



Blocos de Construção

Exposição de APIs:
do Backend ao Engajamento de Parceiros



O que são Blocos de Construção?


Assim como crianças brincam com seus kits de blocos de madeira e produzem formidáveis castelos, ou marmanjos brincam no Minecraft e produzem [cidades, relógios e calculadoras](#), existe uma base para se construir projetos incríveis, que é simples e comum a todos esses projetos.

Ou seja, todos têm à disposição o mesmo conjunto inicial de blocos para montar e criar. O diferencial que faz cada projeto ser incrível (ou não) é a visão e criatividade do construtor e não o conjunto de blocos.

Mesmo assim, saber quais são os blocos disponíveis e quais combinações costumam ser as mais bem sucedidas não é má ideia. É isso que você terá nesse Ebook!



O que são Blocos de Construção?



É importante notar que as dicas dadas nesse Ebook são baseadas em diversos exemplos reais de APIs, mas de modo geral, você verá mais Blocos sobre a parte visível da API do que com seu Design. Ou seja, não será abordado tanto aqui de Desempenho ou Segurança (só um pouquinho).

Então, esses Blocos se encaixam mais na parte de Exposição e Gerenciamento da API do que em Design.


Se você estiver buscando conteúdo para o Design, temos outros materiais disponíveis [aqui](#), em forma de Blog posts, Webinars e Slides. Também temos [vídeos no Youtube](#). Não deixe de conferir ;)

Vamos começar!




Termos úteis

(ou como usar esse Ebook da melhor forma)





Ao longo do Ebook, serão usados alguns termos que talvez não sejam óbvios para todas as pessoas. A grande maioria deles está a um Google de distância, mas para que digitar? Aqui está a ajuda, para não gerar tantas dúvidas:

- 
- 1 - Blocos e Conjuntos de Blocos:** Esse Ebook é dividido em diversos Conjuntos, que unem tipos de Blocos de Construção parecidos. Por exemplo, daqui a alguns slides, você verá o título “Integração do dev/Onboarding”, que é um conjunto de Blocos de Construção referentes à primeira impressão de alguém com sua API. As subseções (“Portal de devs” e “FAQ”, por exemplo) são os Blocos de Construção, em si.
 - 2 - API:** Chegou a esse Ebook tentando entender o que é uma API? Não é o melhor lugar. Aqui, é assumido que você entende o que é API e está pensando em implementar uma. Se quiser entender melhor o que é uma API e como ela funciona, [clique aqui](#) ou [aqui](#). Se já souber o que é uma API, mas quer entender como encaixá-la nos seus negócios, também [temos um material bacana para você](#).




Termos úteis

(ou como usar esse Ebook da melhor forma)

- 
- 3 - Dev:** Desenvolvedor. Aquela pessoa que terá interesse em usar sua API para integrá-la à outra aplicação. O Dev pode ser um parceiro, alguém da sua própria empresa querendo integrar sistemas internos ou mesmo um Dev externo, que você não conhece (e talvez nunca venha a conhecer). [Saiba um pouco mais sobre o assunto aqui](#).
 - 4 - App:** O objetivo final de alguns devs interessados na sua API. App significa aplicação, ou seja, um software que faça algo, chamando a sua (e talvez outras) APIs.
 - 5 - URI:** é o identificador de acesso à algum recurso na Internet. Através da URI [sensedia.com/blog](#), um browser consegue acessar a página /blog dentro do site da sensedia. Algo semelhante ocorre para os métodos acessíveis de uma API. Esse assunto é bastante técnico e influencia diretamente no Design de sua API. Recomendo que [assista a esse Webinar](#) para entender um pouco mais do assunto.
- 



Integração do dev/Onboarding



Começando pelo começo. Quando um dev tem o mais básico contato com sua API, você já deve começar a aplicar alguns Blocos. Esse contato deve ser a cara da sua API, a primeira impressão. E você sabem, normalmente essa impressão é a que fica.

Lembre-se de caprichar bastante no Onboarding, porque quando estivermos falando de Suporte, mais à frente no Ebook, você perceberá a importância de ter as informações mais introdutórias e facilitadoras de fácil acesso. Senão, você estará gerando demanda desnecessária para outras equipes, como a de Suporte.

Estamos assumindo, nesse conjunto de Blocos, que já existe alguma quantidade de pessoas chegando a sua API. Se quiser saber mais sobre como trazer devs, você deve estudar um pouco sobre Marketing Digital. Felizmente, temos um conteúdo específico para [Marketing de APIs](#) para ajudá-lo.

Integração do dev/Onboarding

Portal de devs



O portal é composto por diversos dos itens que serão descritos abaixo. Em especial, ele precisa trazer informações ricas e de fácil acesso ao mesmo tempo. Isso pode ser um desafio porque as APIs costumam ser repletas de detalhes (o que é ótimo), mas nem sempre é fácil de categorizar e organizar isso.

Em particular, o Portal deve conter links para acesso rápido à Documentação da API, meios de comunicação (como fóruns e Suporte) e um guia para começar a usar a API (Getting Started). Esses itens são Blocos por si só e serão abordados no Ebook em mais detalhes.



Integração do dev/Onboarding

Lembre-se que uma métrica importante nessa etapa é o TTFHW, ou o Time To First Hello World: o tempo necessário para que qualquer pessoa interessada na sua API consiga efetivamente codificar algo com ela. (se quer aprender um pouco mais sobre métricas importantes para APIs, confira nossos [Slides de Indicadores para APIs](#)).


Exemplos: [Google Developers](#), [SmartThings](#), [Noteflight](#), [Spotify](#)

FAQ (frequently asked questions)

Forneça uma lista de perguntas comuns e as respostas para tais dúvidas. Isso reduz as dificuldades no Onboarding (o que aumenta as chances de cativar o dev) e reduz a necessidade de pessoas em prontidão para tirar dúvidas, como a equipe de Suporte.




Integração do dev/Onboarding



Muitos FAQs também não possuem uma barra de busca, o que na minha opinião, deixa de fazer sentido. Se alguém está com dúvida sobre sua API e quer tirar essa dúvida, como você espera que ela a encontre no meio de outras 30 perguntas? O objetivo de um FAQ é facilitar a vida das pessoas, então esforce-se para isso!

Exemplos: [SmartSheet](#), [OpenOffice](#), [TextWise](#), [Intercom](#), [Mailgun](#), [Discogs](#)

Getting Started



Essa é a menina dos olhos do Onboarding. A página de documentação que contenha uma Visão Geral da API, juntamente com exemplos de uso e uma lista de ações simples é o ideal para aquele dev recém-chegado.

Integração do dev/Onboarding

Novamente, tome consciência de que a ideia é minimizar os passos para começar o uso da API (o já comentado TTFHW, Time To First Hello World).

Exemplos: [HipChat](#), [GitHub](#), [Youtube](#)

Email para cadastro e comunicação

[Hoje em dia, as pessoas já tem email antes de ter RG.](#) O email está tão centralizado em nossas vidas que eu não consegui encontrar uma API que não tivesse como forma de comunicação o email. Até em casos como do Facebook, em que ele usa sua conta pessoal do Facebook para o cadastro, o seu email já está registrado na sua conta.

O email permite comunicação eficiente e consagrada com seus devs. Não reinvente a roda.

Integração do dev/Onboarding

Cadastramento automático

Há APIs que necessitam que o cadastro de um novo dev seja aprovado por um ser humano.

Há casos em que isso faz sentido, mas de modo geral, a vasta maioria das APIs abertas ao público devem possuir um canal de cadastros automático, que funcione 24 horas por dia e 7 dias por semana.

