TCM://
THE CODING MACHINE

ECRIT PAR
Nicolas PEGUIN

DATE
Version révisée
Novembre 2020

SERIE : COMPRENDRE LA TECH

DO YOU SPEAK TECHNIQUE ?

Comprendre et maîtriser les principaux concepts techniques du digital : de la méthodologie de projet, aux principales architectures techniques web et mobile en passant par les API ou bien encore les bases de données... Vous saurez tout ce que vous n'avez jamais osé demander sur la technique !

Note sur cette nouvelle version :

Ce livre blanc fait partie d'une série "Comprendre la tech" qui vise à faire comprendre au plus grand nombre les principaux concepts techniques. La précédente version a été téléchargée plus de 2 000 fois depuis son lancement en 2016 ! Merci à tous les lecteurs de ce livre blanc qui nous ont fait des remarques ou proposé des idées. Votre contribution est précieuse !

Cette version a été améliorée pour préciser les **méthodologies d'un projet**, mettre à jour les principales **stacks techniques** (vous découvrirez ce qui se cache derrière ce terme barbare dans ce document) et rafraîchir la partie mobile.

Il s'adresse à tous les acteurs du digital qui ne maîtrisent pas forcément la technique : product owner, directeur de projet, designer, UX, DSI, responsable innovation. Bref, tous ceux qui souhaitent comprendre ce qui est en jeu dans les projets digitaux des entreprises comme par exemple le développement d'un nouveau projet de plateforme web, une application mobile etc.

SOMMAIRE

INTRODUCTION

Les développeurs ont imaginé un jargon qui est souvent hermétique pour les profanes… Alors si vous êtes en relation avec des développeurs et que vous vous sentez parfois perdus, c'est normal. Chaque métier possède ses propres codes. C'est ce que ce livre blanc vous propose de décrypter.

Nous espérons qu'une fois que vous aurez lu ce document, vous pourrez assister à une réunion où l'on parle de technique sans vous sentir dépassé ! Avec un peu d'effort (et ce document), vous pourrez engager une discussion sans complexe.

1. LES APPROCHES/METHODOLOGIES PROJET

Avant de rentrer dans la technique, détaillons quelques deux approches qui sont de plus en plus répandues et les différentes méthodologies de gestion de projet.

Note : le propos n'est pas de critiquer ou de comparer ces approches/méthodologies mais juste de présenter les concepts et le vocabulaire qui y sont associés.

1.1 Approche « Minimum Viable Product » ou MVP

L'approche **MVP** consiste à réduire le temps entre le démarrage du projet et sa mise sur le marché. Cette approche est souvent associée à une méthodologie agile. Elle consiste principalement à répondre à la question : « Quelles sont les fonctionnalités indispensables à mon produit pour le mettre sur le marché ? ». Ainsi toutes les fonctionnalités accessoires vont être écartées.
L'avantage principal de cette approche est d'accélérer le « time-to-market », de capitaliser sur les retours des premiers utilisateurs (les « early adopters ») et évidemment de réduire le coût du développement de la solution.
La principale critique est que produire une solution parfois incomplète ne permet pas d'avoir une réelle validation du concept. Contre toute attente, ce sont parfois les fonctionnalités accessoires qui font le succès d'une solution.

1.2 Approche « Mobile first »

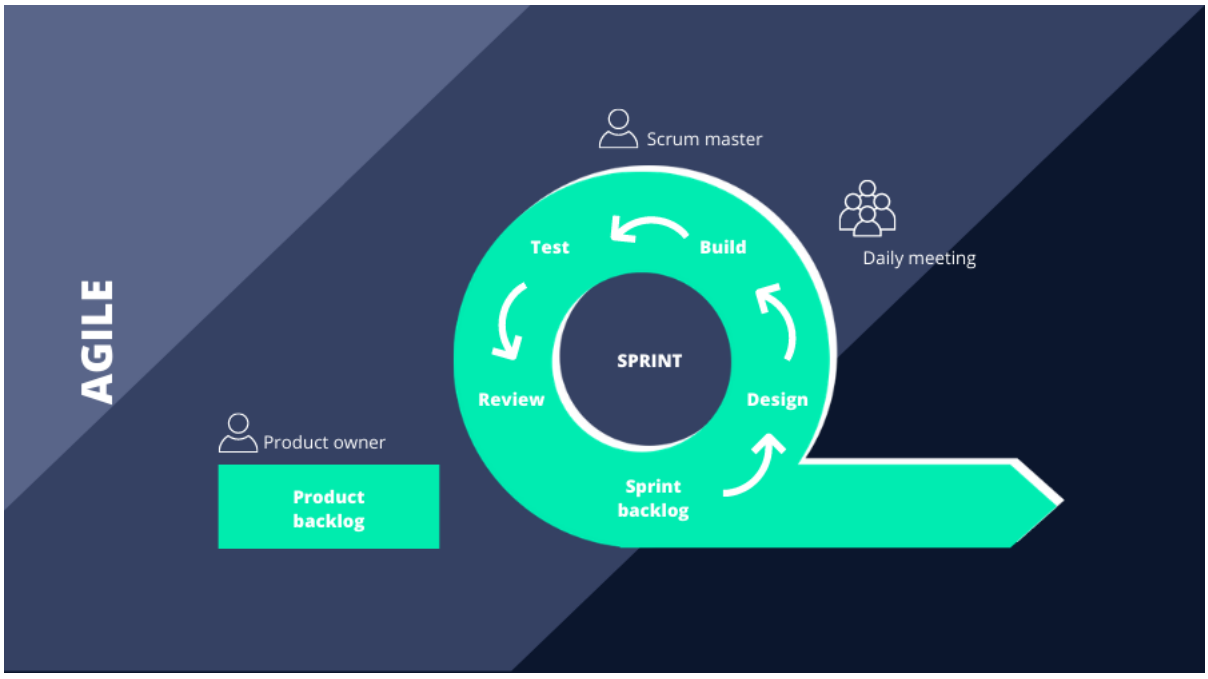
Cette approche vise à concevoir l'application mobile en premier. Un projet web a souvent une déclinaison mobile. Concevoir l'application mobile d'abord permet de simplifier l'application au maximum. Ce qui est assez proche de l'approche MVP.
En effet, sur mobile, l'expérience utilisateur doit être la plus claire possible. Les fonctionnalités « nice-to-have » ne seront pas présentes et les chemins de navigation réduits au minimum. Il y a donc beaucoup de chance que votre application soit plus efficace et que son développement soit plus simple.

