

TECH/TRENDS

#5

DÉCEMBRE
2014

PERCEVOIR LE FUTUR

FRONT



Maquetter

Créer

Industrialiser



Maquetter




Créer



Industrialiser



Introduction



Ces dernières années, les applications informatiques ont grandement évolué. Initialement pensées pour une utilisation desktop, leur développement était exclusivement centré sur le serveur (Back-end), le navigateur de l'utilisateur n'étant qu'une « extension » servant à l'affichage.

L'apparition et la démocratisation de nouveaux vecteurs d'informations (smartphone, tablette, TV connectée, etc.), l'amélioration continue du matériel et des connexions internet et la montée en puissance de nouveaux navigateurs (Chrome, Firefox) sont venues bouleverser les codes. Ainsi, une nouvelle « branche » des projets Web est apparue : les applications Front-end, pour lesquelles le serveur n'est qu'un vecteur d'information parmi d'autres, et pour qui le navigateur est un réel moteur d'exécution.

De plus, la multiplication des points de contact avec l'utilisateur final a ouvert le champ des possibles ; la conception et la diffusion des produits et services de l'entreprise sont aujourd'hui une discipline à part entière.

Ces évolutions de l'écosystème Web amènent, comme souvent, une complexité accrue. Les processus de création, de développement et d'évolution des interfaces sont devenus plus exigeants, et font l'objet d'études spécifiques en amont des projets « classiques ».

Comme tout changement dans le monde informatique, ce dernier s'accompagne de l'apparition de nouveaux outils et méthodologies pour penser, architecturer et industrialiser ces nouvelles interfaces et les intégrer totalement dans le cycle de création d'un projet d'entreprise. Les nouveaux outils ont une vitesse d'apparition bien plus élevée que leur vitesse de maturation, il est donc souvent difficile de dégager un standard sur lequel s'appuyer.

Ce TechTrends, à travers la gestation d'une application Web - maquettage, développement et industrialisation - tentera de vous aiguiller dans ce dédale et de pointer les technologies d'aujourd'hui et de demain afin de créer des interfaces modernes, ludiques et interactives fluidifiant l'accès des utilisateurs à l'information.

Maquetter



La réflexion générale sur le design,

l'interactivité, l'ergonomie d'une application et leurs impacts sur l'utilisateur sont généralement désignés sous le terme d'expérience utilisateur ou User eXperience (UX). Créer et maintenir une expérience client de qualité est un domaine d'expertise à part entière, qui nécessite de multiples compétences. Investir dans une bonne UX, c'est capitaliser sur la satisfaction qu'éprouvera votre utilisateur futur et assurera sa fidélité.

Pour cela, la clé du succès est de placer l'utilisateur au centre de toute réflexion.

IDENTIFIEZ VOS UTILISATEURS

La réussite d'une application repose fondamentalement sur une identification précise et pertinente de ses utilisateurs. Il est inenvisageable de commencer le moindre développement d'un site sans connaître la population visée. Appelés personas, ces utilisateurs cibles guideront toutes vos décisions de conception d'interfaces graphiques et d'organisation de votre application.

Des études marketing, des séries d'interviews ou des données issues de votre propre système d'information vous permettront de définir des critères démographiques, socio-professionnels ou culturels propres à chacun de vos personas. Formalisez une fiche d'identité et rendez-les réels en définissant un nom, un sexe, un métier et une courte biographie :



SOPHIE

« Je suis
raisonnée et
pragmatique »

26 ans - Responsable de magasin

CONTEXTE D'UTILISATION

Utilisation
quotidienne sur
son lieu de travail.
Sophie utilise l'outil
de gestion depuis
son arrivée en
poste il y a 3 ans.

VALEURS

Terre-à-terre et
perfectionniste

CONFORT AVEC LA TECHNOLOGIE

Sophie n'a aucune
affinité particulière
avec la technologie.

L'OBJECTIF DE SOPHIE

Augmenter son efficacité personnelle

SES BESOINS



Minimiser son temps de
saisie lorsqu'elle utilise
son outil.

Disposer d'une vue
globale sur l'ensemble
des paramètres.

SES FREINS



Elle n'aime pas utiliser
des interfaces qu'elle
ne connaît pas.

SON ÉQUIPEMENT



Ordinateur de travail
(Windows XP)
Mobile personnel
(Samsung galaxy, Android)

SES TÂCHES



Manager les équipes
en rayon.
Mettre à jour les quantités
restantes en fin de journée.
Passer les commandes
pour approvisionnement.

Fiche d'un persona

Vous aurez besoin de plusieurs personas, certains seront plus importants que d'autres et seront qualifiés de Primaires, Secondaires et Tertiaires. Chaque persona a une relation particulière avec l'outil informatique, liée à son expérience, ses attentes et ses objectifs. C'est pourquoi il est important de changer de point de vue régulièrement dans chaque étape de conception et de prendre la place de chacun des personas.

DÉTERMINEZ LES PARCOURS UTILISATEURS

Pensez expérience plutôt qu'interface !

Il est tentant de vouloir se pencher immédiatement sur l'aspect visuel d'un site ou d'une application mais ce serait une gageure. Tout d'abord parce que vos utilisateurs désirent avant tout répondre à un besoin. Attardez-vous sur la liste des cas d'utilisation auxquels il vous faudra répondre efficacement. Vous concevez un site e-commerce ? Certains de vos utilisateurs viendront pour comparer des prix et ne passeront pas par le tunnel d'achat classique, d'autres savent déjà quel produit ils recherchent et veulent aller vite. Il faut donc penser à tous les cas de figure afin de ne pas perdre l'utilisateur dans les méandres du site au risque de manquer des ventes.

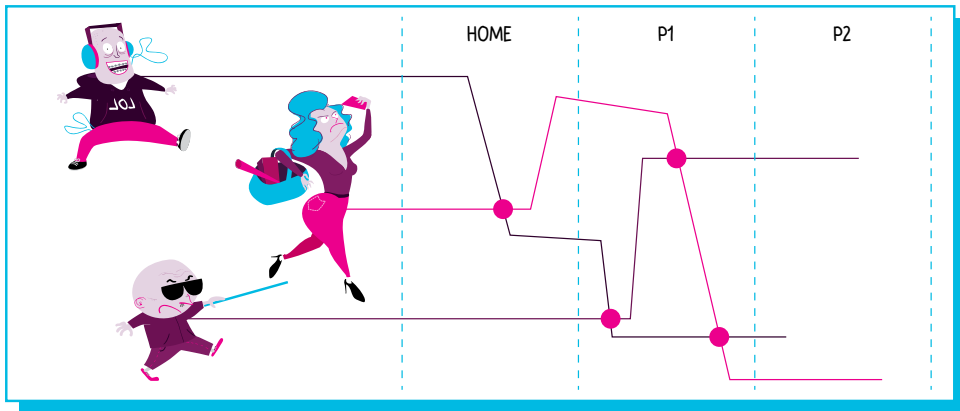
Vous l'aurez compris, nous parlons de parcours utilisateurs différents pour chacun des personas. Ces parcours sont comparables à ceux définis dans le cadre de la grande distribution. Les rayons des supermarchés ne sont pas disposés au hasard mais découlent d'une longue série d'expérimentations, d'analyses et de conclusions : une véritable science du comportement en soi. Il en va de même lors de la conception d'un site ou d'une application.

Dessinez l'ensemble des parcours, superposez-les, étudiez les embranchements possibles : les points d'entrée, les points de sortie, les impasses. A ce stade, vous obtiendrez la liste des pages ou des grandes fonctionnalités attendues. Il s'agira alors de tout organiser en un ensemble cohérent.



Il est tentant de vouloir se pencher immédiatement sur l'aspect visuel d'un site ou d'une application mais ce serait une gageure. Tout d'abord parce que vos utilisateurs désirent avant tout répondre à un besoin.





Étapes de création d'une interface utilisateur

L'étude des parcours doit faire émerger une liste de contenus attendus par l'utilisateur. Dans le cas d'une fiche produit, cela pourrait se résumer aux éléments tels que le nom, le prix, une description, une photographie. Rassemblez ces informations dans un document, ne vous souciez pas de leur agencement ou de leur style, pensez « contenu » !

ORGANISEZ VOTRE CONTENU

Maintenant que les points clés sont connus, il est temps de se concentrer sur la disposition du contenu dans les différentes interfaces. Nous parlons cette fois-ci de la forme :

- Quels seront les canaux de diffusions ?
- Quelles informations doivent être affichées, à quel moment ?
- Dans quel ordre ?
- Quelles sont les actions possibles ?
- Que faut-il mettre le plus en avant ?

