



Conception d'applications web

Ce qu'il faut savoir

Patrice **BERTRAND**
Directeur général

Smile
OPEN SOURCE SOLUTIONS

www.smile.fr • +33 (0)1 41 40 11 00 • contact@smile.fr
www.smile-oss.com • blog.smile.fr • [twitter: @GroupeSmile](https://twitter.com/GroupeSmile)

PREAMBULE

Ce livre blanc

Pour beaucoup d'entreprises, les applications web déployées en intranet ont remplacé les solutions client-serveur.

Avec des centaines ou des milliers de pages, des bases de données de centaines d'entités, des processus de travail qui peuvent être sensiblement plus complexes que la réservation d'un billet SNCF, la conception d'ergonomie des applications web est un domaine d'une grande richesse, où les bonnes pratiques commencent à émerger.

La conception et l'ergonomie de ces applications n'obéit pas tout à fait aux mêmes règles que la conception de sites Internet: l'efficacité de l'outil de travail est l'objectif principal, et l'industrialisation du développement est une nécessité.

Or s'il existe de nombreux ouvrages traitant de la conception ergonomique des sites web, il y a peu de littérature sur le thème de la conception des applications web.

Pourtant, si certaines des bonnes pratiques de la conception de sites s'appliquent aux applications web, il en est d'autres qui restent à définir.

Spécialistes de ces grandes applications web déployées en Intranet, Smile vous fait partager son savoir-faire au travers de ce livre blanc.

Ce que vous y trouverez

Nous avons cherché à réunir ici les règles essentielles qui rendront une application web utilisable. Certaines relèvent de l'interface utilisateur proprement dite, d'autres de la conception plus largement.

Mais toutes nos recommandations sont issues de notre expérience, et validées sur le terrain.

Ce que vous n'y trouverez pas

Notre livre blanc s'efforce de traiter des choix fondamentaux de la conception d'applications web. Il est d'autres choix, qui occupent parfois trop les esprits, que nous n'évoqueront pas ici. Placera-t-on plutôt un menu en haut ou plutôt à gauche ? Quelles fontes, quelles couleurs ? La conception d'une application web inclut une part de conception graphique et une part de conception ergonomique. Nous ne traitons pas ici les aspects esthétiques, bien qu'ils aient un rôle important à jouer au regard du confort d'utilisation.

Nous ne fournirons pas ici de *technique* pour la mise en œuvre de nos recommandations : ni morceaux de Html, ni Javascript. Ce document décrit ce qu'il faut faire, de nombreux ouvrages existants vous diront comment le faire.

Un domaine fermé

Parce que les sites Internet recherchent la visibilité, les bonnes pratiques sur l'Internet se diffusent rapidement. Quand Yahoo lance le portail MyYahoo, ou Amazon le « 1 click order », tout le monde le sait, et les meilleurs exemples servent rapidement d'inspiration aux autres sites.

Or par nature, les applications web vivent dans un monde beaucoup moins ouvert. Seules quelques centaines de personnes ont accès à une application, et rares sont les utilisateurs qui ont vu et pratiqué plus d'une demi-douzaine d'applications.

En outre, la compétition est réduite. Sur l'Internet vous irez peut-être réserver une voiture de location sur tel site parce qu'il est mieux conçu et plus rapide que ses concurrents. Sur l'Intranet, votre entreprise ne vous propose pas 3 outils de travail différents pour faire la même tâche. Or la concurrence est l'un des moteurs de la diffusion des meilleures pratiques.

Le manque d'ouverture et de compétition fait de l'ergonomie des applications web un sujet encore mal stabilisé, où chacun se constitue ses bonnes pratiques dans son coin, sans qu'un ouvrage fasse référence.

Application web

Qu'est ce qu'une application web ? C'est une application construite avec les technologies du web, mais répondant à un besoin '*métier*'. C'est donc une application qui aurait existé quelle que soit la technologie, qui aurait peut être été réalisée en mode terminal 3270 il y a 20 ans, en mode client-serveur il y a 10 ans, et que l'on choisit aujourd'hui de réaliser en mode web.

L'application web, si elle est le plus souvent déployée sur un Intranet, est très différente d'un *site Intranet*. Elle n'a pas une finalité de *communication interne*, ni de *portail, bouquet de service*, elle vise à être l'outil de travail de tout ou partie des métiers de l'entreprise.

Smile

Smile est une société d'ingénieurs experts dans la mise en œuvre de solutions open source et l'intégration de systèmes appuyés sur l'open source. Smile est membre de l'APRIL, l'association pour la promotion et la défense du logiciel libre.

Smile compte 290 collaborateurs en France, 320 dans le monde (septembre 2009), ce qui en fait la première société en France spécialisée dans l'open source.

Depuis 2000, environ, Smile mène une action active de veille technologique qui lui permet de découvrir les produits les plus prometteurs de l'open source, de les qualifier et de les évaluer, de manière à proposer à ses clients les produits les plus aboutis, les plus robustes et les plus pérennes.

Cette démarche a donné lieu à toute une gamme de livres blancs couvrant différents domaines d'application. La gestion de contenus (2004), les portails (2005), la business intelligence (2006), les frameworks PHP (2007), la virtualisation (2007), et la gestion électronique de documents (2008), ainsi que les PGI/ERPs (2008). Parmi les ouvrages publiés en 2009, citons également « Les VPN open source », et « Firewall est Contrôle de flux open source », dans le cadre de la collection « Système et Infrastructure ».

Chacun de ces ouvrages présente une sélection des meilleures solutions open source dans le domaine considéré, leurs qualités respectives, ainsi que des retours d'expérience opérationnels.

Au fur et à mesure que des solutions open source solides gagnent de nouveaux domaines, Smile sera présent pour proposer à ses clients d'en bénéficier sans risque. Smile apparaît dans le paysage informatique français comme le prestataire intégrateur de choix pour accompagner les plus grandes entreprises dans l'adoption des meilleures solutions open source.

Ces dernières années, Smile a également étendu la gamme des services proposés. Depuis 2005, un département consulting accompagne nos clients, tant dans les phases d'avantprojet, en recherche de solutions, qu'en accompagnement de projet. Depuis 2000, Smile dispose d'un studio graphique, devenu en 2007 Smile Digital – agence interactive, proposant outre la création graphique, une expertise e marketing, éditoriale et interfaces riches. Smile dispose aussi d'une agence spécialisée dans la TMA (support et l'exploitation des applications) et d'un centre de formation complet, Smile Training. **Enfin, Smile est implanté à Paris, Lille, Lyon, Grenoble, Nantes, Bordeaux, Poitiers, Aix-en-Provence et Montpellier. Et présent également en Espagne, en Suisse, au Benelux, en Ukraine et au Maroc.**

Quelques références de Smile

Intranets - Extranets

- Société Générale - Caisse d'Épargne - Bureau Veritas - Commissariat à l'Energie Atomique
- Visual - Vega Finance - Camif - Lynxial - RATP - AMEC-SPIE - Sonacotra - Faceo - CNRS
- AmecSpie - Château de Versailles - Banque PSA Finance - Groupe Moniteur - CIDJ - CIRAD
- Bureau Veritas - Ministère de l'Environnement - JCDecaux - Ministère du Tourisme
- DIREN PACA - SAS - Institut National de l'Audiovisuel - Cogedim - Ecureuil Gestion - Prolea
- IRP-Auto - AFNOR - Conseil Régional Ile de France - Verspieren - Zodiac - OSEO
- Conseil Général de la Côte d'Or - IPSOS - Bouygues Telecom - Pimkie Diramode
- Prisma Presse - SANEF - INRA

Internet, Portails et e-Commerce

- cadremploi.fr - chocolat.nestle.fr - creditlyonnais.fr - explorimmo.com - meilleurtaux.com
- cogedim.fr - capem.fr - editions-cigale.com - hotels-exclusive.com - souriau.com - pci.fr
- gdf.fr/presse - dsv-cea.fr - egide.asso.fr - osmoz.com - spie.fr - nec.fr - vizzavi.fr - sogeposte.fr
- metro.fr - stein-heurtey-services.fr - bimp.org - buitoni.fr - aviation-register.com - cci.fr
- schneider-electric.com - calypso.tm.fr - inra.fr - cnil.fr - longchamp.com - aesn.fr
- Dassault Systemes 3ds.com - croix-rouge.fr - worldwatercouncil.org - projectif.fr
- editionsbussiere.com - glamour.com - fratel.org - tiru.fr - faurecia.com - cidil.fr - prolea.fr
- ETS Europe - ecofi.fr - credit-cooperatif.fr - odit-france.fr - pompiersdefrance.org - cetiom.fr
- watermonitoringaliance.net - bloom.com - meddispar.com - nmmedical.fr - medistore.fr
- Yves Rocher - jcdecaux.com - cg21.fr - Bureau Veritas veristar.com - voyages-sncf.fr
- eurostar.com - AON conseil - OSEO - cea.fr - eaufrance.fr - banquepsafinance.com
- nationalgeographic.fr - idtgv.fr - prismapub.com - Bouygues Construction

Applications métier

- Renault - Le Figaro - Sucden - Capri - Libération - Société Générale - Ministère de l'Emploi
- CNOUS - Neopost Industries - ARC - Laboratoires Merck - Egide - Bureau Veritas
- ATEL-Hotels - Exclusive Hotels - Ministère du Tourisme - Groupe Moniteur - Verspieren
- Caisse d'Épargne - AFNOR - Souriau - MTV - Capem - Institut Mutualiste Montsouris
- Dassault Systemes - Gaz de France - CFRT - Zodiac - Croix-Rouge Française

Systèmes documentaires Xml

- Centre d'Information de la Jeunesse (CIDJ) - Pierre Audoin Consultants - EDF R&D

SOMMAIRE

PREAMBULE..... 2

- CE LIVRE BLANC 2
- CE QUE VOUS Y TROUVerez 2
- CE QUE VOUS N'Y TROUVerez PAS 3
- UN DOMAINE FERME..... 3
- APPLICATION WEB 3
- SMILE..... 4
- QUELQUES REFERENCES DE SMILE 4

SOMMAIRE..... 6

EXIGENCES DE L'APPLICATION WEB 8

- UTILISABILITE ET EFFICACITE 8
- DES PROCESSUS DE TRAVAIL PLUS COMPLEXES 8
- UNE UTILISATION REGULIERE 9
- SEDUCTION VS CONFORT..... 9
- PERFORMANCES 10
- COUTS ET INDUSTRIALISATION..... 11
- SIMPLICITE, FIABILITE, ADHESION 11
- ACCESSIBILITE 12

PRINCIPES GENERAUX..... 13

- WINDOWS 13
- CONSTANCE..... 13
- LA PAGE..... 14
- UN RESEAU DE PAGES 14
- BACK 15
- FRAMES..... 16
- FENETRES MULTIPLES..... 17

SESSIONS ET CONTEXTE DE TRAVAIL..... 18

- SESSIONS ET CONTEXTES 18
- TRANSACTION DE SAISIE ET CONTEXTE..... 19
- EXPLICITER LE CONTEXTE DE TRAVAIL 20
- TRANSACTIONS ET VERROUILLAGES 21
- TRANSACTIONS LONGUES 22

ENCHAINEMENTS..... 23

- ENCHAINEMENTS..... 23
- RECHERCHE-LISTE-DETAIL 24
- CREATION 25
- CREER N OCCURENCES..... 25
- UNE VARIANTE D'ERGONOMIE POUR LA MISE A JOUR..... 26
- LA MISE A JOUR STANDARD..... 27
- SUPPRESSION 27
- LE SCHEMA STANDARD 29
- SCHEMA RELATIONNEL ET SCHEMA NAVIGATIONNEL..... 30

MISE EN PAGE	32
AGENCEMENT SOUPLE	32
DEFILEMENT.....	32
MENUS	33
LISTES.....	33
LISTES INDEXEES.....	34
MEMORISER LES RESULTATS DE RECHERCHE	35
PRESENTATION DE LA LISTE	35
TABLEAUX ET PARAGRAPHERS	37
LES PETITES ICONES.....	39
LIENS	40
AIDE A LA SAISIE.....	40
A LA PLACE DE L'UTILISATEUR REGULIER	42
CONTROLE DE SAISIE.....	43
NOTIFICATION DES ERREURS.....	45
LISTES LIEES	45
BOUTONS ET LIENS	46
DIVERS.....	46
OBJETS D'INTERFACE	48
LIBELLES ET CHAMPS	48
BARRE DE PROGRESSION.....	48
ZONES DE TEXTE	49
LISTES DEROULANTES.....	49
LISTES A CHOIX MULTIPLES.....	50
CHOIX DE DATE.....	51
LES ONGLETS.....	52
ENTITES DE REFERENCE	53
SOURIS.....	54
RACCOURCIS CLAVIER	54
IMAGES ET TEXTES.....	55
PORTAIL, PERSONNALISATION, WORKFLOW.....	56
WORKFLOW	57
PORTAIL ET PERSONNALISATION	58
MOTEUR DE RECHERCHE	60
CONCLUSION.....	61

EXIGENCES DE L'APPLICATION WEB

Avant d'en venir aux recommandations, essayons d'analyser ce qui caractérise une application web, en particulier par rapport aux exigences d'un site Internet.

Utilisabilité et efficacité

Le terme consacré pour désigner la qualité de l'ergonomie est l'*utilisabilité*. Mais le sens de l'utilisabilité, n'a pas tout à fait le même sens pour une application web et pour un site Internet.

L'exigence première de l'application web est l'efficacité, c'est à dire le temps et les efforts que demande une tâche donnée.

A contrario, sur un site web on privilégie la lisibilité, la simplicité et le caractère *intuitif* de l'interface. Un site web doit faire l'hypothèse que le visiteur n'a pas été formé et en est peut-être à sa première visite : le visiteur d'un site web doit comprendre l'interface sans explication, dès la première fois. L'efficacité, si elle reste aussi un objectif, passe malgré tout au second plan.

Des processus de travail plus complexes

Par rapport à un site web, même incluant une part de transactionnel, les processus de travail d'une application web sont dans l'ensemble plus complexes et plus nombreux.

Acheter sur amazon.com, vendre sur ebay.com, rechercher un job sur cadremploi.fr ou faire une réservation sur sncf.com : ce sont à peu près les processus les plus complexes que l'on rencontre sur des sites transactionnels. Ils impliquent au fond un nombre réduit d'*entités* et de saisies.

Les applications web brassent couramment plusieurs centaines d'entités fondamentales, et un processus de travail peut impliquer une dizaines d'étapes ou plus.

Nous verrons plus loin que l'un des objectifs dans la conception de l'interface web sera de décomposer les processus longs en étapes simples.