1.3 Méthodologie : Agile (SCRUM très souvent)

Mener des développements est une tâche complexe. Il est donc nécessaire de mettre en place une bonne méthodologie pour le faire aboutir. Deux sont assez diffusées : le **cycle en V** ou **waterfall** et la **méthode agile** (très souvent **SCRUM** mais il existe de nombreuses déclinaisons).

La méthode agile propose une approche itérative. Le **Product Owner**, appelé en général « le PO », est le responsable de l'avancement du produit, il définit les fonctionnalités, les correctifs que l'on regroupe dans un « **Product backlog** ».
Le projet est découpé en « sprints » de deux ou trois semaines en général. Il a pour objectif de développer un sous-ensemble d'éléments, le « Sprint backlog ». Puis le sprint comprend l'ensemble des étapes pour produire la solution : conception (design), développement (build), tests et démonstration/rétrospective (review).
Chaque jour, le **Scrum Master** organise une réunion nommée mêlée quotidienne ou encore « **daily scrum**» qui permet de faire le point sur ce qui a été fait et les difficultés rencontrées.

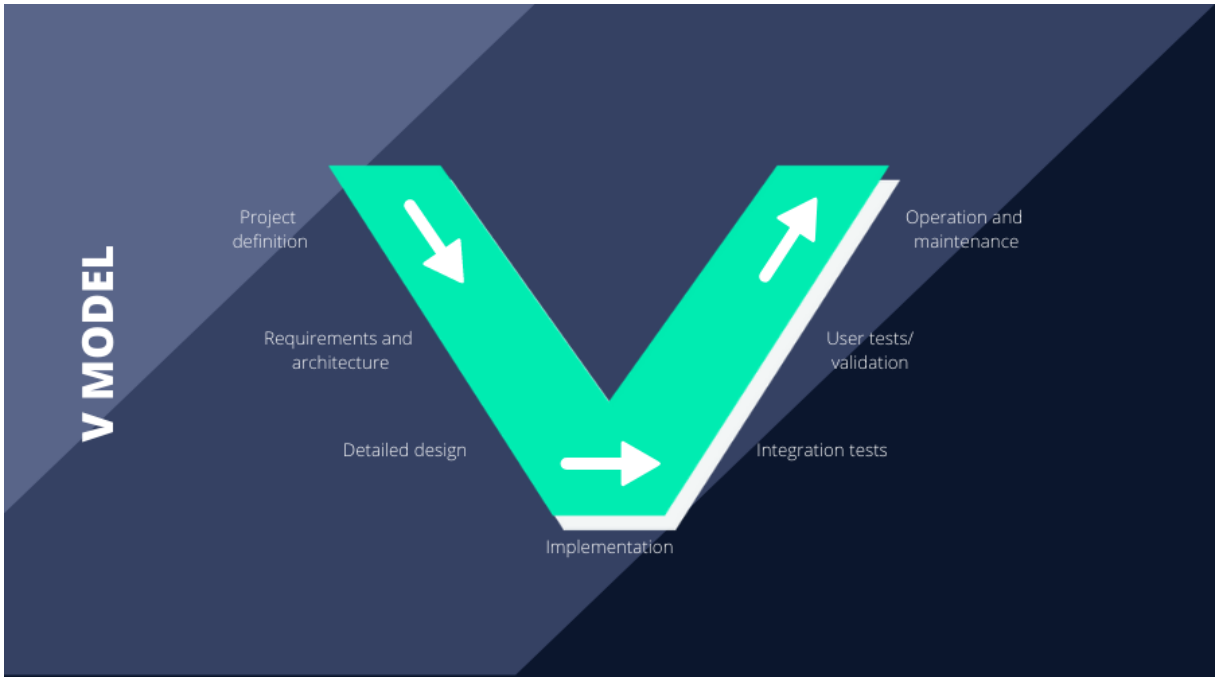
Cette méthodologie est employée lorsque le contour du projet n'est pas forcément bien défini. Il permet de rapidement produire les premiers éléments de la solution ce qui fonctionne plutôt bien avec l'approche MVP.



