الجمهورية العربية السورية
المعهد العالي للعلوم التطبيقية والتكنولوجيا
قسم المعلوميات
العام الدراسي 2024 / 2025

مشروع سنة رابعة

بناء منصة عقارية لقوائم العقارات مع خاصية المزايدات

إعداد الطالب

سلمان عامر محسن

إشراف

ما. محمود إلياس

الملخَّص

استجابةً للحاجة المتزايدة لوجود حلول رقمية موثوقة في سوق العقارات، يتناول هذا المشروع تطوير منصة متكاملة لإدارة وتنظيم المزادات العقارية. يهدف المشروع إلى توفير بيئة شفافة وآمنة تجمع بين البائعين والمشترين، وتضمن سير عمليات المزايدة بكل سهولة وفعالية. يقدم هذا العمل منصة لعرض العقارات والمزايدة عليها وهي نظام إلكتروني شامل تم تصميمه لمعالجة التحديات المرتبطة بالمزادات التقليدية كالتلاعب بالمزادات. تتيح المنصة عرضاً تفصيلياً للعقارات المتاحة، وتوفر أدوات متقدمة لإدارة عمليات المزايدة بشكل حي ومباشر، مما يعزز من عدالة وشفافية المنافسة. كما يشتمل النظام على آلية آمنة للمعاملات المالية ونظام للتواصل الفعال بين جميع الأطراف من خلال الإشعارات والتقييمات.

تم بناء النظام وفق منهجية تضمن القابلية للتطوير والاستدامة، مع التركيز على تقديم تجربة استخدام سلسة وموثوقة. وبذلك، يشكل المشروع حلاً عملياً ومتيناً يهدف إلى رفع مستوى الكفاءة والمصداقية في قطاع المزادات العقارية، ويوفر أساساً قوياً لإدارة جميع مراحل المزاد من عرض العقار وحتى إتمام عملية البيع.

أهداف المشروع

يهدف المشروع إلى تحقيق مجموعة من الأهداف الأساسية التي تضمن إنشاء منصة مزادات عقارية متكاملة وموثوقة، وهي كالتالي:

1. **تطوير نظام مركزي لإدارة المزادات:** بناء منصة إلكترونية متكاملة تعمل كوسيط فعال وموثوق بين بائعي العقارات والمزايدين، مع توفير الأدوات اللازمة لإدارة جميع جوانب المزاد بكفاءة، بدءاً من تسجيل المستخدمين وإنشاء قوائم العقارات، وصولاً إلى إدارة عمليات المزايدة وإتمام الصفقات.
2. **ضمان الشفافية والموثوقية في عمليات المزايدة:** تصميم آلية مزايدة تفاعلية في الزمن الفعلي تضمن العدالة والشفافية لجميع المشاركين. يهدف النظام إلى تعزيز ثقة المستخدمين من خلال توفير سجل دقيق للمزايدات ونظام آمن لمعالجة المعاملات المالية.
3. **تحقيق تجربة استخدام سلسة وفعالة:** إنشاء واجهات مستخدم بديهية وسهلة الاستخدام لجميع أطراف المزاد (المسؤول، البائع، والمزايد)، مما يقلل من التعقيد التقني ويتيح للمستخدمين التركيز على عملية المزايدة نفسها، مع ضمان استجابة سريعة وأداء عالٍ للنظام.
4. **توفير نظام إداري ومراقبة شامل:** إيجاد لوحة تحكم إدارية مركزية تمكّن المسؤولين من الإشراف الكامل على سير عمليات المزاد، وإدارة حسابات المستخدمين، ومراقبة قوائم العقارات، مما يضمن الحفاظ على سلامة النظام والالتزام بالمعايير المحددة.

متطلبات النظام

المتطلَّبات الوظيفية:

1- إدارة حسابات المستخدمين:

يجب أن يسمح النظام للمستخدمين بما يلي:

1-1- إنشاء حسابات جديدة باستخدام البريد الإلكتروني.

1-2- تسجيل الدخول في حال كان المستخدم مسَجلاً بالنظام.

1-3- تسجيل الخروج من حساباتهم.

1-4- تحديث معلوماتهم الشخصية (مثل الاسم، رقم الهاتف، العنوان).

1-5- استعادة كلمات المرور المنسية عبر البريد الإلكتروني ورمز التحقق.

1-6- حذف حساباتهم إذا رغبوا بذلك.

2- عرض العقارات:

يجب أن يسمح النظام للمستخدمين بما يلي:

2-1- البحث عن العقارات باستخدام مُرشحات (مثل الموقع، السعر، عدد الغرف، النوع: شقة/منزل كبير).

2-2- عرض تفاصيل كل عقار بما في ذلك (الصور، وصف شامل للعقار).

2-3- تصفية النتائج حسب حالة العقار (مقبول، انتظار للقبول).

3- المزايدة على العقارات:

3-1- يجب على النظام أن يقوم بإخطار المستخدمين عبر الرسائل عندما يتم تجاوز مزايدتهم.

3-2- يجب على النظام إعلام المستخدمين عند القيام بعملية حجز لمقدار معين من المال في حال كان المستخدم يريد المشاركة بالمزاد.

3-3- يجب على النظام أن يقوم بإغلاق المزايدة تلقائياً بعد انتهاء الوقت المحدد.

3-4- يجب على النظام تحرير الأموال المحجوزة عند انتهاء المزاد لجميع المستخدمين عدا المستخدم الفائز في المزاد.

يجب أن يسمح النظام للمستخدمين بما يلي:

3–5– عرض قائمة بالمزادات الفعالة.

3–6– المشاركة بالمزادات المعروضة.

3–7– عرض أعلى مزايدة حالية لكل عقار في الوقت الفعلي.

3–8– سحب مزايداتهم قبل إغلاق المزايدة.

3–9– عرض سجل المزايدات السابقة الخاصة بهم.

يجب أن يسمح النظام للبائعين بما يلي:

3–10– تقديم عروض على العقارات المعروضة الخاصة بالبائع قبل البدء بالمزاد، مثال (يربح الفائز بالإضافة للعقار شيء معين يحدده البائع في حال تجاوز السعر في المزاد حداً معيناً).

3–11– تحديد وقت بدء وانتهاء المزاد.

3–12– تحديد حد أدنى للزيادة الواحدة في المزايدة.

3–13– المشاركة في المزادات الخاصة بالآخرين.

4– إدارة العقارات:

4–1– يمكن لمدير النظام مراجعة العقارات المضافة والموافقة عليها لضمان الامتثال للمعايير.

4–2– يمكن لمدير النظام حذف عقارات موجودة على المنصة مع إعلام البائع بالسبب.

يجب على النظام أن يسمح للبائعين بما يلي:

4–3– تحرير تفاصيل العقارات المعروضة (مثل تغيير السعر أو الوصف).

4–4– إزالة العقارات من العرض قبل بدء المزاد أو بعد المزاد في حال لم يتم بيعه.

4–5– تجديد عرض العقار في حال لم يتم بيعه.

4–6– إضافة عقارات جديدة مع تفاصيل مثل العنوان، الوصف، الصور، والسعر الابتدائي.

5– الدفع والمعاملات المالية:

5–1– يجب على النظام أن يسمح للفائزين بدفع الفاتورة إلكترونياً.

6- إدارة الإشعارات:

6-1- يجب على النظام أن يقوم بإرسال إخطارات فورية لأحداث مثل بدء مزايدة جديدة، انتهاء المزايدة، فوز المستخدم.

7- التقييمات والمراجعات:

7-1- يجب على النظام أن يسمح للمستخدمين بتقييم البائعين أو العقارات بعد انتهاء المزايدة.

7-2- يجب على النظام أن يقوم بعرض التقييمات للمستخدمين الآخرين لتعزيز الشفافية.

7-3- يمكن لمدير النظام التحقق من المراجعات لمنع المحتوى غير المناسب.

7-4- يجب على النظام أن يسمح بإمكانية الإبلاغ عن تقييمات مزيفة أو مسيئة.