Une utilisation régulière

Une application web est le plus souvent utilisée de manière régulière, sinon tous les jours, du moins une fois par semaine.

Simplicité et caractère intuitif restent d'actualité, mais on peut compter sur un *apprentissage* de la part des acteurs de l'application.

Supposons que la British Airways ait décidé de déployer à l'usage du personnel de ses points de vente une application web pour la vente de billets et de réservation. Il est clair que l'ergonomie de cette application ne sera pas celle du site grand public de la compagnie. Dans un cas le concepteur visera la rapidité du travail, l'économie de gestes, en somme le *rendement*, pour qu'une transaction ne dure pas plus d'une minute. Dans le cas du site grand public, c'est le caractère lisible et intuitif qui est la principale exigence, de sorte que *même un visiteur qui en est à sa première visite* et ne commandera peut être qu'un billet par an, puisse réussir l'opération.

Séduction vs confort

Un site web a souvent un objectif de séduction. L'internaute qui est arrivé sur le site n'a généralement pas un besoin impérieux d'y rester et on a quelques secondes pour le convaincre qu'il trouvera ici ce qu'il recherche.

Sur une application web, l'utilisateur est là parce que c'est son travail. Inutile donc de chercher à le séduire, on risquerait de l'énerver plutôt.

L'utilisateur reste bien sûr au centre de nos préoccupations, mais c'est son *confort* qui prime, et non notre capacité à l'attirer et à le retenir. Cette exigence de confort se concrétisera par l'utilisation de couleurs bien contrastées, mais non agressives, par la constance des mises en pages, le recours à des enchaînements standardisés, etc.

De plus, alors qu'un site web cherchera à profiter de la transaction en cours pour faire valoir d'autres services auxquels le visiteur aurait pu échapper, une application web ne cherchera jamais à détourner l'attention de l'action en cours, et interdira totalement les animations en marge ou en bandeau.

Attention toutefois :

Si la séduction est inutile, l'esthétique est importante : une application web peut être sobre, mais elle doit être agréable, tant à l'œil qu'à l'usage. C'est pourquoi l'intervention d'un graphiste est toujours nécessaire.

Performances

Efficacité, rendement, moindre séduction, ces facteurs amènent à privilégier des pages de faible poids. Pourtant, une application web est souvent déployée sur une infrastructure de réseau local, où le poids des pages importe beaucoup moins.

L'exigence d'efficacité, comme celle de confort, impose plus encore que sur un site web, des temps de réponse excellents, disons de l'ordre du dixième de seconde. Les sites web ne mettent généralement pas la barre si haut, même s'il est établi que la performance est l'un des principaux facteurs de succès.

Dans son ouvrage, Jakob Nielsen¹ cite une étude qui définit trois seuils dans les temps de réponse : jusqu'à un dixième de seconde, un rafraîchissement est perçu comme instantané, jusqu'à une seconde, il est sensible, mais ne suspend pas le fil de pensée de l'utilisateur, et enfin jusqu'à 10 secondes, il suspend la pensée, mais pas la concentration. Au delà de 10 secondes, le cerveau de l'utilisateur est parti ailleurs. Si ces considérations sont importantes pour les sites, on conçoit qu'elles sont fondamentales pour l'outil de travail qu'est une application web : ces secondes perdues, et ces cassures dans les processus de travail coûtent cher à l'entreprise.

Pour un site Internet, la règle est donc de ne pas dépasser les 10 secondes, ce qui correspond à environ 40 KO pour des connexions modem.

Pour une application web, on peut parfois supposer des connexions soit réseau local, soit haut-débit, ce qui donnera des temps de transfert compris entre 1/10^{ème} de seconde et une seconde.

Il serait erroné d'utiliser cette bande passante pour construire des pages plus volumineuses : c'est contraire aux principes d'ergonomie web.

Enfin, même si une application est déployée sur un réseau local, au sein d'un même établissement, il faut toujours la prévoir compatible avec des interconnexions de réseaux, moins rapides.

En effet, l'une des raisons pour lesquelles les entreprises font le choix d'applications web plutôt que client-serveur est précisément la capacité à déployer au travers d'interconnexions de réseaux, que ce soit en national ou international. Et l'expérience montre que tôt ou tard, le besoin d'un accès distant à l'application apparaîtra.

Il faut ajouter également que dans le contexte des applications web, les temps de réponse dépendent souvent plus du traitement coté serveur, qui peuvent être complexes, que du poids des pages.

¹ « Conception de site web : l'art de la simplicité » - Jakob Nielsen – 2000.

Coûts et industrialisation

Pour une application web comme pour un site, le coût de réalisation ne peut pas être oublié.

Mais pour une application web, la conception de l'ergonomie peut avoir une incidence très forte sur ce coût. En effet, lorsqu'une application gère une base de 300 entités, et comprend plusieurs milliers de transactions, le gain qui peut être attendu d'un processus de développement industrialisé devient très important.

Le processus de développement industrialisé s'appuiera sur des techniques de mutualisation de code, ou de génération de code, qui sont particulièrement performantes si l'ergonomie est construite sur la répétition de schémas standards.

Or il faut souligner que cette standardisation à des fins économiques n'intervient pas au détriment de la qualité de l'interface, au contraire, elle y concourt en instaurant des automatismes dans l'utilisation de l'application.

Simplicité, fiabilité, adhésion

L'un des défis les plus importants dans le déploiement d'une nouvelle application web est *l'adhésion des utilisateurs*.

Les bonnes pratiques d'utilisabilité feront beaucoup pour cette adhésion.

Mais il est un autre facteur capital : la fiabilité. L'utilisateur doit avoir une confiance totale dans l'application, tant dans l'exactitude de ce qu'elle fournit, que le caractère prévisible et sûr de ses réponses. Des anomalies que l'informaticien trouverait légères peuvent provoquer un rejet irréversible de la part des utilisateurs.

A quoi tient la fiabilité ? Au savoir-faire des informaticiens ? A la méthodologie ? Aux outils de développements ? A tout cela certainement, mais aussi à la simplicité de l'application, on pourrait l'oublier.

Nous avons vu parfois des applications qui avaient tellement recherché la perfection que leur complexité les rendaient irrémédiablement instables.

C'est donc aussi une des exigences à garder à l'esprit :

Simplicité → Fiabilité → Adhésion

Accessibilité

L'accessibilité au personnes handicapées est trop souvent l'exigence oubliée d'une application web.

Beaucoup de développeurs ignorent même l'existence d'un document de recommandations du W3C en la matière.

Pourtant certaines de ces recommandations sont très faciles à appliquer, et ne pèsent en rien ni sur le développement, ni sur l'esthétique de l'interface.

Ainsi par exemple, le W3C demande que toutes les images aient une description textuelle (balise 'alt'), qui permettra aux logiciels spécialisés de décrire l'image, cela tout particulièrement si l'image est un lien.

Il y a des exigences plus contraignantes, qui visent d'une manière générale, à permettre à des logiciels spécialisés d'analyser la page et de la restituer aux non ou mal-voyants de manière intelligible.

Le W3C définit différents degrés de conformité aux exigences d'accessibilité, notre recommandation est que pour des applications déployées dans de grandes organisations, la conformité au niveau 'A' doit être visée.

Pour plus d'information : <http://www.w3.org/TR/UAAG10/>.

PRINCIPES GENERAUX

Windows²

D'une manière générale, l'une de nos premières recommandations est la suivante

N'essayez pas de singer l'ergonomie Windows.

Bien souvent, des utilisateurs ou développeurs habitués à l'interface des applications Windows essayent de l'imiter sur des applications Web. C'est une erreur fondamentale. Le web a ses caractéristiques et limites propres, qui impliquent *une autre façon de penser l'ergonomie*.

Certains développeurs, croyant que le *nec plus ultra* de l'ergonomie web c'est Windows, arrivent à faire des applications web avec boîtes de dialogue reprenant l'iconographie Windows, barre de boutons, popup de confirmation, menu contextuel. Ces applications sont toujours inutilisables : l'utilisateur ne sait plus dans quel monde, dans quel référentiel ergonomique, il se trouve. Le « glisser-déposer » va-t-il fonctionner ? Le clic droit ? Où sont passés les liens hypertextes ? En outre, ces applications sont généralement très lentes.

On notera qu' à l'inverse, Windows adopte parfois les bonnes pratiques du web. Les 'assistants' de Windows ont retenu la démarche linéaire du web : une question à la fois, un bouton 'précédent', un bouton 'suivant'. C'est une démarche simple et universelle, qui prend l'utilisateur par la main, et le rassure en lui laissant toujours la possibilité de changer d'avis et de revenir en arrière.

Constance

Le plus important dans l'ergonomie des applicatifs web, c'est la permanence des usages : le même enchaînement doit être traité de la même manière d'un bout à l'autre de l'application.

Il faut donc viser la constance d'avantage que la perfection.

Si telle page de l'application semble pouvoir être rendue meilleure en faisant une exception aux principes généraux de navigation, ce sera le plus souvent une démarche erronée : le cas particulier de cette page perturbera l'utilisateur en l'empêchant d'appliquer ses automatismes.

² Microsoft n'a pas inventé avec Windows le principe des interfaces graphiques, qui remonte au projet 'Coyote' du Xerox Parc (1975), en passant bien sûr par le Macintosh. Mais aujourd'hui la référence pour beaucoup est bien l'ergonomie de leur poste de travail Windows et de ses applications, que nous appellerons ici, par raccourci, 'ergonomie Windows'.

La page

L'ergonomie web a pour fondement la notion de *page*.

La décomposition du dialogue en pages ne doit pas être vue comme un contrainte technique regrettable que l'on essaierait de masquer ou de contourner. La notion de page fait partie du paradigme d'interface.

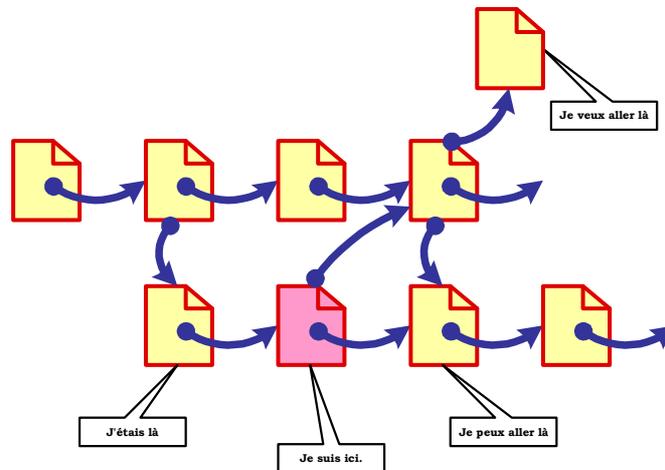
Par *paradigme d'interface*, on entend la représentation que l'utilisateur se fait – parfois inconsciemment – de l'interface. L'utilisateur *perçoit* qu'en cliquant sur des liens il adresse des *requêtes* à l'application, et que celle-ci répond à ces *requêtes* en lui adressant les *pages* correspondantes.

Tout ce qui casse ce paradigme va déstabiliser l'utilisateur. On pourrait tenter par exemple de gérer des onglets au sein d'une page, qui feront apparaître instantanément plusieurs pavés différents au même endroit de la page. En première analyse, il semble que ce contournement habile de la contrainte requête-page facilitera la vie de l'utilisateur, lui permettant *comme il le ferait dans Windows*, de passer d'un onglet à l'autre sans le moindre délai. En réalité cette technique risque de casser la *logique* de l'interface, la compréhension que s'en fait l'utilisateur. « J'ai saisi une valeur dans un onglet, j'ai changé d'onglet, ma saisie est-elle enregistrée ? » Je n'en sais rien parce que je ne sais plus quand (et si) ma requête a été envoyée.

Redisons-le : la logique requête-page est perçue par l'utilisateur, et contribue à sa compréhension des échanges, il n'est pas bon que la conception d'interface cherche à nier cette logique.

Un réseau de pages

Comme un site web, une application web est constituée d'un ensemble de pages liées entre elles. A un instant donné, l'utilisateur est 'positionné' sur l'une des pages, comme il pourrait être situé dans une station de métro parisien.



C'est une chose sur laquelle il faut insister : cette perception de l'application en tant qu'un ensemble de pages organisées en réseau est très différente de la perception d'une application Windows.

Et cette représentation mentale du réseau de pages est un facteur essentiel dans l'utilisabilité de l'application : l'utilisateur se positionne dans ce réseau.

Il sait qu'il peut revenir en arrière et revoir la page sur laquelle il était tout à l'heure, il perçoit que des liens le conduiront à d'autres pages, mais que de ces autres pages il pourra revenir là où il est, s'il s'était trompé.

Cheminer dans ce réseau de pages comme on chemine dans le métro est tout à fait rassurant.

A contrario, imaginez une application Windows qui vous présente une boîte de dialogue. Vous remplissez du mieux que vous pouvez les différents champs proposés, cochez des cases, sélectionnez des boutons. Puis enfin vous validez et la boîte de dialogue se ferme. Erreur : la conséquence de votre saisie n'est pas celle que vous souhaitiez. Comment revenir à la boîte de dialogue ? Ce n'est pas si facile... Peut-être aurez vous oublié quelle action avait ouvert cette boîte. Comment l'obtenir à nouveau ?

Dans une navigation web, on peut *toujours* revenir sur ses pas, grâce au merveilleux bouton 'précédent' ('back').

Back

Comme on l'a vu, le bouton 'précédent' fait partie intégrante de l'ergonomie Web : ce n'est pas une *'possibilité intéressante'* ajoutée par hasard au navigateur, c'est l'un des fondements de l'ergonomie. C'est – entre autres – ce qui rassure l'utilisateur sur le fait qu'il peut sans risque aller voir telle page, puisqu'il pourra toujours revenir en arrière.

Il est donc rigoureusement interdit d'interdire l'utilisation du bouton 'précédent'.

Une des nombreuses manières de perturber le bon fonctionnement du bouton 'précédent' est d'insérer une page d'attente provoquant après un certain délai une redirection vers la page de résultat. C'est ce que l'on constate sur le site de la SNCF, qui devrait pourtant être un modèle du genre, étant certainement le site transactionnel le plus visité de France. Ainsi, si après avoir obtenu une liste de trains correspondants à votre recherche vous cliquez sur 'précédent' afin de modifier vos critères, vous retrouvez la publicité de la page d'attente, qui vous ramène automatiquement sur la même page de résultat : impossible de revenir en arrière avec le bouton 'précédent' !

Frames

L'utilisation des *frames* a été quelques temps l'un des moyens utilisés pour contourner le paradigme requête-page, en permettant à des *morceaux de pages* d'être réactualisés tandis que d'autres restaient inchangés. Cette actualisation partielle était vue comme un moyen d'économiser de la bande passante et donc d'améliorer la réactivité de l'interface.