Les réponses à ces questions dépendent d'un élément très important : l'écran de votre utilisateur. Votre produit doit-il être manipulé depuis un smartphone, une tablette, un ordinateur ou même une borne tactile verticale ? Vous avez probablement entendu parler de Responsive Web Design ou Site Web Adaptatif pour répondre au besoin de redimensionnement automatique des interfaces graphiques selon la surface d'affichage. L'indéniable avantage de cette solution est le développement d'un seul projet, accessible sur tous les supports quelque soit la plate-forme : un argument très séduisant, surtout en terme de coûts, de temps, de développement et de maintenance. C'est aussi la garantie d'avoir une charte graphique unifiée et une expérience utilisateur homogène que ce soit sur ordinateur, mobile ou tablette.

Cependant, au-delà d'une solution technique, c'est tout un mode de pensée qui doit dorénavant être adopté : le « Mobile First ». Les interfaces à produire sont vouées à être épurées, affichables et utilisables sur une surface réduite.

Des ateliers UX de Sketching tel que le Six-to-One permettent de répondre à ces questions. Les participants doivent imaginer six gabarits pour chacun des écrans, proposer des représentations différentes et les présenter par la suite aux autres participants avant d'entamer la deuxième étape. Sur la base de tous les gabarits fournis, il faut piocher les bonnes idées pour aboutir à une représentation unique et détaillée.

À l'issue de l'atelier, vous obtiendrez une première vision de l'interface générale qu'un UX Designer devra reproduire dans un logiciel de Wireframing comme [Axure](#) ou [Balsamiq](#). Ces outils permettent de créer un prototype très rapidement et de vérifier que la navigation entre écrans est pertinente. Pour cela, organisez des tests utilisateurs pour valider la bonne compréhension des concepts, tester le wording, etc. Impliquez des développeurs pour vous assurer que les idées sont techniquement réalisables et évaluer leur complexité.

stiles

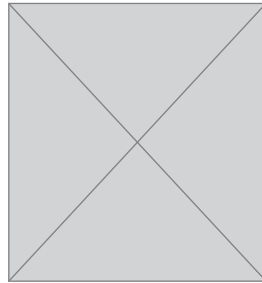
Total Production Solutions

PRODUCTS SERVICES SOLUTIONS PARTS SUPPORTS ABOUT US

PRODUCTS

Applications
Brands
Products Types

Product Selector
Specials



IRONWOOD PS 1000

Bis estio. Et elluptatius et andae mod qui dipsanto magnimp eligent audae nes re et quunder speriatiant estotaq uaspicaboris eos es reVit, volupta tiationsed ea nonet offici tem faceate eictur audant, qui vitataiae.

Request Information

View Pricing
& Technical Specs

RELATED PRODUCTS

Applications: Classical
Machining Solid Wood
Brands: Ironwood
Product Types: Shapers
Series: Artisan Shaper Series

FEATURES

- Cia nones ut molorit vololib eruptibus rerupta tempori tatat.
- Sa aut aut pro blantis trunto ommolup idicima quiderchil eiciis voloreictas ut lacidiam repelique nest,
- quiae pa delendae nonecese evel et vls mo bera destorum esto doluptae idem asitatem venti
- omnihiliquam faciendam essitat eniscia sam dolupti

PRODUCT SPECIFICATIONS

working Table Dimensions	850 mm x 1300 mm
Vertical Spindle Stroke	150 mm
Spindle Height Adjustment	Electronic
Spindle Position Display	Digital Display
Spindle Tilt	No

RELATED PRODUCTS & ACCESSORIES



Label



Label



Label



Label



Label

Première maquette de l'interface générale

Assurez-vous que les utilisateurs ne sont pas perdus, que tout s'enchaîne de façon logique, attendue et naturelle. Vous avez le droit à l'erreur et pouvez corriger facilement les interfaces avant de passer à l'étape suivante : le design.

PENSEZ VOTRE DESIGN

Cette étape, réalisée par un Directeur Artistique, permet de valider la facette émotionnelle du produit. Pour mener à bien cette mission, il devra s'inspirer de votre univers. Fournissez-lui un maximum de ressources : charte graphique, images, croquis, palette de couleurs, etc. Il créera un produit dont le design sera dans la parfaite continuité de votre marque.

En plus de fournir des maquettes utilisables par les développeurs, il pourra mettre à disposition un « Style Guide ». Cette boîte à outils contiendra l'ensemble des composants prêts à l'emploi : polices de caractères, titres, icônes et divers composants HTML.

Maintenant que vous avez des maquettes de votre site adaptées à vos clients, à leurs besoins primaires (acheter un produit, rechercher une information, etc.) ou secondaires (avoir accès à mon site n'importe où et n'importe quand) mais aussi aux supports de diffusion choisis, il est temps de créer votre site.

ETUDE DE CAS : REFONTE D'UN LOGICIEL

Voyons ensemble une étude de cas dans laquelle une entreprise fait appel à des spécialistes en User eXperience pour repenser un de leurs principaux logiciels.

Cassiopae, le logiciel de gestion de stock développé en interne depuis maintenant une quinzaine d'années, n'est pas en mesure de satisfaire la vision 2015 de l'organisation. Le rôle de l'expert UX sera de vous accompagner tout au long de la refonte. Les objectifs qui lui sont assignés sont divers :

- Repenser l'outil actuel en gardant les repères des utilisateurs.
- Animer les différents ateliers UX.
- Faire converger les idées vers des maquettes.

Notre étude de cas explicitera principalement cette première phase de reconnaissance, les 2 autres points étant étudiés lors des différentes étapes précédemment énoncées dans ce TechTrends.

Dans le cas d'une refonte de site, il est inconcevable de faire table rase du site existant au risque de désorienter les utilisateurs. Pour cela, appuyez-vous sur vos utilisateurs réguliers pour construire la nouvelle version du logiciel.

Dans un premier temps, l'expert doit comprendre comment Cassiopae est utilisée au quotidien. Pour se faire, il utilise la pratique du **Shadowing**. Comme son nom anglais l'indique, l'expert sera pour quelque temps l'ombre de l'utilisateur. Il l'accompagne durant quelques jours et regarde comment il manipule l'outil : les tâches récurrentes, les défauts d'interfaces, le nombre de clics nécessaires pour obtenir une information importante, l'utilisation de Post-It pour noter une information et la saisir dans un formulaire, etc.

Cette étape lui permet de lister les fonctionnalités primordiales de l'outil ainsi que les pertes de temps pour l'utilisateur afin de dégager les premiers axes d'amélioration tant sur l'ergonomie que sur le vocabulaire utilisé.

Ces observations sont les premières briques de l'analyse, il faut maintenant comprendre le « pourquoi » de ces pratiques. L'expert UX décide de compléter son analyse par une série d'interviews de certains utilisateurs. A travers des questions ouvertes, les personnes interviewées expliquent concrètement leurs points de blocage, les raccourcis qu'ils souhaitent voir apparaître dans cette nouvelle version, etc. Les nouvelles données recueillies vont permettre d'avancer sur la conception de l'interface mais aussi d'agréments les personas.

L'association de ces 2 ateliers UX met parfois en exergue un réel problème de navigation dans le logiciel. Afin d'y palier et de trouver une nouvelle organisation, l'expert UX lance un nouvel atelier : **le tri de carte**. Chaque participant (parties prenantes au projet et utilisateurs) note sur des cartes les termes, concepts et actions qu'il souhaite avoir sur le nouveau logiciel. Elles sont ensuite regroupées par thématique commune puis classifiées selon l'ordre d'importance. Cette pratique permet d'aboutir en fin de séance à une idée de la nouvelle arborescence du site à l'image des besoins actuels des utilisateurs, mais qui sera ensuite affinée lors des prochaines étapes.

Une fois tous ces éléments définis, la démarche UX précédemment décrite peut commencer : description des personas, définition des parcours utilisateurs, organisation du contenu, phase de conception et design des interfaces.



Dans le cas d'une refonte de site, il est inconcevable de faire table rase du site existant au risque de désorienter les utilisateurs. Pour cela, appuyez-vous sur vos utilisateurs réguliers pour construire la nouvelle version du logiciel.



