

الفهرس

مقدمة: 3

الفصل الأول: منهجية المشروع: 4

1-1 أهداف المشروع: 4

2-1 طريقة العمل: 4

الفصل الثاني: دراسة نظرية في قواعد البيانات: 5

1-2 لمحة عامة عن قواعد البيانات: 6

2-2 قواعد البيانات العلائقية: 7

3-2 قواعد البيانات غير العلائقية: 11

4-2 مفاهيم عامة: 12

5-2 قابلية التوسع: 15

الفصل الثالث: تجربة عملية: 18

1-3: التوصيف الوظيفي للنظام: 18

2-3: مخطط حالة الاستخدام: 19

3-3: مخطط ERD: 20

4-3: توصيف بنية النظام في MongoDB: 21

5-3: نظرة على أبرز الاختلافات بين النموذجين: 23

3-6: هل يمكن الاستغناء عن تطبيع البيانات؟ 25

3-7: شرح لقسم الواجهة الخلفية في المتجر: 26

3-8: توليد البيانات من أجل الاختبار: 32

3-9: تجهيز بيئة الاختبار: 34

3-10: اختبار سرعة التنفيذ لبعض الاستعلامات في النموذجين: 34

الفصل الرابع: طرق ترميز المعلومات في تطبيقات الويب: 39

4-1: تعريف طرق ترميز المعلومات في تطبيقات الويب: 39

4-2: الترميز وفك الترميز Serialization vs Deserialization: 39

4-3: أشهر الصيغ القياسية المستخدمة في التصميم: 40

الفصل الخامس: بروتوكولات الاتصال: 43

5-1: تعريف: 43

5-2: بروتوكول HTTP: 44

5-3: بروتوكول WebSocket: 44

5-4: الفرق بين WebSocket و HTTP: 45

المراجع: 47

مقدمة

تشكل تطبيقات الويب في يومنا هذا نواة أساسية في حياة الكثير من الناس والصناعات، فبعد أن بدأ الموضوع على شكل مواقع ستاتيكية موصفة بلغتي HTML و CSS هدفها عرض المعلومات والتنقل بين الصفحات فقط، انتقلت المواقع الالكترونية الى مرحلة مختلفة تماماً تتسم بالديناميكية والتفاعلية بشكل كبير، فأصبحنا اليوم نرى جميع أنواع التطبيقات المختلفة كمواقع التواصل الاجتماعي وتطبيقات المحادثة والمتاجر الالكترونية وغير ذلك من التطبيقات التي أصبحت جزءاً رئيسياً في حياتنا.

غالباً ما تتكون تطبيقات الويب الحديثة من ثلاث مكونات رئيسية هي:

- تطبيق برمجي يعمل على الخادم
- تطبيق برمجي يعمل على جهاز العميل
- قاعدة بيانات

ويوجد في يومنا هذا العديد من التقنيات التي تمكننا من بناء هذه التطبيقات، وتعدد هذه التقنيات سببه هو اختلاف المتطلبات التقنية بين التطبيقات، فمثلاً المتطلبات لموقع محادثة يقوم باستقبال وتحديث الرسائل تلقائياً ستختلف عن المتطلبات لموقع يقوم بعرض كتب الكترونية يتم تحديثه فقط عن طريق المستخدم.

في بحثنا هذا سوف نقوم بدراسة بعض الاختلافات ضمن محورين من التقنيات المستخدمة في تطبيقات الويب، في المحور الأول سنستعرض أهم الفوارق بين قواعد البيانات العلائقية وغير العلائقية وما هي العوامل التي من الممكن أن تساعدنا في اختيار النوع المناسب، أما في المحور الثاني فسوف نتعرف الى بروتوكول الاتصال Web Socket وكيف يمكن استخدامه لبناء تطبيقات يستطيع فيها الخادم ارسال التحديثات الى الزبون تلقائياً دون الحاجة لطلب من الزبون، الأمر الذي لا يمكن تحقيقه عن طريق بروتوكول الاتصال الشهير HTTP بنسخته الأولى HTTP 1.X.

الفصل الأول

منهجية المشروع

1-1 أهداف المشروع:

- 1- التعرف الى الأسباب التي أدت الى ظهور قواعد البيانات غير العلائقية NoSQL وكيفية اختلافها في نموذج تخزين البيانات عن قواعد البيانات العلائقية SQL.
- 2- التعرف الى الخصائص ACID في قواعد البيانات العلائقية وخصائص أنظمة قواعد البيانات الموزعة.
- 3- القيام بعمل مقارنة عملية من خلال بناء تطبيق متجر الكتروني باستخدام كل من نظامي إدارة قواعد البيانات SQL Server و MongoDB ودراسة إيجابيات وسلبيات كل منهما.
- 4-لقاء نظرة على الطرق المختلفة لترميز المعلومات من أجل ارسالها عبر الشبكة في تطبيقات الويب ومعرفة حالات استخدام كل منها.
- 5-لقاء نظرة على بعض بروتوكولات الاتصال المستخدمة في تطبيقات الويب ومعرفة حالة استخدام كل منها.

1-2 طريقة العمل:

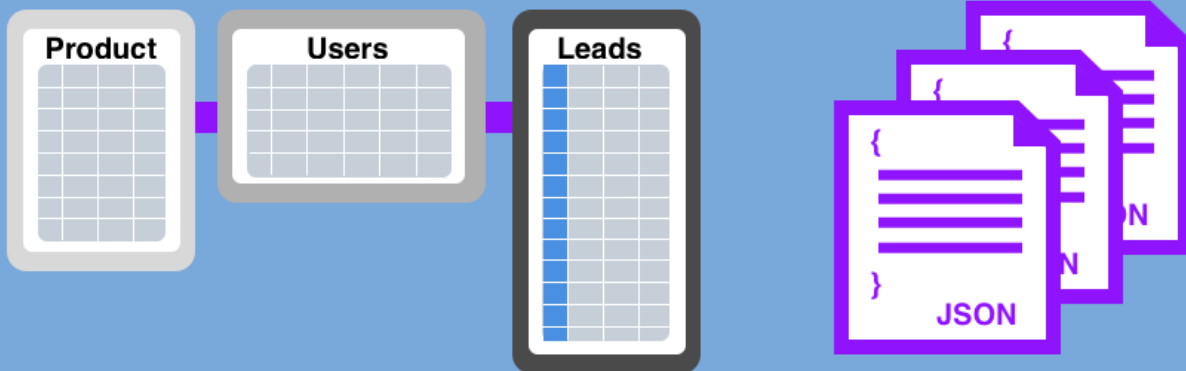
سنقوم بدراسة وبناء المتطلبات الوظيفية لنظام متجر الكتروني بسيط من نوع B2C. سيتم بناء مخطط نموذج تخزين البيانات لهذه المتطلبات مرة أولى من أجل قاعدة بيانات علائقية حيث سيتم رسم مخطط ERD للكيانات والعلاقات فيما بينها، ومرة ثانية من أجل قاعدة بيانات MongoDB حيث سيتم بناء Collections Scheme.

بعد الانتهاء من بناء المتجر بكل من الطريقتين، سوف نقوم بعملية ملئ لقاعدتي البيانات ببيانات وهمية عن طريق بناء توابع مخصصة لهذا الغرض، ثم سنقوم بإجراء اختبار للأداء من أجل بعض وظائف المتجر ودراسة النتائج وتحليلها.

الفصل الثاني

دراسة نظرية في قواعد البيانات

SQL vs. NoSQL



1-2 لمحة عامة عن قواعد البيانات:

بطريقة بسيطة مجردة من مفاهيم التقنية، قاعدة البيانات هي مكان لحفظ بيانات معينة على نحو مستمر بهدف الرجوع إليها وقت الحاجة، فدقتر أرقام الهواتف الذي كنا نستعمله في الماضي يُعدّ قاعدة بيانات؛ والكم الهائل من الفواتير المحاسبية الورقية المحفوظة في خزانات الأقسام المالية في الشركات قديماً، أيضاً هو قاعدة بيانات. وقس على ذلك العديد من الأمثلة الواقعية والملموسة.

نستنبط من هذا التعريف البسيط وجود خاصية هامة لقاعدة البيانات، ألا وهي "الاستمرارية" أو "الدوام" في حفظ البيانات.

في الجانب التقني والبرمجي، فإن قاعدة البيانات Database هي عبارة عن مستودع تُحفظ البيانات فيه داخل جهاز الحاسوب أو الخادم، ويتمتع هذا المستودع بخاصية الاستمرارية في حفظ البيانات. ونعني بخاصية الاستمرارية هنا أنه في حال إطفاء جهاز الحاسوب أو إعادة تشغيله أو انقطاع التواصل معه، فإن قاعدة البيانات وما تحتويه من بيانات تبقى موجودة ومحفوظة دون أي خلل.

بشكل رسمي، تشير "قاعدة البيانات" إلى مجموعة من البيانات ذات الصلة وطريقة تنظيمها. عادة ما يتم توفير الوصول إلى هذه البيانات من خلال "نظام إدارة قواعد البيانات (DBMS)" الذي يتكون من مجموعة متكاملة من برامج الكمبيوتر التي تتيح للمستخدمين التفاعل مع قاعدة بيانات واحدة أو أكثر وتوفر الوصول إلى جميع البيانات الموجودة في قاعدة البيانات (على الرغم من القيود قد توجد التي تحد من الوصول إلى بيانات معينة). يوفر نظام إدارة قواعد البيانات (DBMS) العديد من الوظائف التي تسمح بدخول وتخزين واسترجاع كميات كبيرة من المعلومات وتوفر طرقاً لإدارة كيفية تنظيم هذه المعلومات.

بسبب العلاقة الوثيقة بينهما، غالباً ما يتم استخدام مصطلح "قاعدة البيانات" بشكل عرضي للإشارة إلى كل من قاعدة البيانات و DBMS المستخدمة في معالجتها.

خارج عالم تكنولوجيا المعلومات الاحترافية، غالباً ما يتم استخدام مصطلح قاعدة البيانات للإشارة إلى أي مجموعة من البيانات ذات الصلة (مثل جدول بيانات أو فهرس بطاقة) لأن متطلبات الحجم والاستخدام تتطلب عادة استخدام نظام إدارة قاعدة البيانات.

توفر نظم إدارة قواعد البيانات الحالية العديد من الوظائف التي تتيح إدارة قاعدة بيانات وبياناتها والتي يمكن تصنيفها إلى أربع مجموعات وظيفية رئيسية:

- تعريف البيانات - إنشاء وتعديل وإزالة التعاريف التي تحدد تنظيم البيانات.
- تحديث - إدخال وتعديل وحذف البيانات الفعلية.

- استرجاع - توفير المعلومات في نموذج قابل للاستخدام مباشرة أو لمزيد من المعالجة بواسطة التطبيقات الأخرى. قد يتم توفير البيانات التي تم استردادها في نموذج بنفس الطريقة التي يتم تخزينها بها في قاعدة البيانات أو في نموذج جديد تم الحصول عليه عن طريق تغيير أو دمج البيانات الموجودة من قاعدة البيانات.
- الإدارة - تسجيل ومراقبة المستخدمين، وفرض أمان البيانات، ومراقبة الأداء، والحفاظ على سلامة البيانات، والتعامل مع التحكم في التزامن، واستعادة المعلومات التي تضررت بسبب بعض الأحداث مثل فشل النظام غير المتوقع.

تتطابق كل من قاعدة البيانات و DBMS مع مبادئ نموذج قاعدة بيانات معين.

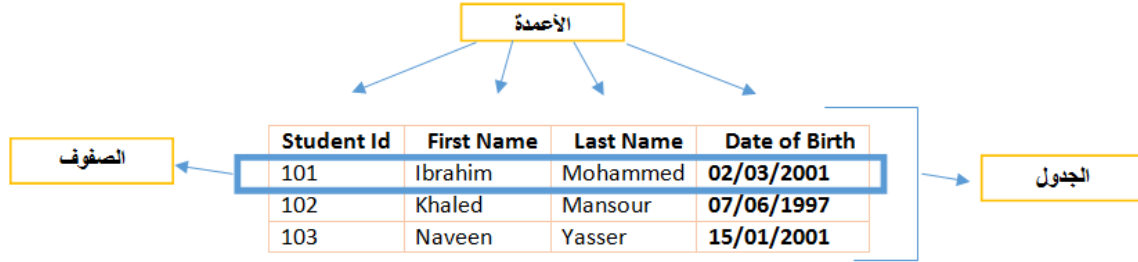
يشير "نظام قاعدة البيانات" بشكل جماعي إلى نموذج قاعدة البيانات ونظام إدارة قاعدة البيانات وقاعدة البيانات. من الناحية الفعلية، تعد خوادم قواعد البيانات عبارة عن أجهزة كمبيوتر مخصصة لها قواعد البيانات الفعلية وتعمل فقط على قواعد البيانات والرسائل ذات الصلة. عادةً ما تكون خوادم قواعد البيانات عبارة عن أجهزة كمبيوتر متعددة المعالجات، مع شرائح ذاكرة كبيرة و SSD تستخدم للتخزين الثابت. تستخدم سرعات قاعدة بيانات الأجهزة، المتصلة بخادم واحد أو أكثر عبر قناة عالية السرعة، أيضًا في بيئات معالجة المعاملات كبيرة الحجم. توجد قواعد بيانات إدارة قواعد البيانات (DBMS) في قلب معظم تطبيقات قواعد البيانات. قد يتم إنشاء قواعد بيانات إدارة قواعد البيانات (DBMS) حول نواة متعددة المهام مخصصة مع دعم شبكة مضمن، لكن نظم إدارة قواعد البيانات الحديثة تعتمد عادةً على نظام تشغيل قياسي لتوفير هذه الوظائف.

يمكن تصنيف أنظمة إدارة قواعد البيانات (DBMS) وفقًا لطراز (نماذج) قاعدة البيانات التي يدعمونها (مثل العلائقية أو غير العلائقية على سبيل المثال)، ولغة الاستعلام التي تستخدم للوصول إلى قاعدة البيانات (مثل SQL أو XQuery)، والهندسة الداخلية الخاصة بهم، والتي تؤثر على الأداء، والتدرجية، والمرونة، والأمن.

