Orange
Digital Center

@Orange Digital Center
Club Fianarantsoa

Découvrez la programmation backend

Cachez-vous à l'arrière plan de votre utilisateur

02-05/09/2025

ODC Club Fianarantsoa
032 07 110 38

Presentation

Qui vous êtes?

Engage 2025

1 Concepts fondamentaux

Tout ce qu’il faut savoir sur backend

Qu’est-ce que le Backend ?

Le backend est la **partie invisible d'une application** ou d'un site web qui s'occupe de la **gestion des données**, de la **logique applicative** et des interactions avec **les serveurs et les bases de données.**

Il assure les **fonctionnalités non visibles** pour l'utilisateur, telles que le **stockage des données**, le **traitement des requêtes** envoyées par le frontend, la sécurité et la performance de l'application.

Différence Backend vs Frontend

La **façade** d'une application est ce que l'utilisateur voit et avec quoi il interagit (design, menus, formulaires), tandis que le **fond** est la partie invisible en coulisse qui gère les données, le serveur, les bases de données et assure le bon fonctionnement de l'application.

Le **front-end** se concentre sur l'expérience utilisateur et l'interface visuelle, tandis que le **back-end** assure la logique métier, la sécurité et l'infrastructure nécessaire pour que tout fonctionne.

Ce que le backend gère

Serveurs

Le code qui fonctionne sur le serveur pour gérer l'ensemble des fonctionnalités.

Logique d'application

Les règles et processus qui font fonctionner l'application.

Bases de données

Stockage, récupération et organisation des informations.

API

Permet au backend de communiquer avec d'autres services ou applications.

Applications Web et Sites Web

1 E-commerce

Site spécialisé dans la **vente en ligne**.
Fonctionnalités : panier, paiement sécurisé, gestion des produits, comptes clients.
Exemple : Amazon, Jumia.

2 Software as a Service

Application **hébergée sur le cloud** et accessible par navigateur.
Pas besoin d'installer de logiciel, l'utilisateur paie un abonnement.
Exemple : Google Drive, Canva, Trello.

3 Blog

Site avec des **articles régulièrement publiés**.
Orienté partage d'informations, opinions, actualités

Applications Web et Sites Web

4 Landing Page

Page web **unique** conçue pour une campagne marketing.
Objectif : inciter à une action (achat, inscription, téléchargement).
Exemple : page d'inscription à un webinar.

5 Portfolio

Site personnel mettant en avant les **réalisations et compétences**.
Utilisé par les développeurs, designers, photographes…
Exemple : portfolio d'un graphiste avec ses projets.

Différence clé : Web statique vs Web dynamique

Critère	Web statique	Web dynamique
But	Présenter un contenu fixe	Générer un contenu en temps réel
Interaction	Faible (lecture uniquement)	Forte (actions utilisateur, personnalisation)
Backend	Pas nécessaire (HTML/CSS)	Toujours nécessaire (PHP, Node.js, Python, ...)
Base de données	Aucune ou minimale	Reliée à une base (SQL, NoSQL)
Exemple	Site vitrine simple, portfolio statique	Réseau social, blog avancé, e-commerce, SaaS

Communication Client vs Server

Serveur ?

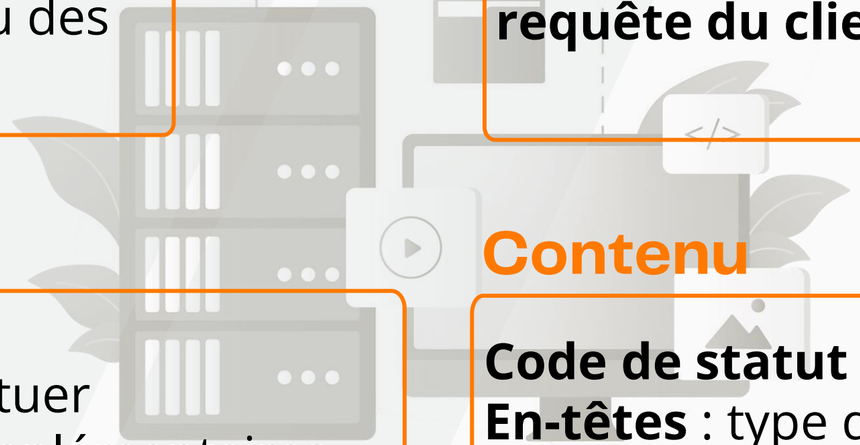
Client ?

