

A background image showing a business meeting. In the foreground, a person's hands are holding a tablet. In the background, another person is holding a smartphone. A semi-transparent grid is overlaid on the entire image. The text is centered in the middle of the image.

Stratégie et architecture des API : une approche coordonnée

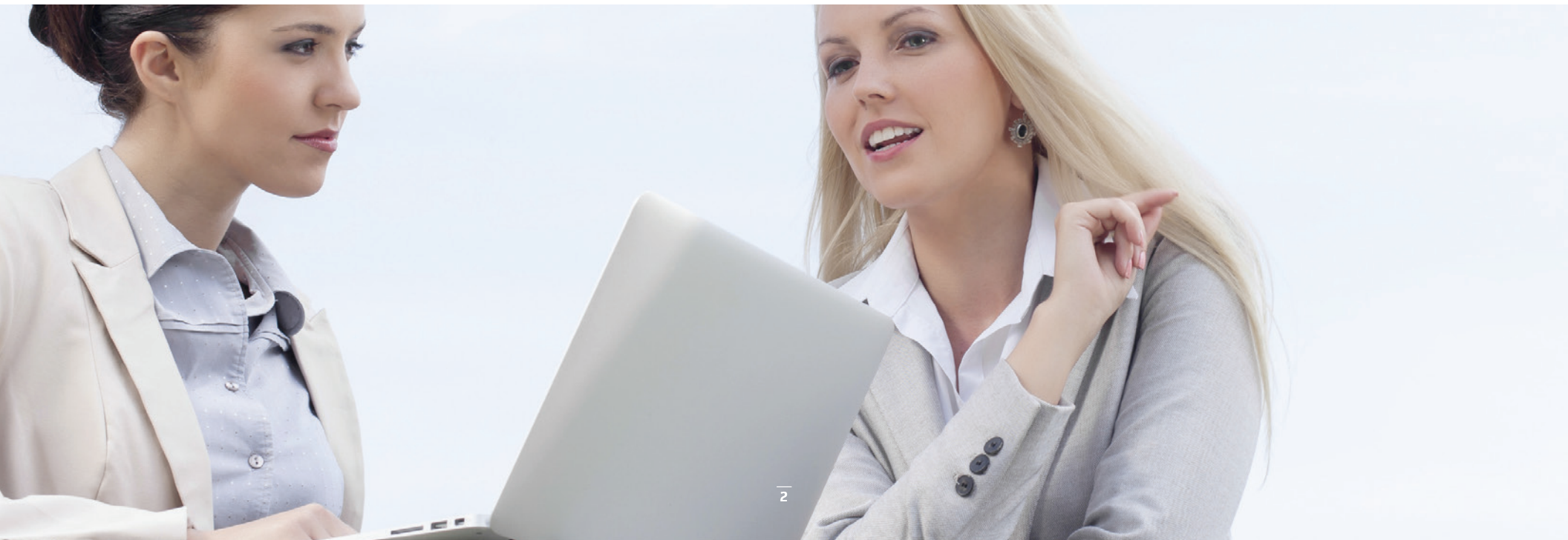
Introduction

L'essor des interfaces de programmation d'applications (API) représente à la fois une opportunité commerciale et un défi technique. Pour les dirigeants d'entreprise, les API sont synonymes de nouvelles sources de revenu et d'optimisation de la valeur pour le client. Ce sont toutefois les architectes d'entreprise qui sont chargés de créer ces API permettant de réutiliser les systèmes back-end dans les nouvelles applications Web et mobiles.

Il est crucial que tous les intervenants comprennent que les objectifs métier et les défis techniques d'un programme d'API sont intimement liés. Les responsables de programmes ont pour rôle de présenter clairement les objectifs commerciaux clés d'une API proposée aux architectes qui seront en charge de la conception de l'interface.

Les architectes, à leur tour, doivent s'assurer de ne pas perdre de vue ces objectifs pendant tout le processus de déploiement de l'infrastructure de l'API et de conception de l'interface. Toutes les décisions techniques doivent contribuer à créer une interface permettant aux développeurs de concevoir des applications clientes réellement indispensables pour les utilisateurs finaux.

Cet eBook présente les meilleures pratiques en matière de conception d'API centrées sur les résultats, véritable pierre angulaire du succès de votre programme d'API.








1re partie : de l'architecture SOA à l'interface API

L'informatique d'entreprise au 21^e siècle se caractérise par une ouverture des bases de données et applications traditionnelles en silos, permettant d'accéder aux données et fonctionnalités au-delà des frontières organisationnelles et de les réutiliser dans de nouveaux systèmes. Cette tendance s'est d'abord manifestée par l'architecture orientée services (SOA) et, plus récemment, par l'explosion des API orientées Web.

À un certain niveau, les « services Web » centraux à l'architecture SOA sont équivalents aux API Web : il s'agit dans les deux cas d'interfaces utilisées pour ouvrir les systèmes back-end. Il existe toutefois des différences fondamentales entre ces deux technologies, qui doivent être prises en considération pour les décisions relatives à la conception :

- La principale différence technique vient du fait que les programmes SOA sont consacrés à la création de services Web pour faciliter les intégrations internes, de serveur à serveur, tandis que les API Web sont conçues pour accélérer la création d'applications Web et mobiles, souvent à des fins d'interface client.
- Les programmes SOA sont généralement initiés par les départements IT et centrés sur la réduction des coûts, alors que les programmes d'API émanent en général des organisations de développement commercial et visent à générer de nouveaux revenus.
- La plupart des projets SOA sont créés par et pour des architectes d'entreprise, afin de leur permettre d'intégrer plus facilement des systèmes hétérogènes et d'offrir de nouveaux services IT. Les programmes d'API, au contraire, sont conçus pour répondre aux besoins des développeurs d'applications.

Illustration 1 : SOA et API

	 SOA	 API
 Objectif d'intégration	Interne ou auprès de partenaires	Externe, souvent auprès de clients
 Motivation du projet	Coûts IT	Chiffre d'affaires
 Public de l'interface	Architectes d'entreprise	Développeurs d'applications

Objectifs en matière de conception des API

Néanmoins, de nombreux programmes d'API sont développés à partir d'initiatives SOA précédentes. Les services Web centrés sur les intégrations internes ou de partenaires s'ouvrent aux développeurs, qu'ils soient internes ou externes à l'entreprise. Au cours de ce processus, il est important que les concepteurs d'API gardent à l'esprit que les motivations et exigences d'un programme d'API sont très différentes de celles qui ont initialement poussé les entreprises à ouvrir leurs actifs IT via les services Web.

Dans cette optique, les objectifs majeurs en termes de conception des API peuvent être définis comme suit :

- Permettre le self-service pour les développeurs comme pour les utilisateurs d'applications
- Réduire les barrières freinant l'accès aux ressources précieuses de l'entreprise
- Hiérarchiser les besoins et préférences des développeurs d'applications clientes
- Encourager la collaboration entre et parmi les ressources internes et externes
- Traiter les problèmes de sécurité et d'évolutivité liés à l'exposition des actifs IT au marché ouvert

Surtout, la conception des API doit viser à optimiser la valeur métier de l'interface. Dans la 2^e partie, nous allons examiner en quoi les API apportent de la valeur à l'entreprise.