2-2 قواعد البيانات العلائقية :Relational Databases

هو النوع الأشهر والأكثر استخدامًا منذ بداية ظهوره، يُعدّ الجدول العنصر الأساسي في قواعد البيانات العلائقية، وعليه تعتمد أغلب مكونات قاعدة البيانات من مشاهد Views ودوال Functions وجزم Packages وغيرها من العناصر الأخرى. يتكون الجدول من أعمدة Columns وصفوف Rows، حيث تمثل الأعمدة ما يسمى بالخصائص Features، والصفوف عبارة عن القيم التي تأخذها الأعمدة وتسمى بالسجلات Records.

يوضح الشكل التالي مثالا لجدول يحتوي على بيانات تواريخ ميلاد وأسماء طلاب في مدرسة، وفي المثال نوضح مكونات الجدول في قاعدة البيانات.



خصائص قواعد البيانات العلائقية:

ظلت قواعد البيانات العلائقية مهيمنة منذ بدايات ظهور النموذج الأساسي لها عام 1970 على يد عالم الحاسوب Frank Codd أثناء عمله لصالح شركة IBM، ولم تكن هذه الأفضلية التي يتمتع بها نظام قواعد البيانات العلائقية تأتي من فراغ، بل من الخصائص التي تتمتع بها.

البساطة

تُرتَّب البيانات في أنظمة قواعد البيانات العلائقية وتُحفظ بطريقة بعيدة عن التعقيد، حيث يعدّ الجدول الذي تُحفظ فيه البيانات مفهوماً لأغلب المستخدمين وخاصة الذين مارسوا أعمالاً في مجال البيانات المجدولة أو مراجعة السجلات.

سهولة الاستعلام عن البيانات

بعد عمليات الإضافة على قاعدة البيانات، وعند الحاجة للرجوع لها، فإن نظام قواعد البيانات العلائقية يوفر آلية سهلة للاستعلام عن هذه البيانات واستردادها، وذلك عن طريق لغة SQL، بالإضافة إلى وجود إمكانية للمستخدم أن يستعلم عن البيانات من أكثر من جدول في نفس الوقت باستخدام جمل الربط Joins كما أن خاصية ترشيح Filtering البيانات وتحديد شروط خاصة لظهور سجلات معينة هو أمر متاح بكل سهولة.

سلامة البيانات

تعدّ هذه الخاصية أساسية في أي نظام قواعد بيانات بغض النظر عن نوعه. ونعني بهذه الخاصية أن تتوفر جميع القدرات والإمكانات في نظام قواعد البيانات لضمان دقة وصحة المعلومات الموجودة فيه. ويندرج تحت هذه الخاصية ما يسمى بقيود التكامل Integrity constraints والتي هي عبارة عن مجموعة من القيود التي يجب الالتزام بها عند التعامل مع البيانات في الجدول، وسنتكلم عنها في مقال متقدم.

المرونة

تتمتع قواعد البيانات العلاقية بطبيعتها بالمرونة والقابلية للتطوير، مما يجعلها قابلة للتكيف مع طلبات التغيير والزيادة في كم البيانات. وهذا يعني مثلاً أنك تستطيع التغيير على هيكلية جدول معين دون التأثير على البيانات الموجودة فيه أو على قاعدة البيانات ككل، كما أنك – مثلاً - لن تحتاج إلى وقف قاعدة البيانات وإعادة تشغيلها مرة أخرى لتنفيذ بعض لتغييرات عليها.

البرمجيات التي تقدم قواعد البيانات العلاقية:

تتعدد الشركات والبرمجيات التي تُقدم أنظمة إدارة قواعد البيانات، وكل منها له سوقه ومجاله الذي يشتهر به. نُقدم لكم في الفقرات القادمة بعضاً من أشهر أنظمة إدارة قواعد البيانات العلاقية.

قواعد بيانات MySQL

أحد أشهر أنظمة قواعد البيانات العلاقية مفتوحة المصدر. تستطيع إنشاء العديد من قواعد البيانات بداخلها، وتستطيع الوصول لها عبر الويب. تعمل MySQL على هيئة خدمة Service تُتيح لأكثر من مستخدم الوصول إلى أكثر من قاعدة بيانات، وتشتهر بين معشر مبرمجي تطبيقات الويب لارتباطها الشائع مع لغة البرمجة PHP ، ويمكن تنصيبها على أكثر من نظام تشغيل مثل ويندوز أو لينكس أو ماك.

تعدّ MySQL الخيار المفضل للشركات الناشئة أو المتوسطة وذلك لسهولة التعامل معها وانخفاض تكاليف تشغيلها مقارنة بخيارات أخرى.

قواعد بيانات أوراكل Oracle

تعدّ شركة أوراكل عملاق الشركات البرمجية التي تقدم أنظمة إدارة قواعد البيانات العلاقية، وتأتي قاعدة البيانات أوراكل بأكثر من إصدار (حسب البيئة والغرض) تبدأ من الإصدار الشخصي والخفيف، وتنتهي بالإصدار المتقدم Enterprise.

تتميز قواعد بيانات أوراكل بكم كبير من الإمكانيات التي تسهل عليك حل العديد من المشاكل والعقبات في التطبيقات التي تديرها وتنشئها، مع وجود دعم فني قوي عبر مجتمع أوراكل، لذلك فهي تعتبر الخيار الإستراتيجي (البعيد المدى) للعديد من الشركات الكبيرة والجامعات والحكومات.

قواعد بيانات مايكروسوفت Microsoft SQL Server

من قواعد البيانات الشهيرة، والذي تأتي أيضا بأكثر من إصدار، لتلبي احتياجات المستخدمين المختلفة وبيئات عملهم، ولكي تتعامل مع البيانات في هذا النوع تحتاج لاستخدام النسخة الخاصة من SQL والمسماة T-SQL اختصارا ل Transact SQL والتي هي عبارة عن نسخة SQL مضاف عليه دوال خاصة وتعديلات على طريقة حذف وتعديل السجلات.

قواعد بيانات PostgreSQL

قواعد بيانات PostgreSQL من قواعد البيانات العلاقية المفضلة لدى بعض مطوري تطبيقات الويب وتطبيقات سطح المكتب، وهو نظام إدارة قواعد بيانات مفتوح المصدر. توجد الكثير من الشركات الكبيرة والعامة في مجال نطاقات إنترنت تعتمد على هذا النوع من قواعد البيانات.

3-2 قواعد البيانات غير العلائقية Non-Relational Databases:

إن التقدم المتسارع في متطلبات الأنظمة المعلوماتية، والتضخم الكبير في حجم البيانات التي تعالجها تلك الأنظمة، أدى الى ظهور أنواع جديدة من قواعد البيانات تتبع نماذج غير علائقية في تخزين البيانات، ولاحقاً أصبح يُشار إليها بـ NoSQL وهذا الاسم هو اختصار لـ Not Only SQL.

يعتبر نموذج قواعد البيانات غير العلائقية أيضاً كاستجابة للجيل الثاني من الويب (web 2.0)، حيث ظهرت الحاجة لمعالجة البيانات غير البنيوية Unstructured Data بطريقة مرنة وسهلة، وهو الامر الذي لا توفره أنظمة البيانات العلائقية التي يجب أن تكون فيها بنية الجداول محددة ومعرفة مسبقاً بشكل دائم.

إن مصطلح NoSQL لا يشير الى نموذج واحد لتخزين البيانات، حيث أن كل نوع من قواعد البيانات غير العلائقية يتبع نموذج تخزين معين، سوف ندرس في هذا المحور نموذج يسمى بقواعد البيانات الموجهة نحو المستندات Document-Oriented Databases، حيث يتم تخزين البيانات على شكل مستندات Documents ضمن مجموعات Collections وكل مستند يتألف من ثنائية حقل – قيمة Key – Value Pair، ويعتبر هذا النموذج مشابه نوعاً ما لنموذج الجداول والأعمدة المتبعة في قواعد البيانات العلائقية، إلا أن الاختلاف يبرز في امكانية تخزين مستندات ذات بنية مختلفة ضمن نفس المجموعة، وأيضاً في امكانية استخدام ما يعرف بالمستندات المتداخلة Nested Documents حيث يمكن لمستند ما أن يحوي على مستند جزئي بداخله أو على لائحة من المستندات الجزئية الداخلية والتي من غير الضروري أن يكون لها نفس البنية.

سوف نقوم في هذا المحور بـ اجراء مقارنة عملية ما بين MongoDB و SQL Server، حيث أن MongoDB هي قاعدة بيانات غير العلائقية وتتبع نموذج Document-Oriented Database المذكور سابقاً، ظهرت MongoDB بداية في عام 2007 وتطورت لاحقاً لتصبح من أشهر الأنظمة المستخدمة في قواعد البيانات غير العلائقية لما تتميز به من سهولة ومرونة في التعامل وقوة الميزات التي توفرها للمطورين، أما SQL Server فهو نظام إدارة قواعد البيانات علائقي RDBMS تم تطويره من قبل شركة مايكروسوفت.

4-2 مفاهيم عامة:

المناقلة Transaction:

في سياق قواعد البيانات وأنظمة تخزين البيانات، فإن المناقلة هي عملية أو أكثر على البيانات يتم التعامل معها كوحدة عمل واحدة، والتي إما تكتمل بالكامل أو لا تكتمل على الإطلاق، وتترك نظام التخزين في حالة متسقة Consistent. المثال الكلاسيكي للمناقلة هو عملية نقل مبلغ مالي بين حسابين مصرفيين، فالمناقلة هنا تحتاج لعمليتين، الأولى هي إنقاص المبلغ من الحساب المرسل، ثم اضافته الى الحساب المستقبل، فإذا حصل خطأ في أي من العمليتين السابقتين تعتبر المناقلة غير ناجحة ولا يتم بتم حفظ أية تعديلات.

خصائص ACID:

ACID هو اختصار يشير إلى مجموعة من 4 خصائص رئيسية تحدد المناقلة: الذرية Atomicity و الاتساق Consistency و العزلة Isolation و المتانة Durability. إذا كانت عملية قاعدة البيانات تحتوي على خصائص ACID هذه، فيمكن تسميتها مناقلة ACID، وتسمى أنظمة تخزين البيانات التي تطبق هذه العمليات أنظمة مناقلات Transactional Systems.

تضمن مناقلات ACID أن كل قراءة أو كتابة أو تعديل للجدول لها الخصائص التالية:

- **الذرية Atomicity:** يتم التعامل مع كل عبارة Statement في المناقلة (لقراءة البيانات أو كتابتها أو تحديثها أو حذفها) كوحدة واحدة. إما أن يتم تنفيذ العبارة بأكملها، أو لم يتم تنفيذ أي منها. تمنع هذه الخاصية حدوث فقدان البيانات وتلفها إذا حصل خطأ ما خلال تنفيذ العبارة Statement Execution، مثلاً انقطاع التيار الكهربائي خلال تنفيذ تعليمة تقوم بتعديل 1000 سجل.
- **الاتساق Consistency:** يشير الاتساق إلى الحفاظ على قيود سلامة البيانات، حيث لن تنتهك المناقلة المتسقة قيود التكامل الموضوعة على البيانات، يضمن فرض الاتساق أنه في حالة دخول قاعدة البيانات إلى حالة غير قانونية (في حالة حدوث انتهاك لقيود تكامل البيانات)، فإنه سيتم إحباط العملية وإرجاع التغييرات إلى حالتها القانونية السابقة، مثلاً ادخال قيمة سالبة الي حقل الرصيد في حساب بنكي.
- **العزلة Isolation:** وتعني أنه عندما يقوم العديد من المستخدمين بالقراءة والكتابة من نفس الجدول دفعة واحدة، فإن المناقلات المتزامنة لا تتداخل مع بعضها البعض أو تؤثر على بعضها البعض. يمكن أن يحدث كل طلب كما لو أن كل الطلبات كانت تحدث طلباً تلو الآخر، على الرغم من أنها تحدث في وقت واحد فعلياً.

- **المتانة Durability:** تضمن هذه الخاصية حفظ التغييرات التي تم إجراؤها على البيانات من خلال المناقلات المنفذة بنجاح، حتى في حالة فشل النظام.

أهمية مناقلات ACID:

تضمن مناقلات ACID أعلى وثوقيه Reliability وتكامل Integrity ممكنين للبيانات، حيث إنها تضمن عدم وقوع البيانات في حالة عدم تناسق بسبب عملية اكتملت جزئياً فقط. على سبيل المثال، بدون مناقلات ACID، إذا كنا نكتب بعض البيانات إلى جدول في قاعدة البيانات، لكن الطاقة انقطعت بشكل غير متوقع، فمن المحتمل أنه لم يتم حفظ سوى بعض البيانات ولم يتم حفظ البعض الآخر، في هذه الحالة أصبحت قاعدة البيانات الآن في حالة غير متسقة ومن الصعب للغاية استرجاعها وسيستغرق وقتاً طويلاً.

مبدأ CAP:

في عام 1999 قدم العالم Brewer, Fox ما سماه ب مبدأ CAP (CAP Principle) والذي عرف لاحقاً باسم نظرية CAP (CAP Theorem).

بشكل رئيسي فإن أي نظام قاعدة بيانات موزع يهتم باكتساب ثلاث صفات رئيسية:

1. **الاتساق Consistency:** وتعني أن أي عملية قراءة سوف تحصل على المعلومات المحدثة في آخر عملية كتابة.
2. **التوافر Availability:** وتعني أن النظام متاح بشكل دائم ويمكن إجراء عمليات قراءة منه في أي وقت.
3. **سماحية التقسيم Partition Tolerance:** القدرة على إضافة عقد بسهولة إلى النظام بحيث يتم توزيع البيانات ضمن هذه العقد لتحقيق عملية قياس أفقية Horizontal Scaling.

يشير مبدأ CAP إلى أنه لا يمكن لنظام قاعدة بيانات موزع، تحقيق هذه الصفات الثلاثة بشكل كامل ولكن يمكن تحقيق اثنتين من هذه الصفات الثلاث بشكل صحيح.

تم اثبات هذه النظرية لاحقاً في عام 2002 عن طريق ورقة بحثية للباحثين Nancy Lynch and Seth Gilbert من معهد MIT.

