Architectures

Voici quelques aides / explications vis à vis de ce que vous apprenez et de votre parcours.

Vous adressez en premier lieu 3 langages (HTML / CSS / JavaScript) qui vous permettent de développer ce que l'on qualifie de ressources statiques.

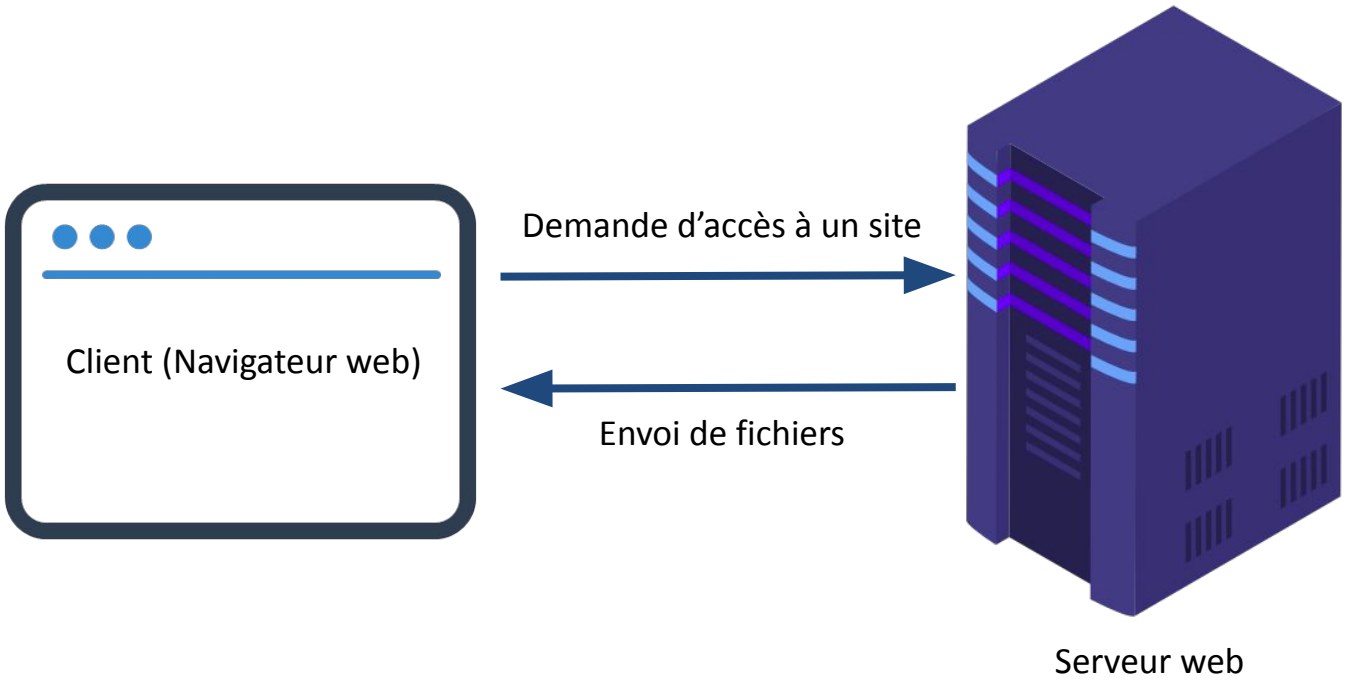
Une ressource statique est une ressource dont le contenu ne changera pas en fonction de la personne qui les récupère. Ce sont des fichiers destinés à être hébergés sur un serveur web.

Un serveur web est un serveur sur lequel on installe une application dont le rôle est de mettre à disposition ces ressources statiques.

Typiquement, un site web vitrine est constitué uniquement de ressources statiques.

Les 3 grandes applications du marché pour remplir ce rôle sont : Microsoft IIS , Apache HTTP Server, Nginx

Architectures



Architecture client / serveur classique.
Un navigateur web interroge un serveur web afin de récupérer des ressources statiques.
Sur ce serveur web, on installe une application de type Apache / IIS ou Nginx.
Le rôle de cette application est de comprendre les requêtes HTTP, d'aller récupérer les fichiers demandés sur le disque dur et de les envoyer aux clients.

Architectures

Dans le cas de figure du “site web”, ou des “ressources statiques”, il n’y a pas :

1. De sauvegarde de données en base de données
2. D’affichage différencié en fonction des utilisateurs
3. D’exécution de logiques métier spécifiques, en dehors de logiques d’affichage.

