرنامج الأساسي– دون الحاجة لتعديل أي متحكم يستخدمها.

5.5بنية قاعدة البيانات (Database Design)

تم إنشاء مخطط قاعدة البيانات العلائقية في SQL Server ويتألف من الجداول الظاهرة في الشكل التالي:



الشكل 1: تصميم قاعدة البيانات للتطبيق

1. **AspNetUsers**(**المستخدمون**): يعبر هذا الجدول عن مستخدمي النظام من مديرين وصيادلة. نهتم فيه بتخزين معلومات تسجيل الدخول بشكل آمن، بالإضافة إلى الأسماء الشخصية لتمييزهم.
2. **AspNetRoles**(**الأدوار**): يعبر عن الأدوار المتاحة في النظام. نهتم بتخزين اسم الدور (مثل"Admin" أو "Pharmacist") لربطه بالصلاحيات المحددة.

3. **InsuredPerson** (**الأشخاص المؤمن عليهم**): يعبر عن السجل الأساسي لكل شخص يستفيد من خدمات الصيدلية. نهتم بتخزين معلوماته الشخصية، وحالته في النظام.

4. **Employee**(**الموظفون**): جدول يمثل دور الموظف، وهو يرث من جدول InsuredPerson ويضيف معلومات وظيفية خاصة به، كما يحتوي على مفتاح أجنبي لجدول الأقسام بهدف معرفة القسم الذي يعمل به الموظف.

5. **FamilyMember** (**أفراد العائلة**): جدول يرث من InsuredPerson ويخزن معلومات حول أقارب الموظف، حيث يحتوي على مفتاح أجنبي لجدول الموظفين لتحديد الموظف المسؤول عنه، بالإضافة إلى تحديد نوع صلة القرابة.

6. **Medication**(**الأدوية**): يعبر عن كتالوج الأدوية. نهتم بتخزين المعلومات التعريفية للدواء كالاسم والجرعة والباركود، بالإضافة إلى ربطه بالشركة المصنعة وتصنيفه وشكله الصيدلاني.

7. **InventoryItem** (**عناصر المخزون**): يعبر عن وجود دواء معين في المخزون. نهتم بربطه بالدواء وتخزين سعر بيعه.

8. **InventoryItemDetail** (**تفاصيل المخزون**): يمثل دفعة (batch) محددة من دواء في المخزون. نهتم بتخزين الكمية المتوفرة وتاريخ انتهاء الصلاحية لهذه الدفعة.

9. **Diagnosis**(**التشخيصات**): يعبر عن التشخيصات الطبية التي يمكن ربطها بالوصفات. نهتم بتخزين وصف التشخيص.

10. **Prescription** (**الوصفات الطبية**): يعبر عن عملية صرف متكاملة. نهتم بتخزين معلومات المريض والطبيب والصيدلي الذي قام بالصرف، بالإضافة إلى تاريخ وقيمة الصرف الإجمالية.

11. **Sale** (**المبيعات**): يمثل الجانب المالي لكل وصفة. نهتم بتخزين القيمة الإجمالية، وقيمة التغطية التي قدمتها المنظمة، والمبلغ المستلم من المريض.

12. **PrescriptionItem** (**عناصر الوصفة الطبية**): هو جدول كسر علاقة many-to-many بين الوصفات والأدوية. نهتم فيه بتخزين الكمية المصروفة وسعر الوحدة لكل دواء في الوصفة.

13. **Department**(**الأقسام**): يعبر هذا الجدول عن الأقسام المختلفة داخل المنظمة. نهتم بتخزين اسم القسم لربط الموظفين به.

14. **Doctor** (**الأطباء**): يعبر عن الأطباء الذين يكتبون الوصفات. نهتم بتخزين أسمائهم وتخصصاتهم.

15. **Manufacturer** (**الشركات المصنعة**): يعبر عن الشركات المنتجة للأدوية. نهتم بتخزين اسم الشركة.

16. **MedicationClass** (**تصنيفات الأدوية**):يعبر عن التصنيفات العلاجية. نهتم بتخزين اسم التصنيف ووصفه.