ابتكر Brewer قائمة بالخصائص تمت تسميتها لاحقاً بـ "تكوينات CAP" "CAP Configuration"، حيث يمكن استخدامها لتحديد نقاط القوة المعمارية لأنظمة قواعد البيانات المختلفة:

1- **CA:** "لا يمكن لقواعد البيانات تحقيق الاتساق والتوافر معاً إلا في حالة غياب التقسيم عبر شبكة" تعتبر أنظمة قواعد البيانات العلائقية قواعد بيانات "CA"، حيث تتمتع بالقدرة على توفير مناقلات ACID قوية.

2- **CP:** "يمكن تحقيق الاتساق والتقسيم عن الطريق التضحية بالتوافر". فمثلاً عند تحديث بيانات ضمن عقدة ما، يتم تحديث البيانات على جميع العقد التي تمثل تكرار Replica Set للعقدة المحدثة بشكل عاجل، وتكون العقد غير متوافرة حتى انتهاء عملية التحديث.

3- **AP:** "يمكن تحقيق التوافر والتقسيم عن طريق التضحية بالاتساق". فمثلاً عند فمثلاً عند تحديث بيانات ضمن عقدة ما، لا يتم تحديث البيانات على جميع العقد التي تمثل تكرار Replica Set للعقدة المحدثة بشكل عاجل، وبالتالي تكون العقد متوافرة ولكن من الممكن الحصول على نتائج غير متسقة من إحدى العقد إلى أن يتم تحديثها.

نتيجة:

بشكل أساسي، تختلف قواعد بيانات NoSQL من الناحية المعمارية عن قواعد البيانات العلائقية لأنها مصممة لجني مزايا أداء القراءة والكتابة عن طريق سماحية التقسيم (التوسع أفقياً Horizontal Scaling) مع ترك الاتساق أو التوافر للتفاوض بناء على متطلبات المشروع. معرفة هذا الاختلاف أمر بالغ الأهمية لفهم المواقف التي قد تكون فيها قاعدة بيانات NoSQL أكثر ملاءمة من RDBMS.

5-2 قابلية التوسع Scalability:

تعرف قابلية التوسع على أنها القدرة على جعل النظام يستطيع تحمل المزيد من البيانات، المستخدمين، الطلبات والعمليات بحيث يلبي أهدافه بتكلفة مناسبة بدون التأثير على أداء واستجابة النظام User Experience وفي نفس الوقت إمكانية الزيادة Scale Up أو النقصان Scale Down عند الحاجة.

عادة التحديات التي يحتاج أي نظام بدء الضغط عليه هي:

- **التعامل مع المزيد من البيانات Data :**

وهي أول هذه التحديات وأشهرها ايضاً، فكلما توسع المشروع وأصبح شائعاً فكلما زاد البيانات فيه أكثر وأكثر. لذلك يجب أن يتم التعامل معها بشكل أكثر كفاءة مثلاً حسابات المستخدمين User Accounts ، المنتجات Products وغيرها من المعلومات في النظام. معالجة هذه المعلومات سوف يضع حملاً على النظام فهذه المعلومات من الممكن أن يتم البحث خلالها Search، أو جلبها من ملفات Reading ، أو كتابتها على القرص Writing ، أو تُرسل عبر الشبكة Sending. وهذه الأيام مع توفر أدوات تحليل البيانات الضخمة Big Data Analytics فالشركات أصبحت شهيتها مفتوحة لتخزين المزيد والمزيد من البيانات.

- **التعامل مع المزيد من الطلبات في نفس اللحظة Concurrency :**

وتعنى عدد المستخدمين الذين يمكن أن يستخدموا النظام في نفس اللحظة بدون التأثير على الأداء، وهذا الأمر صعب لأن السيرفرات التي يعمل فيها النظام لديها عدد محدود من الموارد (عدد معين من المعالجات CPUs) وال Threads ايضاً. وكلما زاد ال Concurrent Users كلما زاد عدد الطلبات المفتوحة Open Connections وزاد عدد ال Active Threads وزادت الأشياء التي ينبغي معالجتها في نفس اللحظة وبالتالي يزيد ال CPU Context Switching .

- **التعامل مع زيادة معدل ارسال واستقبال المعلومات Interaction Rate :**

ونقصد معدل ارسال واستقبال المعلومات بين المستخدمين والنظام وهي قريبة من فكرة ال Concurrency ولكن باختلاف بسيط، على سبيل المثال في المواقع العادية فإن المستخدمين يقوموا بالتحرك في صفحاته كل 15 او 20 ثانية بينما مثلاً في الألعاب المتعددة المستخدمين فمعدل ارسال واستقبال المعلومات يكون أكثر من مرة في الثانية. لذلك فإن معدل التفاعل Interaction Rate يمكن أن يكون أكبر أو اقل على حسب نوع التطبيق الذي نقوم ببنائه

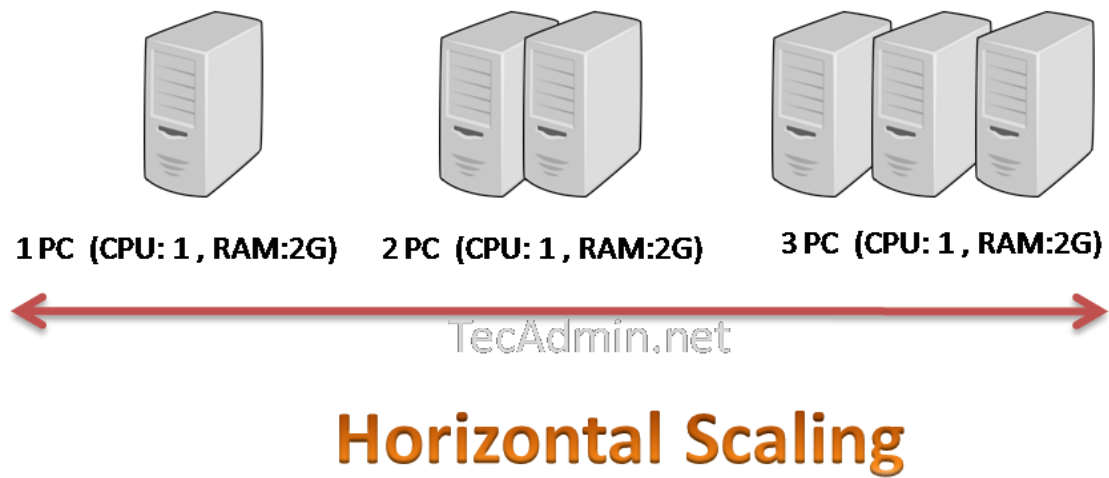
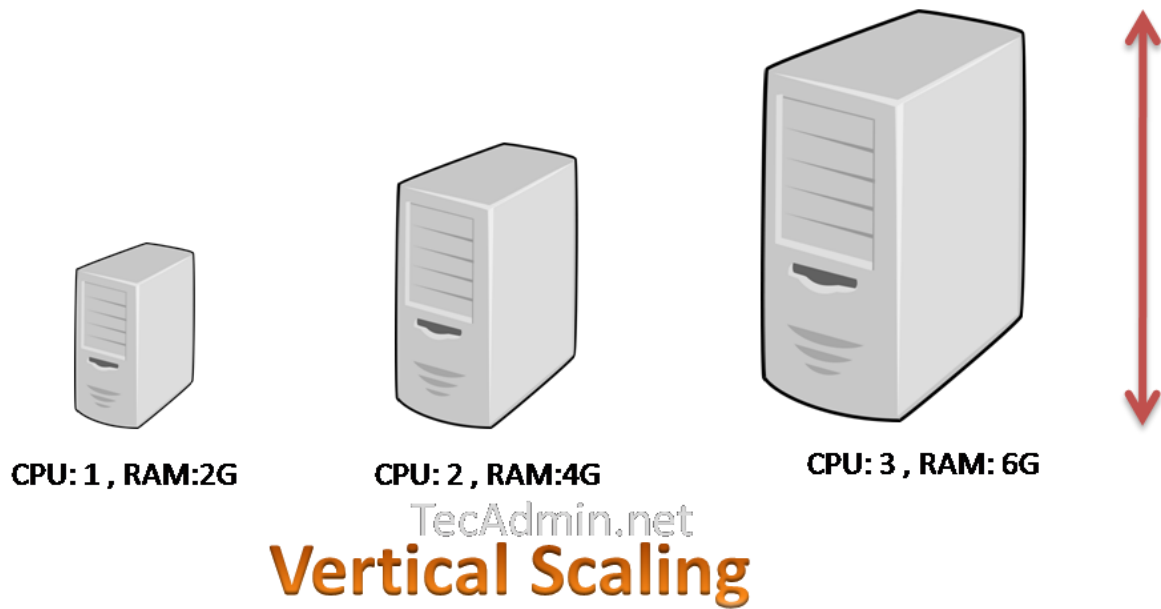
مثال: إذا كان لديك 100 مستخدم حالي Concurrent وكل منهم يرسل طلب كل 5 ثواني ففي هذه الحالة سوف نحتاج الى ان تكون استجابة الموقع هي حوالي 20 طلب في الثانية ويمكن ان توضح ضمن المتطلبات للمشروع (أي Performance requirement). فالأداء Performance هنا يقيس الوقت المطلوب لخدمة 20 طلب في الثانية، بينما ال Scalability تقيس كم أكبر عدد مستخدمين وطلبات يمكن معالجتهم بدون تقليل في الأداء.

قابلية التوسع العمودية وقابلية التوسع الأفقية Vertical Scaling vs Horizontal Scaling:

عندما نريد القيام بعملية توسع من أجل استيعاب الحمل المتزايد للبيانات المطبق على النظام، فإننا نستطيع القيام بالعملية بطريقتين، أفقية أو عمودية.

التوسع العمودي يعني استخدام سيرفرات ذات قدرة حاسوبية أعلى، الأمر الذي من شأنه زيادة الموارد المخصصة للنظام وبالتالي التسريع من انجاز الوظائف، المشكلة في هذه الطريقة هي أنها قد تكون محدودة في حال كانت معدل تضخم البيانات في النظام يفوق القدرات الحاسوبية المتوفرة في السوق، وأيضاً الحصول على حواسيب ضخمة قد يكون ذو كلفة اقتصادية كبيرة.

في التوسع الأفقي بدلاً من اللجوء الى سيرفر بقدرات عالية يتم توزيع الحمل على عدة سيرفرات ذو قدرات متشابهة للسيرفر الحالي، هذه الطريقة تعتبر حلاً للمشكلة السابقة في التوسع الأفقي حيث أنه كلما زاد معدل نمو البيانات يمكن زيادة عدد السيرفرات، ولكن تطبيقها يحتاج الى مهندسين ذو خبرة عالية لأنها ليست بسهولة تشغيل النظام مرة أخرى على سيرفر بقدرة أعلى.



الفصل الثالث

تجربة عملية

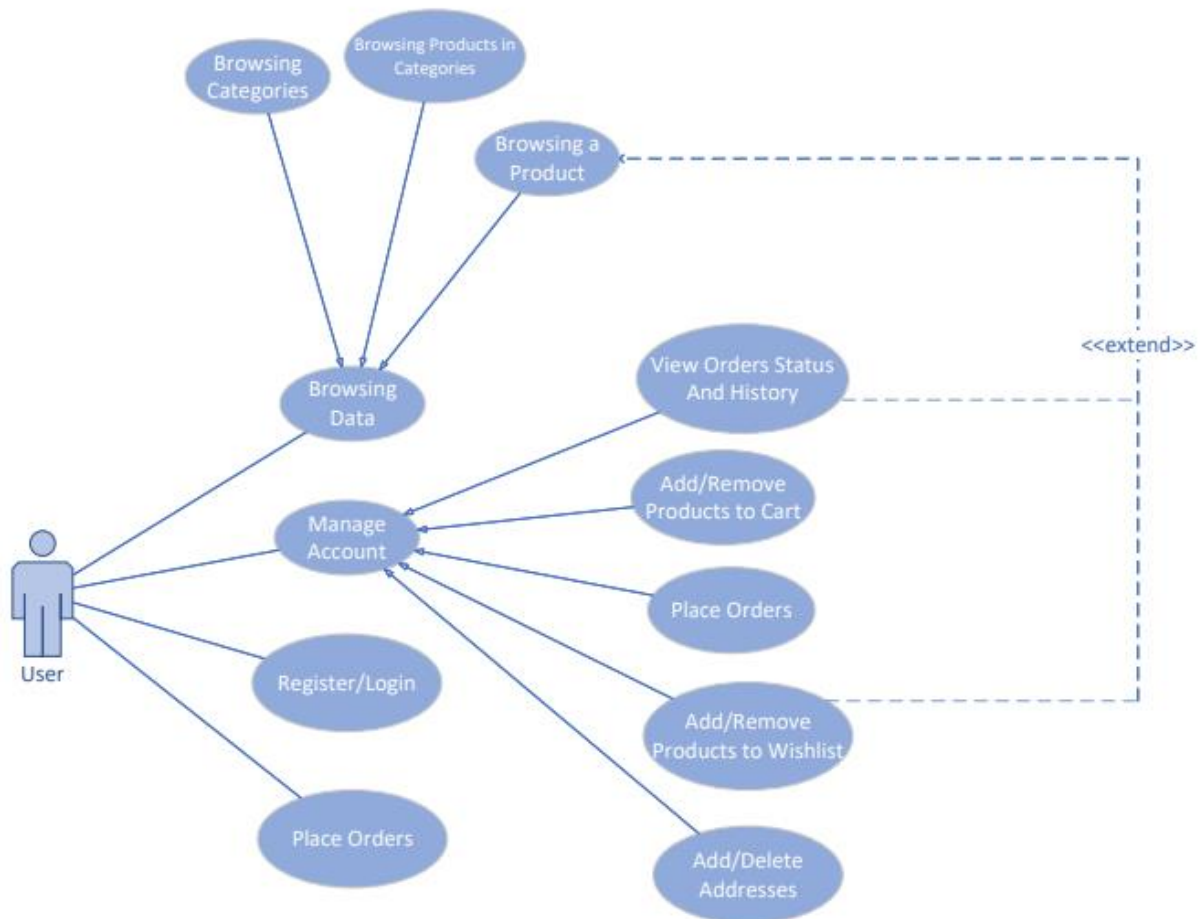
سنقوم في هذا الفصل بتحليل وبناء نظام متجر إلكتروني بسيط من نوع B2C وسنستخدم كل من MongoDB و SQL Server كقاعدة بيانات من أجل هذا النظام، وسندرس أهم الخصائص والفوارق في كل من النظامين وسنقوم باختبار الأداء لبعض وظائف المتجر في كل من قاعدتي البيانات السابق ذكرهما.