Mais, on l'a vu, le poids des pages est rarement le facteur déterminant dans la performances d'applications web. Economiser 15% en évitant de rafraîchir une partie de la page n'aura pas une incidence visible sur les performances. De plus il existe d'autres techniques pour conserver une partie de page en cache du navigateur sans recourir à des frames.

Les *frames* peuvent être utilisées sans inconvénient – mais sans bénéfice important - lorsqu'une partie de la page, typiquement un bandeau et un menu est *totalelement fixe*.

En revanche, il faut éviter d'utiliser des frames pour faire varier successivement telle ou telle partie de la page. Cette approche, en cassant le paradigme de la page, apporte d'importantes complications, tant pour le développement que pour l'utilisateur.

Entre autres inconvénients, l'usage des frames casse le principe de correspondance URL – Page. Il s'ensuit différents dysfonctionnements dans la navigation, la mémorisation d'URLs en tant que signets (*bookmarks*), ou bien l'indexation par des moteurs de recherche.

Sur une page multi-frames, l'URL visible par l'utilisateur ne correspond pas à la page affichée, ce qui nuit à sa compréhension du positionnement.

Fenêtres multiples

L'utilisation de fenêtres multiples dans une application web est à proscrire.

Ceci pour plusieurs raisons :

- Le principe du multi-fenêtrage est issu de Windows et n'appartient pas au paradigme de l'interface Web. Certes certains utilisateurs ont l'habitude d'ouvrir plusieurs fenêtres de navigateur, mais ce n'est pas le principe d'interface qui les y conduit.
- D'autre part, et c'est essentiel :

l'interface web offre naturellement à l'utilisateur la possibilité d'ouvrir une nouvelle fenêtre s'il le souhaite. Inutile donc de le lui imposer.

Dans la pratique, on constate que les utilisateurs ont rarement le souhait d'ouvrir plusieurs fenêtres dans une application web, car cela compliquerait beaucoup leur perception de l'application.

En outre, les applications web contrôlent mal l'empilement des fenêtres, de sorte qu'une fenêtre passée en dessous des autres sera la plupart du temps oubliée, laissant éventuellement une transaction inachevée.

De plus l'interface web ne permet pas de définir de manière fiable une fenêtre 'modale'. Dans les interfaces Windows, on appelle ainsi une fenêtre qui reste obligatoirement active tant qu'on ne l'a pas fermée ; on l'utilise pour des boîtes de dialogues. Sur une interface web, on contrôle difficilement quelle fenêtre est active, de sorte que l'utilisateur peut avoir activé une fenêtre pendant que l'application attend une action de sa part dans une autre fenêtre.

Rappelons-le : l'ergonomie web est par essence 'linéaire', c'est à dire qu'elle est basée sur le principe d'une succession de pages. A un instant donné, l'attention de l'utilisateur est focalisée sur une page et il n'a pas à se demander s'il devrait s'intéresser à autre chose.

Comme on le verra plus loin, nous acceptons une exception à cette règle de fenêtre unique, pour la gestion de l'aide à la saisie.

SESSIONS ET CONTEXTE DE TRAVAIL

Sessions et contextes

Par essence, le protocole Http est un protocole *sans connexion*, c'est à dire que rien ne relie entre elles les différentes requêtes venant d'un même utilisateur.

L'enchaînement des différents échanges avec un même utilisateur constitue une *session*.

La plupart des applications web ont besoin de mémoriser certaines informations au travers d'une même session, c'est à dire que le traitement d'une requête pourra dépendre d'informations mémorisées lors du traitement de requêtes antérieures de la même session. Le cas le plus courant est l'identification de l'utilisateur. Si l'accès à une application implique identification, alors le traitement des requêtes au sein de l'application n'est pas le même selon que l'utilisateur a été identifié au travers d'une requête antérieure ou non.

L'ensemble des informations mémorisées au travers d'une même session constitue le contexte de session. Ainsi le contexte de session peut contenir une variable indiquant s'il y a eu identification ou non au sein de la session, et le cas échéant, l'identifiant de l'utilisateur et son profil.

On dit qu'un tel contexte de session est géré « coté serveur », c'est à dire du coté de l'application, sans apparaître du coté du poste de travail, le « coté client ». Un tel contexte est un contexte *implicite*, c'est à dire que l'URL ne porte pas l'information de contexte.

Notre recommandation est d'utiliser le moins possible de contexte de session implicite.

Précisément, on ne rangera dans le contexte de session que des informations caractérisant la session dans son ensemble, et non pas des informations propres à un contexte de travail.

Quand nous disons *des informations caractérisant la session dans son ensemble*, il s'agit typiquement de l'identifiant utilisateur, de son profil, de ses préférences, par exemple la langue d'interface pour une application multilingue, etc...

A l'inverse, des informations *propres à un contexte de travail* seraient par exemple des critères de filtrage sur telles entités, le contenu d'un formulaire

issu d'une première étape de saisie, le numéro de la dernière page vue dans une liste, etc. Ces informations doivent être gérées *dans l'URL*.

Supposons par exemple que sur une page l'utilisateur définit des critères de recherche. L'application peut mémoriser ces critères coté serveur dans son contexte de session. Lorsque l'utilisateur revient à la page de liste, l'application retrouve les critères et filtre la liste en conséquence.

Cette technique présente plusieurs inconvénients :

- On casse le principe fondateur d'une association un-pour-un entre une URL³ et une page web. Si l'utilisateur définit un *bookmark*, ou bien sélectionne l'URL pour l'envoyer par e-mail à un collègue, cette URL *ne produira pas la page attendue*.
- L'association URL-page est également l'hypothèse de base des moteurs d'indexation, qui ne pourront donc pas œuvrer sur ces pages.
- Si l'utilisateur a ouvert deux fenêtres et définit dans l'une et l'autre des critères de recherche différents, l'application mélangera les deux contextes, et le résultat sera un filtrage imprévisible.
- De même l'utilisation du bouton 'précédent' ou plus largement de la fonction historique du navigateur sera totalement perturbée.

Ainsi imaginons que l'utilisateur a dans l'historique de son navigateur parcouru une page de liste filtrée selon des critères C_1 , puis plus tard la même page filtrée selon des critères C_2 . Lorsqu'il revient, par le bouton 'précédent' sur la première page, elle est restituée en cache correctement. Mais s'il demande à rafraîchir la page, il la verra soudainement filtrée selon les critères C_2 !!

Pour toutes ces raisons, on s'efforcera de conserver le lien URL-page, c'est à dire qu'une URL identifie une page de manière complète et unique, et donc de limiter l'utilisation d'un contexte de session coté serveur.

Transaction de saisie et contexte

Cette dernière remarque amène également la recommandation suivante :

Dans une transaction de saisie, la totalité des champs définissant la transaction doit être intégrée au formulaire lors de la validation, éventuellement sous la forme de champs cachés.

³ Pour être précis, il faudrait parler d'URI, 'Uniform Resource Identifier', qui est constituée de l'URL et des paramètres.

Supposons que l'application web enregistre un virement. L'opération est critique, par nature, et en particulier il ne doit y avoir aucune ambiguïté possible au moment de valider ce virement.

L'utilisateur a saisi les caractéristiques de son virement : compte débiteur, compte créditeur, libellé, montant. Supposons que l'application web ait conservé ces informations dans son contexte de session, puis ait présenté à l'utilisateur une page de confirmation du virement avec deux boutons : confirmer ou annuler. C'est typiquement une situation où le caractère implicite du contexte de session induit un risque d'erreur important.

Si l'utilisateur a entamé deux virements en parallèle, ou bien qu'il est revenu au moyen du bouton 'précédent' sur la page d'un premier virement, l'application mélange la validation du premier et du second. C'est pourquoi *le formulaire de confirmation doit comporter toutes les caractéristiques du virement en champs cachés, de telle manière que la validation du virement ne fasse référence à rien d'implicite : la transaction est entièrement décrite par elle-même.*

Expliciter le contexte de travail

On a vu précédemment que la capacité pour l'utilisateur à *se situer presque d'une manière géographique* au sein d'une l'application vue comme un réseau de pages, était un facteur essentiel d'utilisabilité.

Or comme on vient de le voir, dans l'applicatif web il ne suffit pas d'un nom de page pour décrire « où je suis ? ». Il y le plus souvent un véritable *contexte de travail*.

« Je suis sur la page qui liste les clients et j'ai demandé que cette liste soit filtrée pour ne présenter que les clients de la zone Amérique du Nord ».

L'utilisateur doit connaître à tout à la fois où il est, et également son contexte de travail, ce qu'il est en train de faire.

De même que nous avons proscrit l'utilisation d'un contexte implicite du point de vue technique, nous donnons pour règle que tous les éléments du contexte doivent être explicitement indiqués à l'utilisateur.

Pour que ce positionnement lui demande un effort minimal, il faut aussi que le contexte de travail soit aussi limité que possible.

Nous verrons ainsi qu'il ne faut pas autoriser des chemins de navigation qui construirait un contexte de travail trop large : *« Je suis en train de créer une nouvelle zone géographique, de manière à pouvoir finir de saisir un nouveau client, pour lequel j'ai commencé à saisir une commande. »* Un utilisateur normal ne retiendra pas un contexte de travail de ce genre.

En synthèse :

Utiliser le moins possible de contexte de travail implicite
Limiter le contexte de session aux informations de session
Rappeler à l'utilisateur les informations de contexte

Transactions et verrouillages

La gestion des transactions au sens de la base de données est un problème qui dépasse l'ergonomie, mais qui a néanmoins des impacts sur celle-ci.

La question est la suivante : que se passe-t-il lorsqu'un utilisateur commence à modifier une entité ? Cette entité doit-elle être verrouillée pour tous les autres utilisateurs ? Que se passe-t-il si un second utilisateur engage à son tour une modification de la même entité ? Faut-il le prévenir ?

Ce n'est pas tant au plan technique qu'il faut analyser cette question, mais sous l'angle des processus de travail.

Que conviendrait-il de faire si Jean a commencé à modifier le client A et que Marie veut aussi modifier le client A ?

D'abord le cas est-il réellement probable ? Jean et Marie s'occupent-ils des mêmes clients ?

Faisaient-ils la même modification ? Auquel cas, il n'y a pas de problème. Faisaient-ils des modifications incohérentes ? Auquel cas il y a peut-être un problème d'organisation qui n'a rien à voir avec notre application.

Dans tous les cas si deux utilisateurs dans deux bureaux différents entreprennent de modifier le même client, le fait que l'un valide sa transaction avant l'autre est totalement aléatoire, de sorte qu'il n'y a aucune raison de penser que le premier soit à privilégier par rapport au second. Et de même, le timing étant imprévisible, le second utilisateur aurait pu commencer sa transaction juste *après* la validation du premier. Dans ce cas, il n'y aurait pas eu de conflit mais la seconde modification aurait néanmoins écrasé la première.

En synthèse, sauf cas très particulier, notre recommandation est que dans une application web on ne cherche pas à verrouiller les entités en cours de modification, et la dernière modification l'emporte toujours.

D'autre part, souvenons nous que la simplicité fera la fiabilité, qui décidera de l'adhésion des utilisateurs.

Transactions longues

Il convient d'éviter les transactions s'étendant sur plusieurs pages. En saisie, chaque page web de formulaire est validée indépendamment des précédentes et suivantes.

Il peut être nécessaire que des phases de saisie s'étendent sur plusieurs pages web.

Lorsqu'il valide une page de formulaire, l'utilisateur doit être assuré que sa saisie ne sera pas perdue quoi qu'il advienne, c'est à dire même si un événement l'oblige à interrompre la saisie à la page suivante.

Il s'ensuit bien sûr qu'il peut y avoir des saisies incomplètes, qui pourront éventuellement être reprises lors d'une session prochaine, ou bien seront purgées après un certain délai.

Prenons le cas d'un site de crédit en ligne, dans lequel l'utilisateur est invité à saisir un dossier portant sur son projet d'investissement immobilier.

Un tel dossier représente plus de 100 champs de saisie, qui peuvent être liés entre eux ; il n'est donc pas possible de l'envisager au sein d'un unique formulaire.

Du point de vue de l'intégrité des données, on pourrait dire : « il est interdit d'enregistrer un dossier incomplet, et par conséquent la création d'un dossier est une opération 'insécable' ».

Mais ce que veut l'utilisateur avant tout, c'est avoir l'assurance de ne jamais perdre son travail. Aussitôt qu'il a saisi et validé un premier formulaire, il doit pouvoir le retrouver, même le lendemain.

ENCHAINEMENTS

Enchaînements

Avant de s'intéresser aux *éléments d'interface* proprement dits, tels que menus, boutons, tableaux, liens, icônes, etc., il est important de traiter des *enchaînements de pages*.

Un processus de travail implique le plus souvent non pas une page mais une succession de pages de l'application. Ainsi un utilisateur pourra successivement : rechercher un client, visualiser les informations relatives à ce client, visualiser les factures de ce client, accéder à une facture, indiquer que la facture est payée.

Les processus de travail habituels peuvent se ramener à un petit nombre de schémas d'enchaînements standards, qu'il faut mettre en œuvre de manière systématique et homogène.

Cela pour deux raisons :

- Du point de vue de l'utilisateur, les schémas d'enchaînements seront intégrés comme des automatismes. S'il vient un jour dans une partie de l'application qu'il connaît moins bien, il retrouvera ses marques, et saura immédiatement comment procéder.
- Du point de vue du développement, la répétition est bien sûr la base de l'industrialisation. Les mêmes schémas d'enchaînements conduisent à des transactions construites sur les mêmes modèles, qui pourront donc être générées de manière semi-automatique, ou bien dériver de bases communes.

Quels sont ces schémas ?

Recherche-Liste-Détail

L'enchaînement recherche-liste-détail est bien sûr le schéma de navigation le plus courant, que l'on retrouve en particulier dans les moteurs de recherche.

L'utilisateur indique ce qu'il recherche, l'application lui restitue une liste d'items présentés chacun de manière synthétique, avec un lien conduisant à une page de détail, une forme de *zoom* sur un item particulier.

Le schéma précédent peut se représenter, de manière plus concrète, comme suit :



Création

La création d'une entité nouvelle doit être proposée en plusieurs points dans le schéma standard de navigation :

- Avant la recherche.
- Sur la page de liste
- Eventuellement sur la page de détail également.

La création ne doit pas pouvoir être confondue avec la modification. Ainsi il ne faut pas que sur une page de modification l'on puisse provoquer une création d'entité nouvelle simplement parce que l'on aura modifié un champ qui sert de clé ! La création doit être bien perçue comme telle par l'utilisateur.