17. **MedicationForm** (**أشكال الأدوية**):يعبر عن الشكل الصيدلاني للدواء. نهتم بتخزين اسم الشكل ووحدة القياس.

18. **ActiveIngredient** (**المكونات الفعالة**): يعبر عن المواد الكيميائية الفعالة في الأدوية. نهتم بتخزين اسم المكون.

19. **MedicationActiveIngredient**: هو جدول كسر علاقة many-to-many بين الأدوية والمكونات الفعالة. نهتم فيه بتحديد كمية كل مكون في الدواء.

20. **Supplier** (**الموردون**): يعبر عن الشركات التي تورد الأدوية للصيدلية. نهتم بتخزين اسم المورد.

21. **Order** (**طلبات الشراء**): يعبر عن طلبات الشراء من الموردين. نهتم بتخزين معلومات المورد وتاريخ الطلب.

22. **OrderItem** (**عناصر طلب الشراء**): هو جدول كسر علاقة many-to-manyبين طلبات الشراء والأدوية. نهتم فيه بتخزين الكمية المطلوبة وسعر الوحدة وتاريخ انتهاء الصلاحية لكل دواء في الطلب.

23. **InventoryCheck** (**جرد المخزون**): يعبر عن عمليات الجرد الدورية. نهتم بتخزين تاريخ الجرد والملاحظات المتعلقة به.

24. **InventoryCheckItem** (**عناصر جرد المخزون**): هو جدول كسر علاقة بين عمليات الجرد والأدوية. نهتم فيه بتخزين الكمية المتوقعة في النظام والكمية التي تم عدها فعلياً لكل دواء.

.

الفصل السابع

التنفيذ

يشرح هذا الفصل كيف تم تنجيز النظام.

1.6تنفيذ الواجهة الخلفية (Backend)

تم تنجيز الواجهة الخلفية باستخدام إطار العمل ASP.NET Core، مع استخدام البنية النظيفة (Clean Architecture) التي تم استعراضها في الفصول السابقة. حيث تتألف الواجهة الخلفية من أربعة مشاريع (Projects) تمثل طبقات البنية النظيفة، موجودة داخل الحل البرمجي (sln)

1.1.6 طبقة النطاق (Domain Layer)

تعد هذه الطبقة هي نواة التطبيق لأنها تحتوي على منطق العمل الجوهري، هذه الطبقة تم تنجزيها في مشروع Domain المستقل الذي لا يعتمد على أي مشروع آخر، مما يحقق المبدأ الأساسي في البنية النظيفة.

تحتوي طبقة النطاق على عدة مجلدات فرعية:

1.1.1.6الكيانات (Entities):

تمثل كيانات العمل الأساسية التي تعكس مفاهيم الصيدلية. تم تصميمها كصفوف C#بسيطة مع التركيز على تمثيل البيانات والعلاقات الأساسية بينها. سنتطرق إلى بعض الصفوف الأساسية منها التي قد تحتاج إلى تفسير:

- **كيان Insured Person**: يعتبر الكيان المحوري الذي يمثل المستفيدين من خدمة التأمين التي تقدمها الصيدلية. تم تصميمه ليحتوي على المعلومات الشخصية مثل الاسم وتاريخ الميلاد. يحتوي أيضاً على الواصفة Typeمن نوع boolالتي تستخدم للتمييز بين الموظف (false) وفرد عائلة (true). هذا القرار التنفيذي يسمح للطبقات العليا بتطبيق قواعد التغطية المالية المختلفة بحسب نوع الشخص المؤمن.

- **كيانات FamilyMember,Employee** تمثل تخصصات للكيان InsuredPerson، ترتبط كل منها مع الصف الأخير بعلاقة "واحد إلى واحد" (One-to-One) عبر مفتاح أساسي (Primary Key) هو نفسه أجنبي (Foreign Key)

- **كيان InventoryItemDetail**: هو التنفيذ العملي لمبدأ إدارة الدفعات (Batches) بدلاً من تخزين كمية إجمالية للدواء، يمثل كل سجل في هذا الجدول دفعة محددة لها كمية (Quantity) وتاريخ انتهاء الصلاحية (ExpirationDate) خاص بها.

