تطبيق موبايل لبيع وشراء المستعمل

الملخص

Anstract

الفهرس

[1 الفصل الأول التعريف بالمشروع 7](#_Toc204935723)

[1.1 خلفية المشروع 7](#_Toc204935724)

[1.2 أهمية المشروع 7](#_Toc204935725)

[1.3 هدف المشروع 7](#_Toc204935726)

[1.4 منهجية العمل 7](#_Toc204935727)

[1.5 حدود المشروع 8](#_Toc204935728)

[1.6 المتطلبات الوظيفية 8](#_Toc204935729)

[1.7 المتطلبات غير الوظيفية 9](#_Toc204935730)

[1.8 دراسة الجدوى: 10](#_Toc204935731)

[2 الدراسة المرجعية 12](#_Toc204935732)

[2.1 الأنماط المعمارية في الأنظمة C2C: Modular Monolith vs. Microservices 12](#_Toc204935733)

[2.2 وسطاء الرسائل (Message Brokers) 12](#_Toc204935734)

[2.3 الذكاء الاصطناعي وتحليل المشاعر في تطبيقات المستخدم النهائي 13](#_Toc204935735)

[2.3.1 النماذج المعتمدة على المحولات Transformers: 13](#_Toc204935736)

[2.3.2 النموذج الهجين CNN–LSTM 14](#_Toc204935737)

[2.3.3 SVM + TF-IDF 14](#_Toc204935738)

[2.3.4 نماذج اللغة الضخمة (LLMs) 15](#_Toc204935739)

[2.4 النماذج الرئيسية في التجارة الإلكترونية 16](#_Toc204935740)

[2.5 تطبيقات تقنية البلوكتشين في التجارة الإلكترونية 19](#_Toc204935741)

[3 الفصل الثالث الدراسة التحليلية 21](#_Toc204935742)

[3.1 مخطط حالات الاستخدام: 21](#_Toc204935743)

[3.1.1 مخطط حالات استخدام ال Authentication & User Management 23](#_Toc204935744)

[3.1.2 مخطط حالات استخدام Product Catalog Management 24](#_Toc204935745)

[3.1.3 مخطط حالات استخدام Shopping Cart & Orders 24](#_Toc204935746)

[3.1.4 مخطط حالات استخدام Payment System 25](#_Toc204935747)

[3.1.5 مخطط حالات استخدام Communication System 26](#_Toc204935748)

[3.1.6 مخطط حالات استخدام System Administration 27](#_Toc204935749)

[3.2 السرد النصي لحالات الاستخدام: 27](#_Toc204935750)

[3.3 مخططات تسلسل النظام SSD: 35](#_Toc204935751)

[3.3.1 كتابة مراجعة: 35](#_Toc204935752)

[3.3.2 معالجة الدفع: 36](#_Toc204935753)

[3.3.3 إتمام طلب: 37](#_Toc204935754)

[3.3.4 مخطط تدفق النظام: 38](#_Toc204935755)

[3.4 مخططات التعاون: 39](#_Toc204935756)

[3.4.1 مخطط التعاون لكتابة مراجعة: 39](#_Toc204935757)

[3.4.2 مخطط التعاون لمعالجة الدفع: 39](#_Toc204935758)

[3.4.3 مخطط التعاون لإتمام طلب: 40](#_Toc204935759)

[3.4.4 مخطط تعاوني يوضح التكامل بين حالات الاستخدام المذكورة: 40](#_Toc204935760)

[4 الفصل الرابع الدراسة النظرية 41](#_Toc204935761)

[4.1 البنية الخلفية (Back-End) 41](#_Toc204935762)

[4.1.1 Modular Monolithic 42](#_Toc204935763)

[4.1.2 Clean Architecture 42](#_Toc204935764)

[4.1.3 RabbitMQ 42](#_Toc204935765)

[4.2 الذكاء الاصطناعي وتحليل المشاعر 41](#_Toc204935766)

[4.3 نظام التسجيل Logginig system: 41](#_Toc204935767)

[4.4 تقنية سلسلة الكتل (Blockchain) ونظام Hyperledger Fabric ودورهما في تطبيقات التجارة الإلكترونية 41](#_Toc204935768)

[4.5 الواجهة الأمامية (Front-End) 41](#_Toc204935769)

[5 تصميم النظام 42](#_Toc204935770)

[5.1 مقدمة: 42](#_Toc204935771)

[5.2 نظام الواجهة الخلفية: 44](#_Toc204935772)

[5.3 نظام نموذج توقع التقييم: 45](#_Toc204935773)

[5.3.1 المنهجيات التجريبية 45](#_Toc204935774)

[5.4 نظام البلوكتشين: 46](#_Toc204935775)

[5.4.1 سلطات الشهادات (Certificate Authorities – CAs) 46](#_Toc204935776)

[5.4.2 المنظمات النظيرة (Peer Organizations) 47](#_Toc204935777)

[5.4.3 خدمة الترتيب (Ordering Service) 47](#_Toc204935778)

[5.4.4 طبقة التطبيق (Application Layer) 47](#_Toc204935779)

[5.4.5 تفاعل المكوّنات في الشبكة 48](#_Toc204935780)

[5.5 نظام الـ Front-end 49](#_Toc204935781)

[Presentation Layer (طبقة العرض): 50](#_Toc204935782)

[Business Logic and State Management Layer (طبقة المنطق وإدارة الحالة): 50](#_Toc204935783)

[Data Layer (طبقة البيانات): 50](#_Toc204935784)

[Core / Infrastructure (الطبقة التحتية الأساسية): 50](#_Toc204935785)

[5.5.1 تفاعل المكوّنات في نظام الواجهة 51](#_Toc204935786)

[5.6 لوحة التحكم: 51](#_Toc204935787)

[6 الفصل السادس الأدوات واللغات المستخدمة 52](#_Toc204935788)

[6.1 Back-End: 52](#_Toc204935789)

[6.2 لوحة التحكم: 54](#_Toc204935790)

[6.3 AI: 54](#_Toc204935791)

[6.4 Blockchain: 54](#_Toc204935792)

[6.5 Front-End: 54](#_Toc204935793)

[7 الفصل السابع القسم العملي 55](#_Toc204935794)

[7.1 البنية الخلفية: 55](#_Toc204935795)

[7.1.1 هيكلة الوحدات في النظام: 55](#_Toc204935796)

[7.1.2 البنية المعمارية لكل وحدة: 57](#_Toc204935797)

[7.1.3 تنفيذ نظام التسجيل باستخدام Serilog وSeq 67](#_Toc204935798)

[7.2 لوحةِ الإدارة 67](#_Toc204935799)

[7.3 البنية الأمامية 69](#_Toc204935800)

[7.4 الذكاء الاصطناعي 69](#_Toc204935801)

[7.4.1 مقدمة: 69](#_Toc204935802)

[7.4.2 تفاصيل الإنجاز: 69](#_Toc204935803)

[7.5 البلوكتشين 72](#_Toc204935804)

[7.6 توزيع المهام والتنسيق بين أعضاء الفريق 77](#_Toc204935805)

[8 الفصل السابع النتائج والاختبار 79](#_Toc204935806)

[8.1 اختبار الوحدات Unit test 79](#_Toc204935807)

[9 الخاتمة والآفاق المستقبلية 79](#_Toc204935808)

[10 المراجع 79](#_Toc204935809)

# الفصل الأول التعريف بالمشروع

## خلفية المشروع

يشهد العالم اليوم تحولاً رقمياً متسارعاً في مختلف القطاعات، وعلى رأسها قطاع التجارة. ومع التوسع الكبير في استخدام الهواتف الذكية وتطبيقات الهاتف المحمول، أصبحت الأسواق الإلكترونية وسيلة فعالة لتسهيل عمليات البيع والشراء. وفي السياق المحلي، تعاني التجارة التقليدية، خاصة في سوق المنتجات المستعملة في سوريا، من مشكلات تتعلق بعدم التنظيم، وغياب الثقة بين الأطراف، وصعوبة الوصول إلى معلومات دقيقة وموثوقة عن المنتجات والبائعين.

## أهمية المشروع

ينطلق هذا المشروع من الحاجة إلى توفير حل رقمي حديث يدعم التجارة المحلية، ويساهم في تنشيط الاقتصاد السوري من خلال أتمتة عمليات البيع والشراء وتبادل السلع المستعملة. يتميز المشروع بإدماج تقنيتين متقدمتين:

* الذكاء الاصطناعي (AI) لتحسين تجربة المستخدم وتقديم تقييمات للبائعين الموجودين في النظام.
* سلسلة الكتل (Blockchain) لتعزيز الثقة والشفافية من خلال تخزين جميع المعاملات بطريقة غير قابلة للتعديل، مما يحد من حالات الاحتيال ويُوفر سجلاً موثقاً وآمناً لكل عملية تبادل.

## هدف المشروع

يهدف المشروع إلى تطوير تطبيق إلكتروني متكامل يعمل على نظام أندرويد، يتيح للمستخدمين عرض سلعهم وخدماتهم، ومبادلتها مع مستخدمين آخرين بطريقة موثوقة وآمنة. تتلخص الأهداف فيما يلي:

* رقمنة سوق السلع المستعملة في سوريا.
* تعزيز الثقة باستخدام تقنيات العقود الذكية وتسجيل المعاملات على البلوك تشين.
* استخدام خوارزميات الذكاء الاصطناعي لتقييم البائعين بناء على سجلهم السابق وسلوكهم داخل التطبيق، مما يساعد المستخدمين على اتخاذ قرارات شراء أكثر وعياً وثقة.
* بناء تجربة مستخدم سلسة وعملية ضمن بيئة آمنة وشفافة.

## منهجية العمل

يعتمد المشروع على منهجية Agile في تطوير البرمجيات، مما يسمح بالمرونة والاستجابة السريعة للتغييرات. كما سيتم اتباع أفضل ممارسات هندسة البرمجيات في التحليل، التصميم، التنفيذ، والاختبار، لضمان جودة المنتج النهائي واستقراره.

## حدود المشروع

يركز المشروع في مرحلته الحالية على السوق المحلي السوري فقط، مع إمكانية التوسع مستقبلاً.

في هذه المرحلة الأولى، سيكون التطبيق موجهاً بشكل حصري لأجهزة الهواتف الذكية التي تعمل بنظام التشغيل Android، نظراً لانتشاره الواسع بين المستخدمين في سوريا، وانخفاض تكلفته مقارنة بمنصات أخرى مثل iOS. وهذا يساهم في ضمان وصول شريحة أكبر من المستخدمين إلى الخدمة المقترحة، ويعزز من فرص نجاح المشروع في البيئة المستهدفة.

## المتطلبات الوظيفية

تهدف المتطلبات الوظيفية إلى تحديد الوظائف الأساسية التي يجب أن يقدمها النظام للمستخدمين النهائيين. وتشمل ما يلي:

إدارة المستخدمين

* يجب أن يتيح النظام إنشاء حسابات للمستخدمين الجدد وتسجيل الدخول.
* يمكن للمستخدم تعديل معلومات حسابه الشخصي.
* يجب ان يسمح النظام للمستخدمين متابعة مستخدمين اخرين

إدارة الفئات والفئات الفرعية

* يمكن للمسؤول إنشاء فئات رئيسية غير محدودة (مثل: مركبات، دراجات، دراجات نارية)
* يجب أن يتم بناء هذه الفئات بطريقة ديناميكية بحيث يمكن التحكم بها من لوحة الإدارة دون تعديل في الكود.
* يمكن للبائع إضافة خصائص فرعية وتحديد قيمها (Attributes and its values).

إدارة المنتجات

* يمكن للمستخدمين إضافة منتجات جديدة، وتعديلها، وحذفها.
* يمكن للمستخدمين استعراض المنتجات.
* يمكن للمستخدمين استعراض المنتجات الخاصة بهم.
* يجب ربط كل منتج بفئة معينة، وبخصائصها المحددة.

تتضمن بيانات المنتج: العنوان، الوصف، الصور، السعر، وتاريخ الإضافة.

البحث والتصفية المتقدمة

* يتيح النظام للمستخدمين البحث عن المنتجات.
* تظهر عوامل التصفية الديناميكية بحسب الفئة المختارة او السعر أوالخصائص الثانوية، مما يساهم في تحسين تجربة المستخدم.

إدارة المفضلة

* يمكن للمستخدمين إضافة وازالة منتجات من وإلى قائمة المفضلة الخاصة بهم.

لوحة تحكم المسؤول

* توفر لوحة الإدارة إمكانية عرض الإحصائيات.
* يمكن للمسؤولين إدارة الفئات الموجودة.
* يمكن للمسؤولين مراقبة الطلبات والسلات والمستخدمين ومنتجاتهم والمعلومات اللازمة.

نظام الدفع

* يتيح النظام للمستخدمين الدفع مقابل المنتجات باستخدام وسائل متعددة.

إدارة المعاملات

* يتم تسجيل كل عملية بيع وشراء في قاعدة بيانات آمنة.
* يجب أن يُسجل كل عقد بيع بين البائع والمشتري في شبكة بلوكتشين لضمان الشفافية وعدم إمكانية التلاعب.

## المتطلبات غير الوظيفية

تهدف المتطلبات غير الوظيفية إلى تحديد الخصائص النوعية للنظام، والتي تضمن الأداء الجيد وسهولة الاستخدام والصيانة. وتشمل ما يلي:

القابلية للتوسع (Scalability)

* يجب أن يُصمم النظام ليدعم النمو المستقبلي سواء من حيث عدد المستخدمين أو المنتجات.

سهولة الاستخدام (Usability)

* يجب أن يتمتع النظام بواجهة استخدام بديهية وسهلة التصفح لكل من المستخدم العادي والمشرف.
* دعم تصفح التطبيق من خلال الأجهزة الذكية.

قابلية الصيانة (Maintainability)

* يعتمد النظام على هيكلية برمجية قابلة للتعديل والتوسعة.
* يمكن للمسؤول تعديل الفئات دون الحاجة لتعديل الكود

## دراسة الجدوى:

1. ملخص المشروع:

هدف المشروع: ربط البائعين والمشترين من الأفراد لتداول المنتجات المستعملة والجديدة (C2C).

أهميته: زيادة حركة البيع والشراء والاستفادة من المستعمل.

اسم المشروع:

الموقع والعملاء المستهدفون: سوريا، الشعب السوري.

فكرة المشروع:

تطبيق إلكتروني يعمل كمنصة تربط البائعين والمشترين من الأفراد لتداول المنتجات المستعملة والجديدة بطريقة مباشرة (C2C)، مما يخدم احتياجات شريحة واسعة من المستخدمين.

1. الأهداف:

* تسهيل تداول المنتجات المستعملة والجديدة بين الأفراد دون وسيط تقليدي.
* تقليل الهدر وإعادة تدوير المنتجات المستعملة.
* التوسع في السوق السوري والاستفادة من التحول الرقمي المتزايد.

1. تحليل السوق:

1) نظرة على السوق السوري:

* كان حجم الطلب على جميع أنواع السلع في السوق السورية هو الأقل منذ 50 عاماً.
* تحول اقتصادي جذري.
* أدى تخفيف القيود على الدولار وخفض الرسوم الجمركية إلى تدفق غير مسبوق للسلع الأجنبية التي كانت غائبة عن الأسواق السورية.
* نتج هذا التحول الاقتصادي المفاجئ عن تغيرات شاملة ففي السياسات الاقتصادية مما جعل الأسواق تزدهر بالبضائع المستوردة.

2) حجم الطلب

حجم الطلب المتوقع على هذا التطبيق سيكون كبير، نظراً لحاجة المستخدمين إلى بدائل منخفضة التكلفة، خاصة في ظل الأوضاع الاقتصادية الحالية.

3) تحليل فجوة السوق:

* حجم الطلب أكبر من الخدمة المقدمة.

4) تحليل القطاع:

* المنافسين:

بعض الصفحات على الفيسبوك ومجموعات البيع، بالإضافة إلى تطبيقات بسيطة محدودة(مثل أسواق).

* قدرة الموردين/المشترين على المساومة:

متوسطة؛ لأن الأسعار تُحدد باتفاق الطرفين، والتطبيق يعمل كوسيط تقني فقط.

1. تكاليف المشروع:

العناصر الثابتة (تجهيزات – أساسية):

* حاسوب محمول أو مكتبي
* استضافة التطبيق (Server أو Cloud)
* تصميم وبرمجة التطبيق
* اشتراك إنترنت

المتغيرة (اليد العاملة من أجور ورواتب):

* مصمم واجهات
* مطور تطبيقات (Android)
* دعم فني أو مشرف على المنصة
* حملات تسويقية دورية
* تكاليف قانونية (ترخيص، شروط استخدام، خصوصية)
  1. تحليل الربحية

– العائد على الاستثمار (ROI): مرتفع إذا تم إطلاق التطبيق بجودة وتسويق جيد

– فترة استرداد رأس المال: خلال 5 إلى 6 أشهر من التشغيل المستقر

* 1. تحديد الإيرادات المتوقعة:
* عمولات على عمليات البيع (فرض رسوم بعد عدد من عمليات عرض المنتجات)

# الدراسة المرجعية

تهدف هذه الدراسة المرجعية إلى استعراض أبرز الدراسات والتقنيات المستخدمة في بناء أنظمة البيع والشراء بين المستخدمين (C2C)، مع تسليط الضوء على التجارب السابقة في البنية المعمارية، إدارة الرسائل بين المكونات، واستخدام الذكاء الاصطناعي في تحليل المحتوى التفاعلي.

## الأنماط المعمارية في الأنظمة C2C: Modular Monolith vs. Microservices

أشارت دراسة لـ Richardson (2018) إلى أن اعتماد الأنظمة على نمط Microservices منذ البداية قد يُسبب تعقيدات تشغيلية كبيرة، خصوصًا في المشاريع الناشئة التي لا تحتاج إلى تفكيك البنية مبكرًا. بالمقابل، يُعد نمط Modular Monolith حلاً وسطياً ناجحاً، حيث يمكن فصل الوظائف داخلياً على شكل وحدات مستقلة (Modules)، مما يُسهّل لاحقاً عملية التحول إلى Microservices دون الحاجة لإعادة بناء النظام بالكامل.

كما بيّن Milanović (2022) أن Modular Monolith مناسب للمشاريع التي تحتاج إلى قابلية نمو تدريجية دون التضحية بهيكلية واضحة، إذ يمكن بناء كل وحدة باستخدام أسلوب معماري يناسبها (مثل Clean Architecture)، مع الحفاظ على وحدة النشر والإدارة.

## وسطاء الرسائل (Message Brokers)

تشير الدراسات إلى أن أنظمة وسطاء الرسائل تمثّل حجر الزاوية في تصميم بنى برمجية مرنة وقابلة للتوسع، إذ دمجت في بداياتها في نماذج التكامل المؤسسي كما قدمها Hohpe & Woolf (2003)، الذين أوضحوا كيف تسهم آليات قوائم الانتظار ونمط النشر/الاشتراك في فصل المرسل عن المستقبل وتقليل الترابط المباشر. وفي سياق معالجة تدفقات البيانات الضخمة، برزت بنية Apache Kafka التي وصفها Kreps et al. (2011) كوسيط يحقق موثوقيةً عاليةً وعُمرًا طويلًا للبيانات مع معدل معالجة مرتفع، مما أتاح استخدامها على نطاق واسع في تطبيقات تحليل السجلات في الزمن الحقيقي. ومع انتشار هندسة الخدمات المصغرة، أكد Newman (2015) على دور وسطاء الرسائل في تعزيز القابلية على التوسع وعزل الأخطاء fault isolation عبر تمكين كل خدمة من العمل بشكل مستقل مع ضمان تبادل الرسائل بموثوقيةٍ. وقد أظهرت دراسات مثل Balalaie et al. (2018) أنّ إدخال وسطاء الرسائل أثناء ترحيل الأنظمة التقليدية إلى بنية مصغرة يخفّض من زمن التعطيل ويُسهل دمج الخدمات الجديدة دون إعادة هيكلة شاملة. تجمع هذه التجارب على أهمية وسطاء الرسائل في تحقيق التواصل غير المتزامن، وفصل المسؤوليات، وضمان مرونةٍ عاليةٍ في صيانة وتوسيع الأنظمة.

.......................................

Richardson, C. (2018). *Microservices Patterns: With examples in Java*. Manning Publications.

1. Milanović, M. (2022). *Modular Monoliths in Practice*. Techworld Newsletter.
2. Pautasso, C., Zimmermann, O., & Leymann, F. (2020). Microservices in Practice: A Survey and Classification of Architectures and Techniques. IEEE Software.
3. Fowler, M. (2019). Event-Driven Architectures and Messaging. martinfowler.com.
4. Wittern, E., Suter, P., & Rajagopalan, S. (2017). *A Look at the Impact of Messaging in Microservices*. IBM Research.
5. Zhang, Y., Wang, S., & Liu, B. (2020). *Deep Learning for Sentiment Analysis: A Survey*. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery.
6. Liu, B. (2022). *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers.
7. Zhou, L., Zhang, X., & Wang, M. (2021). *Integrating AI Models into Distributed Applications via Message Brokers*. Proceedings of the ACM Symposium on Cloud Computing.
8. Ali, R., Khan, M., & Javed, A. (2022). *Decoupling AI Services Using Message Queues in Scalable Web Applications*. Journal of Software Engineering and Applications.

## الذكاء الاصطناعي وتحليل المشاعر في تطبيقات المستخدم النهائي

تعتمد التجارة الإلكترونية على مراجعات العملاء التي يضعونها حول مزوِّدي الخدمة بعد استلام الطلب، وقد تحتوي هذه المراجعات على فجوة بين مضمون النص والتقييم الرقمي بالنجوم.

لذلك، استُخدمت تقنيات الذكاء الاصطناعي المختلفة لتحليل محتوى المراجعات النصية والتنبؤ بتقييم النجوم (من 1 إلى 5).

### النماذج المعتمدة على المحولات Transformers:

#### **‏** النسخة الأساسية من BERT للغة العربية AraBERT

هو نموذج Transformer مُدرب مسبقاً يتكون من 12 طبقة وخفية بحجم 768، تم تدريبه على أخبار باللغة العربية الفصحى الحديثة (MSA) بحجم بيانات يقارب 2.7 مليار كلمة . يحتوي على حوالي 110 إلى 136 مليون معلمة في نسخته الأساسية.  
يتفوق AraBERT في التعامل مع النصوص الرسمية باللغة العربية، ويتوفر على نطاق واسع (مثل: نموذج aubmindlab/bert-base-arabertv02 على HuggingFace). وغالبًا ما يتفوق على النماذج متعددة اللغات في مهام تحليل المشاعر باللغة العربية .

الإيجابيات:

* دقة عالية في اللغة العربية الفصحى.
* توفر العديد من نقاط التحقق المدربة مسبقًا

السلبيات:

* حجمه كبير (مئات الميغابايت).
* أبطأ في الأداء على المعالجات (CPU).
* دعمه المباشر للهجات العامية محدود.

#### **‏** النسخة الأساسية من BERT للهجات العربية MARBERT

يتكون أيضاً من 12 طبقة وخفية بحجم 768، ويحتوي على حوالي 160 مليون معلمة. تم تدريبه على 128 غيغابايت من النصوص من تويتر والنصوص باللهجات (حوالي 15.6 مليار كلمة).

تم تصميم MARBERT خصيصاً للتعامل مع اللهجات العربية المتعددة، وغالباً ما يحقق أعلى دقة في مهام تحليل المشاعر على وسائل التواصل الاجتماعي والمراجعات، حيث تفوق في عدة اختبارات مرجعية على AraBERT

الإيجابيات:

* أداء قوي مع النصوص المختلطة/العامية، وهو شائع في تعليقات المستخدمين ومراجعاتهم.
* مفتوح المصدر (مثل: نموذج UBC-NLP/MARBERT).

السلبيات:

* لا يزال حجمه كبيراً (~160 مليون معلمة)، أي بحجم مماثل لـ AraBERT.
* أبطأ في التنفيذ.
* نظراً لأنه تم تدريبه على تغريدات تويتر، فقد تكون تغطيته للغة العربية الفصحى الرسمية أو المصطلحات التخصصية محدودة.

### النموذج الهجين CNN–LSTM

هو شبكة عميقة خفيفة الوزن تتكوّن عادةً من تمثيلات كلمات (word embeddings) + طبقات التفاف (Convolution) + طبقات LSTM، ويتم تدريبها من البداية إلى النهاية باستخدام بيانات المراجعات.  
تشير دراسات قديمة إلى أن هذا النموذج حقق دقة في أوائل التسعينات بالمئة على مجموعات بيانات تحليل المشاعر باللغة العربية.  
فعلى سبيل المثال، نموذج CNN–LSTM باستخدام تمثيلات fastText حقق دقة حوالي 92.8% في إحدى الدراسات.

لكن المقارنات الحديثة تُظهر أن نماذج Transformer غالباً ما تتفوق على CNN/LSTM بفارق عدة نقاط، مثل: 91% لـ CNN مقابل 92–97% لنماذج مبنية على AraBERT.

الإيجابيات:

* أصغر بكثير (حجمه في حدود عشرات الملايين من المعاملات فقط).
* تنفيذ سريع على المعالج CPU أو بطاقة الرسوميات GPU.
* سهل النشر، ويمكن تصديره إلى ONNX أو TFLite باستخدام أطر مثل TensorFlow.

### SVM + TF-IDF

هو خط معالجة تقليدي في التعلم الآلي، حيث يتم حساب ميزات TF-IDF أو n-gram ثم تدريب نموذج SVM خطي أو الانحدار اللوجستي (Logistic regression).  
في مهام تحليل المشاعر باللغة العربية، غالباً ما تحقق النماذج الأساسية المعتمدة على SVM دقة في حدود أواخر الثمانينات بالمئة في التصنيفات الثنائية.  
على سبيل المثال، في مجموعة مراجعات من أمازون، حقق نموذج SVM دقة تبلغ حوالي 87.6% مقارنة بـ 93.0% لنموذج AraBERT .

الإيجابيات:

* بسيط جدًا وسريع جداً في التنفيذ (فقط عمليات ضرب مصفوفات).
* سهل التدريب على المعالج (CPU) دون الحاجة إلى عتاد متقدم.

السلبيات:

* يتطلب معالجة لغوية دقيقة للنص، مثل التجذير والتطبيع للغة العربية.
* دقته أقل بكثير مقارنة بالنماذج العميقة مثل Transformers.
* ضعيف في التعامل مع النفي، والتعابير الاصطلاحية، والتبديل اللغوي (code-switching).

### نماذج اللغة الضخمة (LLMs)

مؤخراً، ظهرت حلول مؤسسية تستخدم نماذج لغوية ضخمة (مثل GPT-3 وGoogle Gemini) لمعالجة المراجعات النصية وتوقع تقييم النجوم باستخدام التوجيه (Prompting) فقط. يمكن لهذه النماذج، بفضل قدرتها على فهم السياق والتعامل مع التعابير المعقدة، استنتاج درجة الرضا بدون الحاجة إلى إدخال رقمي مباشر من المستخدم. على سبيل المثال، GPT-3 (بني على 175 مليار مُعامل) حقق أداءً قويّاً في مهام لغوية متنوعة في وضع التعلم قليل اللقطات (few-shot)، مما يشير إلى أنه يمكن تكييفه لإعطاء تقييم نجمي من مراجعة نصية مع أمثلة توجيهية بسيطة. وأعادت Google Gemini صياغة معايير الأداء حيث تفوقت على البشر في العديد من اختبارات الفهم والتمثيل النصي.

الإيجابيات:

النماذج الكبيرة تستفيد من التدريب على كمّ هائل من البيانات العامة، ما يمكنها من التقاط دلالات دقيقة (كالنفي والسخرية) لا تدركها النماذج التقليدية بسهولة. كما أنها مرنة وغير معتمدة على بنية بيانات تدريب تقليدية (فيمكن استخدام «التوجيه» مع نماذج مدرّبة مسبقاً).

السلبيات:

لا تضمن دائماً استقراراً في النتائج (قد تعتمد على صياغة السؤال promptبدقة). كما تظل مسألة تحيز النموذج ومسؤولية التقييم نُقاشاً قائماً. وحتى الآن، لم تتوفر دراسات منشورة كثيرة عن دقة هذه النماذج في التقييم النجمي المباشر، لكن التجارب الأولية تشير إلى وعود قوية في تحسين فهم المشاعر النصية المعقدة.

## النماذج الرئيسية في التجارة الإلكترونية

تشمل التجارة الإلكترونية عدة نماذج تفاعلية بين الأطراف، تختلف باختلاف اتجاه المعاملة والجهات المتفاعلة. ومن أبرز هذه النماذج: C2C، B2C، B2B، C2B، ولكل منها خصائص ومزايا وتحديات خاصة.

أولاً: نموذج العميل إلى العميل (C2C)

يُعد نموذج C2C من أبرز النماذج التشاركية، حيث يسمح للأفراد ببيع وشراء المنتجات والخدمات فيما بينهم عبر منصات وسيطة مثل eBay وFacebook Marketplace. يعتمد نجاح هذه المنصات على تحقيق توازن بين المرونة والموثوقية.

العوامل المؤثرة في فعالية هذا النموذج:

* تجربة المستخدم والمحتوى: سهولة استخدام الموقع ووضوح المعلومات يعززان من رضا العملاء.
* الأمان والموثوقية: غياب جهة مركزية يجعل الثقة مبنية على آليات تقييم المستخدمين، التبليغ عن الاحتيال، وحماية البيانات.
* تنوع المنتجات وسرعة الوصول: تتنوع العروض نتيجة لاختلاف مصادر السلع، مما يعزز من التنافسية والقيمة المقدّمة.

من أبرز المنصات السورية: أسواق، السوق المفتوح.

تم اختيار نموذج C2C في هذا المشروع للأسباب التالية:

* ملاءمته للسوق السوري: معظم عمليات البيع والشراء في سوريا تتم بين الأفراد مباشرة، خاصة في سوق السلع المستعملة.
* النموذج يعكس الواقع العملي للسوق المحلي.
* تكلفة تشغيل منخفضة: لا يتطلب وجود مخزون أو شركة وسيطة، مما يسهّل الانطلاق بموارد محدودة.
* مرونة عالية وتوسّع ذاتي: كل مستخدم يمكن أن يكون بائعاً ومشترياً في ذات الوقت، ما يزيد من ديناميكية المنصة.
* التمكين الفردي: يُعطي للمستخدم العادي القدرة على عرض سلعه أو خدماته بسهولة دون الحاجة لامتلاك سجل تجاري أو بنية تحتية كبيرة.

Rauniar, Rupak, et al. "C2C online auction website performance: Buyer's perspective." *Journal of Electronic Commerce Research* 10.2 (2009): 56.

ثانياً: نموذج الشركة إلى العميل (B2C)

نموذج B2C هو الشكل التقليدي للبيع عبر الإنترنت، حيث تُقدم الشركات منتجاتها أو خدماتها مباشرة إلى المستهلكين عبر منصات إلكترونية. من أبرز المنصات: Amazon وWalmart.

أبرز خصائص النموذج:

* بناء الثقة المؤسسية: تعتمد الثقة هنا على وجود نظام ضمانات مثل تقييمات المستخدمين، سياسة الاسترجاع، ووسائل الدفع الآمنة (مثل PayPal).
* أهمية الطرف الوسيط: تلعب المنصة الوسيطة دوراً مهماً في تعزيز الثقة بين البائع والمشتري.
* تقليل المخاطر: وجود بنية قانونية ومؤسسية متكاملة يقلل من التردد الشرائي ويزيد من ولاء المستهلك.

Pavlou, Paul A., and David Gefen. "Building effective online marketplaces with institution-based trust." *Information systems research* 15.1 (2004): 37-59.

ثالثاً: نموذج الشركة إلى الشركة (B2B)

في هذا النموذج، تتم المعاملات بين كيانات تجارية، ويُستخدم بكثرة في سلاسل التوريد والمصانع ومزودي الخدمات اللوجستية. من أشهر المنصات: Alibaba.

خصائص النموذج:

* حجم المعاملات الكبير والتفاوض: تتميز المعاملات بأنها ذات قيمة عالية وتتطلب اتفاقات مخصصة.
* علاقات طويلة الأمد: تبنى العلاقات على الثقة المتبادلة وعقود متعددة السنوات.
* الدعم اللوجستي والتقني: يتطلب استثماراً في البنية التحتية الرقمية لضمان التكامل بين أقسام المؤسسة.