Exemplos: [Alchemy](#), [Facebook](#) (usando sua conta pessoal), [Ford](#)

Integração do dev/Onboarding

As melhores práticas



Cada API tem as suas peculiaridades. O que pode ser um passo complicado na sua API? Quais métodos e requisições geram mais dúvidas? A fim de melhorar a clareza, há algum ponto em que é interessante ser detalhista?

Em alguns casos, uma página com As melhores práticas pode ser substituída por outros dos Blocos acima, como o Getting Started e o FAQ. Porém, se realmente houver peculiaridades de sua API em relação à outras, não deixe de expor esse fato.

Quão específico você é nisso pode ser a diferença entre devs conseguindo ou não utilizar a API.

Exemplos: [Github](#) (barra lateral à direita), [Bitly](#)

Integração do dev/Onboarding

Introdução aos termos legais



Toda API deve ter alguns Blocos Jurídicos, explicitando os termos de uso, Política de Privacidade, Licenciamento dos Dados e do Código desenvolvido. Esse é um conjunto de Blocos, que será tratado adiante.

Por enquanto, preocupe-se em deixar à mão do dev uma versão resumida e simplificada desses termos (que pode ser lida em 5 minutos ou menos), além de uma forma fácil de encontrar toda a documentação legal.

Exemplo: [Facebook Messenger](#) (Releasing your app)



Documentação



A segunda camada de interação de um dev com sua API, logo após as práticas citadas no Onboarding: a Documentação. Uma vez que o dev está familiarizado com sua API, ele irá começar a procurar informações ricas e completas sobre sua API, uma vez que o objetivo dele é integrar algum sistema com a sua API.

A principal fonte desse conhecimento deve estar facilmente acessível em seu Portal de Devs através da sua página de Documentação.

O balanço entre riqueza de informações e dados simples e organizados é difícil de ser alcançado. Para isso, a equipe envolvida no Design da API (normalmente mais de uma pessoa) precisa delimitar parâmetros de uso da API, que serão os principais blocos de texto na Documentação.




Documentação



Além disso, é bastante interessante que pessoas externas ao Design original da API participem da confecção da Documentação. Isso porque é comum que a equipe que criou a API não perceba detalhes ou pontos críticos, por estar acostumada ao Design e ter uma visão enviesada.

Mas o que deve estar presente na Documentação de uma API? Alguns fatores:



Lista de Recursos



Toda API tem uma série de métodos (disponibilizados através de [endpoints](#)) e dados disponíveis. A listagem desses dados é o mais básico que uma Documentação deve ter, além de ser um primeiro passo que faz muito sentido na criação de mais conteúdo.



Documentação



Ou seja, comece descrevendo o que cada método faz, qual URI acessar, quais são os parâmetros de entrada e as saídas esperadas. Essa parte da Documentação deve se tornar a principal referência para os devs e é através dessa lista que alguém conseguirá delinear um plano para integrar seu aplicativo ao seu serviço.

Exemplo: [Instagram](#), [Bitly \(Metrics\)](#)

Códigos de Erro/HTTP Status

Outra lista importante de ser mantida na Documentação são os códigos de erro ou mensagem de status HTTP da sua API.



Documentação

[Os HTTP Status Codes são padronizados](#). Porém, os códigos de erros podem ser muitos e diversos. Aproveite essa página da Documentação para descrever todas as situações em que o dev pode encontrar problemas e erros com sua API.

Também é interessante que o Bloco de Melhores Práticas sejam descritas em conjunto com os códigos de erro, para que as boas práticas cubram possíveis situações de erro, e assim o dev possa tratar essas mensagens de forma adequada na aplicação que ele estiver criando.

Exemplos: [Twitter](#), [Microsoft Azure](#), [AWS](#)

Documentação

Explorador da API/Documentação Interativa




Essa é uma forma de permitir que pessoas testem sua API de forma rápida e intuitiva, através de uma interface web. Ou seja, sem a necessidade de instalar nada, e com um know how bem menor do que o necessário para explorar todo o potencial da API, qualquer um pode olhar o que sua API pode fazer e explorar seus métodos e recursos.

Dando um passo além da documentação estática e descritiva, a Documentação da sua API pode seguir de forma muito mais dinâmica através de serviços como o [Swagger](#), em que um dev pode autenticar, navegar pelos métodos e realizar chamadas em tempo real, com retornos de saída ou códigos de erro.

Isso permite que estudantes, professores, jornalistas e qualquer usuário com algum conhecimento técnico (mas ainda menor do que um dev) consiga brincar com a API e explorar seus dados.



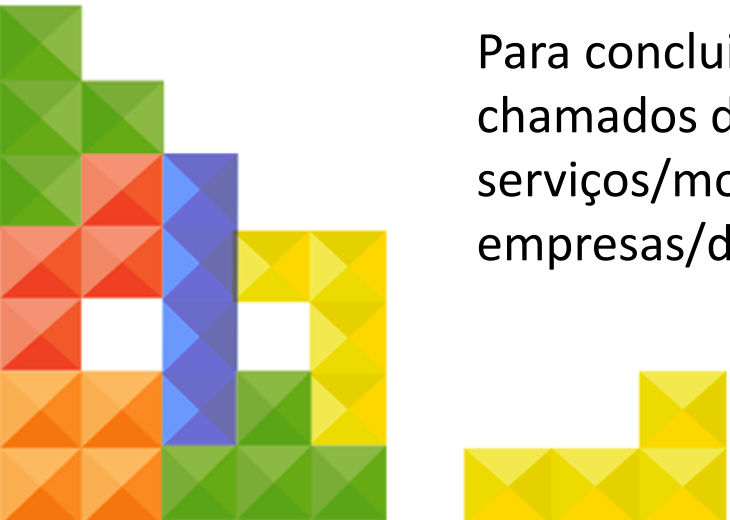
Documentação



É importante notar que esse tipo de documentação não é um Sandbox. O Sandbox é um ambiente de testes completo, em que todo o desenvolvimento de uma aplicação para a API pode ser feito. A Documentação Interativa serve para testes simples, para que o funcionamento de determinado método possa ser entendido com clareza.

Exemplos: [Catho](#), [Adobe](#), [Wolfram|Alpha](#), [Marvel](#), [Klout](#)

Taxas-limite



Para concluir o Bloco de Documentação, uma característica essencial é a quantidade de chamados disponível por usuário ou aplicação. Isso pode variar caso você use serviços/monetização na sua API (por exemplo, você pode fazer parcerias com empresas/devs específicos e eles possuem mais chamadas à sua API).

Documentação

É possível limitar por método/grupo de métodos, por tempo determinado, etc. O caso dos Serviços e Monetização será tratado mais adiante no ebook.

Os demais casos são bem variados e tem forte relação com sua infraestrutura disponível (por exemplo, não vale a pena deixar alguns devs derrubarem seu serviço, usando milhares ou milhões de chamados) e sua estratégia.


De modo geral, o importante é ter essa informação clara e acessível, assim como a mensagem de erro que será gerada caso alguém ultrapasse o limite.

Exemplos: [Fitbit](#), [Zendesk](#)



Materiais educativos

Esses materiais são um conjunto de conteúdos criados com o objetivo de divulgar todo o poder de sua API e impulsionar novos apps e ideias para o ecossistema da sua API.




De modo geral, essa é uma prática mais voltada a área de Marketing, uma vez que busca criar buzz em torno da API, seja usando casos de sucesso, eventos e guias, entre outras estratégias.

No blog da Sensedia você pode ler a [série sobre Como Criar um Ótimo Plano de Marketing](#) para APIs. Os materiais descritos aqui no Ebook complementam essa série. Na série, a suposição é que você conhece (ou precisa aprofundar o seu conhecimento) sobre quem está usando sua API e adequar sua comunicação às diferentes partes interessadas.