Take away Front trends



MAQUETTER

- *Mettre vos utilisateurs au centre de vos réflexions au travers des personas.*
- *Organiser votre contenu en fonction de vos parcours clients.*
- *Adapter votre design aux canaux de diffusion choisis.*

Créer



Lors de la création d'une interface,

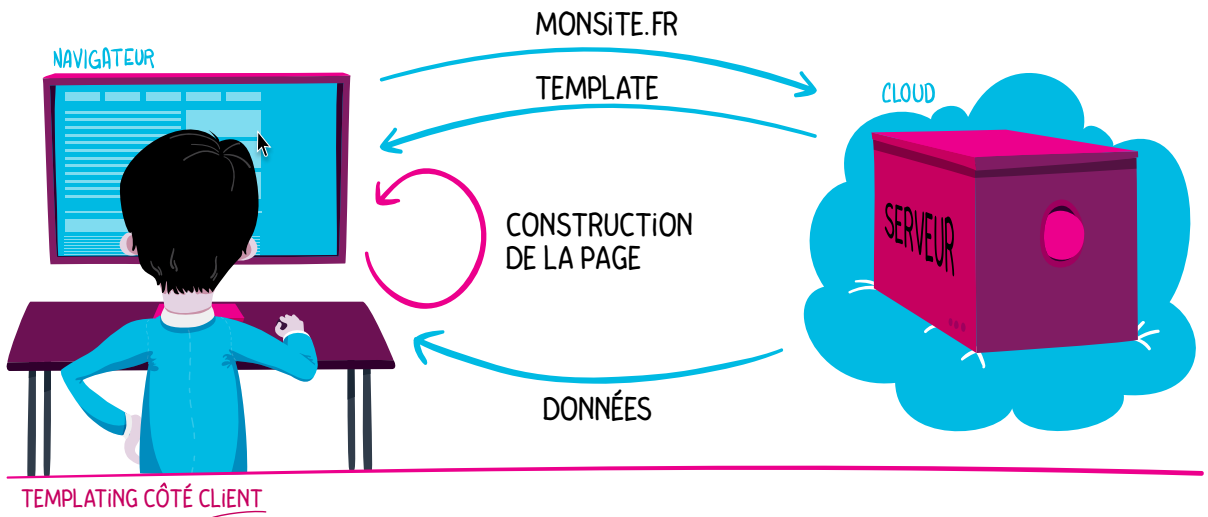
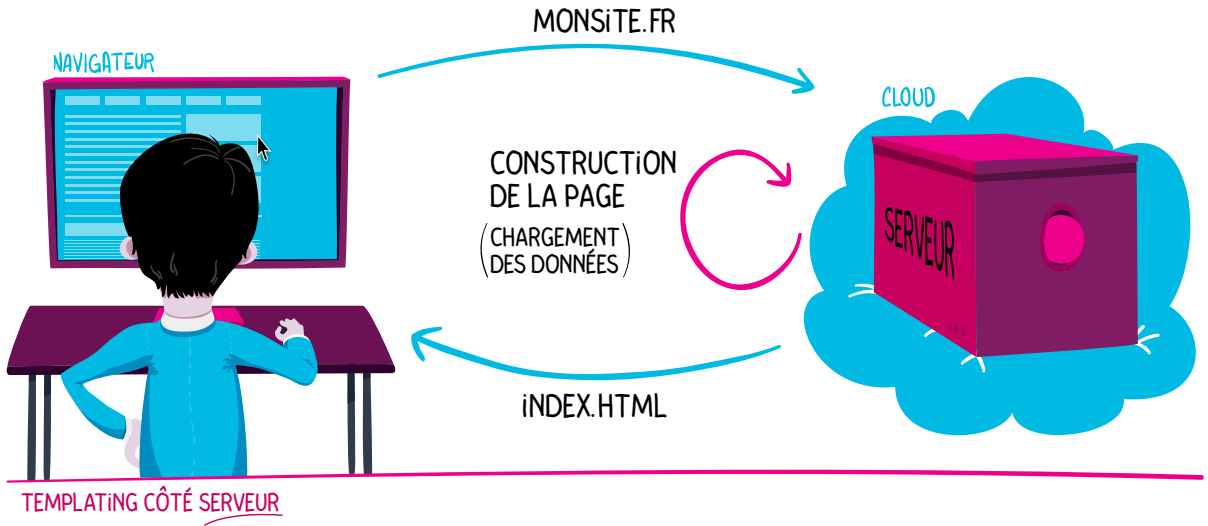
des choix doivent être faits, tant au niveau technologique et architectural que de l'outillage. L'univers du «Front» est un écosystème très dynamique et mouvant, où de nouvelles solutions apparaissent chaque jour. Il est donc essentiel de savoir faire le tri entre le simple effet de mode et une révolution. Ce sera à vous et à vos équipes de déterminer la solution la plus adaptée à vos besoins.

LA RÉVOLUTION HTML

Aller au front ou rester en arrière ?

Un site Web est constitué d'un ensemble de ressources demandées par un navigateur client à un serveur. Parmi ces ressources, on retrouve la page HTML, le résultat affiché à l'utilisateur. Aux débuts du Web dynamique, ces pages HTML étaient générées dans leur état final par le serveur et envoyées telles quelles au navigateur client. Avec l'amélioration des performances des navigateurs, en particulier des interpréteurs JavaScript, il est désormais possible de renvoyer des squelettes de page HTML qui seront ensuite enrichis côté navigateur, sur le poste de l'utilisateur. C'est ce que l'on appelle communément le templating côté client (par opposition au templating côté serveur).

La littérature autour du templating coté serveur étant suffisamment conséquente, ce TechTrends se concentrera sur les technologies de templating côté client et plus particulièrement les Single Page Applications (SPA).



Templating côté serveur vs templating côté client



Le templating côté client présente de nombreux avantages :

- Tout comme le templating côté serveur, il facilite la distinction entre les couches de logique et les couches de présentation.
- Il allège les échanges avec le serveur, qui n'a plus qu'à renvoyer du texte (sous format JSON par exemple), qui sera interprété dans le navigateur. Cela permet de donner un aspect « temps réel » à l'affichage de listes (par exemple les gazouillis de Twitter).
- La manipulation du squelette de la page HTML est plus facile et performante.
- Manipuler les données à partir du navigateur permet une meilleure utilisation des caches, voire même de la création d'un mode offline, qui gère les connexions / déconnexions de l'utilisateur de manière transparente.

Ces avantages permettent d'aboutir au final à une expérience utilisateur plus riche et fluide.

Trois freins principaux pénalisent aujourd'hui la systématisation de l'approche du templating côté client :

- Le premier, et pas des moindres, est que les moteurs de recherche commencent tout juste à être en mesure d'indexer correctement les sites Web s'appuyant sur cette approche. Un site qui souhaite être parfaitement référencé par Google ou Bing préférera donc un templating côté serveur.
- Le second a été mis en exergue par Twitter l'année dernière. Malgré des performances accrues dans l'exécution de JavaScript, le rendu côté client reste plus lent que le rendu côté serveur avec des langages tels que Java ou C#. L'écart croît proportionnellement à la date de mise en service des différents navigateurs.
- Enfin, cette approche nécessite des échanges réseaux plus nombreux entre le terminal client et le serveur. Afin d'offrir une expérience d'utilisation correcte, ces échanges se doivent d'être fluides. Or, malgré les récents progrès, force est de constater qu'en situation de mobilité, il est encore souvent difficile de bénéficier d'une connectivité fiable et performante.

Templating coté serveur ou client, il vous est dorénavant possible de choisir en toute connaissance de cause, la solution la plus adaptée à vos besoins.

L'ère des Single Page Applications

Les Single Page Applications (SPA) sont des applications Web construites autour d'une unique page HTML. Cette approche permet d'éviter le chargement et le traitement d'une nouvelle page HTML à chaque action de l'utilisateur, fluidifiant ainsi son expérience.

Les SPA utilisent donc massivement le templating côté client et prennent en charge la navigation d'un état à l'autre de l'application, la gestion des événements utilisateurs et les communications avec le serveur.

Plusieurs frameworks existent pour bâtir de façon productive ce type d'application. A date, les plus répandus sont AngularJS, Ember.js, Backbone.js et dans une moindre mesure Marionette. Afin de faire votre choix, ces frameworks doivent être comparés sur les critères suivants :

- **La navigation** : Le framework gère-t-il efficacement les différents états de l'application et le passage de l'un à l'autre ?
- **La mise à jour automatique** : Quelles sont les capacités du framework à mettre à jour automatiquement l'interface quand les données changent et à rafraîchir les données suite à une action utilisateur ?
- **La testabilité** : Le framework facilite-t-il l'écriture de tests ?
- **La communauté** qui existe autour du framework est-elle suffisamment importante et de qualité pour le faire prospérer ?
- Quelle est **la facilité de prise en main** du framework ?
- Quelle est **la pérennité** du framework et la stabilité de ses APIs ?

