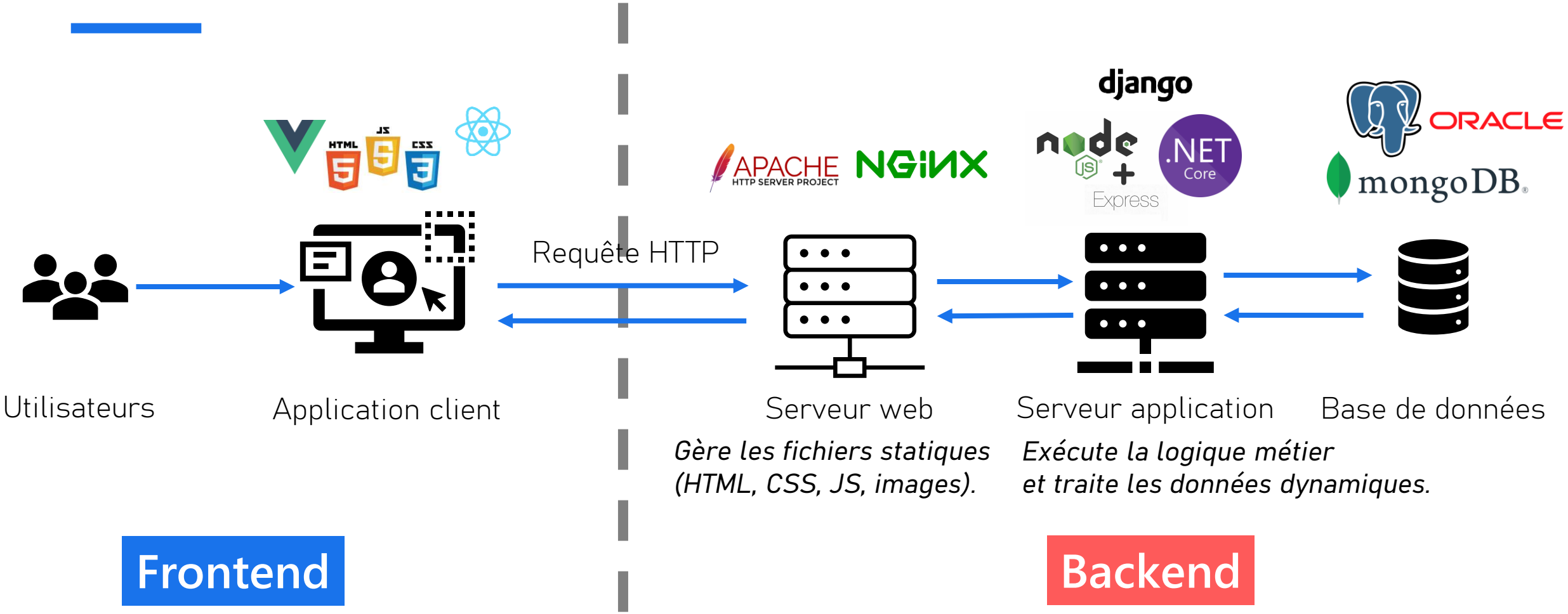
420-707-AH

Architecture REST et Prise en main de Express.js

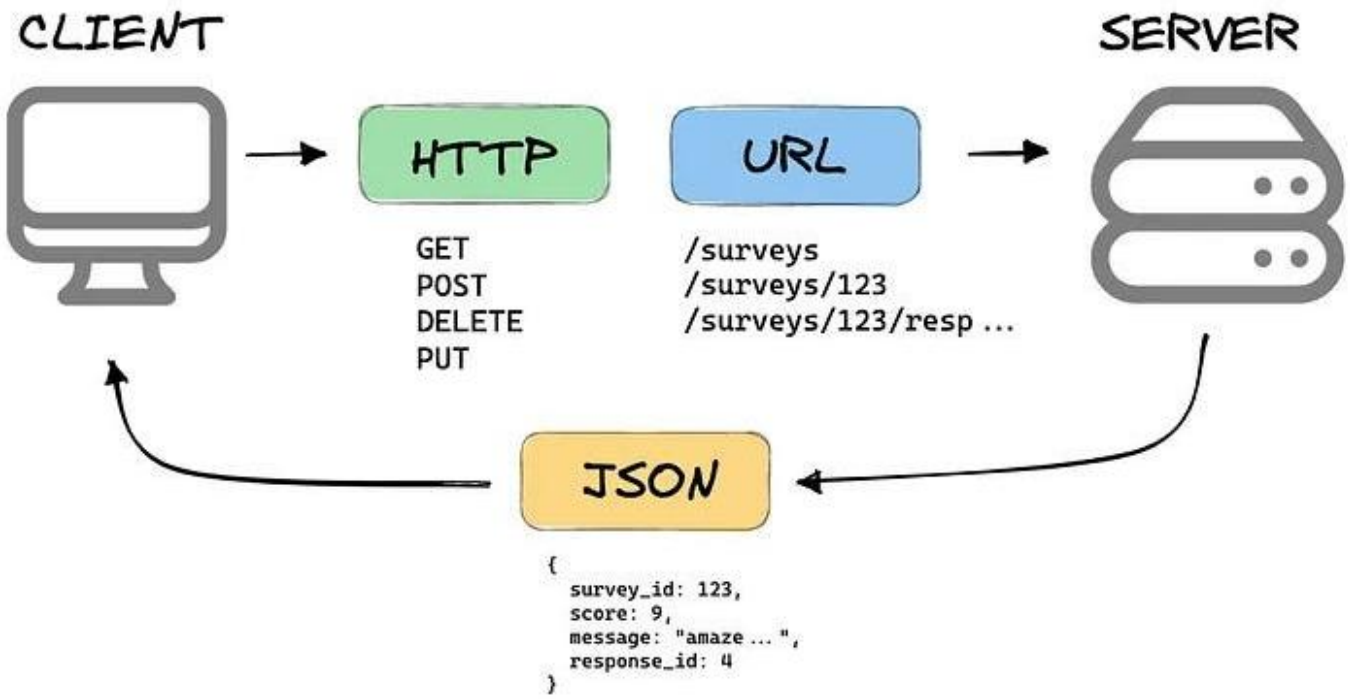
INTRODUCTION

LOUIS-EDOUARD LAFONTANT



Architecture client-serveur

Architecture REST



DÉVELOPPEMENT D'API

Qu'est-ce que REST ?

- REST = Representational State Transfer
- Style d'architecture utilisé principalement pour créer des API web.
- Repose sur un **ensemble de principes** qui permettent aux systèmes informatiques de communiquer facilement via le protocole HTTP en utilisant des méthodes standards
  - **GET**: Lire/Récupérer une ressource (sans modification)
  - **POST**: Créer une nouvelle ressource
  - **PUT**: Modifier ou remplacer **complètement** une ressource existante
  - **DELETE**: Supprimer une ressource

Caractéristiques de REST

- **Stateless (sans état)** : chaque requête du client au serveur doit contenir toutes les informations nécessaires à sa compréhension. Le serveur ne conserve pas d'état entre les requêtes.
- **Ressources identifiables** : les données sont représentées sous forme de ressources (ex : utilisateurs, articles), identifiées par des URI (Uniform Resource Identifier).
- **Utilisation des méthodes HTTP** : chaque méthode a une signification claire (ex : GET pour lire, POST pour créer, PUT pour modifier, DELETE pour supprimer).
- **Représentations** : une ressource peut être représentée sous différentes formes (souvent en JSON ou XML), et le client peut spécifier le format désiré via les en-têtes HTTP.

Qu'est-ce qu'une ressource?

Une **ressource** est une **donnée ou un objet identifiable** par une URL unique (URI). Elle peut être un produit, un utilisateur, un article, une commande, etc.

Caractéristiques d'une ressource

- **Identifiable** par une URI
  - Exemple : **`/produits/42`** identifie le produit avec l'ID 42
- **Accessible via des verbes HTTP**
  - **`GET /produits/42`**: Récupérer la ressource produit associé à l'identifiant 42
  - **`PUT /produits/42`**: Modifier la ressource produit associé à l'identifiant 42
  - **`DELETE /produits/42`**: Supprimer la ressource produit associé à l'identifiant 42