Materiais educativos



Aqui, essa suposição também é válida, mas não é essencial, uma vez que são tratados os Blocos de Construção, e passam a valer mais os formatos e modelos do que a estratégia/plano, em si. Porém, [não deixe de conferir a série](#), seus conhecimentos ficarão bem mais completos ;)

Estudos de caso

Cases são um dos tipos mais convincentes de material que você pode criar.

O Marketing costuma usar esse formato nas etapas finais de venda: aquela em que a pessoa já sabe que tem um problema (relacionado ao que se quer vender), mas ainda não tem certeza que precisa comprar, ou que deve comprar de determinado fornecedor.

Materiais educativos

Ao ver um case, a dor do cliente do case pode se alinhar ao seu prospect e costuma ser tiro e queda.

Portanto, você viu um caso de aplicação que deu muito certo com sua API? Escreva um case, crie uma apresentação, entreviste o dev! Melhor ainda, forneça algum benefício para o dev, como a redução dos limites de chamadas à sua API.

Talvez ele crie apps ainda mais legais, o que significa mais cases para você. Só tome cuidado para não se empolgar, pois os cases devem ser curtos e objetivos.

Exemplos: [Youtube](#), [Moz](#)

Materiais educativos

Loja/Galeria de apps

O propósito final de diversas APIs é ter uma gama legal de aplicações.

Essas aplicações contribuem para satisfazer o efeito Cola de uma API (que foi comentado na de Monetização), trazem prova social à sua API (novos devs podem ter boas ideias e verificar que sua API tem uma comunidade ativa), além de também darem aos criadores da API noções sobre seu próprio produto que talvez eles próprios não tenham tido.

Ou seja, trata-se de uma lista de cases de sucesso e seu papel é tornar essa lista atrativa e útil, tanto para novos devs quanto para usuários finais.

Exemplos: [Evernote](#), [Zendesk](#), [Moves](#)

Materiais educativos

Guias e tutoriais

Por natureza, devs costumam ser autodidatas, fazendo testes e verificando as possibilidades por si mesmos. É claro que alguma ajuda é sempre bem-vinda.

Guias podem estar no [Codecademy](#) ou em praticamente qualquer site de TI (como no [GigaOM](#)), mas é sempre muito interessante ter esse conteúdo educativo acessível a partir da Documentação da API. O formato pode ser qualquer um, desde texto (com exemplos, por favor), passando por vídeos e até formatos interativos.

A profundidade do assunto também pode ser uma preocupação. Normalmente os assuntos mais simples podem ser pulados -- se alguém está considerando usar sua API, provavelmente não é um leigo.

Materiais educativos

E se for um leigo, pode começar a brincar nos tutoriais do Codecademy ou na sua Interface web do Explorador da API (como comentado há algumas páginas atrás).

Exemplos: [MediaWiki](#), [Stream](#) (vale a pena se cadastrar só para ver o tutorial da API).

Webinars e vídeos

Nem todo mundo gosta de ler ou entende o que está escrito. Algumas pessoas entendem melhor em forma de vídeo.

Como comentado acima, produza conteúdo em formatos diversificados. Na Sensedia, gostamos de fazer Webinars ([nosso último foi o de Gerenciamento de APIs](#)) e acabamos produzindo conteúdo relacionado no blog ou em outros canais (como iMasters e [ComputerWorld](#)) com o aprendizado dos Webinars.



Materiais educativos

Então, use esse tipo de formato para diversificar a forma como ensina e também como aprende! Outra forma de divulgar seu material educativo é através do [Slideshare](#).

Exemplos: [Axosoft](#), [Chatter](#)



Eventos

Desenvolvedores adoram eventos. Você também, provavelmente, está bem atento aos eventos específicos da sua área. Mas por que também não participar de eventos de TI ou até criar seus próprios hackathons?

Se for palestrar, melhor ainda: grave sua palestra e deixe gravação e slides acessíveis para quem não viu ao vivo.

Exemplo: [Facebook](#)

Comunicação com devs e parceiros



Talvez esse Conjunto não seja nenhuma novidade. Serão tratadas ferramentas de Comunicação, ou seja, as principais formas de garantir que seus devs e parceiros conheçam sua API, se atualizem a respeito de novas versões ou qualquer informação que você deseje divulgar e, é claro, não sejam somente recebedores passivos de informação, mas possam contribuir e participar da sua API, criticando, sugerindo e elogiando.

São necessárias ferramentas e processos reais para serem usados em uma API, separados em três grandes categorias: Blog, Email e Redes Sociais.

Comunicação com devs e parceiros

Blog




Ter um blog ativo é um portal de entrada para novos devs e é claro, novos negócios. Poucas coisas são tão atrativas quanto um volume de conhecimento e informação gerados em ritmo constante, na raiz do seu site. E atratividade vale tanto para seres humanos quanto para crawlers de SEO (como o Google e o Bing).

Não só isso, mas um blog permite uma forte interação de uma comunidade de devs em torno de uma API. Um blog sem atualizações e sem pessoas envolvidas dá a sensação de estar morto, e a API dará a mesma sensação para alguém que acabou de chegar.



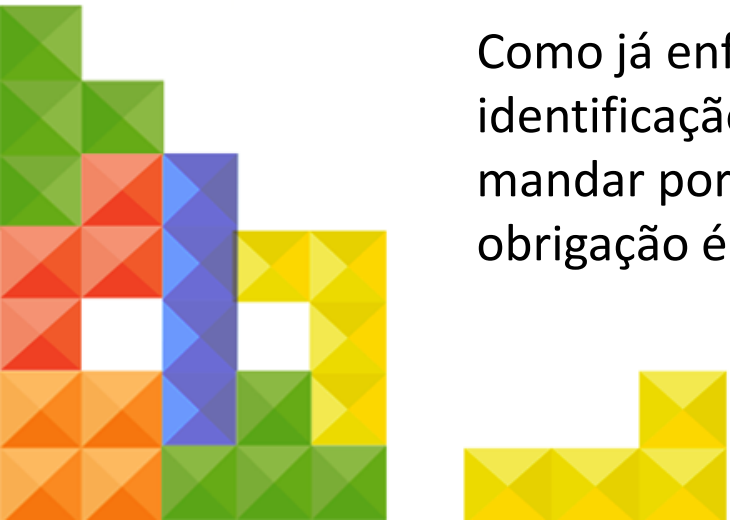
Comunicação com devs e parceiros



Aproveite a existência do seu blog e habilite ferramentas auxiliares como leitura em RSS, inscrição por email e publicações em blogs parceiros. Não fique esperando que todos irão acessar seu site periodicamente e conferir as novidades. Dê o acesso de bandeja e terá mais leitores.

Exemplos: [SoundCloud](#), [8Tracks](#)

Email



Como já enfatizamos na Parte sobre Onboarding, não reinvente a roda: email é a identificação universal das pessoas na Internet. Sua preocupação aqui deve ser o que mandar por email. As taxas de SPAM e emails indesejados é muito alta, então sua obrigação é não entrar nessa estatística.

Comunicação com devs e parceiros

Um dev novo se cadastrou? Que tal mandar sua documentação para ele em um PDF anexo? Péssima ideia (leia o Conjunto de Blocos de Documentação). Emails servem somente para avisos e não para enviar conteúdo que deve ter sua própria página no Portal da API.

Atualizações devem ter seu lugar em uma página do site. Suporte deve ser feito de maneira formal. Sua documentação, como dito, deve estar bem aconchegada na página da API. Cada uma dessas coisas têm o seu lugar na página da API, sendo que o Email é apenas um maneira de informar às pessoas que algo diferente está acontecendo.