1.4 Méthodologie : Cycle en V (Waterfall)

Le cycle en V propose une approche globale. Au départ, on définit le projet complet et donc le périmètre précis de l'application. Chaque étape de la descente du V permet de définir de plus en plus finement l'application jusqu'à aboutir aux développements. Chaque étape de la remontée du V permet de passer des cycles de tests de plus en plus complets jusqu'à la mise en production.

Cette méthodologie est employée lorsque l'on sait ce à quoi il faut aboutir. Elle présente l'avantage de permettre une estimation fine des délais et des budgets. Lorsqu'il y a un prestataire en jeu, cela permet de l'engager au forfait (c'est-à-dire pour un prix fixé à l'avance). L'inconvénient est le temps de développement puisqu'il faut tout faire d'un coup, on parle alors « d'effet tunnel ».

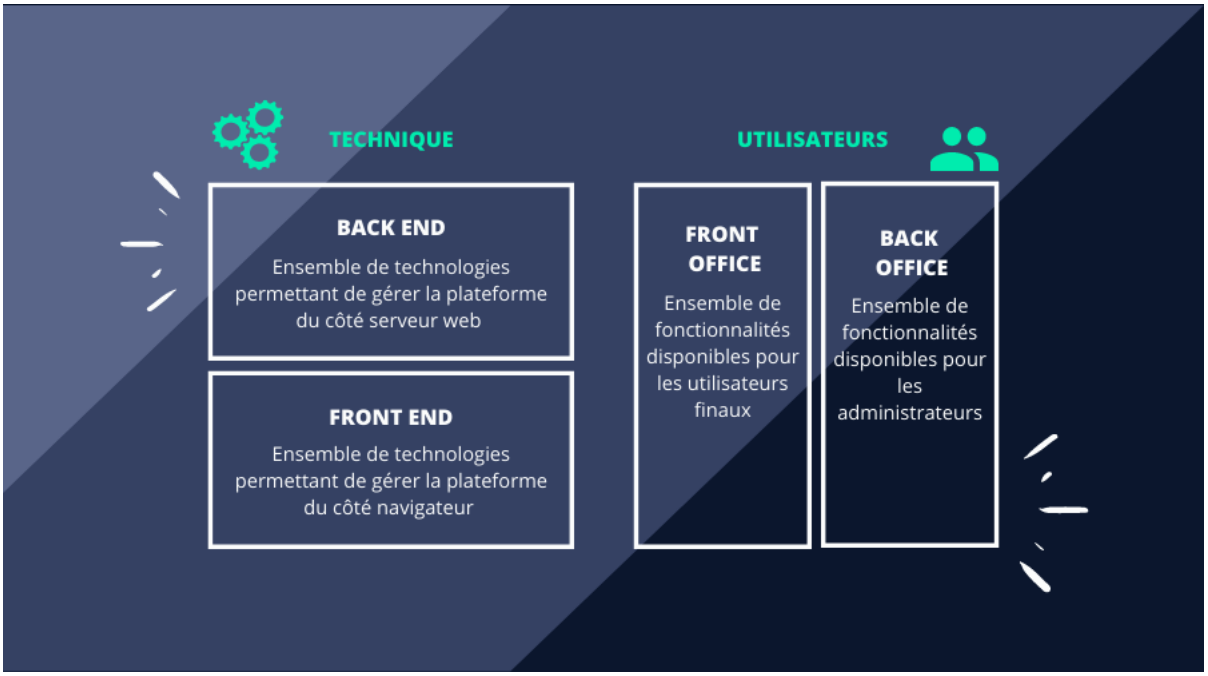


2. QUELQUES CONCEPTS CLÉS D'ARCHITECTURE TECHNIQUE

2.1 Front/Back : un piège pour les non-initiés

Attention au piège ! On peut parler de développement front-end ou back-end en évoquant la technologie respectivement sur le navigateur ou sur le serveur web.

Mais on peut aussi parler de front-office ou de back-office. Et, dans ce cas, on distingue les populations d'utilisateurs : le front-office adressant les visiteurs d'un site et le back-office les gestionnaires.



2.2 Quelle est la différence entre un framework et une librairie ?

Un framework n'est pas une librairie ! Si l'amalgame est assez facile à faire entre ces deux termes, c'est tout simplement parce que dans la plupart des cas, un framework inclut une ou plusieurs librairies.

Ainsi, si une librairie peut être comparée à un ensemble de fonctionnalités, un framework quant à lui peut être perçu comme la structure complète d'un projet ! Ainsi le développeur va appeler une librairie pour disposer de fonctionnalités particulières. Par exemple, notre librairie Open Source Gotenberg permet de transformer n'importe quelle page HTML en

document PDF. Inversement, un framework va permettre de structurer le code pour le développeur en gérant de nombreux aspects tels que la sécurité par exemple.