Alternatives

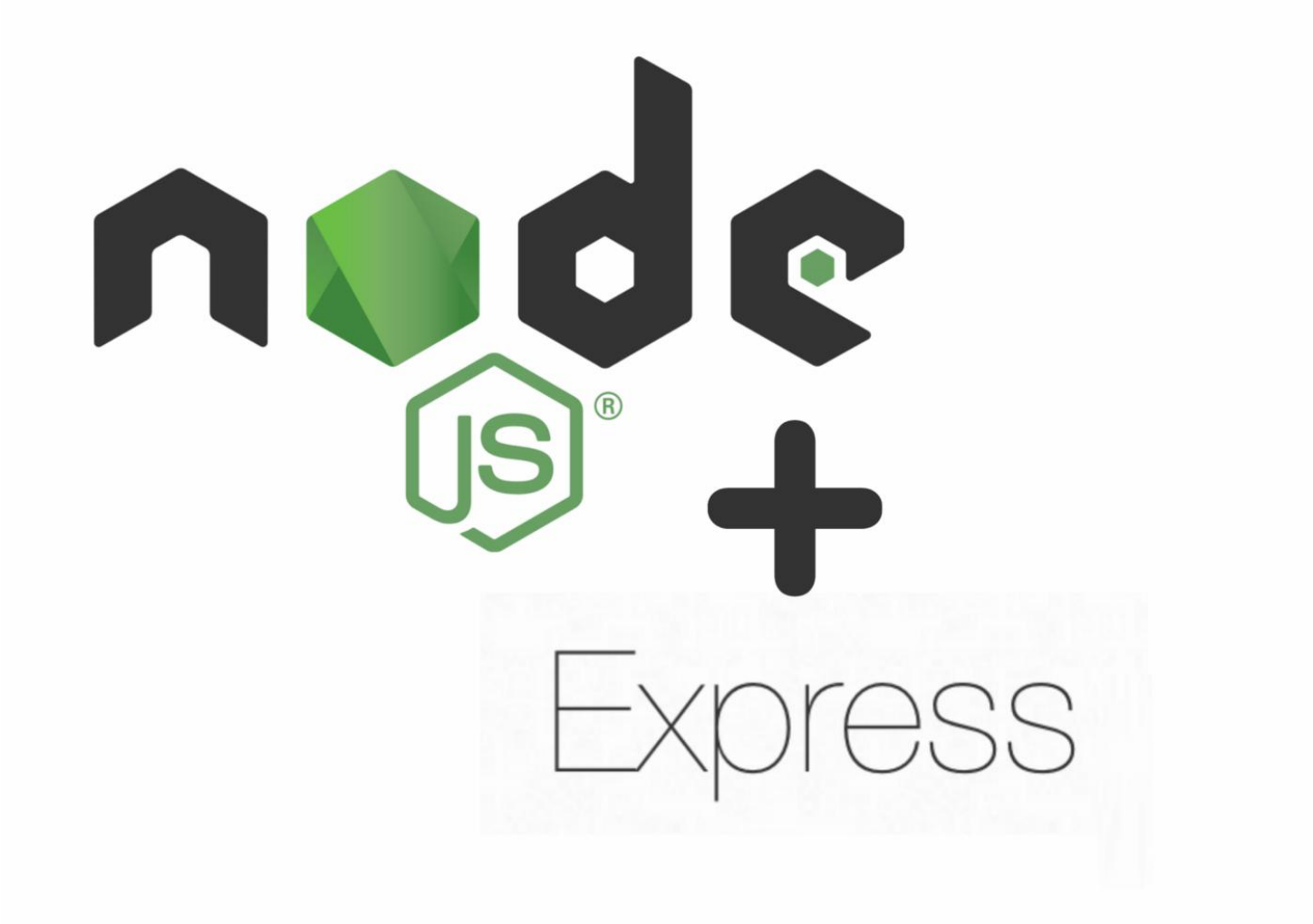
GraphQL – Créé par Facebook.

- Principe : Le client définit précisément les données dont il a besoin, et le serveur répond exactement à cette demande.
  - Une seule requête peut remplacer plusieurs appels REST.
  - Plus complexe à mettre en œuvre.

gRPC (Google Remote Procedure Call) – Créé par Google.

- Principe : Basé sur des appels de procédures à distance (RPC) utilisant Protocol Buffers (format binaire très compact).
  - Idéal pour la communication entre microservices.
  - Moins lisible que REST ou GraphQL (pas recommandé pour une API publique consommée par navigateur)

Express.js



CRÉER UN SERVEUR

Express.js

1. Serveur basé sur Node.js

- Utilise le moteur **JavaScript côté serveur** de Node.js.
- Permet de gérer des milliers de connexions simultanées de manière efficace (asynchrone et non bloquant).

2. Routing (système de routes)

- Associe des URL à des fonctions (appelées gestionnaires).Syntaxe simple :
  ```
  app.get('/accueil', (req, res) => res.send('Bienvenue'));
  ```

3. Gestion des paramètres (req, res)

- **`req`** : objet représentant la requête du client (URL, paramètres, données).
- **`res`** : objet pour envoyer la réponse (texte, JSON, page HTML, etc.).

Express.js

4. Middleware

- Fonctions intermédiaires entre la requête du client et la réponse du serveur.
- Utilisés pour l'**authentification**, la **validation des données**, le **logging**, etc.

```
app.use((req, res, next) => {
  console.log('Requête reçue');
  next(); // Passe à l'étape suivante
});
```

Installation

1. Installer Node.js (si ce n'est pas déjà fait): https://nodejs.org
2. Créer un dossier de projet
3. Initialiser le projet avec Node.js: **`npm init -y`**
4. Installer Express: **`npm install express`**
5. Créer le fichier princial (**`index.js`**)
6. Lancer le serveur: **`node index.js`**

Code minimal (fichier principal)

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Bienvenue sur mon serveur Express!');
});

app.listen(3000, () => {
  console.log('Serveur démarré sur http://localhost:3000');
});
```

Code minimal (version JSON)

```
const express = require('express');
const app = express();
app.use(express.json());

app.get('/', (req, res) => {
  res.json('Bienvenue sur mon serveur Express!');
});

app.listen(3000, () => {
  console.log('Serveur démarré sur http://localhost:3000');
});
```

Postman

TESTER SON API

Exercice: Mini API Étudiants

- Créez une API Express avec une route **`/students`**
- La route **`GET /students`** retourne une liste d'étudiants en JSON
- Bonus : Ajouter POST pour ajouter un étudiant
- Testez l'API avec Postman