É importante notar que o envio de Email segue regras rígidas. Por exemplo, não envie Email para quem não solicitou (não compre listas com Emails), e tenha em todas as suas mensagens um botão que permite ao destinatário optar por não receber mais nada de você. As melhores ferramentas de Email Marketing (como [RDStation](#) e [MailChimp](#)) garantem que isso seja feito.

Comunicação com devs e parceiros

Redes Sociais



Cada rede social tem uma comunidade e um perfil. Mesmo que seja legal ter uma conta em cada uma e republicar conteúdo do seu blog nesses canais de comunicação, melhor mesmo é manter um relacionamento adequado com quem frequenta cada rede.

Assim como no blog, tome cuidado para não deixar alguma conta sem posts por meses. Isso pode dar a impressão de que sua API foi abandonada. E assim como no Email, a comunicação por rede social é informal e não deve substituir seus próprios meios de comunicar e dar Suporte, o que deve ocorrer através do site.

Comunicação com devs e parceiros

Twitter

Bastante usada por devs e útil para manter seus seguidores atualizados a respeito de qualquer novidade. Devido ao fluxo enorme de tweets, uma prática comum é postar o mesmo conteúdo repetidas vezes, em horários e dias diferentes. O principal cuidado é não se tornar alguém chato e acabar manchando sua reputação. Nem tweets demais, nem de menos.

Linkedin

Provavelmente, a maioria de seus devs não virão daqui. Porém, novos e proveitosos negócios podem surgir do Linkedin. A presença da sua API nessa rede social pode abrir as portas para oportunidades que não surgiriam com o fluxo de devs que chega à sua API em meios convencionais.

Comunicação com devs e parceiros

Facebook


Apesar de as publicações orgânicas terem cada vez menos relevância no Facebook, essa rede social tem uma capacidade de prova social muito grande. Isso significa que, mesmo tendo somente 5% dos seus fãs recebendo seus posts na Timeline deles, vale a pena continuar postando e garantir que o número total de Likes aumente e incentive outros a também dar Like.

Google+

Quem usa o Google+? Se a resposta foi ninguém, você está provavelmente errado. É verdade que a rede social tem muitos usuários (mais de 2 bilhões), mas que uma fração mínima dessas pessoas é ativa na rede. Porém, o G+ tem praticamente o mesmo número de usuários ativos que o LinkedIn, e se tornou uma rede de nichos. Hoje é fácil achar comunidades muito aquecidas que tratam de assuntos específicos. Não descarte essa rede antes de ter certeza que seu público não está andando por ali.



Conta de desenvolvedor



Nesse ponto do Ebook, já passou o ponto de convencimento e aclimatação dos devs. Antes, foram citadas formas de informar e educar quem está chegando agora e, não só isso, mas formas eficientes de se comunicar a curto e a longo prazo com seu público alvo.

Se quiser dar mais um passo atrás, saiba que sua [API precisa de um Plano de Marketing](#). A partir de agora, a suposição é que a API já tenha alguns devs cadastrados e interessados em usá-la.

Então, um conjunto de blocos muito rico e interessante é relacionado à Conta do Desenvolvedor. O que seria isso? É simples: quando um novo dev ou parceiro faz cadastro no site da sua API, ele tem a intenção de testar sua API ou até mesmo já tenha planos de uma ou mais apps.

Sua missão então é fornecer as ferramentas mais adequados para que um dev possa gerenciar seus tokens de acesso, apps e o quanto ele consome a API.

Conta de desenvolvedor

Dashboard



Essa é a página inicial da Conta, que os devs cadastrados verão. O mais importante aqui é a usabilidade.

Quais são os botões mais importantes e as informações que podem ser deixadas à mão? Um link para cada um dos blocos descritos abaixo é uma boa forma de começar, mas quais botões terão destaque?

Outra preocupação são as informações disponíveis. Haverá um gráfico de consumo da API? Uma lista com as apps e tokens de acesso?

Lembre-se que o dev não tem que ficar procurando e fuçando a interface. Tudo deve estar à disposição para que ele possa gerenciar o uso da sua API de maneira simples e intuitiva.

Conta de desenvolvedor

Gerenciador de aplicações



É comum que cada dev possua mais de uma aplicação cadastrada na sua API, sendo algumas de testes e outras finais.

Para que ele possa controlar adequadamente seus acessos e chamadas, é bom manter funcionalidades que facilitem o cadastramento de novos apps e obtenção de outros tokens de acesso, assim como ferramentas de gerenciamento de seus projetos com a sua API.

Conta de desenvolvedor

Logs de Consumo e Analytics



Essa aqui é uma das informações mais relevantes para qualquer dev. Qualquer API tem limitação de chamadas. Saber gerenciar quantas chamadas sua app está fazendo é uma das virtudes de um bom dev. Isso vale ainda mais quando sua API é monetizada em cima do volume de chamadas feitas. Nesse caso, vale até avisar o dev (e usar os Blocos de Comunicação) quando ele estiver chegando perto do seu limite.

Portanto, se estiver em dúvida de qual botão colocar em evidência na Dashboard, pense nisso. Melhor ainda: insira gráficos (ou miniaturas) de fácil acesso, deixe os logs e medidores de consumo no topo da lista de prioridades para sua Dashboard.

Conta de desenvolvedor

Configurações de pagamento



Esse aqui só vale para APIs que cobrem algo pelo seu uso, obviamente. Se houver cobrança pelo uso, deixe todas as informações de cobrança simples e cristalinas. Qualquer um deve ter acesso aos seus pagamentos e baixar o histórico e notas dos pagamentos.

Além disso, forneça também ferramentas que possibilitem atualizar os dados de pagamento (número do cartão, telefone, endereço, etc).

Por fim, deixe fácil o acesso às demais opções de faixa de preço, para facilitar o upgrade de uma conta básica para uma Pro, por exemplo. Uma [Landing Page](#) para isso é o ideal, listando os diferentes tipos de pacotes, com seus preços e até um chat ou contatos facilitados do Suporte, de forma a incentivar o upgrade.

Conta de desenvolvedor

Funções básicas de uma Conta



Assim como qualquer Conta, algumas funções são básicas e precisam existir. Se não for fácil atualizar e verificar seus dados, o dev provavelmente irá procurar o Suporte, o que criará demanda desnecessária para sua equipe de Suporte. Veja quais são os blocos de funções:

- 1 - Reset de password;
- 2 - Deletar conta: nesse caso, é importante que o dev tenha opções avançadas em relação às suas apps, histórico de consumo, chamadas, etc. Ou seja, antes de poder deletar a conta, o usuário pode querer ter acesso à detalhes do seu uso da API ou exportar certas informações;



Conta de desenvolvedor

- 3 - Central de comunicações: essa aqui é uma subcomunicação dentre as vistas anteriormente. Você pode usar sempre as comunicações internas, mandando avisos. Porém, não imagine que os devs irão acessar o Portal todos os dias para ler mensagens. De modo geral, qualquer comunicação deve gerar uma notificação por email para aquele usuário (a não ser que ele opte por não receber isso).

Serviços e monetização

Agora, vamos falar de Dinheiro!!!



Você já deve saber que uma API pode ser monetizada de [diversas maneiras diferentes](#). Como o bom negócio que é, ganhar dinheiro com API só depende da imaginação e visão de negócios de cada empresa. Se você não consegue ver com tanta clareza, [leia esse post do blog](#). Eu espero você terminar.