On est dans une logique **informative**.

On oppose le cas de figure du **site web** à celui des **applications web.**

Là ou un site web est informatif, une application web sera **interactive**.

Une application web interagit de manière directe ou indirecte avec une base de données.

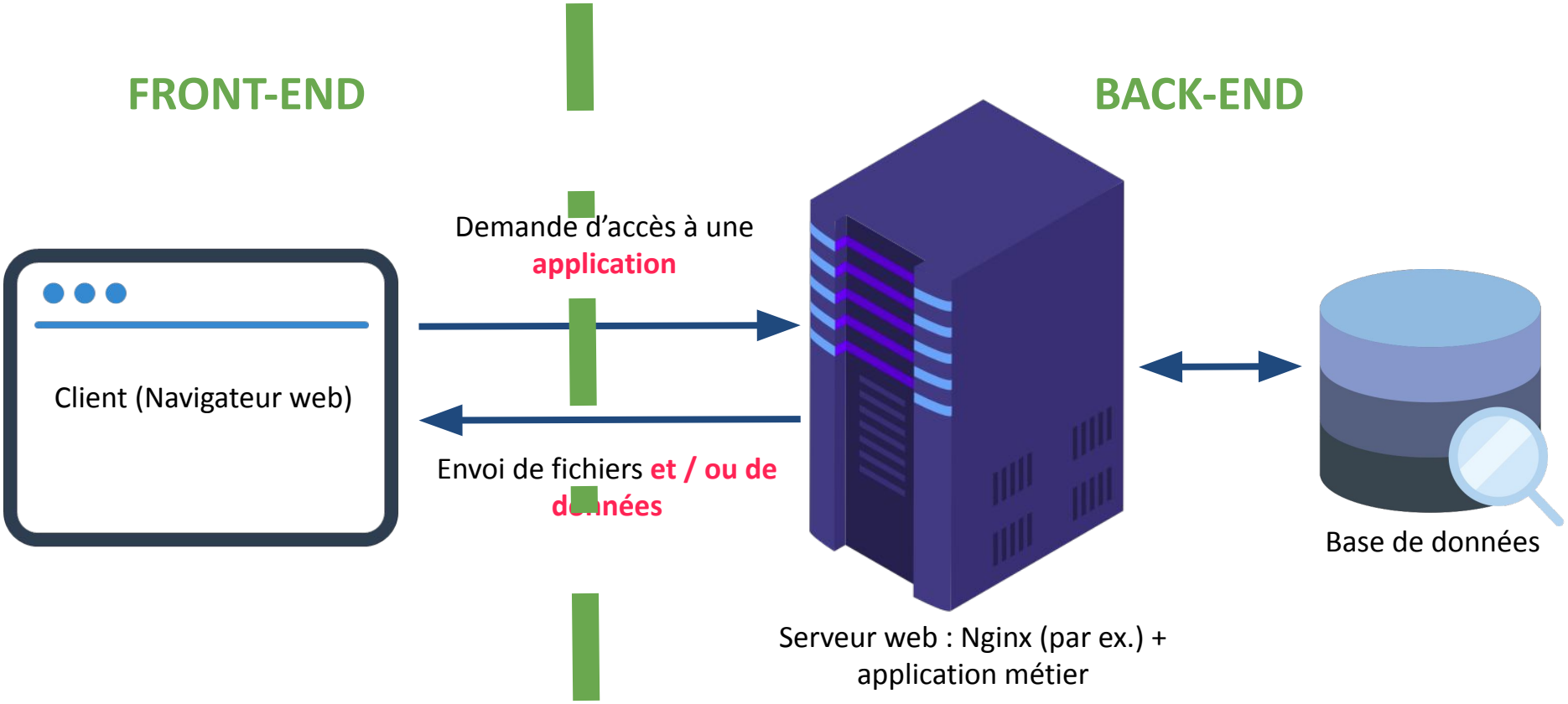
Architectures

Dans le cas d'une application web, chaque utilisateur pourra disposer de sa propre interface.

Par exemple, sur une application e-commerce :

1. Nous n'avons pas tous les mêmes suggestions de produits
2. Nous pouvons retrouver l'historique de nos commandes
3. Nous pouvons ajouter des produits dans un panier
4. Nous pouvons passer une commande et payer

Architectures



Dès lors que l'on parle "d'applications web" on va disposer d'un code "Front" et d'un code "Back".
Pour un site web standard on a uniquement un code "Front".

