ChatGPT

# اليوم 1: مقدمة في تصميم الأنظمة الإنتاجية

## شرح مختصر

تصميم الأنظمة على مستوى بيئة الإنتاج يختلف جذريًا عن بناء نموذج أولي تجريبي. فعند تطوير **نموذج أولي (Prototype)** يكون التركيز على السرعة وتقليل التكلفة بهدف اختبار فكرة أو ميزة بسرعة، حتى لو كان ذلك يعني التضحية ببعض جودة التصميم أو تراكم **الدَين التقني** (أي حلول سريعة مؤقتة ستحتاج لإعادة تطوير لاحقًا) [1]. أما **النظام الإنتاجي (Production System)** فهو المنتج النهائي الموجه للمستخدمين الفعليين، ويتطلب درجة أعلى بكثير من الموثوقية والجودة. في مرحلة الإنتاج، تتباطأ دورة التطوير وتصبح أكثر انضباطًا من مرحلة النموذج الأولي، لأن الهدف هو تقديم تجربة مستخدم مستقرة وخالية من الأخطاء [2]. على سبيل المثال، من الشائع في بيئة الإنتاج اعتماد مراجعات شفرات صارمة واختبارات جودة شاملة (وحدات، تكامل، ضمان جودة) قبل إصدار أي ميزة [3] [4]، في حين قد يتم تجاوز بعض هذه الضوابط أثناء بناء نموذج أولي بهدف التسريع.

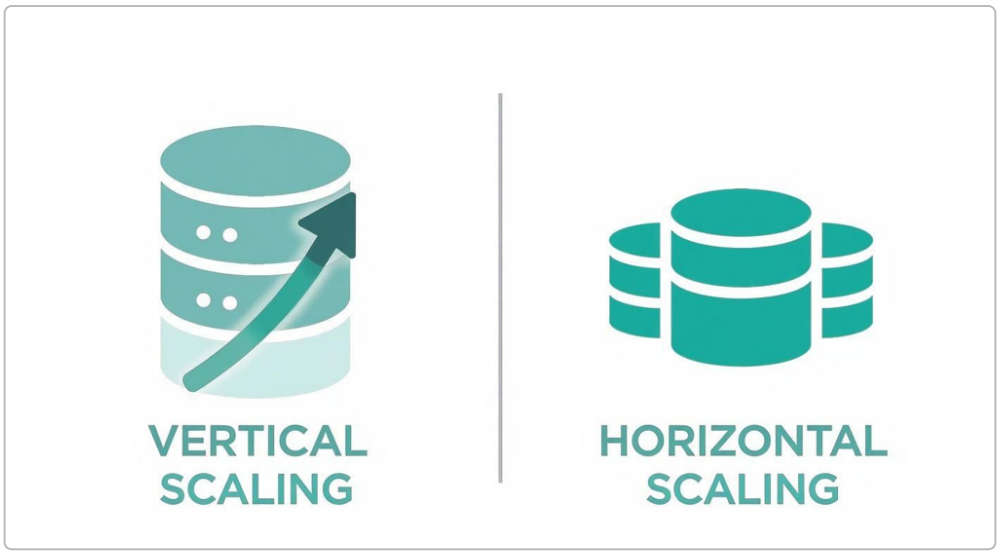
من الفروق الرئيسية أيضًا أن النموذج الأولي قد يعمل بشكل جيد في بيئة تجريبية محدودة، لكن النظام الإنتاجي يجب أن يصمم ليتحمل أحمالًا كبيرة وظروفًا غير متوقعة. فالنموذج الأولي قد يُجرَّب لعدة أشهر بدون مشاكل، ولكن **ماذا سيحدث عند إطلاق النظام لمليون مستخدم دفعة واحدة؟** لا بد من التخطيط منذ التصميم لكيفية الحفاظ على الأداء والاستقرار تحت ضغط هائل من المستخدمين [5]. لذلك يركز تصميم أنظمة الإنتاج على تلبية المتطلبات غير الوظيفية **(Non-Functional Requirements)** مثل **قابلية التوسع، والموثوقية، وقابلية الصيانة** – إلى جانب متطلبات أخرى كالأمان والأداء – بقدر اهتمامه بالمتطلبات الوظيفية الأساسية. المتطلبات غير الوظيفية تعنى بخصائص جودة النظام وقيوده التشغيلية (مثل الأداء والحمولة والأمان)، وهي غالبًا ما تكون عامل النجاح في بيئة الإنتاج [6] [7].

