# livre blanc
# FRONT-END

MAKINA CORPUS

# Livre blanc du développement front-end

Makina Corpus

# Préface

La popularité de JavaScript dans le développement web ne cesse d'augmenter.

La partie cliente (front end) d'une application moderne ne se développe cependant pas en JavaScript pur ou avec jQuery mais avec de nombreux outils améliorant la productivité.

Des dizaines de frameworks, bibliothèques et bonnes pratiques existent et il en apparaît de nouveaux chaque semaine.

Ces innovations perpétuelles donnent un sentiment d'instabilité, de foisonnement inextricable.

La remise en cause continue de l'état de l'art a fait apparaître l'expression « JavaScript Fatigue » : il est facile de s'épuiser à déterminer les bonnes pratiques et les éléments à utiliser pour conduire un projet.

Miser sur le mauvais framework peut vous obliger à redévelopper entièrement votre application quelques mois plus tard lorsque ce framework si prometteur aura été abandonné par sa communauté ou ne tiendra pas ses promesses.

C'est pour vous aider à mieux comprendre le contexte du développement front-end moderne que nos experts ont rédigé le présent livre blanc.

Nous souhaitons apporter un éclairage sur l'écosystème JavaScript qui vous permettra de mener à bien tous les types de projets Web front-end, de la SPA (Single Page Application) à l'application hybride, en passant par les Web Apps plus classiques ou encore les Progressive Web Apps.

Pour construire ce livre blanc, nous nous sommes appuyés sur l'expérience acquise au cours de dizaines de projets Web et sur les évaluations systématiques auxquelles nous procédons pour nos outils.

Société de services du numérique, Makina Corpus propose une approche globale incluant l'analyse des besoins, la conception des interactions, le développement d'applications toujours à la recherche de bonnes performances techniques et d'une sécurité optimale.

Experts en logiciels libres, nous contribuons aux projets communautaires que nous utilisons et publions nos propres développements : http://github.com/makinacorpus/.

Nous offrons également une gamme complète de formations ajustables selon vos besoins ainsi que des audits personnalisés.

Nous espérons que ce livre blanc vous guidera dans la compréhension des enjeux et problématiques liées au développement front-end, ainsi que dans la prise de décisions techniques.

N'hésitez pas à nous contacter si vous souhaitez des conseils ou de l'aide dans la réalisation de votre projet.

Bonne lecture.

# Edito

Dans notre entourage professionnel, nous rencontrons fréquemment des personnes un peu perdues lorsqu'il s'agit de parler du développement front-end. Que l'on pratique le PHP, Python, Java ou C# ou encore l'intégration CSS, le « dev front-end » peut paraître difficile d'accès, voire complexe.

## POURQUOI CE LIVRE ?

Le très fort dynamisme de l'écosystème JavaScript génère des frustrations et le rend parfois peu abordable. Certains commentaires ou titres d'articles sont assez explicites : « JavaScript Fatigue », « Yet Another JavaScript framework », etc.

> « This » en JavaScript est tellement puissant et souple que je n'arrive pas à comprendre comment le faire fonctionner.
>
> — **Kent Beck, inventeur de l'eXtreme Programming**

Pour autant, on ne peut pas laisser de côté ce qui est aujourd'hui un outil fondamental du développement Web.

Nous espérons donc que ce livre vous aidera à mieux appréhender les enjeux, les outils et les tendances du développement front-end.

Ce livre présente les méthodes et les outils qui sont à votre disposition pour réaliser des développements front-end de qualité. Nous ne parlons ici ni d'intégration ni d'ergonomie.

Nous allons d'abord définir les termes qui nous paraissent indispensables à la compréhension du livre blanc. Nous abordons les enjeux du développement front-end, pour mieux comprendre ce que cela implique sur les plans technique, méthodologique et humain. Puis nous détaillons l'ensemble des principaux frameworks JavaScript afin de mieux comprendre leurs différences et leurs cas d'utilisation. Nous levons ensuite le voile sur les besoins de l'environnement technique du développeur JavaScript. Pour terminer, un tour d'horizon du monde mobile et ses spécificités sera proposé.

Nous vous souhaitons une bonne lecture et espérons que ce livre blanc saura répondre à vos interrogations.

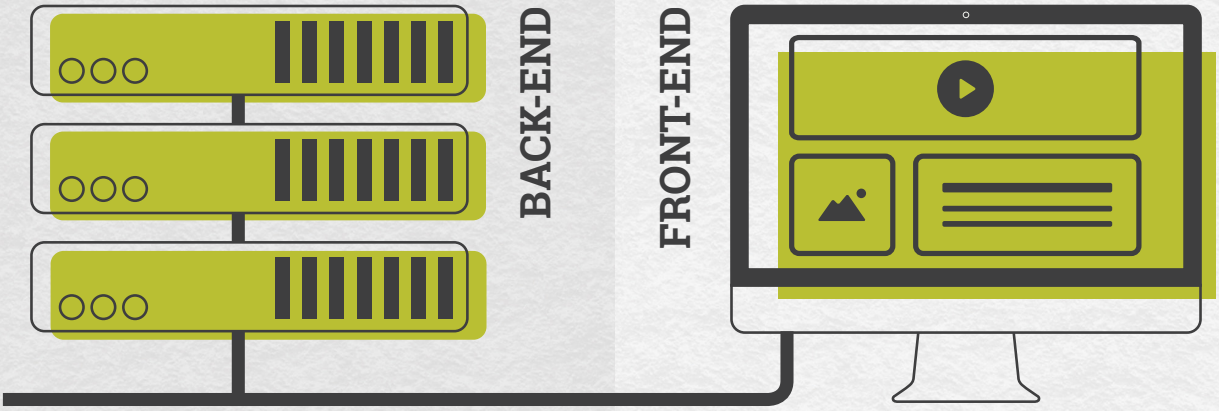
# Les principales définitions

Afin de bien appréhender ce livre, nous proposons les définitions des termes principaux, contextualisés au cas du développement front-end.

Beaucoup de termes anglais sont utilisés dans le monde de l'informatique. Les faux-amis sont légions et il nous arrive même de modifier le sens d'un mot pour l'adapter à notre culture.

Dans le développement Web, nous distinguerons deux grandes familles : le **back-end** et le **front-end**. Le back-end rassemble tout ce qui est déployé et servi par un serveur d'applications (par exemple jetty ou tomcat) ou un serveur HTTP (par exemple nginx ou caddy). Le front-end, quant à lui, rassemble tout ce qui s'exécute dans le navigateur de l'utilisateur.

Le **développement front-end** à proprement parler gravite autour du JavaScript ; il est une des spécialités du monde front-end parmi le graphisme, l'ergonomie, le développement et l'intégration.




*Les deux faces d'une même médaille : back-end, front-end*

D'un autre côté, **l'intégration** est la maîtrise de la mise en page et de l'accessibilité de l'application. Cela englobe notamment l'écriture du HTML et du CSS, ainsi que l'accessibilité pour, entre autres, les différentes formes de consultations (par exemple sur mobile) ou de déficiences (par exemple utilisant un lecteur d'écran).

Dans cet ouvrage, nous évoquons également le **Web mobile**. Lorsque nous parlons de plateformes mobiles, nous englobons à la fois les smartphones et les tablettes. Ces plateformes ont deux spécificités par rapport au classique couple ordinateur et navigateur : elles s'utilisent en mobilité, ajoutant une problématique d'accès au réseau Internet ; elles disposent d'un écran tactile sans clavier physique (pour la plupart).

D'ailleurs le développement front-end englobe également ce que nous appelons **applications hybrides**. Développées avec les technologies du Web, elles sont disponibles sur les plateformes mobiles à l'intérieur d'une WebView, c'est-à-dire dans un navigateur en plein écran.

De manière plus générale, nous parlons indistinctement d'**application front-end**, de **Web App**[1] ou de site Web interactif.

Le ***framework*** est un ensemble de bibliothèques (library) et d'outils donnant un cadre au développement d'une application. Les concepts logiciels sont souvent bien délimités et on retrouve souvent dans un framework l'implémentation, plus ou moins stricte, d'une architecture logicielle.

1_ Web App : application s'exécutant dans un navigateur web.

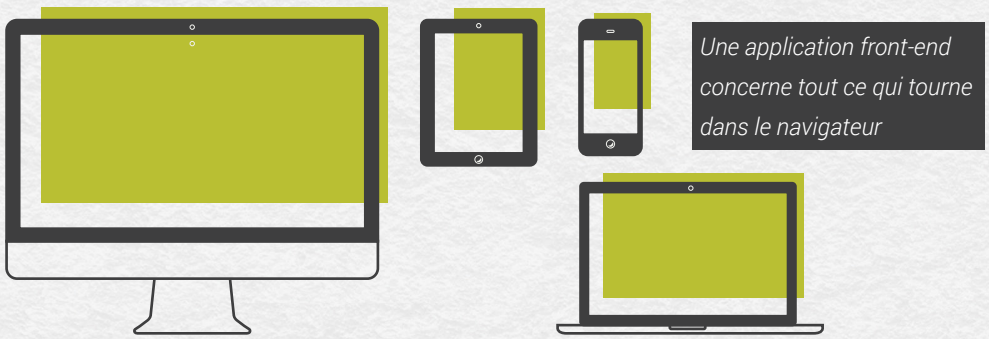
# Comprendre les enjeux du développement front-end

Le développement front-end n'est pas anodin. Son existence est plutôt récente dans l'histoire de l'informatique et son écosystème est si actif qu'il n'est pas rare de voir des bibliothèques dépréciées au bout d'une année d'existence.

Une application front-end fonctionne dans un navigateur Web, il est important de bien comprendre ce que cela implique.

Nous verrons également que le découpage d'une application en deux parties – l'une front-end, l'autre back-end – est déterminant, tant d'un point de vue technique que dans l'organisation des équipes de développement.

Enfin, lorsqu'on parle d'application front-end, on parle forcément de frameworks. Faut-il en utiliser ? Créer son framework maison ? Nous répondrons à ces interrogations dans la dernière partie de ce chapitre.



*Une application front-end concerne tout ce qui tourne dans le navigateur*

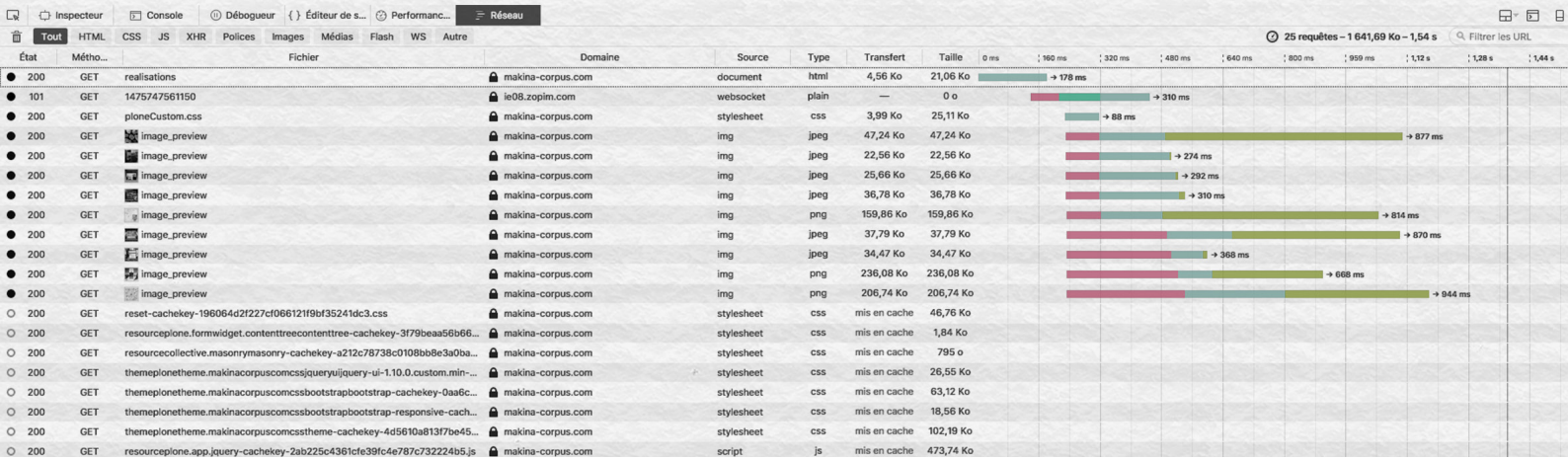
# LE FONCTIONNEMENT D'UNE APPLICATION FRONT-END

Lorsqu'un utilisateur ouvre son navigateur Internet et qu'il se rend sur une page Web, il est déjà en train d'utiliser une application front-end.

Les différents langages que le navigateur va interpréter[2] sont HTML, CSS et JavaScript. À partir du moment où notre application fournit du HTML, CSS et JavaScript, le navigateur est mis à contribution pour interpréter ce code, l'application front-end prend alors vie.

Afin de mieux appréhender le fonctionnement d'une application front-end, il est important de bien comprendre les actions effectuées par le navigateur web pour afficher une page.

*Différentes étapes lors du chargement d'une page Web*




## Le navigateur Web

Tous les navigateurs vont fonctionner de la même façon en suivant ces différentes étapes :

1. téléchargement de la page HTML ;
2. lecture et analyse du contenu ;
3. téléchargement des fichiers JS, CSS et images liés en fonction de leur position dans le HTML ;
4. lecture et analyse des fichiers JS et CSS.