Serviços e monetização

Terminou? Então continuando, é importante enfatizar a existência desse bloco:



Precificação/cobrança

Nem todas as APIs cobram pelo seu uso, mas aquelas que cobram devem ter esse fator cristalino em suas documentações.

No Conjunto de Blocos anterior, foi enfatizada a necessidade de haver configurações de pagamento à disposição do dev. É claro que isso deve vir acompanhado de uma explicação detalhada de quanto o uso é cobrado, quais as condições, pacotes, planos, vantagens de cada um desses planos e assim por diante.

Não deixe essa informação duplicada ou desatualizada. Não deixe que se abram precedentes sobre o uso, ou qualquer forma de gerar conflito com sua comunidade de devs.

Serviços e monetização

Essa é uma questão de confiança, uma vez que os devs estarão criando seus próprios negócios em cima da sua API.

Por fim, caso ocorra qualquer mudança nesses termos, use os modos de comunicação mais adequados, como já foi dito no Conjunto de Comunicação.

Basicamente, é possível listar os blocos de construção da monetização de uma API como:

Receita Direta: a forma mais simples de monetizar, muito utilizada por diversos tipos de SaaS (Software as a Service). Você cria algum tipo de equação, que relacione o volume de chamados à API com o valor a ser pago. Essa estratégia é conhecida como “pay-as-you-go” e é bem simples de entender e implementar. Você pode deixar gratuitos os primeiros meses de uso ou até determinada quantidade de chamados, de modo a incentivar um uso inicial da API. Exemplo: [AWS](#);

Serviços e monetização

Pacotes ou planos: É semelhante à estratégia de Receita Direta, mas com faixas de preço para a quantidade de chamadas. Oferece a vantagem de o desenvolvedor que está usando a API poder usar a vontade (até o limite de seu plano) sem se preocupar em pagar mais por cada chamado feito à API. Exemplos: [Paypal](#), [Sendgrid](#);

Programa de afiliados e marketplaces: Esses programas oferecem recompensas àqueles que enviarem acessos ou clientes à determinado site ou serviço. É uma forma de terceirizar o marketing e os esforços de venda. APIs são úteis para esse modelo porque quem expõe uma API quer que ela seja utilizada e que seu ecossistema cresça. Um ecossistema forte significa mais usuários, mais vendas e mais apps, realimentando o ciclo. É comum que marketplaces usem essa estratégia, com a intenção de ampliar a variedade dos produtos no portal. A API entra para facilitar a integração das lojas com o marketplace. Portanto, aqui está uma forma de dividir os frutos do sucesso da API com a comunidade de Devs/lojas afiliadas. [Veja os posts sobre Marketplaces no Blog da Sensedia](#). Exemplos: [Amazon Associates](#), [Nova Pontocom](#);

Serviços e monetização

Diferenciação e Efeito Cola: Imagine que você já usa a plataforma X como seu ERP. Você sente a necessidade de usar um CRM e um serviço de suporte (*helpdesk*). Pesquisando, você descobre que seu ERP é diferenciado dos demais e é integrado à plataformas Y de CRM e Z de helpdesk. Maravilha! Nada de ficar exportando planilhas de dados ou habituar sua equipe a serviços totalmente novos. Agora, sua equipe usa X, Y e Z e a chance de você passar a usar qualquer outro ERP, CRM ou helpdesk cai vertiginosamente, já que sua equipe está habituada e seus sistemas conversam. Esse é o Efeito Cola. Você pode traçar uma estratégia de parceiros, procurando softwares complementares ao seu, e integrar seus serviços usando APIs! Exemplos: [Salesforce](#), [Slack](#), [IFTTT](#);

Serviços e monetização

Branding e propaganda: provavelmente o modelo mais comum de monetização da sua API é não monetizar ela. Isso parece não fazer sentido nenhum, mas é bem simples: branding. Grandes serviços querem ter cada vez mais usuários, que é de onde normalmente vêm o faturamento através de um outro método não relacionado à API. Populando o serviço e movimentando informação (através da API), o serviço ganha adeptos e, conseqüentemente, mais força. Esse modelo é comum em Redes Sociais. De fato, a API do Twitter gera 75% do fluxo de tweets total. Exemplos: [Games no Facebook](#), [Foursquare](#).

Normalmente, faz mais sentido que você foque em apenas uma das estratégias acima, mas como qualquer bloco de construção que já falamos, a base é semelhante, mas o diferencial está em usar sua criatividade para criar um serviço único.

Ambiente de desenvolvimento

A partir de agora, serão abordados Blocos de Construção mais técnicos. Nos Conjuntos anteriores, foram tratados, principalmente, Marketing, Negócios e Relacionamento.

Como você já deve ter notado, desenvolvedores têm um papel crucial em todas as etapas. Até agora, eles vinham sendo principalmente objetos das ações e escolhas do criador da API.

De agora em diante, eles terão uma relevância mais direta na construção da API, influenciando as principais tecnologias e plataformas presentes na sua API. Você verá como em detalhes.

Para começar o debate sobre os meandros da API, algumas bases de Design de APIs. Lembrando que se você quiser se aprofundar no assunto, a [Sensedia tem alguns conteúdos bem bacanas](#).



Ambiente de desenvolvimento


O que você acharia de algum esporte que realizasse seus jogos e treinos no mesmo dia e local? Seria como alguns pilotos de fórmula 1 fazendo as voltas de teste, outros fazendo as voltas de qualificação e ainda alguns na corrida valendo pontos, tudo ao mesmo tempo e na mesma pista.

Ou dois times de futebol travando uma partida no Maracanã enquanto um terceiro treina e a torcida de um quarto ensaia seus gritos e canções. Você imagina isso acontecendo?

Se não faz sentido imaginar cenários assim, por que algumas apps devem realizar suas chamadas em produção enquanto outros testam e experimentam, tudo no mesmo ambiente?



Ambiente de desenvolvimento



Se algum dev erra a implementação do código de sua app e sobrecarrega a API, todo mundo cai junto? (Se uma única app é capaz de derrubar a sua API inteira, então você tem uma API com falhas de design e ambiente desprotegido. Confira os conteúdos de [Design](#) e [Gerenciamento de APIs](#) para ver como corrigir essas falhas).

Portanto, fica claro que deve haver ambientes em que algumas apps estão sendo testadas e outros em que apps são expostas e podem ser baixadas, usadas e aproveitadas. Veja:


Ambiente de Produção

Por definição, esse ambiente é aquele em que a equipe de desenvolvimento coloca em operação o software, para que seus usuários finais possam desfrutar desse software.

Para sua API, esse é o ambiente em que seus recursos (ou endpoints) estão disponíveis, as apps realizam chamados e os usuários dessas apps acessam sua API.




Ambiente de desenvolvimento



Os usuários, muitas vezes, não tem consciência de que estão acessando uma API, uma vez que a app funciona como interface e abstrai essa relação. Na prática, isso significa que não faz sentido falar em uma API exposta que não tenha um Ambiente de Produção.

Essa é a raiz da exposição. Portanto, toda API, seja aberta ou restrita ao uso de parceiros, tem um Ambiente de Produção.

Sandbox/Ambiente de Testes



Esse é o espaço em que os developers irão brincar com sua API, conhecer os recursos disponíveis, tratar os retornos (e erros), ou seja, colocar em ordem tudo que precisa estar funcionando para o lançamento da app.

Ambiente de desenvolvimento


Praticamente toda app precisa passar por aqui, para que os devs tenham certeza de que está funcionando de acordo com o esperado. Porém, nem toda API tem uma Sandbox, o que é um empecilho à realização dos testes de desenvolvimento e homologação.