La différence entre un framework et une librairie est donc appelée « Inversion of Control (IOC) » ce qui signifie de manière concrète qu'un framework « contrôle » le code d'un développeur alors qu'une librairie est « contrôlée » par le code d'un développeur. Parmi les frameworks les plus connus on retrouve : Symfony ou Laravel (PHP), Angular (Javascript), Django (Python), Ruby on Rails (Ruby), ...

Et cela marche pour tous les langages de programmation !

2.3 Architecture client/serveur, 3 tiers

On oppose souvent les architectures client léger - ou architecture 3 (N) tiers - aux architectures client-serveur - ou client lourd.

On parle de client lourd lorsque le matériel de l'utilisateur est utilisé pour les traitements tandis que l'on parle de client léger lorsque l'ensemble des traitements est effectué à distance (sur un serveur web par exemple). Il y a fort fort longtemps, les débits réseaux ainsi que les ressources serveurs étaient faibles (dans les années 80-90). Une partie des traitements était donc déportée vers les clients (PC des utilisateurs). Depuis, avec l'amélioration des capacités des navigateurs et des connexions internet, les architectures client léger se sont imposées.

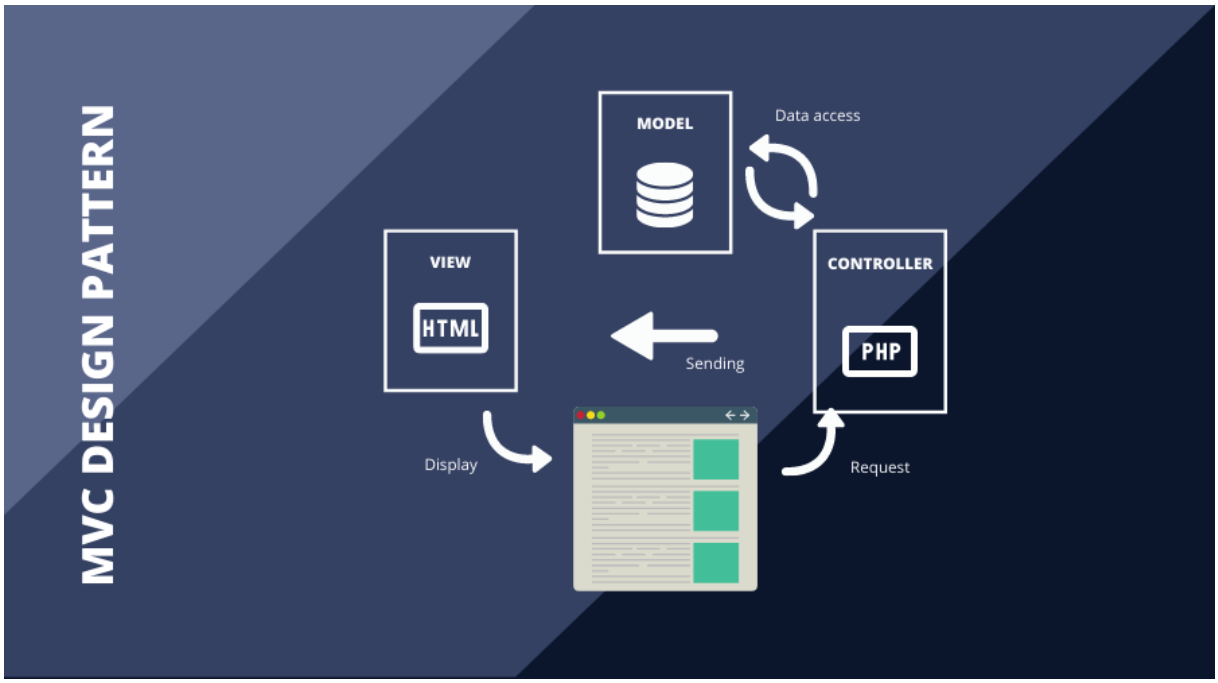
Aujourd'hui, on revient un peu en arrière avec des traitements qui sont effectués sur les devices (le matériel) des utilisateurs en utilisant les navigateurs ou les OS des mobiles. On cherche à améliorer les performances ou pallier une déficience éventuelle de réseau. Parfois, par extension ou facilité, on parle aussi d'architecture 3 tiers pour évoquer les différents composants physiques de la solution : le terminal (navigateur web ou téléphone), le serveur web et le serveur de données.