Notre serveur web embarquera toujours une application type Nginx permettant de délivrer certains fichiers qui resteront statiques (du HTML, CSS, JS, des images etc.) mais exécutera également un programme métier qui travaillera avec une base de données.

Architectures

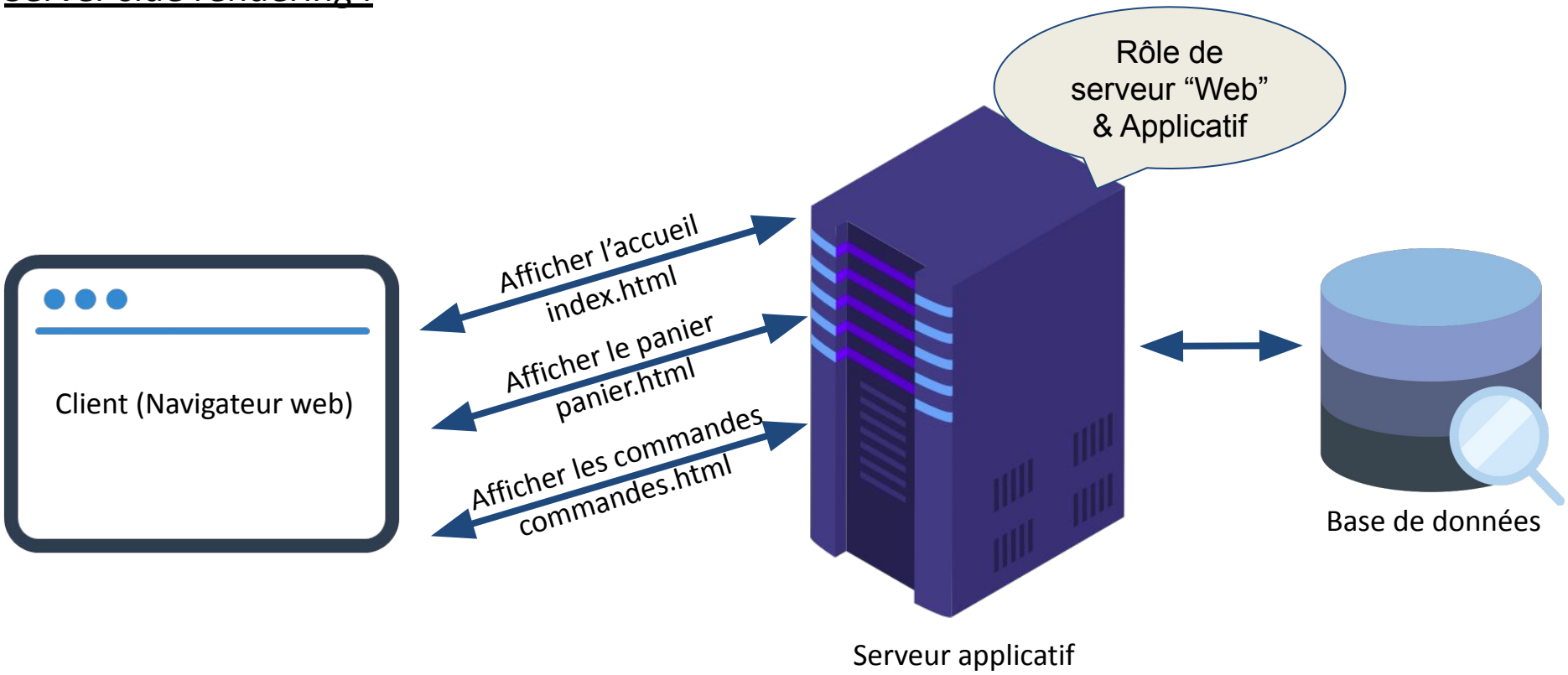
Il faut comprendre qu'il n'existe pas une seule méthode pour développer une application web, et notamment l'approche utilisée côté back dans le programme "métier" peut différer.

On va rencontrer deux grandes approches :

- ➔ le **server side rendering**
- ➔ le **client side rendering**

Architectures

Server side rendering :



Architectures

Dans le cas du server side rendering, nous allons utiliser un langage (par exemple, le PHP) pour générer des pages HTML au besoin en fonction des demandes des clients.