Notez que les divers téléchargements peuvent s'effectuer de manière synchrone ou asynchrone.

2_ interpréter : tâche consistant à analyser, traduire et exécuter les programmes écrits dans un langage informatique.

Lorsque l'utilisateur change de page HTML via un bouton ou un lien, le même processus se lance pour cette nouvelle page. Si certaines ressources peuvent être mises en cache[3], la vitesse d'affichage de la page sera plus rapide.

Il faut savoir que l'exécution des instructions HTML est top-down[4] et séquentielle. L'exécution des fichiers JS est également séquentielle. C'est la raison pour laquelle lorsqu'une page charge un fichier JS, le navigateur suspend le rendu HTML (syndrôme de la page blanche au chargement). On dit que le JavaScript est bloquant.

En revanche, les fichiers CSS peuvent être téléchargés et exécutés simultanément car toutes les règles CSS sont de toute façon toujours appliquées. Ce qui rend le CSS non bloquant.

C'est dans ce contexte qu'est exécutée notre application front-end.

## La maîtrise de l'environnement

La plus grande différence entre un développement front-end et back-end se situe sur la maîtrise de l'environnement d'exécution.

**Un développeur back-end**, s'il ne choisit pas lui-même son environnement, peut néanmoins le prévoir. Il sait, avant de livrer et de déployer son code sur le serveur, quel OS[5] et quelle mémoire vive sont à sa disposition.

**Un développeur front-end** ne peut pas savoir si l'utilisateur interprète son code sur tel OS, avec tel navigateur, sur telle plateforme, mobile ou fixe. Ni si le navigateur autorise JavaScript, ou si le code aura accès aux Web API (géolocalisation, base de données locales, etc.). Dans le monde mobile, on ne sait pas non plus prévoir si l'application sera utilisée en ligne, hors ligne ou en ligne dégradé (ie. avec une faible connexion du type GSM Edge).

## L'expérience utilisateur avant tout

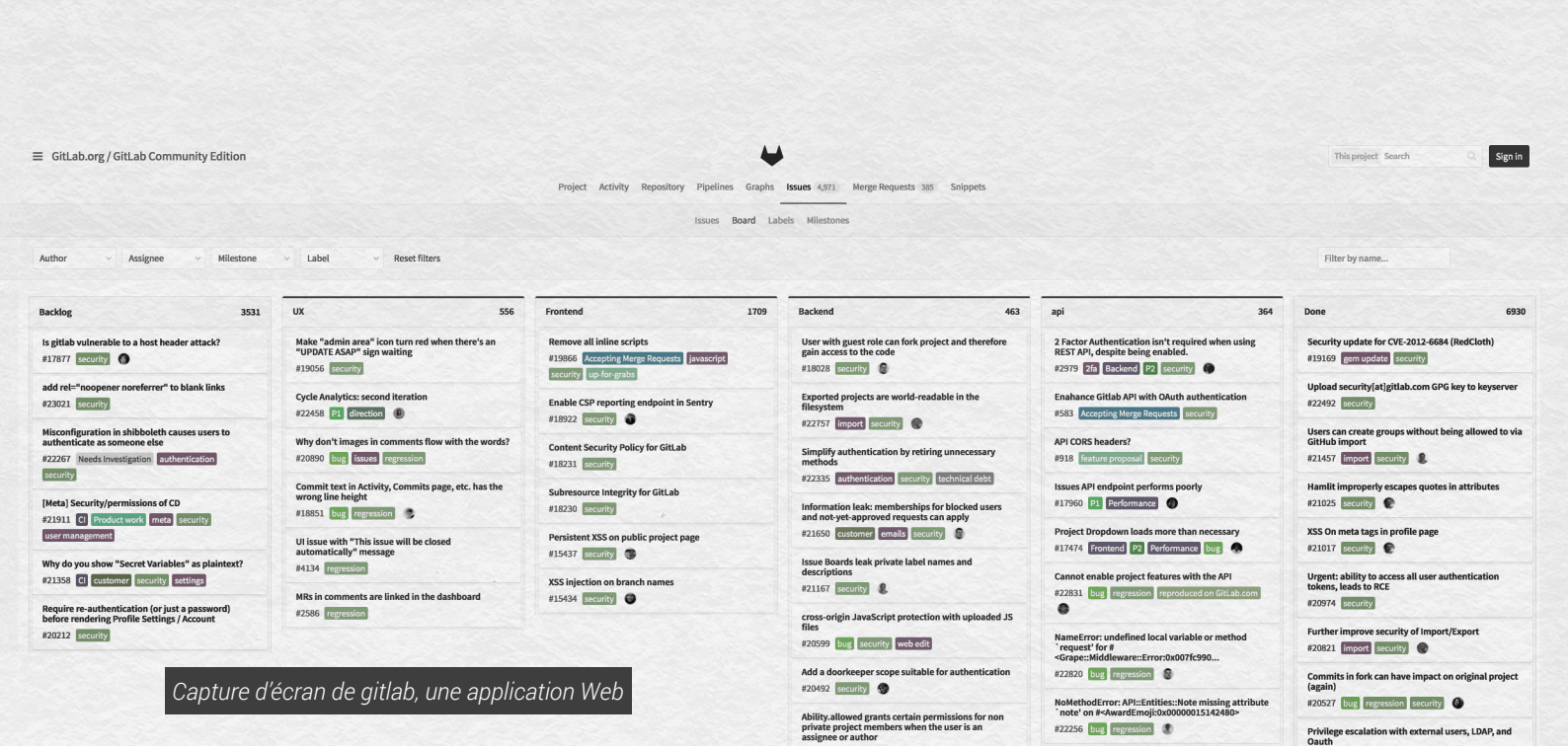
Malgré tous ces inconvénients, le développement front-end est un élément essentiel de l'expérience utilisateur…

Il y a un peu plus de 10 ans, le Web était déjà la plateforme que l'on connaît : incontournable pour les entreprises, omnipotente pour les internautes et déjà beaucoup plus dynamique qu'à ses débuts. Nous développions alors des applications Web en Java ou en PHP (développement back-end) et à chaque interaction avec l'utilisateur, la page toute entière était rechargée, laissant à l'utilisateur une sensation de clignotement de la page.

3_ Le cache est une sauvegarde facilement accessible.

4_ Les premières lignes seront exécutées avant les suivantes.

5_ OS : **Operating System** ou Système d'Exploitation, c'est le logiciel principal qui permet de faire fonctionner un ordinateur. Les OS les plus connus sont Microsoft Windows, Linux ou MacOS ou encore iOS et Android.




*Capture d'écran de gitlab, une application Web*

Pour palier ce clignotement de la page, l'ensemble des acteurs du Web a standardisé XMLHttpRequest[6], créée par Microsoft puis adoptée par Mozilla, Apple et Google. C'est devenu un standard permettant les appels AJAX[7], c'est à dire la récupération de données depuis le serveur sans recharger la page entièrement.

Depuis, nous pouvons écrire des applications Web qui interagissent avec l'utilisateur sans recharger l'ensemble de la page, ce sont les prémices du développement front-end.

Ainsi, JavaScript ne cesse d'évoluer pour être aujourd'hui un langage riche et complet, disponible partout et permettant aux développeurs de proposer une expérience utilisateur qui n'a rien à envier au développement d'application lourde, autrement appelée client lourd.

Lorsqu'un utilisateur se rend sur une application comme gmail ou trello, il n'a plus l'impression d'être sur un site Web. Ses actions sont prises en compte en temps réel, le serveur met à jour la page de manière transparente, et il peut même parfois s'en servir hors-ligne. C'est une véritable expérience utilisateur qui lui est proposée. Cet exemple concret montre bien que la véritable utilité d'un développement front-end est l'expérience utilisateur.

6_ XMLHttpRequest : autrement appelée XHR, XMLHttpRequest est une API JavaScript permettant de faire des transferts de données entre un client et un serveur. Elle permet de récupérer facilement des données depuis une URL sans rafraîchir la page entièrement. (voir https://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/)

7_ AJAX : Asynchronous JavaScript And XML. Cet acronyme définit la possibilité pour une page Web, via du code JavaScript, de faire des appels à un serveur et d'insérer rapidement et incrémentalement la réponse dans le corps de la page sans la recharger. (voir http://adaptivepath.org/ideas/ajax-new-approach-web-applications/)

# LES IMPACTS DE L'APPROCHE FRONT-END / BACK-END

## Une architecture saine

### Un partage des responsabilités plus clair

Le découpage front-end / back-end est structurant pour l'architecture, il force à bien séparer les responsabilités de chaque couche.

En effet, dans une architecture traditionnelle, rien n'empêche d'adopter un mauvais découpage des responsabilités. À partir du moment où l'on obtient les bonnes informations dans chaque page, il n'y a pas de conséquences graves pour le produit.

Par exemple, imaginons une application assez standard, avec une couche d'accès à la base de données, une couche métier et une couche de présentation. Dans cette application, une page est chargée d'afficher les informations personnelles de l'utilisateur. Et pour la construire on a choisi l'approche suivante : la couche présentation demande tous les profils utilisateurs à la couche données, trouve celui de l'utilisateur courant, et en affiche les informations.

Si cette application est réalisée de façon traditionnelle c'est à dire en back-end pur, on peut estimer avec raison que la conception a été un peu défaillante, mais que finalement la page fonctionne sans problème fondamental.

En revanche, si l'application s'appuie sur un front-end JS accédant aux données via une API REST[8], on se retrouve en présence d'une faille de sécurité, car chaque utilisateur charge dans son navigateur les données de tous les utilisateurs. Même si l'application front-end va d'elle-même n'afficher que les siennes, rien ne l'empêche de regarder le contenu des requêtes faites à l'API pour lire les autres informations.

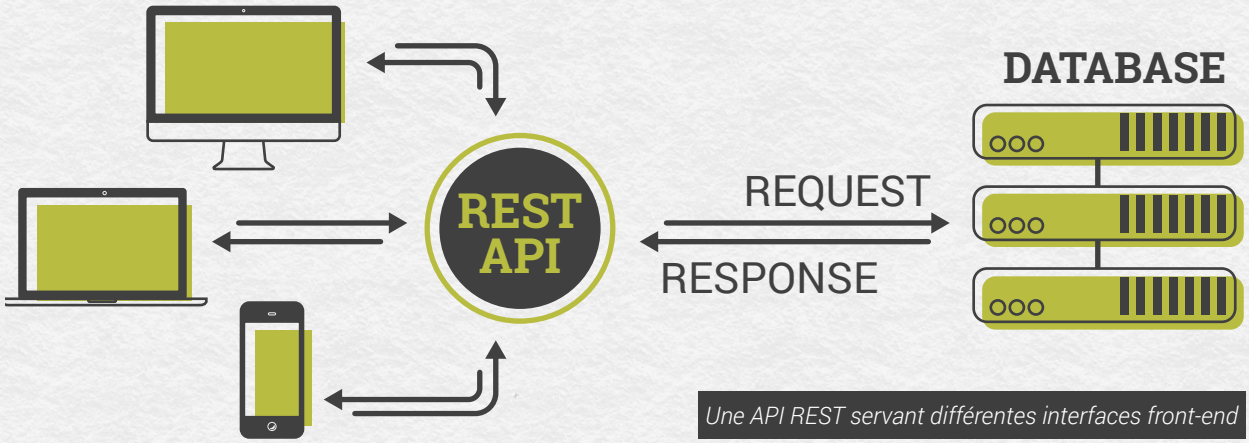
### Une meilleure flexibilité dans l'architecture

D'autre part, la mise en place d'une API REST permet de mieux cerner quelle partie de l'application a besoin de quelles données (ou quelle fonctionnalité back-end). On peut alors envisager de découper le back-end en morceaux ayant des responsabilités spécifiques (on parle alors de services), chacun exposant une API REST.

Par exemple, un service pour la lecture / écriture des données courantes, un service pour les analyses statistiques sur les données archivées, un autre pour la génération de rapports PDF, etc.

---

8_ API REST : (Application Programming Interface, REpresentational State Transfert) une API REST concerne l'architecture d'un service offrant des performances rapides, une fiabilité et la possibilité de mise à l'échelle (ie. grossir et supporter de nombreux nouveaux utilisateurs. (voir https://en.wikipedia.org/wiki/Representational_state_transfer)




*Une API REST servant différentes interfaces front-end*

Les avantages sont nombreux : on peut isoler les goulots d'étranglement (et leurs affecter plus de ressources), on peut ré-utiliser un service dans d'autres applications, ou pour des terminaux différents, on peut migrer vers une autre technologie en plusieurs étapes, etc.

### Des tests plus simples à mettre œuvre

Le back-end et le front-end étant indépendants, il est simple de les tester séparément. C'est avantageux pour chaque partie :

- côté back-end, en comparaison d'une application traditionnelle, tester une API est vraiment élémentaire (on charge des URLs et on lit le JSON renvoyé) ;
- côté front-end, une imitation statique de l'API permet de tester toute l'interface utilisateur sans devoir monter un serveur complet.

## Un partage des tâches de production

Le découpage front-end / back-end implique des compétences très différentes du développement d'applications traditionnelles (full back-end).

