Design de API

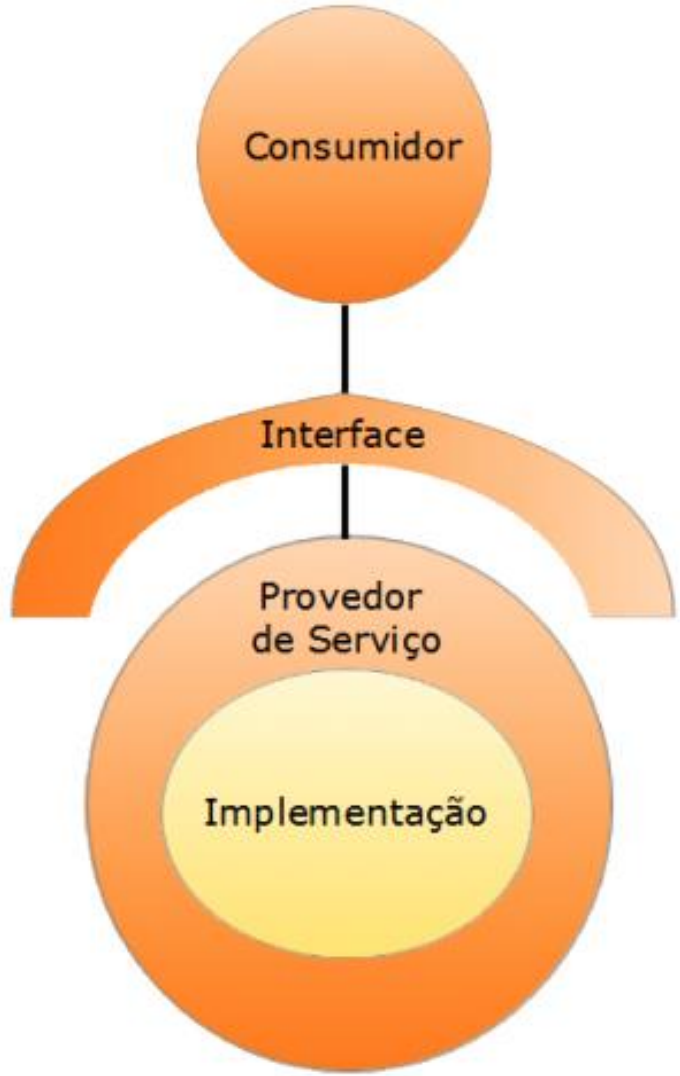
Ricardo Sabatine

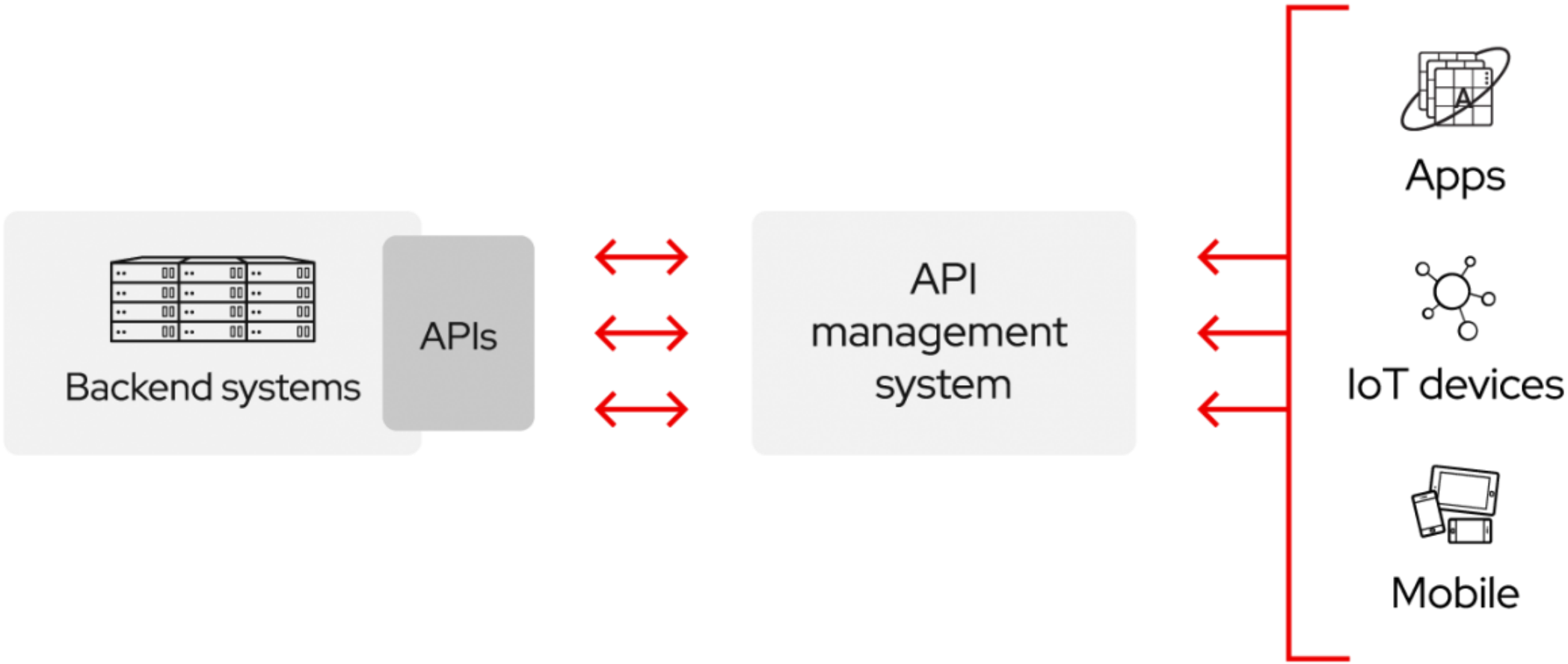
Cronograma

- Introdução
- Tipos de abordagens e estratégias
- API Mock
- Ferramentas de design de api
- Tipos de APIs
  - GRPC
  - REST
  - RESTFUL
  - GRAPHQL

A ideia das **APIs** é prover um mecanismo **simples**, **seguro** com baixo **acoplamento**, **padronizado** e **interoperável** para um desenvolvedor construir suas apps comunicando com um backend no **menor tempo possível**.

- Composição




Backend systems
APIs
API management system
Apps
IoT devices
Mobile

Introdução

- Um bom design de API precisa levar, principalmente, em consideração o conceito de EMPATIA

Introdução

- Com a popularidade do desenvolvimento de APIs, surgiu uma grande necessidade no mercado: a integração entre APIs e, consequentemente, a manutenção e a propagação dos contratos envolvidos entre as partes.

- Chamamos de contrato a comunicação realizada entre duas APIs, onde os dois lados têm acordos de produção e consumo dos dados.

Introdução

- As APIs são importantes para as empresas modernas
  pois proporcionam novos recursos para todas as áreas, de operações
  e produtos a estratégias de parceria.

- Podemos dizer que hoje, a maioria das empresas já nem pergunta se
  deve participar de programas de APIs, mas como fazê-lo.

Introdução

- Os objetivos de negócios mais comuns quando as empresas decidem criar uma API são:
  - desenvolver novas parcerias,
  - aumentar a receita,
  - explorar novos modelos de negócios,
  - acelerar o time to market e estabelecer novos canais de distribuição.

- Os principais objetivos tecnológicos são aprimorar a integração de aplicações e de recursos mobile e oferecer suporte à conexão com mais dispositivos.

- Os benefícios precisam ser sólidos o suficiente para que o investimento em APIs seja uma escolha óbvia para a empresa.

Introdução

- A segunda pergunta é "Que resultados tangíveis queremos alcançar com as APIs?".

- Em outras palavras, "O que as APIs realmente fazem e como elas afetam a estratégia geral de negócios?".

Introdução

- A última pergunta, "Como projetamos o nosso programa de APIs para alcançar os resultados desejados?", refere-se à implementação e execução.
- As equipes precisam se perguntar:
  - Que tecnologia será usada para criar as APIs?
  - Como as APIs serão projetadas?
  - Como será a manutenção das APIs?
  - Como as APIs serão divulgadas dentro e fora da empresa?
  - Quais os recursos disponíveis?
  - Quem precisa estar na equipe?
  - Como monitoraremos o sucesso com base nos objetivos de negócios?

Introdução

- Como John Musser destacou em sua palestra na convenção O'Reilly Open Source de 2012, para que uma API seja ótima ela precisa:
  - Fornecer um serviço valioso
  - Ter um plano e um modelo de negócios
  - Ser simples, flexível e fácil de adotar
  - Ser gerenciada e mensurada
  - Oferecer um excelente suporte aos desenvolvedores

Introdução

- Para definir o valor do seu programa de APIs, considere estas perguntas:

  - **Quem é o usuário?** Para responder, leve em consideração a relação dos usuários com você (são clientes atuais, parceiros ou desenvolvedores externos?), a função que eles desempenham (são cientistas de dados, desenvolvedores mobile ou funcionários operacionais?) e seus requisitos e preferências.

  - **Que problemas do usuário estamos resolvendo e/ou que benefícios vamos gerar para ele?** Considere os negócios do cliente, os desafios e os ganhos definidos na proposta de valor, se alguma necessidade crítica está sendo atendida (é um problema ou oportunidade de gerar lucros?) e que métricas serão aprimoradas (velocidade, receita, economia de custo ou implantação de inovações).

Introdução

- Para definir o valor do seu programa de APIs, considere estas perguntas:
  - **Como o valor para usuário será ampliado ao longo do tempo?** Elabore a proposta de valor considerando mudanças futuras. Quais são os seus principais marcos de projeto futuros relacionados a mudanças externas e internas?
  - **Qual é o valor gerado internamente para a sua organização?** Considere os benefícios internos e como a API pode ser valiosa para os negócios.

Introdução

Quando você conseguir consolida e resumir todas as suas afirmações em uma única frase, terá definido a proposta de valor da sua APIs.

- Nossa API de mensageria oferece aos desenvolvedores corporativos uma funcionalidade de mensagens de texto confiável, com garantia e sem latência para aplicações empresariais críticas. A API também conta com o suporte de kits de desenvolvimento de software (SDKs) que abrangem a maioria das linguagens de programação para uma integração mais rápida.

Introdução

- Um bom design de API segue alguns princípios básicos, que podem ser implementados de formas diferentes.
- **Mudar software é complicado**
- A simplicidade do design da API depende do contexto.
- Um determinado design pode ser simples em um caso de uso, mas muito complexo em outro.
  - Por isso, é necessário equilibrar o nível de detalhes dos métodos de API.