2.1.1.6الواجهات (Interfaces)

تحتوي هذه الطبقة أيضاً على مجموعة من الواجهات (Interfaces) التي تعتبر مكون أساسي لتطبيق مبدأ انعكاس التبعية (Dependency Inversion Principle) الذي يعتبر من أهم مبادئ البنية النظيفة. تم تعريف الواجهات التالية ضمن طبقة النطاق:

- **الواجهة العامة للمستودع (IRepository<T,Key>): هذه** الواجهة تمثل التجريد الأساسي لعمليات الوصول إلى البيانات لأي كيان في النظام. تحتوي هذه الواجهة على دوال عدة. دالة لإضافة كيان جديد، دالة للحذف، دالة للتعديل ودالة لحفظ التغييرات في قاعدة البيانات ودالة لجلب جميع السجلات ودالة لجلب سجل واحد بناءً على المفتاح الأساسي.

  الجدير بالذكر، أنه تم تطوير الدوال المسؤولة عن جلب السجلات لكي تتيح خيار التحميل السريع (Eager loading) لتضمين الكيانات المرتبطة.

- **الواجهات المتخصصة للمستودعات:** في بعض الحالات، تحتاج بعض الكيانات إلى استعلامات خاصة بها لا يمكن تعميمها في الواجهة العامة للمستودع. لذلك تم إنشاء واجهات متخصصة ترث من الواجهة العامة وتضيف دوالًا خاصة بها. على سبيل المثال، خدمة الأدوية تحتاج إلى جلب دواء معين مع جميع تفاصيله المرتبطة (الشركة المصنعة، الشكل الدوائي، المكونات الفعالة إلخ) على مرحلتين، لذلك تم تعريف دالة خاصة لهذه الحالة.

- **واجهة وحدة العمل (IUnitOfWork)** :تعتبر هذه الواجهة هي المنسق لجميع عمليات قاعدة البيانات. الهدف منها هو ضمان أن العمليات التجارية التي تتضمن تعديل أكثر من جدول (مثل صرف وصفة طبية) تتم كوحدة واحدة متكاملة (ذرية) (Atomic Transaction).

تحتوي على دالة لحفظ جميع التغييرات التي تم تتبعها في سياق قاعدة البيانات (DbContext) ضمن معاملة واحدة.

2.1.6طبقة التطبيق (Application Layer)

هذه الطبقة هي التي تحتوي على منطق العمل الخاص بالتطبيق وتنسق التفاعل بين كيانات المجال لتنفيذ المهام المطلوبة. تم تنفيذها في مشروع التطبيق (Application) الذي يعتمد فقط على مشروع النطاق (Domain)، لأن منطق التطبيق يعتمد على كيانات وقواعد العمل الأساسية، ولكنه يبقى معزولاً تماماً عن تفاصيل الواجهة الخارجية (Presentation) والبنية التحتية(Infrastructure) .

تتكون هذه الطبقة من عدة مكونات رئيسية تعمل معاً لتنفيذ وظائف النظام:

1.2.1.6كائنات نقل البيانات :(DTOs – Data Transfer Objects)

تم تصميم الـ DTOs لتكون مجرد حاويات بسيطة للبيانات تحتوي فقط على الخصائص التي تحتاجها الواجهة الأمامية. على سبيل المثال، كيان Saleيرتبط بكيان Userالذي يحتوي على معلومات حساسة. عند عرض بيانات المبيعات، يتم استخدام GetSaleDTOالذي يقوم بتجريد هذه العلاقة المعقدة ويعرض فقط اسم الصيدلي (PharmacistName).هذا يضمن أن تفاصيل تنفيذ طبقة المجال تبقى مغلفة ومحمية.

2.2.1.6 التحويل التلقائي (Auto Mapper)

مع وجود عدد كبير من الDTOs ، تصبح عملية تحويل البيانات بين الكيانات و DTOsعملية متكررة ومصدرًا محتملاً للأخطاء. لحل هذه المشكلة، تم استخدام مكتبة **AutoMapper**لأتمتة هذه العملية. تم تنظيم قواعد التحويل في ملفات توصيف (Mapping Profiles) متخصصة لكل مجال وظيفي، مثل PrescriptionProfile و InventoryProfile.