Mais il permet de spécialiser les membres d'une équipe pour fournir la pleine mesure d'un savoir-faire particulier, tout en permettant la bonne adéquation des deux parties autour d'un contrat commun que constituera l'API REST.

Ainsi, imaginons une application traditionnelle en Java. Un développeur qui souhaite simplement modifier des sources HTML, CSS et JS doit déployer cette application Java sur son poste, aller chercher ces fichiers dans les méandres d'une arborescence de répertoires pléthoriques, et également maîtriser les outils de build[9] propres à cette application pour pouvoir observer les résultats de son travail.

9 Le **build** est un anglicisme pour parler de l'étape de la conception logicielle pendant laquelle on "construit" le livrable, il inclut souvent : la compilation et l'empaquetage.

C'est bien entendu possible, mais il est clair que ce développeur est tenu d'avoir des compétences qui ne sont pas directement utiles à sa tâche.

Dans une application front-end / back-end, il va travailler dans un projet NPM[10] uniquement, utiliser Gulp ou Webpack[11] pour le build, et il n'a pas besoin de déployer un serveur local, il peut se connecter à l'API que lui fournit l'équipe back-end. API qui peut d'ailleurs être inachevées mais où les fonctionnalités manquantes sont remplacées par des messages codés en dur imitant les réponses attendues, sans que cela ne vienne bloquer le travail du développeur front-end.

Le découpage de l'architecture permet un couplage faible entre les différentes parties de l'application.

## De nouvelles compétences difficiles à acquérir

Autant, comme nous venons de le voir, un développeur front-end aguerri se trouve plus à l'aise s'il n'est pas contraint de déployer le back-end, autant un développeur habitué à travailler sur une architecture traditionnelle peut avoir quelques difficultés à prendre en main les spécificités du développement front-end.

Il va devoir apprendre de nouvelles pratiques, il va être confronté à des concepts nouveaux qu'il aura peut-être du mal à accepter car il va se rendre compte qu'il peine énormément pour effectuer une tâche simple qu'il aurait pu effectuer sans problème avec ses outils habituels.

Prenons le cas d'un formulaire. Créer un formulaire avec un *framework* back-end comme Django par exemple est extrêmement simple et demande une quantité de code très réduite (car le *framework* fournit déjà les mécanismes de validation, les actions d'enregistrement, de suppression, etc.), ainsi toute modification ultérieure sur ce formulaire est alors simple à mettre en œuvre.

Si on travaille maintenant avec *Django REST framework* en back-end et un framework JavaScript en front-end, le travail côté back-end est finalement très proche de celui d'avant, mais il faut encore construire toute la partie front-end, et probablement à la main, car s'il existe des modules front-end capables de générer un formulaire sur la base d'un schéma, ceux-ci restent encore très rudimentaires par rapport à leurs équivalents back-end.

Si par exemple on choisit d'utiliser le format JSON Schema, on est limité car celui-ci décrit des modèles de données, mais pas des formulaires (pas de widgets, de fieldsets, etc.), on va donc devoir l'étendre. cela signifie alors qu'il faudra alors implémenter dans notre back-end la production d'un format spécifique, et dans notre front-end le rendu de celui-ci.

10 Voir le chapitre sur l'environnement du développement front-end

11 Voir le chapitre sur l'environnement du développement front-end

## Une possible fragilisation de la relation client

Le découpage front-end/back-end peut avoir des conséquences dans la relation d'un prestataire avec son client.

D'une part, l'équipe back-end est moins en contact avec le client par rapport à une réalisation traditionnelle, car finalement son travail est peu visible, et les demandes ou attentes des clients ne les concernent pas directement (elles sont adressées à l'équipe front-end qui à son tour fait des demandes à l'équipe back-end).

Cela peut donc entraîner une implication moindre dans le projet, ce qui n'est jamais positif.

D'autre part, l'équipe front-end est en première ligne puisque c'est par son travail que se concrétise les fonctionnalités pour le client. Or, en moyenne, les membres des équipes front-end sont plutôt jeunes, et ce ne sont donc pas toujours les plus expérimentés pour faire faces aux difficultés, faire des estimations, saisir les attentes implicites du client, ou faire remonter un état des lieux clair au chef de projet.

Il peut donc y avoir un risque si la gestion de projet ou l'organisation des équipes ne sont pas bien anticipées. Si les équipes ne sont pas séparées par technologies, mais par fonctionnalités, les responsabilités sont mieux partagées. Une équipe composée de développeurs back-end, de développeurs front-end et des autres acteurs transversaux du projet (UX, chef de projet, etc.) se chargera de la création, de la validation et de la persistance d'un formulaire, de la base de données jusqu'à l'affichage des erreurs sur le navigateur de l'utilisateur.

# LES APPROCHES POSSIBLES

Nous avons vu qu'une application front-end nécessitait de fournir aux navigateurs du code HTML, du CSS et du JavaScript. Le code HTML va mettre en place le contenu de notre application (menu de navigation, contenu principal, articles, etc.), le code CSS va permettre de mettre en forme le contenu (en incluant des effets d'animations par exemple), enfin le code JavaScript va permettre d'animer et de dynamiser notre application en manipulant son contenu.

Nous avons besoin d'accéder aux données de notre application. Cela se fait grâce à JavaScript en accédant à un back-end via une API REST, ou en accédant aux données stockées localement grâce aux API Web d'IndexedDB ou LocalStorage.

Il nous faut organiser notre application grâce à différents écrans qui se succèderont avec ou sans transition ; cela grâce aux liens HTML mais aussi à JavaScript.

Enfin, nous avons besoin de répondre aux actions de nos utilisateurs en inscrivant des comportements à certains composants de notre application. Encore une fois, c'est JavaScript qui va nous permettre d'interagir avec les utilisateurs.

Ainsi, même si nous n'écrivons pas notre application directement en JavaScript, ce dernier est incontournable et doit être fourni au navigateur pour faire tourner une application front-end. Il est alors nécessaire de bien le connaître.

Et si les API Web sont de plus en plus riches, écrire du code ex nihilo n'est peut-être pas la solution idéale pour développer notre application front-end. Il est donc judicieux de bien choisir ses outils. Parmi lesquels le framework.

## Qu'est-ce qu'un framework ?

> Un framework ou structure logicielle est un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture).
>
> — définition Wikipedia.org

Le framework est au développement d'applications ce que les briques de Lego sont à la construction : ils facilitent beaucoup le travail et permettent d'aller plus vite.

Mais comme les briques Lego, ils obligent le développeur à respecter certaines règles : ils ne sont pas forcément compatibles avec d'autres briques de construction, ils ne permettent pas de faire des constructions en bois, etc.

Les frameworks peuvent être assez spécifiques et proposer des fonctionnalités inédites, voire une architecture particulière. La courbe d'apprentissage peut être plus ou moins raide. L'expertise d'un *framework* sera une qualité à développer et il faudra donc préférer ne pas en changer régulièrement.

Pour autant, le développement front-end est toujours basé sur JavaScript, que l'on utilise un *framework* ou non. Il est donc quand même nécessaire de bien connaître le langage JavaScript pour se spécialiser ensuite dans un *framework*.

## Vanilla JS, du code JavaScript pur

Si les *frameworks* apportent beaucoup d'avantages, ils enferment le projet dans leurs spécificités. Sans compter qu'il faut comprendre leurs concepts avant de pouvoir développer une application de qualité.

C'est pour cette raison qu'il peut être préférable, dans le cadre de petits projets, d'écrire du code JavaScript sans framework.

On peut se servir de bibliothèques pour soulager les API Web, pour des raisons de compatibilités ou d'efficacité. On parle alors, par exemple, de jQuery[12] ou de fetch[13].

Il n'y a pas de mauvaises raisons à développer une application front-end sans *framework* si ce n'est la taille de l'application et donc sa complexité. Plus l'application devient complexe — navigation et historique, mise à jour avec les données du back-end, animation, etc. — plus il deviendra pertinent d'envisager l'utilisation d'un *framework*.

## Écrire son propre framework ou utiliser un framework dominant ?

Puisqu'il faut écrire du code pour faire tourner notre application, il peut être utile de regrouper le code générique au sein d'une bibliothèque. Cela permet une meilleure organisation du code, une maintenance plus facile et une abstraction plus directe.

Et si nous n'utilisons pas de *framework*, parce que nous en maîtrisons les défauts et les avantages, il nous viendra peut-être l'envie ou le besoin d'écrire notre propre *framework*.

Dans une grande équipe, cela peut permettre d'unifier les méthodes de développement, partageant au sein du *framework* maison les bonnes pratiques de l'équipe. Ce n'est d'ailleurs pas un hasard si les *frameworks,* qui font tourner la plupart des applications Web, ont été développés par des équipes de développement conséquentes, comme celles de Yahoo avec Yui (désormais abandonné), Google avec Angular, Facebook avec React – bien que React ne puisse être considéré comme un *framework*, nous le rangeons dans la même famille par souci de simplicité.

Maintenir un *framework* maison est au moins aussi coûteux que de maintenir une application front-end complète. Si notre équipe de développement est restreinte, il n'apparaît pas judicieux d'augmenter la charge de travail en créant puis en maintenant un *framework* maison.

C'est pourquoi il sera souvent préférable d'adopter un *framework* dominant.

12_ jQuery : Bibliothèque JavaScript créée pour simplifier l'écriture de scripts front-end et assurer une compatibilité avec la plupart des navigateurs. (voir https://github.com/jquery/jquery)

13_ Fetch : Bibliothèque qui permet de simuler le nouveau standard permettant de simplifier l'écriture des requêtes XMLHttpRequest. (voir https://github.com/github/fetch)

# Les grands choix possibles, les comprendre

Comme précisé précédemen, bien qu'il soit possible de construire une application front-end ex nihilo, il sera toujours plus pérenne de s'appuyer sur un *framework* dominant.

C'est la raison pour laquelle nous avons choisi de présenter les différents *frameworks* majeurs afin de vous aider dans votre décision.

Cette sélection n'est pas le fruit du hasard. Chez Makina Corpus, nous pensons que la meilleure raison pour choisir un *framework* (ou une bibliothèque) est toujours sa communauté[14].

Avoir une bonne communauté assure la disponibilité :

- de ressources documentaires (tutoriaux, réponses sur Stack Overflow[15], articles de blog, livres, formations),
- de ressources humaines (candidats recrutables, prestataires de services),
- et bien sûr de ressources techniques (modules additionnels, mises à jour, correctifs de sécurité, etc.).

Pour chacun des *frameworks* présentés, nous avons essayé d'être objectifs en traitant à chaque fois les mêmes sujets :

- ses principes ;
- son lexique ;
- sa typologie de projets ;
- ses exemples de projets.

N.B : les exemples de sites utilisant les *frameworks* sont donnés à titre indicatif.

14_ Il esr possible de retrouver la même sélection auprès du sondage stateofjs.com : https://medium.com/@sachagreif/the-state-of-javascript-front-end-frameworks-1a2d8a61510

15_ Stack Overflow est une site de questions réponses pour les développeurs professionnels et passionnés. (voir https://stackoverflow.com/tour)

Les *frameworks* que nous présentons sont les suivants :

- AngularJS (la version 1);
- Angular (la version 2);
- React ;
- Vue.js ;
- Ember.js ;
- Backbone.js.

Afin d'offrir une vision globale des choix possibles, nous présentons un résumé succinct des différentes solutions dans une synthèse en fin de chapitre.


## Les principes du framework

AngularJS a été créé en 2009 par les équipes de Google. C'est l'un des tous premiers frameworks front-end du genre, à la base principalement conçu pour communiquer avec une API REST, manipuler du JSON et faciliter la modification du DOM.

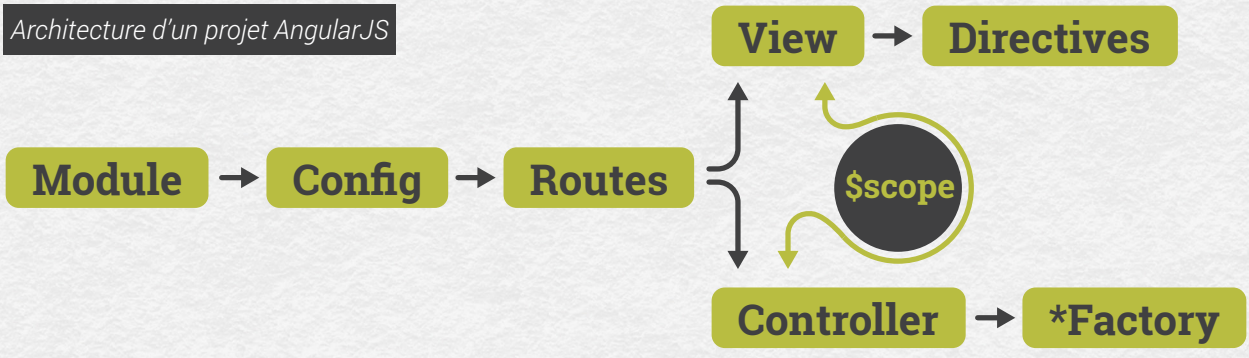
Il est souvent utilisé afin de structurer et simplifier le développement d'applications mono-pages (*Single Page Application*).