Requêtes HTTP

Un **message envoyé par le client** au serveur pour demander une action ou des données.

Reponses HTTP

Le serveur renvoie une **réponse à la requête du client**.



Contenu

URL : adresse de la ressource
Méthode HTTP : type d'action à effectuer
En-têtes (Headers) : informations supplémentaires
Corps (Body) : données envoyées au serveur (par POST/PUT)

Contenu

Code de statut : indique le résultat (200 ,404,..)
En-têtes : type de contenu, taille,...
Corps : données demandées (JSON, XML,..)

Principales méthodes HTTP

Méthode	Utilisation	Exemple d’usage
GET	Récupérer des données	Obtenir la liste des utilisateurs
POST	Créer une nouvelle ressource	Ajouter un nouvel utilisateur
PUT	Mettre à jour une ressource complète	Modifier toutes les infos d’un utilisateur
PATCH	Mettre à jour partiellement une ressource	Modifier seulement l’email d’un utilisateur
DELETE	Supprimer une ressource	Supprimer un utilisateur spécifique

HTTP Status code

Classe	Code -- Nom
1xx – Information	**100 --** Continue
2xx – Succès	**200 --** OK
	201 -- Created
	204 -- No Content
3xx – Redirection	**301 --** Moved Permanently
	302 -- Found
	304 -- Not Modified
	400 -- Bad Request
4xx – Erreurs Client	**401 --** Unauthorized
	403 -- Forbidden
	404 –- Not Found
	500 -- Internal Server Error
5xx – Erreurs Serveur	**502 --** Bad Gateway

Exemple concret requête avec JSON

Requête GET :

```
GET /users HTTP/1.1
Host: api.example.com
Authorization: Bearer <token>
```

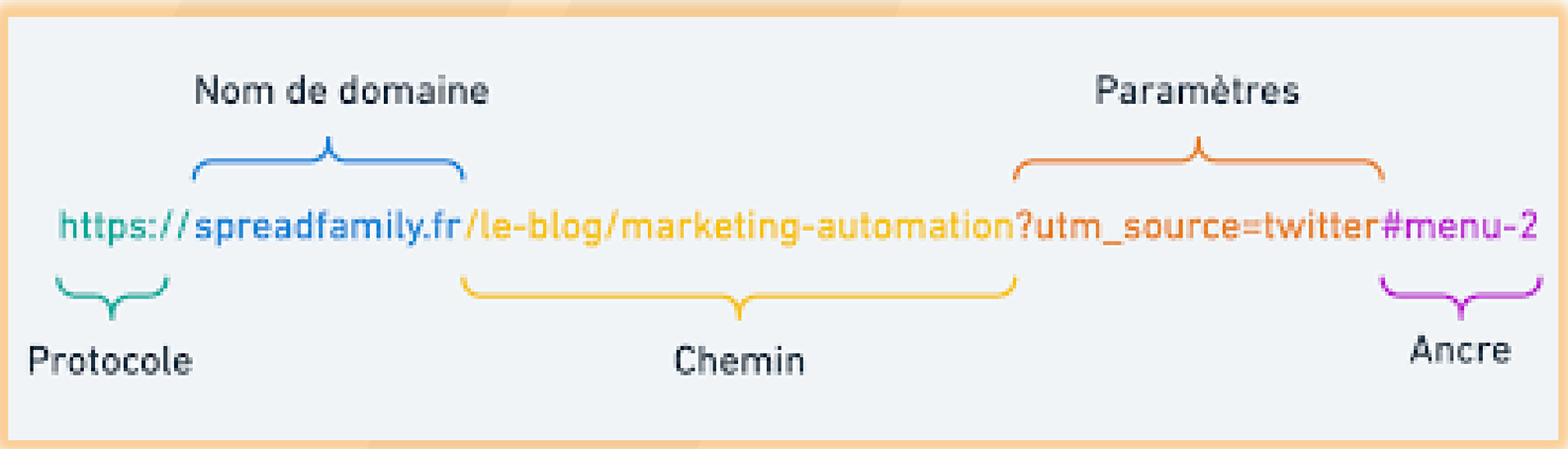
Réponse :

```
[
  { "id": 1, "name": "Amoros" },
  { "id": 2, "name": “Jean" }
]
```