على سبيل المثال في ملف PrescriptionProfile.cs، تم تعريف قاعدة تحويل مخصصة لملء خاصية PatientNameفي GetPrescriptionDTOعن طريق دمج خاصيتي FirstNameو LastNameمن

كيان Patientالمرتبط. هذا يتم باستخدام دالة ForMemberالتي تسمح بتحديد سلوك مخصص لكل خاصية في الكائن الهدف.

3.2.1.6الخدمات(Services)

تم تغليف منطق العمل في خدمات متخصصة، كل منها مسؤول عن تنفيذ مجموعة من حالات الاستخدام (Use Cases). نذكر منها:

- **AuthService.cs**: هي المسؤولة عن عمليات المصادقة والتفويض. عند تسجيل دخول المستخدم، تقوم بالتحقق من بياناته باستخدام UserManagerمن ASP.NET Core Identity. في حال نجاح العملية، تقوم بإنشاء **JSON Web Token (JWT)**يحتوي على "ادعاءات (Claims) "مثل معرّف المستخدم ودوره، والذي يتم إرساله إلى الواجهة الأمامية لاستخدامه في الطلبات اللاحقة.
- **PrescriptionService.cs**: تحتوي على منطق العمل الأكثر تعقيدًا في النظام. عند استدعاء دالة ProcessPrescriptionAsync، تقوم بتنسيق عملية صرف الوصفة الطبية التي تشمل التحقق من المخزون، تطبيق منطقFIFO ، حساب التكاليف والتغطية المالية، وتحديث جميع السجلات ذات الصلة في قاعدة البيانات ضمن معاملة واحدة من خلال UnitOfWork.

3.1.6طبقة البنية التحتية(Infrastructure Layer)

تتولى هذه الطبقة مسؤولية التعامل مع كل ما هو خارج نطاق التطبيق، وأهمها قاعدة البيانات.

- **PharmacyDbContext.cs**: يمثل سياق قاعدة البيانات (Database Context) الخاص بإطار العمل Entity Framework Core. في دالة OnModelCreating، تم تكوين العلاقات بين الجداول بشكل صريح باستخدام **Fluent API**، وتحديد سلوك الحذف (OnDelete) لضمان تكامل البيانات المرجعية.(Referential Integrity)
- **Repository<T, TKey>**: هو التنفيذ العام لواجهة IRepository. يستخدم DbSetمن DbContextلتنفيذ عمليات قاعدة البيانات القياسية. تم استخدام تعابير LINQو IQueryable

لبناء استعلامات فعالة، مع الاستفادة من ميزة **التحميل السريع** (**Eager Loading**) عبر دالة Include لجلب الكيانات المرتبطة في استعلام واحد، مما يقلل من مشكلة N+1.

- **UnitOfWork.cs**: هو التنفيذ الفعلي لواجهة IUnitOfWork. يقوم بإنشاء نسخ من جميع المستودعات ويمرر لها نفس نسخة DbContext، ويحتوي على دالة SaveChangesAsync التي تقوم بتنفيذ _context.SaveChangesAsync()، مما يضمن أن جميع التغييرات يتم حفظها كوحدة ذرية واحدة.

4.1.6 طبقة العرض (**Presentation Layer**)

تمثلها واجهة برمجة التطبيقات (Web API) في مشروع PharmacyManagmentApp.

1.4.1.6 Program.cs: هو نقطة الدخول للتطبيق حيث يتم تكوين وتسجيل جميع الخدمات والتبعيات باستخدام حاوية حقن الاعتماديات المدمجة في ASP.NET Core (builder. Services.AddScoped). كما يتم فيه تكوين وسيطات (Middleware) المصادقة والتفويض الخاصة بـ JWT وسياسة CORS.