Pavlou, Paul A., and David Gefen. "Building effective online marketplaces with institution-based trust." *Information systems research* 15.1 (2004): 37-59.

رابعاً: نموذج العميل إلى الشركة (C2B)

يُعد هذا النموذج من النماذج الحديثة والمرنة في عالم التجارة الإلكترونية، ويقوم على عكس الاتجاه التقليدي للتبادل التجاري، حيث يبدأ المستهلك العملية التجارية من خلال عرض خدمة أو طلب منتج معين، وتقوم الشركات بالاستجابة لهذا الطلب. ينتشر هذا النموذج بشكل خاص في مجالات العمل الحر (Freelancing)، التسويق بالمحتوى، واستطلاعات الرأي المدفوعة، حيث يقدّم الأفراد خبراتهم أو أفكارهم للشركات مقابل تعويض مالي.

أبرز خصائص النموذج:

* الاستهلاك قبل الإنتاج: الركيزة الأساسية في هذا النموذج هي أن المبادرة تبدأ من المستهلك وليس من الشركة، بمعنى آخر، لا تقوم الشركة بإنتاج منتج أو خدمة مسبقاً، بل تنتظر طلباً أو حاجة محددة من المستخدم، ثم تبدأ بالتصنيع أو التخصيص بناءً على ذلك، وهذا يختلف جوهرياً عن النموذج التقليدي (B2C) الذي فيه تقوم الشركة أولاً بتقديم المنتج، ثم ينتظر أن يقوم العميل بشرائه.
* تحقيق الكفاءة في الموارد: لا يتم الإنتاج إلا عند وجود طلب فعلي، مما يقلل من الهدر في المخزون والتكاليف التشغيلية.

من أبرز الأمثلة: مواقع مشاركة المحتوى مثل YouTube، حيث ينشئ المستخدم المحتوى وتقوم الشركات بالإعلان من خلاله

Wu, Qinglie, Jing Ma, and Zhong Wu. Consumer-Driven E-commerce: a Study on C2B applications. 2020 International Conference on E-Commerce and Internet Technology (ECIT). IEEE, 2020.

فيما يلي جدول يلخص أهم المفاهيم ومقارنة حول هذه النماذج

| النموذج | التعريف | الجمهور المستهدف | المزايا | التحديات | أمثلة |
| --- | --- | --- | --- | --- | --- |
| C2C | تبادل السلع والخدمات بين الأفراد عبر منصة وسيطة توفر آليات للتقييم والثقة | الأفراد الراغبون في البيع والشراء بشكل مباشر | تكلفة تشغيل منخفضة  مرونة عالية  تنوع العروض | قضايا الثقة مخاطر الاحتيال صعوبة مراقبة الجودة | أسواق |
| B2C | بيع المنتجات والخدمات من الشركات مباشرة للمستهلكين | المستهلك النهائي | تحكم أكبر في جودة المنتجات  دعم قوي لخدمات ما بعد البيع  حملات تسويقية متطورة | منافسة شديدة  استثمارات تسويقية كبيرة | Amazon Walmart |
| B2B | إجراء معاملات تجارية بين الشركات والمؤسسات | الشركات والكيانات التجارية | صفقات ذات قيمة كبيرة  علاقات تجارية طويلة الأمد  عمليات تفاوض مباشرة | - دورات مبيعات طويلة - إجراءات إدارية معقدة - متطلبات تقنية عالية | Alibaba |
| C2B | تقديم الأفراد خدماتهم أو منتجاتهم للشركات | الأفراد الراغبون في عرض خبراتهم أو منتجاتهم | حلول مبتكرة للشركات  تنوع العروض والمنتجات | تنظيم ضعيف  اختلاف توقعات الجودة | منصات العمل الحر مثل Upwork |

## تطبيقات تقنية البلوك تشين في التجارة الإلكترونية

مع الازدهار المتزايد للمعاملات الإلكترونية، قامت العديد من منصات التجارة الإلكترونية باعتماد تقنية السجلات الموزعة اللامركزية (البلوكتشين) للوصول إلى المستهلكين وضمان الكفاءة بأقل التكاليف الممكنة. وفيما يلي بعض التطبيقات والمشارع الحالية المستخدمة في التجارة الإلكترونية والمدعومة بتقنية البلوكتشين:

نظام إدارة سلاسل التوريد

تُعد الشحنات الدولية خصوصاً تلك التي تشمل بضائع مبردة من التحديات الكبرى أمام شركات الشحن العالمية. فالعملية تتطلب سلسلة طويلة من الإجراءات الورقية مثل الأختام والموافقات، ما يرفع التكاليف بشكل كبير.

استخدام تقنية البلوك تشين يلغي العديد من أوجه القصور في النظام التقليدي ويُحوّل الوثائق الورقية إلى سجلات رقمية يمكن مشاركتها بأمان.

على سبيل المثال، تستخدم شركة Maersk، وهي أكبر شركة شحن في العالم، بالتعاون مع IBM، تقنية البلوكتشين تحت اسم TradeLens لإنشاء سجل تدقيق كامل يتيح تبادل المعلومات بأمان بين مختلف الجهات: الناقلين، الموانئ، الجمارك... الخ، مما يسمح لجميع المشاركين برؤية موحدة ومشتركة.

يُظهر البحث الأصلي تصوّراً لأداء يصل إلى أكثر من 3500 معاملة في الثانية، بزمن استجابة تحت الثانية الواحدة، حتى مع وجود 100 عقدة

Lim, Yi Han, et al. "Blockchain technologies in e-commerce: Social shopping and loyalty program applications." *International Conference on Human-Computer Interaction*. Cham: Springer International Publishing, 2019.

نظام تتبع النتجات

يمثّل اعتماد شركة وولمارت لمنصّة Hyperledger Fabric مثالاً عمليّاً على توظيف البلوك تشين في التجارة الإلكترونيّة لتعزيز الشفافيّة والثقة في سلاسل الإمداد الغذائي. فقد كشفت الشركة أنّ تعقّب مصدر المنتجات الطازجة كان يستغرق أيّاماً، ما يُعقّد سحب الدفعات الملوَّثة ويقلل ثقة المستهلكين. بعد دراسة عدّة تقنيّات، وقع الاختيار على Fabric لكونه إطاراً مفتوح المصدر ذا طابع مؤسّسي مُصرَّح، يسمح بتركيب مكوّنات الإجماع وإدارة العضويّة بصورة مرنة، وهو ما لبّى حاجة الشركة إلى إشراك مورّدين ومنافسين ضمن شبكة واحدة آمنة.

أطلقت وولمارت في أكتوبر 2016 مشروعين تجريبيّين لتتبّع منشأ المانجو في متاجر الولايات المتّحدة ولحم الخنزير في متاجر الصين، بالاشتراك مع IBM. أظهرت التجارب أنّ زمن تتبّع المانجو انخفض من سبعة أيّام إلى 2.2 ثانية فقط، بينما أتاح النظام رفع شهادات الأصالة للحوم الخنزير إلى السلسلة، ما عزّز الثقة في الأسواق التي تعاني من ممارسات تقليد الشهادات.

بفضل هذا النجاح، وسّعت الشركة نطاق التطبيق فصارت تتتبّع أكثر من 25 منتجاً ينتمي إلى خمسة مورّدين، تشمل الفواكه، والخضروات، واللحوم، ومشتقّات الألبان، ومنتجات متعددة المكوّنات، مع خطط لإلزام جميع مورّدي الخضروات الطازجة بالانضمام إلى النظام في المستقبل القريب.

Pham, Hanh. "The impact of Blockchain Technology on the improvement of Food Supply Chain Management: Transparency and Traceability: A case study of Walmart and Atria." (2018).

اختيارا الخرائط

خرائط Google:

نبدأ بـخرائط Google بوصفها المعيار الأشهر لسنوات طويلة؛ فهي تقدّم تغطية عالمية عالية الدقة لطبقات الخرائط والصور الجوية والمرور، وتوفّر حزمة SDKs للهواتف والويب تجعل التكامل سريعاً وموثوقاً. قوة Google تكمن في اتساع منظومتها: التوجيه وحساب المسافات، الترميز الجغرافي والبحث عن الأماكن، وقاعدة بيانات نقاط اهتمام غنية تُحدَّث باستمرار، وهو ما يفسّر اعتماد تطبيقات كبرى عليها؛ ومع ذلك فاختيار خرائط Google اليوم يصطدم بقيد عملي وهو لا يمكننا الاستفادة المجانية دون تفعيل الفوترة وربط بطاقة دفع، وهو سبب رئيسي لاستبعادها مرحلياً إلى أن تتوافر وسيلة دفع وميزانية ثابتة من الأمثلة على التطبيقات التي تستخدم خرائط Google هو تطبيق Yalla go السوري المشهور من أجل خدمة نقل الركاب حيث يختار المستخدم نقطة الانطلاق والوجهة على خريطة تفاعلية، ثم يرسل طلباً يُطابَق تلقائياً مع أقرب سائق متاح وفقاً للمسافة وزمن الوصول المتوقع. يعرض التطبيق المسار المقترح على الخريطة، والوقت والتكلفة التقديريَّيْن قبل التأكيد، ويوفّر تحديثات لحظية لحركة المركبة وإشعارات بالحالة (قبول السائق، الوصول، بدء الرحلة، الإنهاء).

Mapbox

أمّا Mapbox فيقدّم طريقـة مختلفة تركّز على تخصيص شكل الخريطة بما يناسب هوية التطبيق. يمكننا بسهولة تغيير الألوان والرموز وطريقة العرض، وبناء تفاعل بصري، ولهذا تستخدمه تطبيقات معروفة تهتم بشكل الخريطة وتجربتها مثل Snapchat وBooking.com في أجزاء من حلولها.

لكن لهذه المزايا ثمنٌ عملي: ستعتمد على مزود واحد لخدمات الخريطة، وتدفع حسب حجم الاستخدام ما يستلزم متابعة الاستهلاك بدقة لكن الثمن الأكبرهو أنّنا نحتاج إلى وسيلة دفع مفعّلة منذ البداية حتى لو كنت ضمن الحصص المجانيّة.

OpenStreetMap

أمّا OpenStreetMap (OSM) فهو مشروع خرائط مفتوح المصدر يوفّر بيانات مجانية يمكن استخدامها في التطبيقات دون اشتراك أو بطاقة دفع. فكرته بسيطة: هناك قاعدة بيانات عالمية للمناطق والطرق والأماكن، ونحن نختار كيف نعرض هذه الخريطة داخل التطبيق، وكيف نبحث عن العناوين، وكيف نحصل على تفاصيل موقع محدد.

قيمة OSM العملية تكمن في التحكّم الكامل في التكلفة والشكل. أثناء التطوير لن تظهر فواتير مفاجئة، ويمكن تعديل ألوان الخريطة والأيقونات لتناسب هوية تطبيقنا.

في مرحلتنا الحالية مع غياب بطاقة دفع يأتي اختيار OpenStreetMap بوصفه الأنسب عملياً. مع إمكانية إعادة استخدام خرائط Google أو Mapbox لاحقاً عندما تتطلب الاحتياجات ميزات مُدارة جاهزة (مثل البحث عن المتاجر القريبة، أو مشاهدة صور الطريق على مستوى الشارع) ومعها ميزانية وبطاقة دفع، بينما يمنحنا OSM الآن منصة مرنة قابلة للنمو دون عوائق دخول مالية أو قيود ترخيص صارمة.

# الفصل الثالث الدراسة التحليلية

## مخطط حالات الاستخدام:

الفاعلين:

المسؤول (Admin) يدير المستخدمين، والفئات، ويستطيع تسجيل الدخول إلى حسابه إلى لوحة الإدارة.

المستخدم(User) يستطيع إنشاء حساب وتسجيل الدخول إليه وتعديل معلوماته الشخصية( بما فيها العناويين وملفه الشخصي)، كما يستطيع تصفح العروض(منتجات وخدمات) وإضافتها إلى المفضلة أو السلة الخاصة به، وإضافة عروض، كما يستطيع تعديل عروضه أو حذفها، بالإضافة إلى قدرته على الاطلاع على العروض التي أضافها إلى سلته وتعديل كميتها أو حذفها من السلة، ثم يمكنه طلب محتويات سلته، يحتاج حينها لتحديد عنوان التوصيل الخاص به من عناوينه المضافة، مع إمكانية إلغاء الطلب، ولكن عند إتمام الطلب يجب أن يحدد طريقة الدفع التي يرغب باستخدامها، وكذلك يستطيع تصفح الطلبات الخاصة به ورؤية حالتها، وعند استلام الطلب يقوم بتحديث حالة الطلب الى "تم التوصيل" ويقوم بعدها بإعطاء مراجعة تفيد في تقييم المزود الذي طلب منه الطلب.

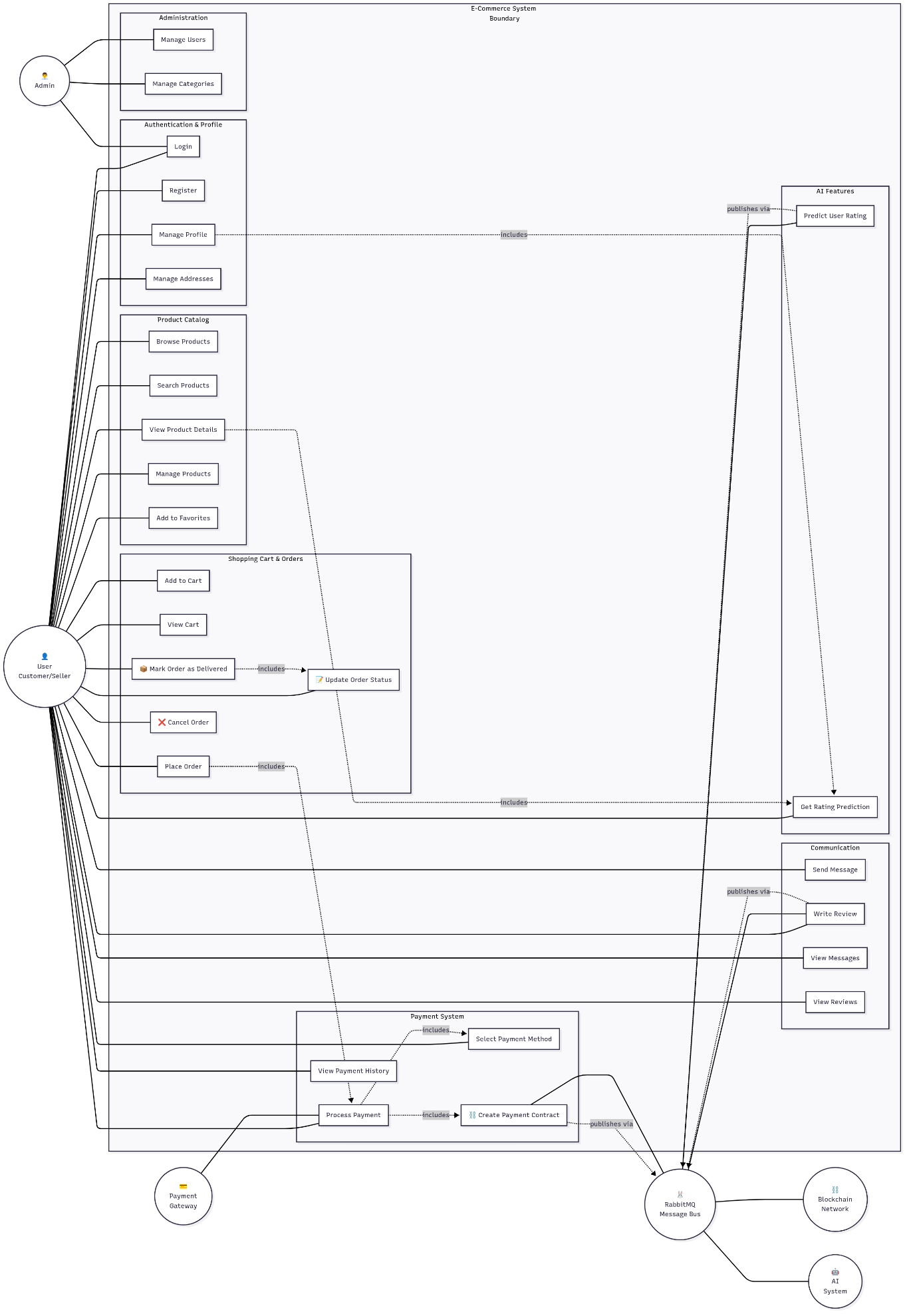
بوابة الدفع (Payment Gateway) لإتمام الدفع عن طريقها.

RabbitMQ لإيصال البيانات للجهة المعنية بمعالجة هذه البيانات (AI and Blockchain)

AI Model لتقييم المزود بالاعتماد على المراجعات المقدمة عنه.

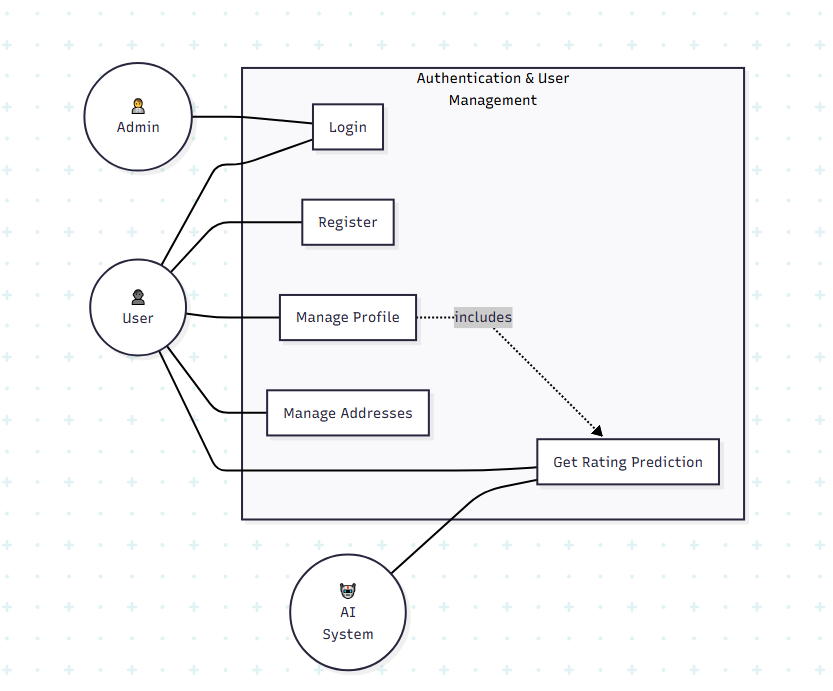
Blockchain لكتابة عقود البيع والشراء وضمان المصداقية.

فيما يلي مخطط حالات الاستخدام العام في النظام يوضح ما يلي:

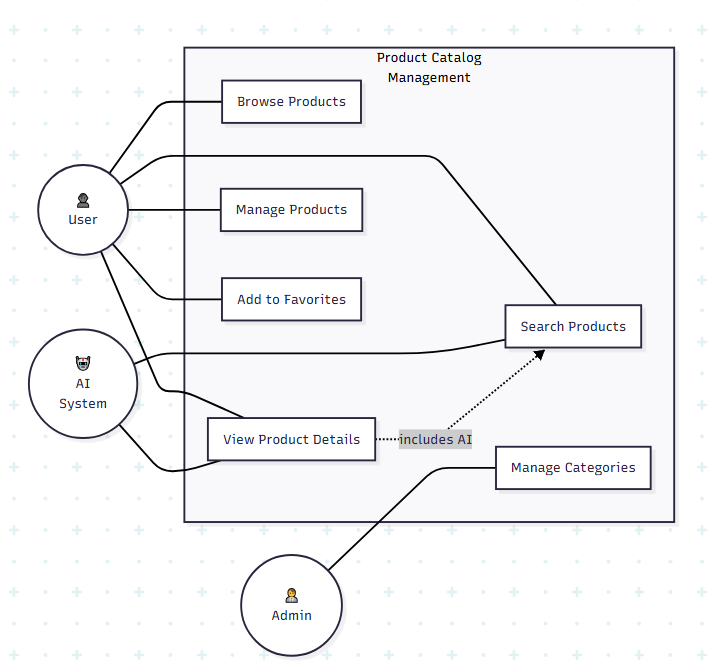


فيما يلي صور أوضح لمخطط حالات الاستخدام لكل جزء على حدى:

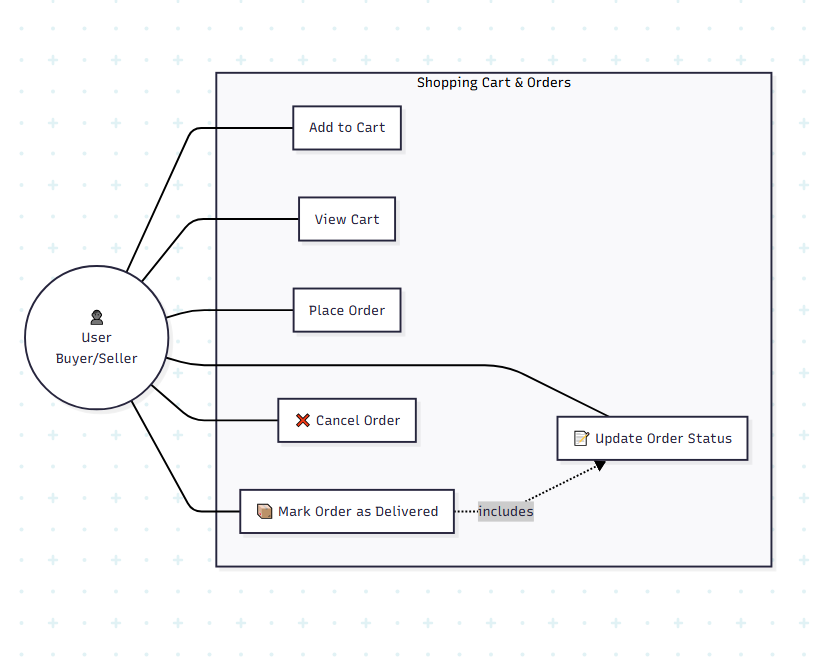
### مخطط حالات استخدام ال Authentication & User Management



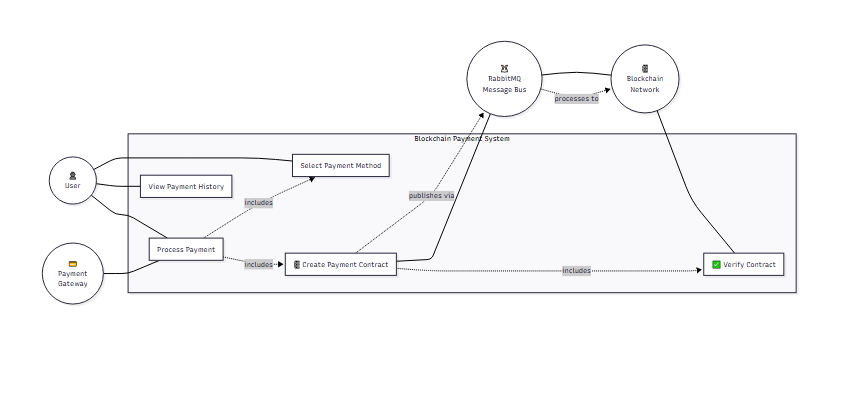
### مخطط حالات استخدام Product Catalog Management



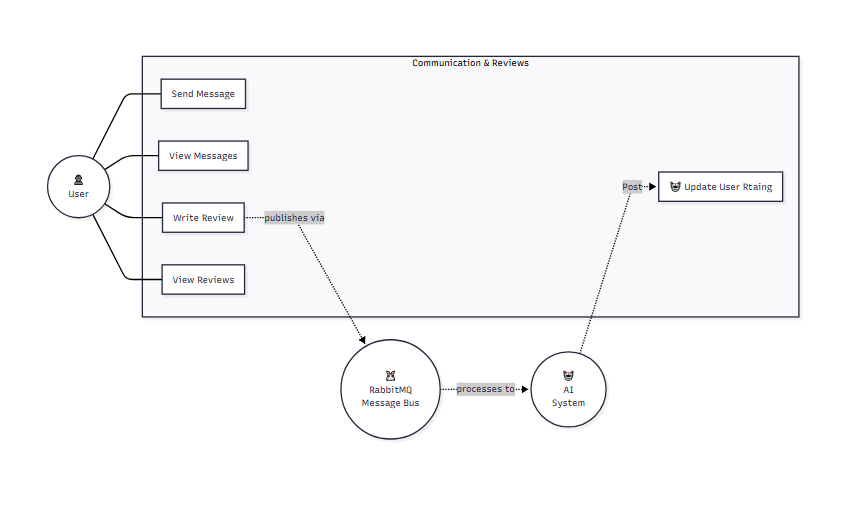
### مخطط حالات استخدام Shopping Cart & Orders



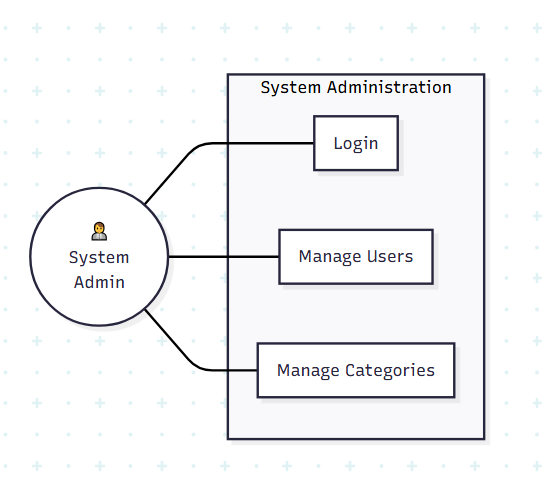
### مخطط حالات استخدام Payment System



### مخطط حالات استخدام Communication System



### مخطط حالات استخدام System Administration



## السرد النصي لحالات الاستخدام:

حالة الاستخدام 1: كتابة مراجعة

* 1. معلومات حالة الاستخدام
* **اسم الحالة:** كتابة مراجعة
* الفاعل الرئيسي: المستخدم
* **الهدف:** تقديم مراجعة لعملية شراء مكتملة مع توقع تقييم مدعوم بالذكاء الاصطناعي
* **النطاق:** وحدة التواصل.

الشروط المسبقة

1. يكون المستخدم مسجَّل دخول إلى النظام.
2. استلم المستخدم المنتج المطلوب.
3. تكون حالة الطلب "تم التسليم (Delivered)".

شروط النجاح (الشروط اللاحقة)

1. تُحفظ المراجعة بنجاح في قاعدة البيانات.
2. تُنشر المراجعة بشكل غير متزامن عبر RabbitMQ.
3. يعالج نظام الذكاء الاصطناعي المراجعة لتحليل المشاعر.
4. يؤثّر تقييم المستخدم على التقييم العام للبائع.
5. تظهر المراجعة ضمن قائمة المنتج.

السيناريو الأساسي الناجح

المرحلة 1: بدء المراجعة

1. ينتقل المستخدم إلى قسم “طلباتي”.
2. يعرض النظام قائمة الطلبات المكتملة.
3. يختار المستخدم “تم التوصيل” للطلب الذي تم استلامه.
4. تتغير حالة الطلب إلى تم التسليم "Delivered".
5. يعرض النظام للمستخدم واجهة أسئلة للإجابة عليها لتقييم رضى المستخدم عن الطلب وتبعياته.

المرحلة 2: كتابة المراجعة

1. يجيب المستخدم عن الأسئلة المطروحة.

المرحلة 3: إرسال المراجعة

1. يؤكد المستخدم إرسال المراجعة.
2. يتحقق النظام من صحة جميع بيانات المراجعة.
3. يحفظ النظام المراجعة في قاعدة البيانات.
4. ينشر النظام المراجعة على ناقل الرسائل RabbitMQ.
5. يعرض النظام رسالة نجاح للمستخدم.
6. يمرر RabbitMQ المراجعة للمعالجة غير المتزامنة.

المرحلة 4: المعالجة الخلفية

1. تعالج خدمة خلفية المراجعة عن طريق RabbitMQ.
2. يقوم الذكاء الاصطناعي بإعطاء تقييم بناءً على المراجعة.
3. يحدث النظام التقييم الكلي للبائع.
4. تصبح المراجعة مرئية في قوائم المنتجات.

التدفقات البديلة

* E1: فشل التحقق من محتوى المراجعة (الخطوة 8)
  1. يكتشف النظام محتوى غير مناسب أو احتيالي.
  2. يميّز النظام الأجزاء المشكلة للنص.
  3. يعرض النظام إرشادات لتصحيح المحتوى.
  4. يعدّل المستخدم المحتوى.
  5. يعود التنفيذ إلى الخطوة 8.
* E2: خدمة التنبؤ بالذكاء الاصطناعي غير متاحة (الخطوة 14)
  1. تفشل دعوة خدمة الذكاء الاصطناعي.
  2. يحتفظ العامل الخاص بالذكاء الاصطناعي بالبيانات في رتل حتى تتوفر الخدمة من جديد ويعيد ارسالها للمعالجة.
  3. يعود التنفيذ إلى الخطوة 12.
* E3: فشل RabbitMQ (الخطوتان 10–11)
  1. يفشل النظام في الاتصال بـ RabbitMQ.
  2. يحفظ النظام المراجعة محلياً مؤقتاً.
  3. يجدول النظام إعادة محاولة نشر المراجعة.
  4. يؤكد النظام للمستخدم أن المراجعة محفوظة.
  5. تعود خدمة الخلفية لإعادة المحاولة لاحقاً.

حالة الاستخدام 2: معالجة الدفع

معلومات حالة الاستخدام

* **اسم الحالة:** معالجة الدفع
* الفاعل الرئيسي: المستخدم
* **الهدف:** إتمام معاملة الدفع مع تأمين العقد الذكي على البلوكشين
* **النطاق:** وحدة المدفوعات

الشروط المسبقة

1. يكون المستخدم مسجَّل دخول إلى النظام.
2. توجد عناصر في عربة التسوق.
3. يمتلك المستخدم عنوان شحن صالحاً.
4. تكون بوابة الدفع عاملة.
5. تكون شبكة البلوكشين متاحة.

شروط النجاح (الشروط اللاحقة)

1. تُعالج عملية الدفع بنجاح.
2. تُنشأ وتُؤكَّد الطلبية.
3. يُنشر العقد الذكي على البلوكشين.
4. يتلقى المستخدم تأكيد الدفع.
5. يُحدَّث المخزون.

السيناريو الأساسي الناجح

المرحلة 1: بدء الدفع

1. ينتقل المستخدم إلى صفحة السلة.
2. يعرض النظام ملخص الطلب مع المبلغ الكلي.
3. يراجع المستخدم تفاصيل الطلب ويؤكدها.
4. يعرض النظام طرق الدفع المتاحة.
5. يختار المستخدم طريقة الدفع المفضلة.

المرحلة 2: إدخال بيانات الدفع

1. يعرض النظام نموذج الدفع للطريقة المختارة.
2. يدخل المستخدم بيانات الدفع.
3. يتحقق النظام من صحة تنسيق البيانات.
4. يُشفّر النظام بيانات الدفع.
5. يراجع المستخدم ملخص الدفع النهائي.

المرحلة 3: معالجة الدفع

1. يؤكد المستخدم عملية الدفع.
2. يبدأ النظام إجراء الدفع عبر بوابة الدفع.
3. تعالج بوابة الدفع المعاملة.
4. تُرجع بوابة الدفع حالة المعاملة.
5. يتحقق النظام من نجاح الدفع.

المرحلة 4: إنشاء العقد الذكي

1. يطلق النظام إجراء “Create Payment Contract”.
2. يولّد العقد الذكي متضمنًا تفاصيل الطلب، والمبلغ، والأطراف.
3. ينشر النظام العقد عبر RabbitMQ.
4. يمرّر RabbitMQ العقد إلى البلوكشين.
5. ينشر البلوكشين العقد

المرحلة 5: إتمام الطلب

1. ينشئ النظام سجل الطلب.
2. يُحدّث النظام المخزون.
3. يتغير وضع الطلب إلى “مدفوع”.
4. يرسل النظام تأكيد الدفع إلى المستخدم.
5. يعرض النظام رسالة نجاح.