Nous allons donc générer des fichier HTML **dynamiquement**, par opposition aux ressources statiques, dont le contenu est toujours le même.

Il existera toujours des ressources statiques, comme des images, du CSS, des polices de caractère… mais plus de fichier HTML “en dur”.

Nos fichiers HTML seront générés spécifiquement selon notre utilisateur cible et à la demande.

Cette génération de fichiers se fera côté serveur, dans le programme “métier”.

<u>Notre serveur aura toujours 2 rôles :</u>

1. Héberger les ressources statiques (rôle du serveur web)
2. Exécuter le code permettant de générer les fichiers HTML au runtime (ie. à l’exécution) lorsque les navigateurs feront des demandes d’accès à certaines pages.

Architectures

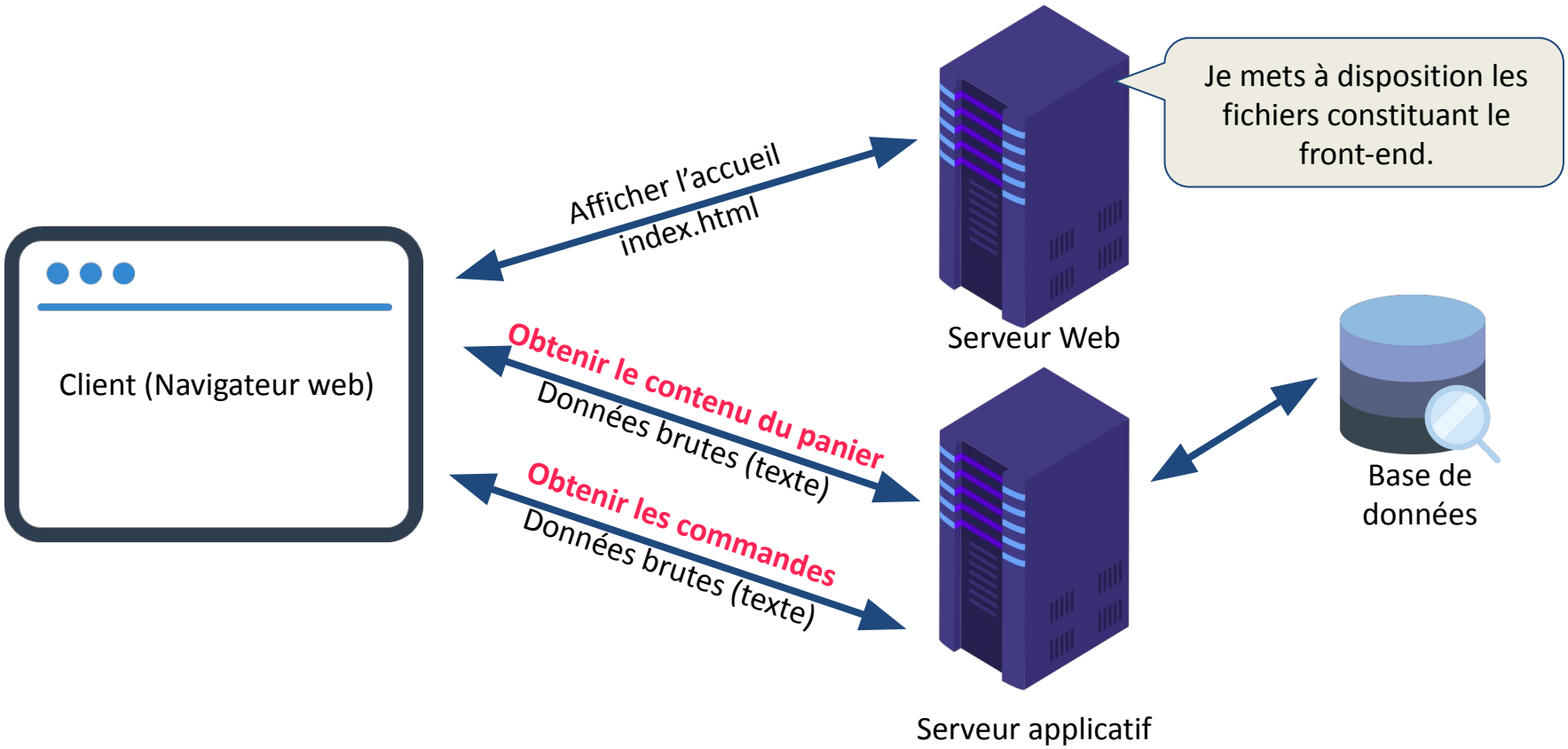
Dans le cas du server side rendering, notre application suivra en général l'architecture "**multi-page application**" (MPA).

