◆ What, When, and Why — API Versioning

What:

API Versioningيعني إنك تحط رقم إصدار (v1, v2, ...) للـ API بتاعك.
زي مثلًا:

/api/v1/courses

/api/v2/courses

When:

لما تبدأ تغيّر في الـ — API تضيف حقول جديدة أو تغيّر الـ — Response Structure لازم تحتفظ بالإصدار القديم علشان العملاء (clients)اللي لسه بيستخدموه مايتأثروش.

Why:

علشان تقدر تطور الـ API بدون ما تكسر النظام عند الناس اللي شغالة عليه.
ببساطة :**تحافظ على التوافق**.**(Backward Compatibility)**

◆ REST Principles

RESTعبارة عن مجموعة مبادئ لتنظيم شكل الـ:API

1. **Stateless:** كل Request منفصل عن التاني.
2. **Client–Server:** الـ Frontend والـ Backend مستقلين.
3. **Uniform Interface:** نفس الشكل ونفس طريقة التعامل في كل.Endpoint
4. **Cacheable:** الردود ممكن تتخزن لتسريع الأداء.
5. **Layered System:** ممكن يبقى فيه طبقات وسيطة.(Proxy, Gateway)
6. **Code on Demand (اختياري :**(السيرفر ممكن يبعث كود يتنفذ عند العميل (نادراً ما بنستخدمها).

◆ REST Naming Conventions

عشان الـ API يبقى سهل ومفهوم:

- استخدم **أسماء جمع:(Plural nouns)**
- /api/users
- /api/products
- استخدم **HTTP Methods** بدل الأفعال في الاسم:
- GET /users → جِب المستخدمين

- POST /users → أضف مستخدم
- استخدم **Lowercase + Hyphens** مش CamelCase:
- ✅ /api/user-orders
- ❌ /api/UserOrders
- استخدم Sub-Resources للعلاقات:
- /api/users/5/orders

🔷 What, When, and Why — Rate Limiting

What:
Rate Limiting يعني تحدد عدد الطلبات اللي المستخدم يقدر يبعتها في وقت معين.
زي: "100 request لكل دقيقة."

When:
لما تبدأ الـ API تتعرض لضغط كبير أو استخدام مفرط (من مستخدم أو من هجوم).

Why:
علشان:

- تحافظ على استقرار السيرفر
- تمنع إساءة الاستخدام
- تضمن عدالة بين كل المستخدمين

عادةً الـ API بيرجع Status Code زي:

429 Too Many Requests

لو المستخدم عدّى الحد المسموح.

الخلاصة:

- Versioning يحافظ على التوافق.
- REST Principles بتنظم تصميم الـ API.
- Naming Conventions بتخلي الكود واضح وسهل.
- Rate Limiting بتحمي النظام من الانهيار.