1-3 التوصيف الوظيفي للنظام:

بشكل بسيط، فإن المتجر الإلكتروني يحوي الوظائف الرئيسية التالية:

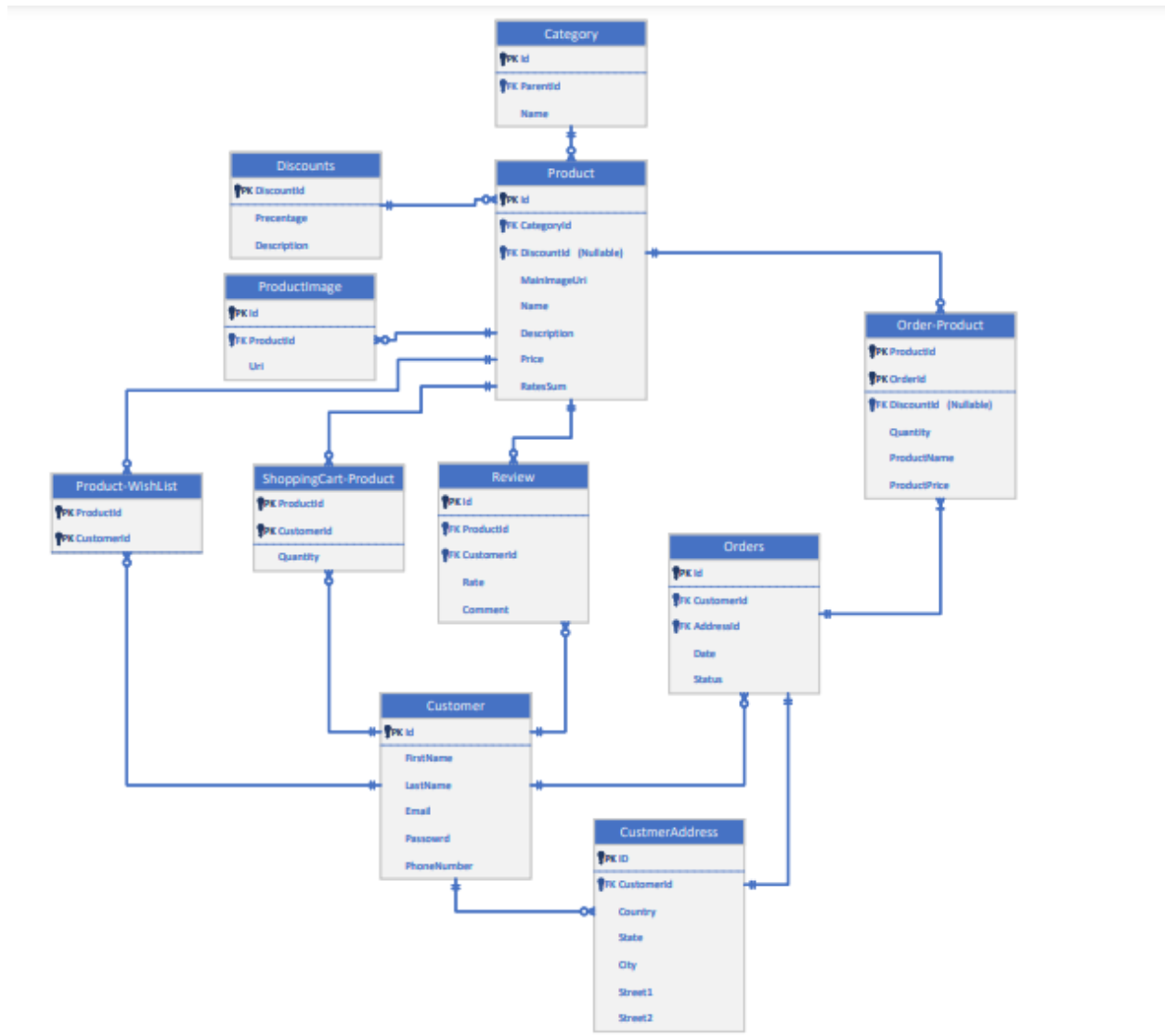
- إنشاء حساب جديد أو تسجيل الدخول.
- إمكانية اختيار صنف من الأصناف وتصفح المنتجات بداخله.
- اختيار أحد المنتجات وتصفح المعلومات الخاصة به.
- إضافة المنتجات إلى عربة التسوق وقائمة الأمان.
- مشاهدة والتعديل على المنتجات الموجودة في عربة التسوق وقائمة الأمان.
- القيام بطلب شراء المنتجات.
- إضافة عنوان أو أكثر للمستخدم.
- تتبع تاريخ وحالة طلبات الشراء.

2-3 مخطط حالة الاستخدام:



3-3 مخطط ERD :ERD

يُستخدم مخطط ERD بشكل رئيسي لتوصيف قاعدة بيانات علائقية لنظام ما حسب متطلباته، ويُبنى في ثلاث مستويات هي المستوى المفاهيمي والمستوى المنطقي والمستوى الفيزيائي، ويتم فيه توضيح الكيانات المكونة للنظام والعلاقات فيما بينها والتعددية في كل علاقة من هذه العلاقات، يوضح الشكل في الأسفل مخطط ERD في المستوى المنطقي لنظام المتجر الإلكتروني الذي تم توصيفه سابقاً.



4-3 توصيف بنية النظام في MongoDB:

```
Category{
  "Id": " ",
  "Name": " ",
  "ParentId": " "
}
```

```
Discount{
  "Id": " ",
  "Percentage": " ",
  "Description": " ",
}
```

```
Customer
{
  "FirstName": " ",
  "LastName": " ",
  "Password": " ",
  "Email": " ",
  "PhoneNumber": " ",
  "Addresses": [
    {
      "Country": " ",
      "State": " ",
      "City": " ",
      "Street1": " ",
      "Street2": " "
    }
  ],
}
```

```
Product
{
  "Id": " ",
  "Name": " ",
  "Price": " ",
  "Description": " ",
  "RatesSum": " ",
  "Discount":{
    "Id": " ",
    "Percentage": " ",
    "Description": " ",
  },
  "Images": [
    "uri"
  ],
  "CategoryId": " ",
  "CategoryName": " ",
  "LatestReviews": [
    {
      "Id": " ",
      "CustoemerId": " ",
      "CustomerName": " ",
      "Rate": " ",
      "Comment": " ",
    }
  ]
}
```

WishList

```
{
  "CustoemerId": " ",
  "Items": [
    {
      "ProductId": " ",
      "ProductName": " ",
      "ProductPrice": " ",
    }
  ]
}
```

Cart

```
{
  "CustomerId": " "
  "Items": [
    {
      "ProductId": " ",
      "ProductName": " ",
      "ProductPrice": " ",
      "Quntity": " "
    }
  ]
}
```

Review

```
{
  "Id": " ",
  "CustoemerId": " ",
  "CustomerName": " ",
  "Rate": " ",
  "Comment": " ",
}
```

Order

```
{
  "Id": " ",
  "CustomerId": " ",
  "Date": " ",
  "Status": " ",
  "Subtotal": " ",
  "Items": [
    {
      "ProductName": " ",
      "ProductPrice": " ",
      "ProductSubTotal": " ",
      "DiscountPercentage": " ",
      "Quntity": " ",
    }
  ]
}
```

5-3 نظرة على أبرز الاختلافات بين النموذجين:

نلاحظ أنه في ال Scheme الخاصة ب MongoDB تم كسر قواعد تطبيع البيانات حيث يمكن أن يوجد تكرار في المعلومات ضمن أكثر من مكان، وفيما يلي نبين أبرز نقاط الاختلاف في التصميم بين النموذجين:

- نلاحظ أنه في ال ERD الخاص بنموذج قاعدة البيانات العلائقية يتم تخزين معرف الحسم Discount Id فقط ضمن كيان المنتج Product Entity، وبالتالي عندما نريد قراءة المنتج مع نسبة الحسم الخاص به ووصف هذا الحسم فإنه يتوجب علينا اجراء عملية JOIN لجدول الحسم مع جدول المنتج، أما في نموذج MongoDB فإننا نكرر حفظ نسبة الحسم ووصف الحسم في كل منتج وبالتالي فإنه من المتوقع أن تكون قراءة المعلومات اللازمة للمنتج هنا أسرع بما أن القراءة تتم من مكان واحد.
- الأمر نفسه بما يتعلق بالصنف الخاص بالمنتج، حيث يتم في SQL Server تخزين معرف الصنف فقط أما في MongoDB فيتم تكرار حفظ اسم الصنف في المنتجات.
- في ERD تم استخدام جدول خاص من أجل حفظ الصور الخاصة بالمنتجات، ويتم استخدام معرف المنتج Product Id كمفتاح أجنبي في الجدول الخاص بصور المنتجات، وهذه هي طريقة تحقيق العلاقة 1 to N ضمن صيغة تطبيع البيانات 3NF، أما في MongoDB فلا يوجد داعي لإنشاء مجموعة خاصة بصور المنتجات حيث يمكن تخزين لائحة من القيم داخل أي مستند، وبالتالي قمنا بتخزين الصور الخاصة بالمنتج ضمن المستند نفسه الأمر الذي من شأنه أن يحسن من أداء القراءة أيضاً بسبب أن القراءة تتم من مكان واحد، أما في حالة SQL Server، حتى نقوم باستعادة المنتج مع الصور الخاصة به فإننا أمام خيارين، الأول أن نقوم بعملية JOIN واستخدام تابع التجميع STRING_AGG على الحقل Uri، بهذه الحالة سيمكن قراءة المنتج مع الصور باستعلام واحد ولكن سيتوجب فصل الصور المجمعة في الكود البرمجي، الخيار الثاني هو أن نقوم بقراء معلومات المنتج باستعلام أول، ثم نقوم بقراءة الصور الخاصة به باستعلام ثاني.
- في قاعدة البيانات العلائقية تكون العلاقة بين الطلب Order والمنتج Product من نوع N to N، ونلاحظ أننا قمنا باستخدام جدول كسر في مخطط ERD باسم Order_Products، والبيانات المخزنة في هذا الجدول لا تعتبر تكرار للبيانات من جدول المنتجات، وذلك لأنه يجب حفظ هذه البيانات حسب قيمها عند اجراء طلب الشراء، فمثلاً عند تحديث سعر منتج ما لا يتم تحديث سعر المنتج في طلبات الشراء السابقة، عندما نريد قراءة معلومات الطلب مع ومعلومات المنتجات فيه، فإنه لا يمكن استخدام التوابع التجميعية كما في المثال السابق بين المنتج وصوره، وذلك لأن بنية المعلومات المراد قراءتها هنا أكثر تعقيداً وليسيت حقلاً نصياً فقط كما في الحالة السابقة، وبالتالي فإننا حصرنا بحاجة لإجراء استعلامين، الأول يتم فيه جلب المعلومات الخاصة بالطلب والثاني يتم فيه جلب المعلومات الخاصة بعناصر هذا الطلب، أما في MongoDB فإنه يتم تخزين عناصر الطلب

داخل مستند الطلب نفسه بالاستفادة من الخاصية Nested Documents وبالتالي تتم القراءة من مكان واحد الأمر الذي من شأنه أن يحسن من أداء القراءة.

- في قاعدة البيانات العلائقية تم استخدام جدول خاص لكل من سلة المشتريات وقائمة الأمانى يتم فيه تخزين معرف العميل ومعرف المنتج، وبالتالي عندما يقوم المستخدم باستعراض السلة أو قائمة الأمانى الخاصة به يتم جلب معرفات المنتجات المخزنة الخاصة بالزبون من الجدول ثم جلب معلومات المنتجات من الجدول الخاص بها، في MongoDB تم استخدام مجموعة خاصة أيضاً لكل من السلة وقائمة الأمانى يتم فيهما تخزين معلومات المنتج مباشرة، إن هذه الطريقة وعلى الرغم من أنها قد تسرع جلب المنتجات من السلة أو قائمة الأمانى، إلا أنها تفرض قيود خاصة بها في عملية التحديث فعند تحديث سعر المنتج يجب أن يتم تحديثه بشكل مباشر في جميع السلل الشرائية وقوائم الأمانى ولا يمكن اعتماد مبدأ ال Eventual Consistency حيث أن العميل سيرى السعر القديم للمنتج في السلة الى أن يتم التحديث.

تم سرد أبرز نقاط الاختلاف في التصميم بين نموذجي SQL Server و MongoDB، يترك للقارئ تصفح بقية الاختلافات حيث أنها بنفس مبدأ النقاط التي تم ذكرها في الأعلى.

نلاحظ أن تصميم نموذج للبيانات في قاعدة بيانات علائقية باتباع صيغة تطبيع البيانات القياسية 3NF يفرض قيود محددة، هذه القيود تؤدي في غالب الأحيان الى الحاجة لإجراء عملية JOIN بين جدولين أو أكثر للحصول على البيانات اللازمة لعمل التطبيق، يُعتبر هذا الأمر كنقطة سلبية في التطبيقات ذات الحمل العالي التي يتم فيها قراءة البيانات بشكل كبير، ولكن في المقابل فإن هذه القيود تضمن أن النظام في حالة مستقرة وصحيحة دائماً وإن التعديل على المعلومات لن يترك بيانات النظام في حالة خاطئة أو غير مستقرة، ويكفي التعديل على البيانات في مكانها الأصلي وسينعكس هذا التغيير على جميع القراءات التالية بشكل مباشر.

أما في قواعد البيانات غير العلائقية فإنه لا توجد قيود في التصميم، الأمر الذي يترك للمطور تصميم البيانات بالطريقة التي يحتاجها التطبيق، الأمر الذي قد يؤدي الى تسريع قراءة البيانات بطريقة ملحوظة، ولكن هذا التحسين في الأداء بالقراءة ليس مجانياً وإنما أتى على حساب أن البيانات أصبحت موزعة ومكررة في النظام، وتعديل البيانات في هذه الحالة يصبح معقداً وبحاجة الى دقة تفادياً لوقوع الأخطاء.