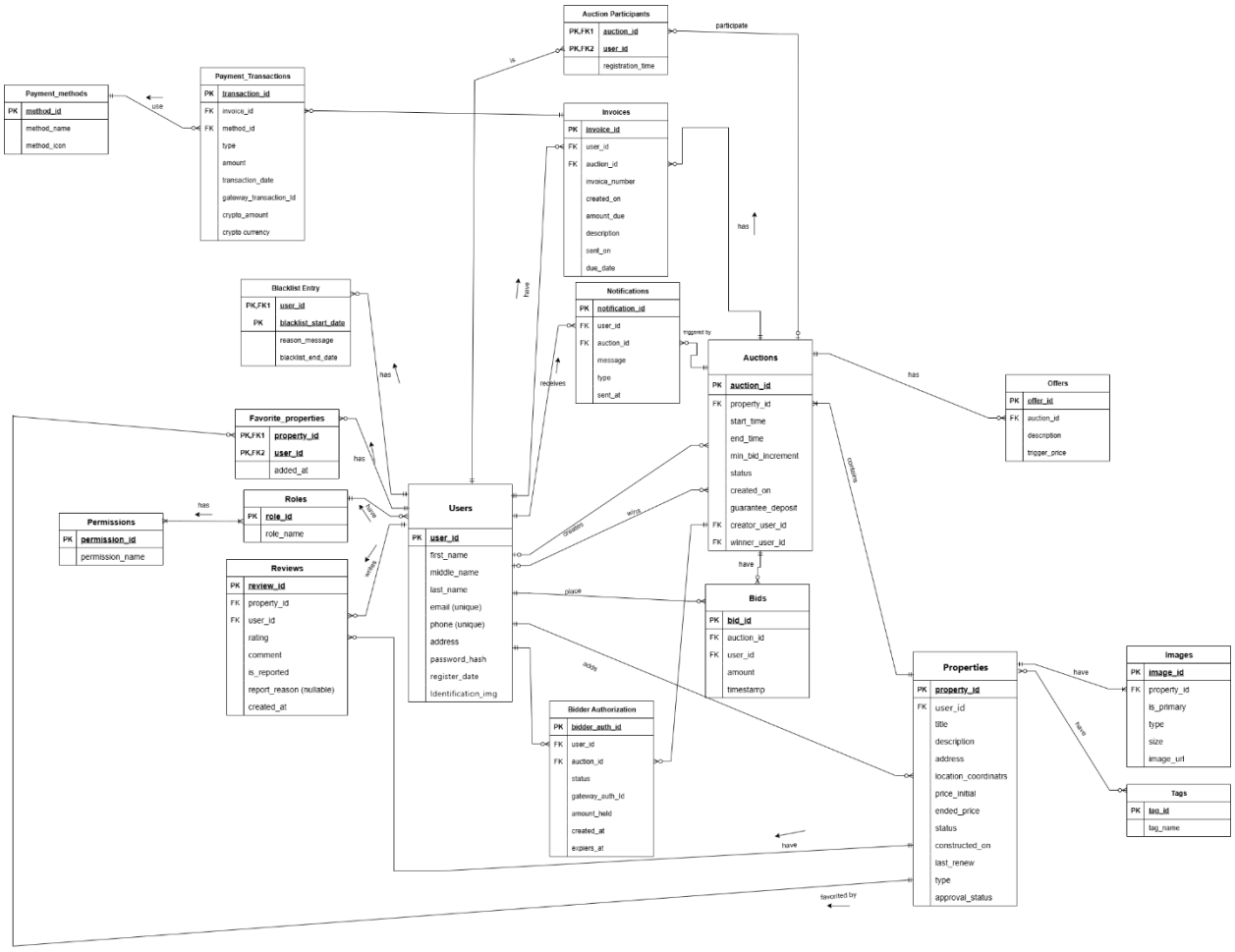
المتطلبات غير الوظيفية:

1. يجب أن يستجيب النظام لطلبات المستخدمين خلال 2-3 ثوانٍ كحد أقصى.
2. يجب أن يتم تحديث بيانات المزايدة في الوقت الفعلي دون تأخير.
3. يجب أن يكون النظام متجاوباً مع جميع قياسات الأجهزة (هواتف ذكية، أجهزة لوحية، حواسيب).
4. يجب أن يدعم النظام أنواع الصور الشائعة (PNG, JPEG, ...).
5. يجب ألا يزيد عدد الصور المرفوعة عن 25 صورة للعقار الواحد.
6. استخدام تقنيات لدعم الوقت الفعلي عند تنفيذ المزاد.
7. يجب على النظام أن يكون آمناً، حيث يسمح فقط للمستخدمين المسجَّلين باستخدامه.
8. جيب أن يوفر النظام واجهات سهلة الاستخدام وجيدة المظهر.

الدراسة التحليلية

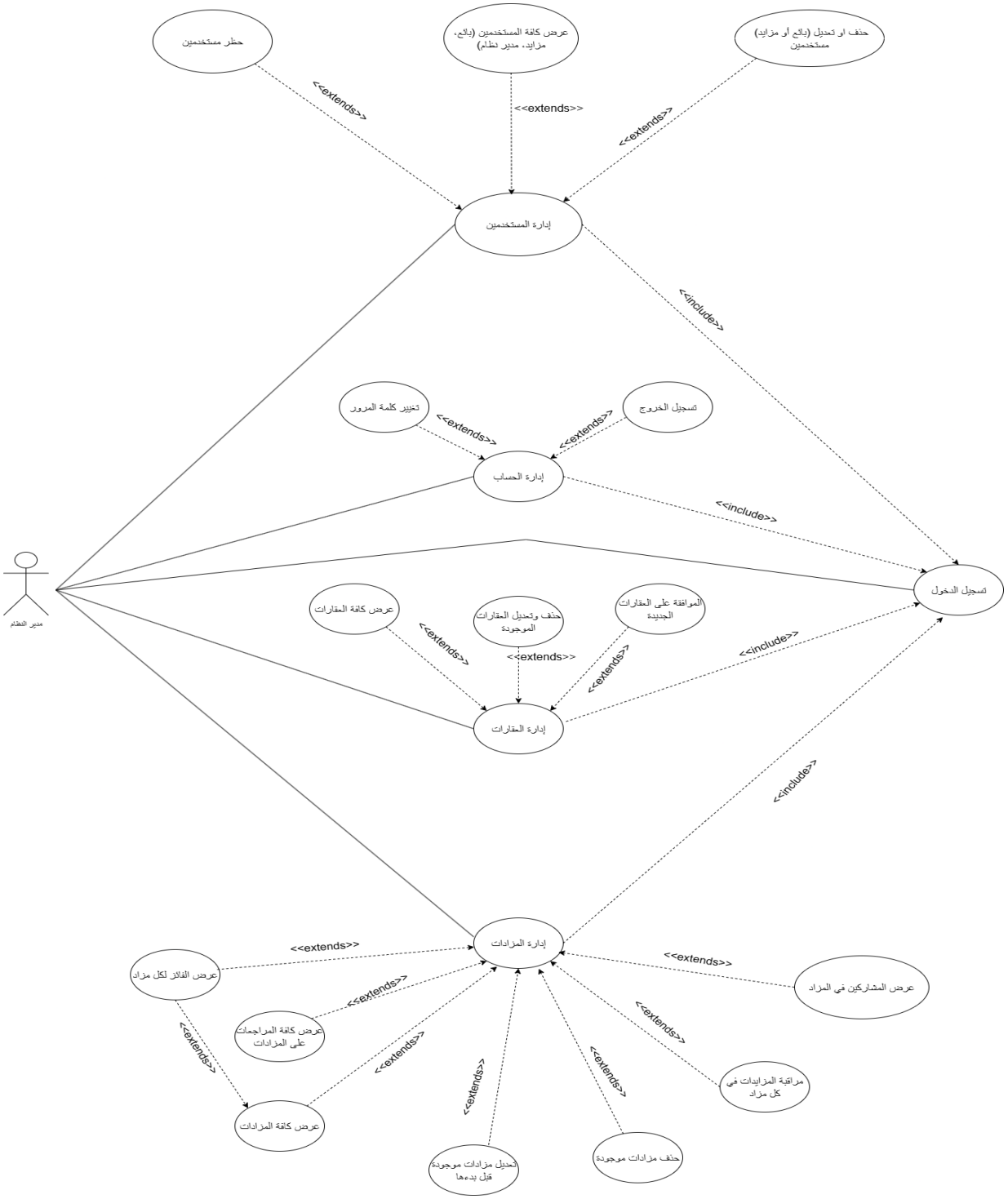
في هذا القسم نستعرض المخططات اللازمة لتصميم النظام