2.4.1.6 المتحكمات (Controllers): تم تنظيم نقاط النهاية (Endpoints) في متحكمات متخصصة مثل SalesController و MedicationsController. تم استخدام سمات التوجيه ([Route]) والسمات الخاصة بأفعال HTTP ([HttpGet], [HttpPost], إلخ) لتعريف نقاط النهاية. يتم تأمين الوصول إلى هذه النقاط باستخدام السمة [Authorize]، مع تحديد الأدوار المسموح بها مما يضمن أن المستخدمين المصرح لهم فقط يمكنهم الوصول إلى الوظائف الحساسة.

2.6 تنفيذ الواجهة الأمامية (**Frontend**)

تم بناء الواجهة الأمامية كتطبيق صفحة واحدة (**Single–Page Application – SPA**) باستخدام مكتبة **React** مع لغة **TypeScript**. هذا الخيار يوفر تجربة مستخدم سريعة وسلسة، حيث يتم تحميل الصفحة الرئيسية مرة واحدة فقط، ويتم تحديث المحتوى ديناميكيًا دون الحاجة لإعادة تحميل الصفحة بالكامل عند التنقل. تم الاعتماد على حزمة **Vite** كأداة بناء (Build Tool) للمشروع، لما توفره من سرعة فائقة في تشغيل خادم التطوير والتحديث الفوري (Hot Module Replacement)، مما يسرّع من وتيرة عملية التطوير بشكل ملحوظ.

1.2.6هيكلية المشروع والبنية القائمة على المكونات (Component-Based Architecture)

تم تنظيم الشيفرة المصدرية للواجهة الأمامية في مجلد PharmacyApp-Frontendبطريقة منهجية تتبع مبدأ البنية القائمة على المكونات، حيث يتم تقسيم واجهة المستخدم إلى أجزاء صغيرة، مستقلة، وقابلة لإعادة الاستخدام. هذا النهج لا يسهل فقط إدارة الشيفرة المصدرية، بل يعزز أيضًا من إمكانية إعادة استخدام المكونات في أجزاء مختلفة من التطبيق.

1.1.2.6الصفحات:(src/pages)

يمثل كل ملف في هذا المجلد صفحة كاملة في التطبيق، وتُعرف هذه المكونات بـ "المكونات الحاوية (Container Components)" لأنها مسؤولة عن منطق الصفحة، مثل جلب البيانات وإدارة حالتها. على سبيل المثال، صفحة DispensePage.jsxتحتوي على المنطق اللازم للبحث عن المرضى والأطباء، إضافة الأدوية إلى وصفة، وحساب التكاليف النهائية قبل إرسالها إلى الواجهة الخلفية.

2.1.2.6المكونات القابلة لإعادة الاستخدام:(src/components)

هذا المجلد يحتوي على مكونات عرض (Presentational Components) يمكن استخدامها في أي صفحة.

- **MainLayout.jsx**يُعد هذا المكون هو الهيكل الرئيسي الذي يلتف حول معظم صفحات النظام. وهو مسؤول عن عرض الشريط الجانبي للتنقل، والشريط العلوي الذي يعرض معلومات المستخدم، وتوفير منطقة المحتوى الرئيسية التي يتم فيها عرض الصفحة النشطة باستخدام مكون Outletمن مكتبة react-router-dom.
- **مجلد ui/**: تم استخدام مكتبة **shadcn/ui**لتوفير مجموعة من مكونات واجهة المستخدم الأساسية والمصممة بشكل احترافي مثل Button, Card, Input, وDialog. الميزة الفريدة لهذه المكتبة هي أنها لا تفرض أسلوبًا تصميميًا صارمًا، بل توفر شيفرة مصدرية للمكونات يمكن دمجها مباشرة في المشروع وتخصيصها بالكامل باستخدام **TailwindCSS**، مما يمنح المطور تحكمًا كاملاً في مظهر وسلوك الواجهات.

2.2.6إدارة الحالة(State Management)

تمت إدارة حالة التطبيق على مستويين: الحالة العامة للتطبيق، والحالة المحلية للمكونات.