2^e partie : La chaîne de valeur des API

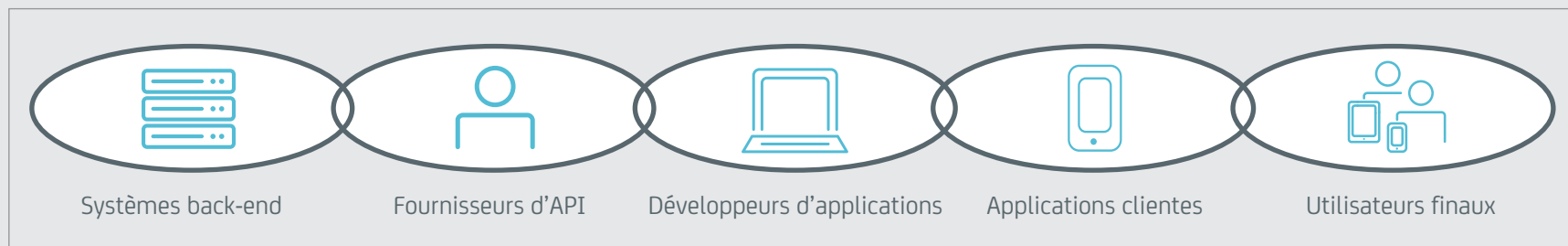
Si les API ne possèdent pas de valeur intrinsèque, elles apportent une valeur considérable à l'entreprise par le biais de leurs données back-end et de la fonctionnalité d'application offerte par l'interface. Dans cette optique, l'API est un simple facilitateur qui permet aux systèmes présentant une grande valeur pour l'organisation d'être réutilisés dans des applications plus susceptibles d'offrir une valeur métier directe.

Cette perspective est utile, mais une analyse plus approfondie révèle clairement qu'une API bien conçue est en fait un connecteur complexe et puissant. Elle permet de lier une grande variété d'actifs métier (systèmes IT, personnel interne et externe, applications clientes et

utilisateurs) afin de réaliser plus efficacement la valeur potentielle de ces actifs. Nous pouvons faire référence à cet état de fait en tant que « chaîne de valeur des API ».

Il est important de comprendre ce processus relativement complexe afin de ne pas perdre de vue que les API sont conçues pour offrir de la valeur métier, pas de l'efficacité technique. Bien que les API apportent de la valeur de manière plus directe que les architectures SOA, elles restent moins directes que le Web, dans la mesure où un site peut concrètement susciter des prospects ou des ventes. Les API génèrent des revenus de manière plus subtile, en reliant les différents actifs présentés ci-dessous.

Illustration 2 : La chaîne de valeur des API



Exemples de génération de valeur par les API

Chaque API a une valeur qui lui est propre, mais de manière générale, les entreprises peuvent se servir d'une API aux fins suivantes :

Génération directe de nouveaux revenus

Une API peut être une source directe de revenus si l'accès est payant pour les développeurs ou si l'interface est utilisée pour faciliter la création en interne d'applications fonctionnant sur le principe du paiement à l'utilisation, ou encore pour permettre le commerce électronique.

Augmentation de la portée et de la valeur client

Les API simplifient l'accession à de nouveaux clients ou l'augmentation de la valeur des clients actuels en offrant des services existants via de nouvelles plates-formes et de nouveaux périphériques.

Support des activités de vente et de marketing

Une API peut également aider une entreprise à proposer ses produits et services sur le marché en permettant la création d'une fonctionnalité attrayante et de qualité, inspirée des meilleures pratiques du marketing en ligne.

Stimulation de l'innovation métier et technique

Les API aident les organisations à développer de nouveaux systèmes ainsi que de nouvelles offres et stratégies car, en permettant l'implémentation d'idées sans changer les systèmes back-end, elles réduisent les obstacles à l'innovation.

Les décisions de conception

Les décisions relatives à la conception des API devraient être motivées précisément par les éléments que l'API va réunir, qui se trouveront de part et d'autre de l'interface : dans l'infrastructure IT de l'organisation et à l'extérieur du pare-feu de l'entreprise. Il est particulièrement crucial de répondre à ces deux questions :

- Quels sont les systèmes exposés et où (auprès de qui) résident-ils ?
- Qui sont les développeurs ciblés et quels types d'applications vont-ils concevoir ?

Il est primordial de déterminer qui sont les développeurs ciblés, car cette question est à la source de la classification fondamentale des API, « privées » ou « ouvertes ». Les API privées sont utilisées uniquement au sein de l'entreprise, ou dans certains cas, par les organisations partenaires. Les API ouvertes sont disponibles pour la communauté plus vaste des développeurs externes, libres de créer leurs propres applications à partir des ressources back-end de l'entreprise.

Les API privées se rapprochent de l'esprit des services Web. Généralement, une API privée vise à aider les développeurs internes, sous-traitants ou partenaires à créer des applications plus efficacement, pour une utilisation interne ou externe. Comme pour les services Web, les réductions de coûts constituent souvent la motivation première, car les API permettent de développer de nouvelles applications de manière plus économique. Cependant, de nombreuses API privées servent à créer des applications Web et mobiles destinées au grand public, qui génèrent une valeur métier plus directement.

Les programmes d'API ouvertes visent plutôt l'adoption. En permettant à des développeurs tiers d'accéder à leurs API, les entreprises cherchent à rendre leurs actifs IT disponibles pour le plus grand nombre d'utilisateurs possible. Ainsi, l'adoption par les développeurs est une métrique essentielle dans la mesure du succès d'une API ouverte. Bien que les API ouvertes soient moins nombreuses que les API privées, elles représentent les meilleures opportunités commerciales et les plus importants défis de conception/risques techniques.

En fait, les API ouvertes créent non seulement une série de défis totalement inédits en termes de conception et d'intégration (par exemple, comment ouvrir les systèmes back-end aux développeurs externes sans exposer lesdits systèmes aux pirates informatiques), mais également de nouveaux risques métier. Un programme d'API ouverte mal conceptualisé peut mener une entreprise à cannibaliser son activité principale et à potentiellement exposer ses actifs critiques à ses concurrents.

De telles considérations métier doivent peser dans les décisions liées à la conception technique. Nous allons à présent expliquer comment aligner considérations métier et décisions techniques dans la 3^e partie.