Introdução

- **Formato dos dados** Suporte a XML, JSON, formatos proprietários ou uma combinação dessas opções.
- **Estrutura dos métodos** Os métodos podem ser muito genéricos, retornando um amplo conjunto de dados, ou bastante específicos para permitir solicitações segmentadas. Eles geralmente são chamados em uma determinada sequência para alcançar certos casos de uso.
- **Modelo de dados** O modelo de dados subjacente pode ser muito parecido com o que realmente é exposto por meio da API ou bastante diferente disso. Isso afeta a usabilidade e de manutenção.
- **Autenticação** Mecanismos de autenticação distintos têm pontos fortes e fracos diferentes. A opção mais adequada depende do contexto.
- **Políticas de uso** Precisa ser fácil entender e trabalhar com os direitos e as cotas dos desenvolvedores.

Introdução

- Para planejar o design da API, considere algumas perguntas a seguir:

- **Temos motivo para usar uma API RESTful?**As APIs RESTful estão na moda, mas você não deve seguir essa tendência apenas por isso. Elas se aplicam muito bem a alguns casos de uso, mas há outras APIs que favorecem diferentes estilos de arquitetura, como a GraphQL.

- **Estamos expondo nosso modelo de dados sem pensar nos casos de uso?** A API precisa ter o suporte de uma camada que faça abstrações a partir do modelo de dados real. Como regra geral, não crie uma API que vá diretamente ao banco de dados (embora haja casos em que isso seja necessário).

Introdução

- Para planejar o design da API, considere algumas perguntas a seguir:

- **Quais as regiões geográficas mais importantes? Planejamos nossos datacenters de acordo com elas?**O design da API também precisa abranger elementos não funcionais, como latência e disponibilidade. Escolha datacenters que tenham uma localização geográfica próxima de onde a maioria dos usuários está.

- **Estamos sincronizando o design da API com nossas outras soluções?** Se a API não for a única solução da sua empresa, garanta que o design dela seja consistente com o das suas outras soluções. Mas também pode ser que você decida criar um design totalmente do das API das outras soluções. Se esse for o caso, os planos precisam ser bem definidos e relatados interna e externamente.

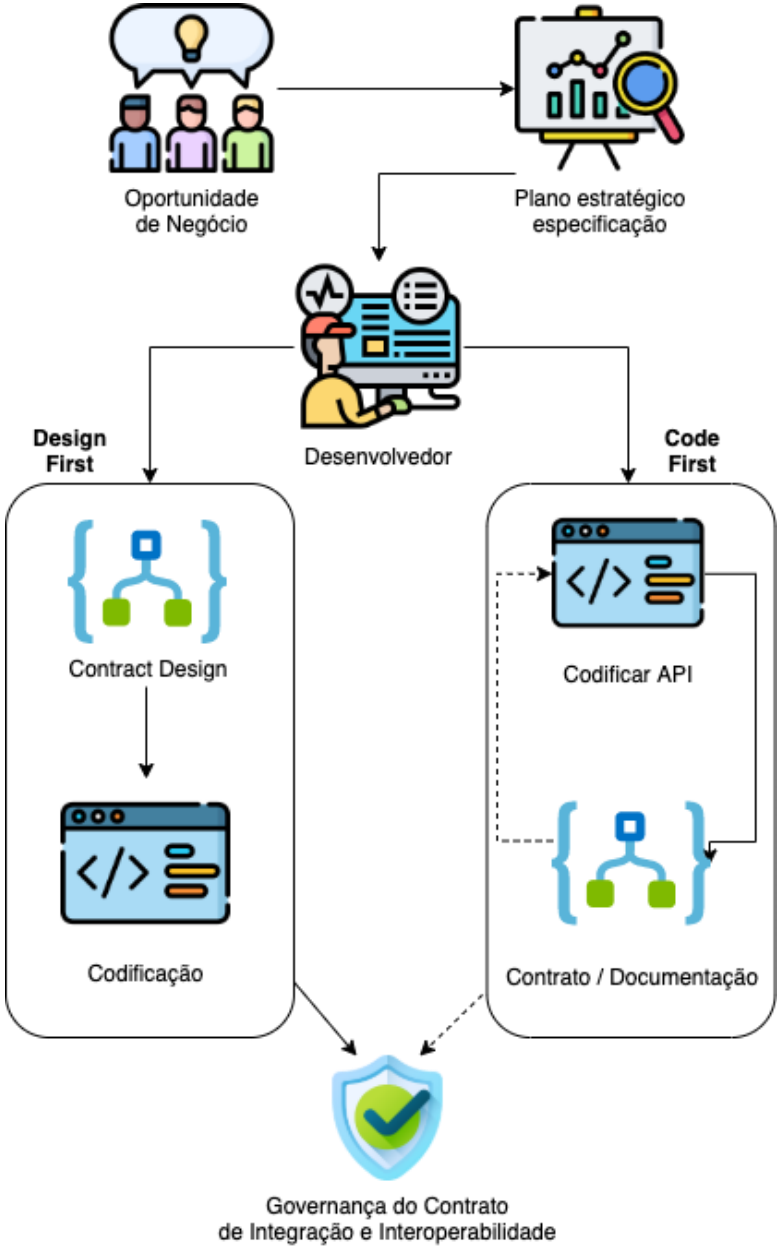
Abordagens sobre Desenvolvimento de API

- Design First vs Code First

  - **Primeiro o Código (Code First)**: Com base no plano da regra de negócio definida, a API é codificada diretamente pela equipe de desenvolvimento e, a partir do código, é criada a documentação de contrato que deve ser legível por humanos e máquinas, como um documento Swagger/OpenAPI.

  - **Primeiro o Design (Design First):** A regra ou plano de negócios é primeiramente convertida em um Contrato legível por humanos e máquinas e descrita em um formato aberto, como o próprio Swagger e OpenAPI. A partir deste Contrato, são gerados os códigos de Provider e Consumer (Server e Client).

Abordagens sobre Desenvolvimento de API



Abordagens sobre Desenvolvimento de API

- Para a escolha da abordagem correta, pergunte-se sempre:
  - Quem são os consumidores finais da sua API? Internos ou Externos?
  - Devo me preocupar com seguir padrões abertos?
  - Que necessidades ou problemas meus consumidores têm em outras integrações?
  - Qual problema sua API está resolvendo para o Consumidor?

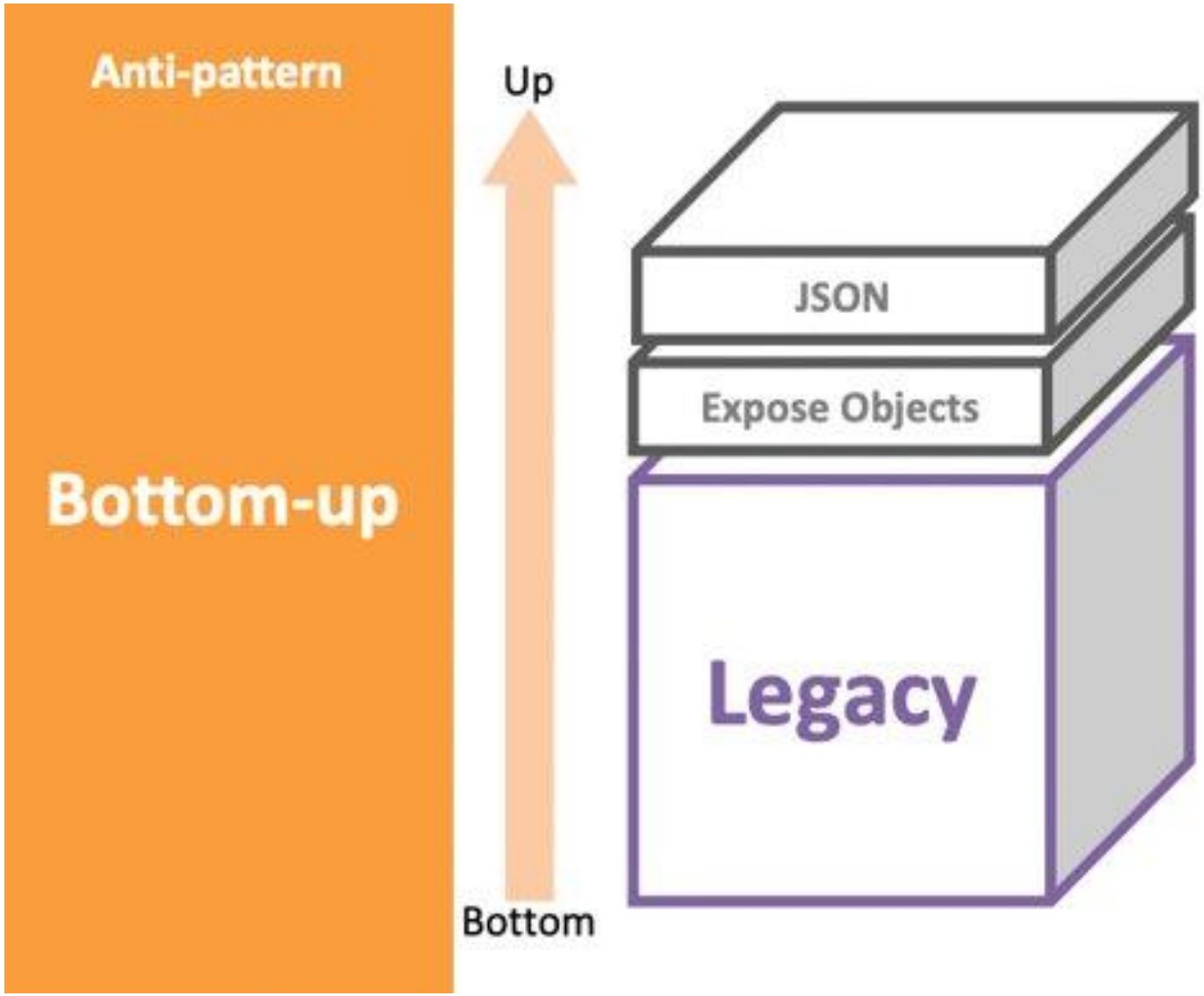
Estratégias de Exposição

Estratégias sem impacto no backend para a exposição de APIs

Estigmas

- Aplicações monílitica nem sempre modularizadas, problemáticas
- Baixa capacidade de conectividade
- Ciclos de entrega longos (meses)
- Dificuldade para evoluir e compor novas tecnologias
- Obsolecência tecnológica ou tecnologias zumbis