Créer N occurrences

Pour une utilisation en saisie de masse, il est courant que l'on souhaite créer plusieurs entités sur un même formulaire, comme par exemple dans cet extrait de page de commande de billets SNCF.

Si vous recherchez un tarif spécifique (promo, découverte, abonnement forfait)

Passagers *	Age / Catégorie	Choisir une carte
1	Adulte entre 26 et 59 ans	Choisir une carte
0		Choisir une carte

[Info cartes](#)

Il est clair que l'on a ici à gérer une relation un-plusieurs : une commande fait référence à plusieurs passagers. Gérer cette saisie selon le schéma standard serait fastidieux : cliquer sur 'ajouter un passager', page de formulaire nouveau passager, remplir le formulaire, valider, retour à la liste avec le nouveau passager ajouté, et recommencer pour le passager suivant.

La solution du formulaire à plusieurs lignes semble donc plus efficace.

Mais dans des cas un peu plus complexes, elle peut poser des problèmes : la conformité de chaque enregistrement doit être vérifiée séparément, et les erreurs signalées à l'utilisateur.

Cela pose plus largement la question des saisies 'de masse', qui est traitée plus loin.

La création de plusieurs entités au moyen d'un unique formulaire n'est pas à interdire strictement, mais doit être limitée aux cas où il n'y a pas de règle de gestion spécifique pour chacune des entités car le traitement de ces erreurs serait trop complexe, tant pour l'application que pour l'utilisateur.

Il faut garder à l'esprit également que la bande passante plus importante généralement disponible amène moins à éviter à tout prix un rafraîchissement de page. Ainsi l'ergonomie suivante pour ajouter à une liste, est-elle tout à fait satisfaisante pour une application web :

Si vous recherchez un tarif spécifique (promo, découverte, abonnement forfait et fréquence).

Passagers * 1 Adulte entre 26 et 59 ans CARTE SENIOR [supprimer]

1 Adulte entre 12 et 25 ans [supprimer]

1 Adulte entre 12 et 25 ans Choisir une carte Ajouter

[Info cartes](#)

Ici, chaque passager est ajouté unitairement à la liste. L'opération « ajouter » peut être traitée en moins d'une seconde.

www.smile.fr

Une variante d'ergonomie pour la mise à jour

Parmi les variantes d'ergonomie utilisées pour la modification, l'une présente quelque intérêt car elle permet de gérer liste, modification et même création sur la même page :

Nom ou raison sociale	Numéro	Type de client
<input type="radio"/> AIR FRANCE	21235	SOCIETE
<input type="radio"/> BRITISH AIRWAYS	31157	ASSOCIATION (AÉR)
<input type="radio"/> BS TARZ	39125	SOCIETE
<input checked="" type="radio"/> SMILE SA	14237	MOTORISTE INTERN
<input type="radio"/> TEXACO INC.	32558	SOCIETE

20927 occurrences , aller à la page

Modifier		
SMILE SA	14237	Motoriste Internet

Cette technique utilise des boutons-radios associés à du code javascript pour gérer la sélection de ligne. Les champs de la ligne dont le bouton est sélectionné sont automatiquement copiés dans le petit formulaire de saisie du bas, où ils peuvent être modifiés. Une fois le formulaire validé, la liste est rafraîchie pour faire apparaître le changement. Un bouton 'nouveau' peut remettre le formulaire à blanc pour accueillir un nouvel item, et de même un bouton 'supprimer' agira sur la ligne sélectionnée. Dans une version encore plus élaborée, le formulaire de modification peut faire apparaître plus de champs qu'il ne figure dans la liste, les valeurs de ces autres champs étant gérées par le programme javascript. Ce serait donc aussi une manière d'accéder directement au détail de chaque item.

Malgré quelques avantages, nous ne recommandons pas cette technique de mise à jour 'mono-page.

Ceci pour les raisons suivantes :

- Elle fait un usage non-standard des boutons radios
- Elle requière *beaucoup* de code javascript intégré à la page et se code est assez spécifique aux entités gérées. Une petite *base de données* image de la liste est gérée au sein du javascript pour permettre la recopie dans le formulaire de modification.
- Elle n'est pas parfaitement explicite quant aux créations, mises à jour et suppressions.
- Elle n'est pas applicable à tous les cas de figure, et donc elle nuit à l'homogénéité des principes d'ergonomie au sein de l'application.

La mise à jour standard

 [Aéronefs](#) > [Cnra](#) > [Recherche](#) > [Liste](#) > [Détail](#) > [Examens](#) > [Détail](#) > Modification d'un Examen

Dossier sélectionné :	
N° du dossier CNRA	A3138 - 12
Immatriculation	F-PRES
Date de l'examen	10/01/1997
N° de révision	<input type="text" value="10"/>
Inspecteur *	<input type="text" value="1"/> [1] - M. VOLANT ALBERT
Lieu de l'examen *	<input type="text" value="AERODROME"/>
Dernier examen	Non
Montant des frais (devise)	<input type="text" value="450"/> <input type="text" value="USD"/> [USD] - Dollar US
Nb de kilomètres parcourus	<input type="text" value="377"/>
Commentaires	<input type="text"/>
<input type="button" value="Enregistrer"/> <input type="button" value="Annuler"/>	

La page ci-dessus est un exemple de mise à jour standard. Nous verrons plus loin les mise en œuvre possibles de cette aide à la saisie.

Suppression

La suppression peut être proposée sur la page de liste, en face de chaque item, ainsi que sur la page de détail. Si la suppression est peu fréquente, on peut éviter de la proposer au niveau de la liste: dans ce cas il faut accéder au détail avant de supprimer. C'est une étape de plus, mais c'est aussi une sécurité supplémentaire: l'utilisateur visualise le détail de ce qu'il va supprimer.

Dans tous les cas, bien sûr, on présentera un message de confirmation avant d'exécuter la suppression.

Une variante d'ergonomie couramment utilisée pour la suppression multiple consiste à présenter une case à cocher en face de chaque item de la liste, et un bouton permettant de supprimer tous les items sélectionnés.

C'est ce que l'on retrouve par exemple sur Yahoo!mail :



L'utilisateur peut sélectionner autant de *mails* qu'il souhaite, puis cliquer sur le bouton 'Delete'.

Cette technique des 'check and delete' est à réserver aux cas où les suppressions multiples sont réellement un cas d'utilisation courant.

En règle générale, on préférera offrir des liens de suppression unitaire, soit sur la page de liste, soit plutôt sur la page de détail.

En effet :

- Elle alourdit la page de liste
- Elle présente un petit risque de suppression erronée
- Mais surtout, dans les applications web, le besoin de suppression *en masse* n'est pas courant.

Dans une application un peu élaborée, les requêtes ne sont pas juste exécutées, le traitement des requêtes coté serveur met en application des *règles de gestion métier*. Ainsi telle requête d'insertion, de modification ou de suppression peut échouer parce qu'elle enfreint certaines règles. Dans ce cas, il convient de signaler à l'utilisateur que la requête n'a pas été traitée et pourquoi.

C'est une des raisons qui font que les traitements de masse, c'est à dire impliquant un ensemble de requêtes groupées, sont à éviter dans une application web.

D'une part il est difficile de restituer à l'utilisateur un ensemble d'erreurs intervenues pour son ensemble de requêtes. Imaginons le message suivant :

« Certaines suppressions n'ont pu être effectuées, pour les raisons suivantes :

- client 'Banque de France' : ne peut être supprimé car il existe des factures associées.

- client 'Nokia' : ne peut être supprimé car la date de création est antérieure de moins de 3 mois.

- client 'Dell' : ne peut être supprimé car ... »

On pourrait, certes, réaliser le traitement dans l'application pour produire ce message de compte-rendu d'opérations groupées.

Mais mettons nous à la place de l'utilisateur qui obtient un tel message, et cherche à traiter une à une toutes ces erreurs : il imprimera sans doute le message, et regrettera certainement de ne pas avoir supprimé les entités une à une.

Le schéma standard

Le schéma d'enchaînement précédent, complété par les pages de création et de modification, devient donc le suivant :

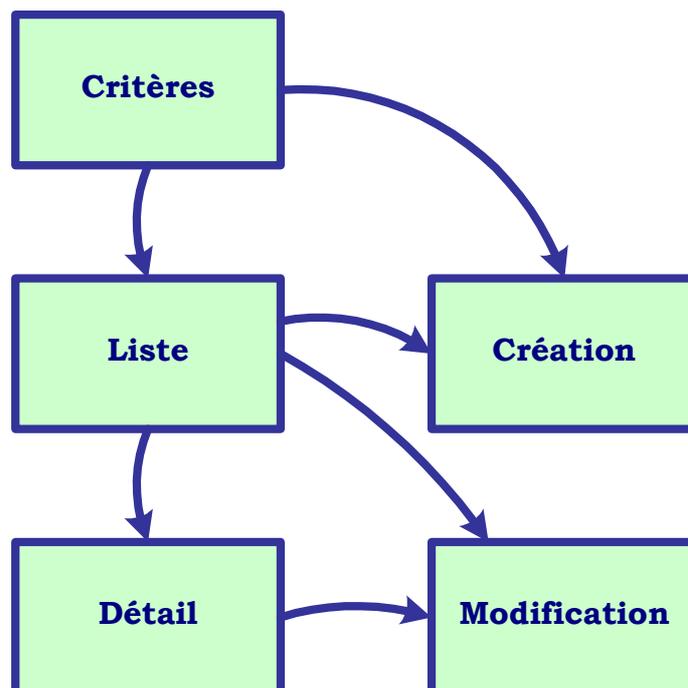
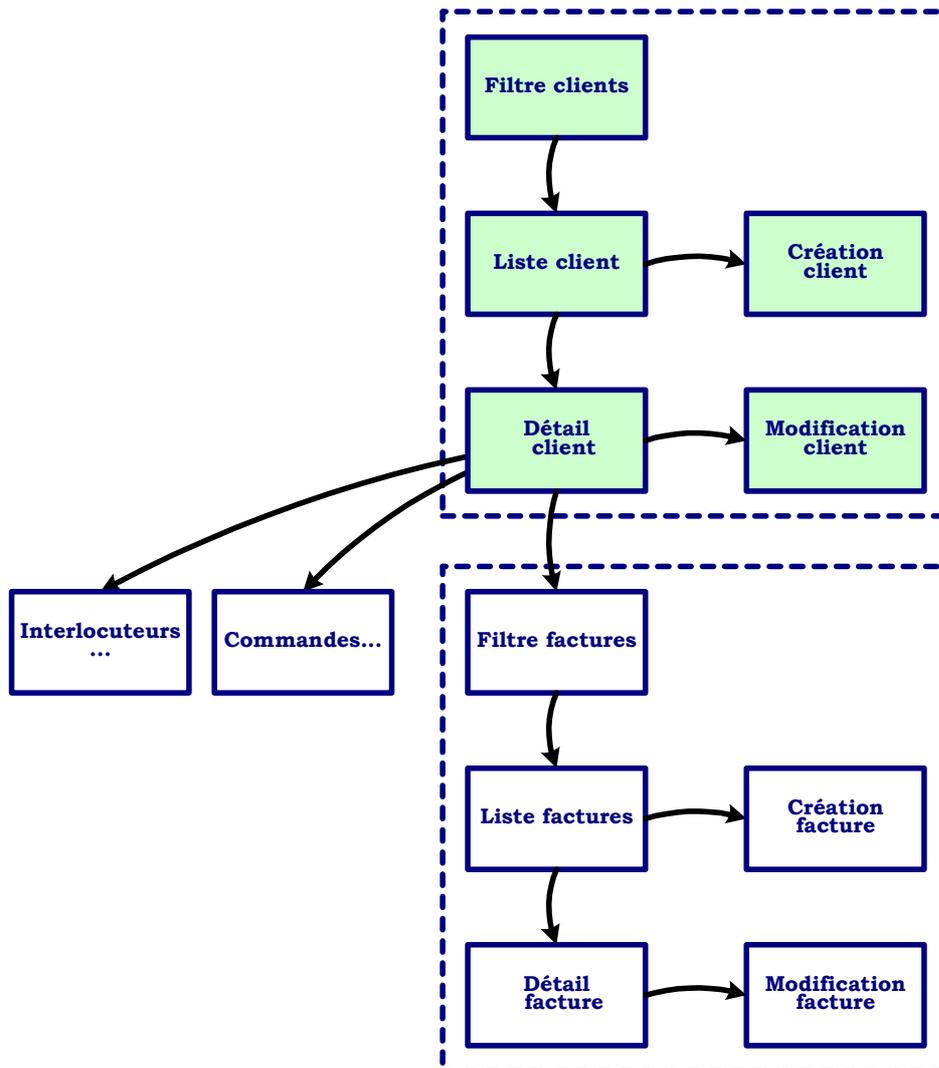


Schéma relationnel et schéma navigationnel

www.smile.fr

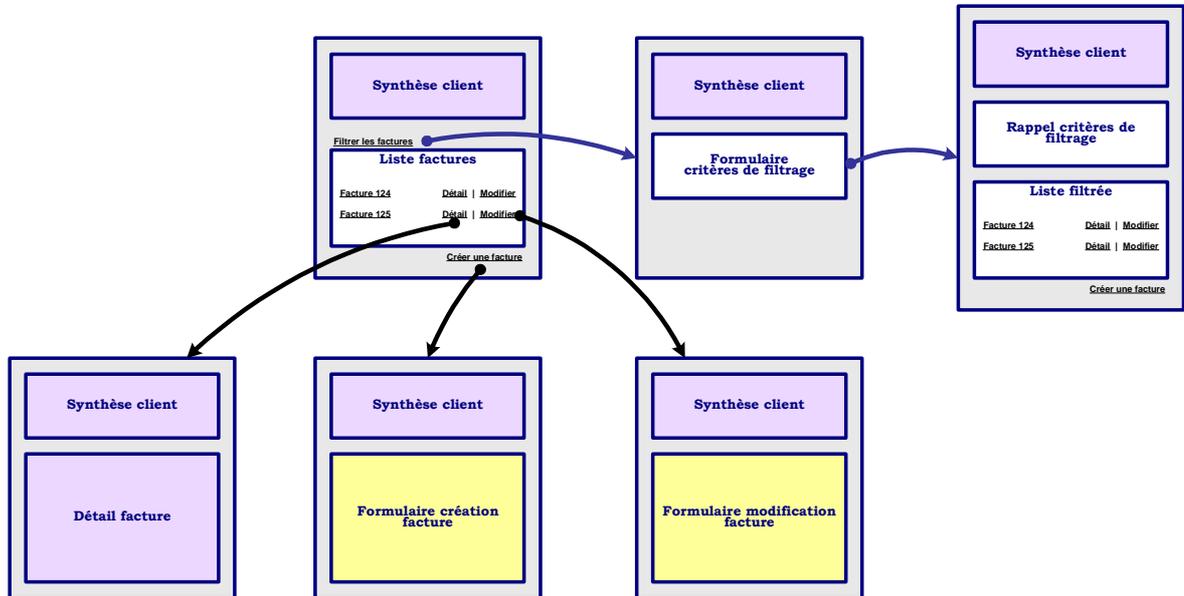


Le schéma de navigation ci-dessus fait apparaître deux applications identiques d'un même modèle recherche/liste/détail, avec création et modification, dans le cas d'une relation un-plusieurs.