3^e partie : Conception des API et objectifs métier

Alors que l'architecture SOA a toujours cherché à améliorer les processus organisationnels, les programmes d'API visent à augmenter le chiffre d'affaires. De ce fait, les décisions de conception relatives aux API doivent véritablement prendre en compte les objectifs métier stratégiques du programme d'API de l'entreprise. Avant de commencer à concevoir une API, vous devez avoir clairement à l'esprit les problèmes que le programme d'API vise à résoudre, les opportunités qu'il cherche à saisir et la manière dont il va y parvenir. Il est particulièrement important de répondre aux questions suivantes :

- Quels actifs seront rendus disponibles ?
- Comment l'API va-t-elle rendre ces actifs disponibles ?
- Quels types d'applications peuvent être développées à partir de cette API ?
- Comment inciter les développeurs à utiliser l'API ?
- En quoi les applications vont-elles créer de la valeur pour l'entreprise ?

La communication et la collaboration sont les bases de la conception d'une API en mesure de répondre à ces défis et opportunités. Tout au long du processus de conception, de déploiement et de gestion d'une interface, les responsables de programmes et les architectes d'API doivent collaborer activement pour s'assurer qu'ils poursuivent des objectifs stratégiques communs et qu'ils s'entendent sur la manière d'atteindre ces objectifs et d'évaluer les résultats de leurs efforts. Les rôles métier et techniques doivent notamment se mettre d'accord sur les points suivants :

- L'objectif et l'état final idéal du programme
- Les tâches initiales qui vont permettre à l'organisation de travailler à la réalisation de ces objectifs
- Les métriques clés qui seront utilisées pour évaluer la réussite
- Les tâches courantes à mener pour garder le programme sur la bonne voie

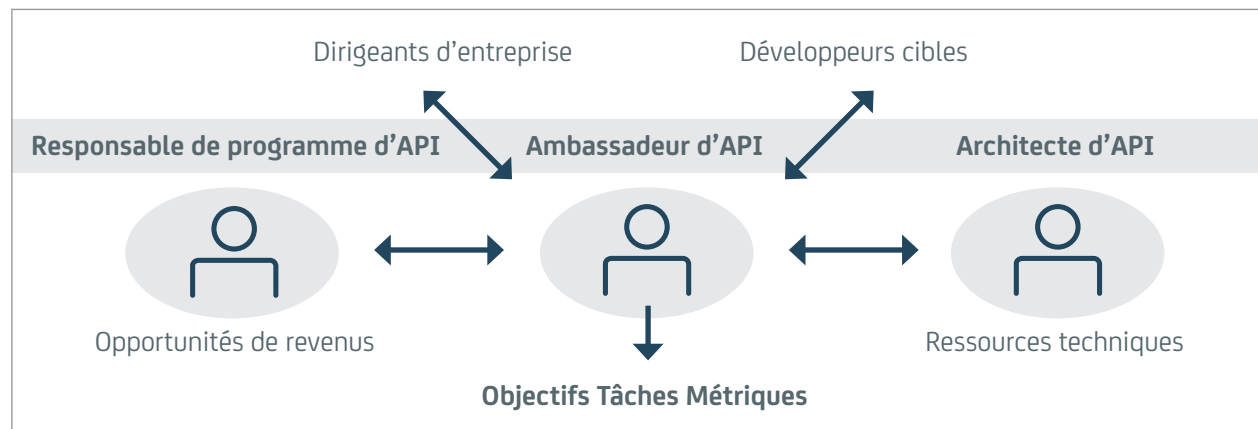
Attribution d'un sponsor

Pour que les responsables métier et les architectes restent sur la même longueur d'onde, le programme doit avoir un « sponsor » en mesure de réduire le clivage qui se crée souvent entre les départements techniques, les responsables métier et les développeurs d'applications. Les organisations commettent souvent l'erreur d'attribuer ce rôle à un responsable marketing non technique alors que cet « ambassadeur d'API » doit pouvoir comprendre les contraintes architecturales de l'organisation tout en partageant l'ardeur des développeurs d'applications.

Le rôle de l'ambassadeur consiste à établir une communication claire avec tous les intervenants, notamment :

- « Vendre » le programme d'API aux cadres et autres décideurs supérieurs
- S'assurer que les architectes d'API comprennent les objectifs métier des responsables de programmes
- Aider les responsables de programmes à comprendre les ressources et contraintes techniques des architectes
- Rassembler des informations sur les préférences et exigences des développeurs cibles

Illustration 3 : L'alignement des divers objectifs



Une fois la communication établie et une entente trouvée sur les objectifs, tâches et métriques, le véritable travail de conception de l'API peut commencer, ce dont nous traiterons dans la 4^e partie.

Remarques sur la stratégie métier relative aux API

Les responsables de programmes (ou « propriétaires d'API »), en collaboration avec l'ambassadeur d'API de l'entreprise, sont responsables de l'élaboration d'une stratégie métier claire pour les API et de la communication de cette stratégie aux décideurs exécutifs, ainsi qu'aux architectes et développeurs qui implémenteront l'aspect technique de cette stratégie.

La première étape consiste à établir un objectif métier clair et une déclaration de vision pour le programme d'API, qui soit alignée avec la vision élargie de l'entreprise. Une API n'est pas une solution purement technique et doit être traitée comme un produit ou une stratégie métier en soi, bien qu'intégrée dans la stratégie métier globale de l'entreprise.

En gardant cela à l'esprit, l'étape suivante consiste à élaborer un modèle métier autour de cette vision, en détaillant notamment les points suivants :

Coûts, ressources et efficacités

- Les systèmes, relations, activités et autres ressources qu'exploitera le programme, et comment il permettra à l'entreprise de mieux utiliser ces ressources

Valeur, revenu et innovation

- Les clients, marchés et canaux que ciblera le programme, et en quoi l'innovation technique permettra de générer de nouveaux revenus à partir de ces cibles

Ce modèle métier doit être centré sur une proposition de valeur qui souligne clairement la valeur concrète et mesurable que le programme d'API offrira à l'entreprise.



4^e partie : Conception d'une API utilisable

D'un point de vue purement technique, il est relativement simple de concevoir une API. Cela se complique lorsqu'il s'agit d'en concevoir une qui contribue véritablement à la valeur de l'entreprise. Outre la fonctionnalité, les architectes d'entreprise doivent prendre en considération les objectifs métier et l'expérience de l'utilisateur final.

Cela peut se révéler un défi de taille quand un projet SOA doit être étendu dans le domaine des API. Dans l'approche SOA, les besoins de l'architecte sont centraux, et l'adoption par l'utilisateur est présumée. Par conséquent, les architectes ayant une expérience SOA aborderont les décisions de conception relatives aux API en considérant que les utilisateurs de l'interface et de l'application auront les mêmes besoins et préférences qu'eux-mêmes. Cela se traduit quasiment toujours par de mauvaises décisions en matière de conception.

Avec les API, l'accent ne doit pas être mis sur la fonctionnalité, mais sur l'expérience utilisateur. La question clé n'est pas « Quelle fonctionnalité ai-je besoin d'exposer ? », mais plutôt « Comment les développeurs vont-ils utiliser cette interface ? ». Si les développeurs ne veulent pas utiliser votre API, elle n'a aucune valeur. De ce fait, la conception doit être centrée sur les développeurs et chercher à limiter au maximum la barrière à l'entrée pour le public de développeurs ciblé.