الاهتمام بهذه الجوانب يضمن أن النظام سيبقى **قابلاً للتوسع** عند زيادة عدد المستخدمين أو حجم البيانات، وسيظل **موثوقًا** يعمل باستمرار دون أعطال جسيمة، ويسهل **صيانته** وتطويره مستقبلًا من قبل الفرق التقنية. كذلك ينبغي أن يكون النظام آمنًا ضد الثغرات والهجمات، خاصة في بيئة الإنتاج الحقيقية حيث السمعة على المحك. على سبيل المثال، ينبغي تعطيل وضع التصحيح التفصيلي (Debug Mode) في الإصدار الإنتاجي لتفادي كشف معلومات حساسة للمستخدمين [8]. أيضًا يتطلب الحفاظ على الاستقرار في الإنتاج وجود آليات مراقبة وتنبيه (Monitoring) للتصرف سريعًا عند حدوث أي خلل أو انخفاض في الأداء. كل هذه الاعتبارات تجعل تصميم النظام الإنتاجي عملية أكثر تعقيدًا لكنها ضرورية لضمان نظام ناجح في الواقع العملي.

أخيرًا، من التقنيات المهمة للتحكم في تعقيد الأنظمة الكبيرة **تقسيم النظام إلى مكونات أو خدمات أصغر ذات مسؤوليات واضحة**. هذا التقسيم يقلل من التشابك بين الأجزاء ويسهل تطوير كل جزء وفهمه بشكل مستقل. فعلى سبيل المثال، يمكن تقسيم تطبيق كبير إلى خدمات منفصلة (مثل خدمة للطلبات وخدمة للمخزون وخدمة للمستخدمين) تتواصل فيما بينها عبر واجهات محددة. هذا النهج المعروف بـ **هندسة الخدمات المصغّرة (Microservices)** يساعد على توزيع التطوير وتحسين قابلية التوسع، بحيث يمكن **تطوير كل خدمة ونشرها وتوسيعها بمعزل عن الأخرى** [9]. حتى في التطبيقات الأحادية (Monolith)، يمكن تحقيق شيء من هذا التقسيم عبر فصل الطبقات (واجهات المستخدم، منطق الأعمال، قاعدة البيانات) ووضع حدود واضحة بين المسؤوليات لتحقيق تصميم أكثر تنظيمًا وقابلية للصيانة.