6-3 هل يمكن الاستغناء عن تطبيع البيانات؟

إن السهولة في العمل والمرونة في التصميم وعدم اشتراط وجود بنية Scheme معرفة مسبقاً للتخزين، الأمور التي توفرهما قواعد البيانات غير العلائقية جذبت الكثير من المطورين الى هذا النموذج، وقد ازدادت شعبيتها في تطبيقات الويب بشكل ملحوظ، وذلك لأن كثيراً ما يكون هناك حاجة للتعامل مع بيانات ليس لها بنية واضحة ومحددة في ذلك النوع من التطبيقات، كل ما سبق يدفعنا الى التساؤل: هل يمكن الاعتماد على البيانات بدون تطبيع بشكل دائم؟

إن الإجابة على السؤال السابق هي حكماً ب لا، وإن اختيار تخزين البيانات بطريقة Normalized أم لا يجب أن يتم دراسته بين فريق التطوير من أجل كل مشروع على حدى، فكل تطبيق له شروطه وحالاته الخاصة التي يجب مناقشتها.

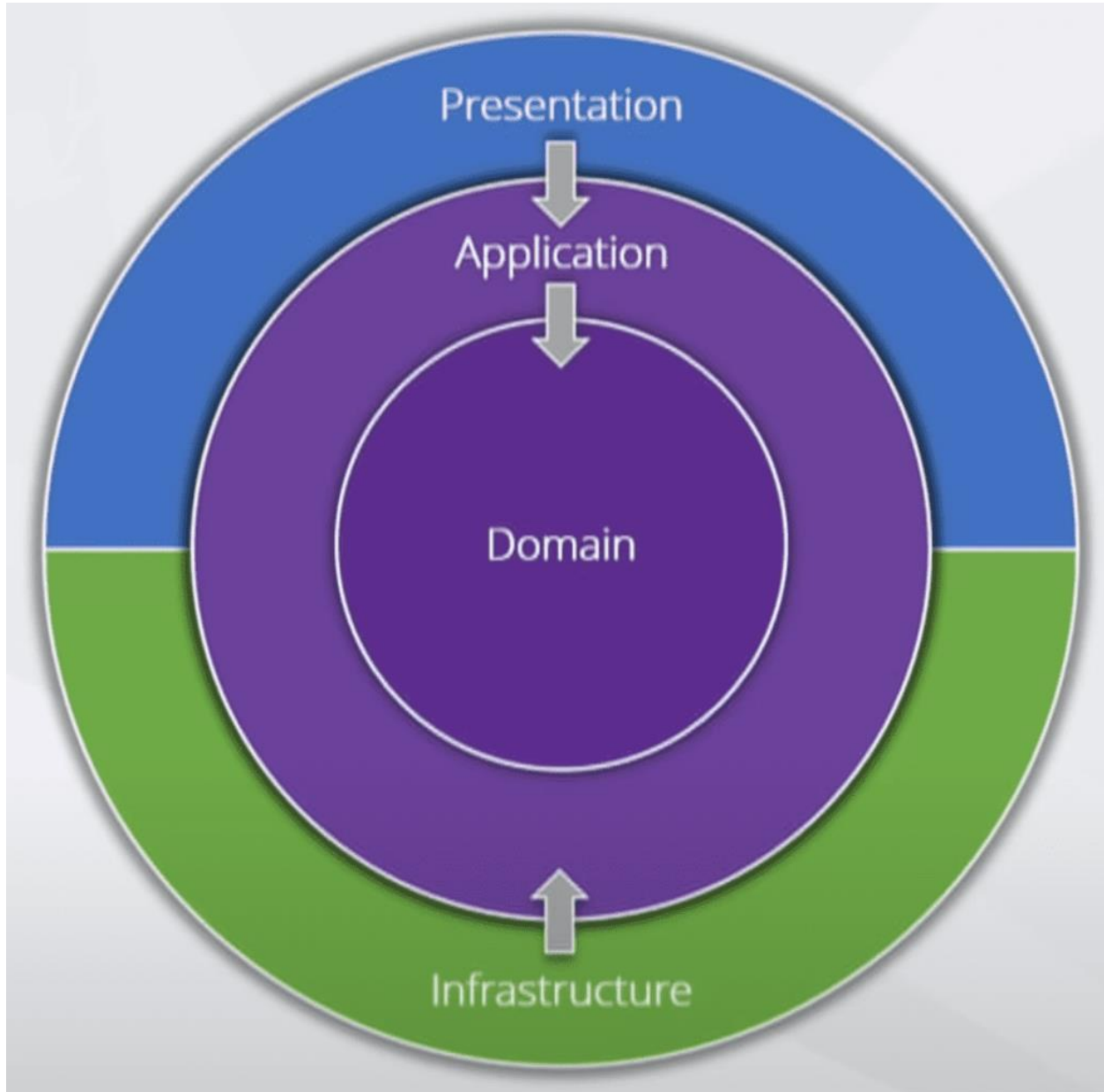
إضافة الى ذلك، إن افتقار قواعد البيانات العلائقية على تزويد مناقلات ACID بشكل كامل، يجعل من استخدامها في التخزين أمراً حرجاً، فمثلاً عند القيام بعملية نقل مبلغ مالي بين حسابين مصرفيين، يجب هنا حصر أن يتم التعديل بشكل مباشر على الحسابين حتى تتم المناقلة بنجاح، ولا يمكن السماح بالتعديل على الحسابين على شكل Eventual Consistency، أي أن المناقلة ذرية ولا توفر جميع أنظمة إدارة قواعد البيانات غير العلائقية هذا النوع من المناقلات.

نستنتج مما سبق أن تخزين البيانات بدون تطبيع له ميزاته ولكن شرط ان لا يسبب أخطاءً كارثية في النظام، فمثلاً نادراً ما نحتاج الى تعديل اسم صنف من الأصناف الموجودة في المتجر، وبالتالي يمكن التوضحية بأداء عملية التعديل مقابل تحسين أداء عملية القراءة، كما أن التعديل على شكل Eventual Consistency مسموح في هذه الحالة، إذ أن عدم ظهور الاسم الجديد للصنف بعد التعديل لفترة من الوقت هو ليس بالخطأ الكارثي بالنسبة للنظام، وبالتالي يمكن تعديل الاسم القديم للصنف بشكل تدريجي ولا حاجة لجعل النظام غير متوافر من أجل عملية التعديل هذه، ولكن في بعض الأحيان، يتطلب النظام عملية تعديل فورية في سياق مناقلات ذرية، ويتم تحديد هذه الأمور عند دراسة متطلبات النظام من قبل المحللين والمطورين.

7-3 شرح لقسم ال Backend الخاص بالمتجر:

لقد استخدمنا لبناء هذا القسم الأسلوب المعماري المسمى Clean Architecture، يبين الشكل في الأسفل الطبقات الأساسية المشكلة لهذا الأسلوب المعماري وهي عبارة عن أربع طبقات:

.Domain Layer, Application Layer, Infrastructure Layer, and Presentation Layer

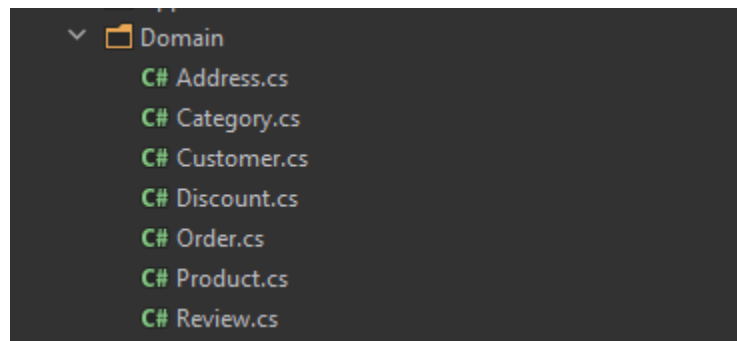


إن شرح هذا الأسلوب المعماري هو خارج نطاق بحثنا هذا لذا سوف نتطرق باختصار شديد الى شرح المكونات في كل طبقة.

:Domain Layer

في هذه الطبقة تتوضع الكيانات (الأصناف) Entities (Classes) الخاصة بنطاق المسألة Problem Domain التي يحلها التطبيق.

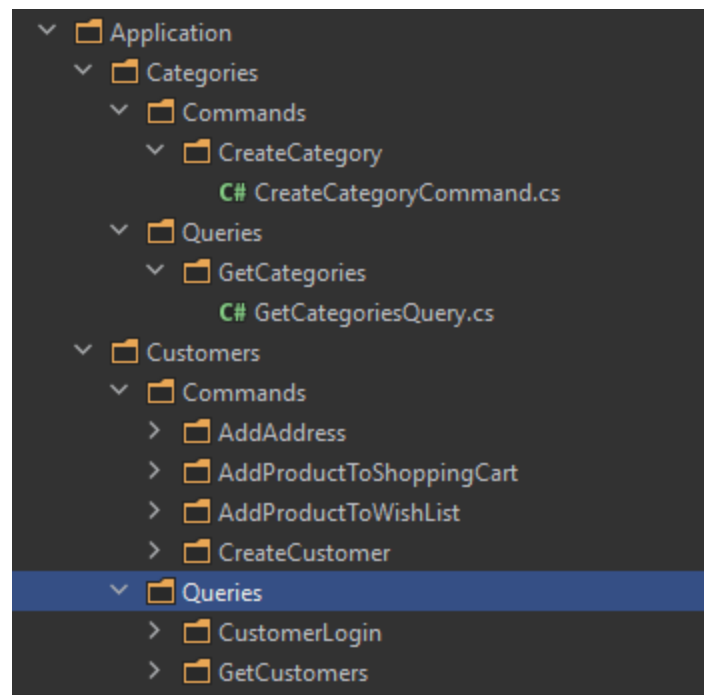
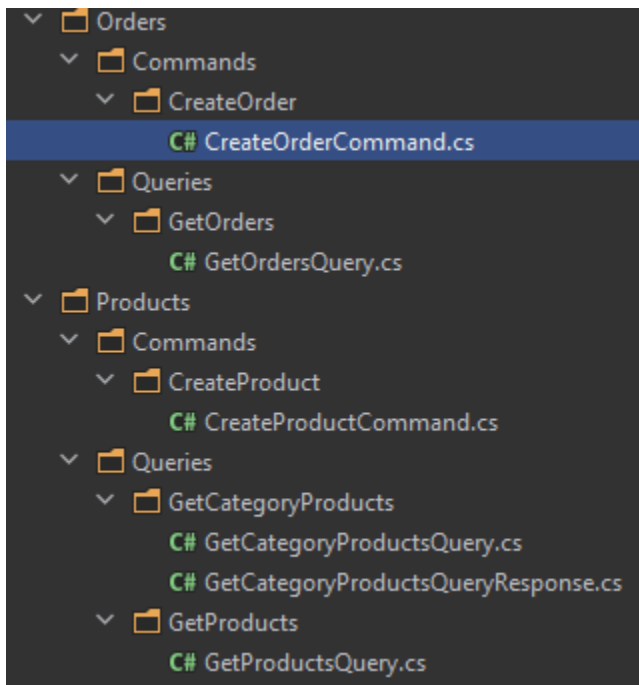
يبين الشكل التالي توضع الأصناف الخاصة بالمتجر ضمن طبقة ال Domain.



:Application Layer

في هذه الطبقة يتم وضع ال Business Logic الخاص بالتطبيق بشكل مجرد Abstract، أي دون معرفة أجزاء البنية التحتية كقواعد البيانات ومزودات خدمات الایمیل الإلكتروني وبوابات الدفع وغيرها من التفاصيل. في متجرنا الإلكتروني قمنا بتعريف ال Business Logic الخاص بالمتجر باستخدام النمط CQRS، أي على شكل مجموعة من ال أوامر Commands والاستعلامات Queries.

تبين الاشكال التالية بعض الأجزاء من طبقة ال Application الخاصة بالمتجر الإلكتروني.