1- مخطط الكيان والعلاقة (ERD):

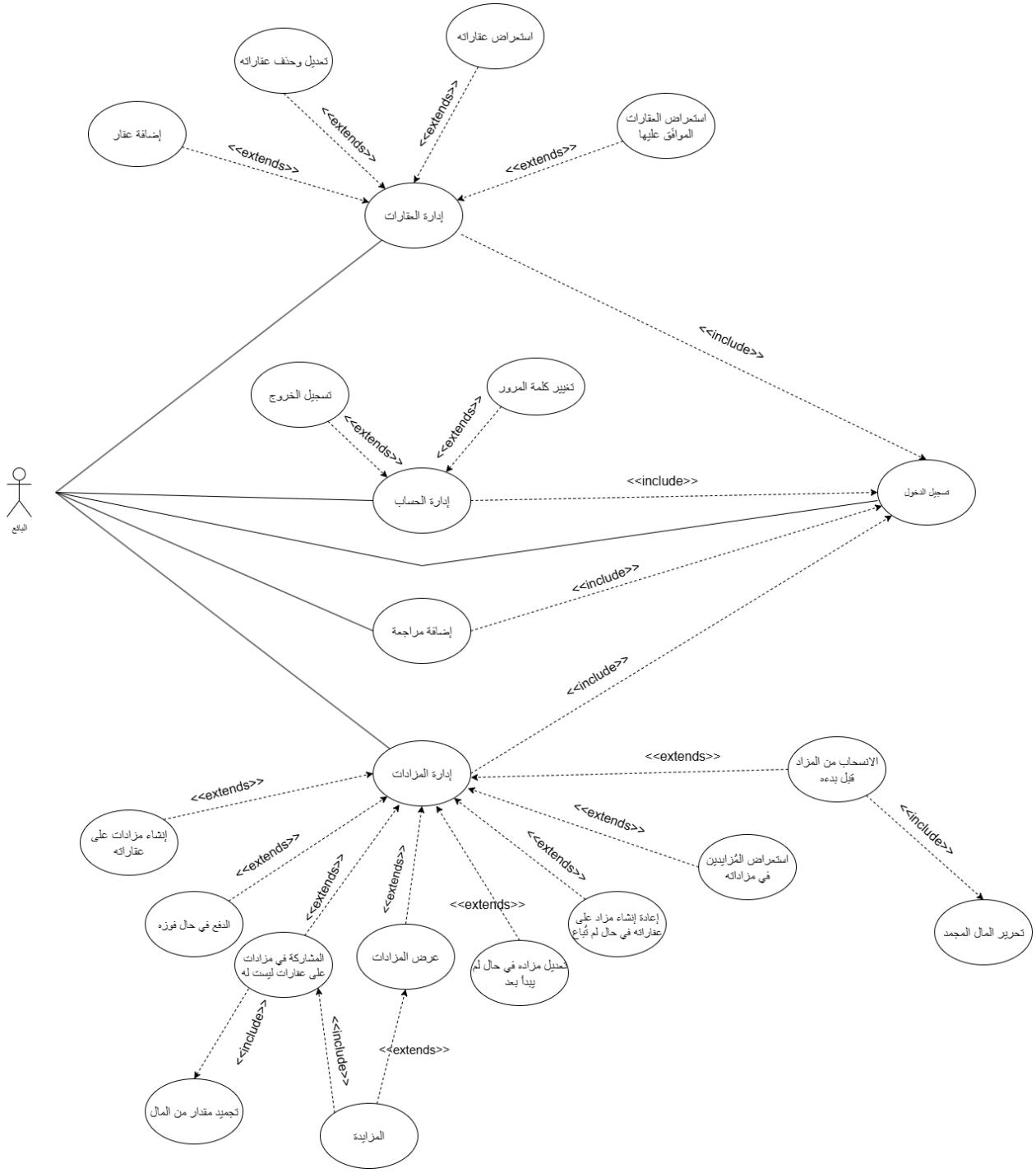


2- مخططات حالات الاستخدام:

2-1- حالات استخدام مدير النظام (Admin):