Le schéma de navigation entre les entités manipulées par l'application peut-être construit en assemblant des schémas standards de navigation, et en proposant de manière systématique des liens vers les entités associées.

C'est en quelques sortes le principe de la navigation web appliqué au modèle relationnel : s'il y a une relation entre l'entité « filiale » et l'entité « établissement », alors la page « filiale » doit proposer un lien vers la liste des

établissements de la filiale, et ainsi pour toutes les relations du modèle de données.



MISE EN PAGE

Agencement souple

Le Html ne décrit pas une page de manière stricte, au pixel près, mais fournit *des principes d'agencement* qui permettent au navigateur de définir la meilleure mise en page possible.

C'est une règle de base, qui devrait s'appliquer également aux sites web. Mais si les sites web y font quelques entorses, les applications web doivent s'y conformer strictement.

Concrètement, cela signifie en particulier que l'application ne doit pas faire d'hypothèse particulière quant au nombre de pixels de l'écran, et doit utiliser au mieux l'espace disponible.

Quand la quantité d'information visible est un facteur important d'utilisabilité, et que l'on a la chance de disposer d'un grand écran, il est rageant de voir une application n'utiliser que 800 pixels et laisser un grand espace vide à droite du tableau.

La mise en page doit être souple et laisser le navigateur agencer la page selon la taille de la fenêtre.

Au passage, mentionnons une autre règle simple mais importante :

Lorsqu'une application ouvre une fenêtre secondaire, elle ne doit pas interdire le redimensionnement.

Cela n'a aucune utilité, et ne fera que frustrer l'utilisateur qui veut voir plus de choses. En outre si l'utilisateur fait varier les tailles de police de son navigateur pour une meilleure lisibilité, il verra apparaître de désagréables barres de défilement verticales et horizontales.

Défilement

Le défilement vertical est tout à fait accepté, le défilement horizontal est à éviter.

Menus

Lorsque le nombre d'options et de sous-options est important, les menus déroulants permettent à la fois de gagner de la place dans la page et de maintenir toutes les fonctions directement accessibles.



Il existe différentes techniques permettant de mettre en œuvre des menus déroulants de manière compatible avec toutes plateformes, et c'est donc un emprunt à l'ergonomie Windows que nous acceptons volontiers.

Les menus déroulants permettent de gagner en place et en efficacité ; ils sont donc bien adaptés aux besoins des applications web.

En revanche, nous préférons les implémentations qui demandent de cliquer pour ouvrir un menu à celles qui fonctionnent au 'survol' de la souris (rollover). En effet, les menus fonctionnant au survol sont difficiles à manipuler. Ainsi dans l'exemple précédent, si l'utilisateur amène sa souris en ligne droite depuis le choix 'Organisation' jusqu'au sous-choix 'Délégation', le sous-menu entier risque de disparaître.

Listes

La première règle en matière de liste est simple :

Toutes les listes doivent être paginées, la pagination doit faire apparaître autant que possible le nombre de pages, et permettre d'aller directement à une page donnée.

Mais il y a bien d'autres choses à dire en matière de listes.

En particulier :

Au delà de 4-5 pages, une liste fait perdre du temps à l'utilisateur, s'il doit parcourir les pages de la liste pour trouver ce qu'il recherche.

Dans une liste importante, l'accès par le seul numéro de page consiste en fait à rechercher à tâtons : l'utilisateur ne sait pas si l'information qu'il recherche est plutôt en page 7 ou plutôt en page 8.

Même avec un dispositif permettant à l'utilisateur d'accéder directement à la page 73, une liste de 87 pages ne présente aucun intérêt. L'utilisateur n'a aucun moyen de savoir que ce qu'il recherche est à la page 73. Donc soit il parcourt toutes les pages, soit il *affine sa recherche*, c'est à dire qu'il fournit des critères plus précis, qui limiteront la taille de la liste.

Cela à moins que la présentation des items ne corresponde à une pertinence décroissante, comme dans un moteur de recherche. Dans ce cas la probabilité est forte que la première page contienne les items recherches. Mais si ce n'est pas le cas, alors la liste *n'a plus aucun intérêt*. Imaginez que Google vous réponde : il y a 8731 pages correspondant à votre recherche, et vous les présente dans un ordre arbitraire. Envisagez vous de parcourir ces pages ? Certes non ! Vous irez probablement affiner vos critères.

Si une liste fait plus de 4-5 pages, on peut directement inviter l'utilisateur à affiner sa recherche.

En fait, on l'aidera à travailler plus efficacement, car il perdrait son temps à parcourir cette liste.

Listes indexées

Nous avons parlé ci-dessus de listes bêtement paginées : 20 items par page et c'est tout.

L'usage de listes paginées est trop généralisé, alors même qu'on a vu que leur utilisabilité était limitée. Il faut leur préférer des listes indexées plus intelligemment.

Le cas le plus courant est bien sûr l'accès par lettre de l'alphabet, qui est à la fois rapide et intuitif.

Mais selon le contexte fonctionnel, on pourra trouver d'autres bonnes manières de décomposer une liste en pages autrement que par tranches fixes : *par direction, par pays, par établissement, ...*

Il est possible également de remplacer la pagination numérique, dénuée de sens, par une pagination intelligente, par exemple en indiquant comme dans un dictionnaire, la clé de début et la clé de fin de chaque page :

ABI-CED | CEF-MIN | MIP-PRO ...

L'utilisateur qui recherche l'entité commençant par « *Norme* » sait qu'il doit cliquer sur la troisième page : on lui donne la clé lui permettant d'accéder directement à la page voulue.

L'objectif est le même :

L'utilisateur ne doit pas avoir à tâtonner dans une liste pour trouver ce qu'il recherche.

Mémoriser les résultats de recherche

Sur le plan du développement, il faut savoir qu'il peut être difficile pour une application web de conserver la longue liste des 87 pages de résultat d'une requête. Soit l'application conserve une connexion à la base de données pour garder le jeu de résultat (un « *curseur* ») ouvert, dans ce cas elle bloque des ressources importantes pendant un temps indéterminé, ce qui est à éviter. Soit elle conserve cette longue liste dans son *contexte de session*, mais nous avons expliqué qu'il était conseillé de ne pas utiliser le contexte de session pour ce genre de chose. Soit enfin, elle répète la requête chaque fois que l'utilisateur demande un nouvelle page, c'est la technique la plus simple, qui est acceptable à moins que la requête ne soit particulièrement complexe.

Lorsque la liste présentée correspond à une requête complexe, dont l'exécution pèse sur les performances de l'application, il n'y a plus d'autre choix que de conserver la liste de réponse en mémoire. C'est le cas par exemple sur un site tel que Cadremploi.fr : le visiteur saisit les critères de sa recherche d'emploi. Cette recherche correspond à une requête complexe sur la base de données, qui peut inclure par exemple des critères 'plein-texte' sur le contenu des offres d'emploi. Dans ce cas, il est impératif de mémoriser le résultat de la requête, pour ne pas la répéter à chaque page.

Présentation de la liste

Plusieurs questions reviennent couramment dans la gestion des listes d'items :

- Où placer la frontière entre l'information disponible dans la liste et l'information relevant du détail ?

La réponse dépend bien sûr du contexte fonctionnel, mais on peut citer quelques recommandations générales.

L'information présentée dans la liste doit être suffisante pour évaluer l'intérêt de chaque item par rapport à sa recherche, et ne pas accéder aux pages détail 'à tâton'.

C'est la raison pour laquelle les moteurs de recherche présentent un petit extrait de chaque page, et non pas une liste d'urls.

Conception d'applications web : efficacité et utilisabilité

Mais la page détail doit avoir *une vraie valeur ajoutée*, il n'est pas utile de zoomer pour n'obtenir que 20% d'information en plus.

Disons que sur 10 champs d'information décrivant chaque item, on en présentera entre 3 et 6 dans la liste, et le reste en détail.

- Combien d'items par page ?

Ici aussi, il n'y a pas de réponse absolue. On estime que le scrolling vertical peut être utilisé jusqu'à une hauteur de 4 à 5 pages au maximum.

Le scrolling vertical est parfaitement accepté par les utilisateurs, d'autant plus que les souris avec molette de scrolling se généralisent.

- Quel lien pour accéder au détail ?

Où se trouve le lien permettant d'accéder à la page détail, et quelle forme prend-il ?

La logique du web veut que le lien hypertexte corresponde au texte qui identifie l'item. Qui identifie non pas sur le plan technique, mais *aux yeux de l'utilisateur*. Ainsi s'il s'agit de clients, il conviendra d'interroger les utilisateurs : utilisent-ils plutôt le *code client* ou la *raison sociale du client* ? Et il conviendra de mettre le lien où le regard se porte.

Toutefois, ce champ identifiant n'est pas toujours à la même position, et n'a pas toujours la même forme. Afin de créer une permanence dans les pratiques, il est bon d'ajouter une icône 'détail', qui – elle – sera toujours la même et toujours à la même place.

	Client	Commande
	[133] PariCable	[ANOMALI] SUIVI ANOMALIES PROD.
	[133] PariCable	[ETUDRAKA] ETUDE GESTION CCIALE
	[133] PariCable	[FORACCE] FORMATION ACCESS
	[133] PariCable	[GESCOML1] GESTION CCIALE LOT1
	[133] PariCable	[SCHDIR] SCHEMA DIRECTEUR

Sur cet exemple, l'icône signifiant 'accès au détail' sera disponible sur toutes les listes de l'application, et toujours dans la colonne de gauche.

Sur les listes présentées en tableaux, on créera une alternance de couleurs de fonds sur les lignes, formant une sorte de 'pyjama' qui permet à l'œil de mieux séparer les items, et aussi de mieux suivre les lignes de la gauche à la droite de la page.

C'est un procédé simple qui contribue au confort d'utilisation, et qu'il faut généraliser.

On voit aussi sur cet exemple l'une des différences citées en introduction : sur un site web cette icône donnant accès au détail serait à déconseiller car

sa signification n'est pas suffisamment claire et universelle. On aurait préféré un lien explicite : [détail]. Sur une application web d'usage régulier, l'utilisateur acquiert des réflexes et dès sa première utilisation, il aura intégré le sens de cette icône.

Dans une liste, chaque colonne doit être cadrée à gauche, des champs centrés dans une colonne sont difficiles à lire car l'œil doit zigzaguer pour trouver le point de départ de chaque ligne.

Tableaux et paragraphes

Le tableau n'est pas toujours la meilleure manière de présenter une liste d'entités.

Le tableau, avec une colonne pour chaque champ, présente des inconvénients importants, et trop souvent les concepteurs n'envisagent pas d'alternative.

Ces inconvénients sont les suivants :

- Dans un tableau, si un champ ne tient pas dans la largeur de sa cellule, il provoque un retour à la ligne qui est propagé à l'ensemble de la ligne. On n'ose donc pas placer des champs longs dans un tableau, car ils imposeraient des lignes d'une grande hauteur, où la plupart des colonnes seraient peu utilisées.
- Le tableau est limité en largeur : comme on s'interdit le scrolling horizontal, la quantité d'information affichée pour chaque ligne est très réduite.

Il existe une alternative au mode tableau, trop peu utilisée selon nous : c'est le mode paragraphe.

Considérons la liste ci-dessous, tout d'abord en mode tableau, puis en mode paragraphe :

numéro	libellé	rédacteur	type
[89.3172]	Qques anomalies de recette WebX V2	Patrice BERTRAND	anomalie
[89.3155]	Liste des projets - Tri	Valéry BONNET	question
[89.3089]	Affichage de hieroglyphes	Alexis HAUMONT	anomalie
[89.3075]	Forum associé à une demande: ne marche pas	Patrice BERTRAND	anomalie

[89.3172] Qques anomalies de recette WebX V2 Rédacteur: Patrice BERTRAND
 Type: anomalie, Gravité: non bloquant, Priorité: Moyenne
 Domaine: Aucun, Etat: A traiter
 Rédigée le: 07/02/2003
 La V2 inclut: tableau de bord et PKI. Sur le tableau de bord: - Toute la partie [...]

[89.3155] Liste des projets - Tri Rédacteur: Valéry BONNET
 Type: question, Gravité: non bloquant, Priorité: Basse
 Domaine: Aucun, Etat: A traiter
 Rédigée le: 05/02/2003
 Pas très pratique : sur la liste des projets, mes projets "courants" sont vers le [...]
 J'aimerais pouvoir gérer l'ordre d'affichage de mes projets. Une idée : ajouter une [...]

Le mode paragraphe permet de présenter plus d'information au regard de chacun des items, sans être contraint par les limites de colonnes. Dans l'exemple cité, on a choisi de fournir le début du texte décrivant chaque anomalie.

Dans beaucoup de cas, le mode paragraphe impose de répéter les libellés de champs, puisque la signification de chaque champ ne peut se déduire simplement de son entête de colonne.

Mais à l'inverse le mode paragraphe a des avantages intéressants. Chaque champ n'étant pas borné par les limites d'une colonne, on dispose d'une grande souplesse dans l'agencement. Il est possible en particulier de présenter plusieurs lignes de texte relatives à un élément donné.

Considérons la page suivante, extraite du site amazon.com :

All results for: usability

Search: for

All 77 results for usability :

Sort by:

31.  **The Wireless Web Usability Handbook**
 by Mark Pearrow (Paperback)
 Avg. Customer Rating: ★★★★★
 Usually ships in 24 hours
 List Price: \$49.95 Used & new from \$24.25
[Buy new: \\$49.95](#)

32.  **Usability: Turning Technologies into Tools**
 by Paul S. Adler (Editor), Terry A. Winograd (Editor) (Hardcover - June 1992)
 Usually ships in 24 hours
 List Price: \$95.00 Used & new from \$33.00
[Buy new: \\$95.00](#)

33.  **Designing Websites to Maximize Press Relations: Guidelines from Usability Studies with Journalists [DOWNLOAD: PDF]**
 by Nielsen Norman Group (Author) (Digital)
 Available for download now
 List Price: \$250.00
[Buy new: \\$250.00](#)

On constate les procédés suivants :

- Amazon a fait le choix d'une liste en mode 'paragraphe' et non en mode tableau. Il n'y a pas de colonage, en revanche chaque item est formaté selon un canevas identique, de telle sorte que le regard trouve ses marques facilement. Le mode paragraphe permet par exemple de faire cohabiter des titres courts et des titres longs sans perturber l'agencement de la page.
- La liste utilise beaucoup les couleurs et fontes pour créer des repères dans chaque paragraphe : prix en rouge, délai de livraison en petite fonte, etc. Notons également qu'ici le lien de l'item 32 a déjà été visité, il est en marron.
- Les liens, pourtant standards, sont proposés en mode texte. Amazon n'a pas cherché à faire une icône 'buy new' et une autre 'used & new'. Quelle que soit la qualité de ces icônes, elles n'auraient pas eu la clarté du texte.