Notre expérience sur les projets nous a permis d'établir la matrice suivante pour les frameworks précédemment cités :

Framework	Navigation	Mise à jour automatique	Testabilité	Communauté	Facilité de prise en main	Pérennité
AngularJS	xxx	xxx	xxx	xxx	xxx	x
Ember.js	xxx	xxx	xxx	xx	x	xxx
Backbone.js	xx	-	xx	xxx	xxx	xx
Marionette	xx	xx	xx	xx	xxx	xx

Comparaison des frameworks

Les éléments différenciateurs à retenir sont :

Backbone.js (associé à **Marionette**) propose une approche MVC1 qui se veut simple et non intrusive. Cette simplicité confère au framework l'empreinte mémoire la plus faible de tous, le rendant idéal pour les smart-phones. Cependant, cette approche lui vaut aussi d'être le framework le moins productif de tous.

AngularJS est la référence du marché. Il cumule beaucoup de bons points tels que la productivité, la simplicité et la testabilité. Ses atouts ont contribué à promouvoir le développement des SPA. Actuellement, le framework est disponible en version 1.3. A moyen terme, il devrait évoluer vers une version 2.0. Celle-ci correspondra à une refonte totale et ne permettra pas une montée de version facile. Elle demandera donc une réécriture complète des applications AngularJS utilisant les versions précédentes !

Ember.js, l'outsider le plus crédible d'AngularJS, a connu des débuts difficiles. Son objectif annoncé est de maximiser la productivité en écrivant le moins de code possible. En contrepartie, il nécessite une connaissance pointue du framework et de ses paradigmes. C'est ce dernier point qui a terni son succès à son démarrage mais il est désormais simplifié, documenté et sa testabilité a été améliorée. Suite à la récente décision d'AngularJS concernant la version 2.0, Ember.js pourrait (re)faire parler de lui en jouant sur la carte de la stabilité.

Face au succès grandissant des SPA, une nouvelle tendance pourrait bien changer la donne dans les prochaines années : les interfaces web modulaires, que nous allons développer dans le paragraphe suivant. Les équipes des différents frameworks de SPA l'ont d'ailleurs bien compris et sont en train de s'adapter : c'est une des raisons de la rupture voulue par AngularJS 2.0. Quant à Ember.js, sa version 2 sera également dotée d'une très forte intégration avec des composants graphiques.



Plusieurs frameworks existent pour bâtir de façon productive ce type d'application. A date, les plus répandus sont AngularJS, Ember.js, Backbone.js et dans une moindre mesure Marionette.

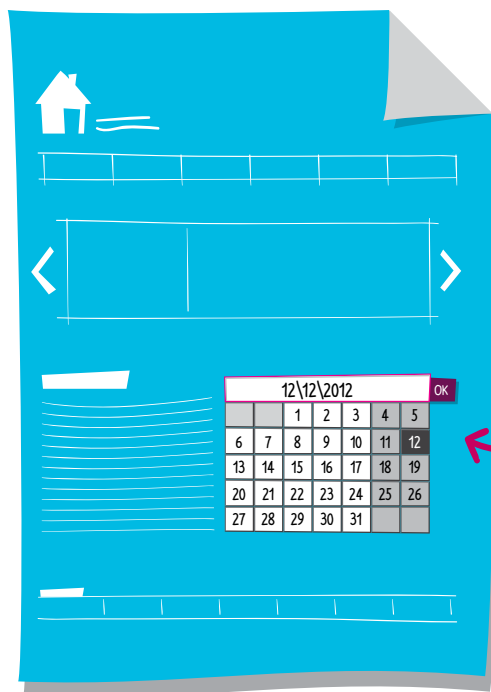


Les interfaces modulaires

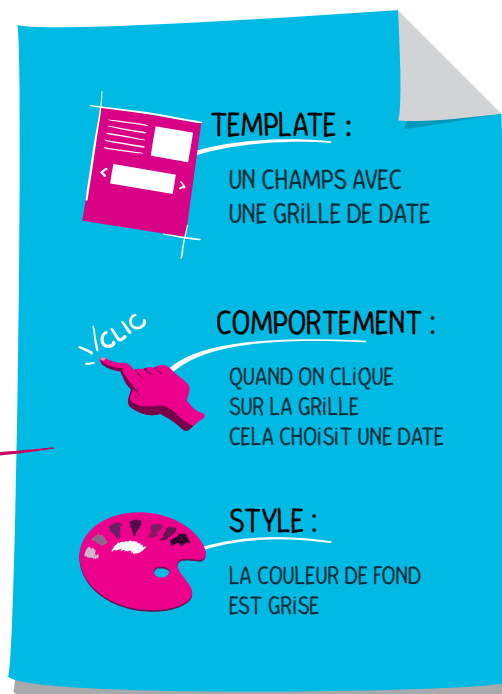
Les interfaces modulaires sont construites en assemblant différents composants complexes (un calendrier, un menu, un tableau triable, etc.). Assimilables à des blocs agnostiques au framework, ils possèdent leurs propres comportements et styles.

Deux implémentations de cette vision sont proposées à la communauté : les web components, poussés et standardisés par la W3C (organisme de normalisation) et React, porté par Facebook.

Un web component pourrait se décrire comme un fichier HTML, embarquant son comportement grâce à du code JavaScript et son style avec du CSS. Il suffit au développeur d'incorporer ce bloc HTML dans son fichier et le tour est joué, il peut utiliser le composant.



INDEX.HTML



CALENDRIER_COMPONENT.HTML

Ces **Web components** et l'ensemble des autres technologies qui les accompagnent sont encore à l'état de brouillon dans la phase de validation des standards W3C. À l'image de HTML5 qui n'a été validé qu'il y a peu de temps, les pionniers de l'industrie sont déjà à pied d'oeuvre pour implémenter la spécification. Les navigateurs n'étant pas tous en phase avec les web components, Google ouvre la voie avec un outil, nommé [Polymer](#), permettant de simuler les fonctionnalités manquantes, mais nécessaires à leur intégration.

Toujours pour promouvoir ces technologies, **Google** a mis à disposition des composants Polymer, les **Paper Elements** utilisant sa fameuse nouvelle « charte graphique », appelée **Material design**. L'idée pour le géant de la recherche est d'unifier le design entre le Web et le mobile.

D'autres outils moins connus, utilisant ce futur standard, ont aussi été créés. On peut notamment évoquer [X-tags](#) de Mozilla et [Bosonic](#), un projet de la communauté open-source.

Dans cette course aux composants, un participant a pris un autre chemin. **Facebook**, qui jusqu'alors était relativement discret sur la scène technologique web, sort depuis quelques temps une suite d'outils visant à bâtir des applications basées sur les interfaces modulaires. À la différence de Google, Facebook a préféré inventer son propre système de composants en créant [React](#). Tout comme les web components, React permet de créer des modules d'interfaces transportables. Il est accompagné de [Flux](#), un outil pour structurer et simplifier le développement, et plus récemment de [Flow](#) qui se situe entre le langage et le vérificateur d'erreur de code.

React se distingue des web components par une syntaxe plus robuste et des performances d'affichage améliorées grâce à l'utilisation d'un DOM virtuel (Document Object Model).

Ainsi, React permet de simuler des événements HTML5 sur les «vieux» navigateurs comme Internet Explorer 8.



React se distingue des web components par une syntaxe plus robuste et des performances d'affichage améliorées grâce à l'utilisation d'un DOM virtuel.



Pas de précipitation !

Vous l'aurez compris, ces solutions techniques sont encore jeunes et même si Google a une bonne position avec Polymer, on ne sait pas vraiment quels seront les détails du standard des interfaces web modulaires de demain. Nous vous conseillons, plutôt que de choisir une de ces technologies, de vous préparer à l'arrivée des interfaces modulaires.

Pour ce faire, vous pouvez instaurer des règles de développement CSS plus strictes comme **BEM** ou **OOCSS**, afin de mieux diviser les styles appliqués à chaque partie de vos interfaces. Vous pouvez aussi mettre en place des catalogues de composants utilisables dans vos applications, comme font les grands frameworks CSS (type Bootstrap ou Foundation). Ces catalogues, plus communément appelés **Kitchen sink**, permettent de mettre en évidence les différents composants utilisés et utilisables aussi bien par vos équipes de développement que vos équipes de designer ou marketing.