Apesar de ser um custo inicial a mais no Design da API, a Sandbox se mostra um ótimo investimento pois reduz o atrito inicial para o desenvolvimento de apps e, é claro, diminui a quantidade de chamados abertos no seu serviço de Suporte. Portanto, ter um Ambiente de Testes na sua API é uma decisão bastante sábia.

Exemplos: [Gengo](#), [eBay](#), [Twilio](#).



Gerenciamento de código



Em Conjunto com as ferramentas da Conta do dev, é importante abordar algumas ferramentas de Gerenciamento de Código. Essas ferramentas, nas mãos de um desenvolvedor, têm um poder de construção incrível!

Git

Uma prática extremamente consolidada entre equipes de desenvolvimento é o uso de uma ferramenta de gerenciamento e manutenção de código Git (com foco em [Github](#) e [BitBucket](#)).

A sugestão é que exista uma conta Git para sua API, onde podem ficar armazenados exemplos de códigos, SDKs e outras ferramentas úteis para desenvolvimento.

Exemplos: [Evernote](#), [Twitter](#), [Facebook](#), [Slack](#), [Box](#)

Gerenciamento de código

Projetos para iniciantes e exemplos de código



Já foi aconselhado que se meça o TTFHW (Time to First Hello World) médio da sua API. Uma das formas de melhorar isso é com exemplos de projetos para iniciantes. Esses projetos permitem que um recém-chegado coloque a mão na massa e tenha uma aplicação funcionando com poucos cliques.


Exemplos: [Office 365](#), [Google Glass](#)

Open Source

Uma API é um pedaço da sua empresa, visível e aberto ao mundo (caso a API seja pública, obviamente).



Gerenciamento de código



Por ter essa natureza expositiva, é sempre importante se preocupar com os aspectos legais de uma API, como a propriedade intelectual envolvida (serão comentados Blocos Jurídicos mais adiante).

Isso é relevante para o dono da API (você), para os devs (criadores de conhecimento em cima da API) e também para os usuários finais.

Uma solução é abrir sua API como Open Source, incentivando a inovação aberta e reduzindo potenciais atritos com devs, relacionados à criações de negócios em cima de APIs.

Gerenciamento de código

Bibliotecas e SDKs (Software Development Kits)



Bibliotecas e SDKs são uma aquisição essencial para qualquer API. Elas permitem que devs aprendam e programem em suas linguagens de programação preferidas.

Esse é outro fator de diferenciação da API, que permite reduzir o tempo de onboarding de devs que estão conhecendo sua API. Você mesmo pode disponibilizar e manter as bibliotecas e SDKs (use a dica do Git, acima, para ter um fácil acesso à tais bibliotecas), ou ainda, deixar que a comunidade de devs crie e compartilhe suas próprias ferramentas, para outras linguagens.



Gerenciamento de código

Se for da sua confiança, contribua com a divulgação dessas bibliotecas e SDKs, e ajude na manutenção. Quanto mais linguagens você disponibilizar, maior será o alcance da sua API.


Priorize as mais populares como Java, PHP, Python, Ruby e .NET

Exemplos de bibliotecas: [Google](#), [Twitter](#), [Plot.ly](#), [Kaltura](#), [CrocoDoc](#)

Exemplos de SDKs: [Facebook](#), [Paypal](#), [Box](#)



Segurança e autenticação



Não é raro ouvir histórias de problemas de segurança em APIs. Alguns simbólicos foram o sofrido pelo Bitly e também pelo [Snapchat, no ano passado](#). Mas como evitar ser a API da vez?

Como seus devs e parceiros podem ter certeza que a API exposta por você é confiável? Afinal, desenvolver uma aplicação sobre a sua API envolve um alto nível de confiança. Sem uma base de segurança sólida, todo o castelo de cartas, que são as apps, cai.

E junto cai sua reputação com a comunidade de devs.

O assunto é extenso e pode ser melhor evidenciado em nosso Webinar de Segurança de APIs. [Confira para conhecer tudo sobre o assunto](#).

Vamos então aos Blocos de Segurança.



Segurança e autenticação

Descrição das medidas de segurança tomadas e formas de autenticação

Por ter características tão únicas e relevantes no escopo de qualquer API, uma parte da sua documentação que explique as medidas de segurança que estão sendo tomadas é algo bem interessante.

Isso também vale para quais protocolos e regras de autenticação (ver mais sobre Autenticação abaixo) estão sendo usadas na API.

Você deve se preocupar com isso porque a relação de confiança precisa ser forte entre a API (você) e as apps (seus devs e usuários finais). Pense na analogia do castelo de cartas: você criaria seu castelo sobre as areias de um deserto?



Segurança e autenticação

Ao mesmo tempo, a contrapartida é interessante, ou seja, o que é esperado de alguém que está usando sua plataforma, em termos de segurança? Deixe isso claro e terá um ambiente mais seguro e saudável.

Exemplos: ver as APIs que estão como exemplo na Autenticação, abaixo.

SSL

SSL é um protocolo de criptografia, que gera uma camada de segurança muito poderosa. Isso permite que os recursos (disponíveis nos endpoints) sejam acessados através de uma comunicação criptografada, em que apenas as partes envolvidas possuem a chave, impossibilitando que pessoas estranhas à comunicação possam ler tais dados.

Exemplos: [Twitter](#), [Flickr](#)

Segurança e autenticação

Autenticação de APIs (OAuth)

Em particular, a autenticação de APIs é uma das principais barreiras contra usuários maliciosos.

De modo geral, uma autenticação permite que determinado dev (ou cada app, se quiser ser mais específico) seja identificado unicamente e em conjunto com um Gateway, que permita controlar o acesso individual de cada parceiro e desenvolvedor aos seus recursos.

Assim, torna-se simples restringir os usuários maliciosos, que estejam consumindo muito mais recursos que o permitido (ou que o necessário).

A seguir, um pouco sobre cada tipo de autenticação.

Segurança e autenticação

Tokens são uma forma bastante simples e interessante de gerar autenticação. Funciona através de uma string de números e letras, normalmente bastante longa e que é passada em todas as requisições à API.

Exemplos: [Slack](#) (que também permite OAuth2), [Pinboard](#), [Librato](#)

O padrão **Basic HTTP** é o que usa apenas usuário e password. Esse não é um modo seguro de autenticação de APIs, então não use. Isso porque essa combinação de usuário e password é menos segura que os tokens e se não for transmitida de forma criptografada (com o SSL, por exemplo), fica aberta para leitura na rede.



Segurança e autenticação

O padrão de autenticação **oAuth2** é o melhor disponível hoje. Apesar de não ser perfeito, ele cria uma camada com responsabilidade do desenvolvedor, outra da API e uma terceiro do próprio usuário final. Isso é feito através de quatro diferente fluxos previstos pelo padrão, para garantir a segurança dos dados.

Por exemplo, pode-se usar o oAuth2 naquela API que pede permissão ao usuário para poder usar a conta dele.

É comum ver isso em APIs do Facebook ou Google, as quais permitem fazer login em serviços de terceiros com suas contas.



Segurança e autenticação

Uma característica bem legal desse protocolo é que há papéis bem definidos (o dono dos recursos, o responsável pela autenticação e a aplicação cliente), além de ser possível delimitar e definir quais recursos ficarão disponíveis ou não para a app.

Por fim, o OAuth2 permite que o dono dos recursos (que normalmente é o usuário final) possa revogar os acessos que ele habilitou a qualquer momento, dando bastante poder aos usuários.

Exemplos: [Coursera](#), [Wordpress](#), [Imgur](#), [GitLab](#), [Gigya](#)

Suporte

Foram citados até agora alguns Conjuntos muito importantes, como **Onboarding, Documentação, Comunicação e Materiais Educativos**.