Après plusieurs années en tant que référence dans le développement front-end, il perd aujourd'hui de sa popularité au profit de frameworks plus récents. La version 2 d'AngularJS est sortie en septembre 2016 mais est radicalement différente de la version 1 comme nous le verrons dans la suite du livre.

Les principes fondamentaux d'AngularJS :

- le *two-way data binding*[16] : le modèle et la vue sont liés. La mise à jour de l'un est automatiquement répercutée sur l'autre ;
- l'injection de dépendances ;
- l'enrichissement du DOM grâce aux directives.

*Architecture d'un projet AngularJS*



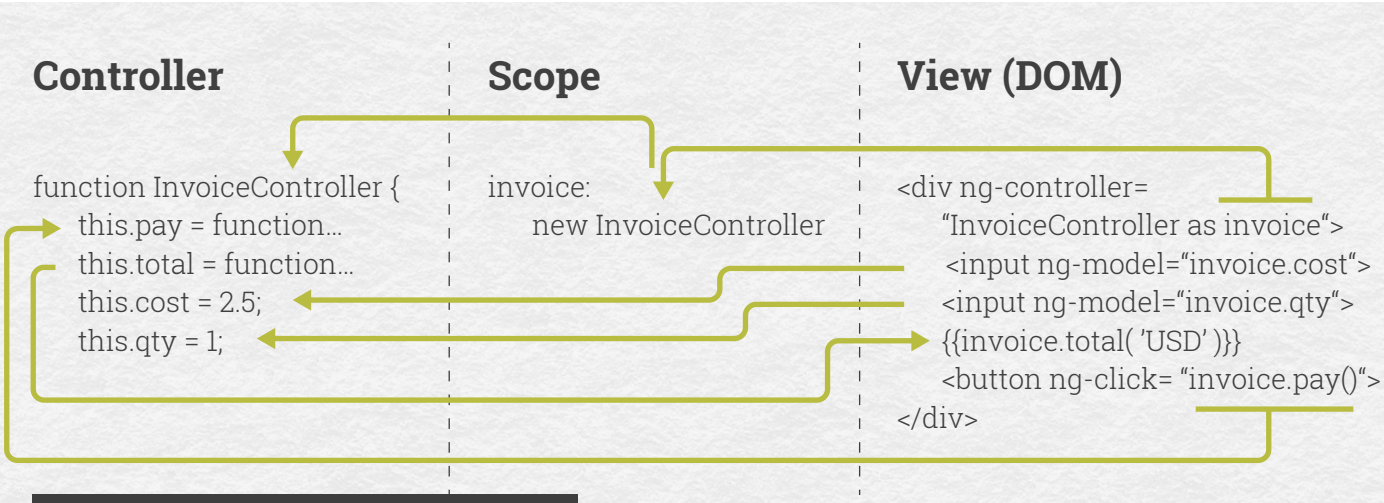

16_ Le two-way data binding désigne le fait que les données sont partagées entre la vue et le modèle. Une modification sur l'un se répercute sur l'autre, dans les deux sens. Pour plus de détails, voir le glossaire.

La courbe d'apprentissage d'AngularJS n'est pas linéaire. Mais si on maîtrise déjà les architectures MVC[17], il sera assez facile de s'y retrouver dans AngularJS. D'ailleurs le *framework* a été construit pour plaire aux développeurs back-end ayant l'habitude de cette architecture (Play framework, .Net ou encore PHP Symfony).

Mais de nombreux principes sont tout de même à assimiler. Par exemple ce qu'on appelle *controller* dans AngularJS ne correspond pas tout à fait au contrôleur de l'architecture MVC, qui conviendra plutôt à un couple service – *controller*. Les *directives* sont assez puissantes à manipuler et permettent de créer des composants ou d'ajouter des comportements à des éléments HTML standards.

Pour une meilleure maîtrise de la base de code, il est conseillé de suivre un *styleguide* qui propose aux développeurs quelques règles et méthodologies simples pour écrire un code propre et efficace. Par exemple ceux de John Papa ou de Todd Moto, deux développeurs très actifs dans la communauté AngularJS 1 et Angular 2.

## Lexique du framework




*Directives, Controller et Scope dans AngularJS 1*

### Directives

Les directives sont les unités essentielles du *framework*. Elles servent à la fois à ajouter du comportement aux éléments du DOM mais également à créer des composants réutilisables et composables. Ainsi une directive peut tout aussi bien ajouter de l'auto-complétion sur un champ texte standard, ou bien être un nouveau type de champ de formulaire qui inclut des nouvelles fonctionnalités.

Depuis la version 1.5, AngularJS propose un nouveau concept : les components. Il permettent d'écrire des directives plus facilement.

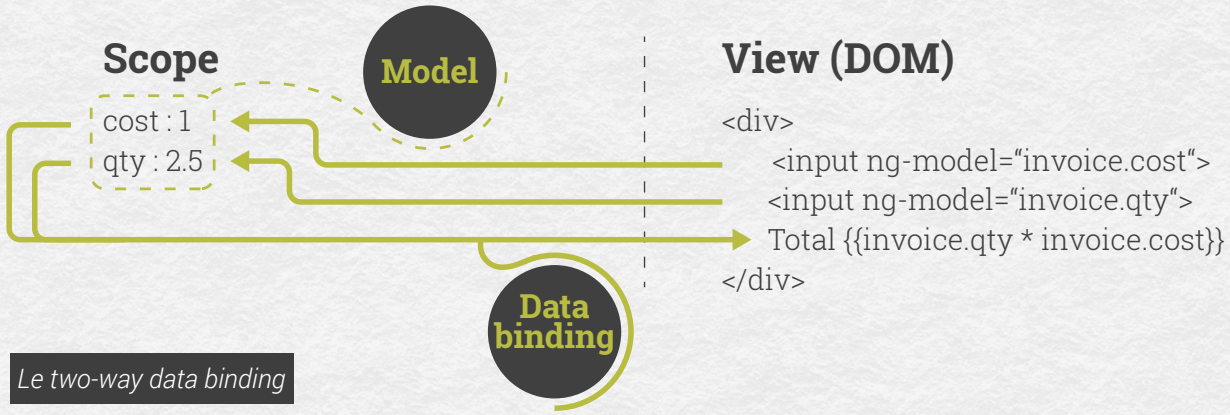
17_ MVC (Model View Controller) : manière de structurer son code de façon à séparer les données (Model) de la vue (View) et des actions (Controller). (voir http://todomvc.com/)

## Controllers

Les *controllers* vont organiser les données pour les mettre à disposition des vues. Les *controllers* n'ont pas vocation à contenir de logique métier mais à faire appel à des services. Ce sont les chefs d'orchestre de l'application.

## Scope

Le *scope* permet d'organiser les données. Elles sont ensuite disponibles dans la vue via les controllers. C'est grâce au *scope* que *controllers* et vues peuvent communiquer dans les 2 sens (vue › *controller* et *controller* › vue), c'est le principe du *two-way data binding*.

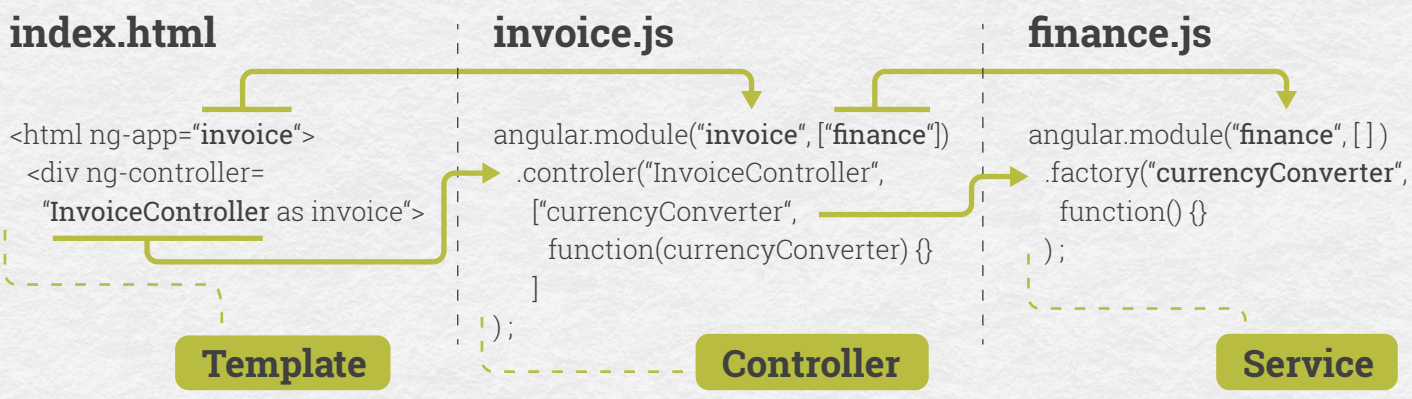



*Le two-way data binding*

## Services

Les *services* sont des modules permettant de fournir des informations aux autres composants de notre application (*controllers*, *directives* ou d'autres *services*). Ils permettent d'organiser la logique métier et le reste du code afin de les rendre disponibles au reste de l'application.

Les *services* sont la partie riche de l'application, il est de fait fortement conseillé de couvrir leur code par des tests unitaires.




*Le Service dans AngularJS*

### Injection de dépendances

L'injection de dépendances est centrale dans AngularJS. Tous les éléments (*controllers*, *services*, etc.) sont enregistrés auprès du moteur d'AngularJS via des modules. Ils sont alors disponibles pour l'injection dans notre application et peuvent être utilisés dans n'importe quel élément en les passant en paramètre de son constructeur.

## Typologie de projets

AngularJS est un *framework* très complet. Il permet de conserver une certaine rigueur apportant un cadre efficace aux développeurs. Les gros projets en bénéficient grandement car le code peut rester facilement organisé grâce à son architecture MVVM (proche MVC).

Les développeurs maîtrisant ses principes sont nombreux, et malgré l'arrivée d'Angular 2, il reste encore très utilisé.

Il est également utilisé avec Ionic Framework, un *framework* permettant de construire des applications hybrides pour le mobile dont nous parlerons plus loin dans le livre.

## Migration vers Angular 2

Si la migration vers Angular 2 ne se fera pas de manière instantanée, il reste possible de mettre en place une stratégie de mise à jour progressive. En effet, une application Angular 2 est capable d'encapsuler une application AngularJS moyennant quelques changements mineurs. Il devient alors possible de migrer des morceaux de l'application petit à petit. Bien sûr, une couverture de tests sera indispensable pour s'assurer de repérer toute fonctionnalité qui cesserait de fonctionner.

Également, la version 1.5 apporte les composants, la transclusion multiple, ainsi que les événements de cycle qui ont été empruntés à Angular 2 afin de faciliter la transition.

Enfin, il existe désormais de nombreux exemples pour utiliser les dernières versions de JavaScript (ES2015) avec Angular, facilitant encore plus la transition.

## Exemples utilisant AngularJS

- https://www.indiegogo.com/ ;
- http://www.gettyimages.fr/ ;
- https://www.bouyguestelecom.fr/ ;
- https://support.microsoft.com/ ;
- https://www.google.com/partners/about/index.html ;
- https://weather.com/

## Pour aller plus loin

- Développez vos applications web avec AngularJS ;
- AngularJS Style Guide par John Papa ;
- Le blog Thoughtram ;
- NgBook ;
- Ionic, framework mobile basé sur AngularJS et Cordova ;
- AngularUI.

# ANGULAR 2

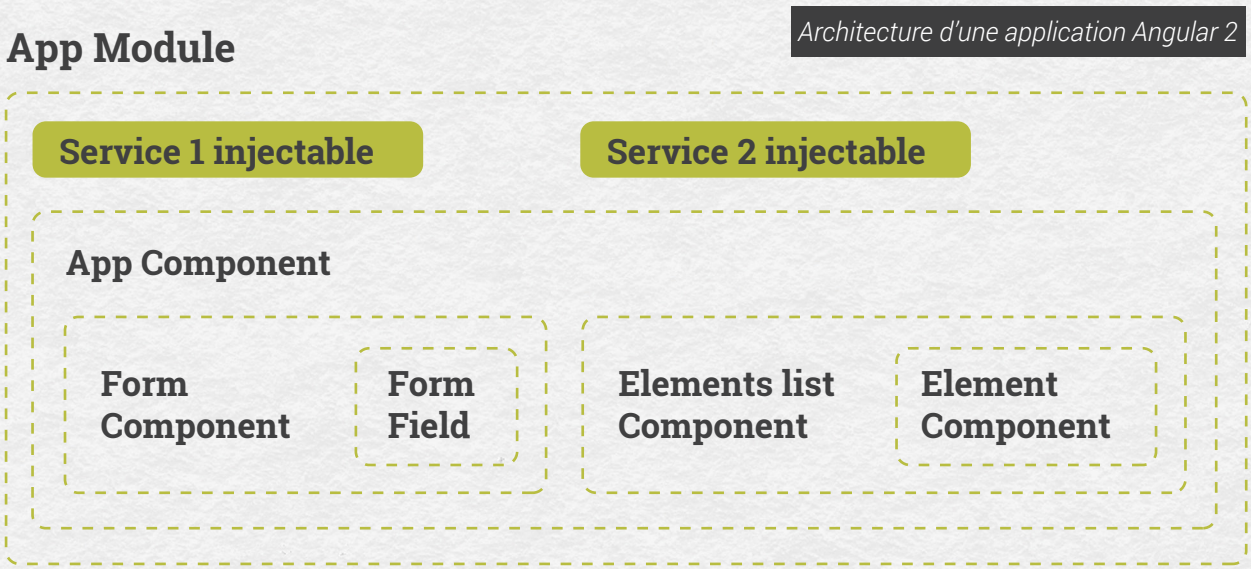
## Les principes du framework

La seconde version du *framework* de Google a été entièrement réécrite par les équipes de Google pour donner un outil plus mature, accessible et dans l'air du temps.