2.4 Le design pattern MVC

Il y a des problèmes en programmation qui reviennent tellement souvent qu'on a créé des bonnes pratiques (qui résolvent ces problèmes) que l'on a réunies sous le nom de design pattern. Le **design pattern MVC** ou Model-View-Controller (Modèle-Vue-Contrôleur) est l'un des plus importants. Il s'agit d'un pattern qui sépare la logique du code en trois parties afin de clarifier la conception des développements :

- Le Contrôleur gère l'enchainement des pages, les URL... (le code PHP qui interroge le modèle et renvoie les éléments à afficher à la vue)
- Le Modèle gère la logique métier et les données (les requêtes SQL)

- La Vue affiche la page (le code HTML et quelques boucles et conditions PHP très simples, pour afficher par exemple des listes)



3. LES STACKS LES PLUS CLASSIQUES

3.1 Les technologies front-end : HTML, CSS et JavaScript

Le format HTML permet de décrire la structure des pages Web. Il indique la nature des éléments d'une page (lien, bouton, etc.) et l'organisation des blocs. La version 5 a apporté beaucoup de nouveautés pour définir les principaux éléments de la page, intégrer des médias (vidéo, audio) ou fournir des éléments utiles (saisie de dates ou de nombres).

CSS est le langage qui permet de préciser la mise en forme des balises HTML. Par exemple la police d'affichage, la taille ou la couleur du texte, les images de fond… La version 3 permet notamment de gérer les animations et les transitions d'état de manière standardisée. On peut même adapter le rendu d'une même page à la taille des écrans (ordinateur, tablette, smartphone) grâce aux media queries. C'est le Responsive Design.

JavaScript servait à l'origine de gadget pour animer les pages web. Mais, au fur et à mesure du temps, son utilisation s'est généralisée et des frameworks de plus en plus modernes sont apparus comme par exemple **Angular**, **React** ou **Vue**. Ces frameworks permettent désormais d'apporter un aspect applicatif à la partie front : gestion des formulaires, des templates et même le routage (pour une single-page application).

3.2 Les technologies Back-end

Note : nous présentons les stacks que nous maîtrisons le mieux (PHP et Node.js) mais il en existe beaucoup d'autres : Ruby ou Python par exemple.

PHP :

Il s'agit sans doute de la technologie la plus utilisée sur Internet ! PHP, langage de programmation coté serveur, traite les requêtes des utilisateurs. Il s'interface avec une base de données relationnelle comme MySQL pour lire ou stocker de l'information. PHP peut générer ensuite le HTML qui est affiché en retour (c'est moins vrai avec l'utilisation de manière intensive des frameworks JavaScript). La plupart du temps, le serveur web Apache (ou Nginx) assure la transmission des requêtes.

Il existe aujourd'hui un très grand nombre de librairies PHP qui permettent de développer rapidement les fonctionnalités récurrentes des applications web :

- La gestion des utilisateurs et des droits,
- Le routeur qui prend en charge les différentes URL de l'application,

- L'implémentation native du MVC, appelé scaffolding, qui génère le squelette du code,
- La gestion simplifiée des accès aux bases de données, appelé ORM,
- La Génération automatique de listes ou de formulaires …

Les frameworks PHP les plus répandus, comme **Symfony** et **Laravel** comprennent tous ces composants assemblés de manière cohérente.

Si vous entendez parler d'une architecture **LAMP*** - ou stack LAMP, pas de panique ! Il s'agit simplement de l'acronyme Linux, Apache, MySQL, PHP.

JAVASCRIPT (NODE.JS) :

On pensait avoir trouvé une architecture universelle : un navigateur, un serveur PHP, des bases de données MySQL. Hé ben non !

Parfois, ce type d'architecture ne suffit pas… le temps réel ou le volume des traitements nécessite de nouvelles architectures. **Node.Js** permet par exemple de développer des applications très scalables (c'est-à-dire qui montent en charge rapidement). Le mécanisme asynchrone permet de gérer facilement les accès simultanés. Par ailleurs, la persistance des données est assurée par une base de données non relationnelle comme MongoDB ou Cassandra. Ce standard porte l'acronyme de stack MEAN (Mongo, Express, Angular, Node).

3.3 Les autres solutions : propriétaires ou Open Source

Souvent, une solution prête à l'emploi correspond au besoin ! Ces solutions permettent de mettre en place très rapidement et souvent à moindre coût votre application pourvu que vous restiez dans le standard. Pour citer les acronymes les plus connus : ERP, CMS, CRM ou encore e-Commerce.

Ces solutions peuvent être propriétaires, c'est-à-dire proposées par un éditeur, ou Open Source qu'elles passent par un éditeur ou non. Impossible de vous dire quelles sont les bonnes ou les mauvaises solutions. Ce n'est pas le propos de ce livre blanc et les besoins sont trop divers.

Ces solutions embarquent une architecture technique particulière. Il est donc impossible de vous dire précisément la stack technique qui est associée. Mais vous trouverez en général beaucoup de littératures sur le sujet. La question est de déterminer ce qu'il reste à développer. Ayez à l'esprit que ces développements seront souvent un peu complexes à réaliser car ils devront se plier aux contraintes de la solution que vous aurez choisie.

4. LES DIFFÉRENTES BASES DE DONNEES

La question qui se pose ne concerne pas les caractéristiques des sauvegardes de données, mais plutôt la manière dont on les sauvegarde ! Il y a quelques années, avec l'utilisation de bases de données relationnelles comme Oracle, cette question n'avait pas lieu d'être soulevée. Aujourd'hui, il est souvent souhaitable de conserver de nombreuses informations qui étaient considérées comme accessoires comme celles liées au comportement du visiteur par exemple. Les quantités de données explosent et les technologies de stockage de données ont donc évolué.