Qu'une API soit mise en ligne de manière privée ou ouverte, une expérience développeur (DX) positive est essentielle à son succès. L'expérience développeur est bien plus difficile à quantifier qu'une fonctionnalité exposée. Bien qu'elle puisse être définie comme la somme des interactions entre le fournisseur de l'API et le développeur, le résultat tient moins du chiffre que du ressenti : comment les développeurs perçoivent-ils l'interface ?

De toute évidence, il s'agit d'une métrique plutôt nébuleuse, mais il existe des mesures pratiques que vous pouvez prendre dans le monde réel pour comprendre comment vos développeurs risquent de réagir face aux différentes approches que vous adopterez pour concevoir votre API. Vous pouvez notamment :

- Créer des profils de développeurs
- Concevoir un prototype pour tester votre API sur le terrain

Profils de développeurs

Vous ne pouvez pas créer une API utilisable si vous ne connaissez pas les besoins et préférences de votre développeur cible. Généralement, les développeurs qui conçoivent les applications clientes à partir d'API sont décrits comme de jeunes « pirates informatiques », obsédés par les derniers langages et protocoles. Or, dans de nombreux cas (et notamment pour les API privées), les développeurs de services d'entreprise restent fidèles à des méthodes plus traditionnelles.

Le fait est que chaque projet d'API doit viser un public de développeurs spécifique pour réussir. Dans certains cas, il peut s'agir d'un groupe très homogène ayant des besoins communs. Dans d'autres, les préférences peuvent être très variées. Quoi qu'il en soit, vous devez discerner qui utilisera votre API et comment vous pouvez définir l'interface pour garantir que ces développeurs pourront rapidement et efficacement utiliser vos ressources back-end.

La première étape consiste donc à définir un personnage (ou un ensemble de personnages) pour établir le type ou les types de développeurs que vous ciblez avec vos API. Informations à inclure :

- Employeur (département) et raison pour laquelle ils développent une application
- Compétences en programmation, contraintes techniques et préférences en matière de langage/protocole
- Tempérament personnel et contexte idéal de travail

Prototypage

Une fois que vous avez saisi les objectifs professionnels, exigences techniques et préférences personnelles de vos développeurs cibles, vous pouvez commencer à concevoir une interface adaptée à ces critères. Toutefois, avant de créer une API de production liée à de véritables données ou systèmes back-end, il est préférable de commencer par développer un prototype léger, plus facile à modifier. Ce prototype vous permettra de tester les présupposés de conception établis à partir de votre personnage cible.

Illustration 4 : Outils utiles pour le prototypage d'API

Divers outils existent en ligne pour simplifier le processus de création et de test des prototypes d'API légers. Exemples connus...	1	Apiary apary.io	Outil de conception qui permet de créer rapidement un prototype d'API, sans écrire aucun code.
	2	RAML raml.org	Langages de description d'API qui peuvent aider les développeurs à découvrir et commencer à utiliser votre prototype d'interface.
	3	SWAGGER swagger.io	

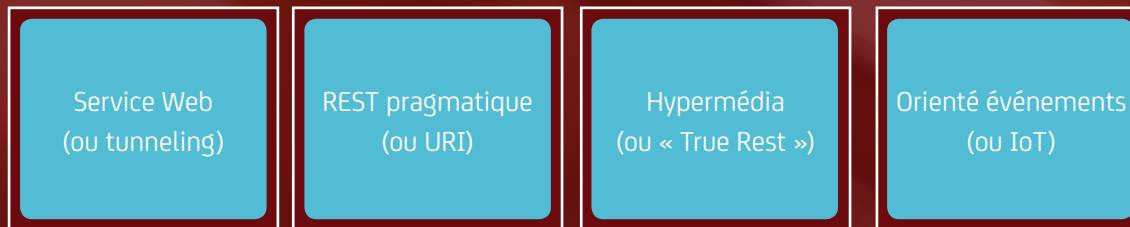
La conception d'un prototype léger basé sur des données ou une fonctionnalité « jetables » présente l'avantage de vous permettre d'appliquer une sécurité minimale et d'établir la barrière à l'entrée la plus faible possible pour les développeurs. Cela signifie que vous pouvez mobiliser vos développeurs cibles tôt dans le processus. Ils écriront des applications légères pour tester la conception de votre API et la commenter. Vous pourrez ensuite apporter des modifications à l'interface et la tester de nouveau. Après quelques itérations, vous devriez être sur la bonne voie.

Bien sûr, cela ne vous aide en rien pour la prise de décisions fondamentales et concrètes relatives à la conception de l'interface. Dans la 5^e partie, nous allons commencer à présenter les options effectives de conception des API.

5^e partie : Styles d'API

Le choix d'un style d'API est l'une des décisions les plus importantes pour un concepteur d'interface. De telles décisions sont inévitablement affectées par des considérations techniques, telles que la nature spécifique des ressources back-end exposées ou les contraintes de l'organisation IT. D'autres aspects, tels que les objectifs métier du programme d'API et les besoins et préférences du public de développeurs ciblé, doivent également être pris en considération.

À l'heure actuelle, les styles de conception des API peuvent être classés comme suit :



Service Web

Le style Service Web est une approche de la conception des API indifférente au transport et basée sur les opérations, qui utilise le langage WSDL (Web Services Description Language) pour décrire les interfaces. Il vient de l'univers SOA, où les interfaces Service Web étaient utilisées pour intégrer des réseaux hétérogènes. Ainsi, cela peut être un bon choix de style si votre programme comprend des interfaces SOA en expansion. Le vaste éventail d'outils qui existe pour les services Web signifie également que les applications clientes peuvent souvent être conçues rapidement et facilement.

Cependant, ce style présente d'importantes limitations. Tout d'abord, si ce style indifférent au transport peut utiliser le protocole HTTP (Hypertext Transfer Protocol), il est inefficace dans ce contexte. De ce fait, il ne s'agit pas du meilleur choix si vos services s'étendent vers le Web ouvert. De plus,

ce style n'est pratique que si vos développeurs cibles sont à l'aise avec les standards SOA tels que WSDL, SOAP (Simple Open Access Protocol) et RPC (Remote Procedure Call). Pour la plupart des développeurs clients, la courbe d'apprentissage risque d'être plutôt raide.

Cela se vérifie particulièrement pour les API ouvertes, et notamment celles centrées sur les équipements mobiles. En règle générale, les développeurs d'applications n'aiment pas le langage de programmation SOAP, et les outils disponibles pour la conception de clients Service Web ne prennent pas les appareils mobiles en charge. Outre les considérations pratiques, un problème de perception demeure : l'utilisation du style Service Web peut présenter votre organisation comme rétrograde, ce qui réduira inévitablement l'adoption parmi les développeurs d'applications mobiles.