L'important est de garder à l'esprit la philosophie assez englobante d'Angular. Cette volonté de cadre fort est toujours présente. Il est alors important de voir si cela convient, ou non, au projet.

Voici les principes fondamentaux d'Angular 2 :

- Tout est composant : nous devons construire notre application avec de nombreuses briques plutôt que de créer des blocs monolithiques.
- L'injection de dépendances : cela a toujours fait partie du socle Angular. Cependant elle devient plus compréhensible que dans AngularJS.
- L'universalité : la possibilité de fonctionner sur des plateformes variées, du serveur au mobile en passant par l'ordinateur.




*Architecture d'une application Angular 2*

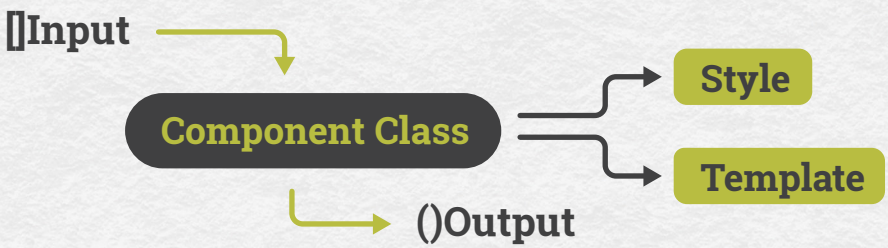
En somme le *framework* tourne essentiellement autour de l'utilisation de classes[18]. Angular 2 a conservé le choix de permettre la séparation nette du templating, du style et du scripting. La courbe d'apprentissage est plus douce par rapport à AngularJS, notamment grâce à la réduction du nombre de concepts. Il est important de suivre les recommandations[19] officielles fournies par Google pour un bon maintien des applications.
Enfin, le *framework* a laissé un peu de côté le *two-way data binding* très présent dans la première version. Par défaut le lien est désormais unidirectionnel mais il est possible de mettre en place le two-way via la directive *ng-model*.

## Lexique du framework

### Composants

Les composants sont les briques de notre application. Ils vont nous permettre de la façonner à volonté en encapsulant les fonctionnalités principales.

*Fonctionnement d'un composant Angular 2*




### Directives

Les directives sont un moyen d'ajouter des comportements à des éléments de nos templates, comme par exemple des interactions avec l'utilisateur ou des affichages différents selon certaines conditions. La directive est généralement utilisée sous forme d'attribut.

### Services

Les services sont des modules permettant de fournir des informations aux autres composants de notre application. Ils vont permettre notamment d'organiser la logique métier ou la communication avec les sources de données.

---

18_ Les classes n'existaient pas dans JavaScript avant l'implémentation des spécifications ECMAScript2015. Cependant elles existent dans TypeScript, un langage de transpilation pour JavaScript.

19_ La documentation d'Angular 2 propose des recommandations sur le site https://angular.io/docs/ts/latest/guide/style-guide.html

### Modules

Les modules vont servir à encapsuler les fonctionnalités de notre application en regroupant un ensemble de composants, directives, services, ou pipes selon une responsabilité commune.

### Injection de dépendances

Dans Angular 2, l'injection de dépendances est majoritairement utilisée au niveau des modules. Cela permet de rendre leurs contenus disponibles dans toute l'application plutôt que de devoir importer chaque élément indépendamment aux endroits où ils sont utilisés.

## Typologie de projets

Angular 2 est un *framework* très complet et très dogmatique. Il impose un cadre assez fort, bien que moins important par rapport à la version 1. Les gros projets en bénéficient grandement de par la rigueur que cela apporte. Un petit projet ou un composant minimaliste verra un temps de mise en place peut-être trop important.

## Exemples utilisant Angular 2

- Ionic framework v2 ;
- NG2 Admin.

## Pour aller plus loin

- NgBook 2 ;
- Le blog Thoughtram ;
- Liste officielle des ressources Angular 2 ;
- Une recette efficace pour commencer un projet Angular 2.

# REACT

## Les principes du framework

Nous avons choisi de parler de React dans ce livre blanc, car intégré dans une architecture particulière, elle se révèle être une formidable bibliothèque.

*Votre code appelle une bibliothèque, un framework appelle votre code*[20]

React est une bibliothèque JavaScript développée et maintenue par Facebook pour réaliser des interfaces utilisateurs. Elle a été proposée pour répondre à la problématique suivante : construire de grosses applications dont les données changent continuellement.

Associée à la bibliothèque react-dom, React devient une API pour le DOM. Plutôt que d'utiliser des templates comme les autres *frameworks*, React s'organise autour de l'assemblage de composants. Ainsi elle propose d'utiliser JavaScript pour créer des interfaces utilisateurs, à l'instar de jQuery.

Bien que React peut être considérée comme la « Vue » de MVC, le développeur React sera plus à l'aise s'il l'intègre dans une architecture *one-way data flow* (flux de données à sens unique). La gestion des différents états possibles d'une application n'est pas le point fort d'une architecture MVC. En revanche, c'est exactement dans ces cas là que React va être utilisée au sein d'une architecture que Facebook a appelé Flux.

Flux est une architecture de programmation réactive[20]. React va permettre le rendu des composants selon certains critères, et Flux va lui permettre de les rendre de manière adéquate, en fonction des interactions de l'utilisateur, des données d'un serveur et de l'état de l'application.

Tous ces concepts ne sont pas classiques, et une équipe de développement front-end familière de l'architecture MVC rencontrera quelques difficultés à intégrer les concepts radicalement différents qu'implique React.

Finalement, cette bibliothèque n'a qu'une seule fonctionnalité : la manipulation de l'interface utilisateur. Pour l'ensemble des autres fonctionnalités d'une application front-end, il faudra faire appel à d'autres bibliothèques[21] ou API comme :

- react-router pour la gestion des routes ;
- redux ou MobX pour la gestion des états ;
- fetch ou XMLHttpRequest pour les appels serveurs ;
- indexedDB ou localForage pour la persistence des données ;
- jsdom ou enzyme pour le test des composants.

# Lexique du framework

## Composant

Le composant est le concept central de React. Comme un élément HTML classique, il va avoir un rendu particulier associé à un comportement. Il pourra être créé à partir d'une composition de composants.

Nous pouvons imaginer les composants React comme des fonctions qui prennent des paramètres en entrée et retournent du HTML. La composition de composants est aussi triviale que la composition de fonctions.

20_ Programmation réactive : consiste à maintenir une cohérence entre les données grâce à la propagation d'événements. Basée sur les flux, un événement est déclenché dès qu'un changement est survenu dans la source de données.

21_ Les listes awesome-foobar sont formidables pour trouver des outils, bibliothèques et exemples. Voici la liste pour react https://github.com/brillout/awesome-react-components.

## Programmation déclarative et réactive

React propose un style de programmation déclarative, tout comme HTML. En modélisant les états plutôt que les transitions, cette bibliothèque s'intègre bien dans une architecture orientée programmation réactive.

> When the data changes, React conceptually hits the «refresh» button, and knows to only update the changed parts.
>
> – facebook.github.io/react

La programmation réactive[22] est un style de programmation dirigée par les données et les flux de données. Un tableur est un cas classique de programmation réactive : lorsqu'une valeur change, une cellule avec un calcul affiche immédiatement le résultat avec la nouvelle valeur.

L'association de l'architecture Flux et de React permet de développer une application basée sur ces concepts.

## JSX

Afin de convenir au mode de programmation déclarative, React propose une couche de sucre syntaxique proche du XML pour décrire les composants.

L'utilisation de JSX n'est pas obligatoire avec React, mais après avoir adopté les concepts de programmation déclarative, il devient un atout évident.

## Props et States

Les *props* et *states* sont les données d'un composant React. Les props sont l'équivalent des attributs d'un élément HTML. Les states sont internes aux composants et permettent de stocker des données utiles au rendu du composant.

Il est recommandé d'écrire des composants avec le moins de states possibles, on les appellera stateless components.

> The stateful component encapsulates all of the interaction logic, while the stateless components take care of rendering data in a declarative way.
>
> – https://facebook.github.io/react

22_ The Introduction to Reactive Programming you've been missing, Andre Staltz https://gist.github.com/staltz/868e7e9bc2a7b8c1f754#file-introrx-md

### Flux

La proposition de Facebook pour manipuler les éléments graphiques d'une application au travers du DOM a été suivie d'une proposition d'architecture basée sur les concepts de *Event Sourcing*[23] et de *CQRS*[24].

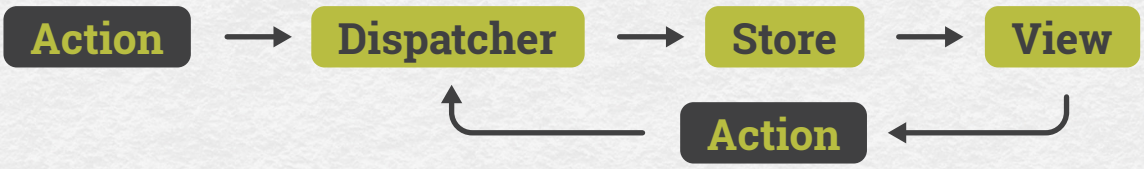



*Schéma de l'architecture Flux de Facebook*

Cette architecture a été nommée Flux car elle crée un cycle unidirectionnel (*one-way data flow*) le long duquel transitent les données de l'application et les actions de l'utilisateur.

> Capture all changes to an application state as a sequence of events.
>
> – Martin Fowler

### Redux

Redux est une bibliothèque implémentant l'architecture Flux. Elle est particulièrement efficace pour répondre aux concepts préconisés par React comme la programmation réactive.

L'ensemble de l'état de l'application est défini dans un seul objet. Les interactions génèrent des actions qui vont permettre la création d'un nouvel état. Les composants qui sont associés à certaines valeurs de l'état seront mis à jour si nécessaire.

## Projets utilisant React

- http://www.6play.fr/ ;
- https://www.airbnb.com/ ;
- https://www.arte.tv/guide/en/ .

23_ Event-Sourcing est un terme qui propose de modéliser l'ensemble du domaine d'une application à partir d'événements qui deviennent la seule source de vérité. (voir http://www.martinfowler.com/eaaDev/EventSourcing.html)

24_ CQRS signifie Command-Query Responsibility Segregation. C'est une architecture qui stipule la séparation du modèle qui lit les données avec celui qui écrit les données. (voir http://martinfowler.com/bliki/CQRS.html)

- http://www.bbc.co.uk/ ;
- https://clashofclans.com/ ;
- http://opengov.com/ ;
- http://business.stackoverflow.com/careers/us/platform/customer-support ;
- http://netflix.com/ ;
- et bien d'autres.

## Pour aller plus loin

- Penser en React : https://facebook.github.io/react/docs/thinking-in-react.html ;
- Une approche linéaire et basée sur un projet pour apprendre React et son écosystème : http://www.reactjsprogram.com/ ;
- Une guide compréhensible au Test-First Development avec Redux, React et Immutable : http://teropa.info/blog/2015/09/10/full-stack-redux-tutorial.html ;
- Composants intelligents et idiots : https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0 ;
- Présentation de React pour les non initiés : https://youtu.be/HG3Glpv8b6o ;
- Une approche TDD pas-à-pas pour écrire une application React : http://thereignn.ghost.io/a-step-by-step-tdd-approach-on-testing-react-components-using-enzyme/ ;
- Comment aborder l'écosystème React ? https://github.com/petehunt/react-howto/blob/master/README-fr.md.

# VUE.JS

## Principes fondamentaux

Vue.js est une bibliothèque permettant de développer des applications basées sur des composants web. Elle implémente l'architecture MVVM[25] (*Model-View-ViewModel*).

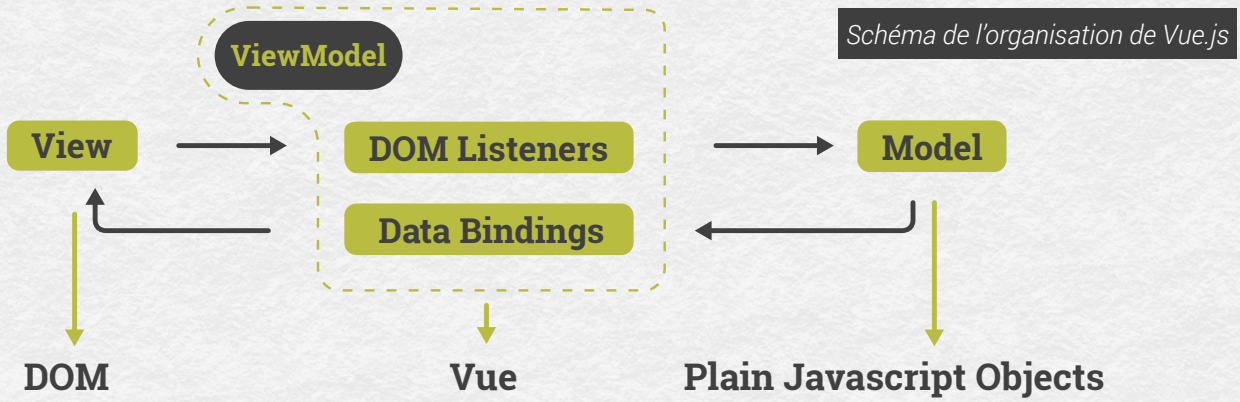



*Schéma de l'organisation de Vue.js*

Vue.js a été écrit par un ancien employé de Google, Evan You. Il est actuellement le mainteneur principal de la bibliothèque.