4.1 La référence SQL

Le **modèle relationnel** (**SQL** signifie Structured Query Langguage) est une manière de modéliser les relations existantes entre plusieurs informations, et de les ordonner entre elles. Des tables à deux dimensions (comme une fichier Excel) représentent les données. Par exemple, la table Livre contiendra les colonnes : titre, auteur, éditeur…
Et puis, on parle de relations entre ces tables : la table Bibliothèque contiendra plusieurs éléments de la table Livre (on parlera de relation 1:N).
Citons pour les plus connues : Oracle, SQL server, **MySQl** ou encore **PostgreSQL**.

4.2 Logique « not only SQL »

Malheureusement, le modèle SQL (modèle relationnel) n'est pas bien adapté aux très grands volumes. Les temps de traitement sont trop longs. Aussi, apparaît le **NoSQL** qui signifie "not only SQL". Le NoSQL regroupe de nombreuses bases de données, récentes pour la plupart, qui se différencient du modèle SQL par une logique de représentation de données non relationnelles. Cette logique a le double avantage d'augmenter les performances et de permettre le traitement de très grands volumes de données.

Ainsi, dans les projets, il ne faut pas opposer ces deux approches mais bien souvent les faire cohabiter ! Cette technologie NoSQL ne vise finalement pas à remplacer les bases de données traditionnelles mais plutôt à les compléter en déportant une partie de la charge. Parmi les plus populaires : **MongoDB**, **couchDB** ou encore **Redis**.

Note : Le big data est en plein développement. Des outils comme **Hadoop**, **Spark** (frameworks pour le développement d'applications distribuées) ou encore **Kafka** (gestion des flux de données) permettent de gérer ces grandes quantités de données (mais là, le sujet devient trop pointu par rapport à l'objectif de ce document).

5. API – ÉCHANGES DE DONNÉES

Les échanges entre les applications sont de plus en plus nombreux. Par exemple, un site web va permettre à un visiteur de se créer un compte qui sera ensuite intégré dans un CRM, Customer Relationship Management. S'il effectue une commande, cette commande va être intégrée dans un ERP - Entreprise Ressource Planning - et le paiement va se faire grâce à une intégration avec une banque. On parle alors de Web Service ou d'API (Application Programming Interface) pour qualifier l'interface qui orchestre ces échanges.

5.1 Les différents types d'API

Il existe deux grands protocoles de communication sur lesquels s'adossent les API : Simple Object Access Protocol (SOAP) et Representational State Transfer (REST). Le second s'est désormais largement imposé face au premier car il est plus flexible. Il a donné naissance aux API dites **REST** ou RESTful.
Récemment, un troisième protocole est de plus en plus populaire. Il s'agit de **GraphQL**. L'avantage principal est que le client (le navigateur) décrit la réponse qu'il souhaite recevoir. Cela évite au serveur de retourner trop d'information.

5.2 Les différents formats d'échange de données

Il y a quelques années les formats d'échanges étaient fixes sur un certain nombre de caractères : 50 pour le prénom, 50 pour le nom, 10 pour le numéro de téléphone et ainsi de suite… Ce n'était pas un système très souple !

Un nouveau format a été inventé où les différentes informations du message étaient séparées par un caractère spécial, comme un point-virgule par exemple. Ce format est parfois utilisé, il s'agit par exemple du format **CSV** (Comma-separated value) qui est sous Excel. Toutefois, à chaque nouvelle donnée ajoutée, il fallait redévelopper le message.

Aujourd'hui, les formats d'échanges sont une suite de couples variable-valeur, potentiellement récursives comme le format **XML** (Extensible Markup Language) ou **JSON** (JavaScript Object Notation). Extrêmement bien structuré et possédant de très nombreuses fonctionnalités, le format XML a tendance à céder la place à son alter-égo JSON, qui est moins normé mais nettement plus simple d'utilisation, et nativement compatible avec les navigateurs.

6. TECHNOLOGIES MOBILES

Pour développer une application mobile vous avez le choix des technologies : le natif, les technologies cross-platform ou des technologies plus proches du développement web (PWA). Chacun de ces choix présente des avantages et des inconvénients !

6.1 Les applications natives

Les développements d'applications mobiles dépendent étroitement des systèmes d'exploitation (OS) des terminaux mobiles :

- **Java** ou Kotlyn pour **Android** ;
- Objective-C ou **Swift** pour **Apple**.

Le développement natif présente de nombreux avantages comme par exemple des meilleures performances ou probablement une meilleure gestion des interfaces utilisateurs. D'autres éléments (complétement étrangères aux technologies) sont à prendre en compte comme la validation des applications dans les stores.

6.2 Les technologies cross-platform (React-Native et Flutter)

Certains clients ne souhaitent pas développer une application pour chaque OS (sauf si l'on fait quelque chose de très particulier comme un jeu par exemple) car les coûts sont doublés. Aussi, certains ont eu la bonne idée de développer des technologies qui permettent de compiler un seul code (JavaScript) pour les deux plateformes phares (Android et Apple). On parle alors de technologie cross-platform comme par exemple : **React-Native** (développé par Facebook) et **Flutter** (par Google).