التدفقات البديلة

* E1:فشل بوابة الدفع (الخطوات 12–15)
  1. ترجع بوابة الدفع خطأ.
  2. يعرض النظام رسالة “فشل الدفع”.
  3. يقترح النظام إعادة المحاولة أو اختيار طريقة أخرى.
  4. إذا اختار المستخدم إعادة المحاولة، يعود التنفيذ إلى الخطوة 11.
  5. إذا اختار طريقة أخرى، يعود إلى الخطوة 5.
* E2:: انقطاع شبكة البلوكشين (الخطوات 16–20)
  1. تفشل عملية النشر إلى البلوكشين.
  2. يسجّل النظام الخطأ.
  3. يستمر النظام في إكمال الطلب.
  4. يجدول النظام إعادة محاولة لاحقاً.
  5. يعود إلى الخطوة 20.
* E3: نفاد المخزون (الخطوة 22)
  1. يكتشف النظام عدم كفاية المخزون.
  2. يخطر النظام المستخدم بنفاد المنتج.
  3. تنتهي حالة الاستخدام أو تعاد من الاختيار.

حالة الاستخدام 3: إنشاء الطلب

معلومات حالة الاستخدام

* **اسم الحالة:** إنشاء الطلب
* الفاعل الرئيسي: المستخدم
* **الهدف:** إنشاء طلبٍ من عناصر عربة التسوق
* **المستوى:** هدف مستخدم
* **النطاق:** وحدة التسوق

الشروط المسبقة

1. يكون المستخدم مسجَّل دخول إلى النظام.
2. توجد عناصر في عربة التسوق.
3. يمتلك المستخدم عنوان شحن واحد على الأقل.
4. تكون المنتجات متوفرة في المخزون.

شروط النجاح (الشروط اللاحقة)

1. يُنشأ الطلب بنجاح.
2. تُفرغ عربة التسوق.
3. تتغير حالة الطلب إلى “في انتظار الدفع”.
4. يُحتجز المخزون للعناصر.
5. يتابع المستخدم إلى عملية الدفع.

السيناريو الأساسي الناجح

المرحلة 1: مراجعة العربة

1. يتوجه المستخدم إلى عربة التسوق.
2. يعرض النظام محتويات العربة مع:
   * تفاصيل وصور المنتجات
   * الكميات والأسعار
   * المجموع الكلي
3. يراجع المستخدم العناصر.
4. يمكن للمستخدم تعديل الكميات أو إزالة عناصر.
5. يعيد النظام احتساب الإجمالي للتغييرات.

المرحلة 2: معلومات الشحن

1. يضغط المستخدم “إنشاء الطلب”.
2. يعرض النظام العناوين المحفوظة للمستخدم.
3. يختار المستخدم عنوان الشحن.

المرحلة 3: التحقق من الطلب

1. يتحقق النظام من:
   1. توفر المنتجات
   2. صحة عنوان الشحن
   3. توفر طريقة التوصيل
   4. حالة حساب المستخدم
2. يحتجز النظام المخزون للعناصر.
3. يحسب النظام المجموع النهائي.

المرحلة 4: إنشاء السجل

1. ينشئ النظام سجل الطلب متضمنًا:
   1. معرف فريد للطلب
   2. بيانات المستخدم
   3. تفاصيل المنتجات
   4. معلومات الشحن
   5. المبلغ الكلي
2. يضبط النظام حالة الطلب إلى "قيد الانتظار".
3. يفرغ النظام عربة التسوق.
4. يولّد النظام تفاصيل تأكيد الطلب.

المرحلة 5: التحضير للدفع

1. يعرض النظام صفحة تأكيد الطلب.
2. يعرض النظام خيارات الدفع.
3. يختار المستخدم المضيّ قُدماً نحو الدفع.
4. يحوّل النظام المستخدم إلى عملية الدفع (يُطلق حالة الاستخدام “معالجة الدفع”).

التدفقات البديلة

* E1: عدم توفر المنتج (الخطوة 9)
  1. يكتشف النظام نفاد المنتج.
  2. يعرض النظام المنتجات غير المتوفرة.
  3. يعيد المستخدم اختيار منتجات جديدة ويلغى الطلب.
* E2: عنوان شحن غير صالح (الخطوات 8–9)
  1. يكتشف النظام نقصاً في بيانات العنوان.
  2. يوجّه المستخدم إلى “إدارة العناوين”.
  3. يضيف المستخدم عنواناً صالحاً.
  4. يعود التنفيذ إلى الخطوة 8.
* E3: إلغاء المستخدم (أي خطوة)
  1. يضغط المستخدم “إلغاء الطلب”.
  2. يؤكد النظام الإلغاء.
  3. يفرج النظام عن المخزون المحجوز.
  4. يعيد النظام محتويات السلة.
  5. تنتهي حالة الاستخدام.

نقاط التكامل بين حالات الاستخدام

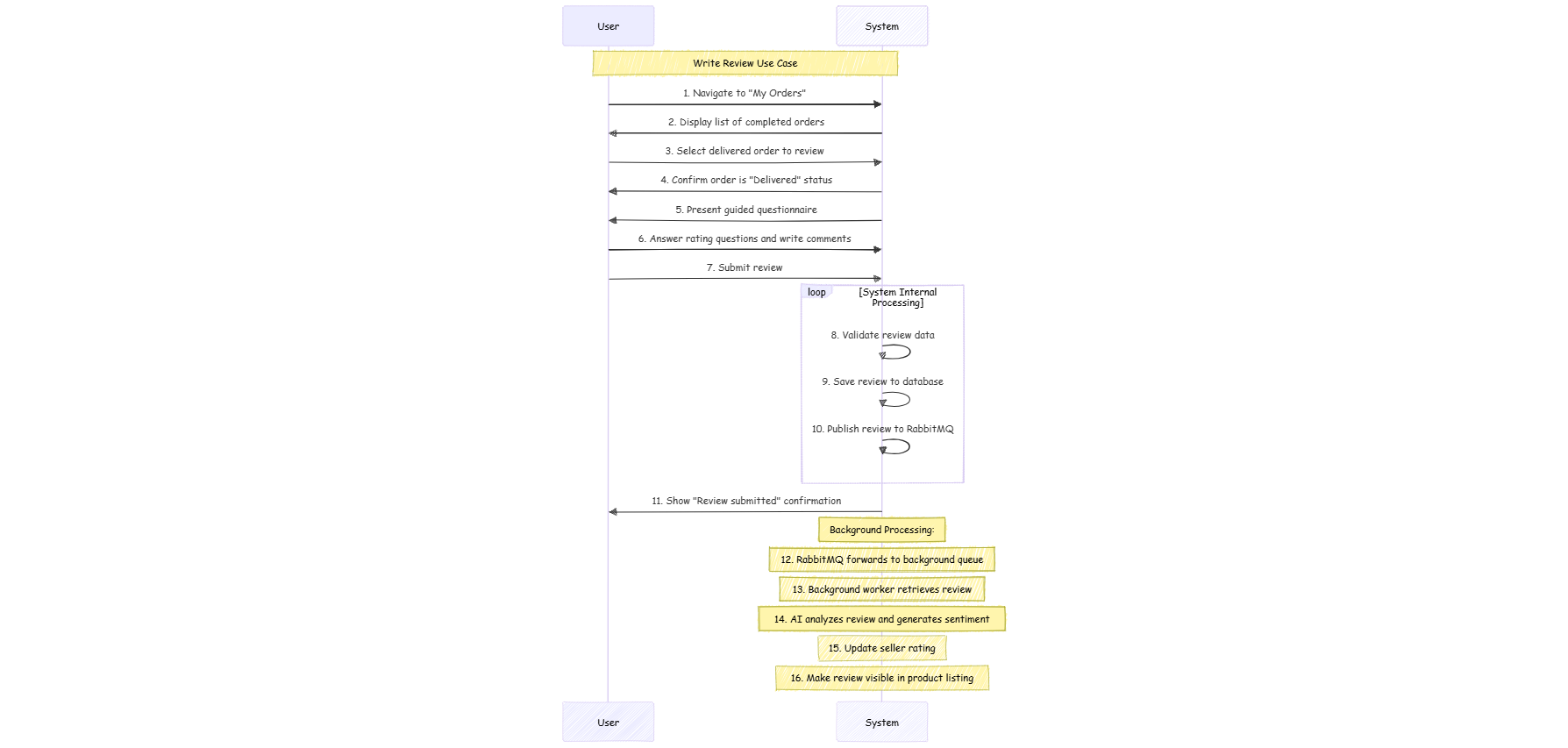
1. إنشاء الطلب ← معالجة الدفع
   * بعد إنشاء الطلب، يُطلَق سيناريو الدفع تلقائيّاً.
2. معالجة الدفع ← كتابة مراجعة
   * بعد اكتمال الدفع والتسليم، يُصبح المستخدم مؤهلاً لكتابة مراجعة.
3. كتابة مراجعة ← نظام الذكاء الاصطناعي
   * تُرسل المراجعات إلى خدمة AI عبر RabbitMQ لتحليل المشاعر وتوقع التقييم.
4. معالجة الدفع ← البلوكشين
   * يُولّد الدفع الناجح عقدًا ذكيًّا يُنشر على شبكة البلوكشين.

## مخططات تسلسل النظام SSD:

### كتابة مراجعة:

يوضح هذا المخطط التسلسلي الخطوات التي يتبعها المستخدم والنظام عند تنفيذ حالة استخدام "كتابة مراجعة" بعد استلام الطلب. تبدأ العملية بتفاعل المستخدم مع واجهة النظام وتنتهي بنشر المراجعة وتحليلها في الخلفية. التسلسل كما يلي:

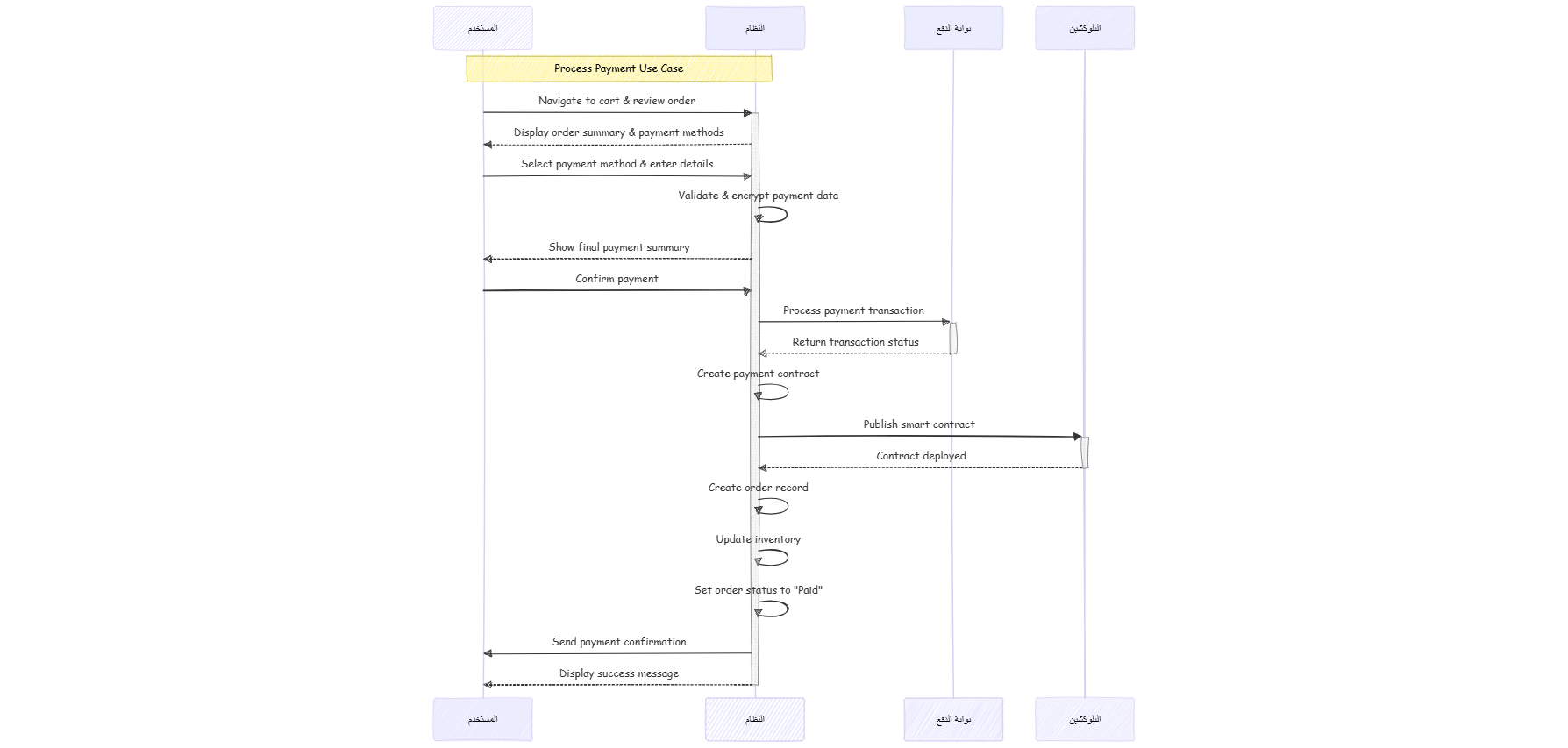
1. ينتقل المستخدم إلى قسم "طلباتي" لعرض الطلبات السابقة.
2. يعرض النظام قائمة الطلبات المكتملة ليتمكن المستخدم من اختيار الطلب المناسب.
3. يختار المستخدم الطلب الذي تم توصيله ويرغب في تقييمه.
4. يتحقق النظام من أن حالة الطلب هي "تم التوصيل".
5. يعرض النظام استبيانًا موجهًا يحتوي على أسئلة تقييم ومجال لإضافة تعليقات.
6. يقوم المستخدم بالإجابة على الأسئلة وكتابة التعليقات حول تجربته.
7. يُرسل المستخدم التقييم للنظام.
8. المعالجة الداخلية للنظام (System Internal Processing):
9. يقوم النظام بالتحقق من صحة بيانات المراجعة.
10. يتم حفظ المراجعة في قاعدة البيانات.
11. يتم نشر المراجعة عبر RabbitMQ لإرسالها إلى نظام المعالجة الخلفية.
12. يُظهر النظام رسالة تأكيد للمستخدم تفيد بأن المراجعة تم إرسالها بنجاح.
13. المعالجة الخلفية (Background Processing):
14. تقوم RabbitMQ بإرسال المراجعة إلى قائمة الانتظار الخلفية.
15. يسترجع العامل الخلفي (Background Worker) المراجعة من القائمة.
16. يستخدم النظام الذكاء الاصطناعي لتحليل المراجعة وتوقع تقييم بناءً على هذه المراجعة النصية.
17. يتم تحديث تقييم البائع بناءً على التحليل.
18. تصبح المراجعة مرئية في صفحة المنتج ليستفيد منها باقي المستخدمين.



### معالجة الدفع:

يوضح هذا المخطط التفاعلات بين المستخدم والنظام وبوابة الدفع وبلوكشين أثناء تنفيذ عملية الدفع. يصف خطوات تنفيذ الدفع بداية من مراجعة الطلب وحتى تأكيد نجاح العملية. التسلسل كما يلي:

1. ينتقل المستخدم إلى سلة المشتريات ويقوم بمراجعة الطلب.
2. يعرض النظام ملخص الطلب وطرق الدفع المتاحة.
3. يختار المستخدم وسيلة الدفع ويُدخل بياناتها.
4. يتحقق النظام من صحة بيانات الدفع ويقوم بتشفيرها لأغراض الأمان.
5. يعرض النظام للمستخدم ملخص الدفع النهائي.
6. يؤكد المستخدم عملية الدفع.
7. التفاعل مع بوابة الدفع:
8. يقوم النظام بإرسال المعاملة إلى بوابة الدفع لمعالجتها.
9. تقوم بوابة الدفع بمعالجة المعاملة المالية ثم تُعيد حالة العملية (نجاح/فشل).
10. التعامل مع البلوكشين والعقود الذكية:
11. يُنشئ النظام عقد دفع يحتوي على تفاصيل المعاملة.
12. يتم نشر العقد الذكي إلى شبكة البلوكشين.
13. تُرسل البلوكشين إشعارًا بأنه تم نشر العقد بنجاح.
14. تحديث النظام الداخلي:
15. يُنشئ النظام سجلًا جديدًا للطلب.
16. يتم تحديث المخزون بناءً على الطلب.
17. يُغيّر النظام حالة الطلب إلى "مدفوع".
18. يُرسل النظام تأكيد الدفع للمستخدم.
19. يعرض النظام رسالة تفيد بنجاح العملية.



### إتمام طلب:

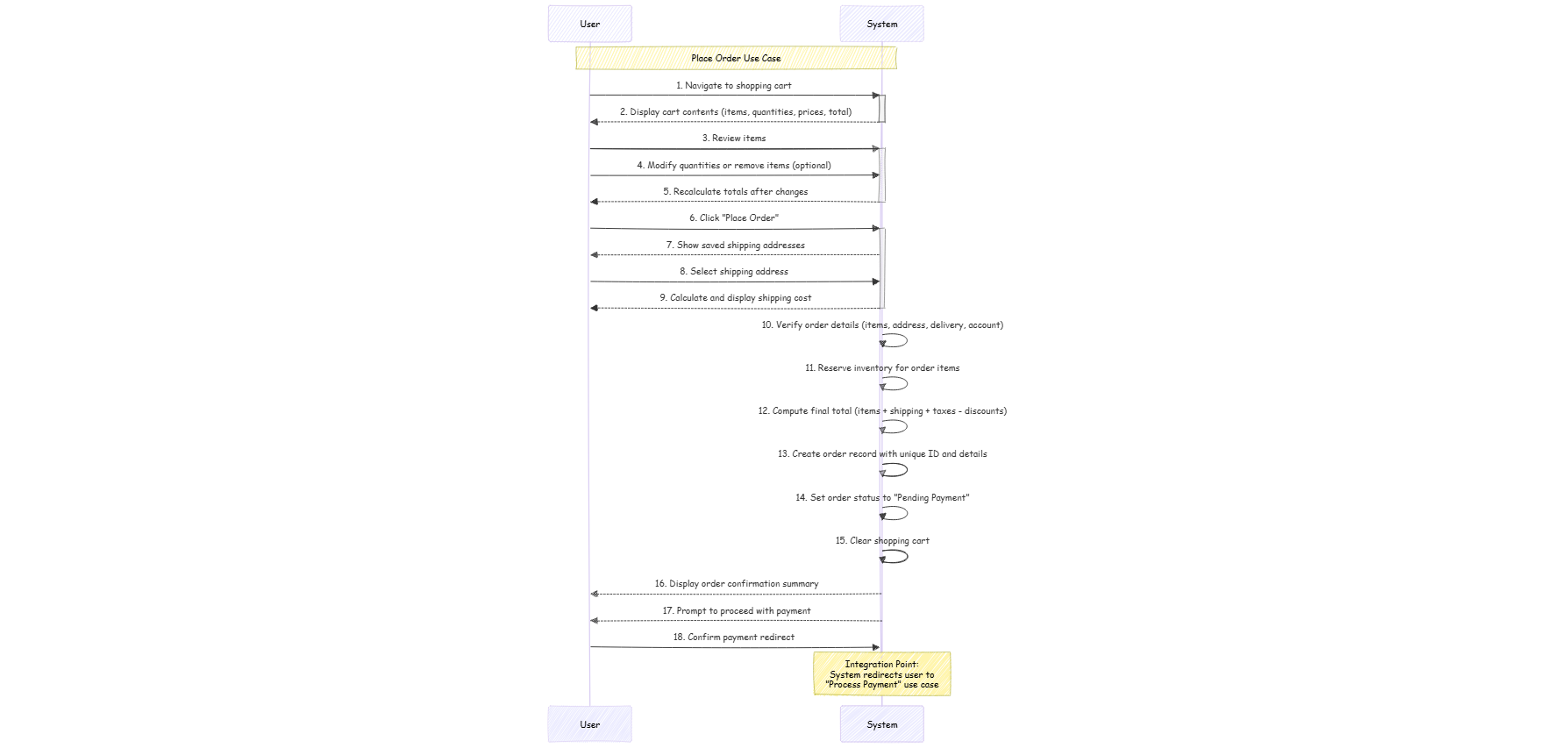
يوضح هذا المخطط تسلسل الخطوات التفاعلية بين المستخدم والنظام أثناء عملية إجراء طلب في نظام التجارة الإلكترونية. يبدأ السيناريو بدخول المستخدم إلى عربة التسوق، ويتدرج حتى يتم توجيه المستخدم إلى واجهة الدفع.

وصف الخطوات:

1. يدخل المستخدم إلى عربة التسوق لبدء عملية الطلب.
2. يعرض النظام محتويات العربة من منتجات وكمياتها وأسعارها والإجمالي.
3. يقوم المستخدم بمراجعة العناصر داخل العربة.
4. يمكن للمستخدم تعديل الكميات أو حذف بعض العناصر (اختياري).
5. يعيد النظام حساب الإجمالي بعد أي تعديل.
6. ينقر المستخدم على زر "إجراء الطلب" للمتابعة.
7. يعرض النظام العناوين المحفوظة لدى المستخدم.
8. يحدد المستخدم عنوان الشحن المناسب.
9. يحسب النظام تكلفة الشحن ويعرضها للمستخدم.
10. يتحقق النظام من تفاصيل الطلب مثل العناصر والعنوان وخيارات التوصيل والحساب.
11. يتم حجز الكمية المطلوبة من المخزون مؤقتًا لضمان توفرها.
12. يحسب النظام المجموع النهائي شاملاً تكلفة العناصر والشحن والضرائب والخصومات.
13. يُنشئ النظام سجلًا للطلب مع رقم تعريف فريد وتفاصيل الطلب.
14. يتم تعيين حالة الطلب إلى "في انتظار الدفع".
15. يتم تفريغ عربة التسوق بعد حفظ الطلب.
16. يعرض النظام ملخص تأكيد الطلب للمستخدم.
17. يطلب من المستخدم المتابعة إلى الدفع.
18. يؤكد المستخدم التحويل إلى بوابة الدفع.

نقطة التكامل:

في نهاية المخطط، يقوم النظام بإعادة توجيه المستخدم إلى حالة الاستخدام "معالجة الدفع" لاستكمال عملية الشراء.

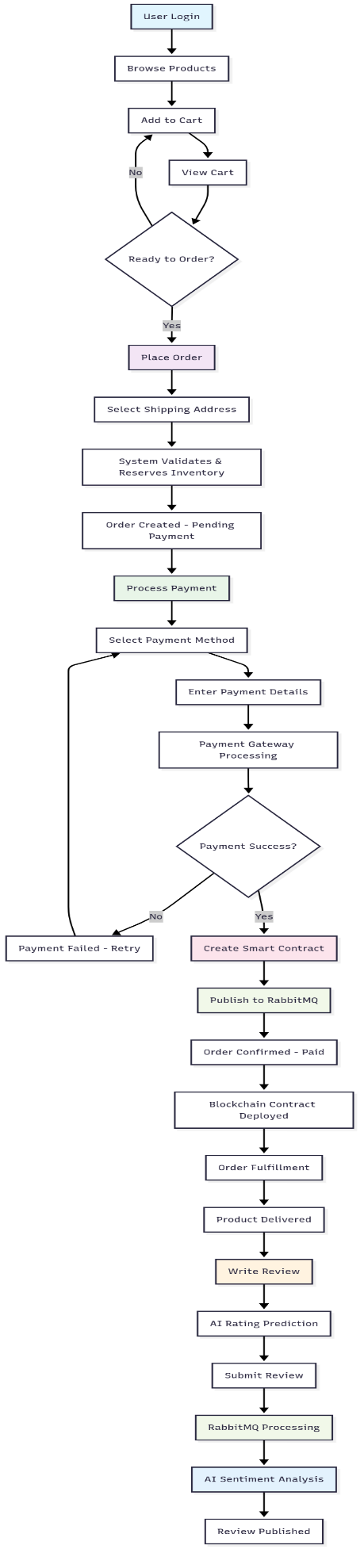


### مخطط تدفق النظام:

يوضح هذا المخطط الانسيابي تسلسل الخطوات التي يمر بها المستخدم في نظام التجارة الإلكترونية، بدءًا من تسجيل الدخول وحتى نشر المراجعة، مرورًا بعملية الدفع وتنفيذ الطلب وتوظيف تقنيات الذكاء الاصطناعي في تحليل المراجعات.

وصف مراحل تدفق العملية:

1. تسجيل دخول المستخدم إلى النظام.
2. تصفح المنتجات المتوفرة في المنصة.
3. إضافة المنتجات إلى عربة التسوق حسب اختيار المستخدم.
4. عرض العربة لمراجعة المنتجات المختارة.
5. التحقق من الاستعداد للطلب، وإذا لم يكن جاهزًا يمكنه التعديل ثم الرجوع.
6. النقر على "إتمام الطلب" للانتقال إلى الخطوات التالية.
7. تحديد عنوان الشحن المناسب من قبل المستخدم.
8. يقوم النظام بالتحقق من البيانات وحجز الكميات من المخزون.
9. يتم إنشاء الطلب وتعيين حالته إلى "في انتظار الدفع".
10. الانتقال إلى معالجة الدفع.
11. تحديد وسيلة الدفع من قبل المستخدم.
12. إدخال بيانات الدفع (مثل البطاقة أو المحفظة الرقمية).
13. معالجة الدفع عبر بوابة الدفع الإلكترونية.
14. التحقق من نجاح عملية الدفع:
15. في حال فشل الدفع: يعاد توجيه المستخدم لإعادة المحاولة.
16. في حال نجاح الدفع:
    * 1. يتم إنشاء عقد ذكي (Smart Contract).
      2. يُرسل الطلب إلى RabbitMQ لمعالجة الرسائل.
      3. يُؤكد الطلب وتُحدث حالته إلى "مدفوع".
      4. يُنشر العقد الذكي على البلوكشين.
      5. تنفيذ الطلب
      6. تسليم المنتج إلى المستخدم.
      7. يتمكن المستخدم من كتابة مراجعة للمنتج.
      8. يستخدم النظام الذكاء الاصطناعي لتوقع التقييم الرقمي بناءً على المراجعة.
      9. يقوم المستخدم بإرسال المراجعة النهائية.
      10. تتم معالجة المراجعة عبر RabbitMQ.
      11. يقوم نموذج الذكاء بتقييم المراجعة.
      12. يتم نشر المراجعة على المنصة بعد المعالجة.



## مخطط تعاوني يوضح التكامل بين حالات الاستخدام المذكورة:

يوضح المخطط التالي التفاعل بين الكائنات الأساسية في نظام التجارة الإلكترونية خلال دورة حياة العملية التجارية الكاملة، بدءاً من التسوق وانتهاءً بكتابة المراجعة.

* **الكائنات المشاركة:**
* **المستخدم**: العميل الذي يقوم بعملية الشراء
* **نظام التجارة الإلكترونية**: النظام الرئيسي الذي ينسق جميع العمليات
* **عربة التسوق**: تحفظ المنتجات المختارة للشراء
* **العنوان**: يحدد مكان توصيل الطلب
* **الطلب**: يمثل عملية الشراء المكتملة
* **الدفع**: يدير العملية المالية للشراء
* **المراجعة**: تقييم العميل لتجربة الشراء
* **مراحل التفاعل:**
* **المرحلة الأولى: التسوق واختيار العنوان**

يبدأ المستخدم بإضافة المنتجات لعربة التسوق، ثم يطلب إتمام الطلب. يعرض النظام العناوين المحفوظة للمستخدم ليختار عنوان التوصيل المناسب.

* **المرحلة الثانية: إنشاء الطلب**

بعد تأكيد العنوان، ينشئ النظام طلباً جديداً يربط بين المنتجات المختارة والعنوان المحدد.

* **المرحلة الثالثة: معالجة الدفع**

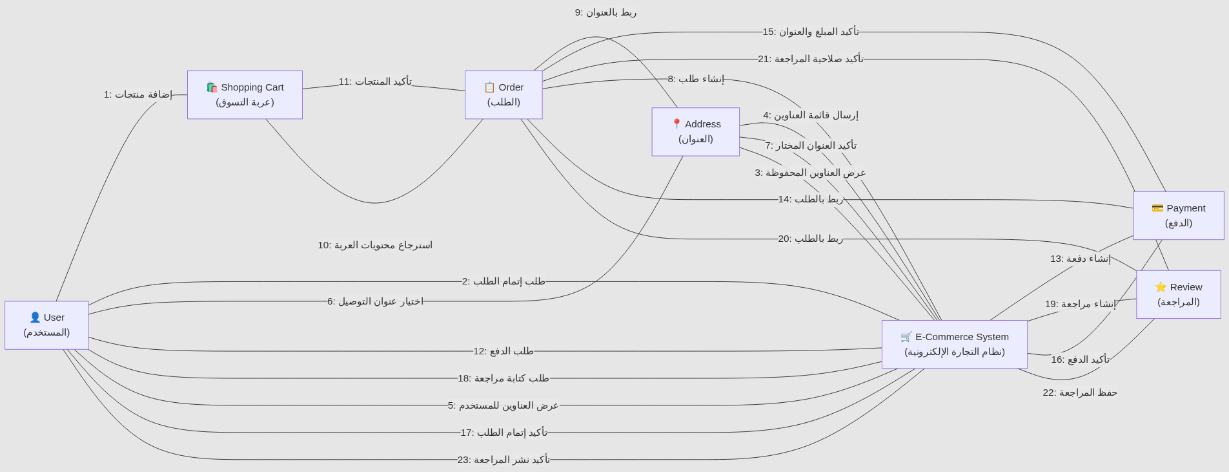
يطلب المستخدم إجراء عملية الدفع، فينشئ النظام معاملة مالية مرتبطة بالطلب ويتم تأكيد إتمام العملية.

* **المرحلة الرابعة: تقييم التجربة**

بعد استلام الطلب، يمكن للمستخدم كتابة مراجعة لتقييم تجربة الشراء، والتي تُحفظ وتُنشر في النظام.

أهمية المخطط:

يوضح هذا المخطط كيفية تفاعل مكونات النظام الأساسية لتحقيق تجربة شراء متكاملة، مع التأكيد على أهمية كل مرحلة في ضمان نجاح العملية التجارية وتحقيق رضا العميل.



# الفصل الرابع الدراسة النظرية

## البنية الخلفية (Back-End)

### Modular Monolithic

#### ما هو Modular Monolith؟

يشير هذا المصطلح إلى نظام أحادي (monolith) لكنه مُقسم داخليًا إلى وحدات واضحة المعالم (Modules)، كل منها مسؤول عن نطاق وظيفي محدد. هذه الوحدات قد تتبع حدودًا واضحة (Bounded Contexts)مشابهة لما هو موجود في DDD (Domain-Driven Design)، ولكنها جميعًا تعمل ضمن نفس عملية التنفيذ (Process).

جميع الوحدات يتم تجميعها وتوزيعها كوحدة تنفيذية واحدة (مثل ملف .dll أو .jar)، على عكس الخدمات المصغرة حيث تكون كل خدمة مستقلة وقابلة للنشر وحدها.

#### الميزات الرئيسية:

1. **فصل واضح للمجالات (Bounded Contexts):**  
   كل وحدة تمثل مجموعة وظائف مرتبطة لديها حدود واضحة ونماذج بيانات وخدمات مستقلة.
2. **التطوير المستقل لكل وحدة:**  
   يمكن لفريق واحد العمل على وحدة دون التأثير على أخرى، ما يعزز السرعة والفعالية في التطوير
3. **الكفاءة والأداء:**  
   كل الوحدات تعمل ضمن نفس العملية (process)، ما يقلل التأخير الناتج عن الاتصالات الشبكية بين الخدمات، مقارنة بالـ Microservices.
4. **بساطة النشر:**  
   طالما هو تطبيق واحد يقوم بنشر دفعة واحدة، مما يقلل تعقيد البُنى التحتية المطلوبة في الخدمات المصغرة.
5. **معاملات موحدة:**  
   مشاركة قاعدة بيانات واحدة تسهل التعامل مع المعاملات الموزعة، فتجنب تعقيدات المزامنة والتوزيع في Microservices.
6. **طريق واضح نحو Microservices لاحقًا:**  
   إن تطلب المشروع ذلك مستقبلًا، يمكن استخراج وحدة أو أكثر إلى خدمات مستقلة بسهولة نسبية.