O Anti-Padrão: A estratégia de baixo para cima



Vantagens

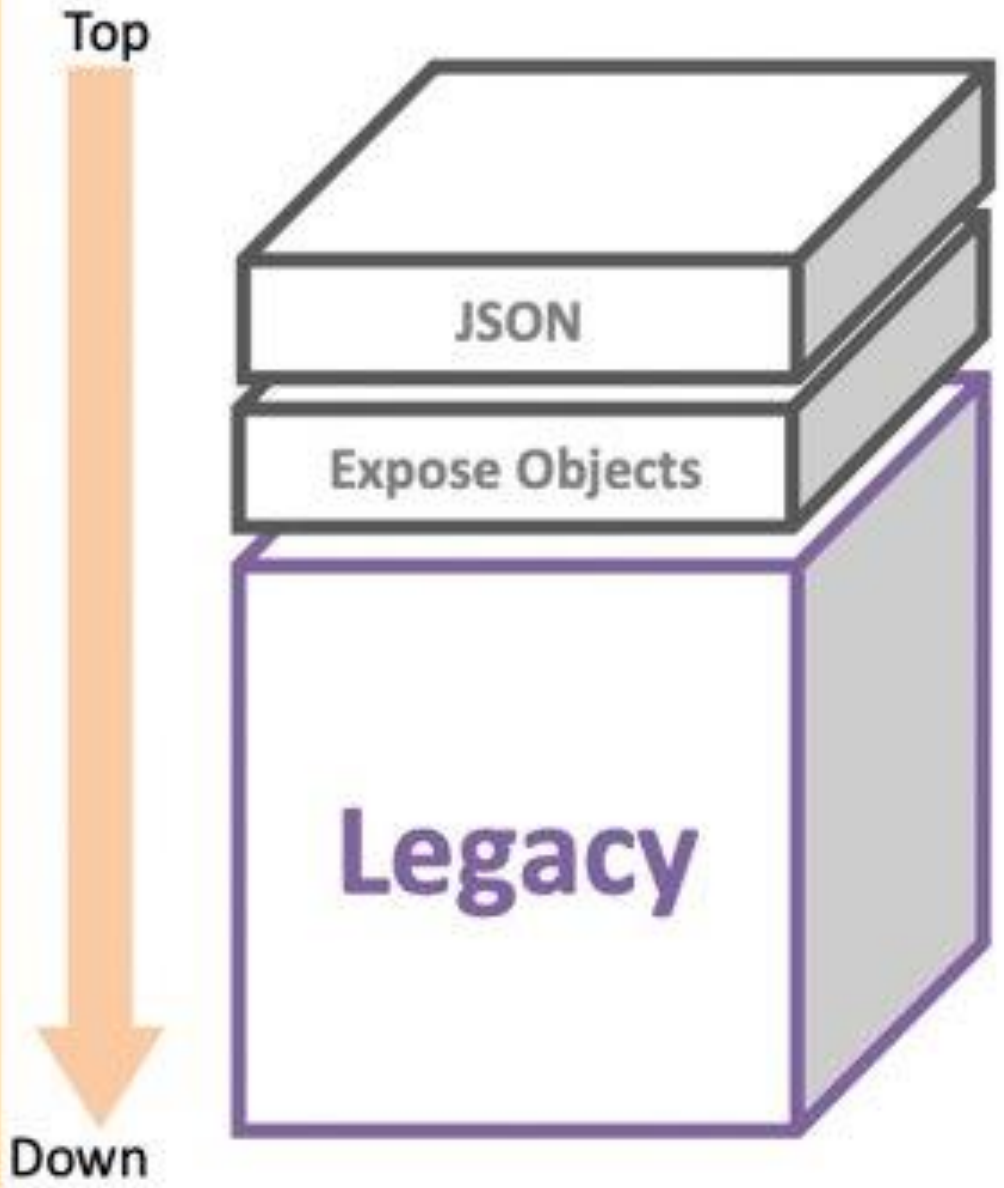
- Rápido
- Ferramentas podem ajudar
- O Arquiteto do legado entende a interface

Riscos

- Design ruim com vícios do legado

Pattern

API-Fist



Vantagens

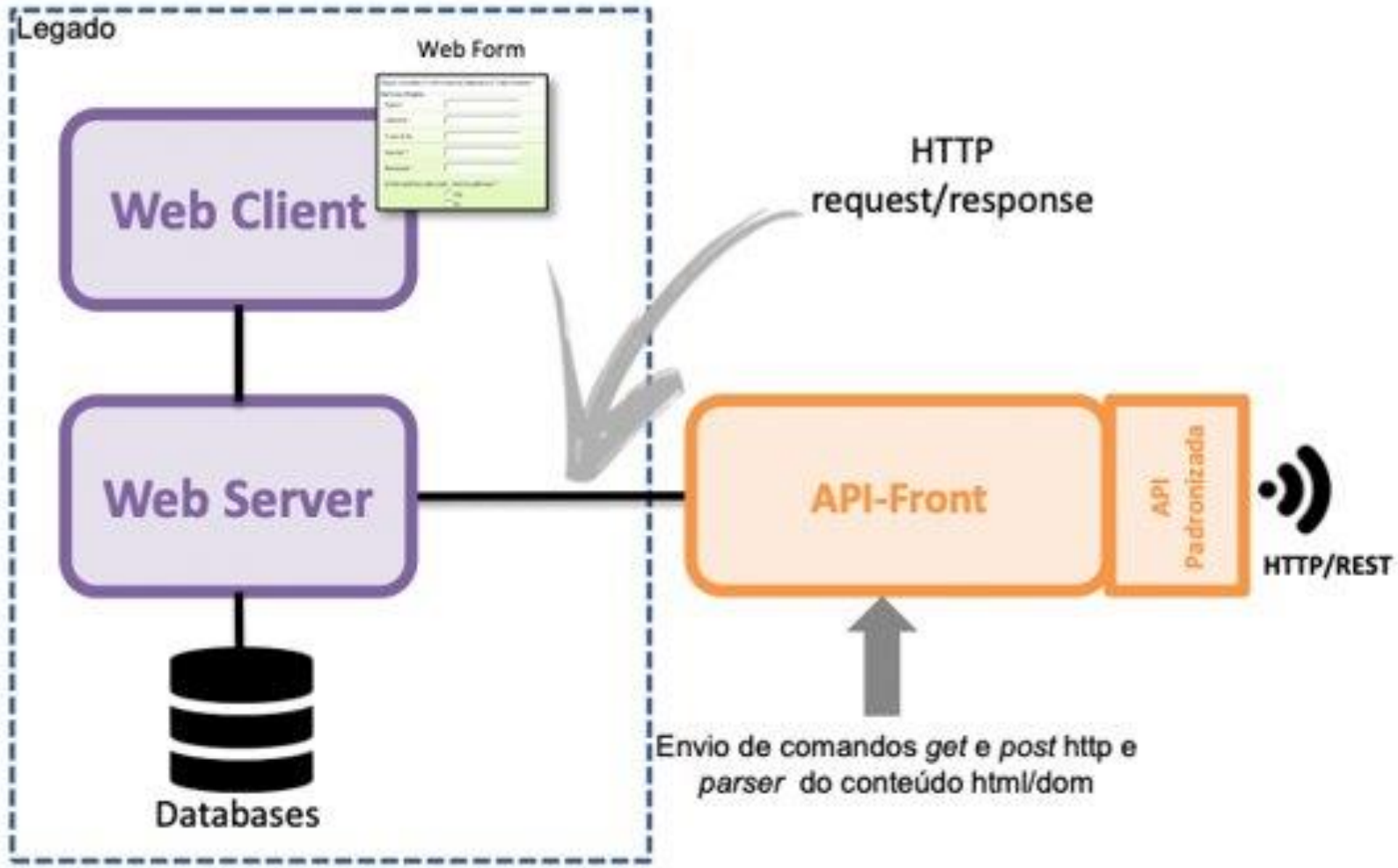
- Usabilidade da API
- Abstração

Riscos

- Entender o domínio
- Os vícios do legado são resistentes

Arquitetura Web (Thin Client)

Web Scraping ou Web Harvesting



Arquitetura Web (Thin Client)

Web Scraping ou Web Harvesting

Vantagens

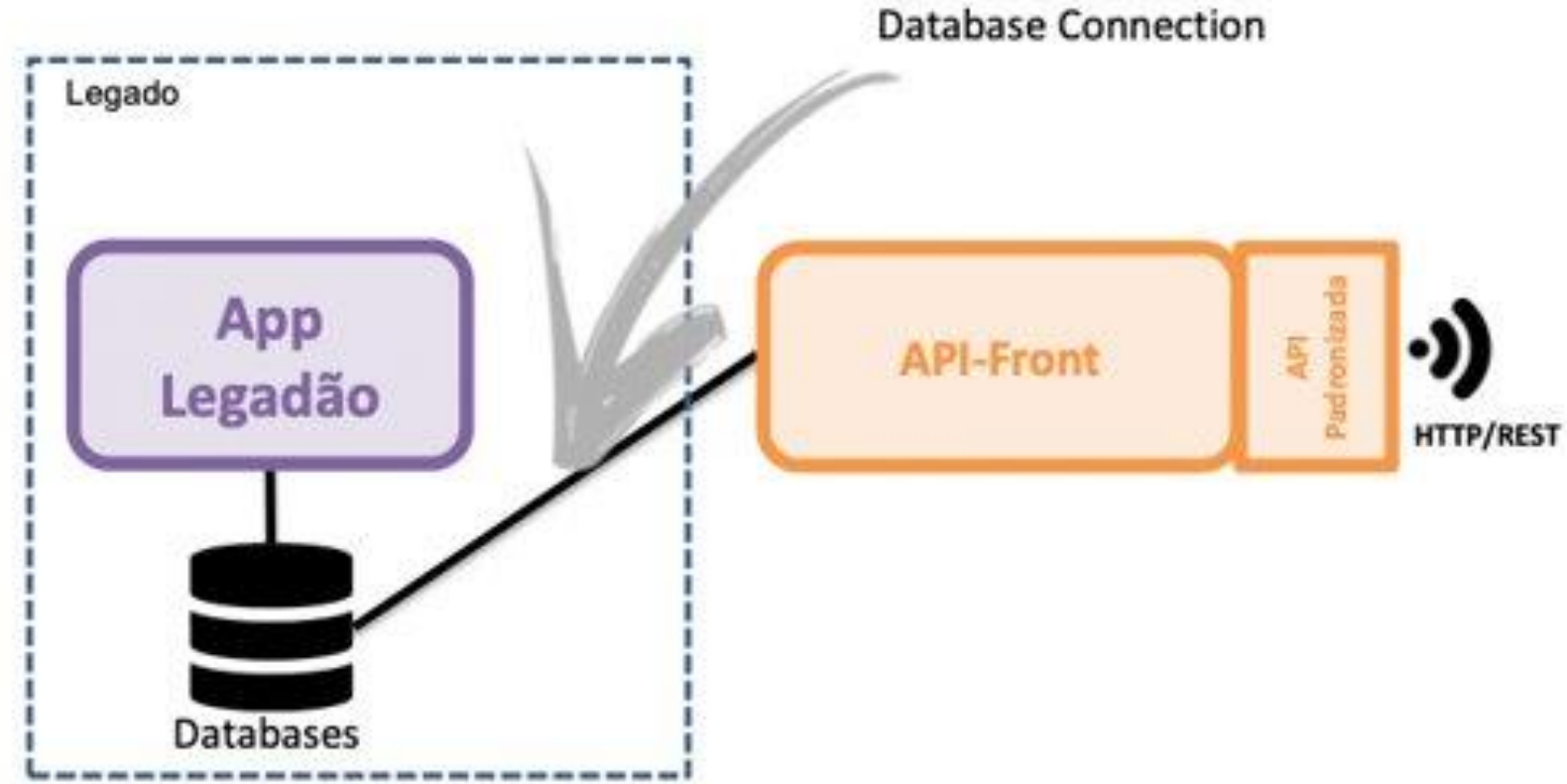
- Uso das funções (camada) de negócio da aplicação
- Uso do próprio protocolo HTTP
- Não requer alterações de código no lado da aplicação
- Interessante para um MVP

Riscos

- Dificuldade de implementação em HTMLs/DOM malformados
- Alterações no HTML/DOM tendem a quebra o código de Scraping
- Questões legais de direitos autorais
- Dados da aplicação que não são expostos em uma View.