Les petites icônes

Sur un site web, les icônes correspondant à des actions sont à utiliser avec modération, car leur sens n'est pratiquement jamais universel : beaucoup d'utilisateurs ne comprendront pas.

Sur une application web, la question est différente, car l'utilisateur est un habitué. S'il n'a pas compris la première fois, il apprendra du moins rapidement le sens de l'icône.

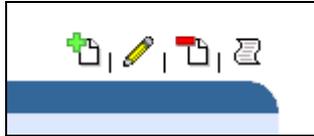
Une fois que l'utilisateur en a appris et intégré le sens, on sait que son regard localise et interprète une icône plus rapidement qu'un texte au sein de la page.

L'usage de petites icônes présente d'autres intérêts : elles occupent une place limitée, en particulier sur un tableau, elles ont un poids très faible, et si elles sont répétées dans la page, ne sont chargées qu'une seule fois.

Une icône dont on recherche la signification fait perdre du temps au contraire. Il est donc impératif d'utiliser les icônes (a) en nombre limité – moins d'une dizaine, et (b) de manière totalement cohérente au sein de l'application, et si possible des applications de l'entreprise.

On veillera aussi à bien utiliser le tag 'ALT' qui permet d'associer un texte à une image, ce texte apparaissant lors du survol de l'image. Ce peut être un moyen efficace de mettre à disposition une information complémentaire, immédiatement accessible.

A titre d'exemple, les liens-icônes suivants



signifient respectivement : créer, modifier, supprimer, voir la liste.

Attention, la notion de *barre d'outils* telle qu'on la retrouve dans certaines applications Windows, est totalement incompatible avec l'ergonomie web. Il ne faut pas confondre les icônes qui représentent des liens hypertexte, avec des *outils* à la mode Windows, que l'on viendrait *sélectionner en vue d'une action future*.

Liens

**Tous les liens hypertextes doivent être soulignés.
Réciproquement, rien ne doit être souligné qui ne soit pas
un lien.**

C'est une règle simple, mais qui est malheureusement souvent mal respectée. Les feuilles de style Html permettent de jouer avec les liens au lieu de les souligner bêtement, et certains s'amuse à les faire grossir au passage de la souris.

**Tout ce qui oblige à tâtonner avec la souris pour découvrir
les liens fait perdre du temps à l'utilisateur. Cliquer sur un
lien étant l'action élémentaire de l'ergonomie web, il ne
doit y avoir aucune ambiguïté sur la localisation des liens.**

En outre, les liens déjà visités changent de couleur, passant par exemple du bleu foncé au violet. Ce comportement standard doit être préservé.

Pour l'utilisateur, c'est une information qui peut être importante. Si par exemple il est en train de parcourir une liste en accédant au détail de chaque item, comme on peut le faire dans le résultat d'une recherche Internet, il verra clairement la partie de la liste qu'il a visitée.

Aide à la saisie

L'aide à la saisie est un élément fondamental dans l'ergonomie des applications web. Lorsque l'utilisateur doit saisir une valeur *qui sera contrôlée par rapport à des données de référence*, alors il est impératif de l'aider dans cette saisie, en lui permettant de choisir la valeur du champ parmi les valeurs possibles.

La liste déroulante est souvent l'outil à tout faire de l'aide à la saisie. Mais comme on l'a vu, la liste déroulante présente de nombreuses limitations.

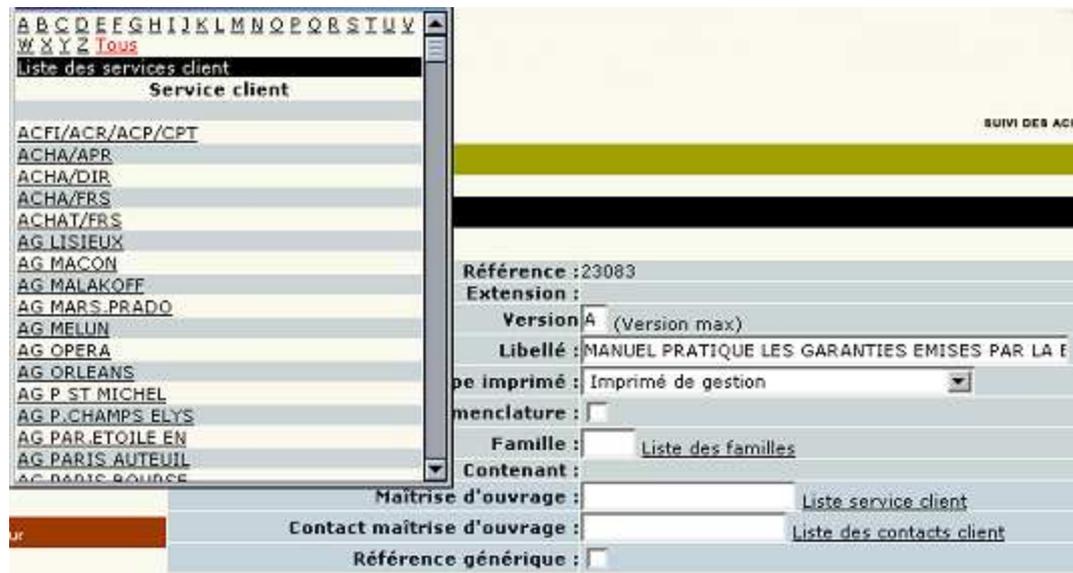
Le mécanisme d'aide à la saisie doit répondre à d'autres contraintes.

Il doit offrir de vraie fonction de recherche, qui sont nécessaires lorsque les valeurs possibles sont trop nombreuses.

Il doit autoriser néanmoins une saisie directe au clavier, qui pour un utilisateur chevronné sera plus rapide qu'une sélection.

Notre recommandation est de généraliser les fenêtres 'popup' d'aide à la saisie. Cette technique consiste à ouvrir une fenêtre indépendante qui présente la liste des valeurs possibles. Lorsque l'utilisateur clique sur l'une de ces valeurs, la fenêtre se referme, et la valeur est reportée dans la zone de saisie.

Regardons les exemples suivants :



Ici, l'utilisateur a cliqué sur le lien « liste service client » pour obtenir l'aide à la saisie, ce qui fait apparaître la fenêtre popup présentée.

Dans cette fenêtre, l'utilisateur peut accéder aux différents services par pages, ou cliquer sur la première lettre du service. Lorsqu'il clique sur un service, la fenêtre popup se referme, et l'identifiant est reporté dans la zone de saisie.

Lorsque l'utilisateur utilise l'aide à la saisie pour le champ suivant, « liste des contacts client », la boîte d'aide est filtrée sur les contacts *du service client* sélectionné.

Cette technique présente de nombreux avantages :

- Elle autorise la saisie directe d'une valeur, de sorte que l'utilisation de l'aide à la saisie n'est pas imposée – c'est un point important dans l'efficacité du travail.
- Elle permet de gérer si nécessaire une vraie interface de recherche dans la fenêtre d'aide : critères de recherche, liste paginée, etc.
- Elle permet de gérer des listes *dépendantes*, c'est à dire que la liste de valeurs proposée pour un champ pourra dépendre de la valeur déjà saisie pour un champ précédent. Comme on l'a vu c'est un cas de figure très courant que les listes déroulantes ne gère pas de manière satisfaisante.

Nous avons dit qu'il fallait éviter d'ouvrir plusieurs fenêtres. Les fenêtres d'aide à la saisie constituent la seule exception acceptable, aux conditions suivantes :

- La fenêtre popup doit être de taille réduite, de sorte que l'on voie bien la fenêtre principale sous-jacente
- La fenêtre popup ne concerne qu'une étape de dialogue, il n'est pas possible d'entamer dans cette fenêtre un nouveau dialogue.
- La fenêtre popup ne peut appeler elle-même une nouvelle fenêtre popup, de sorte qu'il n'y a jamais plus de deux fenêtres ouvertes.
- Et finalement la fenêtre popup doit si possible se fermer automatiquement lorsqu'elle cesse d'être active, cela évitera d'oublier une fenêtre enterrée.

A la place de l'utilisateur régulier

Bien souvent le concepteur, comme le développeur, ont du mal à réellement se mettre à la place de l'utilisateur.

Dans certains cas, ils sous-estimeront la pénibilité de telle tâche, car un clic de plus ou de moins, en phase de test unitaire, c'est réellement insensible. Mais un clic de plus, 100 fois par jour, lorsqu'il pouvait être évité, c'est dommage.

A l'inverse, le concepteur peut aussi sous-estimer la capacité de l'utilisateur. Supposons qu'il faille choisir un établissement parmi 40 en France. Le concepteur n' imagine pas une seconde que l'on puisse retenir 40 codes d'établissements. Il prévoit donc une magnifique liste déroulante dont la manipulation demande : de quitter les mains du clavier pour chercher la souris, viser et cliquer à droite de la liste, cliquer une ou deux fois pour descendre la liste, finalement choisir l'établissement, et ramener les mains sur le clavier. Ou bien, il est vrai, manipuler la liste déroulante au moyen du seul clavier. Mais dans la pratique, un utilisateur régulier aura rapidement

mémorisé les 40 codes d'établissements, comme le personnel d'Air France connaît les codes des aéroports et des compagnies. Dès lors, ce que *souhaite* l'utilisateur, c'est tout simplement d'entrer le code au clavier.

Répétons le : l'aide à la saisie doit impérativement être optionnelle, l'utilisateur régulier pouvant travailler plus efficacement en mémorisant les valeurs de référence.

Contrôle de saisie

Faut-il contrôler les saisies « coté client » ou « coté serveur » ?

C'est sans doute une des questions les plus souvent posées pour les interfaces web.

Contrôler les saisies coté client, c'est à dire par du javascript inséré dans la page Html permet une meilleure réactivité de l'interface. L'utilisateur est averti plus tôt de son erreur, on ne le laisse pas s'enfermer, il peut rapidement corriger sa saisie et poursuivre son action.

En revanche, plusieurs arguments militent contre cette solution.

Le code javascript qui implémente ce contrôle de saisie est partie intégrante de l'application. Cela signifie que l'application est écrite en deux langages, éclatée en deux environnements distincts : l'un coté client, l'autre coté serveur. Sans entrer dans les détails, on perçoit bien que cette dualité posera des problèmes, en maintenance, en gestion des compétences de l'équipe, mais aussi en cohérence.

Ainsi, supposons un champ de saisie qui attend un nombre entier et que du code javascript le vérifie. Imaginons qu'un jour un changement fonctionnel autorise un nombre décimal. Il est clair que des changements seront à opérer à la fois coté serveur et coté client, et le risque est grand que des incohérences soient introduites par de tels changements.

Par ailleurs, le code javascript est plus ou moins mêlé dans une page html dont la fonction principale est de décrire de la présentation et de la mise en forme. C'est un mélange que toutes les démarches de développement cherchent à éviter. Si un graphiste est amené à retoucher cette page, il devra soigneusement éviter de toucher à ce code javascript qui n'est pas de son ressort.

Au delà des considérations portant sur le processus de développement et de maintenance, il faut souligner que les contrôles javascript, quoi qu'il en soit, ne peuvent être complets. Ces fonctions de contrôles n'ont pas à leur disposition d'autres données que celles de la page, et ne peuvent donc mettre en œuvre des contrôles que de premier niveau.

Ainsi dans tous les cas, les contrôles de saisie doivent être refaits et étendus, coté serveur. Nous disons 'refaits' car d'une manière générale l'application web ne doit pas faire confiance aveuglément aux requêtes qu'elle reçoit du poste client.

Conception d'applications web : efficacité et utilisabilité

Ils doivent être étendus aussi pour inclure des contrôles de cohérence plus larges. Par exemple le fait qu'un champ 'client' contient bien un code correspondant à un client existant.

Les erreurs de saisie peuvent être rangées schématiquement de la manière suivante :

- Contrôle syntaxique unitaire : longueur du texte, nombre, date, etc... et champs obligatoires. Ils peuvent être opérés en javascript coté client ;
- Contrôles de conformité par rapport à des données de référence. Ils sont opérés coté serveur. Certains ne portent que sur un champ, d'autres sur la cohérence entre plusieurs saisies : le pays saisi dans le champ 2 doit être dans la liste des pays de la région indiquée dans le champ 1.
- Autres contrôles, plus spécifiques. Dans la pratique, ils constituent une très faible proportion.

De plus, dans la pratique la bonne application de règles de gestion métier doit parfois être vérifiée *dans un ordre donné*, dans la mesure où telle règle pour être applicable, suppose qu'une autre règle est déjà appliquée.

Comment vérifier que le pays indiqué est bien dans la région choisie si déjà la région est incorrecte ?

**Au final, notre préconisation est la suivante :
n'implémenter coté client que des contrôles qui peuvent être standardisés, avec un petit nombre de cas : champ obligatoire, nombre, mini/maxi, entier ou décimal, date, email, téléphone.**

Du point de vue du développement, ces contrôles ne seront jamais codés en spécifique, au cas par cas, mais feront appel à des bibliothèques existantes. Dans certains cas, on pourra même automatiser le lien entre ces contrôles et les objets métier coté serveur, qui eux-mêmes pourront être liés à la base de données.

Bien entendu, les contrôles qui relèvent du Html natif doivent être utilisés impérativement. Il est insupportable qu'un champ de saisie laisse saisir 25 caractères pour s'entendre dire ensuite qu'il n'en faut pas plus de 20.

En outre, l'usage suivant est généralisé :

Les libellés feront toujours apparaître les champs obligatoires, au moyen d'une astérisque.

Et lorsque le format de saisie peut avoir des variantes, on indiquera dans le libellé le format attendu.

