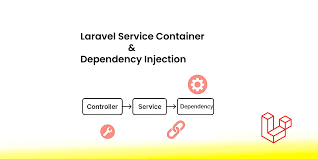
Service Container and Dependency Injection



Dependency Injection

هو نمط تصميم (Design Pattern) يهدف إلى تحقيق مبدأ Inversion of Control (IoC)، حيث يتم تمرير التبعيات (Dependencies) إلى الكائنات بدلاً من إنشائها داخليًا.

فوائد Dependency Injection:

- تقليل الاقتران (Loose Coupling): يصبح الكود أقل اعتمادًا على تفاصيل تنفيذ التبعيات.

- تحسين إمكانية الاختبار (Testability): يمكن استبدال التبعيات بـ Mock Objects أثناء الاختبار.

- زيادة المرونة: يمكن تغيير سلوك الكائن بسهولة عن طريق تغيير التبعيات المُدخلة.

أنواع Dependency Injection:

:Constructor Injection .1يتم تمرير التبعيات عبر المُنشئ (Constructor).



:Setter Injection. 2يتم تمرير التبعيات عبر دوال setter.



: Interface Injection.3يتم تحديد واجهة (Interface) لحقن التبعيات.

---

Service Container

(DI Container) هو أداة تُدار فيها التبعيات تلقائيًا، حيث تقوم بإنشاء الكائنات وحقن التبعيات المطلوبة دون الحاجة إلى التهيئة اليدوية.

مميزات Service Container:

-

الإدارة التلقائية للتبعيات: لا يحتاج المطور إلى إنشاء الكائنات يدويًا.

- تحسين الأداء: يتم إنشاء الكائنات مرة واحدة وإعادة استخدامها (Singleton Pattern).

- سهولة التهيئة: يمكن تعريف التبعيات في مكان مركزي مثل ملف (config).

. الفرق بين Service Container و Dependency Injection

Dependency Injection Service Container

نمط تصميم (Design Pattern) أداة تنفيذية (Implementation Tool)

يحتاج إلى تهيئة يدوية في بعض الأحيان يُدار تلقائيًا

يعمل في أي بيئة برمجية مرتبط بأُطر عمل تدعمه مثل (Laravel)

. 4أمثلة عملية

مثال 1: استخدام DI بدون Container



مثال 2: استخدام Service Container في Laravel



. 5الخاتمة

يُعد Dependency Injection و Service Container من الأدوات القوية التي تُحسن بنية البرمجيات وتجعلها أكثر قابلية للصيانة والتطوير. بينما يُعتبر DI مبدأً عامًا يمكن تطبيقه يدويًا، فإن Service Container يقدم حلاً مُنظمًا لإدارة التبعيات تلقائيًا، خاصة في أُطر العمل الحديثة.

باستخدام هذه المفاهيم، يصبح الكود أكثر نظافة، مرونة، وقابلية للاختبار، مما يجعله اختيارًا مثاليًا للمشاريع الكبيرة والمعقدة.