Arquitetura Web, Client-Server, Database-Centric, Monolitíca, Mainframe...

Acesso a Camada de Dados



Arquitetura Web, Client-Server, Database-Centric, Monolítica, Mainframe...

Acesso a Camada de Dados

Vantagens

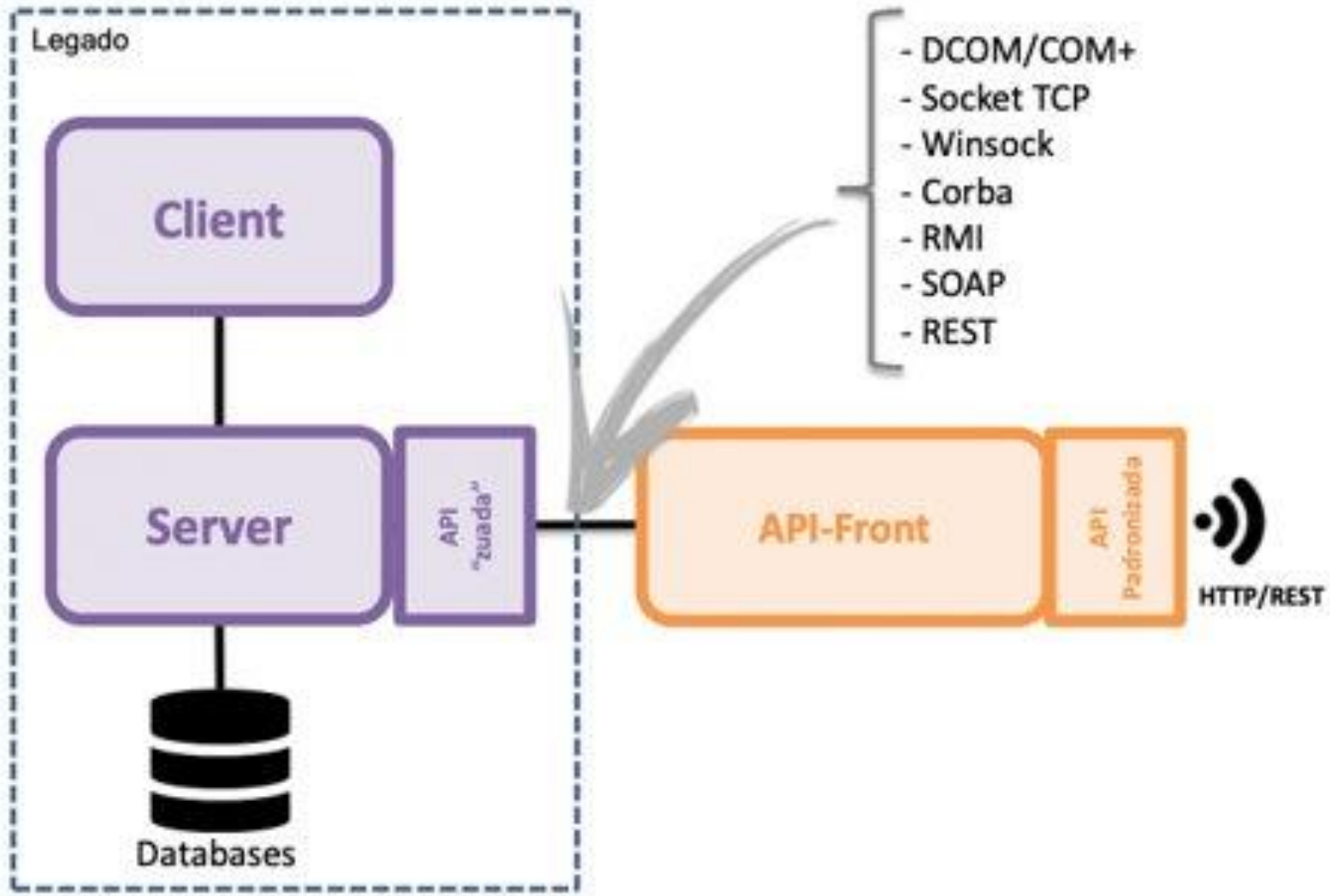
- Não requer alterações de código no lado da aplicação
- Vai direto ao ponto

Riscos

- Não há reaproveitamento de regras de negócio, exceto se as regras estiverem em Stored Procedures
- Pode haver a necessidade de reimplementar algumas regras de negócio
- API-Front tende a ficar complexo e com baixa coesão

Arquitetura Web ou Client-Server (Thin Client)

Acesso a Camada de Serviços/APIs

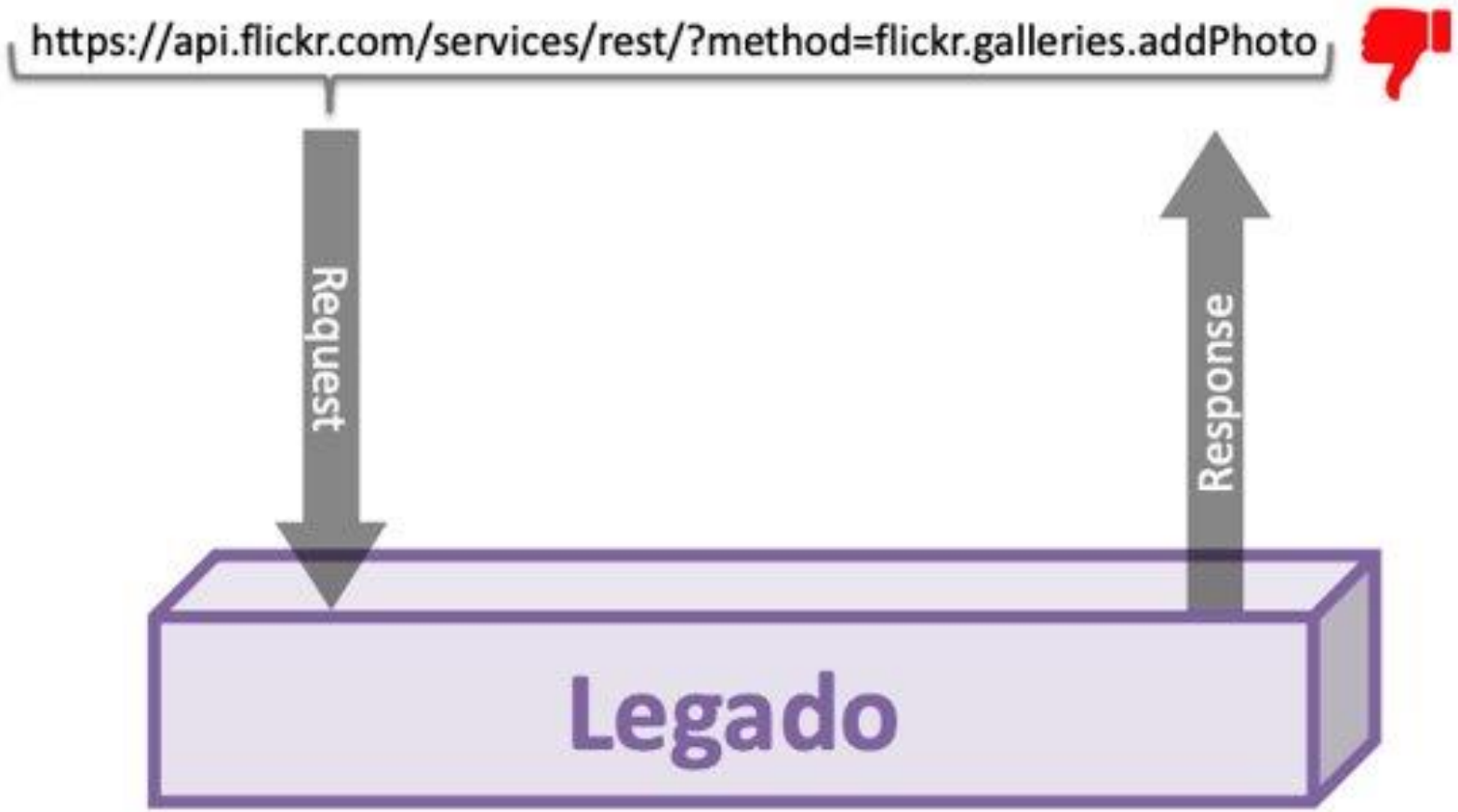


Arquitetura Web
ou Client-Server
(Thin Client)