LA RÉVOLUTION JAVASCRIPT

Le JavaScript n'est pas en reste. Beaucoup disent qu'il est «the next big language» et pour cause. On ne peut que constater l'ampleur de son utilisation sur des sites comme github.com ou stackoverflow.com, qui sont très représentatifs de la communauté des développeurs. Sa simplicité, sa facilité d'accès et surtout son monopole sur la plateforme Web, en font un langage très utilisé. Pourtant, lorsque l'on développe une application d'entreprise, on se rend rapidement compte qu'il comporte de nombreuses lacunes, le rendant ainsi peu adapté et difficile à maintenir.

Le point négatif mis en exergue par ses détracteurs est sans doute sa robustesse. JavaScript, de par sa nature, est un langage hautement dynamique produisant bien souvent des résultats contre-intuitifs pour les développeurs novices. Le langage ne donne que peu d'assurance d'un bon fonctionnement (par rapport à d'autres langages comme Java). De plus, chaque navigateur implémente sa version de JavaScript, ce qui crée des écarts de comportement entre les différentes librairies standards et oblige à se limiter à la portion congrue commune à toutes les plateformes. Enfin, certains concepts intrinsèques au langage sont souvent mal maîtrisés, ce qui occasionne des incompréhensions et une mauvaise utilisation.



Le JavaScript n'est pas en reste. Beaucoup disent qu'il est « the next big language » et pour cause. On ne peut que constater l'ampleur de son utilisation sur des sites comme github.com ou stackoverflow.com, qui sont très représentatifs de la communauté des développeurs.



Pour résoudre ces problèmes, des poids lourds de l'industrie Web, ainsi que la communauté ont créé des frameworks et langages, permettant, comme Monsieur Jourdain, de « faire du Javascript » sans même le savoir.

Ces solutions peuvent avoir plus ou moins d'impacts sur vos équipes et votre organisation. Voici 3 langages dont l'appropriation est plutôt aisée :

- **TypeScript** : un JavaScript plus robuste made in Microsoft qui propose une approche intéressante permettant de l'incorporer dans un projet JavaScript au fur et à mesure. Notons également qu'il servira de base au langage **AtScript**, développé spécialement par Google pour AngularJS 2.0.
- **CoffeeScript** : un langage de la communauté visant à apporter un maximum de concision et de puissance au développeur.
- **Flow** : le « langage » de Facebook a fait son apparition plus récemment, il est donc encore trop tôt pour se prononcer sur ce dernier. Il reste très léger en terme de modification et à l'image de TypeScript, il aide à rendre le code JavaScript plus robuste.

Les nombreuses bibliothèques du monde JavaScript telles que JQuery, Moment.js, Lo-Dash, etc., devenues des standards de facto permettent également de palier aux limites induites par l'hétérogénéité des implémentations.

Deux autres alternatives, plus impactantes en terme de changement, apportent un niveau d'amélioration bien supérieur :

- La nouvelle version standardisée de JavaScript, **EcmaScript 6**, constitue la voie « officielle ». Sa date de sortie est à ce jour inconnue, mais des bibliothèques simulent déjà bon nombre de ses fonctionnalités.
- La plateforme **Dart**, alternative proposée par Google, est composée d'un langage du même nom, d'un jeu de bibliothèques complet, d'outils d'industrialisation et de son propre environnement d'exécution, lui conférant ainsi des performances deux fois supérieures à JavaScript. Néanmoins, à l'heure actuelle, ce langage nécessite d'être transformé en JavaScript pour être exécutable sur l'ensemble des navigateurs du marché.

*web
component*

React

SPA

Templating

JavaScript



Take away Front trends



CRÉER

- Évaluer la pertinence d'un templating côté client.
- Choisir le framework de SPA répondant au mieux à votre stratégie et convenant à votre équipe.
- Préparer l'arrivée des interfaces modulaires.
- Envisager les langages alternatifs à JavaScript.

Industrialiser



Après avoir bien
pensé votre produit,

avoir choisi l'architecture adéquate
et les outils adaptés, une grande partie
du chemin vers le succès a été parcourue,
mais une étape vitale pour la pérennité du
produit reste à réaliser : l'industrialisation.

Ce terme englobe les pratiques visant à maintenir et à faire évoluer votre produit dans les meilleures conditions possibles : rapidement, en gardant une qualité optimale et en évitant les régressions fonctionnelles.

Le monde du Front est parti de très loin (pour ne pas dire de rien) concernant les outils d'industrialisation, mais l'émergence d'une réelle architecture Front a poussé les développeurs à bâtir les solutions nécessaires. Nous disposons donc aujourd'hui d'une grande variété d'outils puissants pour répondre à ces besoins.

TESTEZ VOTRE APPLICATION

Le premier pas vers l'industrialisation doit se faire dès la phase de développement grâce aux tests automatisés. Cette pratique, prônée par le mouvement Craftsmanship (voir le TechTrends #4) est un facteur clé de succès dans le cycle de vie d'un produit. Longtemps oubliés dans les développements d'interfaces Web, la complexification de ces dernières a rendu leur présence obligatoire. Ils sécurisent les développements d'un point de vue technique et fonctionnel, et évitent les régressions lors des différentes évolutions du produit.

Pour valider le bon fonctionnement d'une application Web, il est nécessaire de se placer à différents niveaux de granularité afin de pouvoir détecter et identifier l'erreur isolée ou le dysfonctionnement global. Pour cela il existe trois grandes familles de tests avec des rôles bien spécifiques.



Le premier pas vers l'industrialisation doit se faire dès la phase de développement grâce aux tests automatisés.



La première catégorie, que l'on nomme les tests unitaires sont faits pour détecter rapidement des erreurs isolées. Ils sont rapides (plusieurs centaines peuvent être exécutés en moins d'une seconde), ce qui leur confère d'être souvent les premiers avertisseurs en cas de problème. Leur inconvénient (qui est aussi leur force) est qu'ils sont faits pour ne vérifier que le bon comportement d'une petite partie de l'application. Ainsi, ils peuvent rapidement et précisément signaler quelle brique ne fonctionne pas, mais ne donnent pas d'indication sur la bonne marche de la communication entre les briques elles-mêmes.

Pour résoudre ce problème, il existe les tests d'intégration, plus lents que leurs homologues unitaires. Ils ont pour but de tester le fonctionnement de plusieurs briques travaillant de concert. Sans pour autant aller jusqu'à tester toute l'interface utilisateur, ils font office de deuxième barrage contre les bugs.

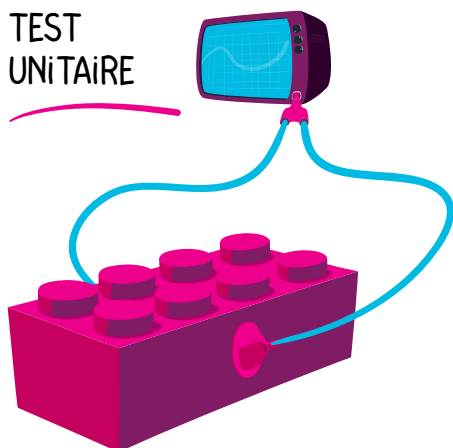
Enfin, les derniers et pas des moindres sont les tests d'acceptance. Plutôt lents (allant de 5 secondes à la minute), ils permettent de simuler un véritable scénario utilisateur (clic sur un bouton, parcours dans l'application, etc.). Ils sont les garde-fous du bon fonctionnement de l'ensemble. Ils ont par contre le défaut d'être très fragiles, car il suffit de déplacer un bouton pour que, potentiellement, ils ne fonctionnent plus.



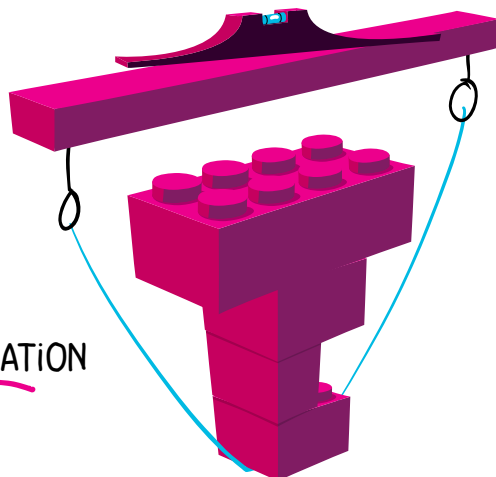
Pour valider le bon fonctionnement d'une application Web, il est nécessaire de se placer à différents niveaux de granularité.