## المصطلحات الأساسية

- **Scalability (قابلية التوسع):** قدرة النظام على التوسع ومعالجة المزيد من الحمل (عدد مستخدمين أو بيانات أكبر) دون تدهور في الأداء. نظام قابل للتوسع يمكنه التعامل مع تضاعف أعداد المستخدمين أو العمليات عبر تحسين الموارد المتاحة أو إضافة موارد جديدة [10] [11] . تشمل استراتيجيات التوسع زيادة قدرة العتاد للخادم الواحد **(Vertical Scaling)** أو إضافة خوادم متعددة وتقسيم الحمل بينها **(Horizontal Scaling)** [12] [13] . النظام المصمم جيدًا يجب أن يستمر بالأداء بكفاءة حتى مع نمو قاعدة مستخدميه عشرة أضعاف أو مئة ضعف.
- **Reliability (الموثوقية):** مدى قدرة النظام على أداء وظيفته باستمرار وتحمل الأخطاء دون انهيار. نظام موثوق يعني وقت توقف قليل جدًا وقدرة على تحمل أعطال العتاد أو أخطاء البرمجة أو أخطاء الاستخدام البشري دون تعطّل الخدمة [14] [15] . لتحقيق الموثوقية يُصمّم النظام مع وجود نسخ احتياطية وتكرار (Redundancy) للأجزاء الهامة، ومعالجة للأخطاء والاستثناءات بحيث لا تؤدي خطأ واحدة لإيقاف النظام بالكامل [16] [17] . الموثوقية العالية تعني أيضًا **الاتساق** في سلوك النظام وأن يحصل المستخدم على نفس الأداء المتوقع في كل مرة يستخدم فيها النظام.
- **Maintainability (قابلية الصيانة):** سهولة فهم النظام وتعديله وتحسينه مع مرور الوقت. النظام القابل للصيانة يتميز بكود منظم وواضح، وبتوثيق جيد، وبهيكلية تسمح بإضافة ميزات جديدة أو إصلاح الأخطاء دون تعقيدات كبيرة [18] [19] . من مبادئ الصيانة الجيدة أن يكون النظام **بسيطًا** قليل التعقيد، وذو **فصل واضح في المسؤوليات** بين المكونات، وأن يكون **قابلاً للتوسع والتغيير** دون الحاجة لإعادة كتابة أجزاء كبيرة منه [20] [21] . قابلية الصيانة مهمة لأن معظم تكلفة البرمجيات تأتي من صيانتها وتطويرها المستمر وليس من بنائها الأولي.
- **Production Environment (بيئة الإنتاج):** البيئة الحقيقية التي يعمل فيها النظام ويستخدمها المستخدمون النهائيون. في هذه البيئة يكون النظام متاحًا للعامة أو لشريحة واسعة من العملاء، بخلاف بيئات التطوير أو الاختبار. بيئة الإنتاج تتميز بأنها تحتاج لإعدادات خاصة بالأمان والاستقرار (مثل استخدام خوادم قوية، إعداد نسخ احتياطية، مراقبة مستمرة، تعطيل خيارات التصحيح) لضمان عمل النظام 24/7 دون انقطاع. أي تغيير على النظام في هذه البيئة يجب أن يتم بحذر وبعد اختبارات مكثفة نظرًا لتأثيره المباشر على المستخدمين والشركة.
- **Non-Functional Requirements (المتطلبات غير الوظيفية):** هي المتطلبات التي تحدد معايير جودة النظام وقيوده التشغيلية بدلاً من وظائفه المباشرة. فهي تشمل جوانب مثل الأداء وقابلية التوسع والموثوقية والأمان وقابلية الاستخدام وغيرها. المتطلبات غير الوظيفية **تركّز على تجربة المستخدم والجودة العامة للنظام** أكثر من تركيزها على وظيفة محددة [6] . على الرغم من أن إهمالها قد لا يمنع النظام من "العمل" ظاهريًا، إلا أنها حاسمة لجعل النظام ناجحًا وقابلاً للاستخدام عمليًا [22] . على سبيل المثال، قد يكون النظام قادرًا على تنفيذ عمليات البيع والشراء (وظيفة أساسية)، لكنه إن كان بطيئًا جدًا تحت الضغط أو غير آمن ضد الاختراقات فإن ذلك يعني فشلًا في المتطلبات غير الوظيفية. لذا يجب تحديد هذه المتطلبات بوضوح (مثل: الزمن الأقصى لاستجابة النظام، معدل تحمل الأخطاء، معايير الأمان...) وتصميم النظام وهندسته بحيث يفي بها.




شكل توضيحي – يبيّن الفرق بين التوسع الرأسي *(يسارًا، تعزيز قدرة خادم واحد)* والتوسع الأفقي *(يمينًا، إضافة خوادم متعددة)* ضمن قابلية التوسع في النظام [23] [24] . التوسع الرأسي سهل التنفيذ لكنه محدود بقدرات الخادم الواحد، أما التوسع الأفقي فيوزع الحمل على عدة خوادم لتحقيق موثوقية أعلى وقدرة شبه غير محدودة على النمو [25] [26] .