كما قلنا سابقاً فإن هذه الطبقة تكون مجردة من تفاصيل البنية التحتية، فإذا نظرنا إلى أحد الأصناف المشكلة لأمر أو استعلام ما في التطبيق سنجد أنه يحوي على الخصائص Properties اللازمة لمعالجة هذا الأمر أو الاستعلام دون أن يحوي على كود المعالجة الفعلي، وسنجد أنه يرث من الواجهة البرمجية IRequest وهي عبارة عن Interface خاص بالمكتبة MediatR سنتعرف إليه أكثر في الطبقة التالية.

```
public class CreateOrderCommand : IRequest<Result>
{
    [3 usages]
    public Guid Id { get; set; }
    [2 usages]
    public DateTime Date { get; set; }
    [1 usage]
    public double Subtotal { get; set; }
    [2 usages]
    public Guid CustomerId { get; set; }

    [1 usage]
    public string CustomerName { get; set; } = "";

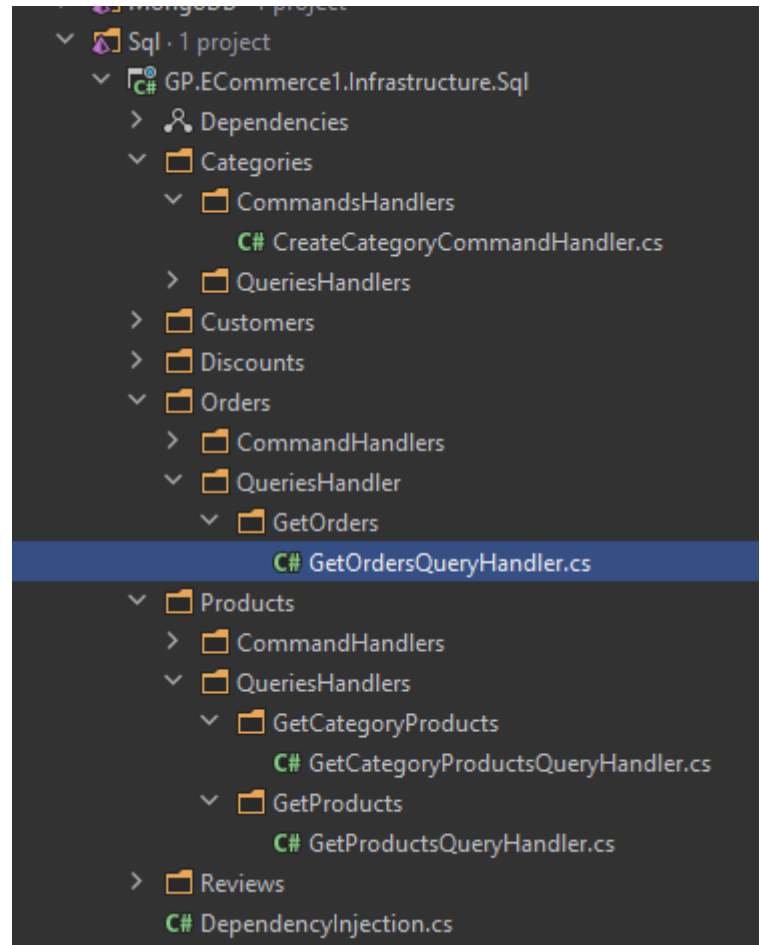
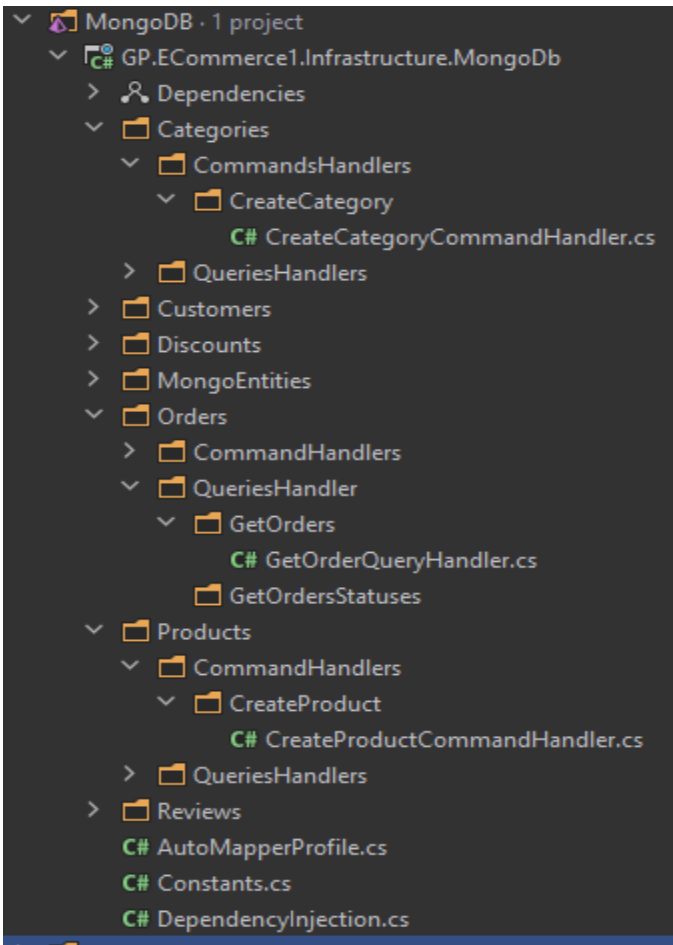
    [2 usages]
    public Address Address { get; set; } = new();

    [2 usages]
    public OrderStatus Status { get; set; }

    [2 usages]
    public List<OrderItem> Items { get; set; } = new();
}
```

:Infrastructure Layer

في هذه الطبقة يتم تنفيذ التفاصيل المجردة المعرفة في طبقة ال Application، والميزة في هذا الأمر أنه يمكن استخدام طبقة Application و Domain مرة واحدة فقط وكتابة طبقة Infrastructure مختلفة، وهذه الحالة هي مناسبة تماماً لبحثنا فطبقة ال Application وال Domain للمتجر هي نفسها سواء استخدمنا MongoDB أو SQL Server حيث أن وظائف المتجر ليس لها علاقة بنوع قاعدة البيانات المستخدمة. يعتبر هذا الأمر هو من أهم مميزات الأسلوب المعماري Clean Architecture لأنه يسمح للتطبيقات أن تغيّر من تفاصيل بنيتها التحتية بسهولة دون الحاجة للتعديل على الأجزاء المجردة من البنية التحتية. بناءً على ما سبق سنرى الآن أجزاء من تنفيذ البنية التحتية للمتجر حيث تم تنفيذها مرة باستخدام SQL ومرة ثانية باستخدام MongoDB، ويكون التنفيذ بكتابة Handler Class من أجل كل Command / Query تم تعريفه في طبقة ال Application.



نلاحظ أن بنية طبقة ال Infrastructure في النموذجين متشابهة الى حد كبير مع وجود بعض الاختلافات، بغض النظر عن هذا التشابه حيث يمكن بناء كل طبقة بالطريقة المناسبة لها، يبقى الغرض الأساسي للطبقة بأن تقوم بتنفيذ العمليات المجردة التي تم تعريفها في طبقة ال Application.

تبين الأشكال في الأسفل مثلاً على تنفيذ استعلام في كل من النموذجين، ونلاحظ هنا أن التنفيذ يختلف بشكل كلي وذلك حسب المكتبات والتوابع الخاصة لقاعدة البيانات في كل نموذج.

```
namespace GP.ECommerce1.Infrastructure.MongoDb.Categories.QueriesHandlers.GetCategories;

public class GetCategoriesQueryHandler : IRequestHandler<GetCategoriesQuery, Result<List<Category>>>
{
    private readonly IMapper _mapper;
    private readonly IMongoDatabase _database;

    public GetCategoriesQueryHandler(IMongoClient client, IMapper mapper)
    {
        _mapper = mapper;
        _database = client.GetDatabase(Constants.GetDatabaseName());
    }

    public async Task<Result<List<Category>>> Handle(GetCategoriesQuery request, CancellationToken cancellationToken)
    {
        var result = new Result<List<Category>> { IsSuccess = true };
        try
        {
            var collection = _database.GetCollection<MongoEntities.Category>(Constants.CategoriesCollectionName);
            var categories :List<Category>? = await collection.Find( filter: new BsonDocument()).ToListAsync(cancellationToken);

            var c = _mapper.Map<List<Category>>(categories);
            result.Value = c;
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            result.IsSuccess = false;
            result.Error = e.Message;
        }

        return result;
    }
}
```

```

public GetCategoryProductsQueryHandler(SqlConnection connection)
{
    _connection = connection;
}

public async Task<Result<GetCategoryProductsQueryResponse>> Handle(GetCategoryProductsQuery request,
    CancellationToken cancellationToken)
{
    var result = new Result<GetCategoryProductsQueryResponse> {IsSuccess = true};
    string stmt = @"SELECT Products.Id, Price, Name, Products.MainImageUri, Discounts.Percentage as 'Discount'
    FROM Products
    INNER JOIN Discounts on Discounts.Id = Products.DiscountId
    Where CategoryId = @CategoryId";

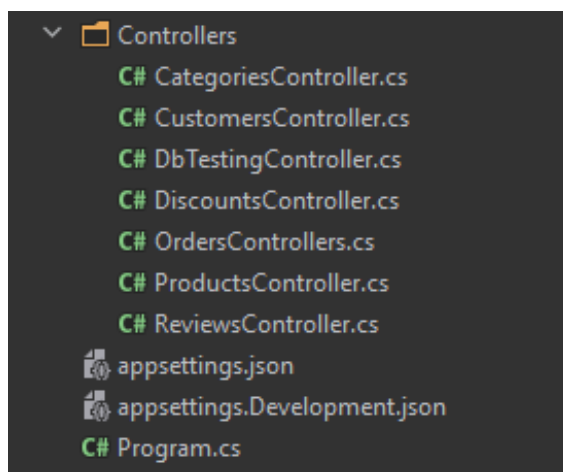
    var command = new SqlCommand(stmt, _connection);
    command.Parameters.AddWithValue("parameterName: "@CategoryId", request.CategoryId);
    List<GetCategoryProductsQueryResponseEntry> entries = new();
    try
    {
        _connection.Open();
        var stopWatch = new Stopwatch();
        int x = 0;
        stopWatch.Start();
        var reader = await command.ExecuteReaderAsync(cancellationToken);

        while (reader.Read())
        {
            var entry = new GetCategoryProductsQueryResponseEntry
            {
                Id = Guid.Parse(Convert.ToString(reader["Id"])),
                Name = Convert.ToString(reader["Name"]),
                Price = Convert.ToDouble(reader["Price"]),
                MainImageUri = Convert.ToString(reader["MainImageUri"]),
                DiscountPercentage = Convert.ToInt32(reader["Discount"])
            };
            entries.Add(entry);
        }
    }
}

```

:Presentation Layer

تمثل هذه الطبقة الواجهة التي يتم من خلالها الوصول للتطبيق، ولا يقصد هنا بالواجهة الرسومية وإنما الواجهة البرمجية التي يمكن للتطبيقات الخارجية والمستخدمين الوصول الى التطبيق من خلالها، وبما أننا نقوم ببناء تطبيق ويب فقد قمنا ببناء هذه الواجهة على شكل نقاط وصول Endpoints يمكن الوصول اليها باستخدام أي أداة تتيح ارسال طلبات HTTP.



8-3 توليد البيانات من أجل الاختبار:

تعتبر أنظمة المتاجر الالكترونية من الأنظمة الشائعة جداً على الانترنت، ويمكن إيجاد الكثير من ال Data Set التي يمكن استخدامها لملئ قاعدة البيانات من أجل اختبارات الأداء، ولكن تطبيق هذا الأمر لملئ قاعدة البيانات لم يكن ممكناً بشكل فعال وذلك أولاً لأننا بنينا نموذج خاص بنا والبيانات المتوفرة لا تطابق نموذجنا الخاص تماماً، وبالتالي يجب اجراء تحويلات من تلك النماذج الى نموذجنا الخاص الأمر الذي لم يكن سهلاً على الإطلاق، ثانياً أن Data Set ذات الجودة العالية غالباً ما تكون مدفوعة، وبالتالي اللجوء ل Data Set جاهزة لم يكن ممكناً.

خيارنا الآخر كان استخدام بعض البرامج المخصصة لملئ قاعدة البيانات، ولكن مجدداً فإن البرامج القادرة على القيام بهذا الأمر بشكل فعال وصحيح ليست متوفرة بشكل مجاني.

لذلك قررنا بناء أدواتنا الخاصة لتوليد البيانات والتي تتكون من مجموعة من الأصناف والتوابع التي تقوم بملء قاعدة البيانات بطريقة صحيحة حسب النموذج الخاص بنا.

في بداية الأمر كنا نقوم بتوليد وإدخال البيانات مباشرة الى قاعدة البيانات، وهذا الأمر كان له عدة تبعات سلبية، حيث بما أننا نستطيع فقط استخدام قاعدة بيانات واحدة خلال كل عملية تشغيل (أي قاعدة بيانات واحدة من أجل كل Process) فإنه سيتم توليد بيانات مختلفة من أجل كل قاعدة بيانات وهذا الأمر لا يعتبر مثالي ضمن بيئة اختبار حتى وإن كانت طريقة التوليد متشابهة، أيضاً فإننا أثناء اجراء الاختبارات بشكل تراكمي سنحتاج معلومات من قاعدة البيانات لنقوم بالاختبار حسبها، وبالتالي سيتوجب علينا القيام بالقراءة من قاعدة البيانات قبل اجراء كل اختبار مما سيؤخر زمن الاختبار.

لتدراك هذه المشكلة قمنا بفصل توابع التوليد عن توابع الملء لقاعدة البيانات، حيث نقوم أولاً بتوليد البيانات وحفظها في ملفات بصيغة JSON، بعد ذلك نملئ قاعدتي البيانات بالبيانات المولدة مسبقاً.

```
public static void GenerateAndStoreAsJson(string fileName, string categoriesFileName, string discountsFileName,
    int count)
{
    Console.WriteLine("Generating Products...");
    var categories :List<Category> = Task.Run( function: () => CategoriesSeeder.GetAllCategories(categoriesFileName)).Result;
    var discounts :List<Discount> = Task.Run( function: () => DiscountsSeeder.GetAllDiscounts(discountsFileName)).Result;
    List<Product> products = new();
    var stopWatch = new Stopwatch();
    stopWatch.Start();
    for (int i = 1; i <= count; i++)
    {
        var imagesUri = new List<string>();
        for (int j = 1; j <= Randoms.RandomInt(0, 8); j++)
        {
            imagesUri.Add(_defaultImageUrl);
        }

        var childCategories :List<Category> = categories.Where(c :Category => c.ParentId != null).ToList();
        var category = childCategories[Randoms.RandomInt(childCategories.Count)];
        var discount = discounts[Randoms.RandomInt(discounts.Count)];

        var product = new Product
        {
            Id = Guid.NewGuid(),
            CategoryId = category.Id,
            CategoryName = category.Name,
            Discount = Randoms.RandomBoolean() ? discount : null,
            Description = _productsDescriptions[Randoms.RandomInt(_productsDescriptions.Length)],
            Name = _productsNames[Randoms.RandomInt(_productsNames.Length)],
            Price = Randoms.RandomPrice(),
            MainImageUrl = _defaultImageUrl,
            Images = imagesUri
        };
        products.Add(product);
        Console.WriteLine(i);
    }
}
```

```

2 usages
public async Task Seed(string fileName)
{
    Console.WriteLine("Seeding Products....");
    var products = GetAllProducts(fileName);
    var stopwatch = new Stopwatch();
    int counter = 0;
    stopwatch.Start();
    foreach (var product in products)
    {
        var command = new CreateProductCommand
        {
            Id = product.Id,
            CategoryId = product.CategoryId,
            CategoryName = product.CategoryName,
            Discount = product.Discount,
            Description = product.Description,
            Name = product.Name,
            Price = product.Price,
            MainImageUri = product.MainImageUri,
            Images = product.Images
        };
        await _mediator.Send(command);
        Console.WriteLine(++counter);
    }

    stopwatch.Stop();
    Console.WriteLine("Seeding Products Succeeded");
    Console.WriteLine($"Time Consumed: {stopwatch.Elapsed}");
}

```

9-3 تجهيز بيئة الاختبار:

نظراً لعدم توافر سيرفر خاص فإننا سنقوم باختبار المشروع على الحاسب الشخصي حيث تم تنصيب نسخة Sql Server Express 2019 ونسخة MongoDB 4.4.

مواصفات الحاسب هي: معالج رباعي النوى Core i5، ذاكرة RAM 8 GB و قرص تخزين SSD Sata 500 GB.

باستخدام أداة توليد البيانات قمنا بملء قاعدة البيانات بـ 250000 منتج وطلب وعميل، حيث تم استخدام البيانات المولدة في ملفات JSON في الفقرة السابقة.