Plusieurs sociétés soutiennent son initiative via un financement de l'activité de développement sur la plate-forme Patreon. D'autres sociétés, dont certaines étant de grands acteurs sur le marché chinois, font aussi confiance à cette bibliothèque et son créateur.

Son souhait a été de développer une bibliothèque agréable et facile à utiliser. Il s'est inspiré des bonnes pratiques d'autres *frameworks* telles que AngularJS 1 et React.

25_ MVVM signifie Model-View-ViewModel. C'est une architecture dérivée de MVC qui propose de lier la Vue et le Modèle par une couche de conversion des données. (voir https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel)

Vue.js a été publié initialement en 2014, ce qui en fait une bibliothèque encore jeune. Une communauté d'environ quinze personnes maintient la bibliothèque à ce jour.

Chaque modification du modèle de données met à jour automatiquement l'affichage du composant, à l'instar du *two-way data binding* d'AngularJS 1.

Vue.js dispose d'un ensemble de bibliothèques (*routing, state management*, etc.) le rendant très modulable.

Quelques exemples de bibliothèques optionnelles :

- gestion d'état centralisé : vuex ;
- gestion du routing côté client : vue-router ;
- gestion des ressources externes, type API : vue-resource ;
- aide à la création d'applications complexes : vue-cli ;
- ajout d'un rendu côté serveur : vue-server-renderer.

La communauté maintient une documentation à jour pour chaque version de la bibliothèque, ce qui en facilite l'apprentissage.

La version 1.0 a été stabilisée en 2015 et la version 2.0 a été finalisée en Septembre 2016. Cette dernière mouture implémente et promeut quelques grandes évolutions :

- implémentation du Virtual-DOM[26], pour l'optimisation de la mise à jour des composants ;
- l'amélioration de la communication entre composants et avec leurs parents, par l'utilisation d'un bus d'événement.

# Lexique du framework

## Composant

Le composant est l'élément de base dans la bibliothèque Vue.js.

Il s'agit d'un template HTML associé à une gestion d'état en JavaScript (modèle de données et interactions, *ViewModel*) et à un style CSS.

Grâce à cette architecture, il est tout à fait possible de construire des applications de taille modeste ou à plus grande échelle.

Il est aussi possible d'intégrer des composants Vue.js à une application web existante, en greffon.

26_ Le virtual-DOM permet la construction d'un arbre correspondant au DOM. Des heuristiques sur les modifications, recherches de nœuds permettent de manière générale d'accélérer la mise à jour du DOM en calculant au préalable les modifications à apporter dans le virtual-DOM.

### Reactive Data Binding

Vue.js analyse l'ensemble des composants, leurs propriétés, leurs méthodes, et peut détecter la mutation d'une donnée du modèle, ce qui lui permet de :

- déclencher la mise à jour des autres données du modèle dépendant sur cette donnée mutée ;
- déclencher la mise à jour des composants dépendants des données mises à jour par la mutation initiale.

En 1.0, le DOM du composant sera entièrement mis à jour, alors qu'en 2.0, l'utilisation du virtual-DOM permettra de mettre à jour uniquement les portions du DOM ayant réellement changé.

## Typologie de projets

Grâce à la flexibilité de la bibliothèque, Vue.js peut correspondre à la fois à des petits projets nécessitant uniquement la couche de vue, ainsi qu'à de plus gros projets utilisant alors les bibliothèques optionnelles.

Né après les autres grands frameworks modernes, Vue.js essaie de combler les manques ou de faciliter les complexités des autres frameworks desquels il s'inspire.

Pour autant il reste une bibliothèque assez jeune, mais avec une popularité grandissante.

## Exemples de projets

À ce jour, la communauté Laravel (framework PHP) a choisi comme bibliothèque front-end par défaut Vue.js.

- Facebook news feed ;
- Gitlab ;
- CMS PageKit.
- Weex

## Pour aller plus loin

- Guide Vue.js ;
- API Vue.js ;
- Pourquoi ISL a changé son framework front-end pour Vue.js ;
- Ressources pour apprendre Vue.js ;
- Exemples Vue.js.


# EMBER.JS

## Les principes du framework

Ember.js est un *framework* OpenSource sous licence MIT qui a longtemps été maintenu par Sproutit et Apple. Désormais il est maintenu par une communauté très active, son dépôt github annonce la participation de plus de 600 contributeurs avec près de 13 000 commits.

### Outils

Le *framework* est contrôlé via un CLI[27] : il permet de créer un projet, d'ajouter une route, un composant, un test ou encore un modèle.

Tous ces modules vont ajouter des fichiers à l'application et seront organisés sans que le développeur ne s'en préoccupe.

### Architecture

De la même manière, l'architecture logicielle est elle aussi assistée. Les développeurs qui connaissent bien MVC ne seront pas dépaysés puisque c'est ainsi qu'est construit Ember.js. La partie vue est déportée sur un dérivé de Mustache : Handlebars (voir *Templating*).

Ember.js privilégie la convention à la configuration. Son CLI va accompagner le développeur tout au long du développement de l'application en lui proposant les bonnes pratiques à suivre.

Il sera alors possible de créer des tests d'acceptance, d'intégration et chaque ajout d'un modèle ou d'une route créera un test unitaire associé.

Toujours grâce à de solides conventions, la persistance des données dans une application Ember.js sera facilitée si on l'associe à un back-end Ruby-on-Rails ou RESTful. Il devient alors très facile, dès la création d'un modèle, de lister, supprimer, ou sauver les données sur le serveur d'API REST.

27_ CLI (Command Line Interface ou interface en ligne de commande) : Programme qui s'exécute en ligne de commande permettant de réaliser certaines tâches : initialisation d'un projet, création d'un modèle, etc.

### Performances

La performance a toujours été un point central dans le développement d'Ember.js. Grâce à ses diverses optimisations et son architecture guidée, les applications développées avec Ember.js grossissent facilement, restent fluides et se chargent rapidement.

Pour autant, le développement d'Ember.js a débuté en 2007, ce qui en fait un des plus vieux *framework* JavaScript front-end. Son code source contient des parties non maintenues et s'appuie sur des bibliothèques vieillissantes.

### Templating

L'utilisation de Handlebars peut être équivoque puisqu'il alourdit le DOM en ajoutant des balises *<script>* pour marquer les parties du template à mettre à jour. Heureusement le projet a prévu de migrer vers HTMLBars, une refonte de Handlebars qui sait manipuler le DOM et qui n'aura donc plus besoin de ces balises.

### Cible

Ce *framework*, avec son CLI et son accompagnement constant vers une architecture MVC classique, facilite l'ajout de tests à chaque étape (création de composant, de modèle ou de contrôleur) et plaira aux développeurs Ruby, Python, Java ou C#. Ember.js ne cible pas le développeur Web aguerri, mais sera parfait pour un développeur back-end qui veut construire une application sans trop se poser de questions.

## Lexique du framework

### Route

Pour Ember.js une route fait correspondre une URL à un template et à un modèle. Lorsque le développeur, via le CLI, crée une nouvelle route, Ember.js crée la configuration nécessaire pour cette dernière :

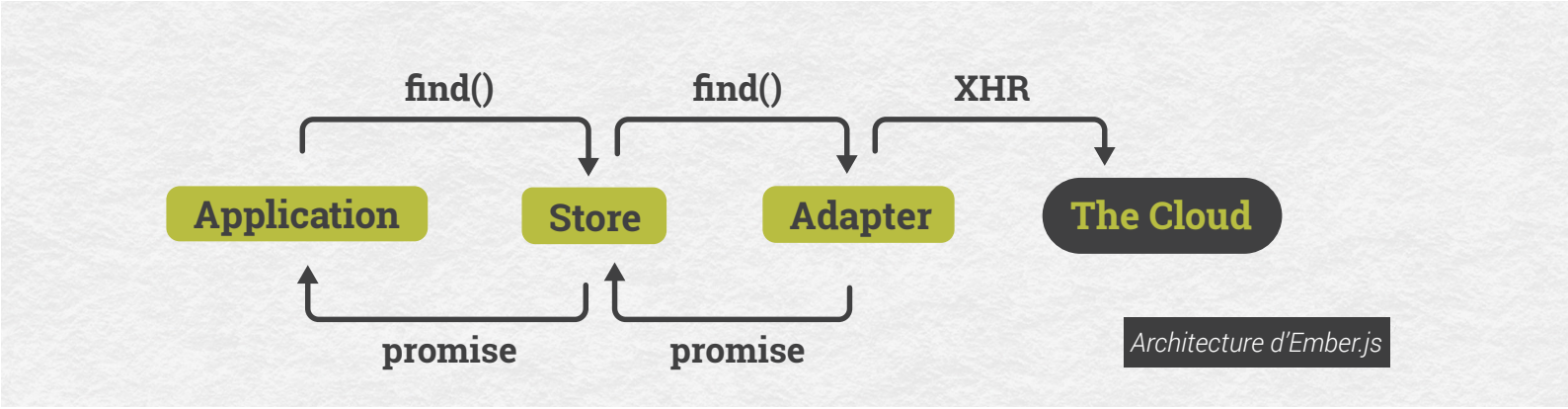
- le template correspondant sous forme d'un fichier Handlebars ;
- le modèle ;
- et un fichier pouvant accueillir les tests unitaires associés.

### Modèle

Comme dans toute application MVC, le modèle d'Ember.js accueille les données de l'application. Par défaut, ces données sont chargées depuis et sauvegardées vers un serveur associé à une base de données. Il est néanmoins possible d'ajouter un comportement pour une sauvegarde en local dans indexedDB pour s'assurer d'une utilisation hors-ligne.

### Store

Afin de conserver une seule source fiable pour la gestion des données du modèle, Ember.js fournit un unique store qui sera le dépôt central des données du modèle. L'implémentation du store d'Ember.js va permettre l'accès concurrent par deux composants à un même modèle.




*Architecture d'Ember.js*

### Templates

Les templates d'Ember.js reposent sur Handlebars. Un template Handlebars peut gérer l'ensemble des éléments HTML disponibles, ainsi que des composants Ember.js créés pour l'occasion.

### Composants

Le composant dans Ember.js est un template associé à un comportement. Lors de la création d'un nouveau composant par le CLI Ember.js, un test d'intégration est automatiquement créé et ajouté au projet.

## Projets utilisant Ember.js

- https://www.zendesk.com/ ;
- https://yahoo.com/ ;
- http://www.discourse.org/ ;
- https://www.groupon.com/ ;
- et bien d'autres.

## Pour aller plus loin

- Guide officiel Ember.js ;
- Questions Stack Overflow.

# BACKBONE.JS

## Les principes du framework

Backbone.js a été publié pour la première fois en 2010. Framework basé sur Underscore.js, il s'est présenté comme une alternative à Angular et Ember plus minimaliste et permissive sur l'architecture et les patterns[28].

Il offre une forte modularité, notamment sur la façon de gérer son templating, ainsi que le lien entre les modèles et les vues. Si vous souhaitez utiliser un modèle, libre à vous mais ce n'est pas obligatoire.

Les principes fondamentaux de Backbone.js sont :

- Beaucoup de liberté sur l'implémentation du modèle MVC ;
- La possibilité de connecter différents systèmes de templating ;
- Peu de magie et une maîtrise importante de son application.

La courbe d'apprentissage est assez douce et on parvient rapidement à des résultats. Obtenir une application bien structurée et maintenable demandera en revanche une certaine expérience.

## Lexique du framework

### Modèle

Le modèle est la partie de l'application responsable de l'orchestration des données et des règles métiers. C'est également lui qui se chargera de déclencher les interactions avec la source de données. Enfin il émet des évènements en fonction des changements de données.

### View

28_ Pattern : En français patron d'architecture ou patron de conception, solution utilisable dans la conception des logiciels (voir https://en.wikipedia.org/wiki/Design_pattern) . Exemple : la structure MVC est un pattern d'architecture technique.

La vue est chargée de gérer l'interface graphique, que ce soit de l'afficher ou de réagir aux interactions de l'utilisateur. Elle récupère les données depuis le modèle, écoute les événements qu'il envoie et lui communique les actions de l'utilisateur susceptibles de le faire évoluer.

### Collections

Une collection permet de gérer et d'agréger plusieurs modèles liés. Elle facilite à la fois leurs interactions internes, mais aussi leurs échanges avec la vue.