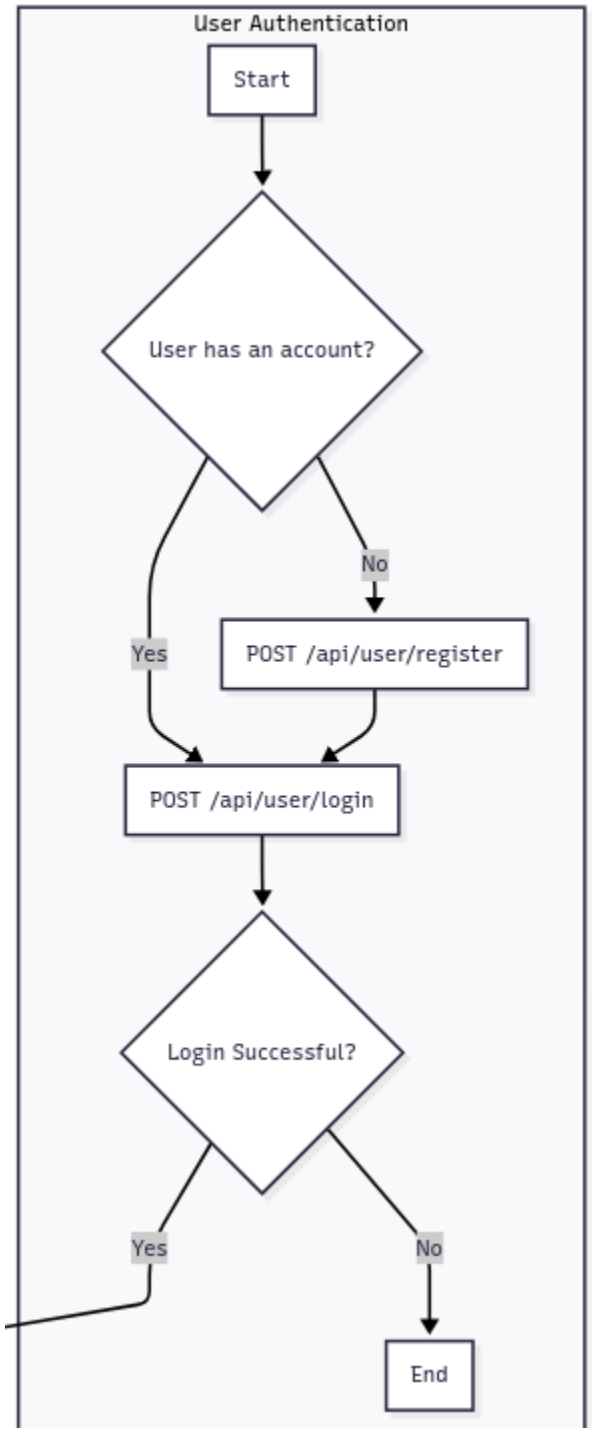


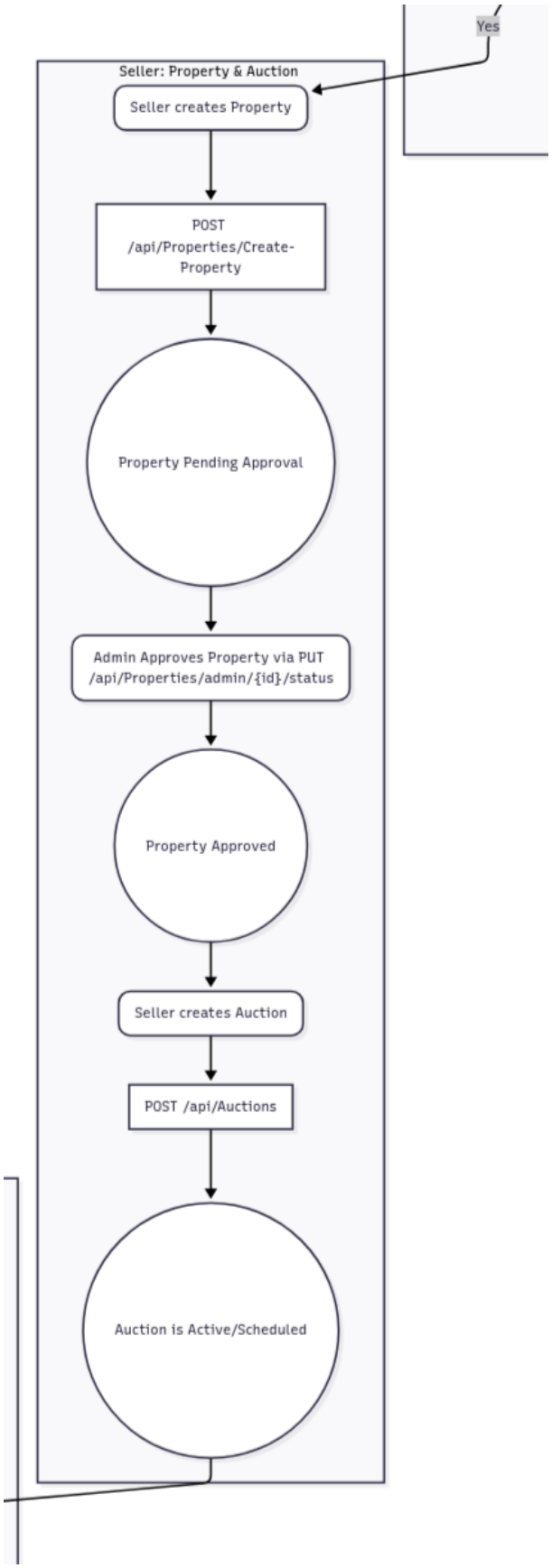
2-2- حالات استخدام البائع (Seller):

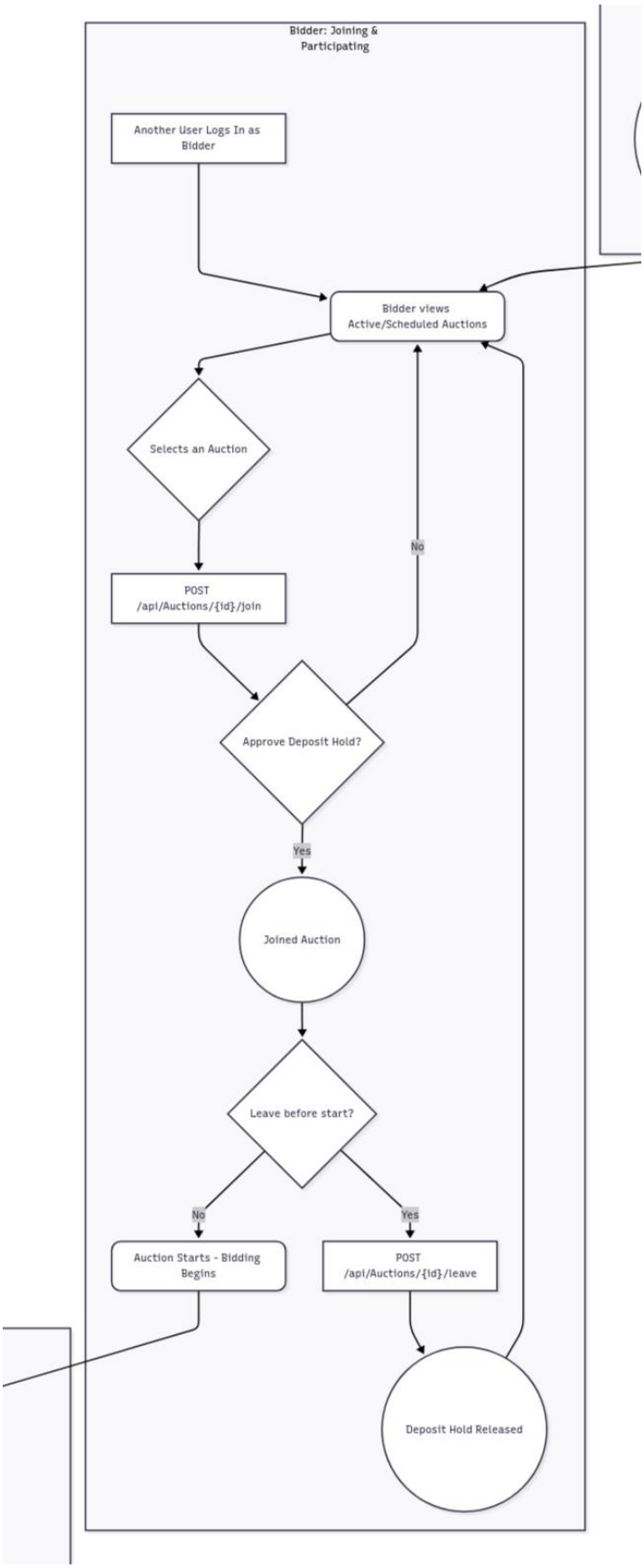


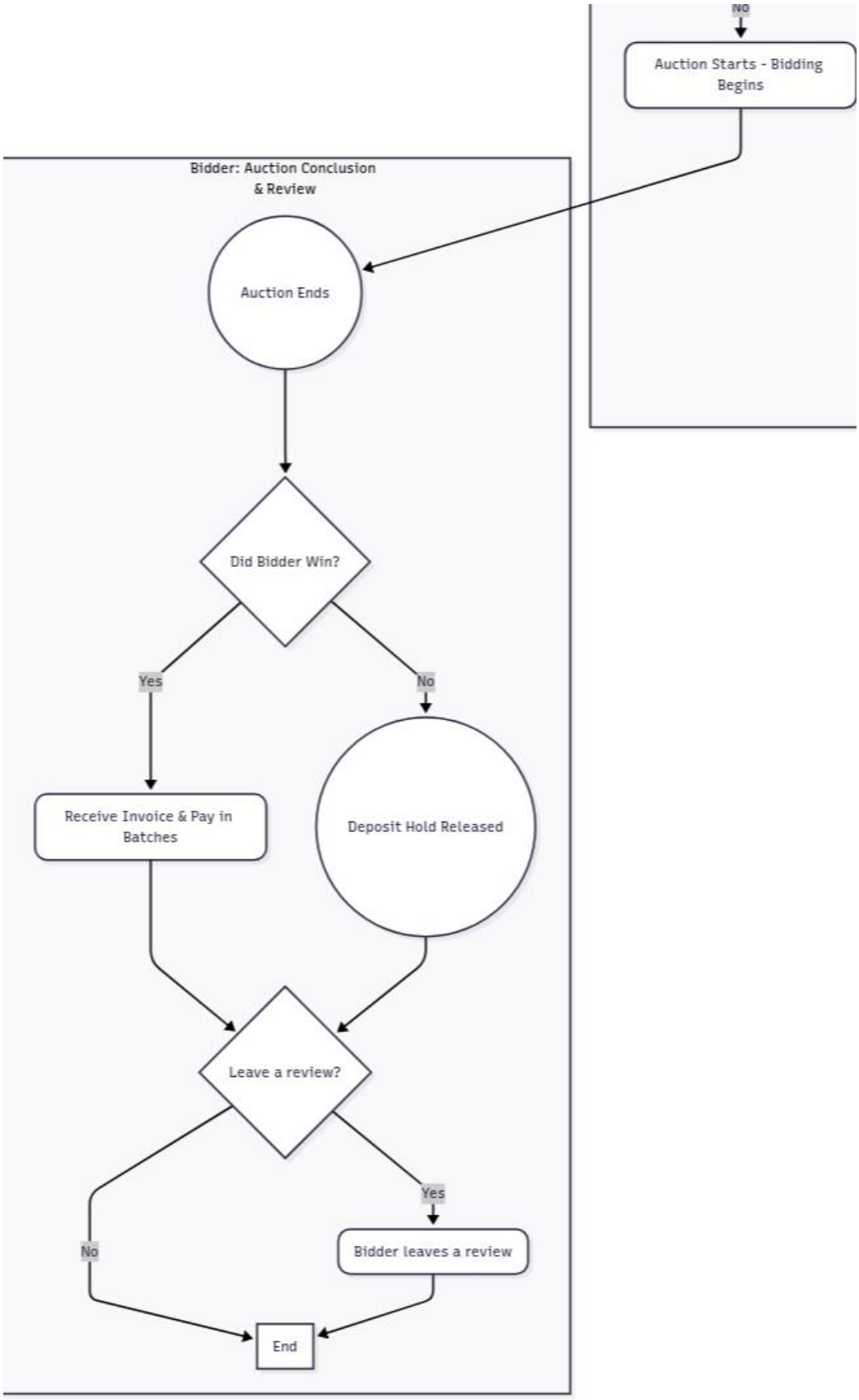
2-3- حالات استخدام المُزايد (Bidder):