Acesso a Camada de Serviços/APIs

Padrões Comuns

URLs

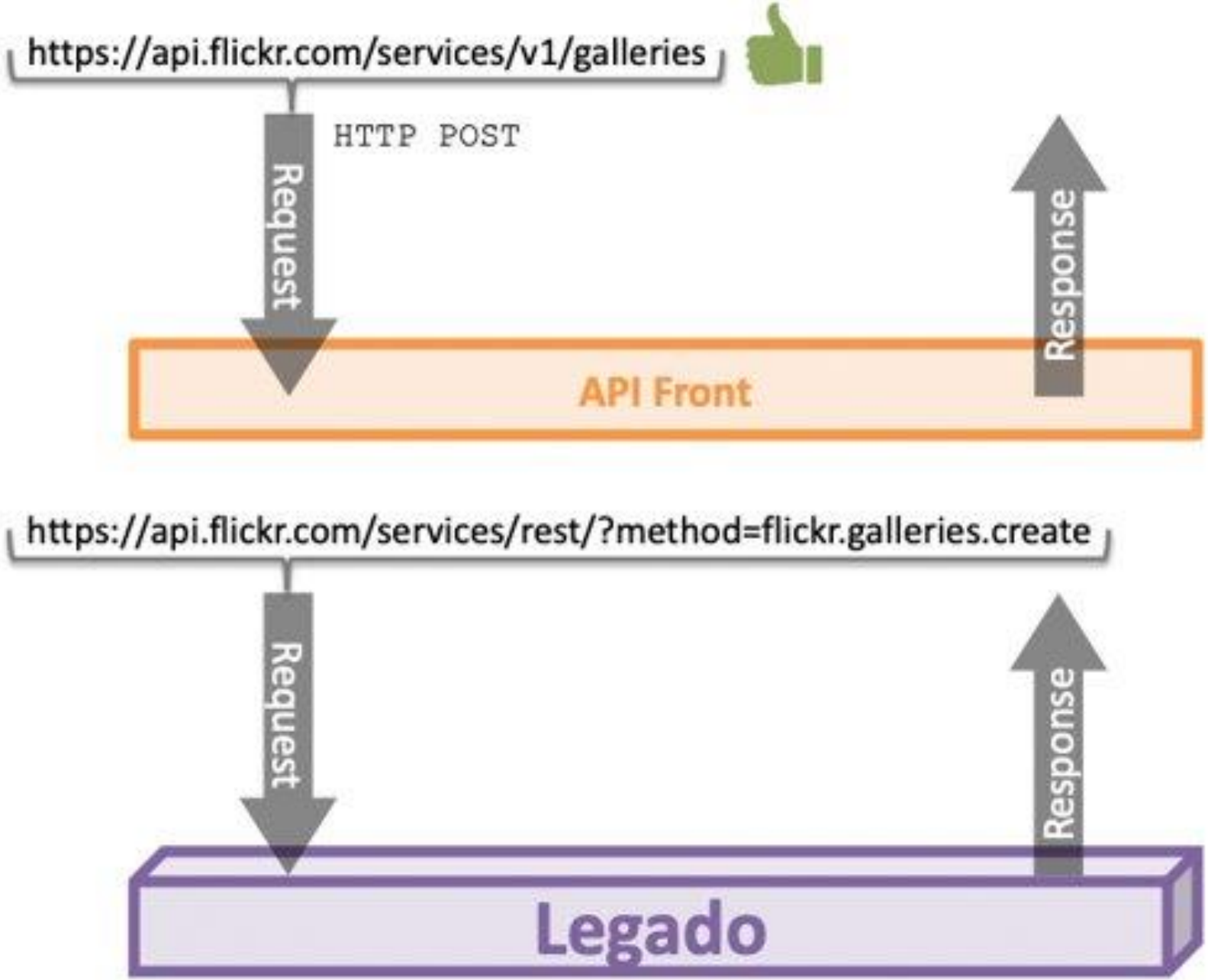


Arquitetura Web
ou Client-Server
(Thin Client)