Notification des erreurs

La question subsidiaire de la précédente : de quelle manière notifier les erreurs de saisie à l'utilisateur. On pourrait trouver idéal de notifier en une fois la totalité des erreurs de saisie rencontrées, en faisant apparaître à côté de chaque champ de saisie le message correspondant à l'erreur portant sur ce champ. Sur des sites grand public, nous avons ainsi construit des dispositifs qui non seulement notifient toutes les erreurs sur le même formulaire, mais positionne chaque message d'erreur en rouge à proximité du champ de saisie concerné.

Pour des applications web, ce procédé n'est pourtant pas toujours le meilleur.

Avant tout, il faut se souvenir que dans une utilisation quotidienne d'une application, les erreurs de saisie ne sont pas nombreuses.

L'utilisateur a déjà rempli plusieurs fois le même formulaire ; il peut encore se tromper, mais ce sera de plus en plus rare. Il n'est donc pas pertinent de faire des efforts pour optimiser le traitement des erreurs multiples.

En conclusion, notre recommandation pour l'applicatif web est de contrôler les règles de gestion côté serveur les unes après les autres, et de signaler la première erreur rencontrée.

Listes liées

Encore un grand classique de l'ergonomie web. L'utilisateur est invité à choisir un pays, puis une ville. Le pays est sélectionné dans une liste, et de même la ville est sélectionnée parmi les villes de ce pays.

Il y a 4 manières de gérer cela.

La première consiste à repeupler la seconde liste par du code javascript chaque fois qu'une sélection est faite dans la première liste. Cette solution est peut-être la plus confortable pour l'utilisateur qui retrouve un schéma habituel de l'ergonomie Windows. D'un point de vue technique, cette méthode oblige à gérer un extrait de la base de données, toute la relation pays-villes, dans du code intégré à la page Html. Si ces tables sont volumineuses, cela alourdira sensiblement la page, et l'efficacité attendue dans l'interface ne sera pas au rendez-vous.

La seconde consiste à utiliser deux pages distinctes : la première pour sélectionner le pays, la seconde pour sélectionner la ville. C'est somme toutes l'application directe du principe de linéarité du dialogue. Cette solution, finalement la plus simple, est trop souvent écartée, non pour des raisons d'efficacité, mais parce que le concepteur a en tête des schémas d'ergonomie hérités de Windows. L'utilisateur, lui, ne sera pas du tout surpris, et

l'efficacité du travail pourra être tout à fait satisfaisante, dans la mesure où les pages sont légères, et servies rapidement.

La troisième voie consiste à réafficher la page après sélection du pays, en peuplant la liste des villes coté serveur. D'une certaine manière, c'est un mélange des deux premières : les deux listes sont sur une même page, mais on passe par le serveur pour obtenir les villes du pays choisi.

Enfin, la quatrième manière de traiter cette question consiste à utiliser, tant pour les pays que pour les villes, des fenêtres popup d'aide à la saisie. Dans ce cas, le filtrage de la liste des villes dans la seconde fenêtre d'aide est facile à réaliser.

Notre recommandation est d'utiliser de manière systématique le système d'aide à la saisie en fenêtres 'popup', qui permet de gérer les listes liées.

Boutons et liens

Boutons et liens sont deux choses différentes, et sont trop souvent utilisés sans distinction.

Le lien est bien sûr l'élément de base de toute l'ergonomie web. Il permet de passer à une nouvelle page, et le libellé du lien annonce une promesse quant à la page vers laquelle il nous conduira.

Le bouton n'existe que par rapport à des formulaires. Et d'ailleurs le Html ne permet pas de faire apparaître un bouton standard en dehors d'un formulaire. Bien sûr cette impossibilité peut être contournée de beaucoup de façon, l'une d'elles étant de créer une image gif figurant un bouton.

Mais c'est une erreur, qui n'aidera certainement pas l'utilisateur.

Il ne faut pas utiliser de boutons, ni même la symbolique du bouton, pour conduire à une nouvelle page car c'est la fonction des liens.

Les boutons provoquent des actions en relation avec des formulaires : valider la saisie, remettre les champs à leur valeur initiale. Et réciproquement bien sûr, on évitera d'utiliser des liens hypertexte pour ces actions.

Divers

- Ne jamais présenter des données en mode saisie sans que cela ait été explicitement demandé. Pour un utilisateur qui ne veut que consulter, le mode saisie est perturbant, l'utilisateur craint de modifier un champ par erreur, il ne sait pas s'il doit abandonner la saisie en quittant la page, ou bien l'annuler explicitement, etc.
- Confirmer toutes les actions. Si une page comporte en face d'un item un lien intitulé 'ajouter à mes favoris', il offrir à l'utilisateur

une forme d'acquiescement de son action. Soit l'on reste en pleine page, et un message confirme l'ajout, soit une fenêtre en popup apporte cette confirmation.

OBJETS D'INTERFACE

Libellés et champs

Une règle simple est que :

L'utilisateur doit séparer sans difficulté les libellés d'interface et les valeurs des champs.

Cette distinction peut se faire de différentes manières : taille de fonte, couleur de fond, etc. mais elle doit bien sûr être parfaitement cohérente au travers de l'application.

Ainsi, sur l'exemple suivant, la distinction est clairement donnée, à la fois par la position (une colonne pour les libellés, une colonne pour les valeurs) et par la couleur de fond.

Detail of a generic element	
	Edit Add Delete Tasks
Code	G1064
Name	ADHESIVES
Type	Product
Category	Non metallic material
Design review	1 : Type approval
Frequency	rare
Applicable rules	Class

On notera également sur cet exemple que les libellés sont justifiés à droite et les valeurs à gauche, de manière à ce que l'œil ait le moins de distance possible à parcourir pour aller de l'un à l'autre.

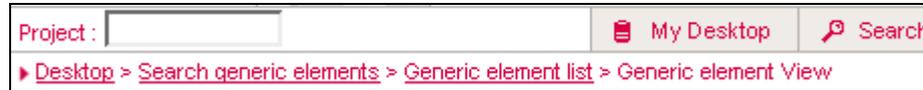
Barre de progression

La technique de la 'barre de progression' doit être généralisée dans toutes les applications.

Cette technique, qui fait apparaître la hiérarchie des étapes conduisant à la page courante, se généralise sur les sites Internet. Elle présente deux intérêts importants :

- Elle permet à l'utilisateur de bien se situer dans le réseau de page de l'application ;

- Elle permet en un click de revenir à n'importe quelle étape antérieure.



Une recommandation complémentaire serait d'insérer quelques informations de contexte dans cette liste de pages.

Au lieu de mettre :

Bureau > Recherche client > Liste client > Détail client > Factures

On apporte plus d'information en proposant :

Bureau > Recherche client > Liste clients 'Europe' > Détail client 'Peugeot' > Factures

Bien entendu, c'est un peu plus dur à mettre en œuvre.

Zones de texte

Les zones de texte doivent toujours être dimensionnées à la longueur maxi du champ.

A la fois pour ce qui est de la dimension visible que pour le nombre de caractères autorisés.

En effet, d'une part les utilisateurs sont très réticents à saisir du texte dans une zone alors que le début du texte a commencé à disparaître à gauche, et d'autre part il est inadmissible qu'une zone de texte laisse saisir 25 caractères pour qu'un contrôle nous indique ensuite qu'il n'en fallait pas plus de 20 !

Listes déroulantes

La liste déroulante est trop souvent le 'couteau suisse' de la conception d'interface. Elle est à utiliser avec modération.

Les listes déroulantes conviennent pour une sélection portant sur entre 4 et 20 à 30 items. Il faut garder à l'esprit que l'action sur une liste déroulante requiert au minimum deux clics, et souvent trois voire plus. De plus l'utilisateur ne voit pas l'étendue des choix possibles, mais seulement la valeur sélectionnée. Les valeurs non-sélectionnées peuvent contribuer à la compréhension de la saisie.

Ainsi jusqu'à 4-5 items, des boutons radios pourront être une meilleure solution : ils sont actionnables en un seul clic, et donnent une meilleure visibilité sur les choix proposés.

Tout le monde a rencontré l'incontournable liste déroulante de *choix de pays*, comme par exemple ici sur le site de Avis :

Et tout le monde a pu constater qu'il fallait jusqu'à 6 clics de souris pour y sélectionner par exemple la France. Pour un site Internet, cela reste sans doute la meilleure solution. Mais pour une application web cette technique n'est pas assez efficace : l'utilisateur doit pouvoir saisir « France » au clavier s'il le souhaite.

Listes à choix multiples

Le Html permet de sélectionner plusieurs items dans une liste, en gardant la touche Ctrl appuyée. Mais rares sont les internautes qui connaissent cela, ce procédé est donc à éviter.

[Sélectionnez vos critères]

Vous pouvez sélectionner plusieurs critères dans un même champ en maintenant la touche CTRL enfoncée sur les ordinateurs de type PC (touche SHIFT sur Macintosh).

Lieu de travail

• France • International

Tous

ILE-DE-FRANCE

.....Paris

.....Essonne

.....Hauts-de-Seine

.....Seine-et-Marne

De plus, en faisant défiler la liste pour trouver de nouvelles valeurs, l'utilisateur perd de vue les valeurs déjà sélectionnées.

Notre recommandation est de ne pas utiliser les listes à choix multiples.

Cette technique est le plus souvent très bien remplacée par une liste de cases à cocher, que l'utilisateur comprendra immédiatement comme signifiant 'choix multiples possibles'.

Il existe une autre technique d'interface peu élégante, qui consiste à présenter deux listes, côte à côte, et des boutons permettant de basculer des items d'une liste à l'autre. Le but est ici aussi d'éviter à tout prix un aller-retour au serveur. Mais il vaut mieux un aller-retour serveur rapide, associé à une ergonomie simple et efficace.

Choix de date

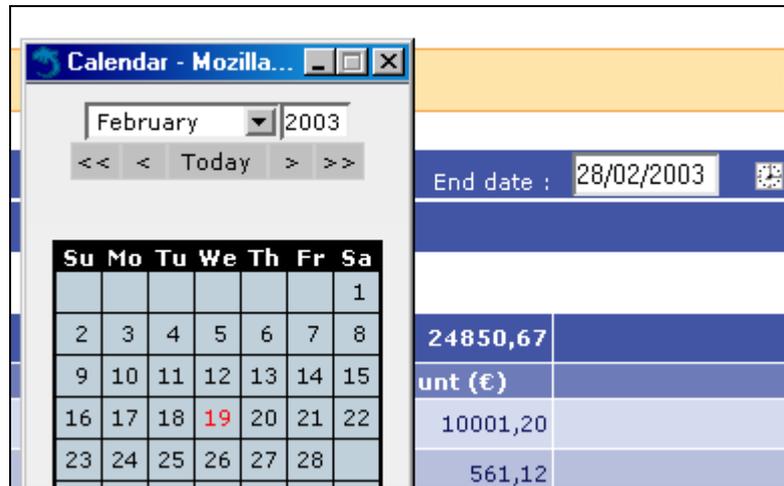
L'une des plus mauvaises interfaces pour saisir une date est celle qui présente trois listes déroulantes : l'une pour la date, l'autre pour le mois, la troisième pour l'année. Non seulement il faut une dizaine de clics pour entrer une date, mais en plus on peut construire ainsi des dates inexistantes, et enfin, l'utilisateur n'a pas la vision du calendrier (« le premier lundi d'avril »).

Le morceau de page suivant est tiré du site d'Air France :

Ici on a évité quand même d'avoir une troisième liste déroulante pour les années ! Pour un site grand public, cette interface est acceptable car relativement intuitive.

Pour une application web, il est impératif de laisser l'utilisateur saisir la date entièrement au clavier : « 20/02/2003 », car dans la plupart des cas ce sera le moyen le plus rapide.

Par contre il est fort utile de proposer aussi une aide à la saisie sous la forme d'un petit calendrier en fenêtre 'popup', qui aidera l'utilisateur à se repérer visuellement dans les jours, comme sur l'exemple suivant :



Les onglets

Les onglets ne sont finalement que des liens présentés dans un habillage graphique particulier, donnant l'apparence de pages. Ils sont inspirés de l'ergonomie Windows. Mais dans une interface graphique Windows, le changement d'onglet est pratiquement instantané, de sorte que l'utilisateur peut parcourir les différentes pages sans la moindre attente.

Il y a un petit risque donc à utiliser des onglets en ergonomie web, car le passage d'une page à l'autre ne sera pas aussi immédiat.

Malgré tout, les onglets sont intéressants car dans la représentation mentale de l'utilisateur, ils matérialisent bien différents groupes d'information, de manière plus palpable que ne le font de simples liens.

> [Inventory](#) > **ACTIVITY**

[Advanced search parameters] Start date : 01/02/2003 End date : [Search]

Activity Postal Product Connections

		Total :	25693,00	24850,67
Meter ID	Group	Items	Amount (€)	
HF15396	Marketing	10250	10001,20	
HF16309	Marketing	4523	561,12	

La structuration de la page en onglets n'est en fait pas très différente d'une structuration verticale, en groupes d'information entre lesquels l'utilisateur naviguerait par un défilement vertical.

Mais pour l'utilisateur, les groupes d'informations sont mieux distingués avec ces sous-pages distinctes.

La symbolique de l'onglet pour représenter différentes sous-pages est une bonne technique d'ergonomie, qui permet de structurer l'information tout en maintenant le principe de page.

En revanche, il faut proscrire l'utilisation d'onglets gérés localement en DHTML, c'est à dire que la même page Html contient le contenu de tous les onglets.

On évitera d'utiliser des onglets en saisie, car il peut y avoir confusion dans l'esprit de l'utilisateur quant à la prise en compte de ses saisies lors du changement d'onglet.

Entités de référence

Une autre question courante concerne la possibilité de créer une entité de référence alors qu'on est en saisie.

Considérons le cas suivant. L'utilisateur est invité à saisir une commande. La commande fait référence à un client, que l'utilisateur peut choisir dans la liste, disons soit en liste déroulante s'il y a peu de clients, soit à l'aide d'une fenêtre d'aide à la saisie. Que se passe-t-il si la commande porte sur un client qui n'existe pas encore ? Faut-il autoriser la création d'un client venant s'insérer au milieu de la saisie de la commande. On ouvrirait donc une fenêtre en popup pour la création d'un nouveau client, on validerait cette création, fermerait la popup, et l'on reviendrait poursuivre la saisie de la commande.

Notre recommandation est de ne pas autoriser ce schéma de navigation : on évitera d'ouvrir une parenthèse dans le processus de travail pour créer une nouvelle entité de référence.

Ceci pour les raisons suivantes :

- L'utilisateur va commencer à empiler des fenêtres à l'écran, mais également empiler des *contextes de travail dans sa tête*. Se souvenir de ces contextes au moment de les dépiler est difficile et demande une concentration importante.
- Lors de la création du client, la question va se poser à nouveau : l'entité client fait peut être référence à une entité 'domaine commercial', qui n'existe pas encore. Faut-il interrompre aussi la saisie du client pour créer un nouveau domaine commercial ? Jusqu'où permettra-t-on cet empilement ?