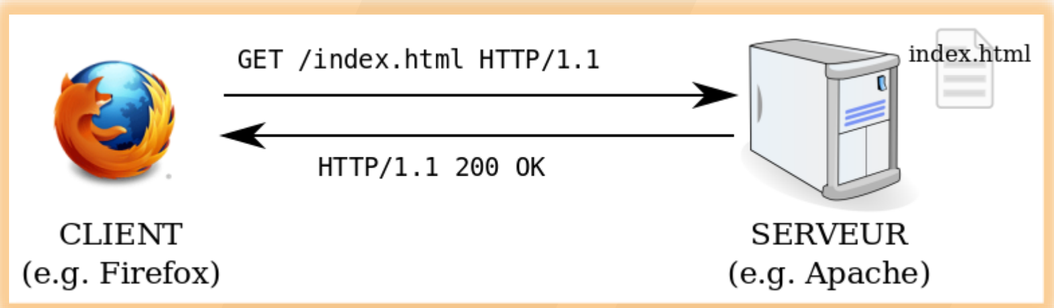
Axios

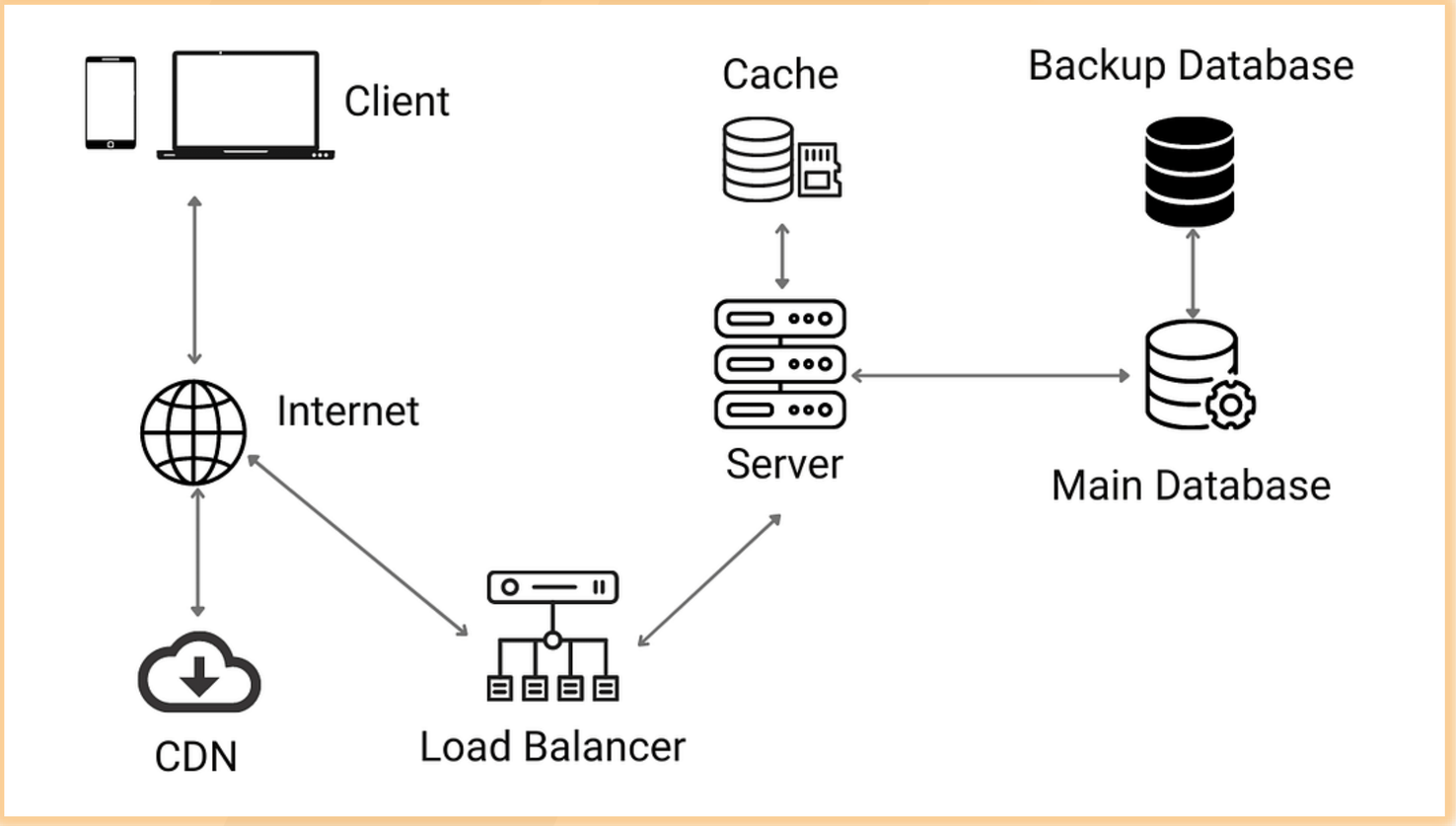
fetch

Structure de l’Url



Exemple concret methode http





2 API (Application Programming Interface)

Interface de programmation d’application

Qu’est ce qu’une API ?