REST pragmatique

Le style REST (Representational State Transfer) pragmatique est une approche de la conception des interfaces d'intégration plus simple et plus centrée sur le Web. Ce style, qui est spécifique au transport (il prend uniquement en charge HTTP) et utilise le standard URI plutôt que WSDL, s'est largement imposé face au style Service Web dans la conception des API d'entreprise. En effet, le terme « API Web » et « API RESTful » sont généralement utilisés indifféremment et atteindre un état de « RESTfulness » est souvent considéré comme un objectif majeur de tout projet de conception d'interface.

En fait, la plupart des API REST utilisées aujourd'hui ne correspondent pas pleinement aux critères REST décrits dans la thèse de doctorat de Roy Fielding rédigée en 2000. Alors que le style REST a été défini pour décrire de manière formelle le type d'interactions dynamiques liées par des liens hypertexte qui alimentent le Web, la plupart des API Web sont consacrées à l'échange de données statiques. Ainsi, il est plus précis de faire référence à ce style de conception en tant que « REST pragmatique ».

Il est aisé de comprendre pourquoi le style REST pragmatique est devenu aussi populaire. Le standard URI étant intuitif et les développeurs Web et mobiles habitués aux interfaces RESTful, l'adoption par les développeurs et leur productivité sont susceptibles d'être élevées. De plus, la focalisation sur HTTP fait des API REST pragmatique la solution idéale pour le développement des applications Web et mobiles d'aujourd'hui. Dans l'immédiat, ce style est vraisemblablement le choix idéal pour la majorité des projets.

Cependant, le style REST pragmatique ne convient pas dans tous les contextes et les évolutions à venir risquent de remettre en cause sa prédominance. Il présente bel et bien quelques inconvénients : il est limité à quatre méthodes, il peut être « verbeux » et la conception URI n'est pas standard. En outre, avec la forte expansion de l'Internet des objets (IoT) et du Big Data, qui affecte le réseautage en ligne, cette approche particulièrement centrée sur le Web risque d'être mise à rude épreuve.

Hypermédia

Le style de conception Hypermédia correspond à une approche basée sur les tâches, qui vise à fournir une alternative plus durable au style REST pragmatique. Comme les API REST pragmatique, les API Hypermédia sont généralement centrées sur les standards URI, HTTP et RESTful. En un sens, le style Hypermédia constitue une application plus fiable de l'architecture RESTful, selon Fielding, ce qui explique pourquoi le Web s'est révélé si évolutif.

En tant que telle, l'approche Hypermédia est encore plus centrée sur le Web : les liens hypertexte et formes du Web sont reflétés dans la manière dont une API Hypermédia fournit des liens pour parcourir le workflow et saisir des modèles à des fins de demande d'informations. Tout comme l'architecture

RESTful du Web s'est montrée hautement extensible et évolutive, une API Hypermédia bien conçue peut continuer à prendre en charge de nouvelles applications pendant de nombreuses années.

Si cette approche architecturale constitue de toute évidence une option attrayante pour les entreprises cherchant à créer des API évolutives qui prennent en charge les applications Web et mobiles sur le long terme, il s'agit d'un style de conception émergent qui manque encore cruellement d'outils associés. Cet état de fait risque d'impacter le taux d'adoption par les développeurs ainsi que de compliquer la tâche de ceux ayant choisi cette API et désireux de créer rapidement des applications clientes puissantes.

Orienté événements





Si les styles centrés sur HTTP tels que REST pragmatique et Hypermédia sont idéaux pour les applications Web et mobiles telles que nous les connaissons, l'émergence de HTML5 et de l'IoT change la donne, en rendant possibles des applications plus dynamiques, mais aussi en requérant des interfaces plus légères. Dans ce contexte, le style Orienté événements s'est révélé être une autre solution indifférente au transport, idéale pour permettre aux applications d'utiliser WebSocket et les autres alternatives à HTTP émergentes.

Ce style, centré sur les événements initiés par le serveur ou le client, constitue une option à faible coût et capable de meilleures performances dans les situations où un nombre élevé de petits messages transitent entre le

back-end et l'application. De ce fait, il est idéal pour l'IoT et toute une variété de cas d'utilisation mobile, notamment les messageries instantanées, les discussions vidéo, les jeux multijoueurs, etc. Il est également susceptible d'attirer les développeurs les plus avant-gardistes.

Bien sûr, tous les développeurs ne cherchent pas à tout prix à être à la pointe du progrès, et dans de nombreux cas, une approche RESTful conventionnelle sera plus appropriée. Le Web repose toujours sur le protocole HTTP et sa compatibilité avec les événements émanant du client est limitée. De plus, le modèle requête-réponse sur lequel ce style se base rend la conception d'applications clientes plus complexe pour les développeurs.

Illustration 5 : Styles architecturaux pour la conception des API

			
<p>Service Web</p>	<p>REST pragmatique</p>	<p>Hypermédia</p>	<p>Orienté événements</p>
<p>Lié à l'architecture SOA Nombreux outils disponibles Non adapté aux équipements mobiles</p>	<p>Idéal pour les applications Web et mobiles Connu de la plupart des développeurs d'applications Risque de manque d'évolutivité sur le long terme</p>	<p>Hautement centré sur le Web Extensible et évolutif Peu connu des développeurs</p>	<p>Adapté à l'IoT et aux périphériques Léger et dynamique Non adapté aux scénarios standard</p>

Votre choix de style dépendra de vos contraintes techniques, de vos objectifs métier ainsi que des préférences de vos développeurs. Veillez à ne pas succomber à un style « à la mode » s'il n'est pas approprié à votre contexte spécifique. Essayez toutefois de choisir un style qui sera évolutif et adaptable sur le long terme, au fur et à mesure de l'évolution de vos ressources, de la croissance de votre public d'utilisateurs et des changements inhérents au réseautage en ligne.

Quel que soit le style que vous choisirez, certains composants architecturaux devront être inclus dans votre API. Dans la 6^e partie, nous allons expliquer ces composants et leur organisation.