10-3 اختبار سرعة التنفيذ لبعض الاستعلامات في النموذجين:

من أجل الاختبار قمنا بإنشاء استعلامات خاصة وبناء معالجات لها في طبقة الـ Infrastructure لكل نموذج، السبب في ذلك أنه في الاستعلامات الأصلية فإننا نقوم بتنفيذ الاستعلام ثم تحويله إلى كائن من الصنف المناسب، إن عملية التحويل هذ تستغرق وقتاً مختلفاً في كل نموذج ولا يجب ان يدخل هذا الوقت في الحساب.

بعد ذلك قمنا بإنشاء صفحة خاصة للاختبار في تطبيق الواجهة الأمامية، حيث يتم تحديد عدد المرات المراد تنفيذ الاستعلام بها عن طريق حقل نصي، يتم إرسال الطلب الى تطبيق الواجهة الخلفية وأجراء الاختبارات وقياس الزمن في كل مرة ثم إظهار النتائج.

Database Testing

Test Counts: 10

Get Products Category

Run MongoDB Test

Run SQL Server Test

Get Product

Run MongoDB Test

Run SQL Server Test

من أجل قياس الزمن فإننا قمنا باستخدام كائن من الصنف Stopwatch الموجود في لغة C#، حيث يوفر هذا الكائن مؤقت يعمل على التفرع مع التعليمات البرمجية الموجودة بعد لحظة بدايته، لذلك فإننا نقوم ببداية هذا المؤقت مع بداية الاستعلام ونوقفه مع نهاية الاستعلام، يبين الشكل في الأسفل مثالاً على قياس زمن استعلام باستخدام المؤقت المذكور.

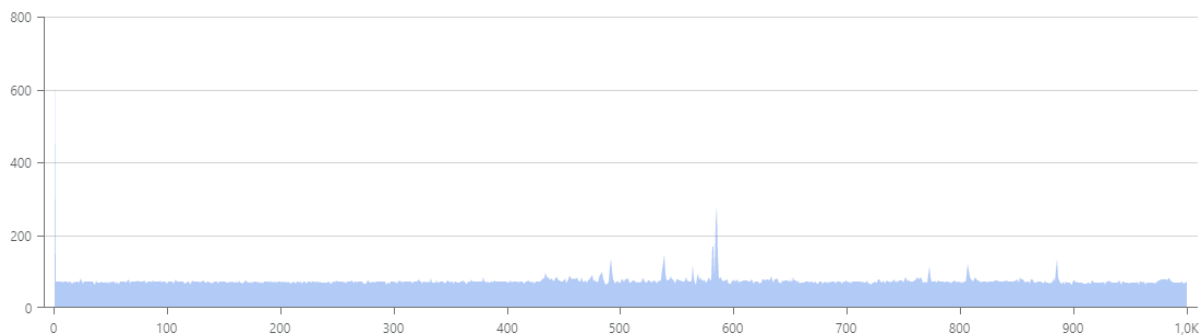
```
var collection = _database.GetCollection<BsonDocument>(Constants.ProductsCollectionName);
for (int i = 0; i < request.TestsCount; i++)
{
    var productId:Guid = products[Randoms.RandomInt(products.Count)].Id;
    var filter = new FilterDefinitionBuilder<BsonDocument>().Eq(field: "Id", productId);
    var productMongo:IFindFluent<BsonDocument,BsonDocument>? = collection.Find(filter);
    var stopWatch = new Stopwatch();
    stopWatch.Start();
    var p:AsyncCursor<BsonDocument>? = collection.Find(filter).ToCursor();
    foreach (var item:BsonDocument? in p.ToEnumerable())
    {
        //
    }
    stopWatch.Stop();
    result.Millis.Add(stopWatch.Elapsed.Milliseconds);
}
```

نلاحظ أن حلقة ال foreach المستخدمة في الكود فارغة، وهذا لأننا نريد فقط قياس وقت تنفيذ الاستعلام في قاعدة البيانات، دون تحميل النتائج الى كائنات في لغة C#.

نبين فيما يلي أبرز نتائج القراءة لبعض الاستعلامات وذلك من أجل 1000 مرة للاستعلامات التالية:

استعلام الحصول على المنتجات الخاصة بصنف ما في:

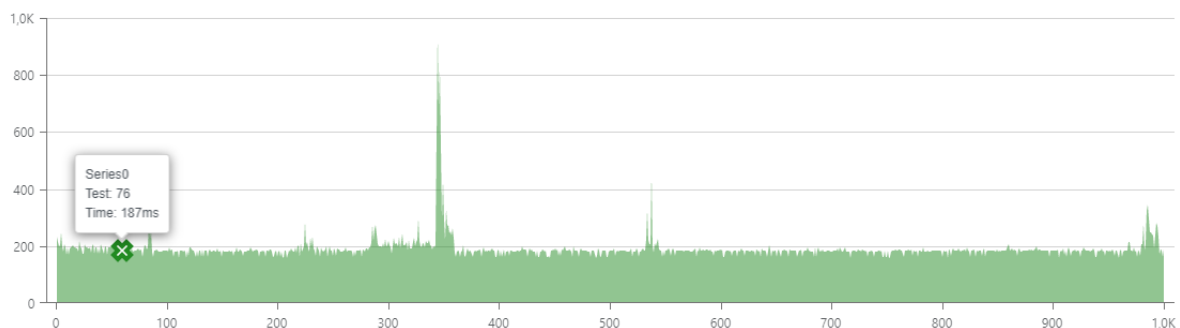
من أجل SQL Server:



Average: 73,83

من أجل MongoDB:

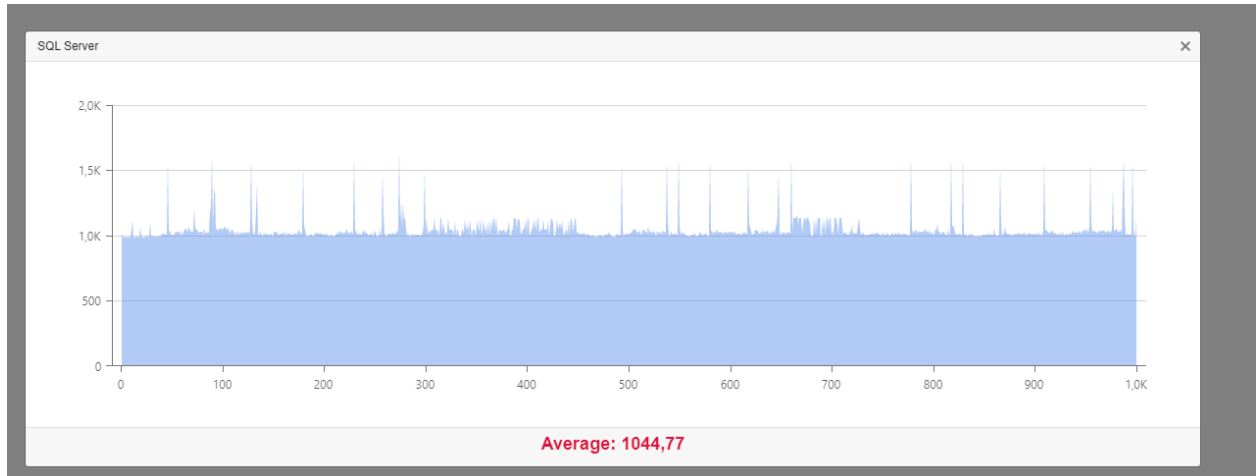
GetCategoryProductsTestingQuery MongoDB



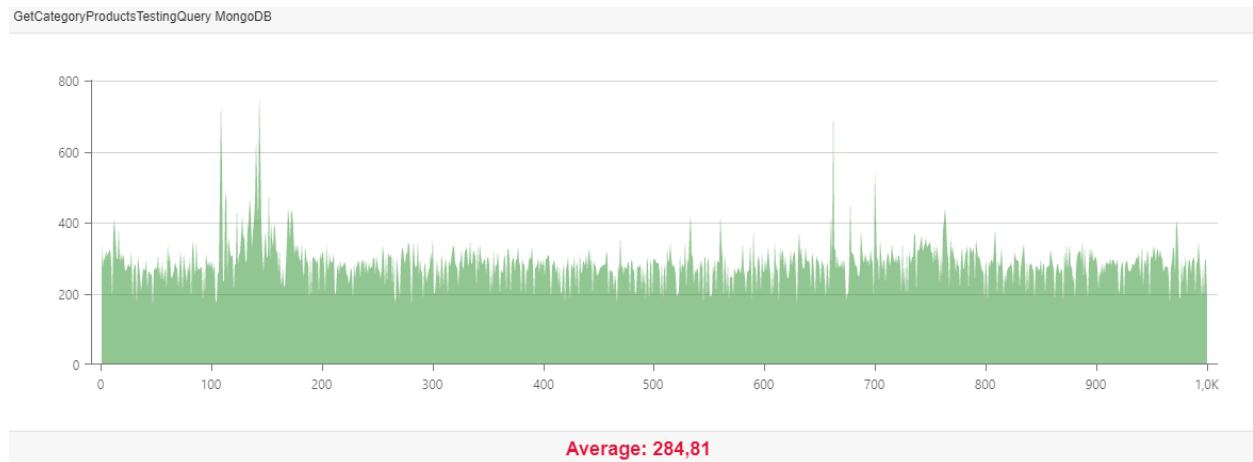
Average: 189,80

استعلام لجلب منتج ما:

من أجل Sql Server:

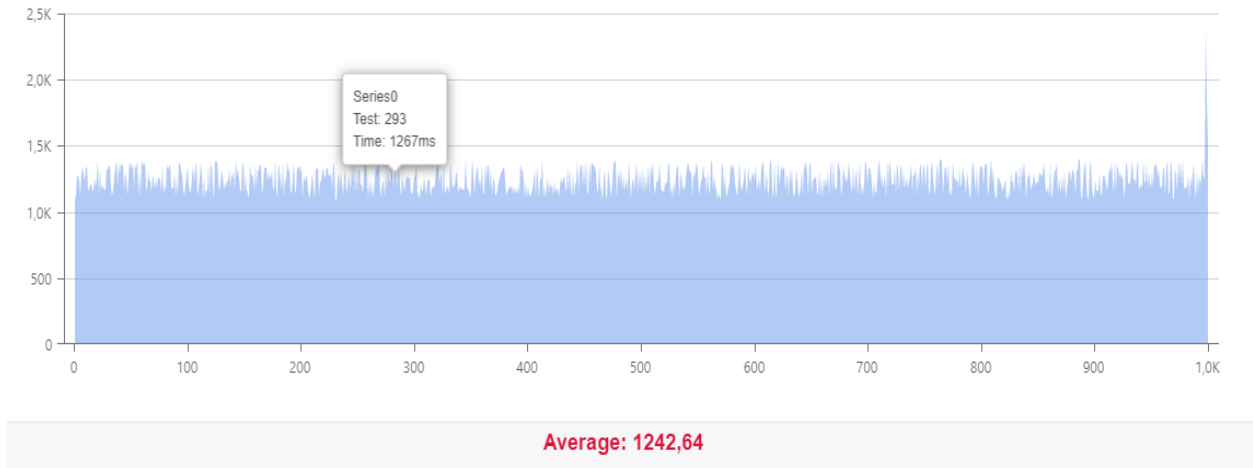


من أجل MongoDB:

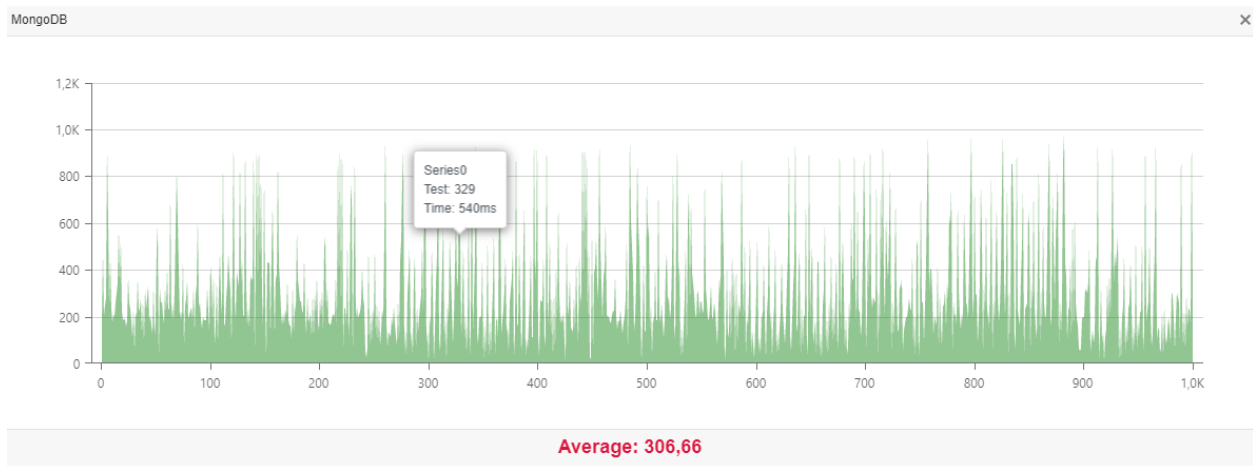


استعلام الحصول على الطلبات التي بحالة انتظار الموافقة:

من أجل Sql Server:



من أجل MongoDB:



الفصل الرابع

طرق ترميز المعلومات في تطبيقات

الويب

درسنا في الفصول السابقة استخدام قواعد البيانات في تطبيقات الويب، حيث يستقبل تطبيق السيرفر (تطبيق الواجهة الخلفية Backend) طلبات معينة لقراءة البيانات أو كتابتها ثم يقوم بتنفيذ هذه الأوامر على قاعدة البيانات، بعد القراءة من قاعدة البيانات فإن النتائج يتم نمذجتها على شكل كائنات من أصناف ال Domain الخاصة بالمسألة، والسؤال هنا كيف يتم إرسال المعلومات في هذه الكائنات عبر الشبكة الى تطبيق العميل الذي ممن الممكن أن يكون مُبرمجاً باستخدام تقنيات مختلفة عن تقنيات السيرفر والذي أيضاً ممن الممكن أن يكون عبارة تطبيق على المتصفح أو تطبيق موبايل أو خدمة ويب أخرى؟ في هذا الفصل سوف ندرس الخيارات الممكنة للإجابة على هذا السؤال.