API (Application Programming Interface) : interface qui permet à deux applications de communiquer entre elles.

Sert à **exposer des fonctionnalités** ou **échanger des données** entre le client (frontend) et le serveur (backend).

Les APIs utilisent différents **protocoles et styles d’architecture**.

Types Architecture d’API

1 REST (Representational State Transfer)

Basé sur le protocole **HTTP**.
Utilise les **méthodes HTTP** (GET, POST, PUT, DELETE).
Format d’échange le plus courant : **JSON**.
Simplicité et compatibilité avec presque tous les langages.

2 GraphQL

Développé par **Facebook**.
Permet au client de **choisir exactement les données** dont il a besoin.
Une seule **requête flexible** au lieu de plusieurs endpoints.
Format : **JSON**.
Avantage : limite la surcharge réseau (on ne reçoit que ce qu’on demande).

Types Architecture d’API

3 SOAP (Simple Object Access Protocol)

Plus ancien, basé sur **XML**.
Utilise souvent **HTTP** mais aussi d’autres protocoles (SMTP, TCP…).
Très **strict et lourd** mais **fiable** pour les environnements critiques (banques, assurance, télécoms).
Supporte des fonctionnalités avancées (transactions, sécurité WS-Security).

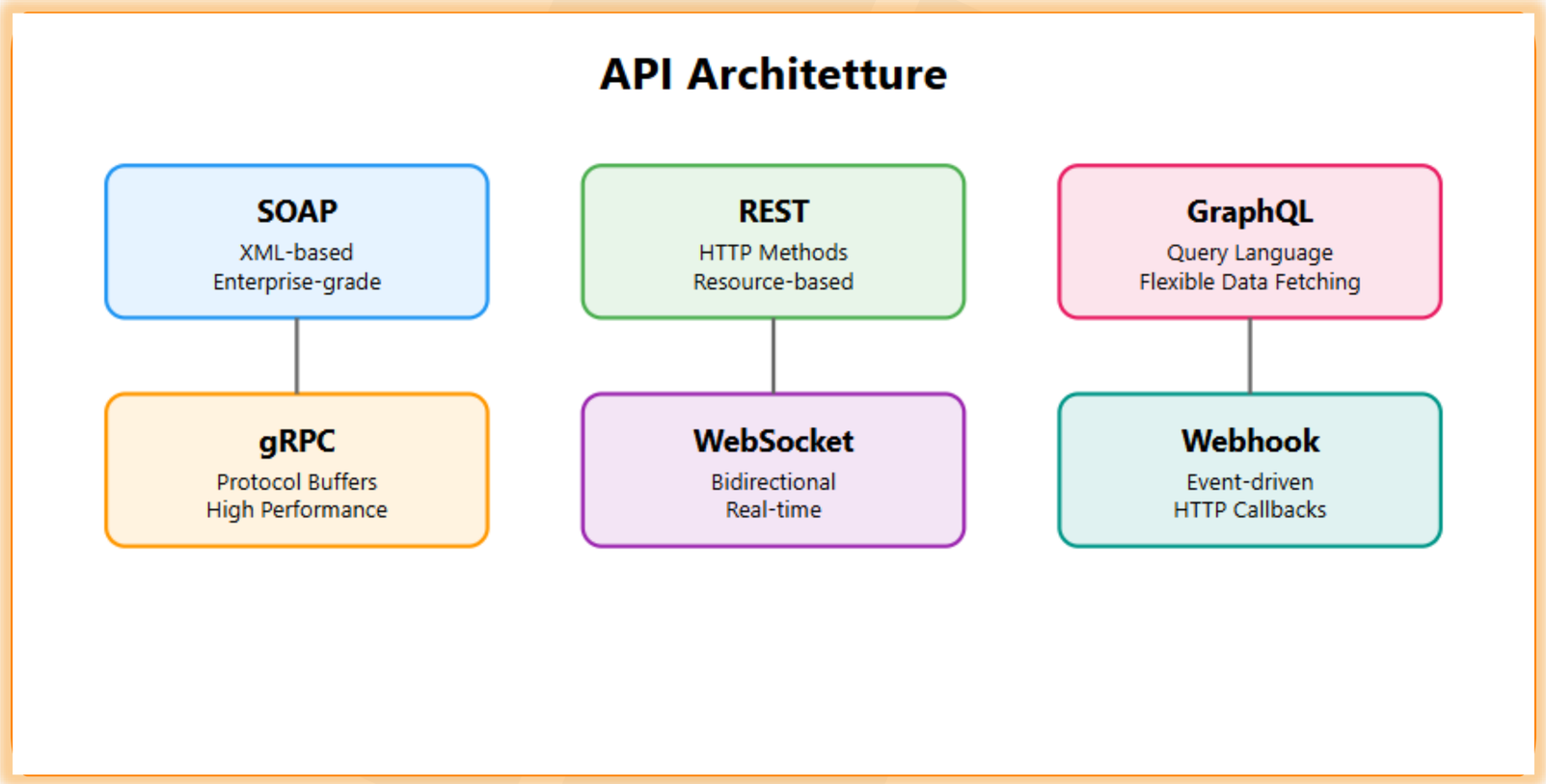
4 gRPC (Google Remote Procedure Call)