#### كيف يبنى Modular Monolith عمليًا؟

* **تقسيم الكود إلى وحدات (Modules):**  
  يُعدّ تقسيم الكود إلى وحدات مستقلة (Modules) هو الأساس في بناء الـ Modular Monolith. حيث تُقسم المنظومة إلى وحدات منطقية (مثل: وحدة المستخدمين، وحدة الفوترة، وحدة المنتجات، إلخ)، وكل وحدة تتولى مسؤولية نطاق وظيفي محدد ضمن النظام، بما يتوافق مع مبدأ **Separation of Concerns** و**Bounded Context** من نمط Domain-Driven Design.

كل وحدة من هذه الوحدات يمكن أن تُبنى باستخدام تصميم معماري خاص بها حسب درجة تعقيدها واحتياجاتها.

هذا التنوع في التصاميم داخل نفس النظام يُعدّ من أهم مميزات Modular Monolith، إذ يُتيح مرونة في اختيار النمط المعماري الأنسب لكل وحدة، دون التأثير على بقية النظام، طالما أن حدود الاتصال بين الوحدات واضحة وثابتة.

* **واجهات واضحة (Clear Interfaces):**  
  يجب تقييد الوصول بين الوحدات باستخدام آليات مثل **Interfaces** أو **Public Contracts** فقط، ومنع الوصول المباشر إلى الكود الداخلي لوحدة من وحدة أخرى، لضمان بقاء النظام مرنًا ومستقلاً.
* **العزل على مستوى البيانات:**  
  لكل وحدة جداولها أو مخططاتها الخاصة في قاعدة البيانات الموحدة، مما يمنع الوحدات الأخرى من الوصول المباشر إلى بياناتها.

#### أنماط قواعد البيانات داخل Modular Monolith

أحد القرارات المعمارية المحورية في هذا النمط تتعلق بكيفية تنظيم قاعدة البيانات. وهناك **نمطين أساسيين** يُمكن اتباعهما:

النمط الأول: قاعدة بيانات مشتركة (Shared Database)

* **الوصف**: جميع الوحدات تتشارك نفس قاعدة البيانات، وتصل إلى نفس الجداول.
* الميزات:
  + سهولة الوصول إلى البيانات المشتركة.
  + بساطة في الإعداد والتهيئة.
* العيوب:
  + ضعف في الفصل بين الوحدات.
  + تعارضات محتملة عند التطوير المتزامن.
  + يصعب تتبع تأثيرات التغييرات على النظام ككل.

يستخدم هذا النمط غالبًا عندما تكون الوحدات مترابطة بشدة أو عندما يكون الأداء أكثر أهمية من الفصل الكامل، أو في حالة

النمط الثاني: قاعدة بيانات مستقلة لكل وحدة (Module-specific Database)

* **الوصف**: كل وحدة تملك قاعدة بياناتها الخاصة (أو جزءاً معزولاً منها)، وتتعامل فقط مع البيانات التابعة لنطاقها (Bounded Context).
* الميزات:
  + عزل قوي بين الوحدات (Low Coupling).
  + سهولة اختبار كل وحدة بشكل منفصل.
  + تحسين الأمان والصلاحيات.
* العيوب:
  + الحاجة إلى آليات تكامل مثل API داخلي أو Messaging بين الوحدات.
  + تعقيد في إدارة البيانات المرتبطة بين الوحدات.

هذا النمط يتقاطع مع مبادئ Microservices من حيث فصل البيانات، لكنه يُنفّذ داخل نفس التطبيق الأحادي.

#### لماذا Modular Monolith؟

الأداء والكفاءة:

التواصل بين الوحدات داخل نفس العملية يقلل زمن الاستجابة ويقي من مشاكل الشبكات.

البساطة التشغيلية:

نظام وحدة نشر واحد يجعل النشر، التكوين، والترقية أسهل بمرونة، مقارنة بالتوزيع متعدد الخدمات.

قابلية الصيانة:

تنظيم الكود ضمن وحدات يمنع التداخل والازدحام في الكود (Big ball of mud)، ويحافظ على هيكل صلب وسهل الفهم.

استمرارية التطوير:

يمكن البدء بنظام أحادي وحدوي، ثم مع النمو وتحقيق الاستقرار، تُستخرج وحدات إلى خدمات مستقلة عند الحاجة.

#### أفضل الممارسات (Best Practices)

* استخدام Contract-based communication بين الوحدات عند الفصل الكامل.
* توثيق حدود البيانات لكل وحدة بوضوح.
* استخدام Database Schema لكل وحدة حتى لو كانت القاعدة مشتركة.
* تأمين الوصول إلى البيانات عبر Repositories أو Services داخل الوحدة لمنع التداخل.
* تجهيز النظام تدريجيًا للانتقال إلى Microservices عند الحاجة، وذلك بجعل كل وحدة قابلة للنشر مستقبلاً.

### Clean Architecture

#### تعريف*:*

هو نمط تصميم هيكلي للبرمجيات يهدف إلى تحقيق خمسة أهداف رئيسية: فصل الاهتمامات، قابلية الاختبار، استقلالية المنطق عن التقنيات والبنية التحتية، سهولة الصيانة، والاعتماد على سياسة الاعتماد الداخلي (Dependency Inversion) CQRS

تُحكم العلاقة بعدم وجود تبعيات من الطبقات الداخلية إلى الخارجية، بل يُعكس اتجاهها دومًا نحو المركز (Dependency Rule)

#### المبادئ الرئيسية الداعمة

Clean Architecture يضع عددًا من قواعد SOLID:

* **SRP:** كل طبقة/كائن مسؤولة عن وظيفة واحدة.
* **DIP:** الاعتماد يتم عبر التجريد وليس مباشرة على التفاصيل.
* **ISP، LSP، OCP** وتعزيز قابلية الصيانة والمرونة.

كما يعتمد على مبدأ **Separation of Concerns** لتمكين التحديث المستقل لكل جزء دون التأثير على الآخرين

#### استقلالية الطبقات

* **استقلال الطبقة الداخلية (Domain)** عن البنية التحتية يضمن تركيزها على منطق الأعمال فقط.
* **استقلال Use Cases (Application)** يسمح بتعريف الوظائف دون الربط بالأدوات الخارجية.
* **التعامل مع التفاصيل** عبر Adapter وInfrastructure يتيح تغيير التقنيات بسهولة

### RabbitMQ

#### التعريف

هو وسيط رسائل برمجي (Message Broker) مفتوح المصدر يعتمد على بروتوكول **AMQP (Advanced Message Queuing Protocol)**. يُستخدم لتمكين التواصل غير المتزامن بين مكونات الأنظمة البرمجية، وهو يُعد من العناصر الأساسية في البنى الموزعة والمعماريات الحديثة مثل **Microservices** و**Modular Monoliths**.

يساهم RabbitMQ في فصل المكونات البرمجية عن بعضها البعض، مما يحقق **المرونة، والموثوقية، وقابلية التوسع** في نقل البيانات عبر الرسائل بين الوحدات دون الحاجة إلى الربط المباشر بينها.

#### المكونات الأساسية وطريقة العمل

يتكوّن RabbitMQ من عدة مكونات أساسية تؤدي أدوارًا محددة في عملية إرسال واستقبال الرسائل:

• المعالج (Consumer): الكيان الذي يستقبل الرسائل من قائمة الانتظار وينفّذ الإجراءات المطلوبة عليها.  
• قائمة الانتظار (Queue): تخزن الرسائل مؤقتًا حتى تتم معالجتها.  
• الموزع (Exchange): يستقبل الرسائل من الناشر ويُعيد توجيهها إلى قائمة أو أكثر من قوائم الانتظار (Queues) وفقًا لنوع التوزيع:

* Direct: التوجيه حسب مفتاح مطابق.
* Topic: التوجيه بناءً على نمط يحتوي على wildcards.
* Fanout: بث الرسالة لجميع القوائم المتصلة.
* Headers: التوجيه استنادًا إلى رؤوس الرسائل (Message Headers).
* الناشر (Producer): الكيان الذي يُنشئ الرسائل ويرسلها إلى الخادم.

#### الخصائص الرئيسية

يمتاز RabbitMQ بعدة خصائص تجعله مناسبًا للتطبيقات المتقدمة:

* **الموثوقية والتأكيد**: يدعم آليات تأكيد الاستلام (Acknowledgements) والتأكيد من جهة الناشر (Publisher Confirms) لضمان عدم فقدان الرسائل.
* **الرسائل الدائمة والإصرار (Durability)**: يمكن تكوين القوائم والرسائل لتُخزّن على القرص، مما يسمح باستعادة الحالة بعد توقف الخدمة.
* **التوجيه المتقدم (Advanced Routing)**: يتيح استخدام أنماط متعددة لتوزيع الرسائل حسب السيناريو المناسب.
* **قابلية التوسع والجاهزية العالية**: يدعم التكتل (Clustering) وتكرار القوائم (Mirrored Queues) لتحقيق استمرارية الخدمة ومقاومة الأعطال.
* **الدعم الإضافي (Plugins)**: يوفر واجهات إدارة رسومية، دعم بروتوكولات إضافية مثل MQTT وSTOMP، وربط RabbitMQ بأنظمة خارجية.
* **أمان الاتصال**: يدعم بروتوكولات تشفير مثل TLS، بالإضافة إلى التحكم الدقيق في صلاحيات الوصول.

#### المزايا في النظم المتقدمة

**فصل المكونات (Decoupling)**: يسمح بفصل الوحدات المنطقية عن بعضها من خلال قنوات اتصال غير مباشرة، مما يعزز من استقلالية التطوير والصيانة.

**تحسين الأداء**: يقلل من زمن الانتظار في العمليات الثقيلة عبر معالجتها بشكل غير متزامن.

**قابلية التوسع الديناميكي**: يمكن زيادة عدد المعالِجين (Consumers) للتعامل مع حجم أكبر من الرسائل دون تعديل على منطق الإنتاج.

**دعم أنماط معمارية متنوعة**: مثل Pub/Sub، Work Queues، وRPC، مما يجعله ملائمًا لتطبيقات مختلفة من حيث البنية والحجم.

## الذكاء الاصطناعي وتحليل المشاعر

يُعنى علمُ الذكاءِ الاصطناعيِّ بفهمِ وتصميمِ الأنظمةِ القادرةِ على محاكاةِ القدراتِ المعرفيّةِ البشريّةِ، مثلَ الفهمِ والاستدلالِ والتعلّمِ، وقد تطوّر هذا المجالُ بفضلِ التقدّمِ في قدراتِ الحوسبةِ وتوفّرِ كميّاتٍ ضخمةٍ من البياناتِ.

في معالجةِ النصوصِ، يُستخدمُ تمثيلُ الكلماتِ عبرَ TF‑IDF (Term Frequency–Inverse Document Frequency) لابتكارِ متجهاتٍ رقميةٍ تعكسُ أهميّةَ كلِّ مصطلحٍ في مستندٍ واحدٍ مقارنةً ببقية المستنداتِ. يقيسُ مكوّنُ TF تواترَ الكلمةِ داخلَ المستندِ، في حين يعبّرُ مكوّنُ IDF عن ندرةِ تلكَ الكلمةِ عبرَ مجموعةٍ كبيرةٍ من المستنداتِ؛ مما يتيحُ زيادةَ وزنِ الكلماتِ المميّزةِ وتقليلَ وزنِ الكلماتِ الشائعةِ.

يعتمدُ Support Vector Machine (SVM) على إيجادِ مستوىٍ فاصلٍ (Hyperplane) يفصلُ بينَ الفئاتِ بأكبرِ هامشٍ ممكنٍ في الفضاءِ المتجهيِّ، مستندًا إلى نقاطٍ حاسمةٍ تُسمّى «دعائمٍ» (Support Vectors). وتُطبّقُ دوالُ النوى (Kernels) مثل Gaussian وPolynomial لتحويلِ البياناتِ إلى فضاءاتٍ أبعادُها أعلى، مما يسهلُ فصلَ الفئاتِ غيرِ الخطّيّةِ.

تُعنى الشبكاتُ العصبيّةُ الالتفافيّةُ (CNN) باستخراجِ الميزاتِ المحلّيّةِ ضمنَ المتونِ من خلالِ تمريرِ مرشّحاتٍ (Filters) داخلَ طبقاتِ الالتفافِ (Convolutional Layers)، ثمّ تقليصِ الأبعادِ عبرَ طبقاتِ التجميعِ (Pooling Layers)، قبلَ ربطِ الخصائصِ المكتشفةِ في طبقاتِ الارتباطِ الكاملِ (Fully Connected Layers) لتوليدِ التوزيعِ الاحتماليِّ للفئاتِ.

في المقابلِ، تسمحُ الشبكاتُ ثنائيّةُ الاتّجاهِ LSTM (BiLSTM) بمعالجةِ التسلسلِ النصّيِّ من الأمامِ والخلفِ معًا، مستفيدةً من وحداتِ الذاكرةِ الطويلةِ والقصيرةِ الأمدِ (LSTM) لالتقاطِ التبعياتِ البعيدةِ داخلَ المتونِ وضمانِ فهمٍ سياقيٍّ أعمقَ.

ثم جاءَ نموذجُ المحوّلات (Transformers) ليسخّرَ آليّةَ الانتباهِ الذاتيِّ (Self‑Attention)، التي تُقَيّمُ أوجهَ الترابطِ بينَ جميعِ كلماتِ الجملةِ في آنٍ واحدٍ، مما يُعزّزُ قدرةَ النماذجِ على استيعابِ العلاقاتِ السياقيّةِ المعقّدةِ.

استنادًا إلى المحوّلاتِ، طوّرَ نموذجُ BERT (Bidirectional Encoder Representations from Transformers) عمليّتيَ التعلّمِ المسبقِ (Pre‑training) عبرَ مهمتَي “ملءِ الفراغاتِ” (Masked Language Modeling) و“التنبؤِ بالتالي” (Next Sentence Prediction)، ثمّ تخصيصُهُ (Fine‑Tuning) لمهامٍ محدّدةٍ، ما مكّنهُ من تحقيقِ أداءٍ عالٍ في استنباطِ المعاني وتحليلِ المشاعرِ.

أخيرًا، يُعتمدُ أسلوبُ التلقينِ القليل (Few‑Shot Prompting) لتمكينِ النماذجِ من تنفيذِ مهامٍ جديدةٍ عبرَ تضمينِ عددٍ محدودٍ من الأمثلةِ داخلَ الطلبِ الموجهِ إليها، مما يخفّضُ الحاجةَ إلى تدريبٍ موسّعٍ ويُعزّزُ التكيّفَ مع السياقاتِ المتنوّعةِ بكفاءةٍ عاليةٍ.

## نظام التسجيل Logginig system:

تُعنىُ أنظمةُ التسجيل (Logging Systems) بجمعِ وتخزينِ المعلوماتِ التشخيصيّةِ الخاصةِ بسيرِ عملِ التطبيقاتِ والخدماتِ، بهدفِ مراقبةِ الأداءِ، وتشخيصِ الأعطالِ، وتحليلِ سلوكِ النظامِ عبرَ الزمنِ. ومن أهمِّ المفاهيمِ المرتبطةِ بهذه الأنظمة هو التسجيل الهيكلي (Structured Logging)، الذي يتيحُ تضمينَ بياناتٍ وصفّيّةٍ لكلِّ حدثٍ في شكلِ حقولٍ منظمةٍ قابلةٍ للاستعلامِ والتحليلِ الآليِّ، مما يفوقُ التسجيلَ التقليديَّ القائمَ على نصوصٍ غيرِ مُنسقةٍ في تعزيزِ الرصدِ والتتبعِ.

يعتمدُ Serilog كمكتبةٍ متقدمةٍ للتسجيل الهيكلي في بيئاتِ .NET على فكرةِ المعالجَات (Sinks) التي ترسلُ السجلاتِ إلى وجهاتٍ متعدّدةٍ (كالملفات، وقواعدِ البياناتِ، وأنظمةِ تحليلِ السجلاتِ). ويتميّزُ Serilog بإمكانيّةِ تعريفِ مستويات التسجيل (Log Levels) وتضمينِ خصائصٍ زمنيةٍ وسياقيةٍ (Timestamps وCorrelation IDs) مع كلِّ رسالةٍ، مما يسهّلُ تجميعَ الأحداثِ المرتبطةِ بمعاملةٍ واحدةٍ أو طلبٍ محدّدٍ عبرَ جميعِ مكوّناتِ النظامِ، ويساهمُ في عزلِ الأعطال وفهمِ تداعياتِها بشكلٍ أعمقَ.

من ناحيةٍ أخرى، يوفِّرُ Seq بيئةً مركزيةً لاستقبالِ وتحليلِ السجلاتِ الصادرةِ من Serilog أو غيرِه من مكتباتِ التسجيلِ، عبرَ واجهةِ برمجةِ تطبيقاتٍ (API) قادرةٍ على استقبالِ البياناتِ بصيغةٍ هيكليةٍ (JSON). ويتيحُ Seq عملياتِ الاستعلامِ الفوريّ (Real‑Time Query) باستخدامِ لغةٍ شبيهةٍ بـ SQL، وتصميمَ لوحاتِ معلوماتٍ (Dashboards) مخصصةٍ للرصدِ والنذّاراتِ، مما يعززُ الرصدَ المستمرَّ وتحديدَ مؤشرات الأداء الحرجةِ (KPIs) والتنبؤَ بالاختناقاتِ قبلَ وقوعِها.

يتشكلُ المشهدُ النظريُّ للربطِ بين Serilog وSeq في كونِهما معًا يوفّران حلّاً متكاملاً للمراقبةِ الموزَّعةِ (Distributed Monitoring)، حيث يُعالجُ Serilog تسجيلَ الأحداثِ في كلِّ خدمةٍ على حدةٍ، بينما يقومُ Seq بمركزةِ هذه الأحداثِ وتحليلِها وعرضِ نتائجِها عبرَ بيئةٍ واحدةٍ. ويسهمُ هذا التكاملُ في تعزيزِ الرصدِ اللامتزامن (Asynchronous Observability)، وتقليلِ تأثيرِ أداءِ التسجيلِ على تجربةِ المستخدمِ، وضمانِ قابليةِ التوسع (Scalability) عند إضافةِ مكوناتٍ جديدةٍ إلى المعماريةِ دونَ الحاجةِ لإعادةِ تصميمِ نظامِ التسجيلِ بأكملهِ.

في ضوءِ ذلك، يُعدُّ استخدامُ Serilog وSeq معًا نهجًا يوازنُ بينَ المرونةِ في تكوينِ نقاطِ التسجيلِ (Sinks) والقوةِ في تحليلِ البياناتِ الهيكليةِ، ما يدعمُ اتخاذَ قراراتٍ مبنيةٍ على بياناتٍ دقيقةٍ ويعزّزُ جودةَ الخدمةِ واستدامتَها.

### تقنية سلسلة الكتل (Blockchain) ونظام Hyperledger Fabric ودورهما في تطبيقات التجارة الإلكترونية

شهدت السنوات الأخيرة تحولاً جذرياً في البنية الرقمية للمعاملات الاقتصادية، تمثل في بروز تقنية سلسلة الكتل (Blockchain) كنظام جديد لنقل وتخزين الأصول والمعلومات بطريقة آمنة، موزعة، وغير مركزية.  
وتعتبر هذه التقنية حجر الأساس للعديد من الابتكارات الحديثة، على رأسها العملات الرقمية مثل بيتكوين (Bitcoin)، والتي ظهرت لأول مرة في ورقة بحثية نشرها شخص أو مجموعة مجهولة الهوية تُدعى ساتوشي ناكاموتو (Satoshi Nakamoto) عام 2008 [1].

وقد مثلت هذه الورقة نقطة تحول تاريخية، حيث طرحت مفهوم عملة رقمية تُرسل مباشرة بين الأطراف دون الحاجة إلى وسطاء ماليين، معتمدة على دفتر أستاذ عام موزع يُعرف اليوم بسلسلة الكتل.

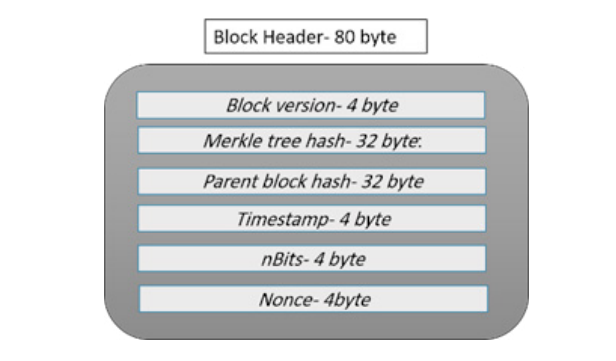
تقنية سلسلة الكتل: المفهوم والآلية

تقنية سلسلة الكتل هي قاعدة بيانات موزعة تتألف من سجل غير قابل للتعديل للمعاملات، حيث تُخزّن البيانات داخل "كتل"، وكل كتلة ترتبط بالتي تسبقها من خلال تجزئة مشفّرة، مشكّلة بذلك سلسلة زمنية متسلسلة تضمن عدم قابلية التعديل أو الحذف، وكل معاملة يتم التحقق منها من قبل غالبية العقد في الشبكة، مما يضمن الشفافية وعدم القدرة على التزوير.



أولاً: بنية الكتلة

كل كتلة تتكون من جزأين رئيسيين:

رأس الكتلة (Block Header) ويحتوي على المعلومات التالية:

* إصدار الكتلة (Block Version): يحدد القواعد المعتمدة للتحقق من الكتلة.
* جذر شجرة ميركل (Merkle Root Hash): قيمة مشفرة تمثل جميع بيانات المعاملات داخل الكتلة.
* تجزئة الكتلة السابقة (Previous Block Hash): لربط الكتل ببعضها بشكل زمني.
* الطابع الزمني (Timestamp): يُحدد وقت إنشاء الكتلة.
* مستوى الصعوبة (nBits): يحدد الهدف اللازم لتوليد التجزئة.
* القيمة العشوائية (Nonce): رقم يستخدم في عملية التعدين لتوليد التجزئة.

بيانات الكتلة (Block Data):

وهي المعاملات التي تم تشفيرها عبر خوارزمية شجرة ميركل (Merkle Tree)، حيث يتم تحويل كل معاملة إلى تجزئة، ثم تُدمج هذه القيم تدريجيًا حتى يتم الحصول على تجزئة واحدة تمثل كل محتوى الكتلة.

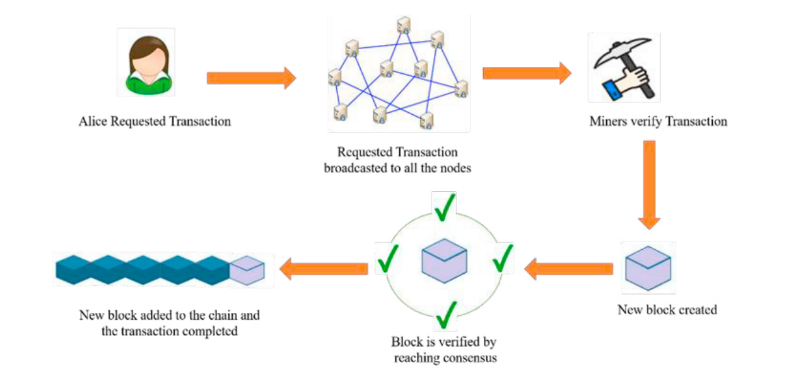
ثانياً: آلية العمل

في شبكة البلوكشين، يُطلق على المستخدمين اسم العُقد (Nodes)، وكل عقدة تمتلك نسخة كاملة من سلسلة الكتل. بعض هذه العقد تُعرف باسم المُعدّنين (Miners)، وهم المسؤولون عن التحقق من صحة المعاملات وإضافتها إلى السلسلة.

عند إنشاء معاملة جديدة (مثل تحويل الأموال أو توقيع عقد ذكي)، يستخدم المرسل توقيعاً رقمياً يتكون من مفتاح عام يعرفه الجميع، ومفتاح خاص يعرفه المرسل فقط.

بعد بث المعاملة إلى الشبكة، تبدأ عملية التعدين، حيث تحاول العقد المتخصصة حلّ لغز حسابي يُعرف باسم "إثبات العمل" (Proof of Work). أول عقدة تُنجز الحل تُكافأ برسوم المعاملة وتقوم بإنشاء الكتلة الجديدة.

بعد ذلك، يتم التحقق من الكتلة الجديدة من قِبل باقي الشبكة وفقاً لبروتوكول إجماع معين (Consensus Protocol) قبل إضافتها للسلسلة.



ثالثاً: تصنيف سلاسل الكتل

تُصنّف شبكات سلسلة الكتل إلى نموذجين رئيسيين بناءً على الصلاحيات المتعلقة بإضافة كتل جديدة.

* إذا كان بإمكان أي مستخدم إضافة كتلة جديدة، فإن سلسلة الكتل تُعد "غير مُصرح بها" (Permissionless).
* أما إذا كان يُسمح فقط لمجموعة محددة من المستخدمين بإضافة كتل جديدة، فتُعتبر سلسلة الكتل "مُصرح بها" (Permissioned).

ببساطة، يمكن تشبيه سلسلة الكتل غير المُصرح بها بشبكة الإنترنت العامة التي يستطيع أي شخص الوصول إليها، بينما تُشبه سلسلة الكتل المُصرح بها شبكة داخلية (Intranet) خاضعة للرقابة. وغالباً ما تُستخدم سلاسل الكتل المُصرح بها من قبل مجموعات أو منظمات.

سلسلة الكتل غير المصرح بها

* من الأمثلة الجيدة على هذا النوع: سلسلة الكتل العامة. في سلسلة الكتل العامة، يمكن لأي مستخدم الدخول أو الخروج من الشبكة في أي وقت. جميع المستخدمين يمتلكون نسخة من جميع المعاملات، ونظراً لأن المعاملات مرئية للجميع، لا يمكن لأحد التلاعب بها. وتُعد "بيتكوين" أشهر مثال على هذا النوع من السلاسل.

سلسلة الكتل المُصرح بها

* تُصنّف سلسلة الكتل المُصرح بها إلى نوعين رئيسيين: سلسلة الكتل الخاصة وسلسلة الكتل الائتلافية (Consortium).
* في النوع الأول (الخاص)، يُسمح فقط لمجموعة مختارة من العقد (nodes) بالدخول إلى الشبكة والتحقق من المعاملات. هذا التقييد يجعل عملية المعالجة أسرع، وحتى مع ازدياد حجم الشبكة تظل فعالية الأداء عالية. كما يتيح هذا النوع إجراء معاملات آمنة تماماً. ومع ذلك، فإن العيب الرئيسي في سلسلة الكتل الخاصة هو أن التحقق من المعاملات يصبح مركزيا لأنه خاضع لجهة واحدة [6].

رابعاً: ميزات سلسلة الكتل

1. اللامركزية: اللامركزية تُعد أهم ميزة في تقنية سلسلة الكتل، حيث تُلغي الحاجة إلى طرف ثالث أو سلطة مركزية للتحقق من المعاملات. في هذا النظام، يمكن لأي عقدة (Node) داخل الشبكة تنفيذ معاملات مع عقد أخرى دون الحاجة لمعرفة الهوية الحقيقية أو الموقع الجغرافي للطرف الآخر. إضافة إلى ذلك، ونظراً لعدم وجود نظام مركزي، فإن كل عقدة تحتفظ بنسخة كاملة من البيانات، ويتم تحديث البيانات في جميع الأنظمة عند إضافة كتلة جديدة إلى السلسلة.
2. الشفافية: جميع الخوارزميات والمعاملات في تقنية سلسلة الكتل تكون شفافة لجميع مستخدمي الشبكة. هذه الميزة تُعزز من النزاهة والمساءلة داخل الشبكة، إذ لا يمكن لأي طرف تغيير أو إضافة أو حذف البيانات. وتُعد هذه الشفافية غير مسبوقة في الأنظمة المالية، مما يفسر شعبية تقنية البلوك تشين في القطاع المالي.
3. الاستقلالية: تُعرف شبكة سلسلة الكتل بأنها "شبكة خالية من الثقة"، أي لا تعتمد على جهة مركزية لضمان الثقة. بدلاً من ذلك، يتم الاعتماد على خوارزميات تشفيرية لضمان أمان المعاملات وثقة المشاركين في الشبكة، مما يُلغي الحاجة إلى سلطات حاكمة [7].
4. عدم القابلية للتغيير: تعني خاصية عدم القابلية للتغيير في البلوك تشين أنه بمجرد تسجيل معاملة داخل السلسلة، لا يمكن تعديلها أو حذفها. المعاملة بعد تسجيلها تصبح قابلة فقط للتحقق وليس التعديل. ولأن المعاملات مخزنة على جميع العقد، فإن أي محاولة لتعديل معاملة يتم كشفها من قبل بقية العقد التي تسعى لإصلاح التلاعب.
5. عدم القابلية للاسترجاع: تشير هذه الميزة إلى أن المعاملة بعد حفظها في الشبكة لا يمكن التراجع عنها. ويُعزى ذلك إلى استخدام دالة "التجزئة" (Hash Function) التي تُنتج نسخة مشفرة من البيانات داخل كل كتلة. وبما أن هذه الدالة تُعد أحادية الاتجاه ولا يمكن استرجاع البيانات الأصلية منها، فإنها تضمن عدم القدرة على التراجع عنها. كما أن الدالة تُعد حتمية، بحيث إن أي تغيير بسيط في المُدخلات يُنتج مخرجات مختلفة تماماً، مما يُعزز من خاصية عدم التراجع.
6. قابلية التتبع والتدقيق: يتم تسجيل كل معاملة داخل دفتر أستاذ موزع مع طابع زمني محفوظ في رأس الكتلة. وبالتالي، يمكن تتبع مصدر المعاملة بالوصول إلى أي عقدة داخل الشبكة. تُعتبر هذه الميزة مفيدة في العديد من التطبيقات، وخاصة في الخدمات المالية، حيث من الضروري معرفة مصدر الأموال أو المعاملات الكبيرة للحد من الاحتيال أو غسيل الأموال.

سلسلة الكتل مقابل قواعد البيانات التقليدية:

تُعتَبَر كلٌّ من البلوكشين وقواعد البيانات التقليدية أنظمةً لتخزين وإدارة البيانات، ويمكن اعتبارهما نوعين من قواعد البيانات، حيث تتيح كلاهما عمليات القراءة والكتابة واسترجاع المعلومات. ومع ذلك فأنّ مجرّد تشابه البلوكشين مع قواعد البيانات في كونهما مخزنين منظمين للبيانات لا يُخفي الفوارق الجوهريّة في آليّات بناء الثقة، التوافق، والحماية. فيما يلي تفصيل لهذه الفوارق:

أولاً: بناء الثقة والحوكمة

تعتمد قواعد البيانات التقليديّة على سلطة مركزيّة واحدة تتحكم في سياسات القراءة والكتابة؛ فإذا كانت هذه الجهة موثوقة فلا إشكال، أمّا في البيئات التي يشوبها عجز ثقة بين أطراف متنافسة فإنّ هذه المركزية تصبح مَصدر خطر واختناق[n]

Chowdhury, Mohammad Jabed Morshed, et al. "Blockchain versus database: A critical analysis." *2018 17th IEEE International conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE, 2018.

هنا تتفوّق سلسلة الكتل بقدرتها على إزالة الوسيط عبر خوارزميات إجماع تجعل كلّ عُقدة صاحبة قرار متساوٍ، ما يشيّد ثقة حسابيّة يصعب تقويضها.

ثانياً: السرّيّة والخصوصيّة

سجلّ البلوكشين شفّاف بطبيعته؛ كلّ معاملة مرئيّة لجميع العُقد المصرَّح لها، مع الاعتماد على التوقيعات الرقميّة لحماية هويّة الأطراف لكنّ البيانات ذاتها لا تُشفَّر افتراضيّاً[n].