6.3 Progressive Web Application (PWA), une solution alternative ?

Les **PWA** (Progressive Web Application) sont des applications web qui se rapprochent du comportement des applications mobiles. C'est un cran au-dessus du **responsive design** qui vise à adapter le design d'un site à une navigation mobile. Une PWA implémente parfois des fonctionnalités telles que : l'accès hors-ligne, les notifications (push), la synchronisation de tâche de fond ou encore la géolocalisation.

L'inconvénient majeur est qu'une PWA doit être téléchargée et installée depuis le site web. Elle ne pourra pas être présente sur le store d'Apple par exemple.

7. UN PEU PLUS DE TECHNIQUE

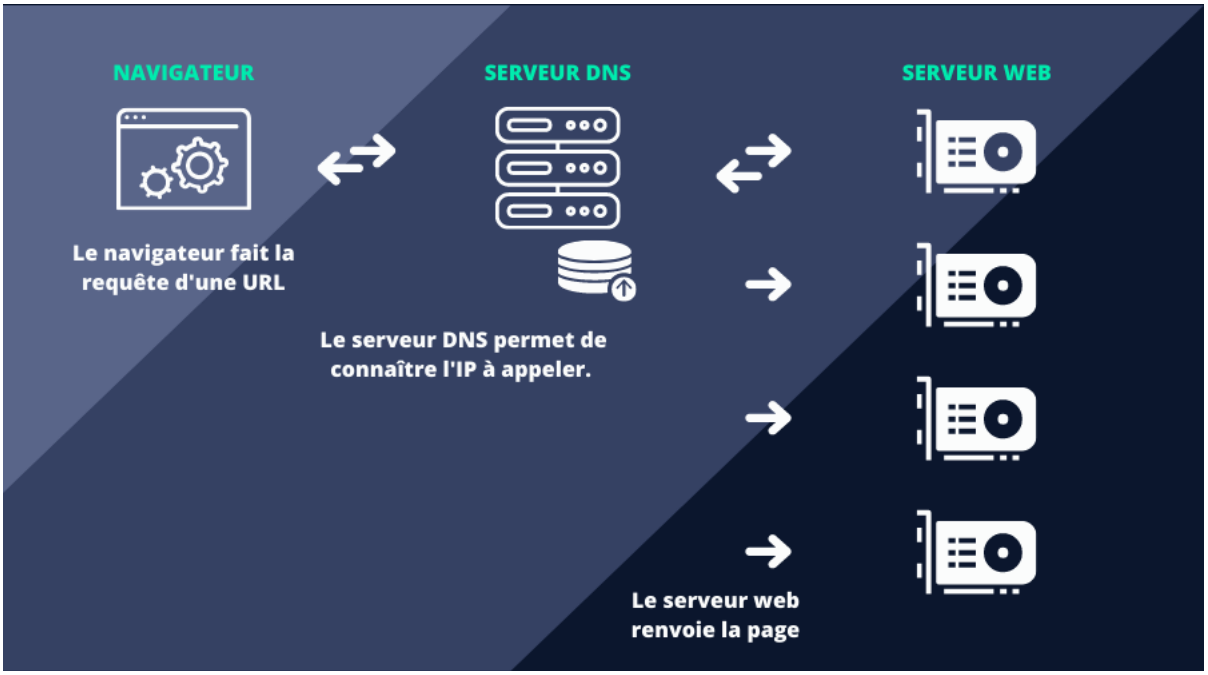
7.1 À l'origine du web : TCP/IP et DNS

Au premier jour, Dieu créa le protocole TCP/IP.
Et puis, comme c'était trop compliqué de retenir toutes les adresses IP sur le navigateur, Dieu dit : « Que le DNS soit ! ». Et le DNS fut.
Dieu vit que le DNS était bon. Les requêtes http s'échangeaient bien, ce fut Internet.

Quand un internaute tape l'adresse d'un site web, le serveur **DNS** (Domain Name System) identifie le nom de domaine et renvoie vers la bonne adresse IP (adresse physique de la machine). Dans la plupart des cas, les serveurs physiques, destinés à héberger un serveur web, ont une IP fixe et publique (contrairement au réseau de votre entreprise qui vous affecte une IP locale aléatoirement dans une plage d'IP).

De manière très simplifiée, les serveurs DNS s'échangent des informations afin de fournir la réponse. Le serveur DNS est généralement la propriété d'un hébergeur de nom de domaine. C'est auprès de celui-ci que vous allez acheter votre nom de domaine et le faire pointer vers l'adresse IP de votre site web. Une fois que l'adresse IP est résolue auprès du DNS, la connexion est établie avec les serveurs web qui peuvent alors échanger des requêtes http, c'est-à-dire les pages de votre site.

Dans les projets, il est possible que l'on parle de ports TCP. Ce sont les terminaisons qui « écoutent » certaines applications, par exemple, les requêtes http sont affectées au port 80 et les requêtes https au port 443.