Développé par **Google**.
Très **rapide** et optimisé pour les communications **service-to-service (microservices)**.
Utilise le protocole **HTTP/2** et des formats binaires.
Plus efficace que **REST/JSON** quand on a beaucoup de données ou des services distribués.

Résumé Comparatif

API Style	Format	Avantage	Inconvénient
REST	JSON	Simplicité, popularité	Données parfois trop volumineuses
GraphQL	JSON	Flexible, pas de surcharge	Plus complexe à mettre en place
SOAP	XML	Fiable, sécurisé (banques)	Lourd, verbeux
gRPC	Protobuf (binaire)	Très rapide, idéal microservices	Moins universel que REST

Résumé Comparatif



Ne pas confondre REST et RESTful

RESTful

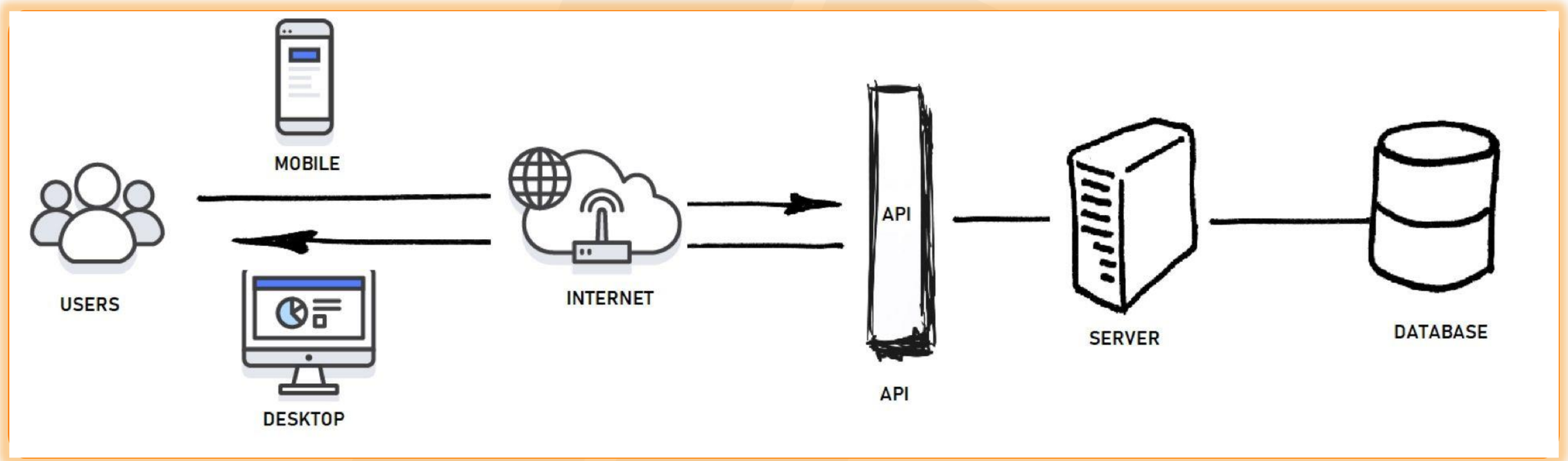
Une **API qui respecte les principes REST** est dite **RESTful**.
Exemples de bonnes pratiques RESTful :
Utiliser les bonnes méthodes HTTP (GET, POST, PUT, DELETE)
Utiliser des **endpoints clairs et logiques** :
    /users → liste des utilisateurs
    /users/1 → utilisateur avec id = 1
Réponses en **JSON/XML**
Respect des **codes de statut HTTP**