3- مخطط تدفقي من تسجيل الدخول إلى انتهاء مزاد:




Yes
Seller: Property & Auction
Seller creates Property
POST /api/Properties/Create-Property
Property Pending Approval
Admin Approves Property via PUT /api/Properties/admin/{id}/status
Property Approved
Seller creates Auction
POST /api/Auctions
Auction is Active/Scheduled


Bidder: Joining & Participating
Another User Logs In as Bidder
Bidder views Active/Scheduled Auctions
Selects an Auction
POST /api/Auctions/{id}/join
No
Approve Deposit Hold?
Yes
Joined Auction
Leave before start?
No
Yes
Auction Starts - Bidding Begins
POST /api/Auctions/{id}/leave
Deposit Hold Released


No
Auction Starts - Bidding Begins
Bidder: Auction Conclusion & Review
Auction Ends
Did Bidder Win?
Yes
No
Receive Invoice & Pay in Batches
Deposit Hold Released
Leave a review?
Yes
No
Bidder leaves a review
End

الدراسة النظرية

نشرح في هذا الفصل عن المعمارية المتبعة لتنجيز المشروع

1- المعمارية النظيفة (Clean Architecture):

تُعد المعمارية النظيفة أحد أبرز نماذج التصميم البرمجي التي تهدف إلى تنظيم الرماز المصدري بطريقة تضمن بناء أنظمة قوية، قابلة للصيانة، ومستقلة عن التفاصيل التقنية الخارجية. تقوم هذه البنية على مبدأ فصل الاهتمامات، حيث يتم عزل منطق العمل الأساسي عن آليات العرض، وقواعد البيانات، وأطر العمل.

1-1- قاعدة الاعتمادية (The Dependency Rule):

المبدأ الجوهري الذي يحكم المعمارية النظيفة هو "قاعدة الاعتمادية". تنص هذه القاعدة على أن جميع الاعتماديات في الرماز المصدري يجب أن تتجه حصراً نحو الداخل .هذا يعني أن الطبقات الداخلية، التي تحتوي على منطق العمل. على سبيل المثال، يجب ألا يحتوي الرماز الذي يدير قواعد العمل الأساسية على أي اعتماديات على نوع قاعدة البيانات المستخدمة. هذا الاتجاه الأحادي للاعتمادية يحمي قلب النظام من التغييرات في التقنيات الخارجية.

1-2- أهداف المعمارية النظيفة

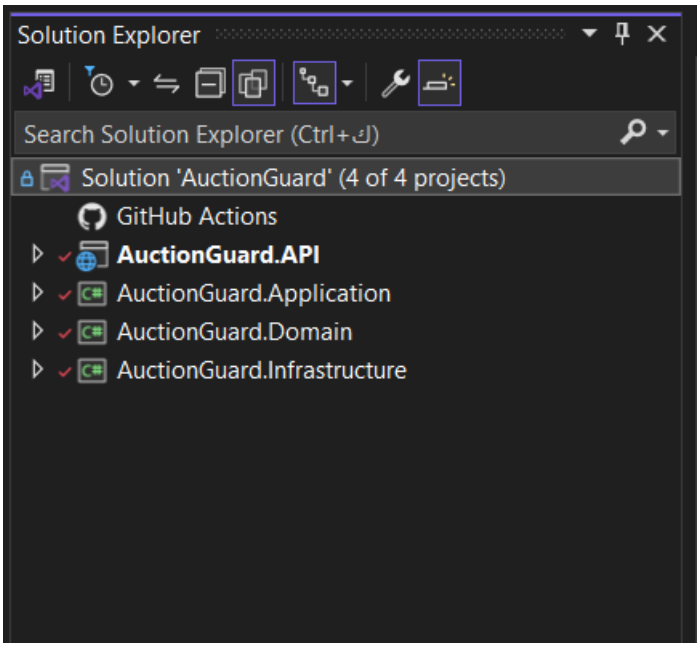
تهدف المعمارية النظيفة إلى تحقيق عدة مزايا جوهرية:

- **الاستقلالية عن أطر العمل** :لا يعتمد منطق العمل على أي إطار عمل محدد، مما يسمح للمطورين بتحديث أو استبدال أطر العمل دون التأثير على قواعد العمل الأساسية.
- **القابلية للاختبار** :(**Testability**) يمكن اختبار منطق العمل وقواعده الأساسية بمعزل تام عن أي عناصر خارجية، مما يجعل الاختبارات الوحدوية (Unit Tests) سريعة، وموثوقة، وسهلة الكتابة.
- **الاستقلالية عن واجهة المستخدم** :يمكن تغيير واجهة المستخدم بالكامل—من تطبيق ويب إلى تطبيق سطح مكتب أو جوال—دون الحاجة إلى إجراء تغييرات في بقية النظام.

- **الاستقلالية عن قاعدة البيانات** :يمكن تبديل نظام إدارة قواعد البيانات (على سبيل المثال، منPostgreSQL إلىOracle) دون المساس بالطبقات الداخلية للنظام.
- **القابلية العالية للصيانة** :نتيجة للفصل الواضح بين المسؤوليات، يصبح فهم النظام وتعديله وتوسعته أكثر سهولة وأقل خطورة على المدى الطويل.

2- تطبيق المعمارية النظيفة في المشروع:

تم تطبيق المعمارية النظيفة في هذا المشروع من خلال تقسيم الرماز إلى طبقات منفصلة ومستقلة، لكل منها مسؤوليات محددة. يتجلى هذا التنظيم في تقسيم المشروع إلى عدة مشاريع مستقلة وهي:



1- طبقة المجال **AuctionGuard.Domain**:

تُمثل هذه الطبقة **قلب النظام** وجوهره، وهي الطبقة الأكثر داخلية وعمقاً في المعمارية. يتضح تطبيق مبادئ المعمارية النظيفة في هذه الطبقة من خلال العناصر التالية:

- **الكيانات:** تحتوي هذه الطبقة على كائنات العمل الأساسية (Business Objects) هذه الكيانات لا تحتوي على أي شيفرة تعتمد على تقنيات خارجية، بل تُمثل المفاهيم والقواعد الحاكمة للمجال نفسه (مثل حالة المزاد أو علاقة المالك بالعقار).

- **واجهات المستودعات:** تُعرّف هذه الطبقة العقود (Contracts) اللازمة لعمليات استمرارية البيانات (Data Persistence) من خلال واجهات مثل IGenericRepository وIUnitOfWork. هذه الواجهات تحدد ماذا يجب أن تفعل عمليات البيانات مثل (GetByIdAsync AddAsync)، ولكنها لا تحدد كيف سيتم تنفيذها. هذا يعزل طبقة المجال تماماً عن تفاصيل تقنية قاعدة البيانات.

2- طبقة التطبيق **AuctionGuard.Application** :

تحيط هذه الطبقة بطبقة المجال وتحتوي على منطق العمل الخاص بالتطبيق (حالات الاستخدام). تم تطبيق مبادئ المعمارية النظيفة هنا كالتالي:

الخدمات: كل خدمة تمثل حالة استخدام محددة (مثل "إنشاء مزاد" أو "تسجيل مستخدم"). تقوم هذه الخدمات بتنسيق تدفق البيانات، حيث تستخدم الكيانات من طبقة المجال وتعتمد على الواجهات مثل (IUnitOfWork) لتنفيذ العمليات على البيانات.