Chowdhury, Mohammad Jabed Morshed, et al. "Blockchain versus database: A critical analysis." *2018 17th IEEE International conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE, 2018.

. في المقابل، تُمكّن قواعد البيانات من ضبط دقيق لصلاحيات الوصول، ما يجعلها الخيار الملائم عندما تكون سرّيّة المحتوى هي الأولويّة القصوى.

ثالثاً: القدرة على التحمّل ومقاومة الأعطال

تُوزَّع نسخة السجلّ في البلوكشين على جميع العُقد؛ فإذا تعطّلت إحداها استمرت الشبكة بالعمل دون انقطاع، كما تصعُب هجمات حجب الخدمة ضد كيان لا مركزي[n].

Chowdhury, Mohammad Jabed Morshed, et al. "Blockchain versus database: A critical analysis." *2018 17th IEEE International conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE, 2018.

. أمّا قاعدة البيانات ذات الخادم الواحد فتظل عرضة لنقطة فشل مفردة ما لم تُنشأ نسخ احتياط وموازنة حمل.

رابعاً: الأداء والزمن الحقيقي

يُعدّ الأداء نقطة ضعف جوهريّة في البلوكشين العام؛ إذ قد يستغرق تأكيد المعاملة من عدة ثواني إلى عشرات الدقائق (وأحياناً ساعة) ريثما يكتمل التوافق، بينما تستطيع الأنظمة التقليديّة إنجاز آلاف المعاملات في الثانية بمهلة زمنيّة شبه فوريّة[n].

Chowdhury, Mohammad Jabed Morshed, et al. "Blockchain versus database: A critical analysis." *2018 17th IEEE International conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE, 2018.

. لذلك تُفضَّل قواعد البيانات في التطبيقات الحسّاسة للزمن كالدفع الفوري أو أنظمة الحجز.

خامساً: التكرار والتوفّر الافتراضيّ

يحمل كل عقدة نظير في البلوكشين نسخة مُحدَّثة من السلسلة، فالتكرار متأصِّل في التصميم ولا يحتاج إلى إعداد إضافي؛ أمّا في القواعد التقليديّة فيتطلب الأمر نسخاً متماثلة أو إعداد Master/Slave لضمان التوفّر.

سادساً: الأمان ومنع التلاعب

يستند البلوكشين إلى اقتران التجزئات المتسلسلة بالتوقيعات الرقميّة، ما يجعل تزوير البيانات يتطلب سيطرة على غالبية العُقد (هجوم ٥١٪) وهو سيناريو مكلف وصعب[n].

Chowdhury, Mohammad Jabed Morshed, et al. "Blockchain versus database: A critical analysis." *2018 17th IEEE International conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE, 2018.

بينما يمكن لاختراق حساب مسؤول قاعدة بيانات أن يمنح المهاجم تحكّماً كاملاً بالسجلّ.

البيتكوين: أول تطبيق عملي لسلسلة الكتل

تُعتبَر عملة البيتكوين أول تطبيق فعلي وناجح لفكرة سلسلة الكتل، حيث طُرحت لأول مرة في ورقة علمية عام 2008 كتبها شخص (أو مجموعة) تحت الاسم المستعار Satoshi Nakamoto. وقد جاءت هذه الورقة لتقترح نظاماً نقدياً إلكترونياً قائماً على مبدأ الند للند (P2P)، يتيح إرسال المدفوعات الرقمية من طرف إلى آخر مباشرة دون الحاجة إلى مؤسسة مالية وسيطة.

يمكّن هذا النموذج من إجراء المعاملات المالية عبر الإنترنت بشكل آمن، ويحلّ محل الثقة بالطرف الثالث باستخدام آليات التشفير والإجماع. وقد بدأت الشبكة بما يُعرف بـ "كتلة الأصل" (Genesis Block) والتي احتوت على 50 بيتكوين، وكان بإمكان أي شخص تحميل البرنامج مفتوح المصدر والانضمام إلى الشبكة والمشاركة في التعدين أو إرسال واستقبال العملات الرقمية.

ورغم أن العملة الرقمية نفسها أثارت جدلاً واسعاً بسبب قدرتها على تمكين المعاملات المجهولة خارج نطاق رقابة الحكومات، إلا أن التكنولوجيا التي تقف خلفها، وهي سلسلة الكتل، أثبتت كفاءة غير مسبوقة وأصبحت تُطبَّق اليوم في مجالات مالية وغير مالية متعددة.

Crosby, Michael, et al. "Blockchain technology: Beyond bitcoin." *Applied innovation* 2.6-10 (2016): 71.

تطوّر سعر البيتكوين منذ إنشائه حتى 2025

عند إطلاقها في عام 2009، لم يكن لها أي قيمة تقريباً. لكن وبحسب أرشيف *arXiv* [2] وACM [3]، فإن البيتكوين شهدت نموا غير مسبوق، حيث:

مايو 2010: حدثت أول صفقة تجارية حقيقية باستخدام البيتكوين، حيث دفع أحد المبرمجين 10,000 BTC مقابل قطعتين من البيتزا.

2010: ظهر أول سعر رسمي للعملة في السوق عند حوالي 0.003 دولار أمريكي.

2013: ارتفعت قيمته بشكل لافت حتى وصلت إلى 1,000 دولار لأول مرة في تاريخه.

2017: شهدت العملة قفزة غير مسبوقة لتصل إلى 19,783 دولاراً بنهاية العام، ما أثار اهتماماً عالمياً واسعاً.

2018: تراجعت القيمة بشكل حاد إلى نحو 6,000 دولار.

2020 – 2021: تضاعف الاهتمام المؤسسي، لتصل العملة إلى ذروتها الجديدة في نوفمبر 2021 بقيمة قاربت 68,742 دولاراً.

2022: واجه السوق موجة هبوط شديدة، وانهار سعر البيتكوين إلى 16,000 دولار تقريباً.

2024 – 2025: بعد موجة من التذبذب، استعاد البيتكوين عافيته تدريجياً، وبلغ في منتصف 2025 سعراً قياسياً جديداً قدره 120,000 دولار أمريكي، وفقا لموقع Yahoo Finance.

هذا النمو يعكس الثقة المتزايدة بهذه التقنية، ويشير إلى قدرتها على إحداث تحول عميق في نظم الدفع والتبادل الرقمي.

Fabric هو نظام مفتوح المصدر، قابل للتوسعة والتركيب، يُستخدم في نشر وتشغيل شبكات البلوك تشين المُصرّح بها ويُعدّ أحد مشاريع Hyperledger التي تُشرف عليها مؤسسة Linux Foundation. تُعد Fabric أول نظام بلوكتشين قابل للتوسعة فعلياً لتشغيل التطبيقات الموزعة. وهو يدعم بروتوكولات إجماع معيارية، ما يتيح تخصيص النظام حسب حالات الاستخدام ونماذج الثقة الخاصة بكل حالة.

كما أن Fabric هو أول نظام بلوك تشين يُمكنه تشغيل التطبيقات الموزعة المكتوبة بلغات برمجة قياسية وعامة، دون الاعتماد الجوهري على عملة رقمية محلية. وهذا يُشكّل فرقاً جوهرياً عن منصات البلوكتشين الأخرى التي تتطلب كتابة "العقود الذكية" بلغات متخصصة، أو تعتمد على العملات المشفّرة.

Androulaki, Elli, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." *Proceedings of the thirteenth EuroSys conference*. 2018.

أهم ميزات Hyperledger Fabric

1. قابلية التوسعة والتعديل

تم تصميم Fabric ببنية معيارية، حيث تفصل المكونات مثل بروتوكول الإجماع، وسياسات العضوية، وتنفيذ العقود الذكية عن بعضها البعض، مما يتيح استبدال أو توسعة أي جزء حسب حاجات التطبيق

Androulaki, Elli, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." *Proceedings of the thirteenth EuroSys conference*. 2018.

هذا التصميم يسمح بفصل المهام الرئيسية داخل النظام (مثل التحقق من الهوية، تنفيذ العقود، وترتيب المعاملات) عن بعضها البعض، بحيث يمكن تخصيص كل مكون بما يتناسب مع متطلبات الأعمال المختلفة دون التأثير على بقية النظام، حيث يمكن استبدال بروتوكول الإجماع (مثل Raft أو Kafka) دون الحاجة إلى تعديل في العقود الذكية أو في إدارة العضوية. كذلك

2. دعم لغات برمجة عامة

يتيح Fabric كتابة العقود الذكية بلغات برمجة عامة مثل Go وJava وJavaScript، بدلاً من لغات متخصصة، ما يُسهل على المطورين اعتماد المنصة

Androulaki, Elli, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." *Proceedings of the thirteenth EuroSys conference*. 2018.

تُعد هذه الميزة من الابتكارات الجوهرية في Hyperledger Fabric، حيث لا يُلزم المطورين باستخدام لغات برمجة متخصصة (مثل Solidity في Ethereum)، بل يسمح لهم باستخدام لغات مألوفة ومعتمدة على نطاق واسع،

3. الخصوصية وقنوات الخصوصية

تدعم Fabric إنشاء قنوات خاصة ضمن الشبكة تُتيح مجموعة محددة من الأعضاء الوصول إلى بيانات محددة دون غيرهم، بما يضمن خصوصية المعاملات

هذا المفهوم يُعد من أبرز مميزات Hyperledger Fabric، ويُسمى بـ "Channels"، وهو يسمح بتقسيم شبكة البلوكتشين إلى شبكات فرعية صغيرة ومستقلة من حيث البيانات، دون الحاجة لإنشاء شبكات منفصلة فعلياً.

كل قناة Channel تشبه دفتر أستاذ خاص (Private Ledger) يتم فيه تسجيل المعاملات التي لا يمكن للأعضاء خارج القناة الاطلاع عليها. وهذا يوفر عزلاً تامًا للبيانات بين كيانات مختلفة تعمل ضمن نفس الشبكة العامة، مثل شركاء سلسلة توريد يعملون مع منافسين لهم في نفس النظام.

Androulaki, Elli, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." *Proceedings of the thirteenth EuroSys conference*. 2018.

4. مرونة في الحوكمة Governance Flexibility

لكل عقد ذكي سياسات توقيع (Endorsement Policies) خاصة به، ما يتيح تحديد من يمكنه تفعيله وفق توافق معين. هذا يضيف طبقة شفافية وثقة عالية، خاصة في بيئات متعددة المؤسسات

تتميز Hyperledger Fabric بآلية مرنة للحوكمة على مستوى الشبكة والعقود الذكية من خلال سياسات التأييد وهي القواعد التي تحدد أي المنظمات يجب أن تُصادق على تنفيذ العقد الذكي قبل أن تُقبل النتيجة كصحيحة وتُكتب في السلسلة.

تتيح هذه المرونة:

* تحكم دقيق في المشاركة: كل مؤسسة تحتفظ بصلاحيات مستقلة، ويمكن لكل عقد ذكي أن يُدار وفقاً لعلاقات الثقة بين الأطراف المعنية.
* حماية من التلاعب: لا يمكن لأي جهة بمفردها فرض نتيجة أو التلاعب بالعقد الذكي.
* دعم بيئات متعددة المؤسسات: في حال وجود شركات أو كيانات متعددة داخل الشبكة، يمكن تطبيق سياسات تضمن التوافق والتوقيع الجماعي قبل تنفيذ أي عملية حرجة.

البنى المعمارية في أنظمة البلوكتشين

أهم البنى المعمارية المعتمدة في تصميم أنظمة البلوكتشين تتعلق بطريقة تنفيذ المعاملات ومعالجتها:

* نموذج "الترتيب ثم التنفيذ" (Order-Execute)، والذي يُمثّل النهج التقليدي الذي تتبعه معظم سلاسل الكتل سواء المُصرّح بها أو غير المُصرّح بها.
* نموذج "نفّذ – ثم رتّب – ثم تحقق" (Execute-Order-Validate)، وهو النموذج المبتكر الذي اعتمدته منصة Hyperledger Fabric لمعالجة أوجه القصور في النموذج التقليدي.

أولاً: بنية "الترتيب ثم التنفيذ" (Order-Execute Architecture)

تعتمد معظم شبكات البلوكتشين التقليدية، سواء كانت مُصرّح بها (Permissioned)، أو غير مُصرّح بها (Permissionless) مثل Ethereum، على نموذج "الترتيب ثم التنفيذ". هذا النموذج يقوم على الخطوات التالية:

* ترتيب المعاملات: يتم جمع المعاملات من قبل أحد الأقران أو العقد، ثم يتم ترتيبها باستخدام بروتوكول إجماع مثل إثبات العمل (PoW) أو PBFT.
* تنفيذ المعاملات: بعد ترتيبها، يتم تنفيذ جميع المعاملات من قبل كل نظير في الشبكة بالتسلسل نفسه لضمان تطابق الحالة النهائية (state).

كما هو الحال في Ethereum:

* يجمع كل نظير مجموعة من المعاملات.
* يحاول حل لغز إثبات العمل (PoW) لتوليد كتلة جديدة.
* بعد النجاح، ينشر الكتلة باستخدام بروتوكول يدعى (Gossip Protocol).
* تتحقق بقية العقد من صحة الكتلة وتقوم بتنفيذ المعاملات بنفس الترتيب.

العيوب الأساسية في هذه البنية:

* أداء منخفض: نتيجة تنفيذ المعاملات على كل عقدة بشكل متسلسل، مما يؤدي إلى استهلاك زائد للموارد.
* اشتراط الحتمية: يجب أن تكون جميع المعاملات حتمية، أي أنها تُعطي دائماً نفس النتيجة عند تنفيذها على أي عقدة في الشبكة. هذا يُقيّد المطورين، لأنهم لا يستطيعون استخدام وظائف غير متوقعة مثل الوقت أو الأرقام العشوائية أو البيانات الخارجية
* غياب الخصوصية: كل نظير يرى وينفّذ جميع المعاملات، وهو أمر غير عملي في بيئات متعددة المؤسسات.

ثانياً: بنية Hyperledger Fabric — "نفّذ – ثم رتّب – ثم تحقق" (Execute–Order–Validate)

تُقدّم Hyperledger Fabric نموذجاً معمارياً مختلفاً يُعالج القيود السابقة، ويعتمد على فصل مراحل التنفيذ والترتيب والتحقق:

* Execute (التنفيذ): يتم تنفيذ المعاملات بشكل موازي من قِبل مجموعة مختارة من الأقران بحسب سياسات التأييد.
* Order (الترتيب): بعد التنفيذ، يتم ترتيب نتائج المعاملات المؤيَّدة باستخدام خدمة ترتيب.
* Validate (التحقق): يتم التحقق من توافق نتائج التأييد مع سياسة التأييد المُعتمدة، ثم يتم تحديث الحالة النهائية.

مكونات Hyperledger Fabric الأساسية

تعتمد بنية Hyperledger Fabric على تصميم معياري وقابل للتوسعة، حيث يتم الفصل بين الأدوار المختلفة للمكونات الرئيسية داخل الشبكة. يسمح هذا الفصل المعماري بتحقيق مستويات عالية من الأمان، القابلية للإدارة، وقابلية التوسع، مما يجعل Fabric خياراً مميزاً مقارنةً بأنظمة البلوك تشين التقليدية. وفيما يلي عرض تفصيلي لأهم مكونات هذا النظام.

1. المؤسسة (Organization)

تمثل المؤسسة كياناً منطقياً داخل الشبكة، وتشير عادة إلى جهة مشاركة في المعاملات مثل شركة أو هيئة حكومية أو كيان تنظيمي. كل مؤسسة تُشرف على بنيتها الخاصة من العقد (Nodes) ضمن شبكة Fabric، وتشمل عادة ما يلي:

* عُقد الأقران أو النظيرة (Peer Nodes): التي تُديرها المؤسسة وتشغلّها.
* هيئة الشهادات (Certificate Authority - CA): التي تصدر الهويات الرقمية لمستخدمي المؤسسة.
* مزود خدمة العضوية (Membership Service Provider - MSP): المسؤول عن تحديد السياسات الأمنية الخاصة بالمصادقة والتفويض.

تمكن هذه المكونات كل مؤسسة من إدارة هويات مستخدميها واستقلالية بنيتها التحتية، مع الحفاظ على التوافق والتكامل مع باقي مكونات الشبكة.

2. مزود خدمة العضوية (Membership Service Provider - MSP)

يُعد MSP المكوّن المسؤول عن إدارة الهويات الرقمية والتحقق منها داخل Fabric. يتم من خلاله تحديد الجهات الموثوقة في الشبكة، وذلك بالاعتماد على الشهادات الرقمية الموقعة من هيئات الشهادات.

تشمل مهام الـ MSP ما يلي:

* التحقق من صحة الهويات (Authentication).
* توقيع المعاملات والتحقق من تأييدها.
* تنظيم سياسات الثقة والتفويض بين المؤسسات.
* دعم نماذج متعددة، منها الشهادات الصادرة عن CA معروفة، أو هويات مجهولة (Anonymous Credentials).

يُكوَّن الـ MSP لكل مكون من مكونات الشبكة (مثل الأقران والعملاء وخدمة الترتيب)، ويُستخدم لضمان توحيد نظام التحقق داخل القناة الواحدة.

3. عقدة الأقران أو النظيرة(Peer Node)

عقدة الأقران هي مكون أساسي داخل شبكة Fabric، وهي مسؤولة عن تنفيذ العقود الذكية، والمحاكاة الأولية للمعاملات، والاحتفاظ بالسجل المشترك (Ledger). تؤدي عقدة Peer دورين رئيسيين يمكن تفعيلهما حسب الحاجة:

* عقدة مؤيدة (Endorser Node): تقوم بمحاكاة تنفيذ المعاملة استناداً إلى حالة السجل المحلي، وتنتج تأييداً (endorsement) يحتوي على نتائج القراءة والكتابة.
* عقدة ملتزمة (Committer Node): تتلقى الكتل المُجمعة من خدمة الترتيب، وتقوم بتثبيت المعاملات داخل دفتر الحسابات وتحديث حالة السجل.

كل Peer في الشبكة يُعد دائماً "عقدة ملتزمة"، بينما يُمنح دور "العقدة المؤيدة" فقط إذا نصّت سياسة التأييد للعقد الذكي على ذلك. هذا التصميم يُتيح مرونة كبيرة في توزيع الأدوار وتقليل الحمل الزائد على الأقران.

4. خدمة الترتيب (Ordering Service)

تُعد خدمة الترتيب المسؤولة عن تحقيق الترتيب الإجماعي (Total Order) للمعاملات في Fabric. وهي مكون مركزي يقوم باستقبال المعاملات المؤيدة من العملاء، وتجميعها في كتل مرتبة زمنياً، ثم توزيعها على العقد.

تشمل وظائفها الأساسية:

* استقبال المعاملات من العملاء بعد تأييدها.
* تجميع المعاملات في كتل ذات تسلسل زمني.
* إرسال الكتل إلى جميع الأقران من خلال بروتوكول Gossip.

لا تقوم خدمة الترتيب بتنفيذ المعاملات أو التحقق منها، بل تُركّز فقط على تنظيم تدفق البيانات، مما يجعلها مستقلة وقابلة للتخصيص من خلال بروتوكولات متعددة مثل Kafka.

5. العقد الذكي (Chaincode)

العقد الذكي هو البرنامج المسؤول عن تنفيذ منطق الأعمال داخل Fabric. وهو يُمثّل الوظائف التي تُحدد كيفية التعامل مع بيانات السجل، ويُنفذ بواسطة العقد المؤيدة خلال مرحلة المحاكاة.

يتضمن العقد الذكي:

* منطق المعالجة للمعاملات.
* التعليمات الخاصة بالقراءة والكتابة إلى دفتر الحسابات.
* سياسة التأييد الخاصة به (Endorsement Policy).

يمكن تطوير العقد الذكي باستخدام لغات برمجة متنوعة مثل Go، Java، وNode.js، مما يُتيح مرونة كبيرة في تكامل Fabric مع أنظمة الأعمال المختلفة.

6. دفتر الحسابات (Ledger)

دفتر الحسابات في Fabric هو المكوّن المسؤول عن حفظ كل المعاملات وحالة الشبكة الحالية. ويتكون من جزأين متكاملين:

* سجل المعاملات (Transaction Log): يُسجّل كل المعاملات التي تمت الموافقة عليها داخل الكتل، دون تعديل.
* حالة العالم (World State): تمثل القيم الحالية للبيانات، وتُخزن في قاعدة بيانات من نوع "مفتاح-قيمة" مثل LevelDB.

يوفّر هذا الفصل بين السجل والحالة مرونة في التعامل مع الاستعلامات، وتاريخية المعاملات، والتدقيق.

7. القنوات (Channels)

القنوات هي آلية فصل منطقية داخل Fabric، تُتيح لمجموعة محددة من المؤسسات تنفيذ المعاملات بشكل خاص وآمن، بعيداً عن بقية الشبكة.

كل قناة تملك:

* دفتر حسابات مستقل.
* مجموعة مشاركين محددة من المؤسسات.
* سياسات خاصة بالوصول والتأمين.

يُتيح هذا المفهوم إنشاء "شبكات صغيرة" داخل شبكة Fabric الرئيسية، مما يُعزز الخصوصية ويفصل البيانات الحساسة حسب الحاجة.

8. بروتوكولGossip

يُستخدم بروتوكول Gossip لتوزيع الكتل والمعاملات بين الأقران في الشبكة بشكل لامركزي وفعّال. يقوم كل Peer بنشر واستلام الكتل الجديدة من الأقران المجاورين، مما يُعزز الاتساق ويُقلل الاعتماد على مكونات مركزية.

9. التطبيق (Application)

يمثل التطبيق الطرف الذي يتفاعل مع شبكة Fabric من خلال واجهة برمجية (API). يقوم التطبيق بإرسال اقتراحات المعاملات إلى العقد المؤيدة، ثم يتلقى الردود والتأييدات، ويُرسل المعاملة النهائية إلى خدمة الترتيب.

يُستخدم التطبيق عادة من خلال أدوات تطوير رسمية مثل Fabric SDK for Node.js أو Java، ويُعد حلقة الوصل بين المستخدم النهائي وشبكة البلوك تشين.

## الواجهة الأمامية (Front-End)

عند بناء الواجهات الأمامية للتطبيقات خصوصاً في بيئات تطوير حديثة مثل Flutter، تظهر الحاجة إلى استخدام نموذج معماري يُساعد على تنظيم الكود وتبسيط التعامل مع البيانات والواجهة. من أبرز هذه المعماريات هي معمارية MVVM (Model-View-ViewModel)، والتي تُستخدم بشكل واسع في بناء تطبيقات الموبايل.

تأتي أهمية MVVM من كونها تُوفّر فصلاً واضحاً بين واجهة المستخدم والمنطق البرمجي الداخلي للتطبيق، مما يُسهل على المطوّرين صيانة التطبيقات، إعادة استخدام المكونات، والقيام بالاختبارات بكفاءة عالية.

يشير اختصار MVVM إلى ثلاث طبقات رئيسية:

* Model (طبقة البيانات): تمثّل البيانات الأساسية والبنية المنطقية للتطبيق، مثل قواعد البيانات، واجهات API، أو الكيانات (Entities).
* View (طبقة العرض): تمثّل واجهة المستخدم التي يتفاعل معها المستخدم مباشرة.
* ViewModel (طبقة المنطق وإدارة الحالة): تعمل كوسيط بين طبقة العرض وطبقة البيانات، وتحتوي على المنطق الخاص بالواجهة وتنسيق البيانات للعرض.

أهم مميزات معمارية MVVM:

1. فصل واضح بين الواجهة والمنطق حيث أنَّ الواجهة لا تتعامل مباشرة مع البيانات، وكل طبقة مسؤولة عن وظيفة محددة، مما يُسهّل الصيانة والتوسعة.

2. الربط البياني حيث يتم ربط واجهة المستخدم مباشرة بالخصائص الموجودة في طبقة المنطق وإدارة الحالة، حيث أي تغيير في البيانات ينعكس تلقائياً على الواجهة، دون الحاجة إلى كود إضافي.

3. دعم قوي لاختبار الوحدات (Unit Testing)

* طبقة المنطق وإدارة الحالة تحتوي على منطق التطبيق ويمكن اختباره دون الحاجة إلى تشغيل الواجهة.
* يساعد على ضمان جودة الكود وتقليل الأخطاء البرمجية.

4. إعادة استخدام الكود (Reusability)

* يمكن استخدام نفس طبقة المنطق وإدارة الحالة في أكثر من واجهة، أو عبر منصات مختلفة.
* يُمكن فصل واجهات الهاتف والويب، لكن استخدام نفس منطق العمل.

5. سهولة الصيانة والتطوير

* بفضل التقسيم المنطقي، يستطيع المطوّرون تعديل طبقة دون التأثير على الطبقات الأخرى.
* تسهّل العمل في الفرق البرمجية: المطوّرون يركزون على طبقة المنطق وإدارة الحالة، والمصمّمون على طبقة العرض.

6. مرونة في إدارة الحالة (State Management)

MVVM يوفر نموذجاً واضحاً لإدارة الحالة، خصوصاً في التطبيقات التي تحتوي على حالات معقّدة.

7. سهولة التوسعة والدعم للميزات المستقبلية

يسهل إضافة ميزات جديدة أو تحسين الواجهة دون إعادة هيكلة التطبيق بالكامل.

# تصميم النظام

## مقدمة:

يتكون نظام بيع وشراء المنتجات المستعملة من مجموعة مكونات مترابطة تعمل بتناغم لتحقيق تجربة استخدام مرنة وآمنة. ينقسم النظام إلى خمس طبقات رئيسية:

1. **الواجهة الأمامية (Front-End)**:  
   تمثل نقطة التفاعل بين المستخدم والنظام، حيث يمكن للمستخدم تنفيذ العمليات مثل تسجيل الدخول، استعراض المنتجات، إرسال الطلبات، وكتابة المراجعات. تم تطوير الواجهة باستخدام تقنيات حديثة تدعم الأجهزة الذكية.
2. **الواجهة الخلفية (Back-End)**:  
   تتولى معالجة الطلبات الواردة من الواجهة الأمامية، وتحتوي على منطق الأعمال الذي يتحقق من صحة البيانات، ينفذ العمليات (مثل إنشاء الطلبات أو معالجة الدفع)، ويتواصل مع قواعد البيانات والأنظمة المساعدة. تعتمد البنية على نمط **Modular Monolithic**، ما يتيح فصل الوحدات المنطقية داخليًا مع الحفاظ على وحدة التطبيق.
3. الأنظمة الداعمة (Supporting Systems):
   * **قاعدة البيانات**: لتخزين المعلومات الخاصة بالمستخدمين، المنتجات، الطلبات، المراجعات... إلخ.
   * **نظام الذكاء الاصطناعي (AI)**: لمعالجة المراجعات وتحليل المشاعر وتوليد تقييمات تلقائية.
   * **نظام البلوكتشين**: لتوثيق المعاملات المالية والعقود الذكية بطريقة غير قابلة للتلاعب.
   * **RabbitMQ**: وسيط رسائل يستخدم لتمرير البيانات بين الوحدات المختلفة بشكل غير متزامن، خاصة إلى AI وBlockchain.
4. لوحة التحكم:

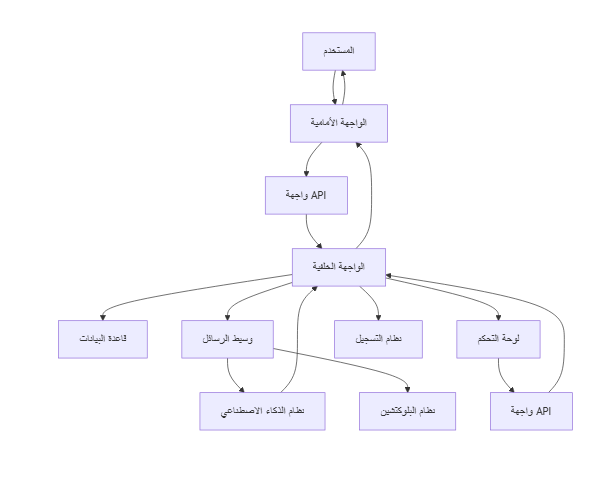
تُعد لوحة التحكم أداة مركزية لإدارة النظام، وتُستخدم من قِبل مسؤولي المنصة لمتابعة سير العمليات والإشراف على المحتوى. تتيح اللوحة مراقبة الحسابات، مراجعة المنتجات والطلبات، إدارة المراجعات والتقييمات، بالإضافة إلى عرض تقارير الأداء والإحصاءات المتعلقة بالنشاطات داخل المنصة. صُممت الواجهة لتكون تفاعلية وسهلة الاستخدام.

1. نظام التسجيل Log system:

يتولى هذا النظام تسجيل كافة الأحداث والأنشطة التي تتم داخل المنصة، مثل عمليات الدخول، إنشاء الطلبات، التعديلات على المنتجات، واستدعاء الأنظمة الداعمة. يُستخدم نظام التسجيل لأغراض التتبع، التدقيق الأمني، وتحليل الأداء، كما يوفّر سجلات زمنية دقيقة تساعد في الكشف عن الأعطال أو السلوكيات غير الاعتيادية. وقد تم اعتماد آلية تسجيل مرنة وقابلة للتوسعة، مع إمكانية الربط بأنظمة مراقبة وتحليل خارجية عند الحاجة.

سير العملية بشكل مبسّط:

1. يُرسل المستخدم طلبًا (مثلاً: "إنشاء طلب شراء") من خلال الواجهة الأمامية.
2. ينتقل الطلب إلى الواجهة الخلفية حيث يتم التحقق منه ومعالجته.
3. تتفاعل الواجهة الخلفية مع القاعدة البيانية لتخزين أو جلب البيانات.
4. عند الحاجة، تُرسل بعض العمليات إلى الأنظمة المساعدة مثل AI أو Blockchain عبر RabbitMQ.
5. تُعاد الاستجابة إلى الواجهة الأمامية لعرضها على المستخدم.



## نظام الواجهة الخلفية:

اعتماد تصميم الخادم الخلفي على نمط Modular Monolithic مع تطبيق مبادئ Clean Architecture على كل وحدة وظيفية بشكل مستقل، مما يضمن فصلاً واضحاً للمسؤوليات وسهولة الصيانة. تتولى وحدة Shared.contract تنسيق تفاعل الوحدات المختلفة عبر واجهة برمجية موحّدة، بينما يُستخدم نمط Mediator كوسيط داخلي بين الوحدات وعبر طبقات التطبيق عند تنفيذ أوامر واستعلامات (CQRS)، مما يعزز الانفصال بين الطبقات ويقلل من الاعتماد المباشر بين المكونات. بهذا التصميم، يمكن تطوير ونشر وتحديث كل وحدة بشكل منفصل ضمن نفس المنشور الخادمي، مع الحفاظ على بنية مترابطة ومرنة تستجيب للتغيرات والمتطلبات المستقبلية بسهولة.

## نظام نموذج توقع التقييم:

انطلاقًا من الأبحاث التي تمّ دراستها بالدراسة المرجعية في مجال التقييم المعتمد على المراجعات النصيّة، تمّ العمل على بناء وتطوير مجموعة من المنهجيات بهدف مقارنة أدائها والوصول إلى حلّ أمثلي يخدم هدف المشروع. من أجل تدريب أو تقييم دقّة نموذج التوقّع، لا بُدّ من وجود مجموعة بيانات موسومة بتقييمات رقمية من 1–5 مرتبطة بمراجعات نصّية لمنتجات على منصات التجارة الإلكترونية، إلا أن هذه المُعطيات غير متوفرة باللغة العربية. لذلك، لجأنا إلى استخدام مجموعتي البيانات HARD وLARB اللتين تحتويان على مراجعات لكتب وفنادق باللغة العربية مع تقييمات رقمية من 1–5، إذ تلبي هاتان المجموعتان الاحتياج الأساسي من حيث تنوّع الأساليب التعبيرية وحجم البيانات. ولم يتم اللجوء إلى ترجمة مجموعات بيانات أجنبية بسبب مخاطر تشويه المعاني الناتجة عن الترجمة الآلية، فضلاً عن تكلفة المعالجة الحسابية الإضافية وافتقار الترجمات المجمعة إلى اتساق الأسلوب والمصطلحات التقنية بين السياقات الثقافية المختلفة.