Tableau comparatif

Critère	REST	RESTful
Définition	Style architectural (théorie)	Application pratique de REST
Nature	Conceptuel	Concret
Objectif	Fournir des règles pour concevoir des APIs	Créer une API conforme à REST
Exemple	« Une API doit être stateless »	Une API /users avec GET, POST, PUT, DELETE
Métaphore	La recette de cuisine	Le plat préparé

Place de l’API

Standarisé la communication client - server



Architecture MVC

Modèle d'architecture logicielle

M Modèle

Responsable de la **gestion des données**.
Accède à la base de données (SQL, NoSQL).
Contient la **logique métier** (règles de validation, calculs).

V Vue

Gère la **présentation** des données au client.
Ne contient pas de logique métier.
Reçoit les données préparées par le **Controller** et les affiche.
Exemple : Une page HTML qui montre la liste des utilisateurs.

C Contrôleur

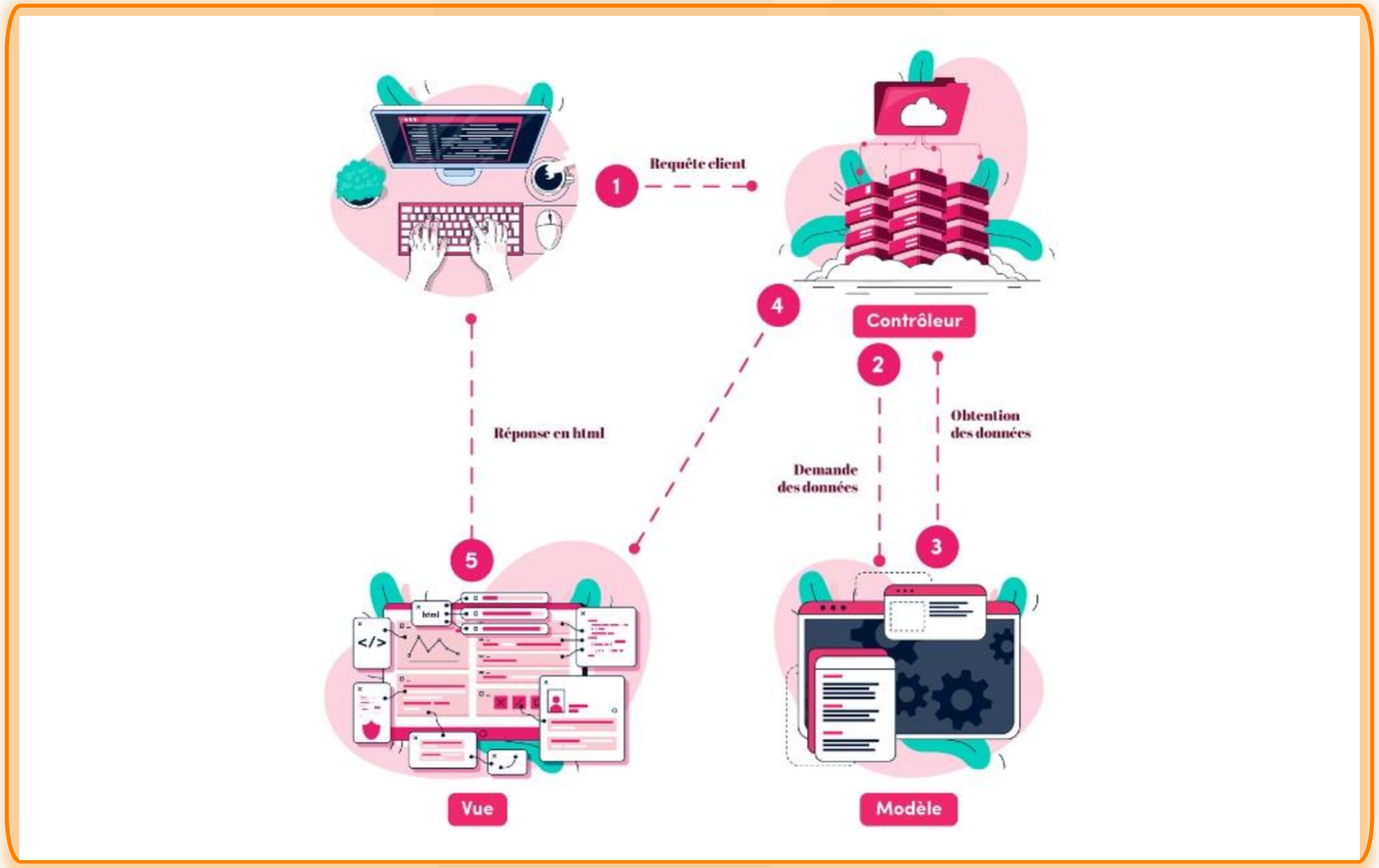
Reçoit les **requêtes du client**
Utilise le **Model** pour récupérer ou modifier les données.
Envoie le résultat à la **View** pour l'affichage.