واجهات الخدمات: تُعرّف هذه الطبقة واجهات للخدمات. تعمل هذه الواجهات كعقود لمنطق التطبيق، مما يسمح للطبقات الخارجية مثل طبقة العرض بالاعتماد على هذه العقود بدلاً من التنفيذ المباشر، وهو ما يسهل عملية الاختبار والاستبدال.

كائنات نقل البيانات (Data Transfer Objects – DTOs): يتم استخدام كائنات نقل البيانات مثل وسيلة لنقل البيانات من وإلى هذه الطبقة. هذا يمنع تسريب كائنات المجال إلى الطبقات الخارجية.

3- طبقة البنية التحتية **AuctionGuard.Infrastructure** :

هذه الطبقة هي المسؤولة عن تنفيذ التفاصيل التقنية التي تحتاجها الطبقات الداخلية. إنها بمثابة "لاصق" يربط منطق العمل بالعالم الخارجي. ويتضح ذلك في:

- **تنفيذ المستودعات:** تحتوي على فئات مثل AppGenericRepository و UnitOfWork التي تقوم بتنفيذ الواجهات المعرَّفة في طبقة المجال .(IGenericRepository, IUnitOfWork) يتم هذا التنفيذ باستخدام تقنية محددة وهي **Entity Framework Core**
- **سياق قاعدة البيانات (DbContext)**: تحتوي على AuctionGuardDbContext، وهو التنفيذ الملموس للاتصال بقاعدة البيانات. هذا هو المكان الذي يتم فيه تحديد تفاصيل قاعدة البيانات العلائقية.
- **الخدمات الخارجية:** تحتوي على تنفيذ لخدمات خارجية مثل التعامل مع بوابات الدفع .(PayPalClientService)

4- طبقة العرض **AuctionGuard.API**:

تُمثل هذه الطبقة **نقطة الدخول إلى النظام** وهي الطبقة الأكثر خارجية. في هذا المشروع، تتمثل في واجهة برمجة تطبيقات (API)مبنية باستخدام.ASP.NET Core

المتحكمات(Controllers) : تستقبل طلبات HTTP لا تحتوي هذه المتحكمات على أي منطق عمل، بل تقتصر مهمتها على استلام الطلبات، وتمريرها إلى الخدمات المناسبة في طبقة التطبيق، ثم إرجاع الاستجابة.

الاعتماد على واجهات الخدمات :تعتمد المتحكمات على واجهات الخدمات التي يتم حقنها باستخدام حقن الاعتمادية بدلاً من الاعتماد على الفئات المباشرة، مما يسهل اختبار المتحكمات بشكل منفصل.

تفاصيل إطار العمل :هذه الطبقة هي المكان الوحيد الذي توجد فيه اعتماديات مباشرة على إطار عمل الويب (ASP.NET Core)، مثل السمات (Attributes) الخاصة بالمسارات ([Route]) والصلاحيات ([Authorize]).

3- تحليل الأنماط التصميمية المطبقة في المشروع:

3-1- نمط المستودع (Repository Pattern):

نمط المستودع هو نمط تصميم هيكلي يهدف إلى فصل منطق الوصول إلى البيانات عن منطق العمل. يقوم هذا النمط بإنشاء طبقة تجريدية بين طبقة التطبيق ومصدر البيانات، مما يوفر واجهة موحدة للتعامل مع عمليات البيانات (الإنشاء، القراءة، التعديل، والحذف) دون الحاجة إلى معرفة تفاصيل التنفيذ الداخلي لمصدر البيانات.

تطبيق نمط المستودع في المشروع:

- الواجهات المجردة: تم تعريف واجهة عامة IGenericRepository<T> في طبقة المجال. هذه الواجهة تحدد العمليات الأساسية التي يمكن إجراؤها على أي كيان في النظام.
- التنفيذ الملموس: في طبقة البنية التحتية، تم إنشاء فئة AppGenericRepository<T> التي تقوم بتنفيذ واجهة IGenericRepository<T>. هذه الفئة تحتوي على الرماز الذي يستخدم سياق قاعدة البيانات لإجراء العمليات على قاعدة البيانات.
- الاستخدام في الخدمات: تعتمد الخدمات في طبقة التطبيق على واجهة IGenericRepository<T> بدلاً من التنفيذ المباشر، مما يجعل منطق العمل مستقلاً تماماً عن تقنية الوصول إلى البيانات.

3-2- نمط وحدة العمل (Unit of Work Pattern):

نمط وحدة العمل يُستخدم لإدارة مجموعة من العمليات على مصدر البيانات كمعاملة مناقلة واحدة. يضمن هذا النمط أنه إما أن تنجح جميع العمليات معاً أو تفشل جميعها، مما يحافظ على تكامل البيانات. يقوم هذا النمط بتتبع جميع التغييرات التي تطرأ على الكيانات ضمن معاملة واحدة، ثم يقوم بتنفيذها دفعة واحدة عند الاستدعاء.

تطبيق نمط وحدة العمل في المشروع:

تم تطبيق هذا النمط لضمان تنفيذ العمليات المعقدة كوحدة ذرية.

على سبيل المثال، عند إنشاء مزاد جديد، قد يتطلب الأمر إضافة سجل في جدول المزادات وتحديث حالة العقار المرتبط به. نمط وحدة العمل يضمن أن تتم هاتان العمليتان كعملية واحدة غير قابلة للتجزئة.

- **الواجهة المجردة** :تم تعريف واجهة لنمط وحدة العمل في طبقة المجال، وتحتوي هذه الواجهة على خاصيات تُمثل مستودعات الكيانات المختلفة بالإضافة إلى دالة CompleteAsync لحفظ جميع التغييرات.

- **التنفيذ الملموس** :في طبقة البنية التحتية، يقوم الصف UnitOfWork بتنفيذ هذه الواجهة. تحتفظ هذه الفئة بنسخة من سياق التطبيق في قاعدة البيانات وتستخدم دالة SaveChangesAsync الخاصة به لتنفيذ جميع التغييرات دفعة واحدة.
- **الاستخدام** :يتم حقن واجهة وحدة العمل في الخدمات التي تتطلب عمليات متعددة، مما يبسّط إدارة المعاملات ويضمن تناسق البيانات.

3-3- نمط حقن الاعتمادية (Dependency Injection Pattern):

حقن الاعتمادية هو نمط تصميم برمجي يهدف إلى تحقيق مبدأ عكس الاعتمادية. يتم حقن هذه الاعتماديات من مصدر خارجي يُعرف بحاوية حقن الاعتمادية حيث يقلل من الارتباطات بين عناصر النظام ويسهل عملية اختبارها واستبدال مكوناتها.

تطبيق نمط حقن الاعتمادية في المشروع:

- **تسجيل الخدمات** :في طبقة العرض، وتحديداً في ملف Program.cs، يتم تسجيل جميع الخدمات والواجهات في حاوية حقن الاعتمادية. على سبيل المثال، يتم تسجيل واجهة نمط وحدة العمل مع التنفيذ الخاص به.
- **الحقن في الباني (Constructor Injection)**: يتم حقن الاعتماديات في بواني الصفوف التي تحتاجها مما يسمح للمتحكم باستخدام خدمات منطق العمل دون أن يكون مسؤولاً عن إنشائها.
- **تقليل الارتباط**: إن تطبيق هذا النمط يجعل النظام مرناً للغاية، حيث يمكن تغيير تنفيذ أي واجهة.

3-4- نمط كائن نقل البيانات (Data Transfer Object – DTO Pattern):

نمط كائن نقل البيانات يُستخدم لنقل البيانات بين طبقات النظام المختلفة، خاصة بين طبقة التطبيق وطبقة العرض. الهدف من هذا النمط هو منع تسريب تفاصيل نماذج المجال إلى الطبقات الخارجية. يحتوي كائن نقل البيانات على خاصيات للبيانات فقط، ولا يحتوي على أي منطق عمل.