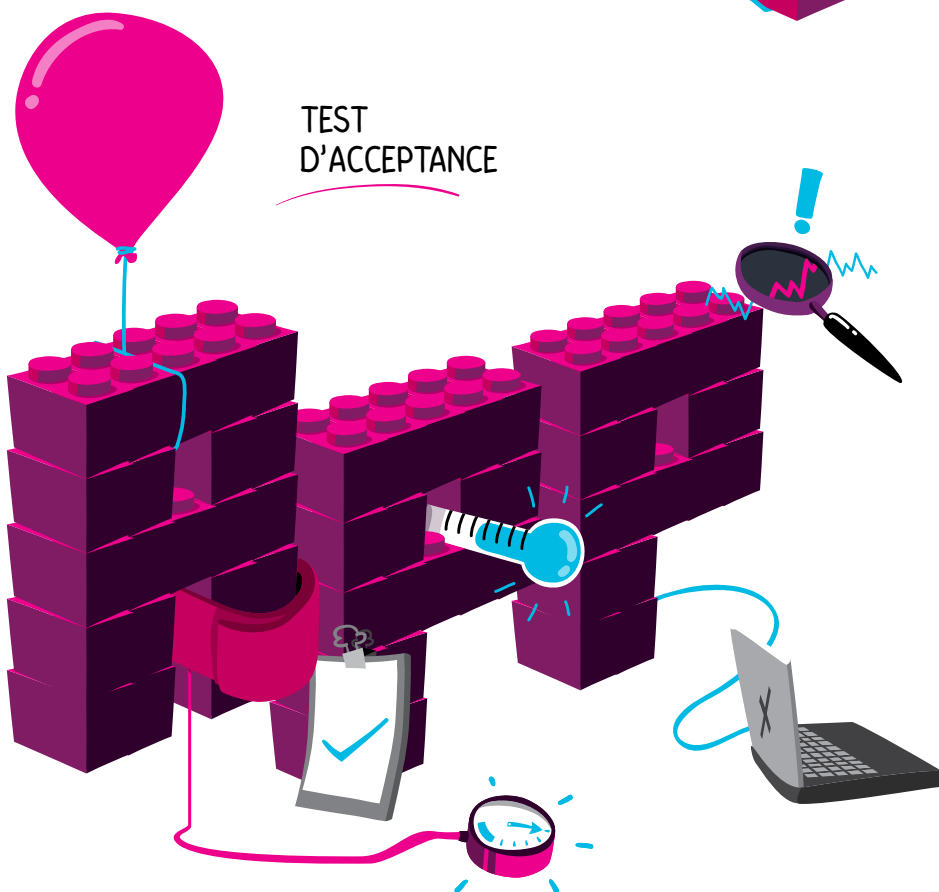
TEST
UNITAIRE



TEST
D'INTÉGRATION



TEST
D'ACCEPTANCE



Les 3 types de tests

Les outils des tests unitaires

Comme dans tous les écosystèmes émergents, il est parfois difficile de faire son choix tant l'écosystème des outils, et plus particulièrement ceux des tests, est foisonnant. D'autant qu'ici, les équipes qui les ont développés ont appris de leurs aînés (Java surtout) pour produire des logiciels de très bonne qualité.

Parmi les frameworks les plus en vue, nous avons sélectionné [Jasmine](#), qui propose un ensemble d'outils relativement complet. Il vise, via les TU, à décrire précisément le comportement et les résultats attendus. Dans la même famille, on peut citer [Mocha](#), très puissant même dans des cas de tests complexes.

Pour faciliter l'écriture de ces tests vous pouvez leur adjoindre [Chaijs](#), qui effectue des vérifications de fonctionnement de façon concise et élégante.

[Sinon.JS](#) facilite l'isolation entre tests unitaires et permet de garder donc des tests concis, maintenables et rapides à exécuter.

Enfin, pour piloter l'exécution de ces tests unitaires, nous citerons l'incontournable [Karma](#). A l'origine, créé par l'équipe d'AngularJS, il est désormais utilisé et maintenu par toute la communauté.

De nouveau, le choix des outils dépend avant tout de la sensibilité de l'équipe. La compréhension et la maîtrise de ces frameworks sont des pré-requis indispensables au respect du dogme «une fonction - un test». L'écriture des tests ne doit pas devenir un repoussoir, et les bonnes pratiques Craftsmanship (pair-programming, revue de code) peuvent aider l'ensemble de l'équipe à monter en compétence sur le sujet.

Les outils des tests d'intégration

Ce type de tests ne nécessite pas d'outil particulier. On peut donc réutiliser ceux déjà utilisés lors des tests unitaires. Ici, il faut garder en tête que c'est l'approche du test qui change. On ne souhaite plus se cantonner à une brique bien particulière, mais à plusieurs briques qui fonctionnent toutes ensemble afin de vérifier que cette combinaison s'articule comme attendue.



La compréhension et la maîtrise de ces frameworks sont des prérequis indispensables au respect du dogme « une fonction - un test ».



Les tests d'acceptance

Les tests d'acceptance permettent de tester l'ensemble d'une application Web d'un point de vue utilisateur, en pilotant un navigateur. Cette idée n'est pas nouvelle, le projet Selenium propose de le faire depuis 2004. Ce type de tests permet de vérifier l'intégration de toute la stack d'une application Web, depuis le navigateur internet jusqu'à la base de données.

L'envie de substituer les tests d'acceptance aux tests unitaires est tentante mais contre productive. En effet, la philosophie et le but recherchés derrière ces deux types de tests sont radicalement différents mais surtout complémentaires. Tandis que le test unitaire valide le comportement attendu d'une partie du code par le « haut », c'est-à-dire d'un point de vue développeur, de façon isolée et indépendante du contexte d'exécution, le test d'acceptance a pour but de tester la tuyauterie de l'application et valider les bonnes liaisons entre chaque fonctionnalité de l'application.

L'indéniable avantage de ces tests d'acceptance est l'automatisation de la phase de recette. Les équipes de développement peuvent se décharger des longues phases de tests manuels, leur permettant de se concentrer sur les besoins métiers.

Cependant, malgré l'importance et les avantages de ces tests, il faut être conscient des quelques contraintes liées à leur réalisation. En effet, ils sont coûteux en temps d'exécution : un test assez complet nécessite 2 à 5 secondes pour couvrir un cas d'usage. En multipliant le nombre de cas et de « user stories », on peut rapidement arriver à un temps d'exécution du jeu de tests dépassant les 20 minutes. Dans ce cas, on ne peut plus parler d'intégration continue, car le temps de feedback est beaucoup trop élevé. Pour conserver un temps d'exécution raisonnable, il faut se concentrer sur les cas d'utilisation les plus communs. Les cas à la marge pourront être couverts par des tests unitaires aux limites, ou bien par une batterie de tests d'acceptance jouée plus rarement (de manière quotidienne plutôt que systématique par exemple).

Plusieurs frameworks de tests d'acceptance sont disponibles dont [Zombie.js](#), [CasperJS](#), [Selenium](#) et [Protractor](#).

	Zombie.js	CasperJS	Selenium	Protractor
Couverture des navigateurs	+	++	+++	+++
Vitesse d'exécution	+++	++	+	+
Maturité	++	++	+++	+
Intégration avec les frameworks Web	-	-	-	+
Facilité de débogage	+	+	++	+
Pour quelles applications le choisir ?	Single Page Applications	Applications Web avec rendu mixte (serveur et Single Page Applications)	Applications Web avec rendu côté serveur ou Single Page Applications	Single Page Applications écrites avec AngularJS

Les frameworks d'acceptance



Cette grille de lecture, tirée de nos expériences passées, ne doit pas être votre seul critère de choix : les objectifs du projet, ainsi que l'appétence des équipes de développement pour tel ou tel outil devront être pris en compte.



Une fois ces tests automatisés mis en place, il est nécessaire de les exécuter souvent, sur une grande variété de navigateurs et de devices.

Il existe deux écoles :

- Créer en interne un browser Lab hébergeant une grande variété de plateformes physiques et logicielles. Malheureusement, ces machines immobilisées demandent un effort d'investissement (machine, espace physique) et de maintenance supplémentaires.
- Utiliser des infrastructures «as-a-service». Des plateformes de tests dans le cloud telles que [SauceLabs](#) et [BrowserStack](#), proposent des combinaisons très larges de navigateurs et de systèmes à des coûts raisonnables. De plus, leur modèle de facturation «à la demande» permet de maintenir une facturation flexible.