## Typologie de projets

Backbone.js brille surtout par ses choix classiques, simples et souples. Si vous souhaitez parfaitement maîtriser votre application et conserver une architecture MVC classique alors Backbone.js peut se justifier. En revanche, il est important de prendre en compte la popularité en baisse du *framework* pour jauger de la difficulté pour les équipes techniques à trouver de la documentation et de l'aide.

## Projets utilisant Backbone.js

- DocumentCloud ;
- Centreon ;
- Playstation Store ;
- BNP Paribas.

## Pour aller plus loin

- Backbone fundamentals - Livre consultable en ligne ;
- Ressources officielles – Tutoriels, articles, exemples.

# CE QU'IL FAUT RETENIR

Il est important de distinguer les *frameworks* en fin de vie et ceux qui sont soit en pleine maturité, soit en essor. Sauf contraintes techniques fortes, nous privilégions React ou Angular 2 dans le cadre d'un nouveau projet, tout simplement parce-que ce sont les deux *frameworks* avec les communautés les plus actives.

Il faut simplement garder à l'esprit que Angular 2 est plus adapté à des équipes moins familières avec JavaScript, alors qu' il faut une bonne connaissance de JavaScript et du DOM pour être serein avec React. Ce dernier nécessite plus de temps et de compétences en architecture logicielle afin d'assembler différents éléments créant un ensemble cohérent, tandis qu'Angular 2 est plus rigide mais plus complet et dirigiste dès le départ.

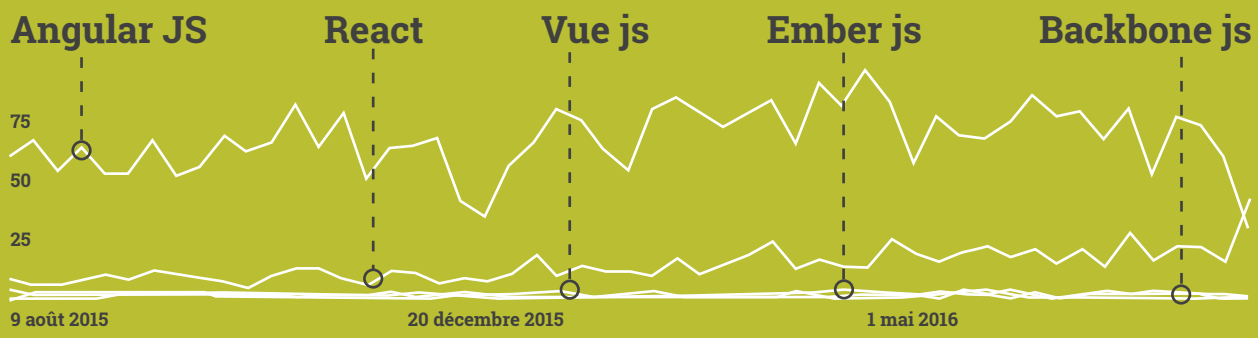
AngularJS 1 est en train d'être remplacé par Angular 2. Un projet à maintenir peut cependant vivre encore quelques années et la migration peut se faire progressivement.

La communauté et l'engouement croissant de Vue.js peuvent séduire les *early adopters* du développement front-end. Mais attention à la pérennité du projet très jeune, maintenu par une petite société.

Backbone.js et Ember.js sont tous les deux des *frameworks* assez vieux. Leurs développements sont stables mais plus lents. Leurs communautés sont très présentes et les questions sur les forums et autre Stack Overflow assez fournies. C'est un choix de raison. Ember.js aura l'avantage d'accompagner plus sereinement les développeurs back-end qui découvrent le front-end. Backbone.js aura l'avantage de la liberté de composition de la pile technique.

| Critère | AngularJS 1 | Angular 2 | React | VueJS | EmberJS | Backbone |
|---|---|---|---|---|---|---|
| Etoiles GitHub | **51k** | **14k** | **46k** | **24,6k** | **16k** | **25k** |
| Questions Stack Overflow | **190k** | **14k** | **20k** | **1,5k** | **19,5 k** | **20k** |
| Année de sortie | **2010** | **2016** | **2013** | **2014** | **2011** | **2010** |

*Recherches par framework sur google*



# L'environnement du développement front-end

Le développement Web devenant de plus en plus complexe et diversifié, il est aujourd'hui primordial pour un développeur front-end de se munir d'un environnement adapté.

Gestion des dépendances, vérification de la syntaxe, analyse du typage, couverture par des tests, intégration continue, déploiement continu… dans le monde front-end, il y a autant d'outils que de *frameworks*. Il n'est alors pas toujours simple de choisir celui qui nous convient et qui peut convenir à son projet, voici ci-dessous quelques conseils.

## LES ÉDITEURS DE TEXTE

Afin de faciliter la lecture du code, nous choisirons un éditeur adapté qui saura répondre aux besoins suivants :

- la coloration syntaxique adaptée au langage (HTML, CSS, JS, JSX, etc.) ;
- la vérification du code ;
- l'indentation automatique ;
- l'auto-complétion.

Quelques exemples d'éditeurs adoptés par les développeurs front-end :

- Atom ;
- Brackets ;
- Sublime Text ;
- vim/neovim ;
- Visual Studio Code ;
- WebStorm.

Les éditeurs ne sont pas tous des Environnements de Développement Intégré (IDE) et il faudra parfois leur adjoindre quelques plugins. Tous les plugins présentés ci-dessous sont disponibles sur les éditeurs listés ci-dessus.

## editorconfig

La première chose à faire est de mettre en place un fichier .editorconfig. Ce fichier permet de définir la configuration par défaut de l'éditeur de texte et d'uniformiser le code partagé au sein de l'équipe (taille d'indentation, espace ou tabulation, encodage, retours à la ligne, etc.).

Natif chez certaines éditeurs, disponibles sous la forme d'un plugin pour d'autres et non spécifique au développement front-end, ce fichier est devenu un pré-requis indispensable.

## JavaScript et les linters

JavaScript est un langage interprété et les erreurs de syntaxe peuvent entraîner des bogues. Par exemple, bien qu'ajoutés par l'évaluateur JavaScript, les points virgule en fin d'instructions ne sont pas obligatoires si l'instruction suivante ne commence pas par une parenthèse ou un crochet[29], mais cela peut entraîner des erreurs difficiles à comprendre[30] .

Pour palier ces problèmes avant l'exécution du script, la communauté a adopté les linters[31]. Ils sont aujourd'hui indispensables à tout projet front-end, la référence actuellement étant eslint.

Les linters permettent de vérifier la syntaxe, de la valider au sein de l'équipe de développement et d'anticiper certaines risques en pointant  par exemple des variables non déclarées ou non utilisées.

# NPM

NPM (Node Package Manager) est le gestionnaire de paquet officiel de Node.js. Il s'utilise en ligne de commande pour ajouter des bibliothèques JavaScript à un projet.

NPM s'utilise avec un fichier de configuration, package.json, dans lequel :

- on peut décrire le projet (nom, url du dépôt, licence, auteur, etc.) ;
- on peut scripter des tâches (*build*, *minifier*, *linter*, etc.) ;
- on peut référencer toutes les bibliothèques nécessaires au projet.

29_ Voici un long article sur les point-virgules http://mislav.net/2010/05/semicolons/

30_ Finalement c'est peut-être mieux de conserver les points-virgules https://davidwalsh.name/javascript-semicolons

31_ Les linters sont des commandes permettant l'analyse syntaxique. (voir https://fr.wikipedia.org/wiki/Lint_(logiciel))

NPM a rendu l'utilisation du fichier package.json indispensable pour tout projet JavaScript.

Le fichier package.json est le point d'entrée d'un projet de développement front-end.

# LES GESTIONNAIRES DE TÂCHES (OU TASKRUNNERS)

Les tâches redondantes ont aussi leurs outils. Configurables à souhait grâce à leurs milliers de plugins, les *taskrunners* viennent suppléer les outils d'intégration continue pour automatiser la vérification du code, le build, la couverture par les tests, l'optimisation des fichiers, le déploiement, etc.

## Les principaux gestionnaires

- Grunt : très connu et très utilisé il y a quelques années, il l'est aujourd'hui un peu moins au profit d'outils plus performants et plus rapides.
- Gulp : il utilise les flux – et n'écrit pas chaque modification des fichiers sur le disque – ce qui le rend plus rapide que Grunt, à privilégier.
- Webpack : très utilisé dans la communauté React et Angular 2 – grâce à son plugin de rechargement à chaud (*hot reload*) – permet d'empaqueter l'application (*bundle*) avec une approche modulaire assistée par les *loaders*.

Ces gestionnaires permettent de configurer des tâches pour le projet tels que l'empaquetage, la transpilation ou encore la minification[32].

Afin d'avoir une vision globale des tâches disponibles dans un projet, nous reportons l'ensemble de celles-ci dans la partie scripts du fichier package.json.

# LES TESTS

Différents types de tests peuvent être effectués sur les projets front-end comme les tests unitaires, les tests fonctionnels ou encore les tests d'intégration.

Les tests unitaires en JavaScript ne fonctionnent pas différemment des tests unitaires avec d'autres langages. Muni d'une bibliothèque d'assertion, nous testons la logique métier de chaque fonction de notre application.

32_ Toutes ces tâches sont très classiques de nos jours (voir chapitre "Le fonctionnement d'une application front-end") : nous écrivons plusieurs fichiers JavaScript lors des phases de développement ; mais pour la livraison, afin d'alléger le réseau, nous ne faisons qu'un seul gros fichier JavaScript (empaquetage) et nous réduisons les noms des variables (minification). Toutes ces tâches peuvent être rediscutées avec l'arrivée de HTTP/2 et des évolutions de JavaScript.

Pour les tests fonctionnels – ou *end-to-end* – il y a des spécificités. Une application front-end repose sur le DOM et attendre des interactions utilisateurs comme le clic ou la saisie clavier. Réaliser un test fonctionnel doit rendre compte de l'ensemble des actions et comportements exécutés après une interaction d'un utilisateur : asynchronisme, tests d'interface graphique.

Par exemple : lorsqu'un utilisateur ajoute un commentaire, est-ce que celui-ci s'affiche et est-ce que le formulaire se réinitialise ?

De nombreux outils sont alors à notre disposition pour mettre en place ce type de test, parmi les plus connus : Selenium, PhantomJS, CasperJS, Robotframework, Protractor, jsdom.

Afin de s'assurer qu'une application est toujours opérationnelle lors des modifications du code et surtout dans le cadre de gros projets, il est nécessaire d'automatiser ces tests.

# L'aspect mobile

Le développement front-end n'a pas pour unique cible l'affichage sur un écran d'ordinateur. La part des accès mobiles est grandissante et il n'est pas envisageable d'en faire l'impasse.

Les terminaux mobiles, encore plus que les écrans d'ordinateur, ont **différentes tailles et résolutions**. Nous devons alors proposer différentes interfaces graphiques pour adapter l'application à la largeur du terminal de consultation.

La taille de l'écran n'est pas la seule spécificité du mobile, nous devons aussi prendre en compte les problèmes ou l'**absence de connectivité**. Pour remédier à cela, nous pouvons utiliser les Services Workers ou envisager une Progressive Web App.

## RESPONSIVE WEB DESIGN

Le Responsive Web Design, ou RWD, ou Responsive, a pour but de rendre accessible l'application et son contenu quelque soit le support de consultation utilisé. C'est l'idée de l'adaptabilité.

Bien que pouvant s'appliquer à d'autres supports comme des Google Glass ou une interface holographique, le RWD est utilisé actuellement surtout pour rendre le contenu fluide et adapté qu'on le consulte sur un ordinateur de bureau, un ordinateur portable, une tablette, ou un mobile. Pour en comprendre les contours, voici les distinctions que l'on fait des différentes présentations de contenu.

### Statique vs adaptative vs responsive

On distingue trois façons de gérer le contenu et sa mise en page.

- Mise en page **statique** : Les éléments ont une taille fixe. Autant dire que c'est exactement ce qu'il ne faut pas faire aujourd'hui. Notre application ne serait consultable de manière optimale que pour une seule taille d'écran. À proscrire. (Exemple : Ma barre latérale fait toujours 300 pixels de large)

- Mise en page **adaptative** : On crée des points de ruptures arbitraires permettant de redimensionner le contenu. On utilise ce que l'on appelle des media queries pour définir ces points de rupture. (Exemple : Ma barre latérale fait 120 pixels de large si la largeur de l'écran est plus petite que 320 pixels. Sinon elle fait 300 pixels)
- Mise en page ***responsive*** : On conserve l'idée de points de rupture, mais le contenu devient entièrement fluide et occupe l'espace de manière plus optimale. (Exemple : Si l'écran fait plus de 320 pixels de large, ma barre latérale prendra 30% de la largeur de l'écran. Ainsi elle peut aller au-delà de 300 pixels de large, mais on conserve une proportion.)


*Différence entre mise en page responsive et adaptative*

## Les avantages et inconvénients du responsive

Avantages :

- L'accessibilité de notre contenu. N'est-ce pas le plus important ?
- Un RWD bien fait sera toujours plus pérenne et maintenable.
- Une interface unique pour mobile et desktop (et donc moins de développement).