تطبيق نمط كائن نقل البيانات في المشروع: تم استخدام هذا النمط لفصل نماذج العرض عن نماذج المجال.

- **تعريف كائنات نقل البيانات**: في طبقة التطبيق، تم إنشاء مجلد يحتوي على كائنات لنقل البيانات بحيث كل كائن من كائنات نقل البيانات مصمم خصيصاً لحالة استخدام معينة.
- **منع تسريب النماذج** :تستهلك المتحكمات في طبقة العرض هذه الكائنات كمدخلات وتقوم بإرجاعها كمخرجات، بدلاً من التعامل المباشر مع كائنات المجال. هذا يضمن أن أي تغيير في نماذج المجال لا يؤثر مباشرة على واجهة تطبيق برمجي والعكس صحيح.
- **التحويل بين النماذج** :يمكن استخدام التحويل اليدوي لتحويل البيانات بين كائنات المجال وكائنات نقل البيانات داخل طبقة التطبيق.

4- أجزاء النظام الخلفي وتكاملها:

4-1- إطار عمل (**NET Core**.):

تم تطوير هذا النظام بالاعتماد على **NET Core**.، وهو إطار عمل حديث ومفتوح المصدر من شركة Microsoft. تم اختياره لعدة أسباب جوهرية، أبرزها:

- **الأداء العالي** :يوفر NET Core. أداءً ممتازاً يضمن استجابة سريعة للنظام حتى مع تزايد عدد المستخدمين والعمليات.
- **متعدد المنصات**: يعمل الإطار على مختلف أنظمة التشغيل مثل Windows، Linux، وMac، مما يمنح مرونة كبيرة في خيارات الاستضافة والنشر.
- **قابلية التوسع** :صُمم ليكون معيارياً وخفيف الوزن، مما يسهل إضافة الميزات وتوسيع نطاق النظام مستقبلاً دون التأثير على أدائه.

4-2- نموذج ASP.NET Core Web API البرمجي:

يعتمد ASP.NET Core Web API بشكل أساسي على النمط المعماري REST (Representational State Transfer)، وهو مجموعة من المبادئ التوجيهية لتصميم تطبيقات الشبكة القابلة للتوسع. يظهر تطبيق هذه المبادئ في الإطار على النحو التالي:

التعامل مع الموارد :يتمحور التصميم حول مفهوم الموارد، حيث يمثل كل شيء مورداً فريداً يمكن الوصول إليه عبر مُعرّف موحد للموارد.(URI)

استخدام أفعال HTTP القياسية :يتم استخدام دلالات بروتوكول HTTP بشكل كامل، حيث تُستخدم الأفعال القياسية (GET, POST, PUT, DELETE, PATCH) لإجراء العمليات الأساسية على الموارد (الاسترجاع، الإنشاء، التحديث، الحذف).

انعدام الحالة :كل طلب HTTP من العميل إلى الخادم يجب أن يحتوي على جميع المعلومات اللازمة لفهمه وتنفيذه، دون أن يعتمد الخادم على أي سياق أو جلسة سابقة للعميل. هذا المبدأ يعزز من قابلية التوسع والموثوقية.

تفاوض المحتوى(Content Negotiation) : يوفر الإطار آلية مدمجة تسمح للعميل والخادم بالاتفاق على تنسيق البيانات المتبادلة، ويعتبر تنسيق JSON (JavaScript Object Notation) هو التنسيق الأكثر شيوعاً والافتراضي نظراً لخفة وزنه وسهولة تحليله بواسطة مختلف المنصات، مع دعم كامل لتنسيقات أخرى مثل XML.

4-3- مكتبة SignalR للتواصل اللحظي:

SignalR هي مكتبة مفتوحة المصدر (Open-Source) تندرج ضمن حزمة تقنياتASP.NET ، تم تصميمها بهدف تبسيط عملية إضافة وظائف الويب اللحظية (Real-time Web Functionality) إلى التطبيقات. في سياق بنية الويب التقليدية القائمة على بروتوكولHTTP ، يتبع الاتصال نموذج الطلب والاستجابة (Request-Response) الذي يبدأه العميل، مما يجعل الاتصالات التي يبدأها الخادم غير فعالة بطبيعتها. جاءت SignalR لمعالجة هذا القصور، حيث توفر طبقة تجريد قوية تُمكّن الخادم من دفع المحتوى إلى العملاء المتصلين بشكل فوري ولحظي.

4-4- نموذج التحكم في الوصول الهجين: دمج الأدوار والأذونات (RBAC – PBAC):

في سياق أمن التطبيقات، يمثل التحكم في الوصول آلية أساسية لفرض قيود على من يمكنه تنفيذ إجراءات معينة أو الوصول إلى موارد محددة.

يعتمد التحكم القائم على الأدوار (RBAC) على تعيين المستخدمين لأدوار محددة (مثل "مدير"، "بائع")، حيث يرتبط كل دور بمجموعة من الصلاحيات الضمنية. بينما يوفر التحكم القائم على الأذونات (PBAC) تحكماً أكثر دقة من خلال منح أذونات صريحة لإجراءات فردية (مثل "عرض الفواتير"، "حذف مستخدم")، بغض النظر عن الدور الوظيفي.

النهج الهجين الذي تبنيناه يجمع بين المفهومين؛ حيث يتم تعريف الأدوار كمجموعات منطقية من الأذونات، مما يسهل إدارة الصلاحيات على مستوى واسع، مع الاحتفاظ بالقدرة على التحقق من أذونات دقيقة ومحددة على مستوى نقطة النهاية

4-3-1- تصميم وتنظيم الأذونات (Permission Design)

يكمن جوهر هذا النموذج في التعريف الدقيق والمنظم للأذونات. في المشروع، تم تحقيق ذلك من خلال static class يسمى Permissions داخل طبقة:

```
/// <summary>
/// A static class to define all available permissions in the system.
/// This makes permissions easy to manage and prevents typos.
/// </summary>
66 references | Salman Mohsen, 6 days ago | 1 author, 2 changes
public static class Permissions
{
    5 references | Salman Mohsen, 13 days ago | 1 author, 1 change
    public static class Users
    {
        public const string View = "Permissions.Users.View";
        public const string Create = "Permissions.Users.Create";
        public const string Edit = "Permissions.Users.Edit";
        public const string Delete = "Permissions.Users.Delete";
        public const string Blacklist = "Permissions.Users.Blacklist";
    }

    11 references | Salman Mohsen, 13 days ago | 1 author, 1 change
    public static class Properties
    {
        public const string View = "Permissions.Properties.View";
        public const string Create = "Permissions.Properties.Create";
        public const string Edit = "Permissions.Properties.Edit";
        public const string Delete = "Permissions.Properties.Delete";
        public const string Approve = "Permissions.Properties.Approve";
    }

    26 references | Salman Mohsen, 6 days ago | 1 author, 2 changes
    public static class Auctions
    {
        public const string View = "Permissions.Auctions.View";
```

تحليل للتصميم:

- **التنظيم الهرمي:** تم تنظيم الأذونات بشكل هرمي باستخدام صفوف متداخلة، مما يعكس بنية وحدات النظام الأسلوب يعزز قابلية القراءة والصيانة، ويمنع تضارب الأسماء.
- **الثوابت النصية:** استخدام الثوابت النصية (const string) كمعرفات فريدة لكل إذن هو ممارسة قياسية تضمن الاتساق وتمنع الأخطاء المطبعية عند استخدام هذه الأذونات في أجزاء مختلفة من الكود.
- **المركزية:** وجود جميع تعريفات الأذونات في مكان واحد (Permissions.cs) يجعل النظام أسهل في الفهم والإدارة، حيث يمكن للمطورين مراجعة قائمة الصلاحيات الكاملة في ملف واحد.

4-3-2- ربط الأذونات بالأدوار (Associating Permissions with Roles)

الخطوة التالية في النموذج الهجين هي ربط هذه الأذونات الدقيقة بالأدوار. يتم ذلك في مرحلة بذر البيانات (Data Seeding) عند بدء تشغيل التطبيق لأول مرة. يقوم النظام بإنشاء الأدوار الأساسية ("Admin" "Bidder","Seller") ثم يمنح كل دور مجموعة من الأذونات.