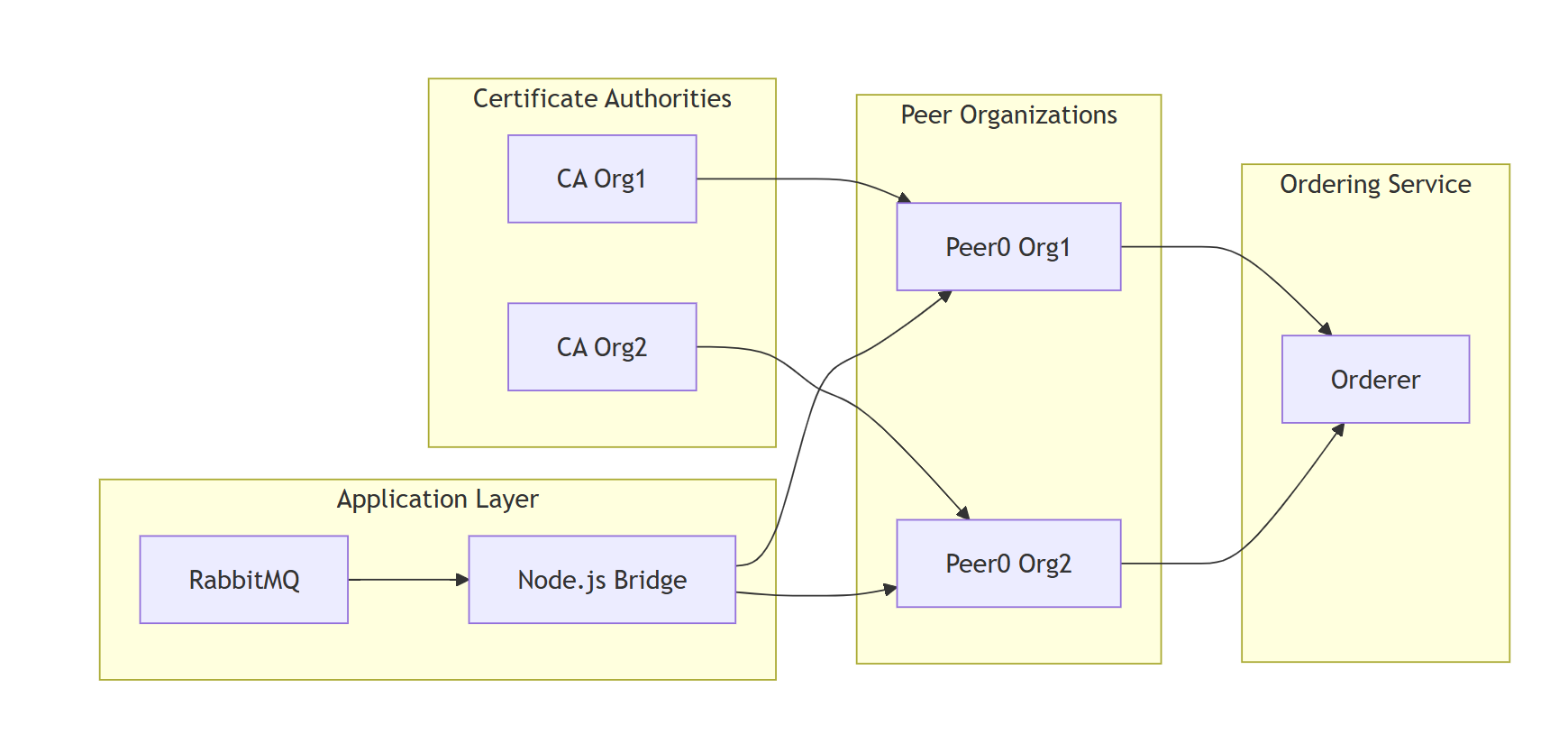
### المنهجيات التجريبية

يشتمل العمل على تجريب عدد من المناهج المتنوّعة لمقارنة دقتها وأدائها:

1. **SVM مع تمثيل TF–IDF**
   * نهج إحصائي تقليدي يعالج المراجعات النصيّة عبر تمثيلها رياضيًا باستخدام TF–IDF، ثم يصنّفها بواسطة خوارزمية دعم المتجهات.
   * تم ضبط معايير النموذج عبر البحث الشبكي (Grid Search) للوصول إلى أفضل توزيع للمعاملات وتقليل خطأ التصنيف.
2. **نماذج المحوّلات (Transformers)**
   * استخدام MARABERT المدرّب مسبقًا للتصنيف المباشر، مع إضافة طبقات تصنيف نهائية (Dense Layers) تعتمد على إخراج الـ[CLS] token.
   * دمج تمثيلات ARBERT مع شبكة BiLSTM ثنائية الاتجاه لتعميق فهم العلاقات السياقية داخل الجمل.
3. **التعلم القليل البيانات (Few-Shot Learning) عبر LLM**
   * استثمار قدرات نموذج لغوي كبير (LLM) بأسلوب التلقين (Prompting) لاقتراح تقييم رقمي من 1–5 بناءً على مراجعة نصّية، دون الحاجة إلى تدريب ضخم.
   * إعداد قوالب prompts تضم أمثلة قليلة مُمَثلة بعناية لضمان وضوح المطلوب من النموذج وقدرته على التعميم.

مع اعتماد مقياس الخطأ متوسط مربعات الأخطاء (Mean Squared Error - MSE) لتقييم أداء النماذج، نظرًا لأنّ المسألة أقرب إلى تقريبٍ لوجيستي (Logistic Regression) أكثر منها تصنيفًا صريحًا؛ مما يتيح قياس مدى انحراف التوقع العددي عن القيمة الحقيقية بصورة دقيقة.

## نظام البلوكتشين:



نظرة عامة شاملة على مكوّنات شبكة Hyperledger Fabric

تتكون شبكة Hyperledger Fabric من مجموعة متكاملة من المكوّنات، كل مكوّن يلعب دوراً حاسماً في ضمان تكامل العمليات وسلامة المعاملات، من إدارة الهويات إلى تنفيذ المعاملات وتحديث دفاتر الأستاذ. وفيما يلي تفصيل موسّع لأبرز هذه المكونات:

### سلطات الشهادات (Certificate Authorities – CAs)

تُعد سلطات الشهادات حجر الأساس في منظومة الهوية داخل Fabric، إذ تُصدر وتدير الشهادات الرقمية التي تُستخدم لمصادقة العقد والمستخدمين والخدمات. يحتوي النظام على سلطة شهادات مستقلة لكل مؤسسة لضمان الفصل الكامل للثقة.

CA Org1: تُمثّل المصدر الرسمي للثقة في المؤسسة الأولى، حيث تُولّد شهادات X.509 لجميع الكيانات (عقد نظيرة، مدراء، مستخدمون) ، تُدير دورة حياة الشهادة بما في ذلك التجديد والإلغاء عند الحاجة، وتُتيح لكل مكوّن داخل Org1 التفاعل بشكل آمن من خلال بروتوكول TLS.

:CA Org2 تُقدّم نفس الوظائف للمؤسسة الثانية، لكن بشكل مستقل تماماً.

أهمية هذه الطبقة تكمن في أنها تُمثّل جذر الثقة (Root of Trust) في شبكة Fabric، وكل معاملة أو اتصال لا يمكن تنفيذه ما لم يكن مدعوماً بهوية صادرة من CA معترف بها.

### المنظمات النظيرة (Peer Organizations)

تمثّل كل مؤسسة مجموعة من العقد النظيرة (Peers) المسؤولة عن تنفيذ المعاملات، تخزين دفتر الأستاذ، وتوفير خدمات التأييد (Endorsement).

Peer0 Org1: تُعد العقدة النظيرة الرئيسية للمؤسسة الأولى، حيث تحتفظ بنسخة من دفتر الأستاذ وتنفّذ العقود الذكية، وتشارك في عملية التوافق عبر توقيع المعاملات بناءً على منطق العمل المُشغَّل، وتستخدم قاعدة بيانات الحالة العالمية (World State) لتقديم استعلامات فورية.

Peer0 Org2: تلعب نفس الدور داخل المؤسسة الثانية، وتُعد جزءاً من الشبكة الموزّعة مما يُعزّز اللامركزية والتكرار، وتُساعد في تنفيذ العقود الذكية متعددة المؤسسات (Multi-Org Chaincodes)، وتوفّر الحماية من الأعطال عبر توفير نسخة متطابقة من دفتر الأستاذ.

هذه العقد هي القوّة الحسابية الأساسية داخل الشبكة، وجميع المعاملات تمرّ من خلالها إمّا للتأييد أو للتحديث. كما تُعد مسؤولة عن الالتزام بآليات سياسة الإقرار (Endorsement Policy).

### خدمة الترتيب (Ordering Service)

تمثّل عقدة الترتيب (Orderer) نقطة التنسيق المركزية المسؤولة عن تنظيم المعاملات داخل الشبكة:

Orderer: ولديه العديد من المهام:

يجمع المعاملات الموقعة من Peers ويضعها في ترتيب زمني دقيق.

يُنشئ كُتل تحتوي على مجموعة من المعاملات ويُرسلها إلى العقد النظيرة.

يضمن سلامة التسلسل الزمني (Chronological Order) وبالتالي عدم التلاعب أو التكرار.

يدعم خوارزمية Solo في البيئات التطويرية، مع إمكانية الانتقال لاحقاً إلى Raft أو Kafka لضمان التوفر العالي وتحمل الأعطال.

تُعد هذه الخدمة مسؤولة عن "إغلاق المعاملات" بشكل نهائي، وهي الجهة الوحيدة المصرح لها بإنشاء كتل جديدة في السلسلة.

### طبقة التطبيق (Application Layer)

تُستخدم هذه الطبقة لربط النظام بالبُنى الخارجية، حيث تُوفّر وسيلة تواصل بين نظام إدارة التجارة وشبكة البلوكشين.

**جسر Node.js:** يعمل كوسيط (Middleware) بين التطبيقات التقليدية ASP.NET وشبكة Fabric، حيث يُحوّل طلبات REST إلى صيغ متوافقة مع بروتوكولات Fabric، يدير الاتصالات مع العقد النظيرة وعقدة الترتيب من خلال Fabric SDK، ويُنفّذ إدارة الشهادات وهويات المستخدمين، ما يعزز الأمان والسيطرة.

**RabbitMQ:** يُستخدم كصف رسائل لفصل طبقة الخلفية عن البلوكشين، فهو يُوفّر نظاماً مرناً وموثوقاً للتواصل غير المتزامن، ويمكّن من التعامل مع حمل عالٍ من المعاملات دون التأثير على الأداء العام، فهو يُحافظ على الرسائل حتى يتم التأكيد على استلامها ومعالجتها من قِبل الجسر.

تضمن هذه الطبقة أن التفاعل مع شبكة البلوكشين يتم بطريقة سلسة، مرنة، وقابلة للتوسّع، دون الحاجة لتعقيد تفاصيل البروتوكول الأصلي لـ Fabric.

### تفاعل المكوّنات في الشبكة

تدفّق الشهادات: تبدأ من سلطات الشهادات بإصدار الهويات حيث تُستخدَم من قبل العقد والمستخدمين لإنشاء الاتصال الآمن باستخدام TLS.

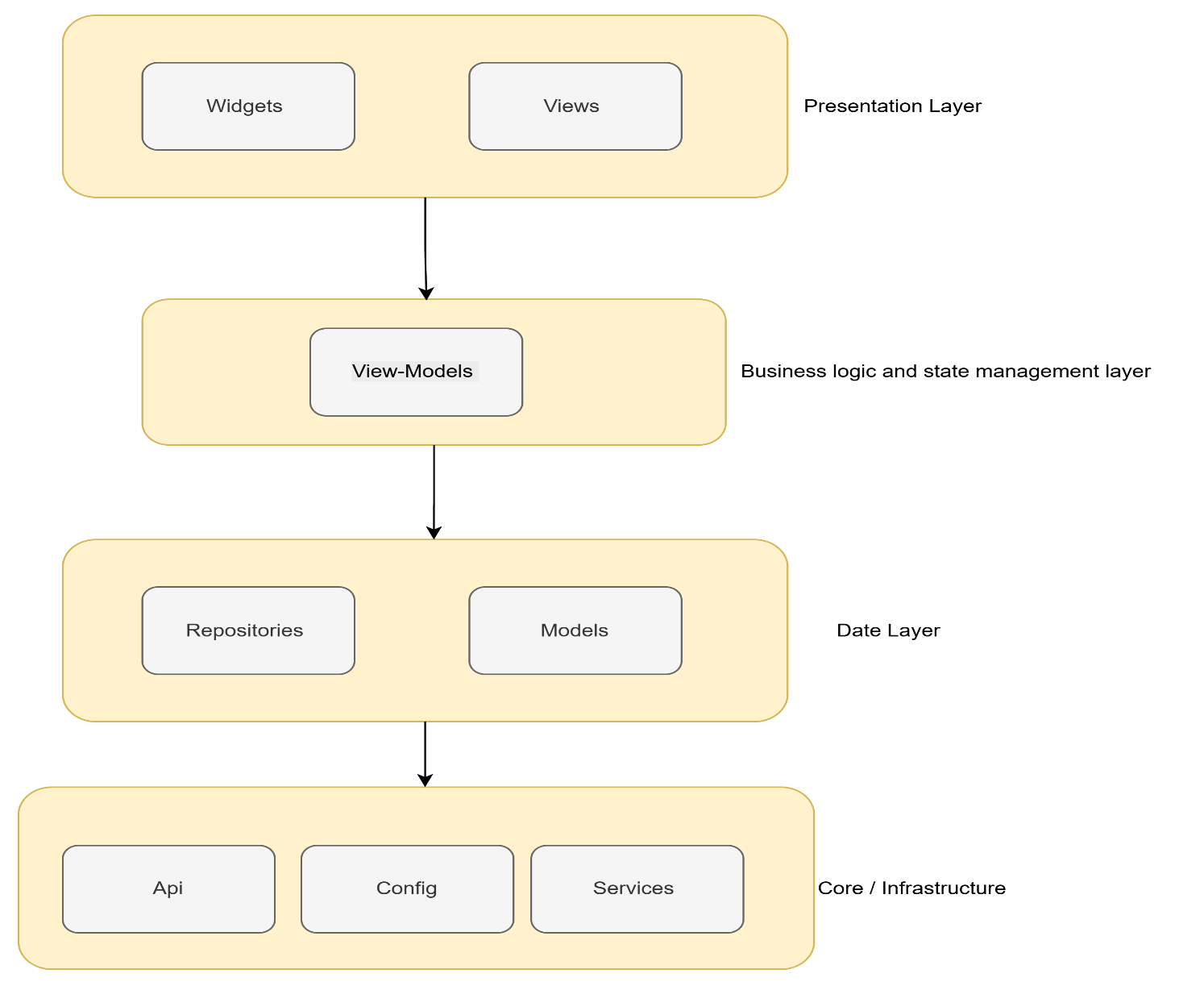
تدفّق المعاملات: تُنشأ المعاملة في الخلفية حيث تُرسَل إلى RabbitMQ ليقوم الجسر بمُعالجها ومن ثم تقوم العقد النظيرة Peers بتأييدها ومن ثم ترسل إلى عقد الترتيب ليتم ترتيبها ومن ثمَّ تُوزَّع على الشبكة.

أمان الشبكة: تستند حماية الشبكة إلى بروتوكول TLS المقترن بشهادات X.509 تُصدِرها سلطات شهادات موثوقة. يَتحقّق كلُّ طرف من هوية الطرف المقابل عبر مصادقة متبادلة قبل بدء التبادل، ثم تُشفَّر جميع البيانات المنقولة، ما يحول دون التنصّت أو انتحال الهوية ويحافظ على سرّية الرسائل وسلامتها.

اللامركزية: التوزيع عبر مؤسستين مستقلّتين يضمن عدم وجود نقطة فشل وحيدة (Single Point of Failure) حيث إذا تعرّضت العقدة النظيرة في مؤسسة ما لعطل، تظلّ العقدة النظيرة الثانية في المؤسسة الأخرى قادرةً على معالجة المعاملات، والعكس صحيح وتُلزِم سياسات القناة موافقة نظير من كلتا المؤسستين على أي معاملة، فلا تنفرد جهة واحدة بالتحكّم في دفتر الأستاذ أو تعديله.

## نظام الـ Front-end

فيما يلي مخطط يوضح تصميم نظام الواجهة:



شرح طبقات بنية النظام:

Presentation Layer (طبقة العرض): تمثل هذه الطبقة الواجهة الأمامية للتطبيق، أي كل ما يراه المستخدم ويتفاعل معه مباشرة، حيث يقوم بعرض البيانات القادمة من طبقة المنطق وإدارة الحالة، وتقوم بجمع مدخلات المستخدم.

وتتكون من جزئين:

* Views: وهي عبارة عن الشاشات التي تظهر للمستخدم والتي يتفاعل معها المستخدم.
* Widgets: وهي عبارة عن مكونات قابلة لإعادة الاستخدام

Business Logic and State Management Layer (طبقة المنطق وإدارة الحالة): الطبقة المسؤولة عن اتخاذ القرارات ومعالجة البيانات وتنسيق التفاعل بين الواجهة ومصادر البيانات، حيث تقوم بتنفيذ منطق الأعمال الخاص بكل خاصية أو خدمة (مثل حساب إجمالي سلة المشتريات، التحقق من شروط، إلخ)، وتتواصل مع مستودعات البيانات لجلب أو إرسال المعلومات، كما إنّها مسؤولة عن إدارة حالة التطبيق.

Data Layer (طبقة البيانات): هذه الطبقة تفصل منطق البيانات عن المنطق المرئي للتطبيق.

تتكون من جزئين:

* Repositories: تتعامل مع جلب البيانات من الـ API أو التخزين المحلي وتقديمها بشكل موحد لـ طبقة المنطق وإدارة الحالة.
* Models: تمثل الكيانات والبيانات مثل المنتجات، المستخدمين، الطلبات، وتُستخدم للتحويل بين البيانات وهيئة الـ JSON.

Core / Infrastructure (الطبقة التحتية الأساسية): تضم الخدمات العامة والمرافق الأساسية التي تستخدمها باقي الطبقات

تتكون من:

* Api: تنفيذ الطلبات الشبكية (GET/POST) والتعامل مع الاستجابات والخطأ.
* Config: إدارة الإعدادات العامة (الألوان، المسارات، عناوين الاتصال، إلخ).
* Services: تقديم خدمات مساعدة مثل إدارة التخزين الآمن، مع حفظ رموز المصادقة (tokens) واسترجاعها بطريقة مشفَّرة ومعزولة،والربط مع مزوّدي طرف ثالث (مثل الدفع).

### تفاعل المكوّنات في نظام الواجهة

عندما يقوم المستخدم بنقر أو إدخال بيانات على إحدى الشاشات (Views). تُحوَّل هذه الأحداث فوراً إلى طبقة المنطق وإدارة الحالة المعنيّة بتلك الشاشة؛ فتقوم هذه الطبقة بقراءة آخر حالة مخزَّنة لديها، وتُطبِّق قواعد الأعمال (تحقُّق صلاحية الحقول، حساب مجاميع، إلخ). إذا احتاجت إلى مصادر خارجية تستدعي Repository مناسباً عبر دوالٍ غير متزامنة:

* Repository تفحص أوّلاً ما إذا كانت البيانات متاحة في Services (مثل التخزين الآمن أو الذاكرة المؤقتة).
* عند غياب البيانات أو انتهاء صلاحيتها يُمرّر الطلب إلى API Client داخل طبقة Core، مستعيناً بـ Config لاختيار عنوان النهاية (Endpoint) الصحيح أو قراءة مفتاح واجهة برمجة التطبيقات.
* يضيف الـ API Client رأس المصادقة (Bearer Token) المستخرَج من Secure Storage Service، ثم يرسل طلب HTTP ويُفسِّر الاستجابة.
* تُحوَّل الاستجابة الخام إلى Model مضبوط، يعود إلى Repository ثم إلى طبقة المنطق وإدارة الحالة.
* تُحدِّث طبقة المنطق وإدارة الحالة حالتها الداخلية وتنشر إشعاراً (notifyListeners / state.update)، ما يدفع Flutter لإعادة بناء الواجهات أو المكونات اللازمة المرتبطة، فتنعكس البيانات الجديدة على الواجهة فوراً.

## لوحة التحكم:

تصميم وتطوير واجهة لوحة التحكم كمشروع مستقل باستخدام React وTypeScript، مع اتباع نهج الهندسة المعتمدة على المكونات (Component-Based Architecture) الذي يضمن بيئة غنية بالمكونات القابلة لإعادة الاستخدام وتنظيم شيفرة نظيفة (Clean Code). بحيث تركز المنهجية على فصل منطق واجهة المستخدم (UI) عن إدارة الحالة (State Management) واسترجاع البيانات (Data Fetching)، مع تطبيق التحميل الكسول للمسارات (Lazy Loading of Routes) لتعزيز سرعة التحميل الأولي وتحسين تجربة المستخدم.

مع اعتماد التصميم المتجاوب أولًا للجوال (Mobile-First Responsive Design) لضمان عمل الواجهة بسلاسة على مختلف الأجهزة، ودمج مخططات تحليلية تفاعلية (Interactive Analytical Charts) لعرض مؤشرات الأداء الرئيسية مثل حجم المبيعات وحالة الطلبات بصورة بصرية واضحة. ولضمان أعلى معايير الجودة.

# الفصل السادس الأدوات واللغات المستخدمة

## Back-End:

فيما يلي عرض تفصيلي للأدوات والتقنيات المُعتمدة في بناء مكوِّنات المشروع، مع التركيز على أدوارها ومساهمتها في تحقيق متطلبات الأداء والقابلية للتوسّع:

**إطار عمل .NET 8.0**  
– منصة تطوير مفتوحة المصدر من مايكروسوفت تدعم إنشاء تطبيقات عالية الأداء وقابلة للتوسع.  
– تعتمد على لغة C# مع إمكانات إدارة الذاكرة الآمنة (Garbage Collection) وتحسينات في سرعة التنفيذ.  
– تتكامل مع حاويات Docker وKubernetes لنشر وحدات الخلفية بشكل مرن داخل بيئات Cloud Native.  
– توفر مكاتب جيل حديث (Modern Libraries) للتعامل مع HTTP، الأمان، والهوية (Identity).

**Git**

هو نظام تحكم في الإصدارات (version control system) مفتوح المصدر يستخدم لتتبع التغييرات في الرماز المصدري (source code) أثناء تطوير البرمجيات. حيث يسمح Git لمطورين متعددين بالتعاون في مشروع واحد من خلال إدارة التغييرات التي تم إجراؤها على الرماز المصدري، مما يجعل من السهل تنسيق العمل وتتبع التغيرات ضمن المشروع.

**GitHub**

هي منصة تعمل ضمن الويب توفر أدوات للتحكم في الإصدارات والتعاون بين المطورين لتطوير البرامج، حيث أنها مبنية على نظام التحكم في الإصدارات Git. تسمح هذه المنصة بتخزين الكود البرمجي ومشاركته بين مجموعة من المطورين كما تقدم مجموعة من الأدوات للتعامل مع بيئات التطوير المستندة إلى السحابة (cloud) وإستضافة مواقع الويب الثابتة.

**Docker**

هي منصة مفتوحة مصممة لأتمتة عملية نشر التطبيقات وتوسيع نطاقها (scaling) وإدارتها في حاويات خفيفة الوزن (containers). الحاوية (container) هي بيئة معزولة تجمع بين التطبيق وجميع تبعياته، مما يسمح بتشغيله بشكل متسق عبر بيئات حوسبة مختلفة، مثل أنظمة تشغيل مختلفة أو مزودي الخدمات السحابية.

**ASP.NET Web API**

هو إطار عمل طورته شركة Microsoft يتيح إنشاء خدمات تعتمد على HTTP. يمكن استخدام هذه الخدمات بواسطة مجموعة متنوعة من العملاء، بما في ذلك متصفحات الويب وتطبيقات الأجهزة المحمولة وخدمات الويب الأخرى. إنه جزء من إطار عمل ASP.NET الأكبر وهو مصمم خصيصًا لإنشاء واجهات برمجة تطبيقات RESTful.

**RabbitMQ**  
– وسيط رسائل (Message Broker) مبني على بروتوكول AMQP لإرسال ومعالجة الرسائل غير المتزامنة.  
– يدعم نماذج التوجيه المتقدمة (Direct, Topic, Fanout, Headers) لفصل الخدمات ومعالجة الأحمال المرتفعة.  
– يتيح تأكيدات وصول موثوقة (Publisher Confirms & Consumer Acknowledgements) لضمان عدم فقدان البيانات.  
– يسهّل تكامل وحدة الذكاء الاصطناعي وخدمات الدفع والبلوكشين بعيداً عن مسار تنفيذ الطلبات المُستخدم.

**SQL Server**  
– نظام قواعد بيانات علائقية من مايكروسوفت يُقدم أداءً عالياً في التعامل مع المعاملات (OLTP) والاستعلامات المعقدة (OLAP).  
– يدعم إنشاء Stored Procedures، Views، وIndexing متقدم لتحسين زمن الاستجابة.  
– يوفر ميزات الأمان المشفرة (Transparent Data Encryption) واسترجاع البيانات بعد الكوارث (Disaster Recovery).

**MediatR**  
– مكتبة تطبيق نمط CQRS (Commands & Queries) وفصل طبقات النقل عن منطق الأعمال.  
– تعتمد على مبدأ “مُرسل الأوامر” (Mediator Pattern) لتوجيه الطلبات في التطبيق دون ارتباطٍ مباشر بين الوحدات.  
– تُسهّل اختبار الوحدة (Unit Testing) بفضل قابلية فصل التبعيات (Dependency Injection) والتحكم بسيناريوهات المعالجة.

**Serilog**  
مكتبة تسجيلٍ هيكليٍّ مفتوحةُ المصدر، تُستخدم ضمن بيئة .NET لتوثيق الأحداث والبيانات التشخيصيّة بصيغةٍ منظّمةٍ قابلةٍ للتحليل والاستعلام. تعتمد Serilog على بنيةٍ مرنةٍ تُمكّن المطوّرين من تحديد مستويات التسجيل (مثل: Debug، Information، Warning)، وإثراء السجلات بخصائص سياقيّة كمعرّف الطلب واسم المستخدم. كما تدعم الإرسال إلى وجهات متعدّدة مثل وحدة التحكم (Console)، والملفّات الدوريّة، وخوادم التحليل مثل Seq. تساهم Serilog في تعزيز موثوقيّة التطبيق، وتسهيل تتبّع الأخطاء، وتحسين أداء عمليّات المراقبة والتشخيص.

**Seq**  
نظام تسجيلٍ وتحليلٍ مركزيّ، يُستخدم لتجميع السجلات من مختلف التطبيقات والخدمات وتحليلها باستخدام واجهة استعلام متقدّمة شبيهة بـ SQL. يُتيح Seq البحث الفوريّ في السجلات، وتصميم لوحات معلومات (Dashboards) لمراقبة مؤشّرات الأداء، وإعداد تنبيهات تلقائيّة عند تجاوز شروط محدّدة. يستقبل السجلات الهيكليّة المُرسلة من مكتبات مثل Serilog بصيغة JSON، ويُعد أداة فعّالة لمراقبة الأنظمة وتحسين الاستجابة للأخطاء في بيئات الإنتاج المعقّدة.

## لوحة التحكم:

**React**

هي مكتبة JavaScript مفتوحة المصدر، تُستخدم لبناء واجهات المستخدم، خاصة لتطبيقات الويب. تم تطويرها بواسطة شركة Facebook، وتستخدم على نطاق واسع لإنشاء واجهات مستخدم ديناميكية وتفاعلية. تتبع React البنية القائمة على المكونات، مما يساعد المطورين على بناء مكونات واجهة المستخدم التي يمكن إعادة استخدامها وتكوينها معًا لإنشاء واجهات مستخدم معقدة.

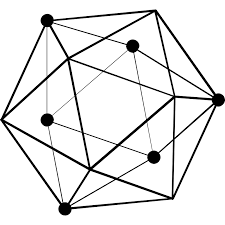
## AI:

لغات النمذجة مثل Python، مكتبات

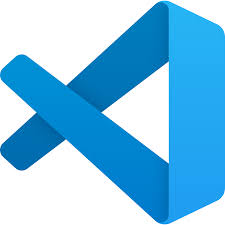
## Blockchain:

VMware Workstation منصّة افتراضيّة لسطح المكتب تمكّن من تشغيل أنظمة تشغيل متعددة في آنٍ واحد على جهاز واحد، مع عزل كامل للبيئات. تُستخدم لتهيئة بيئات تطوير واختبار متوازية.

Ubuntu توزيعة لينكس مستقرة وسهلة الاستخدام توفّر مستودعات حزم واسعة، ودعماً طويل الأمد (LTS)، وتكاملاً جيداً مع أدوات التطوير والبنية التحتية الحديثة مثل Docker وGit.



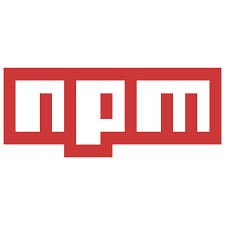
Hyperledger Fabric منصّة بلوكشين مؤسّسية مُصرَّح بها تتيح إنشاء شبكة تضم منظمات وعُقد، ونشر عقود ذكيّة للتكامل مع التطبيقات. تتميّز ببنية معيارية تسمح باختيار آليات الإجماع، وإدارة الهويّات عبر MSP، وعزل البيانات باستخدام القنوات، ما يجعلها قابلة للتوسّع ومناسبة لسيناريوهات متعدّدة.



Visual Studio بيئة تطوير متكاملة من مايكروسوفت مخصّصة للتطبيقات المكتبية والويب والخوادم والهواتف. تدعم لغات وأطراً متعددة (مثل C#, .NET, go، Python).



Node.js: محرّك لجافاسكربت يعمل بالوقت الحقيقي، مفتوح المصدر، ومناسب لبناء تطبيقات متعدّدة المنصّات وخوادم ويب قابلة للتوسّع.

 NPM: هو نظام إدارة الحزم الأكثر شعبية للغة البرمجة جافا سكربت، وهو مدير الحزم الافتراضي لبيئة تشغيل جافا سكربت Node.js.

CURL: هي الأداة التي تتميز بواجهة سطر أوامر، ودورها بشكل عام هو نقل الملفات عبر شبكة اتصال، وهي تقبل العديد من البروتوكولات مثل FPT وبروتوكول HTTP.



Express.js: هو إطار عمل لتطبيق الخلفية لـnode.js وهو مصمم لبناء تطبيقات الويب وواجهات برمجة التطبيقات. وقد تم تسميته بإطار عمل الخادم المعياري الفعلي لـ Node.js.



Go (Golang) لغة برمجة مفتوحة المصدر طوّرتها Google، صُمِّمت لتكون بسيطة وسريعة مع دعم قويّ للتشغيل المتوازي لبناء خدمات وأنظمة موثوقة على نطاق واسع، تتميّز بسرعة الترجمة، ومخرجات تنفيذية صغيرة، ومكتبة قياسية غنيّة.

## Front-End:

Android Studio هي بيئة التطوير المتكاملة (IDE) الرسمية لتطوير تطبيقات أندرويد، مبنيّة على منصة IntelliJ IDEA بما توفّره من محرّر كود قوي وأدوات متقدّمة للمطورين. وتشتمل على مميّزات مثل الإكمال الذكي، وتصحيح الأخطاء، ومحلّلات الأداء، والمحاكي، ونظام بناء قائم على Gradle لتجميع حِزَم ‎APK وإدارة الاعتمادات بكفاءة.

Dart: لغة برمجة طوّرتها Google لبناء تطبيقات حديثة تعمل عبر منصّات متعددة. تتميّز بدعم قوي للبرمجة غير المتزامنة، ويمكن ترجمتها إلى كود أصلي لتطبيقات الأجهزة أو إلى جافاسكربت لتطبيقات الويب.

Flutter إطار عمل من Google لبناء تطبيقات متعدّدة المنصّات انطلاقاً من قاعدة شفرة موحّدة تعمل على Android وiOS والويب وسطح المكتب.