Tableau: Récapitulatif

Composant	Rôle	Exemple
Model	Gère les données et la logique métier	UserModel → récupère les utilisateurs dans la base
View	Affiche les données au client	JSON ou page HTML avec la liste des utilisateurs
Controller	Reçoit la requête, appelle le Model et renvoie la réponse via la View	UserController.getUsers()

Communication MVC en réalité

Model – Vue - Controller



3 Langages et Frameworks populaires

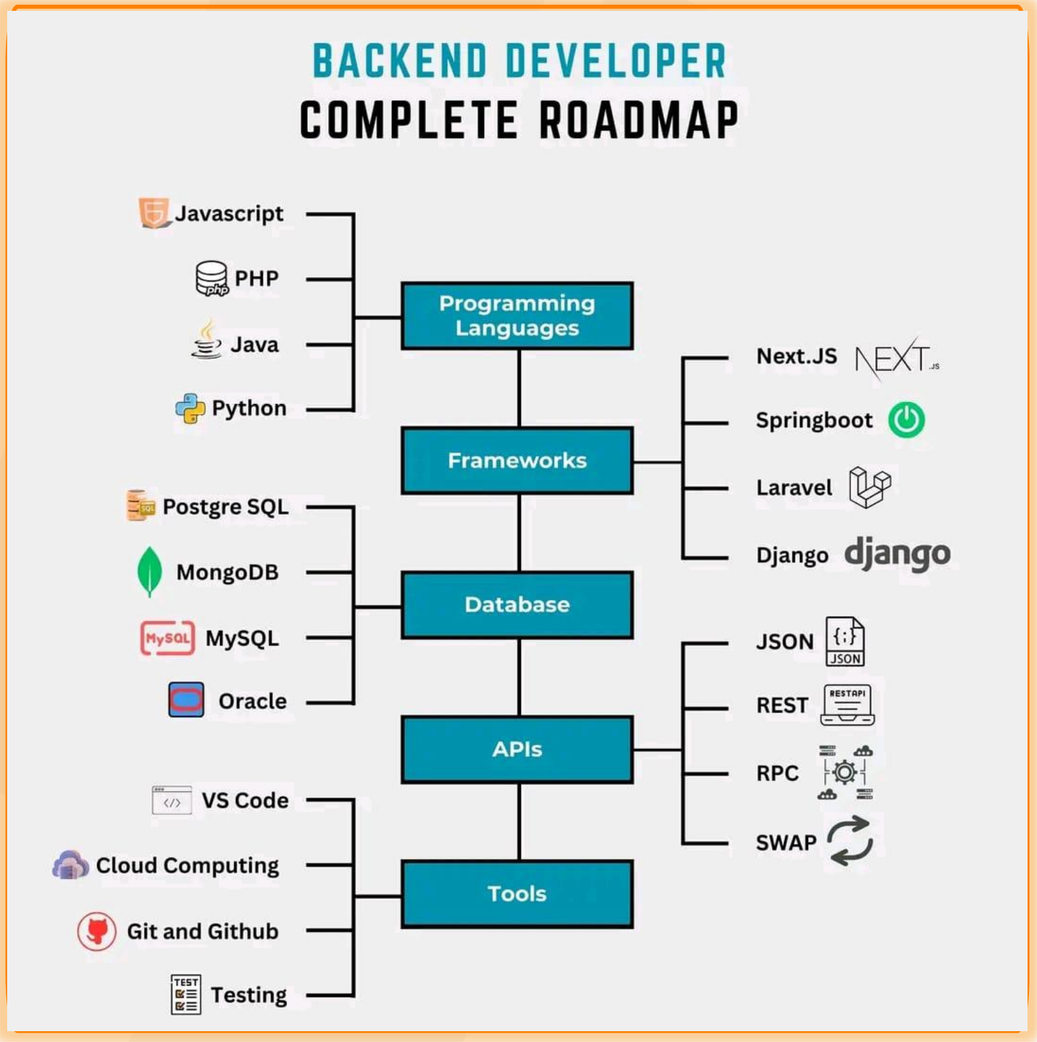
Ce sont les bases qu’un développeur utilise pour écrire du code