Les tests d'acceptance permettent également de mesurer les performances de bout en bout d'une application Web. Il devient possible de construire un bench applicatif mesurant les performances d'ensemble de toute l'application déployée en production. Automatiser cette batterie de tests permet de suivre dans le temps, le comportement sous charge de l'application.

Enfin, les tests d'acceptance Front présentent une autre spécificité : comme ils ciblent une application dont l'apparence est « clé », ils sont utilisés pour traiter ce que sera le rendu utilisateur.

Certains outils permettent donc de prendre des copies d'écran régulièrement lors des tests d'acceptance. Il est ainsi possible de se bâtir une photothèque de référence, et de comparer, à chaque version, le rendu final des pages avec cette référence. Certaines modifications seront voulues, d'autres résulteront d'effet de bord et devront être corrigées.

On citera [PhantomCSS](#) et [Resemble.js](#)

L'un des avantages collatéral est la mise à disposition d'un jeu de captures d'écran à chaque nouvelle version. Celles-ci peuvent être rapidement réutilisées par des équipes transverses : documentation de l'application, communication, marketing, etc.

Quoi qu'il en soit, le pilotage des tests automatisés est de la responsabilité de la plate-forme d'intégration continue. C'est également cette plate-forme qui est chargée de bâtir votre application à chaque ajout de code. Zoomons sur cette phase.

BUILDEZ VOTRE APPLICATION FRONT

Dans le monde du Back-end, la bataille a eu lieu depuis bien longtemps, et deux standards de facto dominent le marché : [Jenkins](#) en tant qu'outil d'intégration continue et [Maven](#) en tant qu'outil de build.

Le développement d'une application Front de qualité nécessite l'utilisation d'outils variés. Certaines phases du build sont communes avec le back-end : lancer facilement les tests automatisés, compiler ses fichiers sources, les packager, etc. D'autres sont plus spécifiques : compiler les langages alternatifs à CSS et JavaScript, minifier, concaténer ces fichiers, etc.

Le monde du build Front n'en est encore qu'à ses balbutiements : chaque mois, de nouveaux acteurs apparaissent sur le marché. La majorité d'entre eux est basée sur le moteur JavaScript V8, le plus souvent à travers le serveur Node.js.

Un standard déjà en place semblait s'être dégagé en la personne de [Grunt](#), outil de build à la syntaxe déclarative et aujourd'hui largement utilisé. Malheureusement, à mesure que les projets grossissent, Grunt a rapidement été montré du doigt pour sa lenteur et sa verbosité. C'est pour cette raison qu'un nouveau projet, arrivé en début d'année 2014, est annoncé comme la future référence. [Gulp](#) est beaucoup plus rapide que son homologue et se base sur une syntaxe plus impérative permettant de rester concis même lorsque l'on crée des étapes de build complexes sortant des sentiers battus.

Ces deux solutions sont des alternatives crédibles à des outils de build plus traditionnels, et là encore, le principal critère de choix sera l'adoption des équipes.

La phase de build est bien souvent l'occasion de parcourir le code et d'en évaluer la qualité. Les applications Front ne dérogent pas à la règle et peuvent être, elles aussi, auditées de manière automatisée.

Validez la qualité de votre code et son rendu final

Les outils de qualité de code sont de mieux en mieux implantés dans les entreprises. Pour n'en citer qu'un, il est probable que vos équipes de développement aient testé [SonarQube](#), qui embarque un certain nombre de frameworks dédiés le plus souvent à Java : [PMD](#), [CheckStyle](#), [Cobertura](#), etc.

Ces outils sont rarement adaptés dans le cas d'applications Front, le plus souvent orientés purement navigateur et dont les langages, en particulier JavaScript, se prêtent très mal à cette analyse statique.

Comme pour les outils de build, les nouveaux outils du développeur Front permettant cette analyse qualitative sont en plein boom.

- **Plato** : il fournit un dashboard de visualisation de complexité cyclomatique sous forme de graphes. On peut ainsi pointer rapidement les fichiers trop longs, les erreurs de formatage ou encore avoir un indice sur la maintenabilité de notre application.

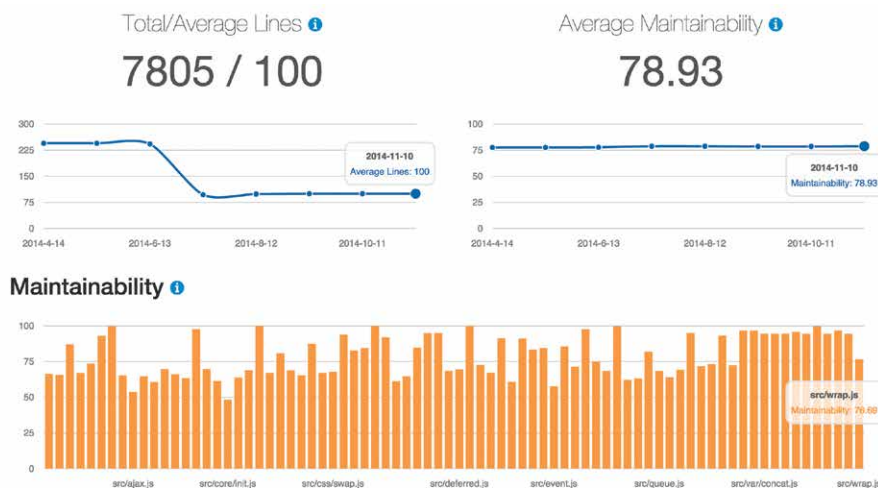


Tableau de bord des métriques Plato sur le projet JQuery

- **Istanbul** : un outil de mesure de couverture de code simple et efficace. Il est embarqué nativement dans Karma et permet de créer des rapports détaillés à destination des développeurs et des serveurs d'intégration continue.

Code coverage report for All files

Statements: 96.88% (2114 / 2185) Branches: 91.97% (859 / 934) Functions: 98.77% (480 / 486) Lines: 97.53% (2053 / 2105) Ignored: 23 statements, 8 functions, 26 branches

File	Statements	Branches	Functions	Lines
istanbul/	100.00% (3 / 3)	100.00% (0 / 0)	100.00% (0 / 0)	100.00% (3 / 3)
istanbul/lib/	98.71% (668 / 697)	98.24% (391 / 398)	99.31% (144 / 145)	98.67% (667 / 676)
istanbul/lib/command/	96.79% (241 / 249)	92.50% (74 / 80)	96.23% (51 / 53)	98.34% (237 / 241)
istanbul/lib/command/common/	92.22% (83 / 90)	79.25% (42 / 53)	100.00% (8 / 8)	93.02% (80 / 86)
istanbul/lib/report/	95.22% (677 / 711)	83.27% (214 / 257)	99.38% (159 / 160)	95.92% (659 / 687)
istanbul/lib/report/common/	100.00% (20 / 20)	100.00% (11 / 11)	100.00% (5 / 5)	100.00% (17 / 17)
istanbul/lib/store/	100.00% (73 / 73)	100.00% (14 / 14)	100.00% (28 / 28)	100.00% (72 / 72)
istanbul/lib/util/	97.05% (329 / 339)	93.39% (113 / 121)	97.75% (87 / 89)	98.45% (318 / 323)


Rapport de couverture de code du projet Istanbul lui-même

Néanmoins, l'analyse automatisée n'est pas un but en soi : en effet, la production d'indicateurs n'aura d'effets bénéfiques que si elle est rendue visible au sein des équipes (management visuel) et qu'elle est suivie dans le temps.


Pour les applications Front, il est même possible d'aller plus loin : la grande majorité de ces applications étant destinée à être déployée sur le Web (que ce soit dans un espace public ou privé), des outils de mesure « externes », disponibles à la demande, ont émergé.

Ces outils ont pour but de tester, en conditions « réelles », la robustesse, la fluidité et les performances de votre application (sous réserve, bien sûr, que le Back-end sur lequel elles s'adossent soit lui aussi solide).

On citera ici deux de ces outils, qui peuvent être intégrés à vos outils de build, votre intégration continue et votre management visuel. [Sitespeed](#) et [gtmetrix](#) proposent tous deux une analyse « externe » de votre site, en classant les éventuels soucis de performance en fonction d'indicateurs communément admis sur le marché (temps d'affichage de la page, nombre d'allers/retours réseau, taille des fichiers CSS et JS, nombre d'éléments dans le DOM, etc.). Intégrer ce type d'analyse à votre build quotidien permet de rapprocher la vision « internaute » (et donc le sentiment perçu par l'utilisateur final) de vos développements.