1-4 تعريف طرق ترميز المعلومات في تطبيقات الويب:

هي عملية تحويل المعلومات الموجودة في خصائص الكائنات الى صيغة معينة بحيث يمكن نقل هذه المعلومات عبر الشبكة، وتتم هذه العملية في الواجهتين الخلفية Backend والأمامية Frontend، على سبيل المثال يقوم تطبيق الواجهة الأمامية بتحويل المعلومات الخاصة ب Form معين في التطبيق والمخزنة ضمن كائن ما الى صيغة معينة يحددها تطبيق الواجهة الخلفية، يقوم تطبيق الواجهة الخلفية باستقبال البيانات الواردة بهذه الصيغة ومن ثم تحويلها الى كائنات برمجية حسب لغة البرمجة، وتكرر العملية بشكل معاكس عند إرسال الاستجابة من الواجهة الخلفية الى الواجهة الأمامية.

2-4 الترميز وفك الترميز Serialization vs Deserialization:

تسمى عملية تحويل المعلومات من كائنات الى صيغة معينة بالترميز Serialization، أما عملية استقبال المعلومات بصيغة معينة وتحويلها الى كائنات برمجية فتسمى بفك الترميز Deserialization.

4 أشهر الصيغ القياسية المستخدمة في التصميم -3:

:Extensible Markup Language (XML)

XML ترمز إلي Extensible Markup Language ، و قد تم تطويرها من قبل W3C (World Wide Web Consortium) في التسعينات.

وبالرغم من أن XML تشبه HTML حيث أنهما يمكن قراءتهما بواسطة الإنسان، إلا أن الغرض منهما مختلف كل الاختلاف. HTML تستعمل لتوصيف هيكل صفحات الويب و محتوياتها بينما تستعمل XML لتوصيف تركيب البيانات.

تمد XML البرامج والمبرمجين بشكل أخص بصيغة موحدة متفق عليها لنقل البيانات عبر الأنظمة المختلفة، و هي من تلك الناحية تمتلك قواسم مشتركة مع JSON أكثر مما لديها مع HTML.

و مع أنها ليست الطريقة المفضلة لتنظيم و نقل البيانات في عصرنا هذا، إلا أنها لا زالت تحتفظ بمكانتها. فهي ما زالت مستخدمة في العديد من الأنظمة القديمة، و كلا من RSS و SVG مبنيين علي صيغة XML.

نعرض في هذا المثال البسيط لعرض ملف XML و كيف يستخدم لهيكلة البيانات:(Data Structuring)

```
<?xml version="1.0" encoding="UTF-8"?>
<fcc_merch>
  <item>
    <name>Triblend T-shirt</name>
    <price>$24.99</price>
    <description>Represent the freeCodeCamp community with pride in this jet-black Triblend T-shirt featuring the iconic "f" logo.</description>
  </item>
  <item>
    <name>Cotton-Poly Pullover Hoodie</name>
    <price>$49.99</price>
    <description>Stay toasty and dress like a developer with this jet-black cotton-poly pullover hoodie.</description>
  </item>
  <item>
    <name>Ceramic Coffee Mug</name>
    <price>$14.99</price>
    <description>Toast to the developer community with your very own freeCodeCamp Bonfire Function Call mug.</description>
  </item>
</fcc_merch>
```

:JavaScript Object Notation Jason

هي صيغة بسيطة وقابلة للقراءة بسهولة من قبل الإنسان وتستخدم لتمثيل البيانات و تبادلها بين الأنظمة البرمجية المختلفة.

ليست لغة برمجية إنما هي طريقة متفق عليها بين لغات البرمجة المختلفة لتمثيل البيانات بهدف سهولة تبادل البيانات بين هذه اللغات.

صيغة جيسون تمثل عن طريق نص، والبنية لهذا النص تشبه الكائن أو Object في لغة البرمجة جافا سكربت، وهذه الصيغة مدعومة من لغات البرمجة الأساسية الأخرى وتستخدم هذه الصيغة بشكل كبير لتبادل البيانات بين الخادم والعميل Client-Server.

كيفية تمثيل البيانات عن طريق

الجزءان الأساسيان اللذان يشكلان JSON هما المفاتيح Keys والقيم Values.

المفتاح : Key يمثل اسم فريد لقيمة البيانات ويتم وضعه عادة بين علامات التنصيص

القيمة : Value تمثل البيانات ويمكن أن تمثل أكثر من نوع بيانات مثل النصوص والأرقام والمصفوفات.

معاً يشكل Key / Value سطر في صيغة جيسون حيث يتم استخدام علامة , كفاصل بين السطور.

لماذا نستخدم جيسون JSON

لفهم فائدة وأهمية JSON ، يجب أن نفهم قليلاً كيف تطور عرض صفحات الويب.

في أوائل عام 2000، بدأ التفاعل بين العميل والخادم في التحول بشكل كبير، حيث كان المتصفح يعمل بشكل أساسي لعرض المعلومات بشكل بسيط، وكان الخادم يقوم بكل العمل الشاق لإعداد المحتوى للعرض بالطريقة المطلوبة. فعندما ينقر المستخدم على رابط أو زر في المتصفح، سيتم إرسال طلب إلى الخادم ، حيث يقوم الخادم بإعداد المعلومات المطلوبة على هيئة HTML ، وسيقوم المتصفح بعرض HTML كصفحة جديدة. كان هذا النمط بطيئاً وغير فعال، حيث يتطلب من المتصفح إعادة تحميل وإعداد كل شيء على الصفحة حتى لو لم يتم تغيير سوى جزء بسيط من الصفحة.

كان إعادة تحميل الصفحة بالكامل مضيعة للوقت ولموارد الخادم، بحث مطورو الويب عن تقنيات أحدث لتحسين تجربة المستخدم بشكل عام. فأثبتت القدرة على إرسال طلبات الويب إلى الخادم في الخلفية أثناء عرض الصفحة، والتي تم تقديمها في Internet Explorer 5 ، وترجع فائدة تقديم الطلبات في الخلفية إلى عدم إعادة تحميل الصفحة كاملة لتغيير جزء معين في الصفحة. على سبيل المثال عند الضغط على رابط معين يتم تحميل بيانات في الخلفية دون إعادة تحميل الصفحة وبالتالي يتم التعامل مع البيانات على المتصفح من قبل لغة Javascript التي تعمل على المتصفح.

في الأساس كان يتم نقل البيانات بتنسيق XML ولكن XML كانت طويلة وصعبة في التعامل في لغة جافا سكريبت. كانت JavaScript تحتوي بالفعل على مفهوم الكائنات أو Objects، وهي طريقة للتعبير عن البيانات داخل اللغة، لذلك أخذ Douglas Crockford مجموعة فرعية من هذا التعبير كمواصفات لتنسيق تبادل البيانات الجديد الذي أطلق عليه اسم JSON. كان JSON أسهل في القراءة بكثير بالنسبة للأشخاص وحتى بالنسبة لتحليل المتصفحات. وسرعان ما بدأ مطورو الويب يفضلون JSON عن XML.

:Protocol Buffer

هذه الصيغة هي جديدة مقارنة مع JSON أو XML وقد تم تطويرها من قبل شركة غوغل، تقوم هذه البيانات بترميز البيانات بالصيغة الثنائية الأمر الذي يجعلها فعالة جداً من حيث حجم الرسالة المنقولة عبر الشبكة وبالتالي تسريع النقل.

توفر غوغل عدة مكتبات لعدة لغات من أجل ترميز وفك ترميز المعلومات بهذه الصيغة. لهذه الصيغة سيّتان، الأولى هو أن البيانات المرمزة غير مقروءة بالنسبة للإنسان ويجب حصر استخدام المكتبات التي توفرها شركة غوغل لترميز / فك ترميز البيانات، وفي حال عدم توفر مكتبة للغة البرمجة المستخدمة فإن استخدام هذه الصيغة غير ممكن، الثانية هي أن بنية البيانات يجب أن تكون معروفة بالنسبة لكل من المرسل والمستقبل، الأمر الذي يجعل إرسال رسائل ديناميكية غير ممكناً.

الفصل الخامس

بروتوكولات الاتصال

Communication Protocols

1- تعريف:

يعتبر بروتوكول الاتصال هو صلة الوصل الرئيسية بين السيرفر والعميل، حيث تعرف البروتوكولات مجموعة من الشروط والقواعد التي تحدد شكل الاتصال بين جهازين أو أكثر، ولكل بروتوكول أشكال مختلفة للاستخدام، ليحقق الاتصال الصحيح بين الأجهزة.

في علم الشبكات تتكون الشبكة من 7 طبقات:

1. الطبقة الأولى (Physical Layer)
2. الطبقة الثانية (Data Link)
3. الطبقة الثالثة (Network Layer)
4. الطبقة الرابعة (Transmission Layer)
5. الطبقة الخامسة (Session Layer)
6. الطبقة السادسة (Presentation Layer)
7. الطبقة السابعة (Application Layer)

يوجد لكل طبقة البروتوكولات والقواعد الخاصة بها مثل بروتوكول الانترنت IP وبروتوكولات TCP وUDP، هذه البروتوكولات تعتبر خارج نطاق بحثنا كونها مرتبطة أكثر بعلم الشبكات، ولكن ما يهمنا في هذا البحث هو البروتوكولات الموجودة في طبقة التطبيقات حيث تعرف هذه التطبيقات شكل التواصل بين السيرفر والعميل.

سوف ندرس في هذا البحث بروتوكولي HTTP و Web Socket حيث أنهما من أكثر البروتوكولات المستخدمة في تطبيقات الويب.

2-5 بروتوكول HTTP:

HTTP هو اختصار لـ Hypertext Transfer Protocol وتعني بروتوكول نقل النص التشعبي أو بروتوكول نقل النص الفائق وهو مجموعة من القواعد لنقل الملفات ، مثل النصوص والصور الرسومية والصوت والفيديو وملفات الوسائط المتعددة الأخرى على شبكة الويب العالمية.

بمجرد أن يفتح مستخدم ويب متصفح الويب الخاص به، يستخدم المستخدم بشكل غير مباشر بروتوكول HTTP.

عندما يتم إرسال هذه الطلبات والاستجابات ، فإنها تستخدم TCP / IP لتقليل ونقل المعلومات في حزم صغيرة من تسلسلات ثنائية من الأصفار والواحد التي يتم إرسالها فعليًا عبر الأسلاك الكهربائية وكابلات الألياف البصرية والشبكات اللاسلكية.

3-5 بروتوكول WebSocket:

يستخدم لإنشاء اتصال بين عميل و خادم (متصفح وخادم) وتمكين الاتصال بينهما في الوقت الفعلي. يتمثل الاختلاف الرئيسي مع WebSocket في أنه يسمح لك بتلقي البيانات دون الحاجة إلى إرسال طلب منفصل، كما يحدث، على سبيل المثال، في HTTP. بعد إنشاء الاتصال، ستأتي البيانات من تلقاء نفسها دون الحاجة إلى إرسال الطلب. إنها ميزة استخدام بروتوكول WebSocket في الردود أو تقارير المخزون، حيث تحتاج إلى تلقي معلومات محدثة باستمرار. يمكن للبروتوكول تلقي المعلومات وإرسالها في وقت واحد، مما يسمح بالاتصال ثنائي الاتجاه ثنائي الاتجاه، مما يؤدي إلى تبادل المعلومات بشكل أسرع.

كيف يعمل WebSocket؟

يظل الاتصال بين العميل والخادم مفتوحًا حتى يتم إنهاؤه بواسطة أحد الأطراف أو يتم إغلاقه بسبب انتهاء المهلة. يقومون بمصافحة لإنشاء اتصال بين العميل والخادم. يظل الاتصال الذي تم إنشاؤه مفتوحًا، ويتم إجراء الاتصال باستخدام نفس القناة حتى يتم إنهاء الاتصال من جانب العميل أو الخادم. يتم تبادل الرسائل ثنائية الاتجاه. يسمح لك WebSocket بتشفير البيانات المنقولة. لهذا الغرض، يتم استخدام وظيفة إضافية عبر بروتوكول WSS، والذي يقوم بترميز البيانات على جانب المرسل ويفك تشفيرها من جانب المستلم. بالنسبة لأي وسطاء، تظل المعلومات مشفرة. بدون تشفير، تصبح البيانات هدفًا للتهديدات.

4-5 الفرق بين WebSocket وHTTP:

إن الفرق الأساسي بينهما هو أنه في البروتوكول HTTP يكون شكل تدفق البيانات هو Half-Duplex بحيث يكون إرسال البيانات باتجاه وحيد في لحظة ما، ولذلك هو بروتوكول عديم الحالة Stateless ويجب أن يقوم العميل بإرسال طلب حتى يرسل الخادم استجابة، ولا يمكن للخادم أن يقوم بإرسال رسائل للعميل مباشرة بدون طلب.

أما في بروتوكول WebSocket يكون تدفق البيانات بالشكل Full-Duplex حيث يمكن إرسال واستقبال البيانات في نفس الوقت، وبمجرد أن تحصل مصافحة بين الخادم وعميل ما، يمكن للخادم إرسال رسائل إلى هذا العميل في أي وقت حتى نهاية الاتصال بينهما.


نستنتج من هذا بأنه لا يمكن استخدام بروتوكول HTTP في التطبيقات التي تحتاج إلى تحديث تلقائي للبيانات لدى العميل

تجربة عملية:

نريد بناء غرفة محادثة على الويب، يقوم المستخدم أولاً بإدخال اسمه ثم بعد ذلك يدخل إلى غرفة المحادثة ويستطيع إرسال الرسائل وقراءة الرسائل الواردة.

نلاحظ أن التطبيق المطلوب يجب أن يتم بناءه باستخدام WebSocket حتى يتمكن الخادم من إيصال الرسائل للمتصفحات عند المستخدمين الموجودين في الغرفة.

Enter your name to start chatting:

 You are connected as Malek

[Notice] Malek joined chat room.

[Notice] ayman joined chat room.

ayman
Hi


Malek
Hi There

enter your comment



Send

Blazor SignalR Chat Sample

 You are connected as ayman

[Notice] ayman joined chat room.

ayman
Hi

Malek
Hi There

enter your comment



Send