Ainsi nous pensons qu'il est finalement plus simple pour l'utilisateur d'annuler la saisie de commande qu'il avait commencée, de créer son nouveau client, et de retourner à sa commande. Cela nous permettra de respecter le principe énoncé plus haut, qui veut que l'on n'ait jamais plus de deux fenêtres ouvertes : une fenêtre principale et une fenêtre en popup.

Souris

C'est un détail, mais aux conséquences lourdes :

Lorsqu'une page comporte des champs de saisie, le 'curseur' doit être automatiquement positionné sur le premier champ.

Ce n'est malheureusement pas le comportement par défaut des navigateurs, et donc demandera un peu de javascript, très simple.

Mais c'est extrêmement important car amener la main du clavier à la souris, viser, cliquer, puis revenir au clavier, fait perdre 3 à 4 secondes, qui se multiplieront par 100 ou 1000 au cours d'une journée.

Une application web doit pouvoir être pilotée le plus possible sans lâcher les mains du clavier.

A contrario, sur des phases de dialogue de pure consultation, l'utilisateur n'utilisera *que* sa souris. Dans ces phases de dialogue, il faut au contraire éviter de lui imposer l'utilisation du clavier.

Pour une efficacité d'utilisation optimale, l'application doit alterner des phases de dialogue tout-souris et des phases de dialogue tout-clavier.

La transition souris / clavier casse le rythme de travail de l'utilisateur.

Raccourcis clavier

Les raccourcis clavier sont omniprésents dans l'ergonomie Windows, et peu usités dans l'ergonomie web.

Pourtant, on sait que le fait de déplacer la main du clavier vers la souris, puis à nouveau au clavier est extrêmement peu efficace et casse le rythme de travail. C'est pourquoi sur une interface Windows, les utilisateurs réguliers utilisent très peu la souris et mémorisent des raccourcis clavier qui rendent leur travail plus efficace.

L'usage des raccourcis n'est pas dans la norme Html : il faut un peu de 'bricolage' javascript pour offrir cette possibilité sur une page web, et il faudra un grand soin pour assurer la compatibilité entre configurations du poste client.

Pour cette raison, nous pourrions être tentés de l'interdire. Mais il nous semble que l'apport des raccourcis clavier peut être suffisamment déterminant pour que l'on accepte le poids du javascript.

Images et textes**Ne pas utiliser de texte en image.**

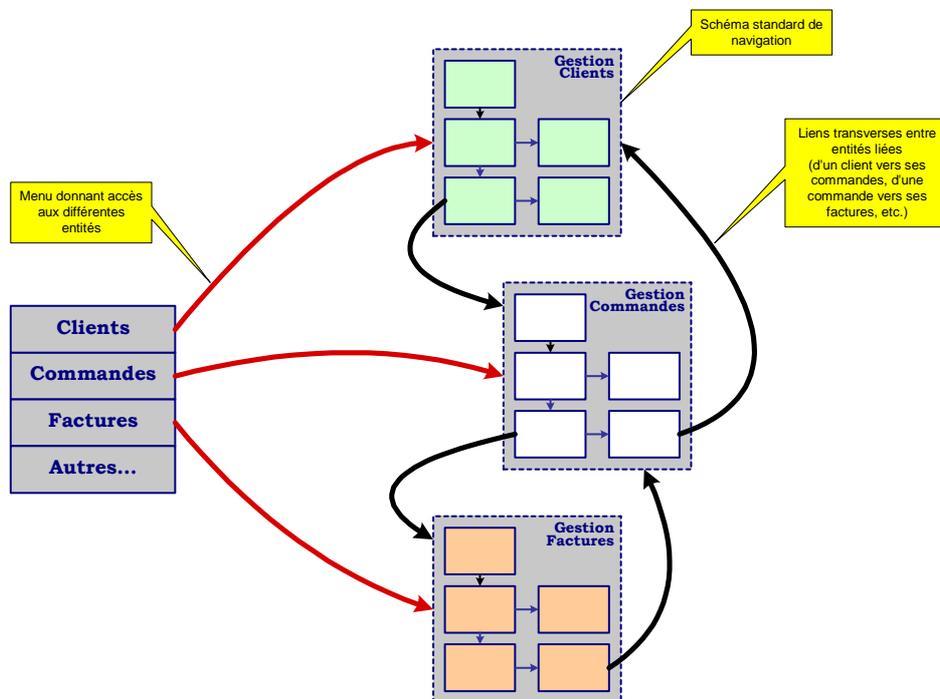
C'est à dire ne pas intégrer d'image Gif représentant des textes. La valeur ajoutée esthétique est généralement faible, et cela n' a pas de justification dans un applicatif web. Cela rendra l'application inutilisable pour les handicapés, rendra la traduction difficile et la page plus lourde.

**PORTAIL, PERSONNALISATION,
WORKFLOW**

Nous avons vu plus haut les enchaînements standards permettant la gestion homogène d'entités au sein de l'application.

Dans la pratique, beaucoup d'applications sont constituées d'un menu, donnant accès à différentes entités : gérer les clients, gérer les commandes, gérer les factures, etc.

www.smile.fr



Cette approche a l'avantage d'une grande homogénéité, et peut être appliquée, sans beaucoup de réflexion, à un grand nombre d'applications.

D'une certaine manière, c'est la manière dont l'informaticien visualise l'application : il gère des entités.

Mais l'utilisateur, lui, effectue des tâches qui entrent dans des processus de travail. Tel utilisateur du contrôle de gestion s'assure que les commandes ont été facturées. Tel autre, commercial, saisit des comptes-rendus de visites.

Une application web n'est pas un simple assemblage de fonctions de consultation et de mise à jour d'entités. Elle doit intégrer la dimension processus de travail.

Workflow

L'approche process est transverse, par rapport à l'approche entités.

Tel intervenant participe à tels process de travail, et par rapport à ces process, telles tâches sont attendues de lui.

Il n'est pas normal que cet intervenant ait à se demander s'il n'y a pas des choses à faire, puis à chercher les entités sur lesquelles son action est attendue : c'est l'application qui doit lui indiquer, dès sa page d'accueil, ce qui est attendu de lui.

Dans telle entreprise, le responsable qualité doit valider le processus de fabrication de tout nouveau produit, après que le chef de produit l'aura défini. Sur la page d'accueil, le responsable qualité ne trouve pas un bête menu : il trouve *son plan de travail*, la liste des nouveaux produits *de son domaine de compétence*, dont le processus de fabrication a été défini et qu'il n'a pas encore validés. Il clique sur le premier, consulte le détail du produit, et procède à la validation. Le produit disparaît bien sûr de sa liste de tâches.*

Pour qu'il opère sa validation, l'application web ne doit pas lui présenter un vaste formulaire de mise à jour, au sein duquel une case le concernerait. Ce serait encore un avatar d'une approche purement 'gestion d'entités'. L'application doit lui présenter la synthèse du produit *le concernant*, avec un lien intitulé '*valider le processus de fabrication*'.

Toute application comporte une part de workflow, pour autant qu'on y prête attention.

L'approche workflow facilite le travail de chacun, et structure l'organisation.

La conception d'une application doit toujours intégrer cette approche, c'est à dire poser la question des typologies d'utilisateurs, des processus de travail, et de la manière de les traiter de manière structurée dans l'application.

Les profils utilisateur sont souvent abordés par l'informaticien en termes de droits d'accès. Selon le profil l'application rend disponible ou non telles fonctions du menu.

Cette personnalisation *par les droits* est loin d'être suffisante : l'approche workflow implique une personnalisation *par les besoins*.

A noter bien sûr que l'e-mail est un complément précieux de l'application Http, dans la mise en œuvre de workflows, particulièrement pour des utilisateurs dont l'intervention n'est pas sollicitée de manière quotidienne : il est alors intéressant que l'application puisse prévenir l'utilisateur de manière asynchrone.

Portail et personnalisation

Tout le monde connaît le site *My Yahoo* (<http://my.yahoo.com/>). Il permet de configurer sa propre page d'accueil d'une part en choisissant des blocs, des pavés, de contenus, en les agençant sur la page, et enfin en les *configurant*. Ainsi on pourra choisir le pavé *Météo*, le placer dans la colonne de droite de la page, et finalement choisir les régions dont la météo nous intéresse.

L'idée bien sûr est de réunir sur la page d'accueil l'essentiel de l'information qui nous intéresse, et donc de la rendre disponible en zéro clics.

La même idée peut s'appliquer à une application web.

Chaque utilisateur n'a usage que d'une petite partie des données disponibles, et l'application dispose souvent de l'information nécessaire pour savoir quelles entités sont utiles à quel utilisateur.

Prenons l'exemple, très banal, de la gestion des *clients*. Imaginons que l'entreprise possède de l'ordre de 400 clients identifiés. Comme on l'a vu plus haut, l'application peut proposer les outils de *recherche-filtre clients*, *liste résultat*, *détail client*, *etc.*, selon un schéma standard de navigation.

Mais dans la pratique, chaque utilisateur n'a réellement besoin d'accéder qu'à un petit nombre de clients, et il n'a que faire d'une fonction de recherche multi-critères.

Ce n'est pas l'utilisateur qui doit dire à l'application quels clients il souhaite consulter, c'est l'application qui doit lui présenter les clients qu'ils souhaite probablement consulter.

Reste à savoir comment l'application déterminera les clients que l'utilisateur peut souhaiter consulter. Il y a pour cela plusieurs approches :

- La première est de définir de manière explicite la relation qui lie un utilisateur à certains clients, dans la modélisation des données. C'est le cas par exemple lorsque un commercial se voit affecter un périmètre de clients : cette relation est définie dans la base de données, et il est alors simple de présenter à chaque commercial ses propres clients.
- La seconde est d'utiliser les informations glanées au travers de l'utilisation du site pour créer une relation implicite entre un utilisateur et certains clients. Ce peut être par exemple les clients *pour lesquels il a travaillé dans les derniers mois*, ou bien *qu'il a consultés le plus récemment*, etc. De même que Amazon présente automatiquement à chacun les livres susceptibles de l'intéresser compte tenu de ses commandes précédentes, l'application présente à chacun les clients susceptibles de l'intéresser. Bien sûr cela signifie que l'application doit enregistrer ces

informations liées à l'utilisation du site, afin d'en faire un usage de *profiling* .

- La troisième enfin est de demander à chaque utilisateur de créer sa propre personnalisation, c'est à dire de sélectionner lui-même les clients avec lesquels il travaille et qu'il souhaite pouvoir consulter rapidement.

La première approche est appliquée assez naturellement : à partir du moment où une relation explicite a été prévue, il est naturel que l'application en fasse usage. Et cela particulièrement si des *restrictions d'accès* ont été définies, par exemple si l'on a défini que chaque commercial ne pouvait accéder *que* à ses propres clients.

Les deux autres sont moins utilisées, et c'est regrettable. Profiling et personnalisation sont des techniques extrêmement puissantes au service des applications.

Trop d'applications ont une page d'accueil vide, réduite à un simple menu, et meublée le cas échéant d'un visuel inutile.

La page d'accueil doit être une page intelligente, qui anticipe sur les besoins de l'utilisateur.

C'est la fin du mois, on propose aux utilisateurs de profil 'développeur' de remplir leur rapport d'activité.

Les statistiques trimestrielles sont disponibles, on les présente à l'utilisateur de profil 'contrôleur de gestion' dès la page d'accueil.

Des achats sont à valider par le chef de service : on lui en présente la liste dès sa page d'accueil.

Tel commercial a rendu visite à 8 clients dans le dernier mois, on lui présente ces clients-ci sur sa page d'accueil, afin qu'il puisse consulter leur fiche rapidement.

Cinq nouvelles candidatures ont été reçues et n'ont pas été qualifiées, elles sont présentées dès la page d'accueil du responsable RH.

Moteur de recherche

Il est courant que les concepteurs d'un site web ne perçoivent pas toute l'importance d'un moteur de recherche, pensant parfois que le contenu du site n'est pas suffisamment vaste que l'on ne puisse pas y retrouver ce que l'on cherche par simple navigation.

Un moteur de recherche est le moyen pour le visiteur de ne pas réfléchir, de ne pas chercher à comprendre l'organisation du site. Il saisit les mots qui lui viennent à l'esprit, valide, et regarde ce qu'on lui propose.

De ce point de vue, l'intérêt du moteur de recherche sur un site n'est pas lié au volume de contenu.

Dans le contexte d'une application web également, fournir le moyen de ne pas réfléchir est toujours apprécié ! Il n'est pas courant malheureusement qu'une application web offre un dispositif de moteur de recherche standard.

L'utilisateur s'intéresse au tableau de bord qualité de l'établissement de Strasbourg. Il saisit « *qualité strasbourg* », et valide. Il ne cherche pas dans les menus, ne navigue pas de page en page. L'application lui propose différentes réponses, dont probablement celle qu'il recherche.

L'intégration d'un moteur d'indexation et de recherche dans le contexte d'une application web offre une voie d'accès directe à l'information, qui peut améliorer sensiblement l'utilisabilité.

Bien sûr, la mise en œuvre du moteur d'indexation dans ce contexte demande quelques dispositions particulières. Il faut en particulier que le robot d'indexation puisse parcourir toutes les pages, c'est à dire qu'il existe un chemin conduisant à toutes les pages à indexer, sans traverser de formulaires.

Avec un peu d'attention, cette disposition n'est pas très contraignante.

Il faudra veiller également à contrôler les droits d'accès de l'utilisateur au moment de constituer la page de réponse.

Malgré ces quelques précautions, il nous semble que le moteur de recherche est un formidable complément sur une application web. Il correspond en outre à une démarche qui devient un réflexe pour les surfeurs chevronnés.

CONCLUSION

Nous avons ouvert le débat en insistant sur ce qui distingue les applications web des sites web, en matière d'utilisabilité.

Il s'avère en fait que la convergence est grande.

D'une part les sites web haut de gamme se sont éloignés des pratiques racoleuses, animations fatigantes, pages trop lourdes, pour privilégier, comme les applications web, l'efficacité, la simplicité, la rapidité.

Réciproquement, nous avons essayé de montrer comment les applications web pouvaient encore s'inspirer largement des bonnes pratiques des sites : portail, personnalisation, profiling, moteur de recherche, etc.

Nous espérons que vous trouverez ces quelques recettes pertinentes, et qu'elles aideront vos projets.

Si vos réflexions vous amènent à les compléter, alors n'hésitez pas à nous faire part de vos propres bonnes pratiques. Avec votre permission, nous pourrions enrichir cet ouvrage.