6^e partie : L'architecture des API

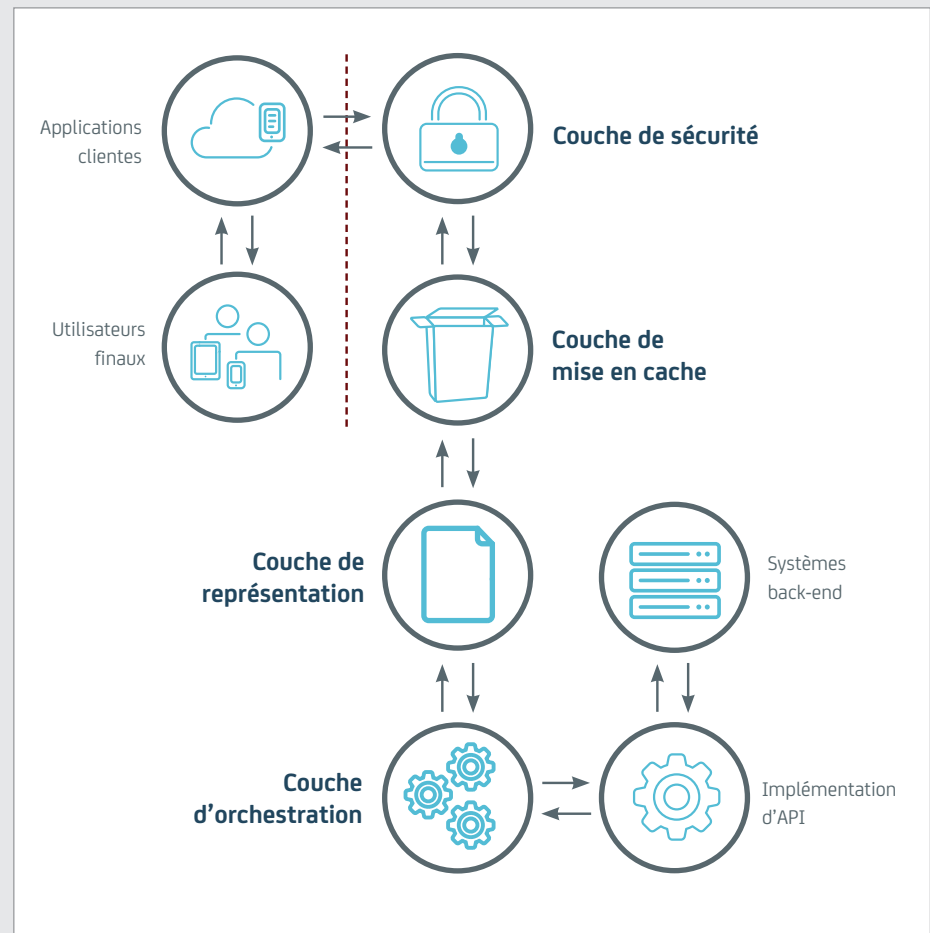
Les styles de conception architecturale précédemment évoqués servent de modèles pour vous permettre d'élaborer le cadre architectural qui soutient la fonctionnalité unique de votre implémentation d'API. Certains cas d'utilisation appellent l'implémentation de styles de conception spécifiques. Il est cependant important de noter qu'un certain nombre de composants doivent être inclus dans toute architecture d'API, quelle que soit l'utilisation visée.

Ces composants architecturaux communs ne doivent pas être intégrés dans l'implémentation d'une API donnée. Ils doivent plutôt être déployés dans une infrastructure d'API centrale, entre les API de l'organisation et les applications clientes qui exploitent ces API. L'extraction de ces composants accélère et simplifie la conception d'API supplémentaires, la mise à jour simultanée d'une série d'API et la supervision du bon fonctionnement des API, des systèmes back-end et des applications clientes.

Pour une efficacité maximale, ces composants doivent être organisés en couches, obligeant tout le trafic de données à passer par chacune des couches nommées à droite, dans l'ordre indiqué.



Illustration 6 : Couches architecturales



La couche de sécurité

Si elles ouvrent l'entreprise sur un monde d'opportunités commerciales, les API la soumettent potentiellement à de sérieuses menaces de sécurité, en exposant des systèmes et données back-end sensibles au monde extérieur. Les API sont vulnérables à nombre des menaces de sécurité qui infestent le Web ainsi qu'à une série de nouvelles menaces spécifiques aux API. De ce fait, il est vital de déployer une sécurité puissante, propre aux API, aux abords de votre architecture d'API.

Ce besoin de sécurisation peut entrer en conflit avec l'un des objectifs premiers de la conception des API : une API bien conçue permet aux développeurs de créer en toute simplicité des applications qui offrent un accès transparent aux ressources de l'entreprise. Une sécurité renforcée risque d'impacter cette facilité d'accès. Le déploiement de la sécurité dans une architecture d'API centralisée (plutôt que dans l'implémentation d'API) permet d'atténuer cet impact et d'utiliser des technologies de gestion des accès flexibles comme OAuth et OpenID Connect.

La couche de mise en cache

L'efficacité de l'interface est essentielle pour offrir les expériences développeur et utilisateur final fluides qui vous permettront d'atteindre vos objectifs en termes d'adoption et de rétention de votre programme d'API. Pour optimiser l'efficacité des API, une solution consiste à placer une couche de mise en cache aux abords de l'architecture d'API. Cette couche permet de mettre en cache les réponses aux requêtes courantes, réduisant ainsi la pression exercée sur les implémentations d'API et les ressources back-end.

La couche de représentation

De toute évidence, la présentation de votre API doit être aussi conviviale que possible pour les développeurs. En isolant cet élément de l'implémentation, vous pouvez vous consacrer à la création centralisée d'un environnement accueillant, sans impacter les API ou les ressources sauvegardées elles-mêmes. Il devient ainsi beaucoup plus simple de présenter des systèmes back-end complexes sous la forme d'interfaces centrées sur le Web et la mobilité, que les développeurs peuvent rapidement comprendre et exploiter afin de créer des applications performantes et conviviales.

La couche d'orchestration

Si certaines applications sont en mesure de fournir de la valeur en accédant à une ressource unique via une seule API, les possibilités augmentent de manière exponentielle lorsque vous combinez les données de plusieurs API (dont celles d'autres entreprises) et ressources back-end. Le déploiement d'une couche d'orchestration à proximité des interfaces permet de telles combinaisons et simplifie le processus de composition de nouvelles implémentations à partir de ressources back-end multiples.

Pour créer une architecture d'API centralisée, la solution la plus efficace consiste à déployer une solution de gestion des API. Dans la 7^e partie, nous allons étudier les composants clés de la gestion des API.

7^e partie : Gestion des API

L'élaboration d'une infrastructure qui centralise les composants architecturaux communs aux API sécurisées et centrées sur les développeurs peut considérablement simplifier l'implémentation d'API précieuses pour votre entreprise. Toutefois, la conception d'une telle infrastructure en interne peut représenter un défi considérable. Heureusement, nombre d'éditeurs de logiciels d'entreprise offrent désormais des solutions de « gestion des API » qui évitent d'avoir à développer cette infrastructure critique en interne.

De plus, comme leur nom l'indique, les solutions de gestion des API incluent également des fonctionnalités de gestion et d'optimisation des performances des API sur le long terme. Les solutions les plus puissantes possèdent en outre des fonctions permettant de créer une interface Web, via laquelle les développeurs peuvent découvrir les API et y accéder. Cet aspect essentiel de la présentation d'une API centrée sur le développeur ne peut pas être directement intégré à l'implémentation.