Isso para que você chegasse aqui e não tivesse surpresas. Caso você tenha prestado atenção, esse Conjunto será uma brisa.

Porém, por melhor que seja tudo isso, você não é perfeito. E portanto, você vai cair em situações em que as **pessoas terão dúvidas** e não saberão como prosseguir.

Respire fundo. Aquela página da documentação que passou por três revisões até ficar maravilhosamente excelente não é à prova de distraídos ou iniciantes. Você fez o seu papel e deu todas as ferramentas que poderia dar. Agora, seu papel é dar **Suporte**.



Suporte

É interessante então dividir o Suporte em duas abordagens. A primeira é **passiva**, em que você dá as ferramentas à comunidade e ela gera conhecimento por si só, de forma *Self-Service*. Idealmente, tendo todas as ferramentas tratadas nesse Ebook, esse tipo de Suporte acontece naturalmente.

Mas não é suficiente. E é por isso que é necessário ter um sistema de *Suporte Direto*, com criação de requisições (ou tickets, ou chamados) para resolução de problemas e dúvidas sanadas.

A segunda abordagem é ativa, em que um time de Customer Success aciona seus parceiros em busca de problemas e dificuldades no uso da API e procura torná-la o mais poderosa possível para a criação de apps.



Suporte

Esse tipo de Suporte é mais comum em APIs totalmente voltadas para parceiros ou ainda em negócios que tem a API como pilar central. Por ser bem específica a certos casos, não será tratada nessa nesse Ebook.

Para a maioria dos casos, a abordagem passiva é suficiente.

Antes de ver os Blocos, tenha em mente que você precisará de uma pessoa (ou algumas, dependendo do tamanho da sua API) dedicada em cuidar do Suporte. A boa notícia é que **se você já se preparou com os demais blocos citados, a equipe de Suporte será bastante reduzida.**

Suporte self-service

Fórum

Esse é o começo de um bom suporte self-service.

Crie um fórum, deixe-o facilmente acessível (com links espalhados pela sua documentação e Conta do Dev) e tenha uma equipe para acompanhar as discussões e popular o fórum com conhecimento.

A medida que os devs forem se ajudando, mais conteúdo e conhecimento é armazenado no fórum e todo mundo ganha! Ajuda muito se você já se adiantar criando uma FAQ antes mesmo das perguntas se tornarem de fato frequentes, como dito no início do Ebook.

Suporte self-service

A desvantagem dessa estratégia é que alguém da equipe técnica precisa estar sempre atento à novidades no fórum e moderá-lo de acordo com tais atualizações. Contudo, essa pode ser a mesma pessoa que irá cuidar do Suporte Direto.

Exemplos: [Extra.com](https://www.extra.com.br/), [Guild Wars 2](https://www.guildwars2.com/), [Google AdWords](https://support.google.com/adwords/), [Vimeo](https://vimeo.com/help), [Shopify](https://shopify.com/help)

Stack Overflow

Mesmo que você tenha as melhores intenções com seu próprio fórum, a comunidade de devs já é estabelecida em torno de alguns sites, como o [Stack Overflow](https://stackoverflow.com/).

Pode ser que devs iniciantes ou que ainda não tiveram contato com seu fórum entrem em sites externos como esse em busca de ajuda vinda de devs mais experientes.

Suporte self-service

O que pode ser melhor do que você responder dúvidas sobre sua API e ainda direcionar essas pessoas para o seu próprio fórum?

Exemplos: [Facebook](#), [Foursquare](#), [SurveyMonkey](#), [Shopify](#), [SoundCloud](#)

Suporte direto

Ticket



Um sistema de tickets para suporte é uma forma bastante eficiente de cuidar do ecossistema da API.

Isso porque a geração de tickets permite um alto nível de organização, e que pode gerar um link direto para o roadmap de melhorias da API.

Use sistemas de Helpdesk consagrados como [Zendesk](#) ou [Freshdesk](#). Ambos, inclusive, tem APIs ótimas, e como deveria se esperar, suas páginas de Suporte estão entre as mais completas.

Exemplos: [Freshdesk](#), [Zendesk](#), [InfusionSoft](#)

Suporte direto

Telefone/Email/Formulário de contato

Esse é um sistema artificial (ou personalizado) de Helpdesk.

Ao invés de criar um ticket, o usuário da API em dúvida manda um email, responde um formulário ou liga para algum telefone. Isso pode ser estendido para a forma de comunicação de sua preferência, como Hangouts ou Skype.

Na prática, pode ser que algumas pessoas prefiram tais métodos de comunicação por facilidade ou conveniência, mas a verdade é que eles não oferecem o mesmo poder de um serviço de Helpdesk completo.

Exemplos: [SurveyMonkey](#), [Twitter](#), [Dropbox](#)

Suporte direto

Suporte pago



Foi falado disso aqui lá nos Conjuntos de Conta do Desenvolvedor e Monetização, como uma forma de trazer recursos à monetização da sua API.

Por exemplo, todos os usuários gratuitos da API tem uma garantia de resposta em até X dias, enquanto que o parceiro (ou usuário que paga alguma taxa para usar a API) recebe respostas às suas dúvidas em questão de horas.


Você pode bolar o esquema que preferir para garantir um acesso preferencial ao suporte.

Exemplos: [Google Cloud](#), [Twilio](#), [Twiddla](#)



Atualizações e Pesquisa

Em qualquer projeto de desenvolvimento de software, o presente não costuma ser o período do tempo que mais preocupa a equipe.



Ao invés disso, **há sempre uma série de funcionalidades previstas e sendo desenvolvidas**, além de uma lista de alterações passadas, mas que podem influenciar fortemente aqueles devs e parceiros que já estejam trabalhando com sua API.

Além disso, há também aquelas funcionalidades não previstas, ou fortemente pedidas pela comunidade.




Nesses aspectos, uma API não difere tanto de outros tipos de software, uma vez que ela também precisa de atualizações e novidades para manter sempre fresco o interesse da comunidade e acompanhar a evolução do seu produto.

Vamos aos blocos.



Atualizações e Pesquisa



Roadmap e Log de mudanças e alterações



Alguém uma vez disse que “Não planejar é se planejar para o fracasso”. É bem provável que você tenha um planejamento bem definido para sua API.


Porém, outra história diferente é o que você fala para a comunidade. E você deve se preocupar com isso, uma vez que a relação de confiança entre você e os devs só tende a aumentar se você divulgar atualizações futuras da API.

Porém, não fale tudo que está vindo por aí! Você deve encontrar um equilíbrio entre uma quantidade boa de informação para a comunidade, mas sem excessos para não alertar seus competidores e também não estragar a surpresa.





Atualizações e Pesquisa



Ao mesmo tempo, outra característica bem interessante de toda API são as mudanças passadas. É possível que nem todos os devs estejam ligados nas mudanças futuras, mas eles com certeza vão querer saber em detalhes o que mudou e como isso influencia suas próprias aplicações.

Exemplos: [Box](#), [Vimeo](#), [Zendesk](#), [Facebook](#)

APIs experimentais, feedbacks e sugestões

Está desenvolvendo sua API e ela ainda não está 100% do jeito que você queria? Então uma boa ideia é já expor sua API como Beta ou com métodos experimentais.

Isso será positivo para seu Roadmap, uma vez que abrir sua API prematuramente para a comunidade pode gerar excelentes feedbacks e sugestões, mudando a sua percepção da API para algo mais próximo ao que a comunidade está esperando e pedindo.



Atualizações e Pesquisa

Caso exponha uma API incompleta ou com bugs, deixe claro que ela está em fase experimental e de testes. Além disso, crie uma página de feedback, e não tenha vergonha de pedir as opiniões da comunidade.