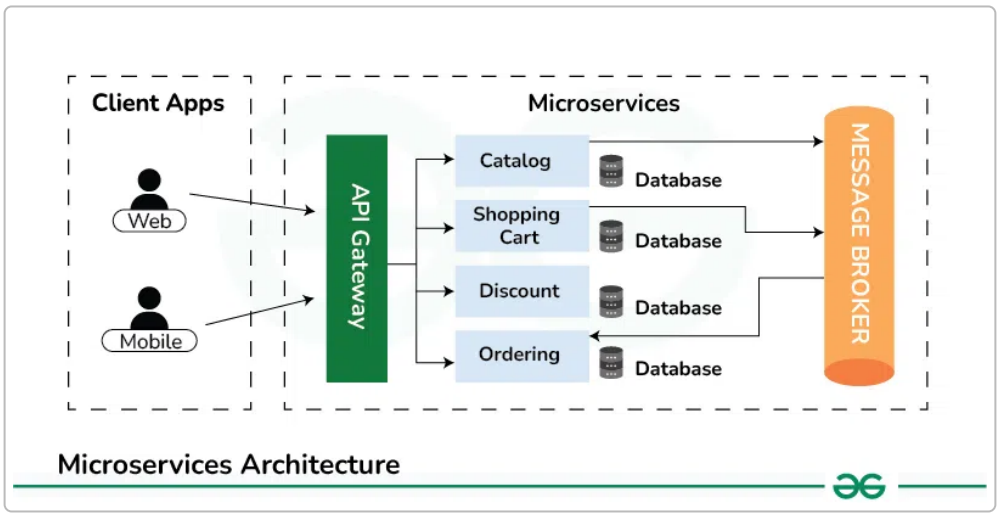
## فكرة التطبيق العملي

لتوضيح ما سبق بصورة عملية، تخيّل أنك مسؤول عن تصميم نظام **موقع تجارة إلكترونية كبير** مشابه لمنصات البيع الشهيرة. حاول وضع قائمة بالمتطلبات الوظيفية وغير الوظيفية لهذا النظام، ثم فكّر في الاعتبارات التصميمية لضمان تحقيق هذه المتطلبات في بيئة الإنتاج الحقيقية. على سبيل المثال:

- **المتطلبات الوظيفية (Functional Requirements):** تشمل ميزات أساسية يتوقعها المستخدمون، مثل: إمكانية تصفح المنتجات وإضافتها إلى سلة المشتريات، تسجيل حسابات المستخدمين وتسجيل الدخول، معالجة طلبات الشراء والدفع الإلكتروني بأمان، نظام تقييم ومراجعة للمنتجات، وإدارة المخزون من قبل المسؤولين. يجب تحديد كل وظيفة يحتاجها النظام لخدمة أعمال التجارة الإلكترونية (بحث المنتجات، فلترة النتائج، تنبيهات للمستخدمين، إلخ) بالتفصيل.

- **المتطلبات غير الوظيفية (Non-Functional Requirements):** تحدد معايير جودة الأداء والتشغيل للنظام. مثلًا: قدرة الموقع على خدمة **عدد ضخم من المستخدمين المتزامنين** (على الأقل عشرات الآلاف في نفس اللحظة) دون انخفاض ملحوظ في سرعة الاستجابة (زمن استجابة الصفحة أقل من 2 ثانية تحت الحمل العادي). أيضًا **موثوقية عالية** بحيث يكون زمن التوقف (Downtime) شبه معدوم مع توفر بنسبة 99.9% على مدار العام. يجب أن يكون النظام **آمنًا للغاية** لحماية بيانات المستخدمين ومعاملات الدفع (تشفير البيانات الحساسة، الامتثال للمعايير الأمنية مثل PCI DSS). كذلك **قابلية التوسع** مهمة بحيث يمكن إضافة موارد خادم إضافية أو نسخ جديدة من الخدمات مع نمو عدد المستخدمين دون الحاجة لإعادة بناء النظام. بالإضافة إلى ذلك، ينبغي مراعاة **قابلية الصيانة والتطوير** بحيث يستطيع فريق التطوير إضافة ميزات جديدة أو تعديل القواعد التجارية بسرعة نسبية ودون أخطاء، وهذا يستدعي اعتماد بنية برمجية نظيفة وموثقة (مثال: فصل الواجهات الأمامية عن خدمات البيانات عبر واجهات API). وأخيرًا لا ننسى **متطلبات الأداء** مثل زمن معالجة عمليات البحث عن المنتجات أو إنشاء الطلبات حتى في أوقات الذروة، و**متطلبات التوافقية** لضمان عمل الموقع عبر مختلف المتصفحات والأجهزة (حاسوب، جهاز لوحي، هاتف).