Acesso a Camada de Serviços/APIs

Padrões Comuns

URLs



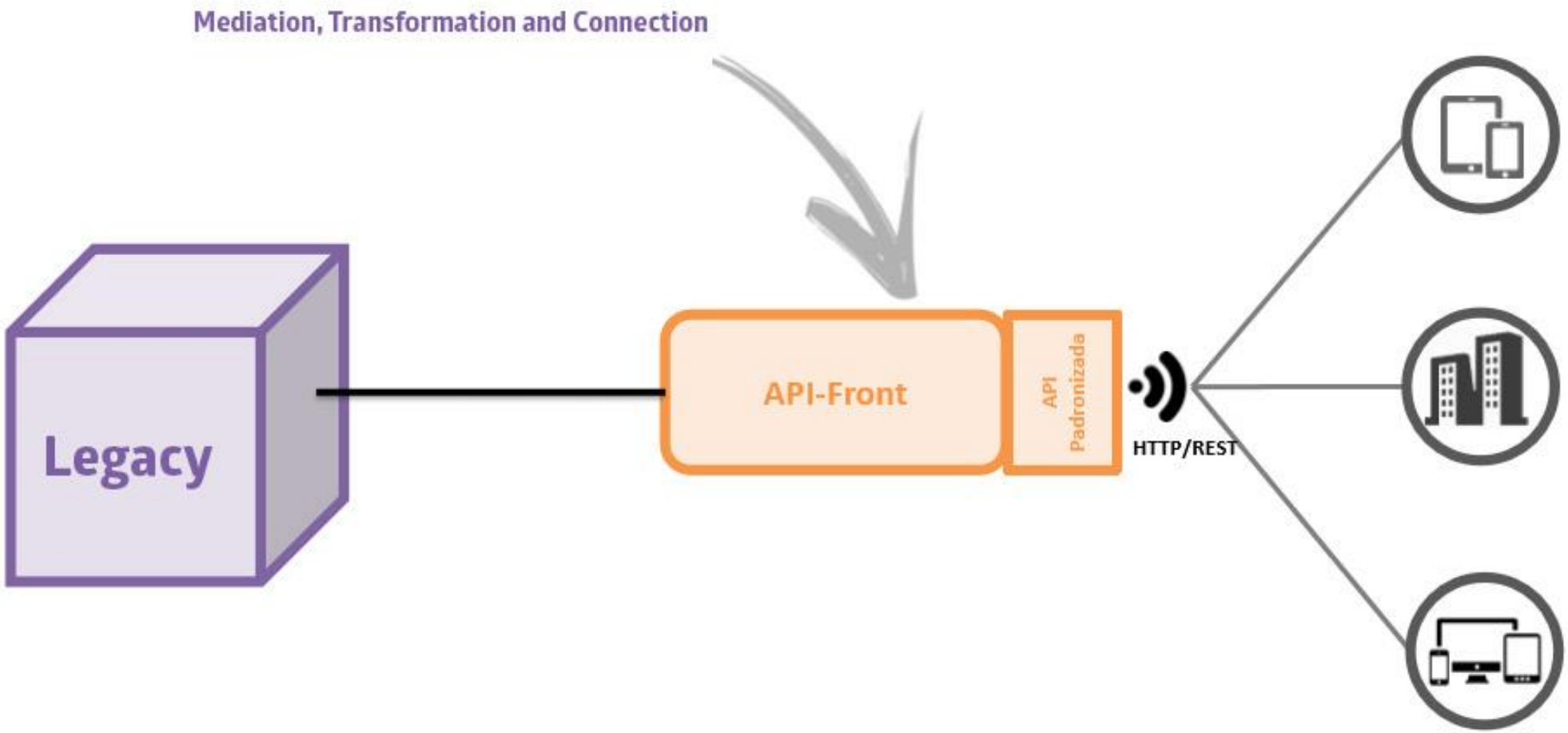
Acesso a Camada de Serviços/APIs

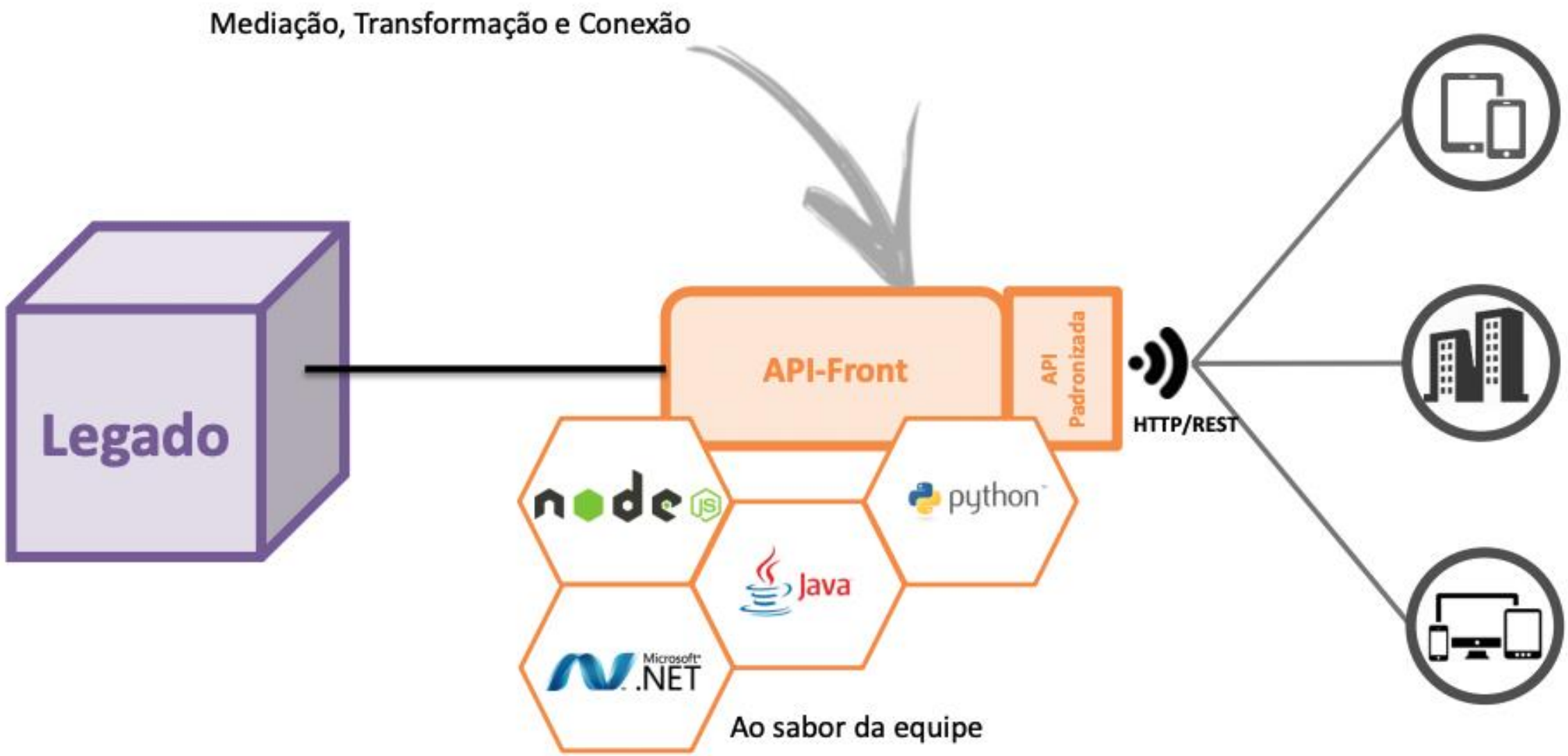
Vantagens

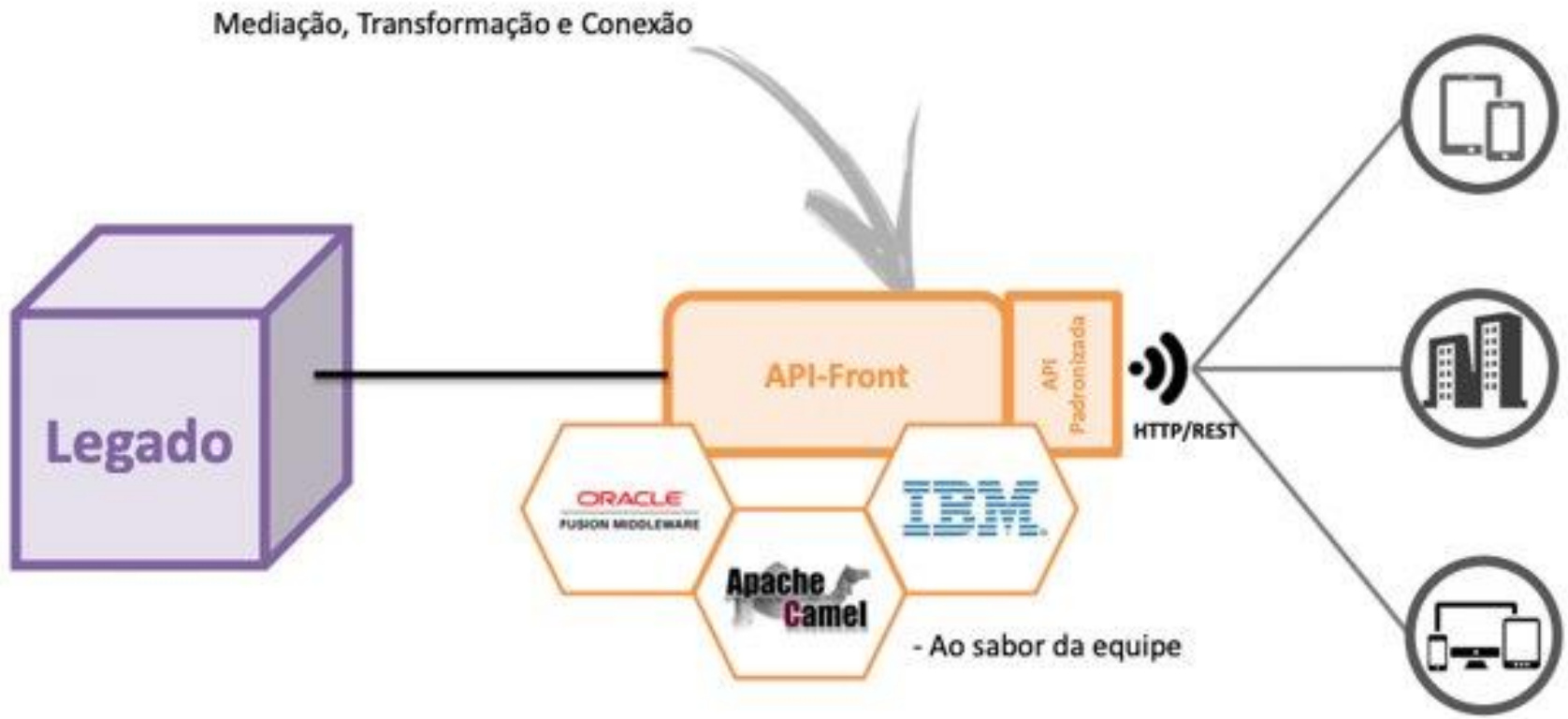
- Uso das funções (camada) de negócio da aplicação
- Não requer alterações de código no lado da aplicação
- Transformação de protocolo e formato de dados pesadas

Riscos

- A diversidade de protocolos e suas peculiaridades
- A tecnologia do API-Front com adaptador do protocolo
- Entender todos os serviços e funções disponibilizadas
- Dificuldade em ter aderência RESTfull
- Dilemas em cenários de aparente composição
- Escalabilidade


Mediation, Transformation and Connection
Legacy
API-Front
API Padronizada
HTTP/REST


Mediação, Transformação e Conexão
Legado
API-Front
API Padronizada
HTTP/REST
node
python
Java
Microsoft .NET
Ao sabor da equipe

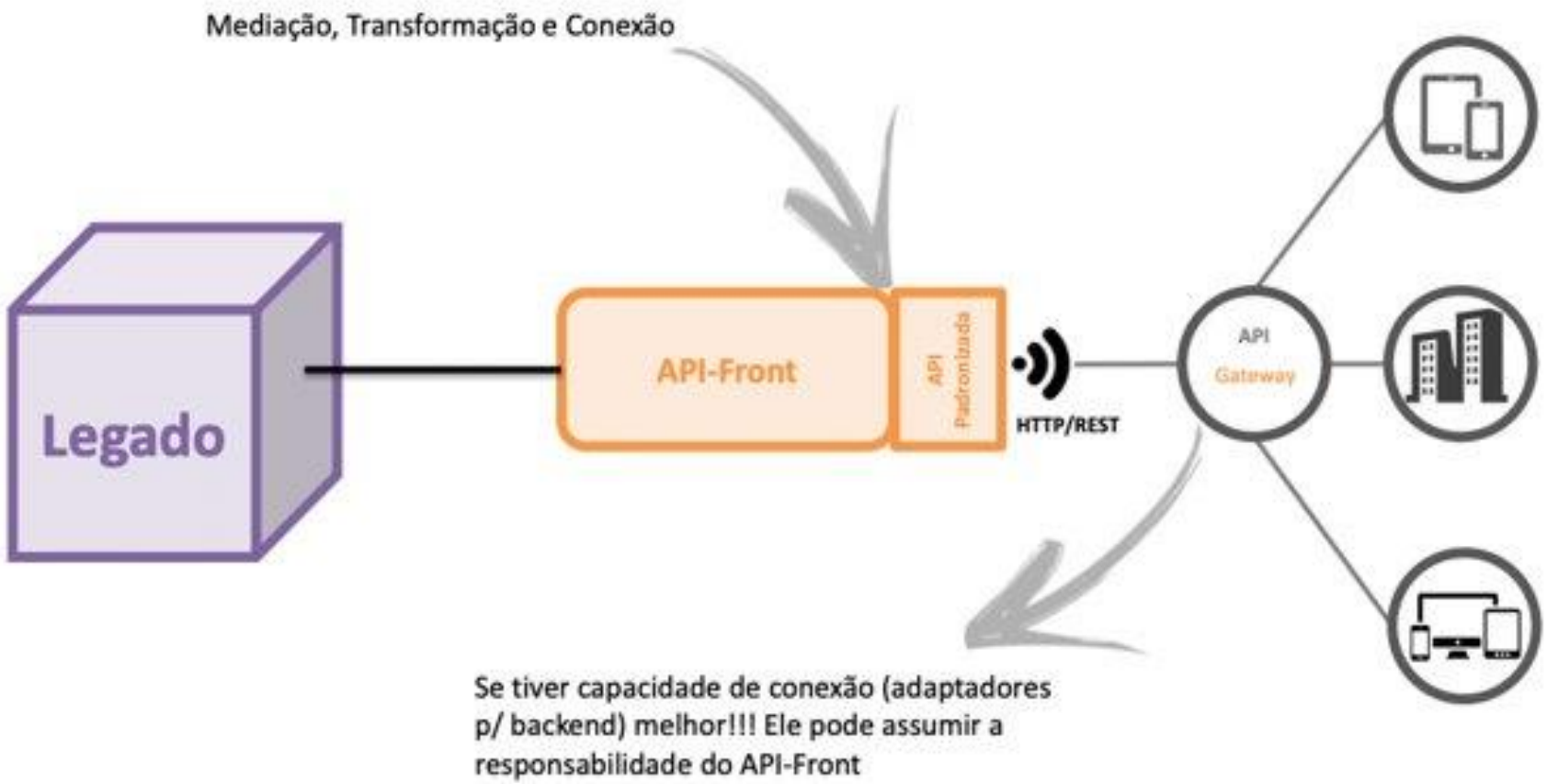

Mediação, Transformação e Conexão
Legado
API-Front
API Padronizada
HTTP/REST
ORACLE
FUSION MIDDLEWARE
IBM
Apache Camel
- Ao sabor da equipe

Atenção

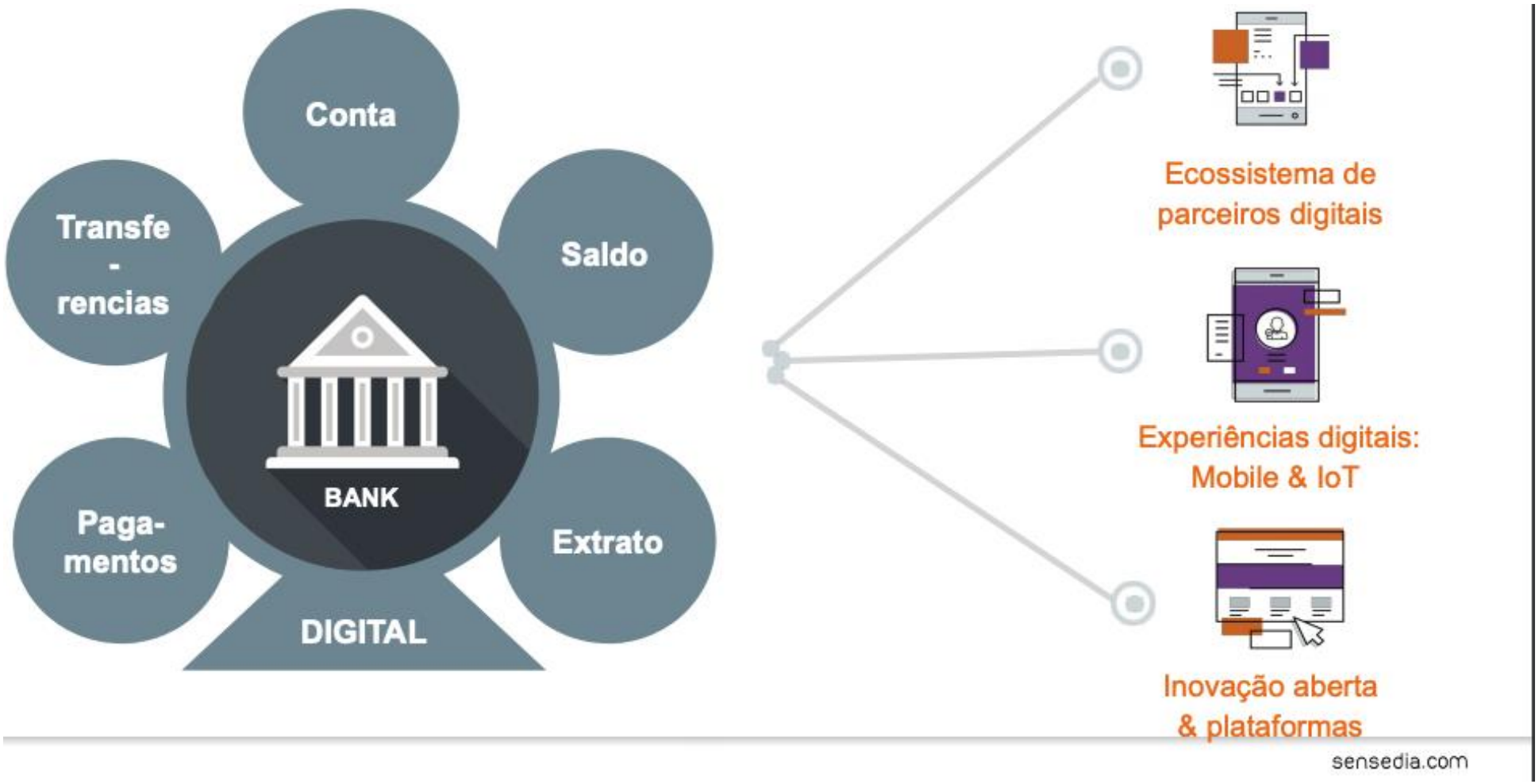
Enderece também no API-Front questões como:

- Rate Limiting
- Monitoring & Alerts
- Authentication Models
- Policy Enforcement
- Exception handling
- Analytics on API Consumption
- JSON Injection/XML Injection
- Cache

Rate Limiting Policy
JSON Threat Policy
Payload Size Policy
IP Filtering Policy

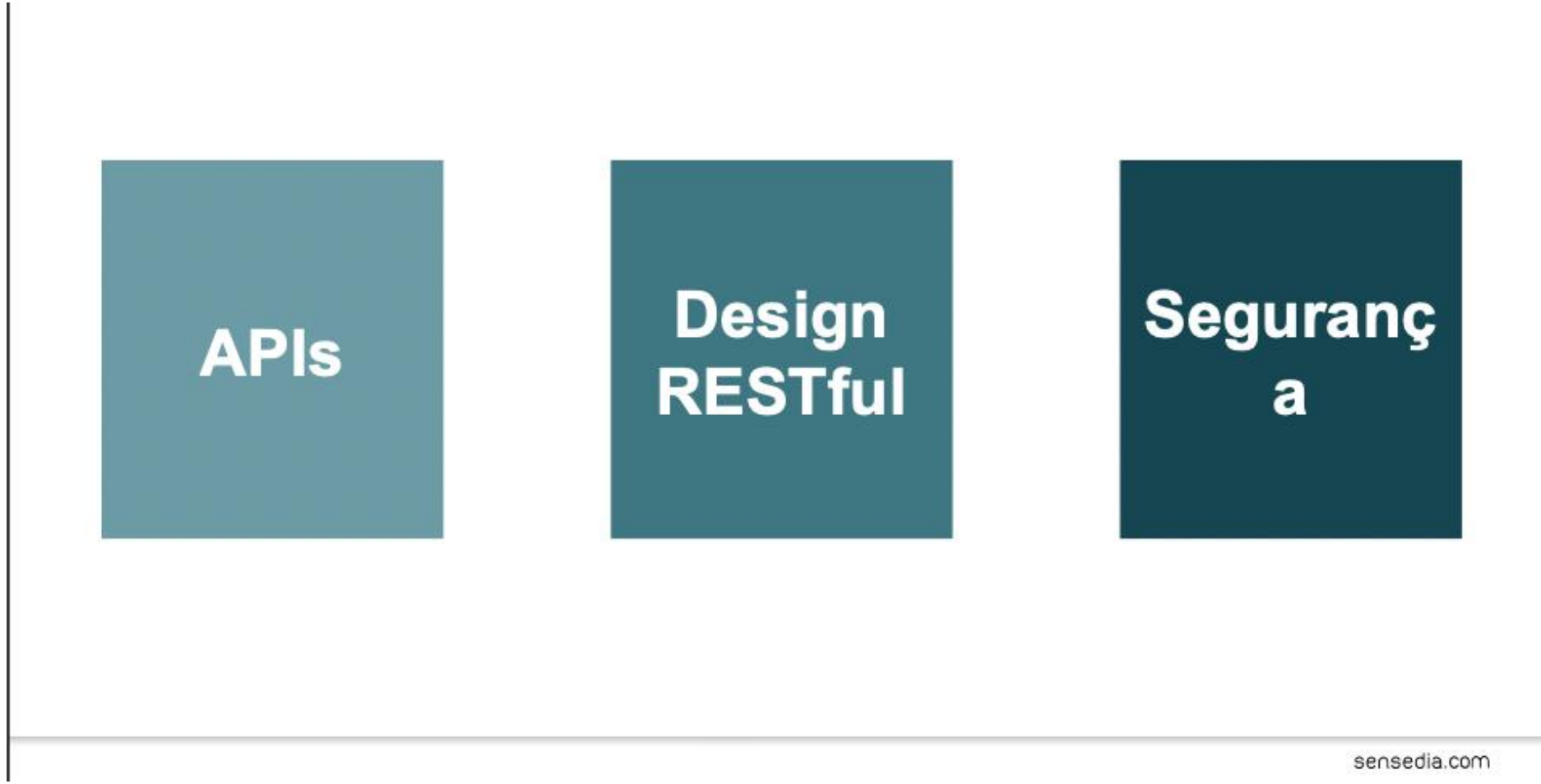

Mediação, Transformação e Conexão
Legado
API-Front
API Padronizada
HTTP/REST
API Gateway
Se tiver capacidade de conexão (adaptadores p/ backend) melhor!!! Ele pode assumir a responsabilidade do API-Front

CASE 1

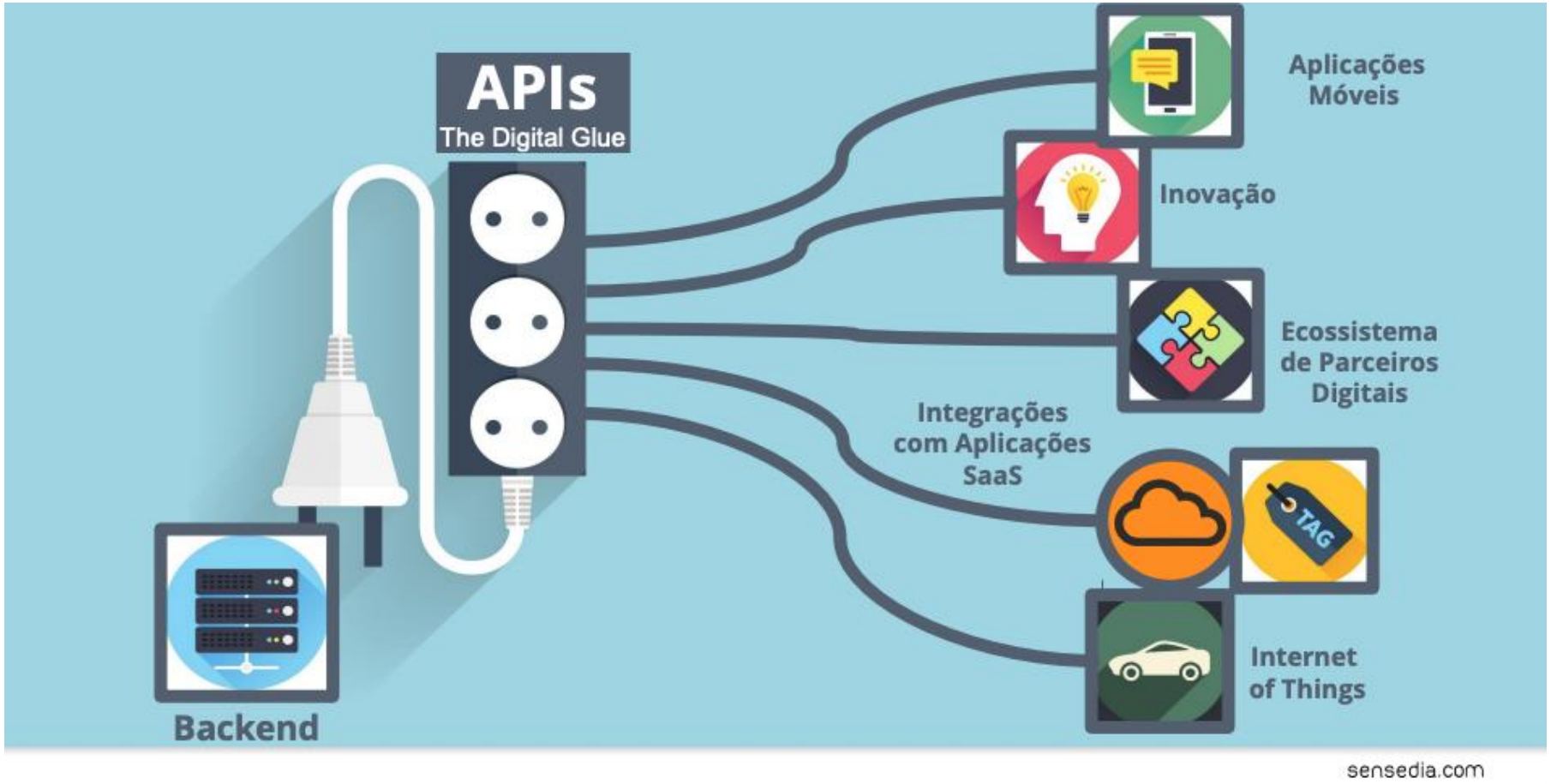

Conta
Transfe
-
rencias
Saldo
BANK
Paga-
mentos
Extrato
DIGITAL
Ecossistema de
parceiros digitais
Experiências digitais:
Mobile & IoT
Inovação aberta
& plataformas
sensedia.com