  
 Provider هي حزمة (package) من متجر الحزم pub.dev تُستخدم في إدارة الحالة داخل تطبيقات Flutter.



flutter\_secure\_storage: حزمة لتخزين البيانات الحسّاسة بشكل آمن داخل التطبيق. تستخدم لتشفير القيم وحمايتها على مستوى النظام. تُستعمل لحفظ رموز الدخول (Access Tokens)، ومعرّفات الجلسة، وأي إعدادات خاصّة لا ينبغي ظهورها في التخزين العادي.

# الفصل السابع القسم العملي

## البنية الخلفية:

### هيكلة الوحدات في النظام:

تم تقسيم النظام إلى وحدات منفصلة (Modules) باعتماد هيكل Modular monolithic:

وحدة المستخدمين (User Module): يضم إدارة المستخدمين والبيانات الخاصة بهم والمتابعات والتقييم.

وحدة الكاتالوج (Catalog Module): يضم إدارة العروض والمفضلة والبيانات الخاصة بالعروض.

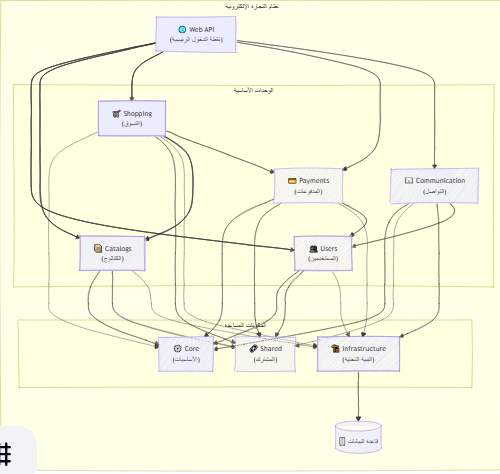
وحدة التسوق (Shopping Module) تضم إدارة سلة التسوق والطلبات.

وحدة التواصل (Communication Module): تضم المراجعات والرسائل.

وحدة الدفع (Payment module): تحتوي على المعلومات الخاصة بالدفع وطرقه.

الوحدة المشتركة (Core module): تضم الواجهات والبيانات المشتركة بين باقي الوحدات وفيها أيضاً قاعدة البيانات المشتركة وتحتوي أيضا على العقود المشتركة(Shared contract) أو ما يسمى orchestrator لتأمين التواصل بين الوحدات المنفصلة دون الحاجة لكسر الاعتماديات فيما بينها من خلال الوسيط mediatr حيث نقوم بتعريف الطلبات التي ستطلبها وحدة من أخرى ويتم تنجيز الطلب في الوحدة المسؤولة عن معالجته، بهذه الطريقة تكون الوحدة الطالبة تتعامل مع تجريد للطلب ولا تعرف أي شيء عن طريقة التنجيز وإنما الوحدة المطلوب منها تقوم بإرجاع البيانات المطلوبة دون تدخل الوحدة الطالبة، مثال:

عندما نريد معرفة معلومات المنتج في وحدة التسوق ولكن وحدة التسوق غير قادرة على الوصول إلى وحدة المنتجات (حيث توجد المعلومات) قمنا بتعريف طلب في العقود (getItemDetailes) كتعريف فقط وتم تنجيزه في وحدة الكاتالوج، عنددها تقوم وحدة التسوق بطلب getItemDetailes وتستخدم الوسيط mediatr لجلب البيانات فيقوم الوسيط بجلب البيانات من الوحدة المطلوبة بتفيذ المعالج لهذا الطلب في الوحدة المسؤولة دون علم الوحدة الطالبة لتفاصيل التنجيز وإنما فقط بالدخل والخرج؛ بهذذه الطريقة تم فصل التبعيات بين الوحدات المختلفة.

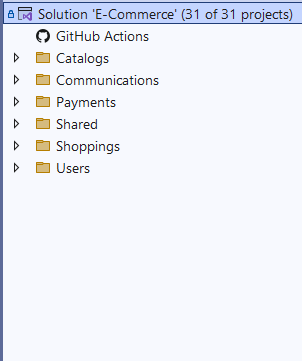


### البنية المعمارية لكل وحدة:

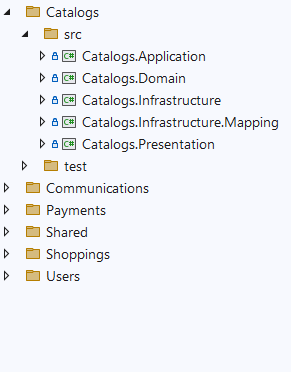
إن اعتماد منهجية واضحة لبناء النظام منذ البداية بساعد على إيجاد رماز مصدري (codebase) قابل للاختبار والتعديل مع مرور الوقت. سيتم الاعتماد على البنية المعمارية النظيفة لكل الوحدات المنفصلة، إذ تعطي إمكانية فصل قواعد العمل وحالات الاستخدام عن الخيارات التشغيلية، مما يعطي إمكانية استبدالها عند الحاجة.

#### هيكلية المجلدات العامة للوحدات:

كل وحدة موجودة في مجلد حل (solution folder) وبداخلها المنطق الكامل لهذه الوحدة.

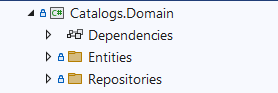


#### هيكلية الرماز المصدري:

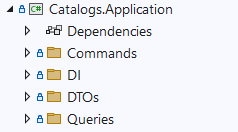


كل وحدة مؤلفة من عدة طبقات متبعة البنية النظيفة، مسماة وفق النمط التالي ([ModuleName].[layerName])، مثلاً:

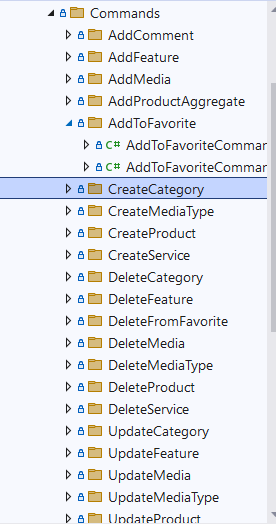
Catalog.Domain: تحتوي على التعريفات الأساسية (مثلاً Domain models) والواجهات (مثلاً IRepositories)



Catalog.Application: يحتوي تنجيز المنطق العمام للوحدة وفق نمط CQRS



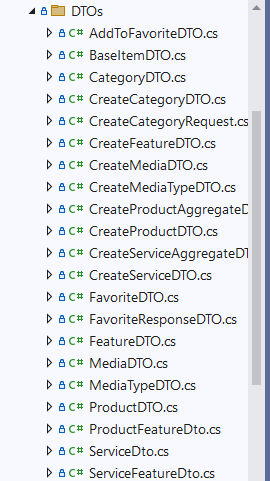
حيث بنية مجلد الcommands موضح في الصورة التالية:



كل مجلد منفصل يعالج حالة مختلفة ويحتوي على تعريف التعليمة و تنجيزها منفصل في الHandler وتتبع نمط تسمية مشابه من حيث

التعليمة([CommandName]Command) والمعالج الخاص بها([CommandName]CommandHandler)

مجلد الDTOs:



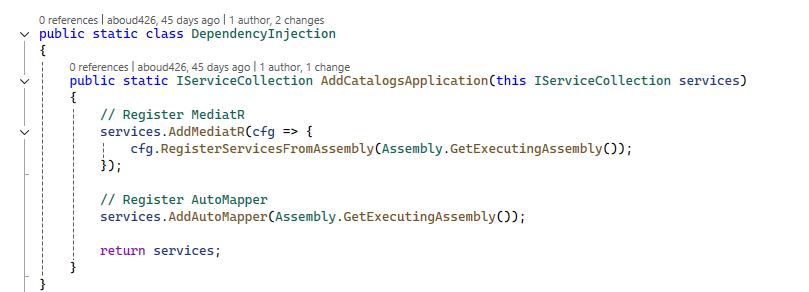
يحتوي على الصفوف المساعدة لإدخال أو تعديل أو جلب البيانات.

مجلد الطلبات:



يتبع نفس نمط مجلد ال commands.

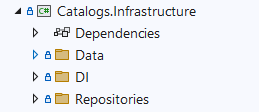
مجلد DI يحتوي على صف حقن الاعتماديات



وذلك منجز في كل طبقة تحتاج الى حقن الاعتماديات الخاص بها في البرنامج الرئيسي (program.cs) وذلك للتنسيق وعدم جعل البرنامج الرئيسي مزدحم بهذه الاعتماديات.

طبقة Catalog.Infrastracture:

تحتوي على المجلدات الموضحة في الصورة التالية:



مجلد البيانات يحتوي على البيانات التي تضاف عند إنشاء قاعدة البيانات وهي البيانات الثابتة مثل أنماط الملفات المدعومة حالياً مثل الصور أو الفيديو للعروض المطروحة(مثل المنتجات).

وكذلك يوجد مجلد حقن الاعتماديات الخاص بهذه الطبقة، بالإضافة إلى مجلد يحتوي على تنجيز الRepositories للتواصل مع قاعدة البيانات.

طبقة Catalog.Infrastracture.Mapping:

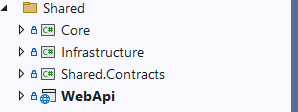
تحتوي على الmappers يحول بين الأنماط DAO وDomain Models بالاتجاهين حسب الضرورة.

طبقة ال presentation:

يحتوي على نقاط النهاية end points للطلبات والتعليمات الموجودة في طبقة التطبيق (أي يحتوي على تعريف Apis هذه الوحدة)

وكذلك الأمر لباقي الوحدات المشابهة.

الوحدة المشتركة:



تحتوي على طبقة Core (النواة الأساسية للنظام):

مجلد Core يمثل النواة الأساسية لنظام التجارة الإلكترونية ويحتوي على المكونات المشتركة والأساسية التي تستخدمها جميع وحدات النظام الأخرى. إليك تفصيل المجلدات والغرض منها:

مجلد Interfaces:

**الغرض**: يحتوي على الواجهات الأساسية (Interfaces) للنظام

* IRepository.cs: واجهة نمط Repository للعمليات الأساسية على قاعدة البيانات
* IUnitOfWork.cs: واجهة نمط Unit of Work لإدارة المعاملات
* IFileService.cs: واجهة خدمة إدارة الملفات
* IBlockchainPublisher.cs: واجهة للنشر على البلوك تشين

مجلد Configuration:

**الغرض**: يحتوي على إعدادات التكوين للنظام

* FileStorageSettings.cs: إعدادات تخزين الملفات مثل مسارات التحميل وأحجام الملفات المسموحة

مجلد DI/ (Dependency Injection)

**الغرض**: إدارة حقن التبعية (Dependency Injection)

* DependencyInjection.cs: يحتوي على تكوين وتسجيل الخدمات الأساسية

مجلد Services/

**الغرض**: الخدمات الأساسية المشتركة

* FileService.cs: خدمة إدارة الملفات (رفع، حذف، تحديث الملفات)

مجلد Results

**الغرض**: آلية موحدة لإرجاع النتائج في النظام

* Result.cs: فئة Result Pattern لإرجاع النتائج مع معلومات حالة العملية (نجاح/فشل)

مجلد Authentication

**الغرض**: أدوات وامتدادات المصادقة

* Extensions.cs: امتدادات ClaimsPrincipal لاستخراج معرف المستخدم من التوكن

مجلد Pagination

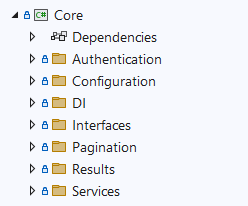
**الغرض**: آلية التصفح بالصفحات للبيانات

* PaginatedResult.cs: فئة لإرجاع البيانات مقسمة بصفحات
* PaginationParameters.cs: معاملات التصفح (رقم الصفحة، حجم الصفحة)

الأهمية الاستراتيجية لمجلد Core:

1. **المركزية**: يوفر مكونات أساسية مشتركة لجميع الوحدات
2. **إعادة الاستخدام**: يمنع تكرار الكود عبر المشروع
3. **الثبات**: يوفر واجهات ثابتة للعمليات الأساسية
4. **التوحيد**: يضمن توحيد طريقة إرجاع النتائج والتعامل مع الأخطاء
5. **المرونة**: يسهل صيانة وتطوير النظام

هذا المجلد يطبق مبادئ Clean Architecture ويضمن الفصل بين الطبقات وسهولة الصيانة والاختبار.



طبقة Infrastructure - الطبقة التحتية للنظام

طبقة Infrastructure تمثل الطبقة التحتية للنظام وهي المسؤولة عن التعامل مع قاعدة البيانات والخدمات الخارجية. إليك تفصيل شامل لمكونات هذه الطبقة:

نظرة عامة

طبقة Infrastructure مبنية على **Entity Framework Core** مع **SQL Server** كقاعدة بيانات رئيسية، وتطبق مجموعة من الأنماط المهمة مثل Repository Pattern وUnit of Work Pattern.

مجلد Models:

يحتوي على نماذج قاعدة البيانات (DAO - Data Access Objects)

مجلد Migrations/

الغرض: إدارة تطور قاعدة البيانات عبر الزمن

 مجلد Common/

الغرض: المكونات المشتركة والأساسية (UnitOfWork وBaseRepository وBaseMapper)

الأنماط المطبقة

Repository Pattern:

فصل منطق الوصول للبيانات

توحيد العمليات الأساسية (CRUD)

دعم الـ Soft Delete

Unit of Work Pattern:

إدارة المعاملات بشكل متسق

ضمان تطبيق ACID Properties

تحسين الأداء بتجميع العمليات

Data Access Object (DAO):

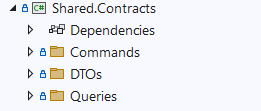
فصل نماذج قاعدة البيانات عن منطق الأعمال

تسهيل صيانة وتطوير قاعدة البيانات

هذه الطبقة تشكل العمود الفقري للنظام من ناحية إدارة البيانات وتطبق أفضل الممارسات في تصميم طبقات الوصول للبيانات.

طبقة Shared.Contracts - العقود والواجهات المشتركة

طبقة Shared.Contracts تمثل طبقة العقود والواجهات المشتركة بين جميع وحدات النظام، وتطبق نمط **CQRS** باستخدام **MediatR.**  
هذه الطبقة مصممة لتحديد العقود المشتركة التي تحتاجها الوحدات المختلفة للتواصل مع بعضها البعض، مما يقلل من الاقتران بين الوحدات ويعزز مبدأ الفصل بين الاهتمامات.



مجلد Commands/ - الأوامر المشتركة

الغرض: يحتوي على الأوامر التي تُستخدم للتواصل بين الوحدات لتنفيذ عمليات تعديل البيانات

مجلد Queries/ - الاستعلامات المشتركة

الغرض: يحتوي على الاستعلامات للحصول على البيانات من وحدات أخرى.

مجلد DTOs - كائنات نقل البيانات المشتركة

الغرض: تحديد هيكل البيانات المتبادلة بين الوحدات

هذه الطبقة تُعتبر حجر الأساس للتواصل بين الوحدات وتضمن تطبيق مبادئ Modular monolithicوالفصل السليم بين وحدات.

وأخيراً طبقة WebApi - طبقة العرض والواجهة الرئيسية

طبقة WebApi تمثل نقطة الدخول الرئيسية لنظام التجارة الإلكترونية. هذه الطبقة مبنية على ASP.NET Core 8.0 وتعمل كـ Composition Root للنظام بأكمله، حيث تجمع جميع الوحدات المختلفة وتوفر واجهة API موحدة.

يطبق نمط Composition Root Pattern:

* تجميع جميع التبعيات في مكان واحد
* تكوين DI Container بشكل مركزي
* فصل تكوين التبعيات عن منطق الأعمال

الأهمية الاستراتيجية:

للنظام ككل:

1. **التجميع**: ربط جميع الوحدات في نظام متماسك
2. **التوحيد**: واجهة API موحدة للعملاء
3. **الأمان**: تطبيق سياسات الأمان على مستوى النظام
4. **المرونة**: سهولة إضافة وحدات جديدة

للتطوير والصيانة:

1. **البساطة**: نقطة دخول واحدة للنظام
2. **التحكم**: السيطرة على تكوين النظام بالكامل
3. **الاختبار**: سهولة اختبار النظام كوحدة متكاملة
4. **النشر**: تبسيط عملية النشر والتكوين

للأداء والتوسع:

1. **التحميل**: إدارة الحمولة على مستوى النظام
2. **التخزين المؤقت**: تطبيق Cache على المستوى المناسب
3. **المراقبة**: مراقبة الأداء والسلوك
4. **التوسع**: دعم التوسع الأفقي والعمودي

طبقة WebApi تُعتبر الواجهة الأمامية للنظام وتلعب دوراً حيوياً في تجميع جميع المكونات وتوفير خدمة متكاملة وآمنة للعملاء.

### تنفيذ نظام التسجيل باستخدام Serilog وSeq

يتكوّن نظام التسجيل من مرحلتين رئيسيتين: تهيئة بيئة العمل ودمج نظام التسجيل ضمن التطبيق، كما يلي:

تثبيت خادم Seq

تمّ الاعتماد على خادم Seq لتجميع السجلات وتحليلها بشكل مركزي. وقد تم تثبيته محلياً باستخدام Docker من خلال الأمر الآتي:

docker pull datalust/seq

docker run --name seq -d -p 5341:80 datalust/seq

بعد ذلك، تم الوصول إلى واجهة الخادم عبر المتصفّح من خلال العنوان:  
http://localhost:5341  
وقد أُنشئ حساب مسؤول لإدارة النظام ومراقبة السجلات.

تثبيت مكتبات Serilog

تمّت إضافة المكتبات الضروريّة لمشروع ASP.NET Core عبر مدير الحزم NuGet، وهي كما يلي:

* Serilog.AspNetCore: للربط مع بيئة ASP.NET.
* Serilog.Sinks.Seq : لإرسال السجلات مباشرةً إلى خادم Seq.

تم تثبيت هذه الحزم من مدير الحزم Nuget Package Manager

إعداد Serilog داخل التطبيق

تمّ تهيئة Serilog في ملف Program.cs لإدارة جميع رسائل التسجيل ضمن التطبيق.

اختبار النظام والتحقّق من السجلات

بعد تشغيل التطبيق وتنفيذ بعض العمليات، تم التحقّق من تسجيل السجلات في:

* ملفّات Logs داخل المشروع.
* واجهة خادم Seq، حيث ظهرت السجلات بشكل هيكلي (Structured JSON)، كما تم إنشاء لوحة معلومات (Dashboard) لمراقبة عدد الأخطاء وعدد الطلبات المنفّذة خلال فترة زمنيّة معيّنة.

## لوحةِ الإدارة

في هذا القسمِ نُوضّحُ الخطواتِ التفصيليّةَ لإنجازِ لوحةِ الإدارة مع إبرازِ أنماطِ التصميمِ المستخدمةِ:

1. تهيئةُ المشروعِ وتقنياتهِ
   * إنشاءُ مشروعٍ جديدٍ عبرَ أداةِ Vite بقالبِ React 18 مع TypeScript، لتوفيرِ سرعةِ التطويرِ وضمانِ صرامةِ أنواعِ البياناتِ.
   * تثبيتُ Tailwind CSS لتطبيقِ “utility‑first” في التنسيقِ المرئيِّ، وإضافةُ مكتبةِ shadcn‑ui المبنيّةِ على Radix UI لإعادةِ استخدامِ أكثرَ من أربعينَ مكوّنًا جاهزًا.
2. إعدادُ إدارةِ الحالةِ والتنقّلِ
   * Provider Pattern: تكوينُ AuthContext لتوفيرِ حالةِ المصادقةِ لكافّةِ المكوّناتِ دونَ تمريرٍ طويلٍ للخصائصِ (prop drilling).
   * State Management Pattern: استخدامُ TanStack Query لإدارةِ حالةِ البياناتِ المسترجعةِ من الخادمِ مع تفعيلِ التخزينِ المؤقّتِ (caching) وتقليلِ التكرارِ في الاستدعاءاتِ.
   * Routing & Protected Route Pattern: تعريفُ المساراتِ عبرَ React Router v6، معَ حجزِ صفحاتِ الإدارةِ في مكوّنِ ProtectedRoute الذي يتحقّقُ من صلاحياتِ المستخدمِ ويعيدُ التوجيهَ إلى صفحةِ الدخولِ عندَ الضرورةِ.
3. تنفيذُ نظامِ المصادقةِ ثنائيّةِ الخطواتِ (OTP)
   * API Facade Pattern: إنشاءُ واجهةٍ موحّدةٍ (lib/api.ts) للتعاملِ مع خدماتِ المصادقةِ، تشملُ:
     1. التحقّقُ من رقمِ الهاتفِ بصيغةٍ عراقيةٍ (09xxxxxxxxx) باستخدامِ التعبيرِ النمطيِّ.
     2. إرسالُ رمزِ التحقّقِ (OTP) واستقبالُهُ للتحقّقِ.
     3. إصدارُ وتخزينُ رمزيّةِ JWT في التخزينِ المحليّ (localStorageٍ).
   * Custom Hook Pattern: تعريفُ خطّافٍ مخصّصٍ (useAuth) لإدارةِ دورةِ المصادقةِ وواجهاتِ برمجةِ التطبيقاتِ ذات الصلةِ، بما يشملُ إعادةَ إرسالِ OTP وجلبَ بياناتِ المستخدمِ بعدَ المصادقةِ.
4. عرضُ البياناتِ والتحليلاتِ

Asynchronous Handling Pattern تنفيذُ مكالماتٍ موازيةٍ عبرَ Promise.all أو عبرَ TanStack Query لجلبِ إحصائيّاتِ الطلباتِ والمستخدمينَ والمنتجاتِ والفئاتِ في صفحةِ Dashboard.

استخدامُ Recharts لعرضِ المخططاتِ التفاعليّةِ مثلَ:

* + 1. مخطّطٌ خطّيٌّ لعرضِ اتجاهاتِ الطلباتِ عبرَ الزمنِ.
    2. مخطّطاتٌ عموديّةٌ لأهمّ الفئاتِ والبائعينَ والعملاءِ.

Memoization Pattern: توظيفُ useMemo لمعالجةِ بياناتِ التحليلاتِ وحسابِ المجموعات   
 (group by) بكفاءةٍ دونَ إعادةِ العملياتِ الحسابيّةِ في كلِّ إعادةِ رسمٍ.

1. التصميمُ والاستجابةُ
   * Mobile‑First Responsive Pattern: اعتمادُ تصميمٍ متجاوبٍ يبدأُ بالشاشاتِ الصغرى ثمّ يتدرّجُ نحوَ الشاشاتِ الأكبرِ، معَ شريطٍ جانبيٍّ قابلٍ للطيِّ على أجهزةِ الحاسوبِ ونافذةٍ منبثقةٍ على الجوال.
   * ضبطُ هوامشِ المحتوىِ تبعًا لحالةِ الشريطِ الجانبيِّ لضمانِ تجربةِ مستخدمٍ سلسةٍ ومتناسقةٍ.

باستخدامِ هذه الأنماطِ التصميميّةِ من Provider وFacade وHOC إلى Custom Hooks وAsynchronous Handling تُحققُ لوحةُ الإدارةِ مرونةً عاليةً، وقابليةَ صيانةٍ، وفصلَ المسؤوليات بشكلٍ يسهّلُ التوسيعَ والتطويرَ المستقبليَّينِ.

## البنية الأمامية

## الذكاء الاصطناعي

### مقدمة:

تم بناء هذه الخدمة باستخدام إطار العمل fastapi بلغة البرمجة python، تم اختيار python عن غيرها من لغات البرمجة بسبب الدعم الكبير لها في مجال التعامل والاتصال مع نماذج اللغات الكبيرة إذ تحتوي على العديد من المكتبات أهمها:

(langchain, crewAi, langFlow, groq, …) التي تساعد على بناء التطبيق بشكل سريع وتحقيق مبدأ don’t repeat yourself (DRY).

تم الاعتماد على إطار العمل fastapi بدلاً من flask و Django، لكونه إطار العمل الأسرع كأداء ضمن هذه الإطارات ولكونه مدعوماً (من حيث المكتبات والمجتمع) أكثر من flask في المقام الأول، وبالرغم من أنه مدعوم بشكل أقل من Django ولكنه يعد كافياً في مجال بناء RESTAPI ويقدم المطلوب.

### تفاصيل الإنجاز:

تم البدء بالبحث عن مجموعة بيانات مناسبة لتدريب النماذج واختبارها، وبالرغم من عدم توفر مجموعة بيانات مطابقة تمامًا لطبيعة البيانات المستهدفة (مراجعات خدمية باللغة العربية تتضمن فجوة بين النص والتقييم)، إلا أنه تم العثور على مجموعات بيانات ذات صلة من نفس المجال، أبرزها:

* LARB Dataset: تحتوي على مراجعات نصية لكتب باللغة العربية مع تقييمات من 1 إلى 5.
* HARD Dataset: تتضمن مراجعات لخدمات مثل الفنادق والمطاعم مع تقييمات مشابهة.

تم استخدام هذه البيانات كنقطة انطلاق، مع تجربة عدة نماذج وأساليب تم استعراضها في فصل الدراسات المرجعية، وتشمل نماذج التصنيف التقليدية مثل SVM))، وكذلك النماذج العميقة المعتمدة على المحولات (Transformers) مثل MARBERT وAraBERT.

وقد تم توثيق النتائج وتحليل أداء كل نموذج وفقًا لمقاييس تقييم متعددة مثل الدقة والتذكر ومقدار الخطأ الوسطي ولكن تم ملاحظة انخفاض في الدقة وذلك لأن التقييم قريب من التقدير المستمر(regression) أي أن احتمال الخطأ بين 1 و2 كبير وكذلك بين 4 و5 وما إلى ذلك أي أنه من المتوقع عند محاولة تطبيق المسألة على مبدأ تصنيف سنواجه هذه المشكلة، يمكن النظر إلى قيمة الخطأ بحيث نعرف وسطياً بماذا يخطئ تماماً فإذا كان يخطئ بين 1 و2 فهي ليست بالمشكلة الكبيرة مثل إذا أخطأ بين 1 و5، وكذلك تم تجربة استخدام نماذج اللغات الكبيرة (LLMs) من خلال أسلوب التلقين القليل (Few-Shot Learning)، وذلك نظرًا لمرونته العالية وقدرته على التكيف مع السياق دون الحاجة لتدريب طويل، وهو مناسب بشكل خاص لحالتنا التي تتطلب فهماً عميقًا للسياق النصي وتقديرًا دقيقًا للتقييم العددي. إلا أن استخدام هذا النهج في البيئات الإنتاجية يبقى مقيدًا بمحدودية عدد الطلبات في الدقيقة والاعتماد على مزودي خدمات خارجية.

نتائج التجارب:

تم إجراء مجموعة من التجارب لتقييم أداء عدة نماذج تصنيف على مجموعتي البيانات LARB وHARD، وذلك باستخدام متوسط مربع الخطأ (MSE) كمؤشر رئيسي لتقييم جودة التنبؤ، نظراً لكون هذا المعيار يعكس الفروق الدقيقة بين القيم الحقيقية والمتوقعة، ويتيح مقارنة أدق لأداء النماذج على المهام متعددة الفئات.

على مجموعة بيانات HARD، أظهر النموذج التقليدي SVM مع تمثيل TF-IDF أداءً قوياً نسبياً، إذ حقق أقل قيمة للـ MSE بحوالي 1.09 (مقابل قيم صحيحة بين 1 و5)، مما يشير إلى قدرته على التنبؤ بالفئة القريبة من الصحيحة، حتى وإن لم يحقق تطابقاً كاملاً في التصنيف. يعود هذا الأداء إلى استخدام ضبط دقيق لفرط المعلمات وتوازن البيانات بشكل نسبي.

في المقابل، فإن نموذج MARBERT، عند استخدامه مباشرة دون إعادة ضبط (Zero-shot)، أظهر أداءً ضعيفاً جداً على مجموعة LARB، حيث بلغت قيمة MSE حوالي 3.04، ما يشير إلى انحراف كبير في التنبؤ عن الفئة الحقيقية، مع ميل مفرط نحو الفئة "1". هذه النتيجة تعكس عدم قدرة النموذج على التعميم على نصوص غير مدربة مسبقاً عليه، إضافةً إلى التأثير المحتمل لخلل التوازن في البيانات.

عند إجراء إعادة ضبط للرأس التصنيفي فقط (fine-tuning classification head)، تحسّن أداء MARBERT على مجموعة HARD، وبلغت قيمة MSE حوالي 2.18. وعلى الرغم من هذا التحسن، إلا أن النموذج استمر في الميل إلى فئة واحدة، مما يدل على تحديات في التصنيف المتوازن ضمن هذا السياق.

أما بالنسبة للنموذج الهجين AraBERT مع BiLSTM، فقد تم تقييمه تحت إعدادات مختلفة لفرط المعلمات (معدل التعلم، حجم الدفعة، وطول التسلسل). وقد بلغ أفضل أداء للنموذج قيمة MSE تقارب 1.05، ما يعكس قدرة تنافسية جيدة عند ضبط الإعدادات بعناية، ويشير إلى استقرار النموذج في التنبؤ بالفئات القريبة من الصحيحة.

تؤكد هذه النتائج أن MSE كمقياس يعكس بشكل فعّال دقة النموذج في توقع الفئات الصحيحة أو القريبة منها، ويوضح تباين الأداء بين النماذج تبعاً لطبيعة البيانات ومدى ضبط النموذج، مما يبرز أهمية المعايرة الدقيقة والتدريب المخصص في تحقيق نتائج موثوقة.

الاستنتاج:

تشير النتائج إلى أن طبيعة المشكلة تميل إلى كونها تقديراً مستمراً (Regression) أكثر منها تصنيفاً صارماً (Classification)، وذلك بناءً على تحليل قيم MSE التي عكست تدرّجاً في دقة النموذج حسب قرب التنبؤ من التقييم الصحيح. على سبيل المثال، فإن الخطأ بين التقييمين 4 و5 يُعد أقل تأثيراً من الخطأ بين 1 و5، وهو ما لا يمكن التعبير عنه بدقة ضمن إطار التصنيف متعدد الفئات، مما يبرّر التفكير لاحقاً في إعادة نمذجة المسألة كمسألة انحدار ذات حدود خطأ مقبولة بدلاً من حصرها في قالب تصنيفي تقليدي.

إضافةً إلى ذلك، فإن البيانات المتاحة تحتوي على أنماط لغوية مغايرة لتلك الموجودة في المراجعات الفعلية للمشروع. فبعض الألفاظ التي تُستخدم كتعبير إيجابي في مراجعات الكتب أو الأدب (مثل "غامض" أو "مربك") قد تُفهم على أنها سلبية في سياقات تقييم المنتجات أو الخدمات، مما يؤدي إلى تضارب دلالي عند تدريب النماذج على بيانات غير متجانسة أو خارجة عن السياق المستهدف.

كما أن طبيعة المراجعات الفعلية في المشروع لا تتسم بالعشوائية، بل تأتي في صيغة موجَّهة ومنسقة ضمن أسئلة محددة وصيغ معيارية، وهو ما لا يتوفر غالباً في مجموعات البيانات العامة، مما يُضعف قدرة النماذج التقليدية على التكيّف مع السياق الفعلي.

بناءً على ما سبق، تم اعتماد نهج نماذج اللغة الكبيرة (LLM) بأسلوب Few-Shot Learning باستخدام التلقين (Prompting)، نظراً لما يوفره من مرونة في الفهم وقدرة على التكيّف مع السياق دون الحاجة إلى تدريب شامل على بيانات كبيرة وغير متجانسة. حيث يمكن تضمين أمثلة حقيقية من مراجعات المشروع داخل الطلب الموجه للنموذج، مما يُمكّنه من تفسير السياق واستنتاج التقييم المناسب بدقة، بما ينسجم مع طبيعة الخدمة أو المنتج قيد التقييم.