عند تصميم الحل المعماري لهذا الموقع التخيلي، فكّر كيف يمكن تحقيق هذه المتطلبات: ربما باعتماد خادم or أكثر لواجهة المستخدم مع موزع حمل (Load Balancer) لتحقيق التوسع الأفقي، واستخدام قاعدة بيانات مُوزَّعة أو مخزونات بيانات متعددة لتحمل عدد العمليات الكبير. قد تحتاج أيضًا لاستخدام أنظمة **الكاش (Cache)** لتحسين سرعة تحميل الصفحات وتقليل الضغط على قواعد البيانات [27]. بالنسبة للموثوقية، يمكن تصميم النظام بحيث تكون هناك خوادم احتياطية (Failover) أو نسخ مكررة من قواعد البيانات بحيث لا توجد نقطة فشل واحدة. ومن ناحية الأمان، يُنصح باستخدام بروتوكولات تشفير HTTPS في كل جزء من النظام، وتخزين كلمات المرور بشكل مشفر، وإجراء اختبارات اختراق دورية. جميع هذه الحلول التصميمية تضمن أن النظام لن يعمل فقط وظيفيًا، بل سيعمل بكفاءة واعتمادية في ظروف الإنتاج الحقيقية.




شكل – مثال لبنية نظام تجارة إلكترونية مقسّم إلى خدمات مصغّرة *(Microservices)* ذات مسؤوليات مستقلة [9] [28]. يوضّح الرسم وجود بوابة واجهة برمجية *(API Gateway)* تستقبل طلبات العملاء وتوجّهها إلى الخدمة المعنية *(*خدمة المنتجات، خدمة الطلبات، خدمة المستخدمين، إلخ*)*. كل خدمة مصغّرة لها قاعدة بيانات خاصة بها ويتم التنسيق بينها عبر رسائل أو واجهات *API*. هذا التقسيم يمكّن كل خدمة من التوسع أفقيًا بشكل مستقل حسب الحاجة، ويعزّز موثوقية النظام عبر عزل الأعطال في نطاق الخدمة المتأثرة فقط.

## ملاحظات

هذه المساحة مخصصة لملاحظاتك الشخصية وتقييمك الذاتي. يمكنك استخدامها لكتابة أي أسئلة لديك حول المحتوى، أو نقاط ترغب في تذكرها، وكذلك لتدوين مدى فهمك لكل مفهوم مما تم طرحه. على سبيل المثال، حاول تلخيص الفروق بين النموذج الأولي والنظام الإنتاجي بكلماتك الخاصة، أو قيّم نفسك في مدى معرفتك للمتطلبات غير الوظيفية المذكورة. إن تدوين الملاحظات والتأمل في ما تعلمته سيساعدك على ترسيخ المعلومات واستكشاف أي جوانب تحتاج لمزيد من المراجعة.

---


[1] [2] [3] [4] [5] Prototyping vs. Production Development: How to Avoid Creating a Monster
https://www.olioapps.com/blog/prototype-vs-production

[6] [7] [22] الفرق بين المتطلبات الوظيفية وغير الوظيفية وتقنيات استنباطهما وأفضل الممارسات - بكه للتعليم
https://bakkah.com/ar/knowledge-center/%D8%A7%D9%84%D9%85%D8%AA%D8%B7%D9%84%D8%A8%D8%A7%D8%AA-%D8%A7%D9%84%D9%88%D8%B8%D9%8A%D9%81%D9%8A%D8%A9-%D9%88%D8%BA%D9%8A%D8%B1%D8%A7%D9%84%D9%88%D8%B8%D9%8A%D9%81%D9%8A%D8%A9%

Deployment - Laravel 12.x - The PHP Framework For Web Artisans [8]
https://laravel.com/docs/12.x/deployment

Monolithic vs. Microservices Architecture - GeeksforGeeks [28] [9]
/https://www.geeksforgeeks.org/software-engineering/monolithic-vs-microservices-architecture

What Do Reliability, Scalability, and Maintainability Mean? - DEV Community [21] [20] [19] [18] [15] [14] [13] [12] [11] [10]
https://dev.to/iggredible/what-do-reliability-scalability-and-maintainability-mean-pjl

Reliability vs. Scalability - GeeksforGeeks [17] [16]
/https://www.geeksforgeeks.org/system-design/reliability-vs-scalability

vertical scaling vs horizontal scaling: A Practical Guide [26] [25] [24] [23]
/https://www.cloudtoggle.com/blog-en/vertical-scaling-vs-horizontal-scaling

Laravel Caching Strategies: From Basics to Advanced Redis Usage [27]
https://medium.com/@ilyaskazi/laravel-caching-strategies-from-basics-to-advanced-redis-usage-c3522b88a2f2