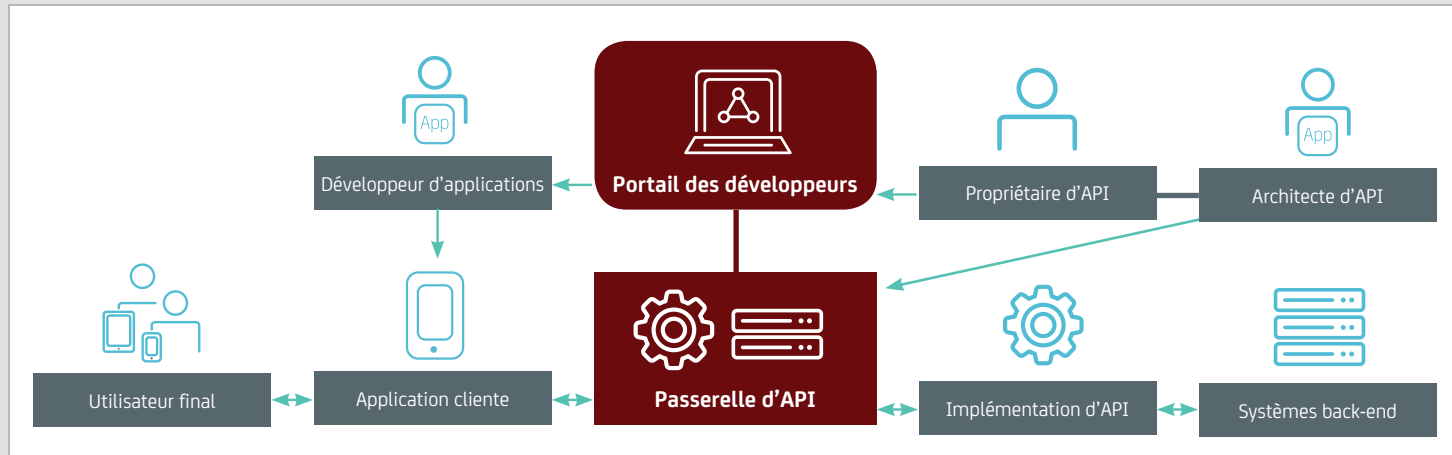


Les composants de la gestion des API

Une solution de gestion des API à l'échelle de l'entreprise comprend deux composants majeurs :

- Une passerelle d'API, qui fournit les fonctionnalités de sécurité, de mise en cache et d'orchestration nécessaires au déploiement d'une architecture d'API centrale
- Un portail des développeurs, qui offre une interface personnalisable via laquelle les développeurs accèdent aux API ainsi qu'à la documentation, aux forums communautaires et à tout autre contenu utile

Illustration 7 : Les composants de la gestion des API



Il est important de noter que la gestion des API n'est pas simplement une exigence technique. Elle joue inévitablement un rôle dans le succès commercial de tout programme d'API d'entreprise. La gestion de la composition, des performances et de la sécurité des API d'entreprise est essentielle pour que l'organisation obtienne un bon retour sur investissement dans le cadre d'un programme d'API. De la même manière, il est crucial de mobiliser et de gérer activement les développeurs afin de s'assurer qu'ils conçoivent des applications qui créent de la valeur métier.

Pour la plupart des entreprises, une infrastructure de gestion des API se révèle essentielle pour la conception, le déploiement et la maintenance des API que les développeurs utiliseront pour créer de nouvelles applications réellement performantes.

Découvrez l'essentiel de la gestion des API en lisant l'eBook « 5 piliers de la gestion des API »

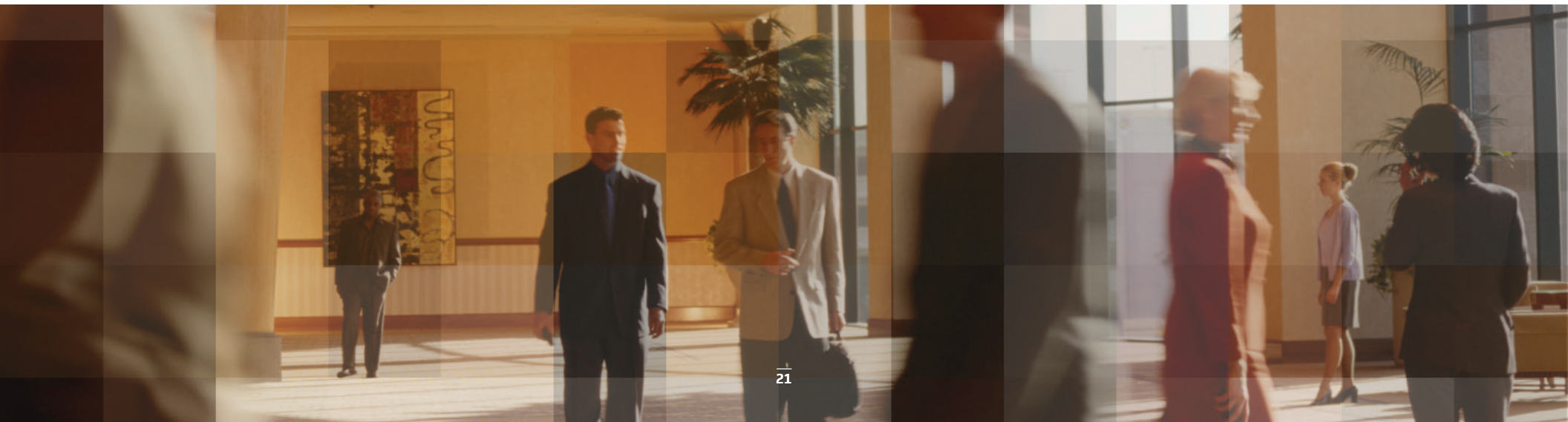
Conclusion

D'un point de vue architectural, les API représentent une extension de SOA. Tout comme l'architecture SOA créait des interfaces pour permettre la réutilisation de systèmes hérités dans de nouveaux services susceptibles de lisser les frontières organisationnelles, les API sont employées pour ouvrir les services back-end de l'entreprise aux développeurs qui conçoivent des applications pour les équipements mobiles et le Web public. Il s'agit d'une extension significative et les exigences de conception inhérentes à une API Web seront certainement bien différentes de celles d'un Service Web SOA.

Alors que les programmes SOA émanent généralement d'un besoin de réduction des coûts IT, les programmes d'API visent à générer de nouvelles sources de revenus. Une API Web connecte divers actifs métier existants afin de créer de la valeur de manière inédite. Une API bien conçue est toujours centrée sur les résultats métier. Par conséquent, les pratiques relatives à la conception et l'architecture des API doivent être alignées avec la stratégie métier de l'organisation, de A à Z.

Les propriétaires et les architectes d'API doivent communiquer pour s'assurer qu'ils sont d'accord sur les objectifs clés, sur la manière de les atteindre et sur la méthode de mesure de leur succès. Pour garantir une communication efficace, un ambassadeur d'API apte à combler les écarts entre les rôles métier et techniques doit analyser les besoins des dirigeants, des propriétaires d'API, des développeurs d'applications et des architectes d'entreprise, afin de négocier un ensemble d'objectifs, tâches et métriques approprié.