7.2 La gestion du code source (GIT)

Le but de cet outil est de permettre de garder un historique de toutes les modifications effectuées au code source, d'aider à travailler en équipe et de gérer les versions. SVN désormais un peu vieillissant, a été supplanté par **GIT**, un logiciel de gestion de versions décentralisé.

Vous pourrez entendre : « Il suffit de forker le projet pour commencer à travailler dessus et ensuite faire un pull request ». Ce qui signifie : « faire une copie du code source, travailler dessus et proposer à celui qui est responsable du code dans son ensemble de relire/valider son code ».

7.3 Les serveurs, le hosting

Les serveurs - ou l'hébergement - en général a aussi un vocabulaire qui lui est propre. On parle de Cloud lorsqu'on ne se préoccupe pas du matériel qui est derrière. Vous entendrez parler d'**AWS** pour Amazon Web Service ou de **GCP** pour Google Cloud Platform.
On parle de serveur dédié lorsqu'on loue ou achète sa machine qui ne sert que ses besoins.

Entre ces deux extrêmes, toutes les solutions sont possibles ! Serveur Mutualisé lorsque l'on partage un serveur avec d'autres clients ou VPS (Virtual Private Server) qui a toutes les caractéristiques d'un serveur dédié mais qui est en fait un serveur virtuel !

7.4 La sécurité

On ne parlera ici que des éléments les plus connus ou les plus critiques.
L'attaque par déni de service (DoS ou **DDoS** en Anglais) est certainement une des attaques les plus médiatisées. Elle consiste à empêcher l'accès à un service en envoyant un nombre de requêtes très important.

Il existe également des failles ou des vulnérabilités. Des mises à jour sont régulièrement publiées et vous devez les mettre en œuvre ! Les principaux défauts associés à un code sont les **injections SQL** (on peut injecter dans un champ de formulaire des requêtes SQL) ou les failles **XSS** (là, on injecte du code JavaScript).

D'autres types de failles sont référencées : CSRF, sensitive data exposure ... Celles-ci sont maintenues à jour dans un document publié par l'OWASP (Open Web Application Security Project) qui fait référence en la matière.

7.5 La gestion du cache

C'est une des problématiques les plus complexes des applications web. Le cache permet de stocker quelque part le résultat d'un calcul complexe pour éviter d'avoir à le refaire plus tard. Il peut s'agir d'éléments très divers, comme les droits associés à un utilisateur, ou une page HTML.

Le résultat du cache peut être stocké côté client (c'est le cache navigateur qui héberge les fichiers CSS, JS et les images), ou côté serveur. Le cache navigateur ne nécessite pas de technologie particulière, seulement de la configuration du serveur Web (Apache, NodeJs, etc.).

En revanche, côté serveur, plusieurs solutions existent : **APC**, **Memcached** et **Redis**. Montés en mémoire, ils sont très rapides d'accès et stockent des données. Le serveur de cache HTTP Varnish stocke des requêtes web : il stocke lui-même les ressources statiques (fichier CSS, JS, images ou HTML) et répond très rapidement pour les servir. Cela permet de décharger le serveur applicatif d'appels « inutiles ».

La complexité du cache est liée à son invalidation : savoir précisément quand ne pas l'utiliser est un art maîtrisé par peu de personnes !

7.6 Les moteurs d'indexation

Le but d'un moteur d'indexation est, logiquement, d'indexer des informations : cela signifie qu'il classe les données ou le contenu de documents pour permettre de retrouver très rapidement un ou plusieurs contenus.

Leur nom varie (Apache **SolR**, **Lucene**, **Elasticsearch**, etc.), mais on les utilise presque toujours pour développer des moteurs de recherche performants :

- Trouver une information dans un très grand nombre de données
- Faire des recherches approximatives (tolérance à la faute de frappe)
- Faire des recherches à l'intérieur de documents texte

CONCLUSION

La technologie est un sujet fascinant. Elle évolue, se transforme, se remet en cause. Chaque ligne de ce document pourrait faire l'objet d'un débat de spécialistes ou d'une page complète d'explication !

Nous savons bien que le sujet est un peu difficile et nous espérons être parvenu à le rendre compréhensible.

Alors, "Do you speak technique" maintenant ? N'hésitez pas à nous le dire !

EN COMPLÉMENT / PROCHAINEMENT

N'hésitez pas à télécharger notre livre blanc **MODÈLE DE CAHIER DES CHARGES PLATEFORMES WEB** qui vous donnera les clés pour comprendre tous les aspects d'un projet digital.

Et, notre série **COMPRENDRE LA TECH** va se poursuivre, avec en cours de rédaction : **COMPRENDRE LES TENDANCES TECHNIQUES 2021**

Si vous voulez recevoir ces documents, n'hésitez pas à nous écrire un email contact@thecodingmachine.com ou via les réseaux sociaux

MERCI !

Si vous avez un projet
de plateforme web, une application métier ou une
application mobile n'hésitez pas à
passer nous voir pour en discuter.

TheCodingMachine
56 rue de Londres – 75008 – Paris
01.85.08.34.98