Inconvénients :

- Le RWD demande à l'équipe de design d'imaginer 3 interfaces pour une meilleure adaptation des fonctionnalités par rapport au support.
- Le RWD demande un peu plus d'efforts et de temps à l'intégrateur, surtout si celui-ci n'a pas l'habitude de travailler avec des unités relatives.
- Il peut être difficile pour l'équipe de design de représenter ce comportement dans les maquettes.

## Pourquoi le Responsive pour notre application web ?

Comme nous venons de l'exposer, il semble difficile aujourd'hui de ne pas se positionner sur le marché mobile. Qu'il s'agisse des tablettes ou des mobiles, il y a de fortes chances que nos utilisateurs aient recours à notre plateforme à travers ces périphériques.

Grâce au Responsive, nous rendons notre contenu accessible partout à un coût de développement moindre.

# RESSOURCES

- liquidapsive.com : site démontrant les différences entre les mise en page statiques, adaptative et Responsive.

# LE MODE HORS-LIGNE OU DÉGRADÉ

Lorsqu'une application front-end est exécutée sur un mobile, il est courant de se retrouver dans des conditions de connexion dégradée, voire sans connexion du tout.

Les applications natives, développées grâce aux SDK des différentes plateformes iOS, Android ou Windows, proposent des solutions pour gérer la connectivité. Quid d'une application Web front-end ?

Par défaut, lorsque l'utilisateur tente d'accéder à une application Web alors qu'il est hors-ligne, il reçoit un message d'erreur de son navigateur. Il existe des solutions pour améliorer l'expérience utilisateur en gérant les accès hors-ligne.

## Le stockage

Les API Web fournissent des solutions de stockages[33] comme les *cookies*, *LocalStorage*, *File API* ou *IndexedDB*.

D'autres solutions via des bibliothèques permettent également de stocker des données pour les accès hors-ligne comme localForage ou *pouchDB*.

L'espace de stockage d'un terminal mobile est souvent restreint, il faut donc être vigilant aux données à installer sur la machine

33_ Pour en savoir plus, découvrez l'article suivant : https://medium.com/dev-channel/offline-storage-for-progressive-web-apps-70d52695513c

## Service Worker

Les navigateurs les plus récents supportent maintenant la mise en place de *Service Worker* pour aider l'application front-end à fonctionner sans connexion ou avec une connexion dégradée.

Développés en JavaScript, les *Service Workers* fonctionnent en parallèle du navigateur. Lorsque l'application va être chargée la première fois, si un *Service Worker* est présent dans la page Web, il est installé dans l'ordinateur ou le téléphone du visiteur.

Le *Service Worker* peut agir tel un proxy entre le navigateur et la connexion réseau. Il permet une expérience hors-ligne en interceptant les requêtes et en prenant des actions en fonction de l'état de la connexion comme servir certaines données depuis la Cache API.

Le Service Worker tournant en dehors du navigateur, en tâche de fond, permet également de gérer l'envoi de notifications à l'utilisateur.

# PROGRESSIVE WEB APPS

Les Progressive *Web Apps* (PWA) sont un ensemble de pratiques et de technologies qui permettent de créer des applications Web pour lesquelles l'expérience utilisateur se rapproche des applications mobiles natives, notamment grâce aux Service Workers.

Autrement dit, ce sont des pratiques qui prennent en compte :

- l'installation d'une icône sur l'écran d'accueil ;
- l'affichage d'une page de garde (*splash screen*) ;
- un chargement instantané[34], peu importe l'état de la connexion ;
- des animations à haute performance[35] (60 fps) ;
- des notifications même si l'app n'est plus ouverte.

Bien que les PWA soient très prometteuses, elles s'appuient sur des technologies récentes (pas encore compatibles sur tous les navigateurs) et peu d'exemples sont disponibles à l'heure où ce livre est écrit.

34_ Notamment grâce à la mise en place de l'App Shell ou coquille de l'application. Ils sont l'interface, les icônes, les logos, les polices de caractères.

35_ En utilisant l'API Web Animations le navigateur met à disposition son propre moteur d'animations. L'animation peut ainsi être déléguée au processeur graphique de la machine, mobile ou fixe, assurant des performances élevées.

# Conclusion

La synthèse du chapitre frameworks constitue donc un bon point de départ pour choisir la technologie qui vous convient le mieux. Mais au-delà d'un choix purement technique par rapport à la typologie du projet, il faudra s'attacher à choisir une technologie qui sera facilement adoptée par l'équipe de développement. Une erreur classique consiste à croire qu'un développeur Web qui écrivait un peu de JavaScript pour animer quelques boutons peut maitrîser l'ensemble du langage JavaScript et tous les rouages du front-end. C'est un piège dangereux, et nous recommandons qu'une équipe se lançant dans le développement front-end soit au préalable formée afin de partir sur les bonnes bases et être performante dès le lancement (gain de temps…).

Pour une meilleure montée en compétence, nous mettons souvent en œuvre une solution d'accompagnement de projets : dans les premières itérations de développement, nous composons une équipe mixte d'experts Makina Corpus et de développeurs de notre client ; le développement se fait ainsi conjointement. Les bonnes pratiques sont mises en place dès le départ et la transmission de connaissances se fait sur des cas réels rencontrés sur le projet. Rapidement, l'équipe cliente devient autonome et prend les commandes du projet.

Concernant l'équilibre entre front-end et back-end au sein d'un projet, il faut garder à l'esprit que la partie front-end peut peser plus que la partie back-end en termes de charge de travail (et donc de coût), et peut faire l'objet d'enjeux et de risques plus grands.

Quelle que soit la qualité ou la part du back-end sur l'ensemble, la partie front-end reste la seule chose que perçoit l'utilisateur final. Aussi un mauvais développement front-end condamnera définitivement le succès du projet.

C'est donc une partie sur laquelle il est vital d'affecter un encadrement fort et de ne pas se contenter de développeurs juniors.

Comme montré tout au long de ce livre blanc, l'écosystème JavaScript est à la fois complexe et changeant. Les outils du développeur front-end sont remis en question très régulièrement.

Un sondage récent, réunissant plus de 9 300 développeurs à travers le monde, montre la perte de repères vécue à la fois par les néophytes et les développeurs plus aguerris face au choix de la bonne technologie front-end.

Nous espérons vous avoir donné une vision synthétique des principaux enjeux auxquels vous serez confrontés.

Pour aller plus loin, Makina Corpus propose de vous accompagner tout au long de vos projets, grâce à nos audits, nos formations et nos prestations de développement.

N'hésitez pas à nous transmettre vos avis et critiques sur ce livre blanc en nous écrivant à contact@makina-corpus.com.

# Glossaire

## Accessibilité

Ensemble des pratiques et techniques permettant à tout le monde d'accéder à du contenu.

## AJAX

Ensemble de technologies utilisées par JavaScript pour mettre à jour une partie d'une page web.

## Application programming interface (API)

Interface de communication entre deux machines, souvent entre deux langages ou implémentations.

## Application hybride

Application mobile écrite avec des technologies web, puis livrée dans un magasin d'applications comme Play Store ou App Store.

## Application monopage (single page application)

Site web dont le routage (la navigation) est fait avec JavaScript.

## Architecture

Quelle soit logicielle ou matérielle, c'est l'organisation des différentes structures d'un système informatique.

## Back-end

Ensemble des logiciels qui fonctionnent sur un serveur web.

## Bibliothèque (library)

Ensemble de fonctionnalités logicielles mise à disposition dans un même module.

## Command line interface (CLI)

Interface d'un logiciel qui se pilote via une ligne de commande.

## Compilation/transpilation (build)

Étape de transformation d'un code source vers un langage adequat pour un processeur ou un interpréteur.

## CSS

Cascading Style Sheet, feuille de style à effets de bord dans laquelle on décrit le style d'une page web.

## Data binding

Association entre le modèle d'une donnée et sa représentation.

## DOM

Document Object Model, interface permettant à des programmes informatiques d'accéder au contenu et à la structure d'un document HTML ou XML.

## Environnement d'exécution

Logiciel responsable de l'exécution d'un autre programme informatique écrit dans un langage de programmation donné.

## Environnement de développement intégré (IDE)

Suite logicielle pour la création de programmes informatiques, intégrant un éditeur, un compilateur et un environnement d'exécution.

## Ex nihilo

À partir de zéro, from scratch.

## Expérience utilisateur (UX)

Reflète l'expérience totale d'une personne utilisant un produit, un système ou un service particulier.

## Framework

Ensemble de bibliothèques empaquetées pour fournir un cadre plus ou moins rigide à la création d'un programme.

## Front-end

Ensemble des logiciels qui fonctionnent sur un navigateur web.

## Gestionnaire de paquets

Outil permettant de trouver, télécharger et mettre à jours des bibliothèques.

## Hors-ligne (offline)

État d'un ordinateur non connecté à internet.

## Hot reload

Technologie permettant la mise à jour d'une partie d'une page sans tout recharger lors de l'étape de développement.

## HTML

HyperText Markup Language, est le format de données conçu pour représenter les pages web.

## Implémentation

Formulation des instructions et des fonctionnalités en terme de code source et d'organisation du code source.

## Intégration continue

Ensemble de pratiques consistant à vérifier que chaque modification du code source ne produit pas de régression dans l'application développée.

## Intégration graphique

Implémentation d'une charte graphique ou d'une création graphique.

## Internet

Réseau informatique mondial reliant les ordinateurs ou des réseaux publics, privés, universitaires, commerciaux et gouvernementaux.

## Interpréteur

Logiciel permettant de lire et d'exécuter du code source d'un langage donné.

## Interpréteur/évaluateur JavaScript

Interpréteur spécifique au JavaScript comme Node.js, V8, Rhino ou SpiderMonkey.

## JavaScript

Langage informatique créé par Brendan Eich interprété par les navigateurs web notamment.

## JSON

JavaScript Object Notation est un format de données textuelles dérivé de la notation des objets du langage JavaScript.

## Lien hypertexte
Lien permettant de passer d'un document consulté à un document lié.

## Linter
Vérificateur syntaxique d'un langage informatique.

## Minifier
Outil permettant de réduire le code source d'un langage interprété.

## Navigateur web (navigateur internet)
Logiciel conçu pour consulter et afficher les pages web.

## Node.js
Plateforme logicielle basée sur l'interpréteur JavaScript V8.

## Package.json
Fichier de description d'une bibliothèque JavaScript ou d'une application JavaScript.

## Progressive web app
Ensemble de technologies mises en œuvre pour fournir des applications web équivalent en terme d'expérience utilisateur à une application native.

## Responsive design
Technologie permettant de proposer un design qui s'adapte au contenant : écran d'ordinateur, grand écran, écran de smartphone ou de montre.

## REST API
REpresentational State Transert API, style d'architecture pour les applications web.

## Service worker
Technologies permettant à un navigateur web d'interpréter du code JavaScript alors que la page web n'est pas chargée.

## Synchronimse/asyncrhonisme
Caractère de ce qui se passe en même temps (syncrhonisme) ou non (asynchronisme).

## Système d'Exploitation (OS)
Ensemble de logiciels qui dirige l'utilisation des ressources matérielles d'un ordinateur.

## Taskrunner

Ensemble de logiciels qui exécute des tâches (pour la validation, le build d'un projet informatique) décrites dans un fichier de configuration.

## Test fonctionnel

Test automatisé qui vérifie le bon fonctionnement d'une partie d'un logiciel.

## Test unitaire

Test automatisé qui vérifie le bon fonctionnement d'une partie d'un code source.

## Vanilla JS

Code JavaScript pur, sans bibliothèque tierce, ni framework.

## Vue/modèle/contrôleur/vue-modèle

Architecture logicielle 3-tiers séparant les données (modèle), du comportement (contrôleur ou vue-modèle) et de la présentation (vue).

## Web

World Wide Web est un système de lien hypertexte public qui fonctionne sur internet.

## Web API

Ensemble des interfaces permettant de manipuler les ressources fournit par un navigateur lors de la consultation d'une page web.

## Web app

Site web particulier fournissant au visiteur les fonctionnalités d'une application native.

# Les auteurs

## ALEXANDRA JANIN

Développeuse Front-End particulièrement à l'aise avec AngularJS, elle aime mettre la technique au service de l'esthétique et de l'expérience utilisateur.

## BASTIEN ALVEZ

Développeur Front-End Mobile aussi intéressé par AngularJS que par React, il aime le souci du détail qui rend l'expérience utilisateur agréable.

## BENJAMIN MARGUIN

Développeur Front-End Senior adorant le Vanilla JavaScript, il aime trouver des solutions.

## ENGUERRAN COLSON

Développeur Web Senior expert React, il aime réaliser des applications web facilement accessibles quelque soit le contexte comme les Progressive Web Apps.

## ÉRIC BRÉHAULT

Directeur technique et développeur Senior séduit par Python et JavaScript, il aime développer, coder et s'impliquer dans la communauté Open Source.

## MATHIEU DARTIGUES

Développeur JavaScript amateur de NodeJS et VueJS, il aime être en contact avec les utilisateurs et savoir qu'ils sont satisfaits.

## SIMON BATS

Développeur Front-End aux multiples compétences JavaScript (React, AngularJS, Angular 2), il aime se challenger à travers la création d'applications web performantes.

Toulouse, Nantes, Paris, Bruxelles
+ 33 (0)9 53 73 22 74
makina-corpus.com
contact@makina-corpus.com
@makina_corpus