التكامل مع عامل المعالجة (Worker) وRabbitMQ

تم تصميم **عامل المعالجة** (Worker) لفصل المنطق الخاص باستدعاء نموذج التقييم عن البنية الخلفية، وضمان تنظيم واستقرار أكبر للنظام كما يلي:

1. فصل المهام وفصل التبعيات
   * يتولى البنية الخلفية استقبال المراجعات الجديدة وتخزينها في قاعدة البيانات دون استدعاء مباشر للـLLM.
   * يقوم عامل المعالجة باستخراج الرسائل من طابور RabbitMQ، مما يقلل عمليات الاستعلام الدوري (Polling) على قاعدة البيانات ويخفض الحمل عليها.
2. آلية العمل
   * **النشر (Publish)**: ترسل خدمة FastAPI رسالة تحتوي على بيانات المراجعة إلى Exchange في RabbitMQ.
   * **الاستماع والاستهلاك (Consume)**: يراقب العامل Queue مرتبطة بنفس Exchange ويستهلك الرسائل واحدًا تلو الآخر باستخدام prefetch\_count=1 لضمان توزيع عادل وتجنب التحميل المفرط.
   * **المعالجة**: يحول العامل الرسالة إلى الصيغة المطلوبة لنموذج الـ LLM، ويُرسل الطلب إلى نقطة النهاية الخاصة بالذكاء الاصطناعي.
3. التكرار والاعتمادية
   * يضمّن العامل آلية Retry لضمان نجاح الاتصال بالـ LLM أو إعادة المحاولة عند فشل الطلب.
   * في حال نجاح المعالجة، يرسل العامل التقييم المُولّد من النموذج إلى الـ Backend API لتحديث بيانات المزوّد وتخزين التقييم الجديد.
   * في حال الفشل (HTTP error أو JSON parsing error)، يبقي العامل الرسالة في الطابور لإعادة المحاولة لاحقًا أو تحويلها إلى Dead‑Letter Queue للمعالجة اليدوية.
4. فوائد التصميم
   * **تحسين الأداء**: يقلل من الاستعلام الدوري على قاعدة البيانات ويتيح للبنية الخلفية الاستجابة بسرعة للمستخدمين.
   * **قابلية التوسعة (Scalability)**: يمكن تشغيل نسخ متعددة من العامل لتحقيق ارتفاع في معدل المعالجة مع الحفاظ على توزيع الرسائل بشكل عادل.
   * **مراقبة ورصد أفضل**: يمكن تتبع طول Queue، معدل الاستهلاك، وحالة ACK/Reject بسهولة باستخدام أدوات RabbitMQ Management.

بهذا التصميم، يتحقق فصل واضح بين استقبال المراجعة وأتمتة التقييم، مع ضمان مرونة وتوسعة عالية للنظام وموثوقية في التعامل مع رسائل التقييم.

## البلوكتشين

تحميل مكونات Hyperledger Fabric

كجزء من مرحلة الإعداد الأولية، يتم تحميل وتثبيت حزمة Hyperledger Fabric بما في ذلك الأدوات الثنائية (CLI binaries)، ملفات التكوين (configuration files)، وصور الحاويات (Docker images) للإصدار 2.5.0 من Fabric والإصدار 1.5.2 من Fabric CA. يتم تنفيذ ذلك عبر سكريبت رسمي يقدّمه مشروع Hyperledger، كما في الأمر التالي:

curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.5.0 1.5.2

ينتج عن هذه العملية إنشاء مجلد باسم fabric-samples يحتوي على البنية التالية:

test-network/ – البيئة التجريبية الأساسية لتشغيل شبكة Hyperledger Fabric متعددة الأطراف.

bin/ – مجموعة أدوات تنفيذ الأوامر مثل peer، orderer، configtxgen، وغيرها، والمستخدمة لإدارة الشبكة.

config/ – ملفات التكوين الخاصة بالشبكة، بما في ذلك إعدادات القنوات، المنظمات، وسياسات الترتيب.

وغيرها من الملفات التي تأتي معها



تُشكّل هذه المكونات الأساس اللازم لتشغيل شبكة Fabric محلياً بهدف تطوير واختبار العقود الذكية وتكوين قنوات الاتصال بين الأطراف المختلفة.

بعد تحميل مكونات Hyperledger Fabric، من الضروري ضبط المتغيرات البيئية الخاصة بالنظام لتسهيل الوصول إلى الأدوات التنفيذية (CLI binaries) وملفات التكوين أثناء تطوير الشبكة وتشغيلها. يتم ذلك من خلال إضافة المسارات الضرورية إلى متغيرات البيئة PATH وFABRIC\_CFG\_PATH

تُمكّن هذه الخطوة من استخدام أوامر مثل peer وorderer من أي موقع داخل نظام التشغيل دون الحاجة إلى تحديد المسار الكامل.

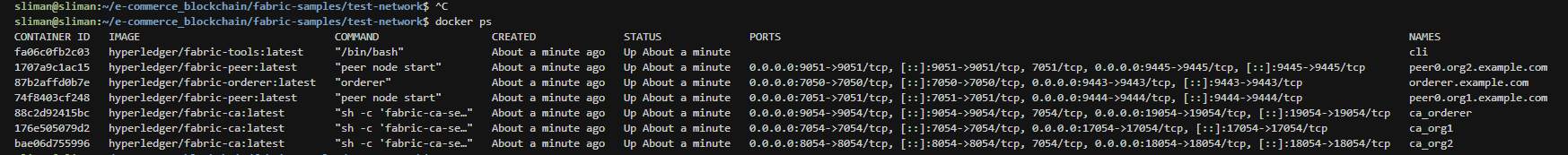
بعد إتمام مرحلة تحميل المكونات وإعداد البيئة، يتم الانتقال إلى تهيئة الشبكة الخاصة ب ـHyperledger Fabric، وهي البنية التي تتيح لنا اختبار العقود الذكية وتدفق المعاملات، بعد ذلك يتم تشغيل الشبكة وإنشاء القناة

./network.sh up createChannel - ca

ينفّذ هذا الأمر عدة مهام تلقائياً:

* تنظيف أي شبكة سابقة قيد التشغيل (إن وُجدت).
* تشغيل الحاويات المطلوبة (Docker containers) لكل من العقد النظيرة (Peers)، عقد الترتيب (Orderer)، وسلطات الشهادات (Certificate Authorities).
* إنشاء قناة اتصال باسم mychannel بين الكيانات المشاركة.
* ضمّ كل من Org1 وOrg2 إلى القناة المنشأة.

حيث بعد التنفيذ يكون لدينا خمس حاويات فعّالة تعبّر عن مكونات الشبكة:



تمثل هذه البيئة شبكة متكاملة جاهزة لاستقبال العقود الذكية وإجراء عمليات التحقق والإقرار على مستوى المؤسسات.

يوفّر إطار عمل Hyperledger Fabric خيارين لتشغيل الشبكة، يختلفان في آلية إدارة الشهادات الرقمية وبنية الشبكة:

1. التشغيل بدون سلطات شهادات (Certificate Authorities)

تعتمد هذه الطريقة على شهادات مُولَّدة مسبقاً يتم توفيرها ضمن ملفات المشروع، وتتميز ب:

* سرعة في الإقلاع وتشغيل الشبكة.
* لا تتطلب إنشاء أو إدارة ديناميكية للهويات.
* مناسبة لأغراض التطوير والاختبار.

2. التشغيل مع سلطات الشهادات (CA-Enabled Setup)

يتم تشغيل حاويتين إضافيتين تمثّلان سلطات الشهادات لكل مؤسسة:

* ca\_org1 على المنفذ 7054
* ca\_org2 على المنفذ 8054

وتقوم بتوليد ديناميكي لهويات الأعضاء وشهاداتهم الرقمية باستخدام X.509.

ثم يتم كتابة العقد الذكي بلغة Go

بعد التأكد من جاهزية ملفات العقد الذكي، يتم الانتقال إلى مجلد الشبكة التجريبية لتنفيذ عملية النشر على القناة التي تم إنشاؤها مسبقاً (mychannel).

يُستخدم السكربت الموحّد network.sh مع خيارات مخصّصة لتحديد اسم العقد (-ccn)، ومسار مجلده (-ccp)، ولغة البرمجة المستخدمة (-ccl):

./network.sh deployCC -ccn ecommerce -ccp ../../transaction-smart-contract-go -ccl go

حيث يتم:

* إنشاء تعريف للعقد الذكي على القناة المستهدفة.
* نشر الحاوية الخاصة بالعقد (Chaincode container) وتشغيلها تلقائياً على العقد النظيرة في المؤسسات.
* تفعيل وظائف العقد الذكي لتصبح قابلة للاستدعاء من خلال التطبيقات الخارجية (مثل Node.js Bridge أو واجهات REST).

وبعدها يكون قد تم تثبيت تعريف العقد على القناة mychannel، وتم تشغيل حاوية العقد الذكي بنجاح.

بعد إتمام نشر العقد الذكي على شبكة Hyperledger Fabric بنجاح، يجب تنفيذ سلسلة من الخطوات لضمان تكامل الجسر البرمجي (Node.js Bridge) مع الشبكة، وتمكينه من التعامل الآمن مع المعاملات عبر الهوية الإدارية (Admin Identity). وفيما يلي وصف مُرتّب لأهم هذه الخطوات:

1. نسخ شهادات المسؤول (Admin Certificates)

* يتم نسخ الشهادة العلنية للمسؤول (Public Key) من هيكل الشبكة إلى مجلد المحفظة (wallet) داخل جسر Node.js.
* يليها نسخ المفتاح الخاص للمسؤول (Private Key) من الشبكة إلى المكان ذاته.
* تمكّن هذه الخطوة الجسر من التصرّف ككيان إداري يمتلك صلاحيات تنفيذ المعاملات والتفاعل مع البلوكشين.

2. تحديث ملف تعريف الاتصال (Connection Profile)

* يتم تحديث شهادة TLS الخاصة بالعقدة النظيرة (Peer) داخل ملف connection-org1.json لضمان الاتصال الآمن.
* يتم تحديث شهادة سلطة الشهادات (CA Certificate) داخل الملف نفسه.

يوفّر هذا التحديث المعلومات اللازمة للجسر حول كيفية الاتصال الآمن بمكونات الشبكة المختلفة (Peers وCAs) من خلال التشفير والمصادقة.

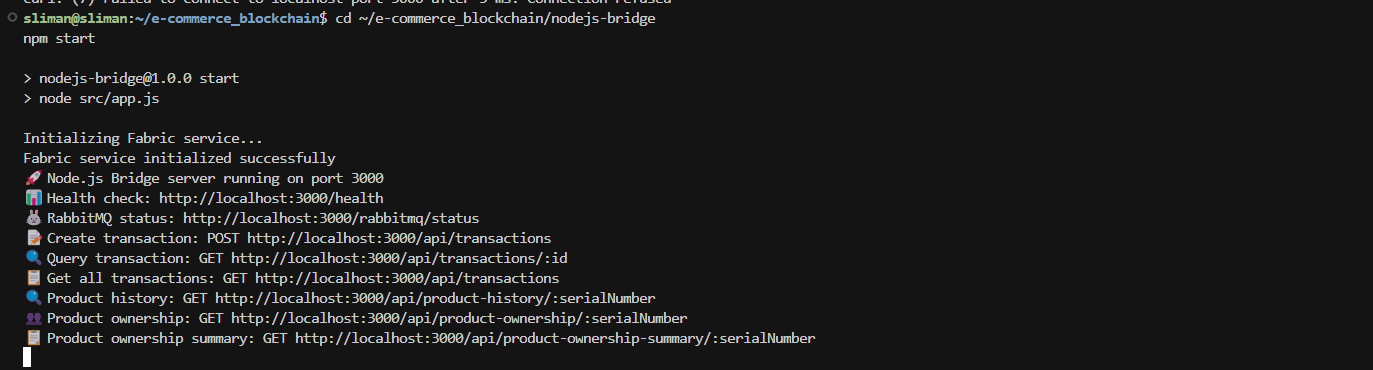
3. استيراد هوية المسؤول (Import Admin Identity)

يتم تشغيل السكربت importAdmin.js لاستيراد هوية المسؤول إلى محفظة Fabric الخاصة بالجسر.

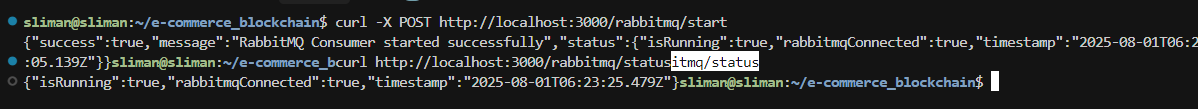
يُسجّل هذا الإجراء الشهادات المنسوخة داخل المحفظة، لتصبح جاهزة للاستخدام في توقيع المعاملات وتنفيذ العمليات الحساسة.

4. تشغيل الخدمات المساندة (Start Services)

يتم تشغيل جسر Node.js.



ثمَّ يتم تشغيل والتأكّد من أن خدمة RabbitMQ قيد التشغيل لتوفير قناة اتصال غير متزامن بين الخوادم والتطبيق.



الآن يصبح النظام في وضع التشغيل الكامل وقادراً على معالجة المعاملات التجارية الإلكترونية بطريقة آمنة، موزعة، وقابلة للتتبع.

إعداد جسر Node.js للتكامل مع شبكة البلوكشين

يمثّل جسر Node.js مكوناً معمارياً أساسياً يتيح التكامل بين تطبيقات الطرف الثالث (الواجهة الخلفية) وشبكة Hyperledger Fabric، وذلك من خلال واجهات REST وطبقة بروتوكولات وسيطة.

تهيئة مشروع الجسر حيث نقوم بتنزيل جميع الاعتمادات اللازمة لتشغيل الجسر، ومن ضمنها:

* fabric-network: مكتبة SDK الرسمية لـ Hyperledger Fabric.
* amqplib: مكتبة الاتصال بـ RabbitMQ لإدارة الرسائل غير المتزامنة.
* express: إطار عمل لتصميم واجهات REST APIs.
* dotenv: لإدارة ملفات التكوين البيئي. env.

تُعدّ هذه المكتبات جوهرية لتشغيل خدمات التفاعل مع السلسلة، تنفيذ الاستعلامات، تقديم المعاملات، وإدارة الهويات.

ومن ثمَّ إعداد ملف التكوين البيئي



يوفّر هذا الملف إعدادات بيئية (Environment Variables) تحفظ التكوينات التالية:

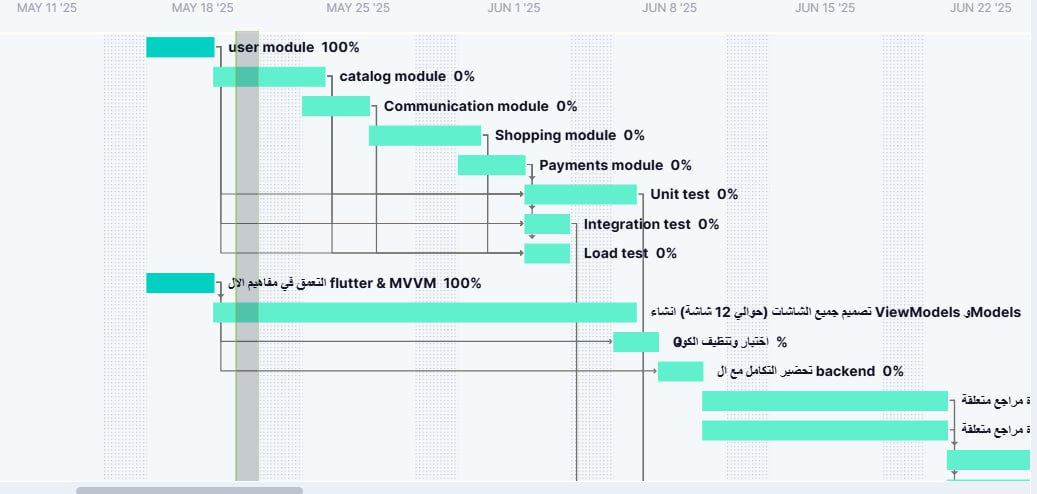
* FABRIC\_CCP\_PATH: مسار ملف تعريف الاتصال بالشبكة (Connection Profile).
* FABRIC\_WALLET\_PATH: مسار محفظة الهويات الرقمية.
* FABRIC\_CHANNEL\_NAME: اسم القناة التي يُنشر عليها العقد الذكي.
* FABRIC\_CHAINCODE\_NAME: اسم العقد الذكي المُستخدم (ecommerce).
* RABBITMQ\_URI / QUEUE: معلومات الاتصال بوسيط الرسائل RabbitMQ لم يتم استخدام localhost لأن وسيط الرسائل في نظام تشغيل المضيف Windows أمّا الشبكة فهي موجودة داخل آلة افتراضية وبالتالي فأننا نحتاج العنوان الصريح للجهاز المضيف.
* PORT: المنفذ الذي سيعمل عليه خادم الجسر.

هذا التكوين يضمن استقرار الاتصال، إمكانية إعادة التشغيل، وضمان عمل الجسر في بيئات مختلفة دون تعديل الكود المصدر.

## توزيع المهام والتنسيق بين أعضاء الفريق

نظرًا لأن المشروع يتكوّن من عدة محاور تقنية متكاملة، فقد تم توزيع المهام بين عضوي الفريق بما يتوافق مع قدرات وخبرة كلٍّ منهما. حيث تولّى الطالب الأول مسؤولية تصميم وتطوير النظام الخلفي **(Back-End)** بالإضافة إلى بناء وتدريب نموذج الذكاء الاصطناعي (**AI**) الخاص بتحليل التقييمات وتحسين التوصيات. بينما ركّز الطالب الثاني على تصميم وبرمجة الواجهة الأمامية (**Front-End**) لضمان تجربة مستخدم فعّالة وجذّابة، إلى جانب تنفيذ بنية البلوكتشين (**Blockchain**) بهدف تأمين سجلات المعاملات وضمان موثوقيتها.

تم التنسيق بين العضوين من خلال عقد اجتماعات دورية لمراجعة مراحل التنفيذ، إضافةً إلى توظيف نظام التحكم بالإصدار (**Git**) لتقسيم المهام وتنظيم عمليات الدمج. كما تم استخدام موقع **Project Management** لمتابعة المهام والجدول الزمني وتوزيع المسؤوليات بوضوح، مما ساعد على ضمان تزامن العمل وتكامل المكوّنات المختلفة بسلاسة وفاعلية.



# الفصل السابع النتائج والاختبار

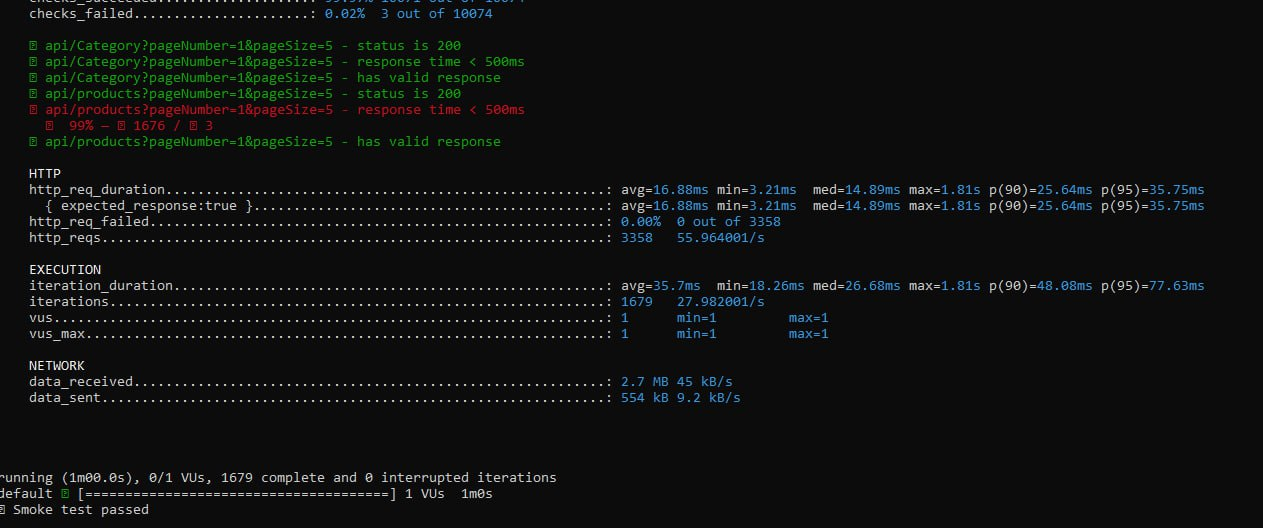
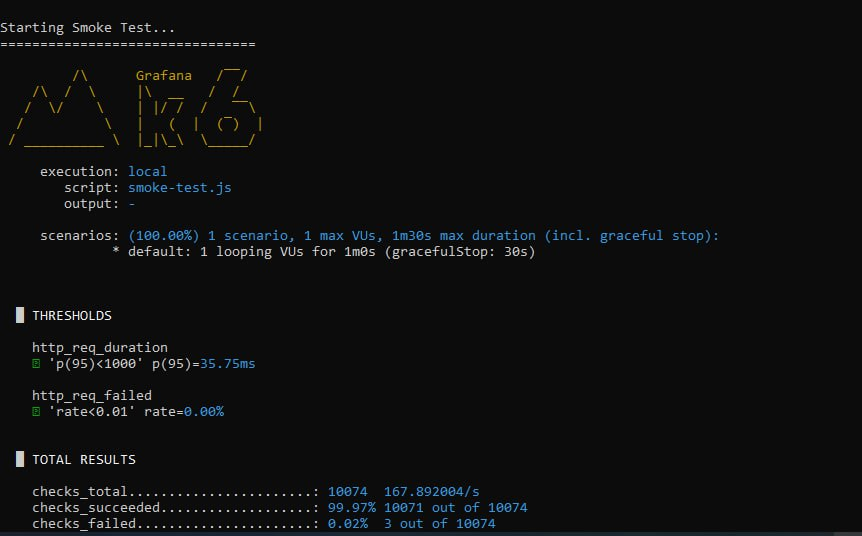
## اختبار الوحدات Unit test

تم تنفيذ أكثر من 50 حالة اختبار تغطي وحدتين أساسيتين ضمن بنية المشروع المعيارية (Modular Monolithic)، ركزت على عمليات قراءة البيانات والتحقق من صلاحية المدخلات في الطبقات المنطقية للتطبيق. ونظراً لضيق الجدول الزمني، لم تُغطَّ جميع الوحدات، إلا أن الاختبارات المنفذة حققت معدلات نجاح مرتفعة دون تسجيل أخطاء، مما يدلّ على استقرار المنطق الوظيفي وسلامة التكامل مع المكونات المحاكاة (Mocks)، ما يلي يوضح واجهة الاختبارات بعد تشغيل الاختبارات:



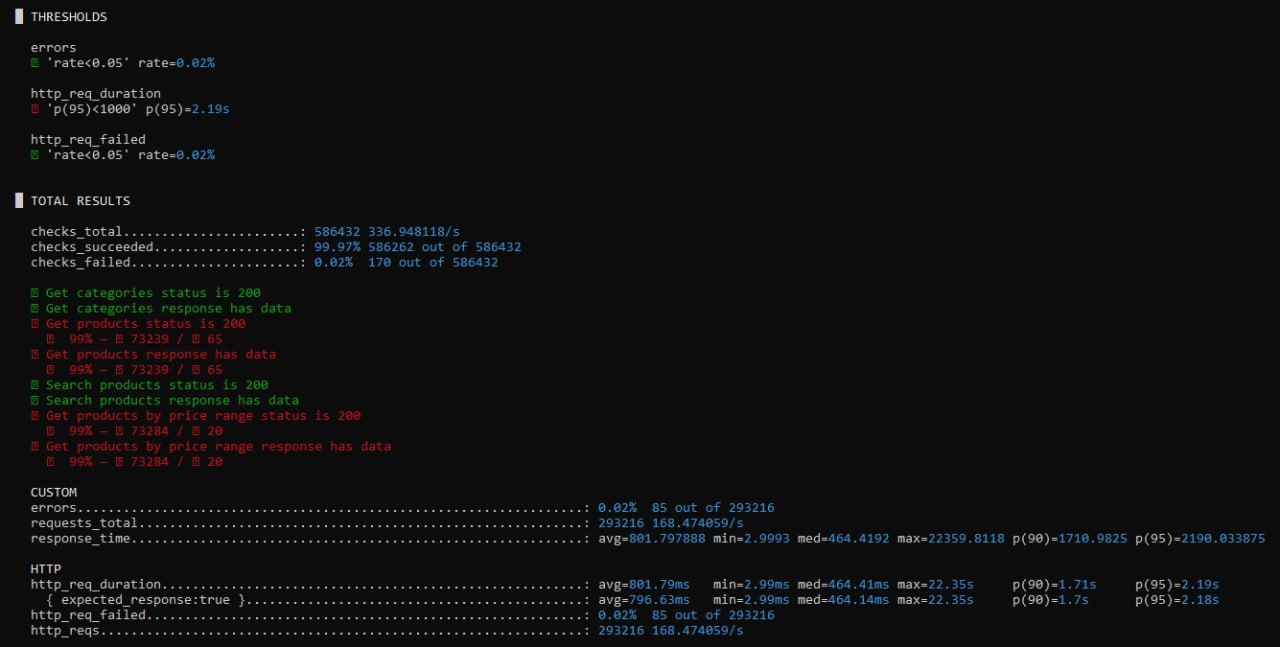
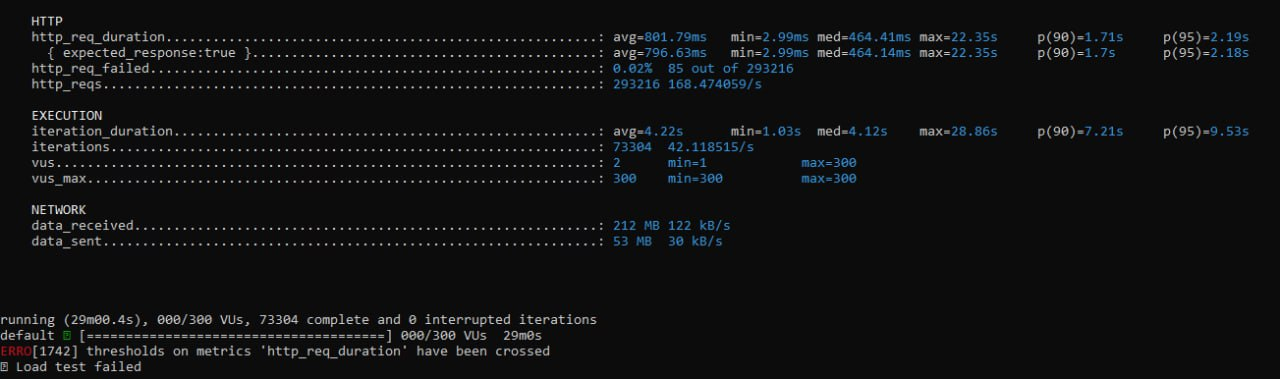
## اختبارات الدخان (Smoke Tests)

استهدفت هذه المجموعة التحقّق من عمل واجهات برمجة التطبيقات الأساسية واستجابتها ضمن زمن لا يتجاوز 1 ثانية للنسبة العظمى من الطلبات. مرّت جميع نقاط النهاية الرئيسية بتحقق صحي (HTTP 200) واستجاباتٍ صالحة مع معدّل خطأ أقل من 1% خلال دقيقة من التشغيل الافتراضي، ما يدلّ على جاهزية التطبيق للنشر في بيئة اختبار أولية دون اكتشاف أعطال كبيرة.



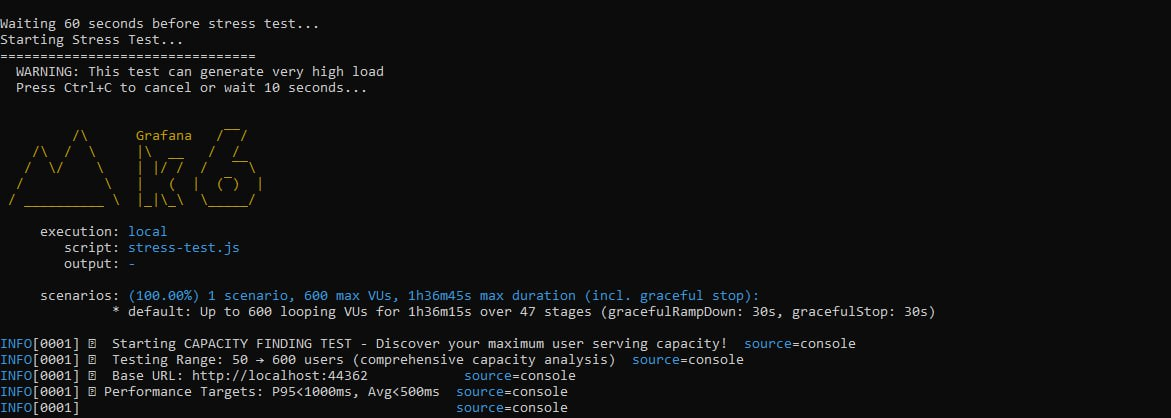
## اختبار الحمولة (Load test):

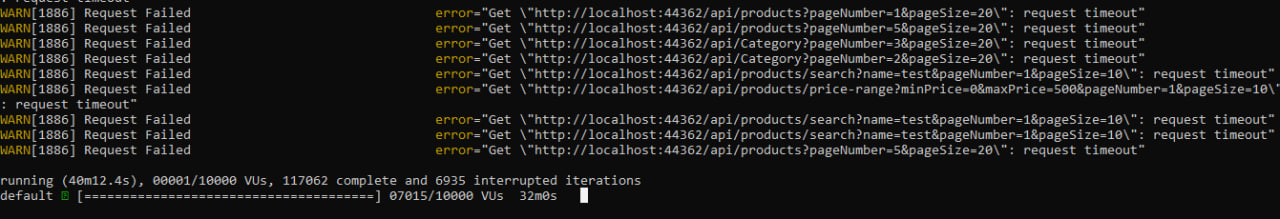
من خلال تصميم سيناريو تصاعدي شمل مراحل متدرجة من 50 إلى 500 مستخدم متزامن، تم تقييم استقرار النظام تحت ضغوط متزايدة. حافظ متوسط زمن الاستجابة عند الحمل المتوسط (200–300 مستخدم) على أقل من 800 ميلي ثانية، مع معدّل أخطاء لا يتجاوز 0.02%. هذه النتائج تشير إلى قدرة المنصة على التعامل مع زيادات معتدلة في أعداد المستخدمين دون تدهور ملحوظ في الأداء.



## اختبار الضغط (Stress test):

**اختبار الإجهاد (Stress Test)**  
تم تصميم هذا الاختبار لزيادة عدد المستخدمين بشكل تدريجي من 5 مستخدمين إلى أن وصلت النقطة الحرجة عند 7,015 مستخدماً متزامناً، وذلك لعدة أسباب: أولاً، تتيح الزيادة التدريجية مراقبة استجابة النظام عند كل مستوى تحميل وتحديد نقطة الانهيار بدقة، وثانياً تساعد على تجنّب الضغط المفاجئ للنظام الذي قد يؤدي إلى نتائج غير متوقعة. عند الوصول إلى 7,015 مستخدمًا، توقف الاختبار تلقائيًا نتيجة وصول معدل الفشل إلى 50%، مما يحدد أقصى سعة تشغيلية مستقرة للتطبيق. بناءً على هذا الاستنتاج، يُنصح باعتماد سعة تشغيلية مستدامة تُقدَّر بحوالي 70% من هذا الحدّ الأقصى أي نحو 4,910 مستخدماً وتفعيل آلية التوسّع التلقائي عند 80% من الحدّ المقدر لضمان مرونة واستمرارية الخدمة في أوقات الذروة.





فيما يلي مخطط يوضح كيفية زيادة عدد المستخدمين بدلالة الزمن في اختبار الضغط:

# الآفاق المستقبلية

نسعى مستقبلاً إلى استكمال معالجة الخدمات اليدوية ضمن النظام، وإدخال نظام توصية ذكي لعرض المنتجات والخدمات بكل دقة أثناء التصفح، فضلاً عن دعم المحادثة والتعليقات داخل التطبيق لتعزيز التفاعل. كما سنطبق نظام إشعارات وتذكير يوافي المستخدمين بالتحديثات والرسائل الجديدة والعروض عبر البريد الإلكتروني والرسائل القصيرة والإشعارات الفورية. وفي حال تحقيق الأهداف المرجوّة، سندرس الانتقال إلى بنية خدمات مصغّرة (Microservices) لضمان مرونة أعلى وسهولة أكبر في الصيانة والتوسع.

# الخاتمة

# المراجع