1.2.2.6الحالة العامة :(Global State)

لإدارة حالة المصادقة التي تحتاجها معظم مكونات التطبيق، تم استخدام واجهة برمجة تطبيقات السياق **Context**) (**API**المدمجة في.React

- **AuthContext.jsx**

يقوم هذا السياق بتغليف التطبيق بأكمله وتوفير معلومات وخدمات المصادقة لأي مكون فرعي. عند تسجيل دخول المستخدم بنجاح، يتم تخزين كائن الاستجابة بالكامل، بما في ذلك **JWT**، في التخزين المحلي للمتصفح .(localStorage)عند إعادة تحميل التطبيق، يقوم AuthContextبالتحقق من وجود هذه البيانات في التخزين المحلي لاستعادة جلسة المستخدم، مما يوفر تجربة مستخدم سلسة. يوفر السياق أيضًا دوال loginو logoutالتي تتفاعل مع خدمة المصادقة .authService

2.2.2.6الحالة المحلية :(Local State)

داخل كل مكون صفحة، تم استخدام خطافات React القياسية مثل useStateو useEffectلإدارة حالتها الداخلية. على سبيل المثال، في صفحة InventoryPage.jsx، يتم استخدام useStateلتخزين قائمة المخزون، حالة التحميل، ومصطلح البحث الذي يدخله المستخدم. يتم استخدام useEffectلجلب البيانات من الواجهة الخلفية عند تحميل المكون لأول مرة.

3.2.6التوجيه وحماية المسارات(Routing and Route Protection)

تم استخدام مكتبة **React Router DOM**لتنفيذ التوجيه من جانب العميل، مما يسمح بالتنقل بين الصفحات دون إعادة تحميل المتصفح.

- **App.tsx**: في هذا الملف، يتم تعريف هيكل التوجيه الكامل للتطبيق. يتم استخدام المكون <Routes> لتغليف جميع المسارات .(<Route>) تم تصميم المسارات لتشمل صفحة تسجيل الدخول العامة، وإعادة توجيه من المسار الجذري (/) إلى لوحة التحكم(/dashboard) ، ومجموعة من المسارات المحمية التي تتطلب مصادقة.

- **PrivateRoute.jsx**: لحماية المسارات التي تتطلب تسجيل الدخول، تم إنشاء هذا المكون المخصص الذي يعمل كحارس للمسارات .(Route Guard) يقوم هذا المكون بالتحقق من حالة المصادقة من AuthContext. إذا لم يكن المستخدم مسجلاً دخوله، يتم استخدام المكون <Navigate> من React Router لإعادة توجيهه إلى صفحة login/، مع حفظ المسار الأصلي الذي كان يحاول الوصول إليه. بعد تسجيل الدخول بنجاح، يتم إعادة توجيه المستخدم مرة أخرى إلى المسار الأصلي الذي كان يقصده.

4.2.6 التواصل مع الواجهة الخلفية (API Communication)

تم عزل جميع عمليات التواصل مع واجهة برمجة التطبيقات (API) في طبقة خدمات متخصصة (src/services) لتحقيق مبدأ فصل الاهتمامات وجعل المكونات أكثر تركيزًا على العرض.

1.4.2.6 طبقة الخدمات :

يحتوي كل ملف خدمة، مثل saleService.ts أو orderService.ts، على مجموعة من الدوال غير المتزامنة (async) التي تتوافق مع نقاط النهاية في الواجهة الخلفية.

2.4.2.6 استخدام axios والمصادقة

تم استخدام مكتبة **axios** لإجراء طلبات HTTP. عند تنفيذ أي دالة خدمة، يتم أولاً استدعاء authService.getCurrentUser() لجلب معلومات المستخدم المخزنة، بما في ذلك JWT. بعد ذلك، يتم تضمين هذا التوكن في رأس Authorization لكل طلب API صادر، باستخدام مخطط Bearer. هذا يضمن أن جميع الطلبات إلى النقاط النهائية المحمية يتم توثيقها بشكل صحيح.

3.4.2.6 إدارة متغيرات البيئة

تم تعريف عنوان URL الأساسي لواجهة برمجة التطبيقات في ملف env.local. هذا يسمح بنشر التطبيق في بيئات مختلفة (تطوير، إنتاج) بسهولة عن طريق تغيير ملف البيئة فقط دون الحاجة لتعديل الشيفرة المصدرية.