Langages de programmation ? Frameworks ?

Langage de programmation est un système formel de règles et de symboles qui permet aux **humains de communiquer** des instructions et des logiques à un ordinateur pour créer des **logiciels et des applications**. Un **framework** est un **cadre de travail logiciel** qui fournit une structure et des composants réutilisables pour développer des applications. Il offre un ensemble **d'outils**, de **bibliothèques** et de modèles de conception **pour simplifier** et accélérer le développement, en évitant aux développeurs de repartir de zéro.

Contemplez l’image ci-dessous

Résumons en clin d’oeil



4 SGBD ou DBMS

Système de Gestion de Base de Données

Qu’est-ce qu’un SGBD ?

Logiciel permettant de **stocker, organiser, gérer** et **sécuriser** les données.

Garantit la **cohérence, l’intégrité** et l’accès concurrent aux **données**.

Exemples : MySQL, PostgreSQL, OracleDB (**relationnel**), MongoDB, Redis, Cassandra (**NoSQL**).

Relationnel (SQL) vs Non relationnel (NoSQL)

S

Relationnel (SQL)

SQL (Structured Query Language) : langage standard pour interagir avec les bases de données relationnelles.
Bases relationnelles : données organisées en **tables**, chaque table ayant **lignes** (enregistrements) et **colonnes** (champs).
Les relations entre tables se font via **clés primaires** et **clés étrangères**.
Ex : MySQL, PostgreSQL, OracleDB, MS SQL Server, SQLite

N

Non relationnel (NoSQL)

NoSQL : “Not only SQL”, désigne des bases de données **qui ne suivent pas le modèle relationnel traditionnel**.
Les données sont **stockées de manière flexible**, souvent sans schéma fixe, pour s’adapter à des volumes massifs ou des structures complexes.
Idéales pour des applications nécessitant **scalabilité horizontale** et **haute performance**.
Ex: MongoDB, Redis, Cassandra

Interaction avec les bases de données

1 Requêtes SQL (langage structuré)

Utilisé dans les **bases relationnelles (SQL)**.
Permet les opérations **CRUD** :

Action	Description	Exemple SQL
C (Create)	Ajouter un enregistrement	INSERT INTO Utilisateurs (Nom, Email) VALUES ('Amoros', 'amoros@mail.com');
R (Read)	Lire/consulter des données	SELECT * FROM Utilisateurs WHERE Nom='Amoros';
U (Update)	Modifier un enregistrement	UPDATE Utilisateurs SET Email='new@mail.com' WHERE Nom='Amoros';
D (Delete)	Supprimer un enregistrement	DELETE FROM Utilisateurs WHERE Nom='Amoros';

Interaction avec les bases de données

2 ORM (Object-Relational Mapping)

Technique permettant d'**interagir avec la base via des objets**, sans écrire directement du SQL.
L'ORM traduit les méthodes et objets en requêtes SQL (ou NoSQL).
Avantages :
Plus simple à utiliser pour les développeurs (travail orienté objets).
Indépendance vis-à-vis du **SGBD** (on peut changer de MySQL à PostgreSQL sans tout réécrire).
Gestion automatique des relations, migrations et validation.

Exemples d'ORM :

Sequelize → pour Node.js et SQL (MySQL, PostgreSQL, SQLite).

Hibernate → pour Java et SQL (OracleDB, MySQL, PostgreSQL).

Mongoose → pour Node.js et MongoDB (NoSQL).

5

Conclusion

Revoyez dès le debut le contenu de formation, ce ça le resumé. 😊 😊

@orangedigitalcenter.fianar@gmail.com