En pratique, la conception d'une API à des fins commerciales implique généralement la création d'une interface que les développeurs auront envie d'utiliser. De ce fait, avant de concevoir quoi que ce soit, il est primordial d'enquêter sur votre public de développeurs afin de comprendre qui vous ciblez et ce qu'ils attendent d'une API. Il peut également être utile de tester vos suppositions quant aux préférences des développeurs en proposant des prototypes d'API légers.



Lorsque vous serez prêt à concevoir votre véritable implémentation d'API, vous devrez choisir le style de conception qui correspond le mieux à votre projet. Les API Service Web conviennent aux programmes internes prévus pour les développeurs ayant l'expérience de l'architecture SOA. Les API REST pragmatique sont mieux adaptées aux projets d'API ouvertes centrées sur les périphériques mobiles et le Web. Les styles Hypermédia et Orienté événements émergent comme des approches potentiellement plus durables dans un avenir reposant sur la mobilité et l'IoT.

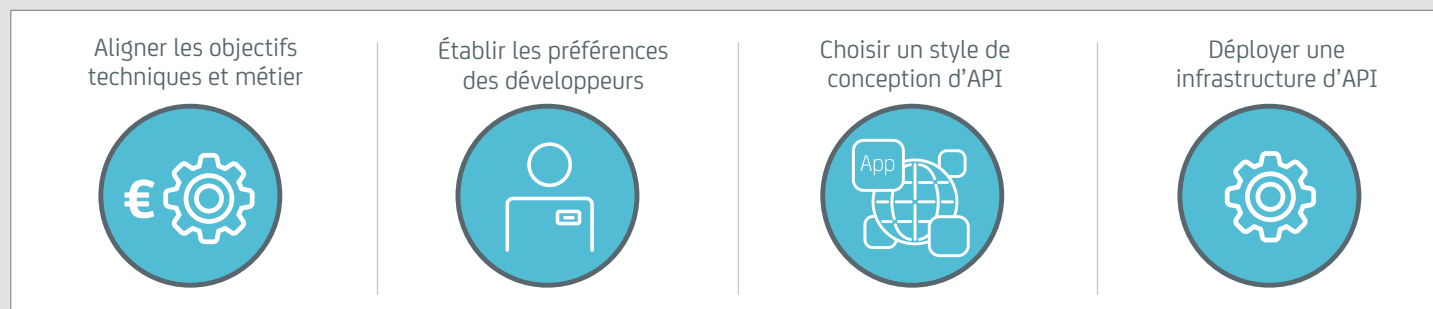
Quel que soit le style, toutes les API doivent inclure certains éléments architecturaux, à savoir la sécurité, la mise en cache, la représentation et l'orchestration. Pour une efficacité et une gestion maximales, ces éléments ne doivent pas être intégrés dans des implémentations d'API spécifiques. Toutes les API doivent plutôt exploiter une architecture d'API centrale, organisée en couches, à la limite de l'entreprise et des API elles-mêmes.

La manière la plus efficace de déployer une architecture d'API centrale, et de s'assurer que le programme d'API reste un succès à long terme, consiste à adopter une solution de gestion des API. Il existe une variété de solutions sur le marché, mais la plupart incluent deux composants communs :

- Une passerelle d'API qui fournit des fonctionnalités de sécurité et l'infrastructure clé
- Un portail des développeurs qui simplifie le processus de mobilisation et d'habilitation des développeurs

Les projets d'API d'entreprise actuels présentent de nombreux enjeux : d'énormes opportunités commerciales, des risques de sécurité significatifs et bien plus. Il est primordial que vous respectiez les étapes préalables à la conception d'une API : aligner les objectifs de conception et les objectifs métier, établir les préférences de vos développeurs cibles, choisir un style d'implémentation approprié et déployer une infrastructure de gestion des API. Alors, vous serez prêt à concevoir une API réellement utile.

Illustration 8 : Prérequis pour une conception efficace



Seule la solution CA API Management permet aux organisations d'intégrer des systèmes, de simplifier le développement des applications et de monétiser les données avec le niveau de sécurité et de protection des API dont les entreprises ont besoin à l'heure actuelle. Pour en savoir plus sur CA API Management, consultez le site ca.com/fr/api.

À propos de CA API Management

Avec plus de 300 clients dans des secteurs aussi divers que la communication, la finance, les services publics et la vente au détail, la solution CA API Management de CA Technologies offre une technologie de pointe et un savoir-faire qui aident les organisations à créer de la valeur via les API. CA Technologies fournit une solution de gestion des API complète, incluant une passerelle d'API entièrement opérationnelle, avec des fonctions de sécurité de niveau militaire, accompagnée dans les versions sur site et SaaS, d'un portail des développeurs. Pour en savoir plus sur CA API Management, consultez le site ca.com/fr/api.

API Academy

Services de stratégie, d'architecture et de conception des API

L'équipe API Academy est composée d'experts du secteur rassemblés par CA Technologies pour développer des ressources gratuites pour la communauté et fournir des services de consultation d'experts pour les organisations qui souhaitent faire passer leurs programmes d'API à la vitesse supérieure. Pour en savoir plus sur la manière dont API Academy peut aider votre organisation en matière de stratégie, d'architecture et de conception des API, consultez le site apiacademy.com.

CA Technologies (NASDAQ : CA) fournit les logiciels qui aident les entreprises à opérer leur transformation numérique. Dans tous les secteurs, les modèles économiques des entreprises sont redéfinis par les applications. Partout, une application sert d'interface entre une entreprise et un utilisateur. CA Technologies aide ces entreprises à saisir les opportunités créées par cette révolution numérique et à naviguer dans « l'Économie des applications ». Grâce à ses logiciels pour planifier, développer, gérer la performance et la sécurité des applications, CA Technologies aide ainsi ces entreprises à devenir plus productives, à offrir une meilleure qualité d'expérience à leurs utilisateurs, et leur ouvre de nouveaux relais de croissance et de compétitivité sur tous les environnements : mobile, Cloud, distribué ou mainframe. Pour en savoir plus, rendez-vous sur ca.com/fr.

Copyright © 2015 CA Tous droits réservés. Tous les noms et marques déposées, dénominations commerciales, ainsi que tous les logos référencés dans le présent document demeurent la propriété de leurs détenteurs respectifs. Ce document est fourni à titre d'information uniquement. CA décline toute responsabilité quant à l'exactitude ou l'exhaustivité des informations qu'il contient. Dans les limites permises par la loi applicable, CA fournit le présent document « tel quel », sans garantie d'aucune sorte, expresse ou tacite, notamment concernant la qualité marchande, l'adéquation à un besoin particulier ou l'absence de contrefaçon. En aucun cas, CA ne pourra être tenu pour responsable en cas de perte ou de dommage, direct ou indirect, résultant de l'utilisation de ce document, notamment la perte de profits, l'interruption de l'activité professionnelle, la perte de clientèle ou la perte de données, et ce même dans l'hypothèse où CA aurait été expressément informé de la survenance possible de tels dommages.