4-3-3- فرض التحكم بالوصول:

يتم فرض قواعد التحكم في الوصول عند نقطة الطلب، حيث طورنا آلية مخصصة لتحقيق ذلك.

السمة المخصصة [HasPermission]: بدلاً من استخدام سمة [Authorize (Roles = "Admin")] التقليدية، تم إنشاء سمة مخصصة تسمى [HasPermission]. هذه السمة تقبل سلسلة نصية تمثل الإذن المحدد المطلوب لتنفيذ الإجراء.

```
/// <summary>
/// Gets a list of all users. Requires Admin role and View Users per
/// </summary>
[HttpGet]
[HasPermission(Permissions.Users.View)]
0 references | Salman Mohsen, 13 days ago | 1 author, 1 change
public async Task<IActionResult> GetAllUsers()
{
```

منطق التحقق من الأذونات: هذه هي العقدة المركزية التي يتم فيها التحقق الفعلي. تعمل هذه الفئة كمعالج تفويض (Authorization Handler) مخصص. عند وصول طلب إلى نقطة نهاية محمية بسمة [HasPermission], يقوم نظام ASP.NET Core بتنشيط هذا المعالج، والذي ينفذ الخطوات التالية:

1. يستخرج جميع مطالبات الأذونات (Permission claims) المرتبطة بالمستخدم الحالي. هذه المطالبات تأتي من الأدوار التي ينتمي إليها المستخدم.
2. يقارن قائمة الأذونات التي يمتلكها المستخدم مع الإذن المطلوب المحدد في السمة [HasPermission].
3. إذا تم العثور على تطابق، يعتبر الطلب مصرحاً به (Authorized) ويكمل مساره. وإلا، يتم رفض الطلب وإرجاع رمز حالة 403 Forbidden.

4-5- إدارة المزادات والعقارات:

- **خدمة العقارات:** مسؤولة عن إنشاء، تعديل، وحذف العقارات. تتكامل مع وحدة العمل لضمان أن جميع التغييرات على قاعدة البيانات تتم كمناقلة واحدة.
- **خدمة المزادات:** تدير دورة حياة المزاد، بدءاً من إنشائه وربطه بعقار معين، مروراً بتحديث حالته (مجدول، نشط، منتهي)، وانتهاءً بإلغائه أو إعادة إنشائه.
- **الخدمة الخلفية:** عمل هذه الخدمة في الخلفية بشكل مستمر للتحقق من حالة المزادات. تقوم تلقائياً بتغيير حالة المزاد من "مجدول" إلى "نشط" عند وصول وقت البدء، ومن "نشط" إلى "منتهي" عند وصول وقت الانتهاء، مما يضمن دقة التوقيتات دون تدخل يدوي.

4-6- نظام المزايدة والتواصل اللحظي:

يستخدم النظام تقنية SignalR لتوفير تواصل لحظي بين الخادم والعملاء أثناء المزايدة.

- **BiddingHub**: هو المحور المركزي لعمليات المزايدة. عندما يقوم مستخدم بعمل مزايدة بسعر جديد من خلال استدعاء دالة PlaceBid في الHub ، تقوم الخدمة (BiddingService) بالتحقق من صحة المزايدة (التأكد من أن قيمة المزايدة أعلى من المزايدة الحالية وبمقدار الزيادة الأدنى المسموح به).
- **تكامل الخدمة مع الHub** : بعد حفظ المزايدة الجديد في قاعدة البيانات، تستدعي BiddingService خدمة (IAuctionNotifierالتي يتم تنفيذها بواسطة SignalRNotifier) لإرسال إشعار فوري لجميع المشاركين في نفس مجموعة المزاد (Auction Group) بالعرض الجديد، مما يضمن تحديث الواجهات الأمامية لجميع المستخدمين في نفس اللحظة.

4-7- التكامل مع بوابات الدفع:

النظام مصمم ليكون قابلاً للتوسع لدعم بوابات دفع متعددة من خلال واجهة IPaymentGatewayService. حالياً، تم تنفيذ التكامل مع PayPal.

- **PayPalClientService**: تتولى هذه الخدمة كافة عمليات التواصل المباشر معPayPal API ، مثل إنشاء طلبات الدفع للحصول على موافقة المستخدم، التحقق من عمليات الدفع، وإلغاء الحجوزات المالية.
- **تدفق عملية الدفع** :عندما يرغب مستخدم في الانضمام لمزاد يتطلب وديعة تأمين، تقوم AuctionParticipationServiceباستدعاء PayPalClientService لإنشاء طلب دفع بقيمة الوديعة. يتم إرجاع رابط موافقة (Approval URL) يُرسل للواجهة الأمامية ليقوم المستخدم بتأكيد العملية. بعد التأكيد، يتم إعادة توجيه المستخدم إلى نقطة نهاية (Callback Endpoint) في AuctionsController، والتي بدورها تؤكد عملية حجز المبلغ وتضيف المستخدم كمشارك في المزاد.

5- الواجهة الأمامية

5-1- React:

تُعد React مكتبة برمجية مفتوحة المصدر مبنية على لغة JavaScript، وهي مُصممة خصيصاً لبناء واجهات المستخدم (User Interfaces)، لا سيما في تطبيقات الصفحة الواحدة (Single-Page Application). تم تطويرها والمحافظة عليها من قبل شركة Meta، وتتمحور فلسفتها حول البنية القائمة على المكونات يُمكّن هذا النموذج المعماري المطورين من إنشاء مكونات واجهة مستخدم مُغلفة ومستقلة، يمكن إعادة استخدامها وتركيبها معاً لبناء واجهات مستخدم معقدة بكفاءة عالية.

من أبرز المزايا التقنية التي تقدمها React هي:

نموذج المستند الكائني الافتراضي (Virtual DOM): تستخدم React آلية ذاكرة وسيطة تُعرف بـ (In-memory cache) Virtual DOM. وهي تمثيل برمجي خفيف للبنية الفعلية لنموذج المستند الكائني(DOM) . عند حدوث أي تغيير في حالة التطبيق، تقوم React بتحديث هذا التمثيل الافتراضي أولاً، ثم تنفذ خوارزمية تسوية (Reconciliation)لمقارنة النسخة المحدثة مع النسخة السابقة، وتُحدد من خلالها الحد الأدنى من التغييرات المطلوبة لتحديث الـ DOM الفعلي. هذه العملية تقلل بشكل كبير من عمليات التلاعب المباشر بالDOM ، والتي تُعد مكلفة من حيث الأداء، مما يؤدي إلى تحسين سرعة الاستجابة وسلاسة تجربة المستخدم. بالإضافة إلى ذلك، تتميز React بكونها مكتبة وليست إطار عمل متكامل، مما يمنحها مرونة عالية للتكامل مع مكتبات وتقنيات أخرى متخصصة في إدارة الحالة مثل Redux، أو مكتبات مكونات واجهة المستخدم الجاهزة مثل Material-UI.

5-2- بنية الواجهة الأمامية

تم بناء الواجهة الأمامية AuctionGuardClient باستخدام إطار العمل React مع TypeScript لضمان كتابة رماز آمن من حيث الأنواع (Type-Safe).

- إدارة الحالة: يتم استخدام مكتبة TanStack React Queryلإدارة حالة الخادم (Server State) بفعالية، بما في ذلك جلب البيانات، التخزين المؤقت وتحديثها تلقائياً. أما لإدارة حالة المصادقة العامة، فيتم الاعتماد على React Context API.

- التصميم وواجهة المستخدم :يُستخدم Tailwind CSS لتصميم الواجهات بأسلوب Utility-First، مدعوماً بمكتبة مكونات shadcn/ui التي توفر مكونات قابلة لإعادة الاستخدام ومصممة بعناية لتعزيز تجربة المستخدم وسهولة الوصول.
- التواصل اللحظي: يتم استخدام مكتبة SignalR للتواصل اللحظي بين العميل والخادم، وهو أمر حاسم في نظام المزادات لتحديث عروض الأسعار بشكل فوري لجميع المشاركين.