Pour les applications Front, il est même possible d'aller plus loin : la grande majorité de ces applications étant destinée à être déployée sur le Web (que ce soit dans un espace public ou privé), des outils de mesure « externes », disponibles à la demande, ont émergé.





Latest Performance Report for: <http://blog.xebia.fr/>

[Download PDF](#)

Report generated: Thu, Dec 4, 2014, 8:45 AM -0800

Test Server Region: Vancouver, Canada

Using: Firefox (Desktop) 25.0.1, Page Speed 1.12.16, YSlow 3.1.8



Looks like you're running WordPress
[Have a look at our WP optimization tips »](#)



Looks like you might not be using a CDN
[Why should I use a CDN? »](#)

Summary

Page Speed Grade:

(80%)

B

YSlow Grade:

(75%)

C

Page load time: 5.26s

Total page size: 1.58MB

Total number of requests: 106

Options

- [Re-Test Page](#)
- [Compare to another URL](#)

Share This Report



I got my scores; what now?

Start optimizing your site! But before you do:

- Understand the recommendations
They are meant to be generic, best practices; not everything will apply to your site.
- Rules are sorted in order of impact upon score
Optimizing rules at the top of the list can greatly improve your overall score.

Read our [FAQ](#) for more info.

Need help?

Gossamer Threads can handle

Breakdown

Page Speed

YSlow

Timeline

History

RECOMMENDATION	GRADE		TYPE	PRIORITY
Leverage browser caching	F (6)	↓	Server	High
Specify image dimensions	F (19)	↓	Images	High
Avoid CSS @import	F (40)	↓	CSS	Medium
Optimize the order of styles and scripts	E (55)	↓	CSS/JS	High
Defer parsing of JavaScript	D (63)	↑	JS	High
Remove query strings from static resources	C (72)	↓	Content	High
Optimize images	C (75)	↑	Images	High
Combine images using CSS sprites	B (80)	↑	Images	Medium
Inline small CSS	B (88)	↓	CSS	High
Minify CSS	A (9)	↑	CSS	High
Serve resources from a consistent URL	A (9)	↑	Content	High

Rapport de gtmatrix concernant le blog de Xebia

Take away Front trends



INDUSTRIALISER

- *Intégrer les tests à chaque étape de développement.*
- *Mettre en place des outils de validation du code fiable.*
- *Installer une intégration continue afin d'avoir un feedback rapide de l'état de santé du projet.*
- *Afficher, partager et diffuser l'information à toutes les parties prenantes du projet.*



Conclusion

L'apparition des applications Front bouscule les codes du développement applicatif, dont les standards ont été bâtis sur des raisonnements Back-end. Elles remettent en cause l'hégémonie des outils de développement et de test bien implantés sur le marché.

La genèse d'un projet Front est un réel challenge, et demande l'intervention de nouveaux spécialistes, les experts UX. La définition des interfaces, des chemins de navigation et des interactions multi-canal devient une discipline à part entière et peut, à elle seule, conditionner la réussite ou l'échec d'un projet. A l'heure où l'offre de service est pléthorique, et l'utilisateur plus versatile que jamais, prendre le risque de lancer un produit offrant une expérience utilisateur médiocre, c'est prendre le risque de perdre définitivement une partie de ses utilisateurs.

L'utilisateur revient donc au centre du projet, et toute la conception en amont sera pilotée par son besoin et son ressenti. Réussir à mettre en lumière ces points, et surtout les expliciter vis-à-vis des équipes de pilotage et de réalisation, est un métier émergent, ayant ses propres outils et techniques, qui aujourd'hui fait la différence auprès des utilisateurs finaux.

De la même façon, les outils de développement sont en perpétuelle mutation. Les développeurs Front ont appris des erreurs du passé et investissent aujourd'hui lourdement sur un écosystème productif, adapté aux standards de l'industrie du Web. Les grands du Web comme la communauté imposent des standards de facto, et ce sont les éditeurs des navigateurs qui doivent être force de proposition ou se plier à ces évolutions, voire souvent les deux à la fois. Si un navigateur n'adopte pas rapidement la nouvelle vague technologique, c'est en embarquant des bibliothèques spécifiques que les développeurs pallient à cette déficience.

Enfin, tous les acquis d'années de pratique sur le Back-end n'ont pas été mis au ban : l'automatisation, la qualité, et la maintenabilité des applications Front représentent un réel enjeu pour les DSI. Le marché des outils de développement et d'automatisation s'adapte, en proposant une réelle alternative aux standards du monde Java ou .Net. Ces outils, partant du postulat qu'une application Front sera souvent disponible sur le Web, vont même plus loin, en proposant des analyses « externes » (du point de vue de l'utilisateur), disponibles en tant que service dans le Cloud.

Le monde des applications Front est en profonde mutation, et les dernières années ont vu apparaître des spécialistes du métier, tant au niveau du recueil et de l'expression du besoin qu'au niveau de la réalisation. La transformation s'accélère, et une nouvelle génération de développeurs Full stack, prêts à bousculer les règles établies, émerge.



Take away Front trends



MAQUETTER

- Mettre vos utilisateurs au centre de vos réflexions au travers de personas.
- Organiser votre contenu en fonction de vos parcours clients.
- Adapter votre design aux canaux de diffusion choisis.



CRÉER

- Évaluer la pertinence d'un templating côté client.
- Choisir le framework de SPA répondant au mieux à votre stratégie et convenant à votre équipe.
- Préparer l'arrivée des interfaces modulaires.
- Envisager les langages alternatifs à JavaScript.



INDUSTRIALISER

- Intégrer les tests à chaque étape de développement.
- Mettre en place des outils de validation du code fiable.
- Installer une intégration continue afin d'avoir un feedback rapide de l'état de santé du projet.
- Afficher, partager et diffuser l'information à toutes les parties prenantes du projet.



Merci à

*Laëtitia Janné, Héloïse Guyot, Chloé Desault
et Joachim Rousseau.*

Les auteurs



Romain
Landsberg



Alexandra
Bernards



Julien
Smadja



Florent
Le Gall



Phillipe
Antoine



Christophe
Heubès



Florent
Duveau



Anne
Beauchart



Pablo
Lopez



Marina
Tracco



Mathieu
Breton



Jeremy
Vinai

Xebia

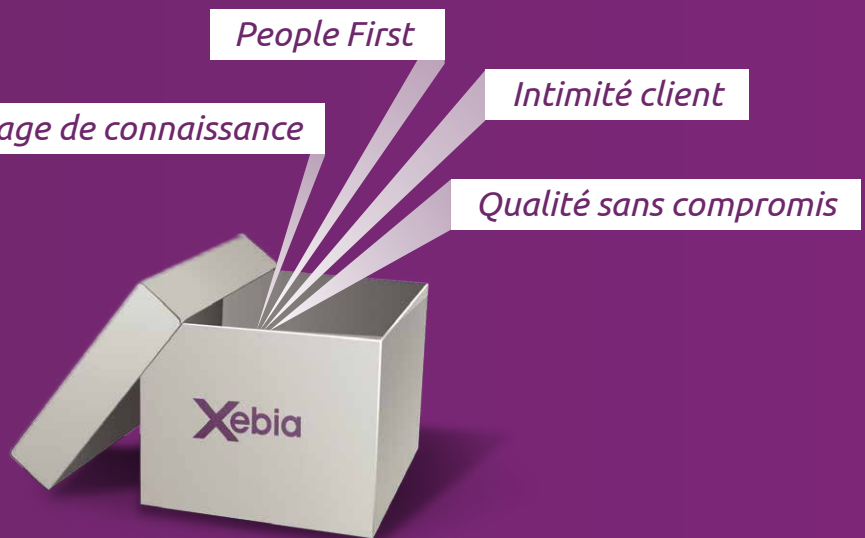


SOFTWARE DEVELOPMENT **DONE RIGHT**

*est une entreprise agile qui délivre des logiciels
sur mesure*

à ses clients

*nos valeurs fondatrices,
clés de notre succès*



*Xebia France
156 bd Haussmann
75008 Paris
+33 (0)1 53 89 99 99
info@xebia.fr*

*UX republic
156 bd Haussmann
75008 Paris
+33 (0)1 53 89 99 99
hello@ux-republic.com*