Quem está usando sua API e encontrar algum erro com certeza ficará feliz em apontá-lo.

Exemplos: [Chrome](#), [Sony Mobile](#), [Indivo](#), [Gnome](#), [Energy Information Administration](#)

Status

“Meu programa está quebrado?”, pensa o dev, que desesperado, imediatamente começa a mexer no código e fuçar páginas de documentação. Em nenhum lugar, contudo, você avisou o dev que a API esteve com problemas por cerca de uma hora, e que já está tudo resolvido.

Atualizações e Pesquisa

Enquanto isso, o dev já refez funções, já contatou colegas e avisou sua base de usuários que sua app estará indisponível por algum tempo. Tudo isso teria sido resolvido com uma página de Status da API, explicando o erro e sua causa.


Essa é uma boa prática para não pegar a comunidade de surpresa e ainda mais, mostrar que sua API fica firme e forte no ar pelo máximo de tempo possível. Não deixe sua comunidade na mão! ;)

Em particular, o [Zapier tem uma ferramenta bem legal](#) que permite alertar qualquer problema em qualquer uma de suas centenas de APIs integradas. Confira o link abaixo.

Exemplos: [Unwired](#), [Riot Games](#) (é necessário fazer login), [Twitter](#), [Github](#), [Heroku](#)




Jurídico



Outra parte vital da Documentação da sua API, mas que não será tratada por devs, gerentes ou marqueteiros é Jurídico, ou seja, quais são as implicações do uso da sua API e regras gerais para manter seguro o conhecimento produzido pelos seus usuários e pela sua comunidade.


No Conjunto de Onboarding, foi comentado sobre uma Introdução aos Termos Legais. De modo geral, essa página simples, com os pontos mais importantes, deve existir.

Porém, uma API tem seu poder na **criação** de conteúdo, código e algoritmos. Portanto, informação rica sobre como proteger isso deve existir, com acesso facilitado, para que qualquer dev saiba se seu trabalho está quebrando alguma regra e que direitos ele tem sobre suas próprias contribuições.






Jurídico



Essa é uma questão básica de confiança entre a comunidade de devs e o dono da API. Obviamente, cada API é um caso diferente. Os Blocos que virão a seguir ajudam a definir as categorias mais importantes no assunto.

Fique atento ao fato de que **você precisa de um advogado** ou alguém que trate desses assuntos profissionalmente. Quando se trata de leis e regulamentações, amadorismo pode ser bastante prejudicial para o seu negócio.

Termos de Uso e Serviço




Já viu aquela parede de texto que muita gente clica "Li e concordo", sem nem bater o olho na primeira palavra? Cada API, software, site e serviço tem suas definições de Termos de Uso.



Jurídico

Infelizmente, esses termos costumam ser longos e bem complexos (e é exatamente por isso que é recomendada a presença de um advogado na confecção desses termos).



Mesmo sendo maçantes, é aqui que você irá proteger a si mesmo de usos abusivos da API, além de definir os direitos da comunidade de devs.

Por exemplo, as taxas limite de uso da API são um Bloco que te protegem de alguém que está consumindo dados excessivamente, em detrimento de outras apps.

Há inúmeros outros tipos de barreiras que devem estar presentes nos Termos de Uso da sua API. Leia os exemplos abaixo e consulte advogados especialistas em software.



Jurídico

Uma forma comum (e menos chata) de delimitar direitos e obrigações dos devs é listar tudo que pode ou não pode ser feito com a API. Veja abaixo o exemplo do LinkedIn, que é bastante completo.

Exemplos: [Foursquare](#), [LinkedIn](#), [Google](#)


Regulamentação financeira

Há dois aspectos diferentes aqui. O primeiro, caso sua API seja paga, é delimitar bem cada pacote ou faixa de preço para usufruir da API. Isso já foi comentado antes no Ebook no Conjunto de Monetização.

Já a outra parte é aquela referente a monetização de apps sobre sua API. Ou seja, de que forma alguém pode ganhar dinheiro usando a sua API. É comum que essa parte seja uma subseção dos Termos de Uso.



Jurídico



Algumas APIs permitem que apps monetizem, enquanto outras se reservam ao direito de taxar os que tiverem intenções comerciais. Ainda há os casos que proíbem qualquer monetização, principalmente se sua app executar funções que gerem concorrência direta para os donos da API.

Exemplos: [Foursquare \(parte V\)](#), [Instagram \(parte 5\)](#), [Linkedin \(seção 1.4\)](#), [Google \(seção 9.c\)](#)

Política de Privacidade

A Política de Privacidade serve para proteger a informação. Por tabela, as pessoas que estão usando o serviço e depositam dados nele também tem sua privacidade resguardada.

Jurídico

Isso ainda vale para parceiros e desenvolvedores, que estão contribuindo para seu crescimento através da criação de apps e atração de novos usuários.

É comum que essa política seja geral para os dados, e não específica para a API, ou seja, se aplica para toda e qualquer informação mantida pelo provedor do serviço.

Você deve se atentar para não criar limitações em excesso para os devs, de forma a buscar um equilíbrio entre os dados expostos através da API (e a capacidade de criar inovação pela comunidade) e a segurança dos usuários.

A API do Twitter tem até [um método que retorna a Política de Privacidade deles](#)! Muito legal.

Exemplos: [StackExchange](#), [API2Cart](#), [Jawbone](#)

Jurídico

Direitos de imagem da marca e atribuição



Toda provedora de API também tem interesse que sua marca e/ou serviço sejam reconhecidos. É claro que isso não deve ser feito de qualquer jeito.

Uma série de regras de como usar a imagem, logo, fontes e fundos devem ser seguidas para que terceiros possam se adequar à exposição daquela marca.

Além da parte visual, algumas APIs também definem certos **direitos de atribuição**. Isso significa que certos dados expostos pelos devs (através da API) devem ser atribuídos ao seus donos, e ao mesmo tempo, não se pode declarar afiliação aos expositores da API.

Jurídico

Esses direitos estão [categorizados pela Creative Commons](#), e conforme essa organização, podem ser especificados pelo autor (no caso, o dono da API).

Exemplos: [Foursquare \(trademark\)](#), [Foursquare \(atribuição\)](#), [Stack Exchange](#), [Citrix](#)

Direitos sobre o código

Tanto o código da sua API quanto o código gerado pela comunidade devem ser protegidos. Muitas APIs expõem seu código através de **Licença de Software Livre**. O Github é o principal repositório para isso e contém uma [descrição bastante instrutiva](#) sobre esse tipo de licença.

Também há casos em que a API usa bibliotecas externas, mantidas sob Licenças específicas que são gratuitas, mas pedem reconhecimento pela propriedade intelectual. O Instagram faz isso bem claramente (ver abaixo).

Jurídico

No Conjunto de Gerenciamento de Código, também foram citados SDKs e Bibliotecas criadas por terceiro para usar a API em linguagens não suportadas pelo dono dela. Muitas vezes esses softwares também têm suas próprias licenças de uso, então fique atento para manter essa informação acessível.

Exemplos: [Docker](#), [Twilio](#), [Instagram](#)



Conclusão

Esse Ebook contou com quase 50 Blocos de Construção, espalhados em diferentes categorias, com diferentes profissionais envolvidos.

Porém, o objetivo disso é único: garantir que a exposição da sua API ocorra da melhor forma possível. Mesmo abrangente, o conteúdo exposto aqui não é tudo que se pode falar de uma API.

Mas não há dúvida que se sua API tiver tudo que está descrito aqui, você terá bem menos surpresas do que teria sem tal conhecimento.

Obrigado!