MON1 — Clean API avec GraphQL

10–15 minutes pour comprendre l'essentiel

- Pourquoi GraphQL (vs REST)
- 3–4 features clés avec exemples
- Mini setup Apollo (server + client)
- Bonnes pratiques et limites

Le problème avec REST

Contexte: Applications mobile (réseau social / blog)

```
GET /users/123         → { id, name, email, avatar, bio, ... }
GET /users/123/posts   → [{ id, title, content, date, ... }]
GET /posts/456/stats   → { views, likes, comments }
```

❌ **3 requêtes réseau** • ❌ **Over-fetching** (données inutiles) • ❌ **Under-fetching** (stats manquantes)

La solution GraphQL

Une seule requête, données exactes:

```
query UserProfile($id: ID!) {
  user(id: $id) {
    name
    avatar
    posts(first: 5) {
      title
      date
      stats {
        views
        likes
      }
    }
  }
}
```

1 requête • Données exactes • Typé

GraphQL aujourd'hui

Adoption en production :

- **Netflix** : Fédération de 100+ microservices
- **GitHub, Shopify, Meta** : APIs publiques
- **Apollo Federation** : Un schema unifié sur plusieurs équipes

Écosystème mature :

- Tooling complet (IDE, tests, monitoring)
- Caches intelligents (Apollo, Relay)
- Types TypeScript générés depuis le schéma

➡️ **Plus qu'un simple remplaçant des REST APIs**

Vocabulaire minimal

- Opérations: Query (lecture), Mutation (écriture)
- Schéma: types, interfaces, fragments
- Résolveurs: "où et comment" récupérer les données
- Erreurs partielles: `data` + `errors` dans la même réponse

Exemple (GitHub GraphQL API):

```
query {
  viewer {
    login
  }
}
```