Cela signifie que nous avons développé le code nécessaire (en PHP ou dans un autre langage) pour générer dynamiquement des fichiers HTML en fonction des demandes des clients, et que chaque action menée par l'utilisateur conduira à recharger une nouvelle page.

Cela n'empêchera pas pour autant que certaines pages HTML embarquent du JavaScript qui modifiera des fragments de page en AJAX lorsque l'on souhaitera éviter un chargement complet d'une nouvelle page.

Architectures

Client side rendering :



Architectures

Dans le cas du client side rendering, notre application suit l'architecture "**single-page application**" (SPA).

En effet le client ne télécharge qu'un seul fichier html : index.html

Ce fichier est autonome, et lorsque l'on navigue dans une SPA, on récupère des données brutes de la part du serveur sans changer de page côté navigateur.

C'est à dire qu'au lieu de nous envoyer de nouveaux fichiers HTML, le serveur web nous envoie uniquement des données non formatées.

C'est à la responsabilité du code JavaScript de récupérer ces données et de les traiter pour les afficher dans le DOM. C'est en cela que l'on dit que le rendering (ie. la génération des balises HTML affichées) est réalisé côté client, et non côté serveur.

Le fichier index.html téléchargé est **autonome**.

Architectures

Lorsque l'on travaille selon cette logique de SPA, notre code JavaScript interroge ce que l'on nommera des API : Application Programming Interface.

Une API est un programme qui permet d'exposer des actions sur des objets (exemple : planifier un rendez-vous médical).

Il existe différentes techniques pour développer une api, la plus connue et largement utilisée aujourd'hui est **REST**.

Notre programme "métier", que l'on embarquera sur notre serveur, aura donc pour rôle d'exposer une ou plusieurs API et de faire le lien avec la base de données.

Architectures

Le principe du REST est de créer des services web, c'est à dire une interface permettant à n'importe quel programme de contacter un autre programme, tout en utilisant les technologies du web.

C'est une architecture très utilisée dans le cadre du développement d'applications web.

On se sert du protocole HTTP pour communiquer entre programmes.

A la différence de l'usage traditionnel du protocole HTTP qui permet de télécharger un fichier (HTML par exemple), on utilisera ce protocole pour échanger des données métier (souvent JSON).

Architectures

REST signifie “representational state transfer”.

REST indique que l’on représente nos données sous la forme de ressources. Une ressource est un objet métier. Par exemple, un client est une ressource. Une commande est une ressource.

Le principe est de proposer les opérations dites CRUD pour récupérer, créer, modifier ou supprimer des ressources :

- CREATE, via le verbe (ou méthode) HTTP POST
- READ, via le verbe GET
- UPDATE, via le verbe PUT
- DELETE, via le verbe Delete

On identifie nos ressources dans l’URL : http://mondomaine.com/films

Dans certains cas on pourra ajouter l’identifiant de l’objet manipulé :
http://mondomaine.com/films/123bd432

Exemple concret : https://geo.api.gouv.fr/departements
Cette URL, associée au verbe GET permet de récupérer la liste des départements Français.

Architectures

A l'issue de votre formation, vous serez capables de développer une SPA ou une MPA :)

Lorsque l'on développe une SPA, on a une isolation forte entre le front et le back. On peut par exemple créer 2 fronts (un mobile et un web) dans des technos différentes, qui interrogeront le même back.

On pourra choisir un Framework front (Angular ) et un framework back dans une autre techno.

Chaque architecture a ses avantages et inconvénients :

- ➔ Meilleure expérience utilisateur avec une SPA et du CSR
- ➔ Moins bon référencement (voire absence de référencement) avec du CSR

Pour votre business case, vous aurez 2 interfaces développées en SSR, et une en CSR. Votre programme "métier" devra à la fois générer le HTML des 2 interfaces front-office et back-office, et la mise à disposition d'une API qui sera interrogée par un projet indépendant en Angular : le dashboard.

NB : Je vous ai présenté une version binaire du monde : CSR vs SSR. Pour aller plus loin, il existe des approches hybrides… :)

Architectures

Business case :