Abordagens sobre Desenvolvimento de API

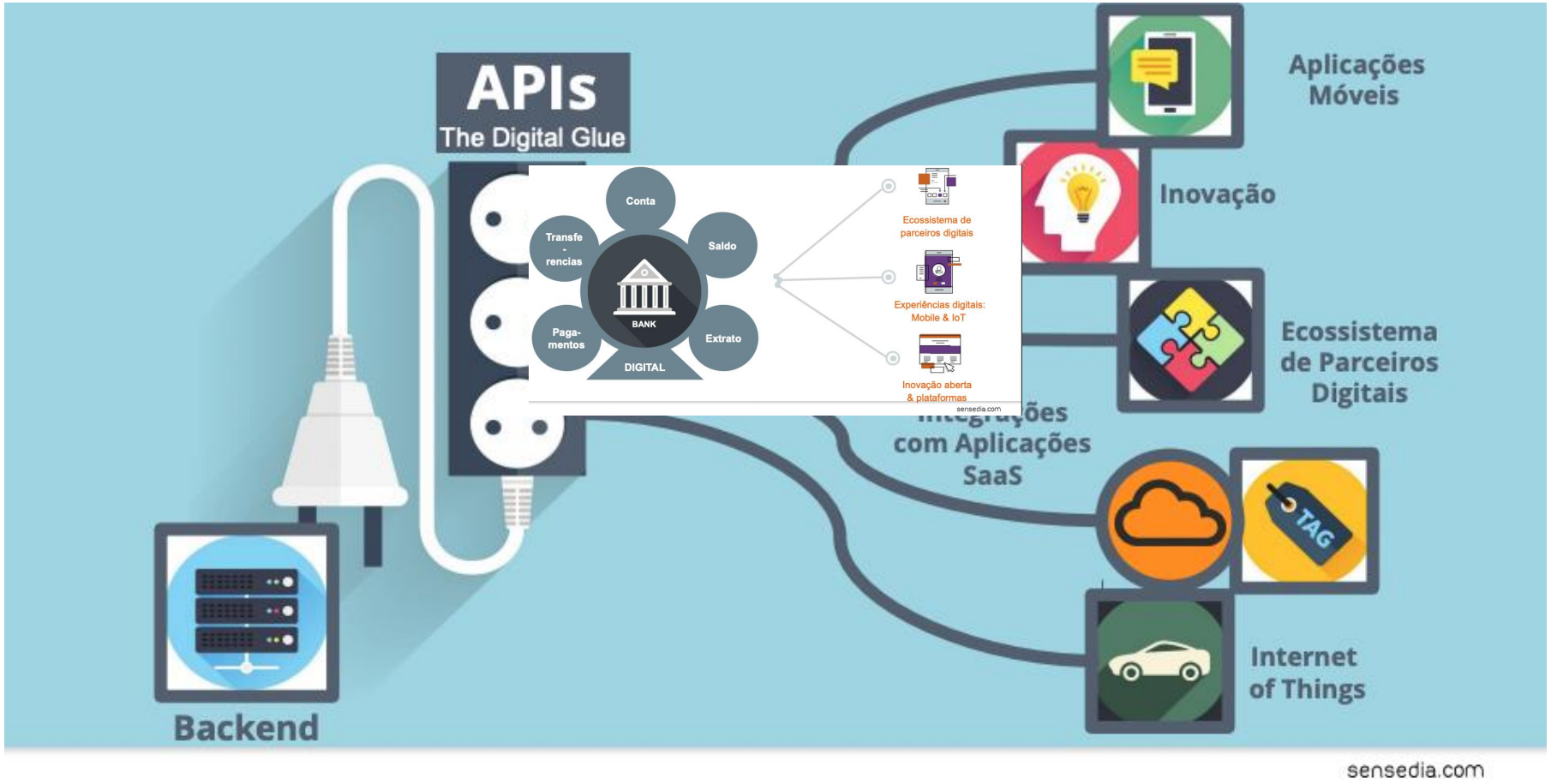
Abordagens sobre Desenvolvimento de API



Abordagens sobre Desenvolvimento de API



Abordagens sobre Desenvolvimento de API



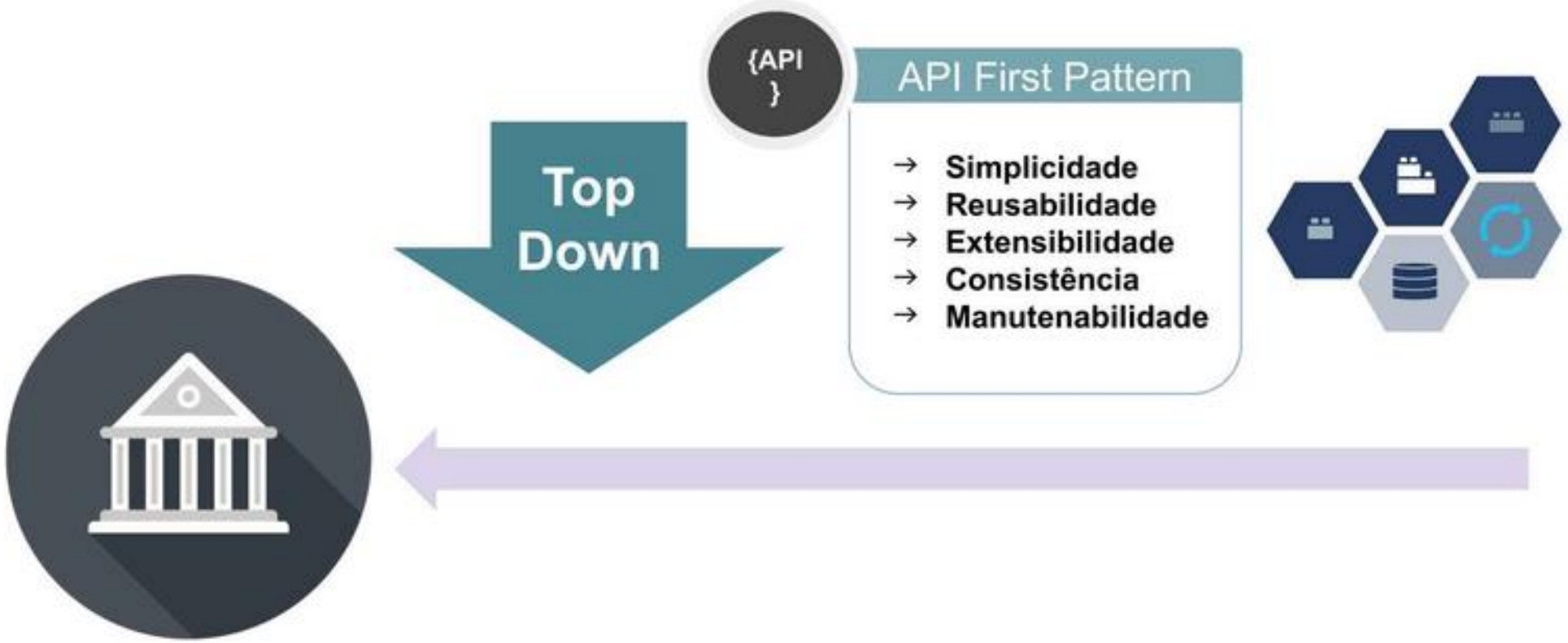
Abordagens sobre Desenvolvimento de API

Abordagens sobre Desenvolvimento de API

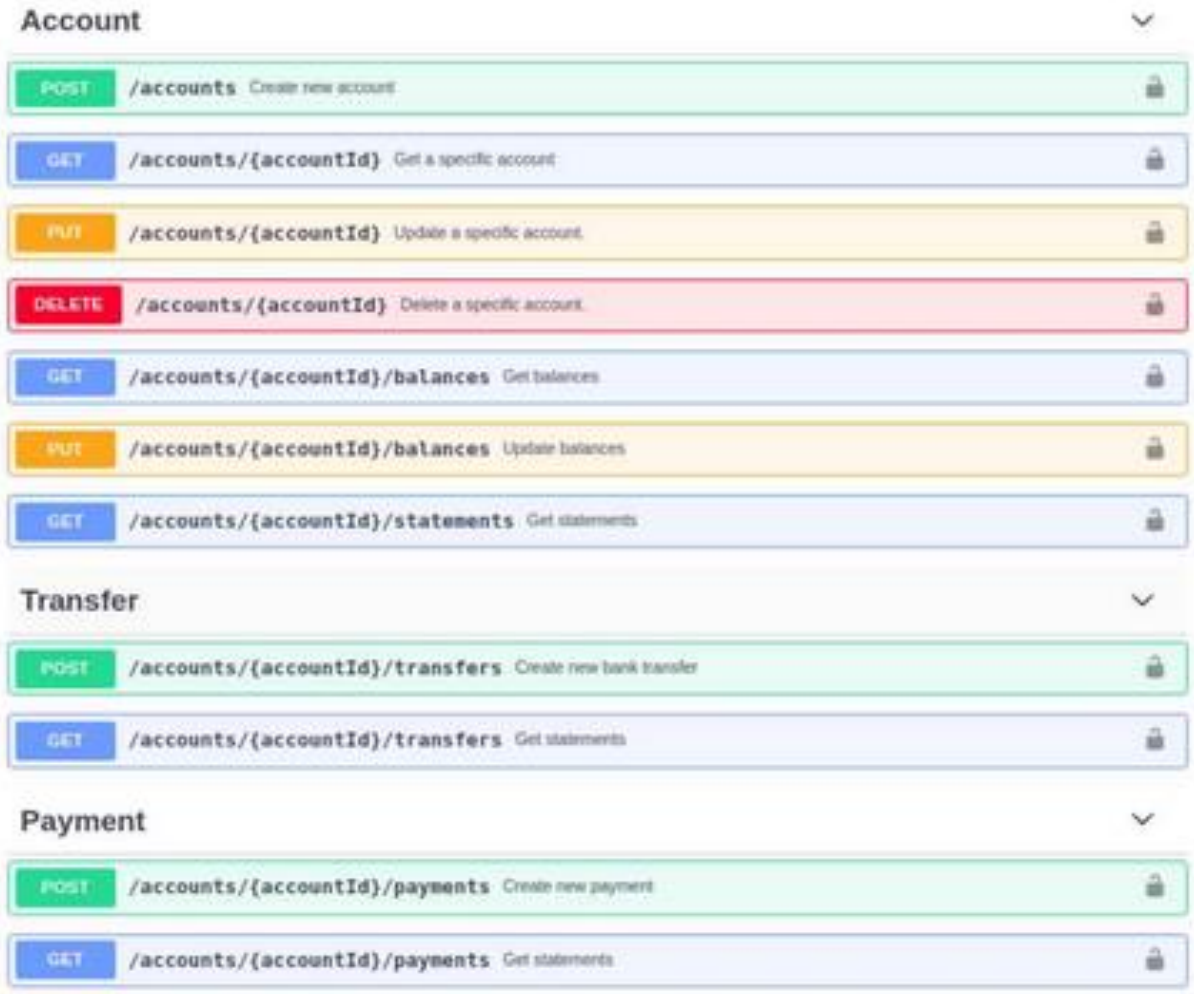
Fonte: State of API Report 2016, SmartBear

Abordagens sobre Desenvolvimento de API

Abordagens sobre Desenvolvimento de API



Abordagens sobre Desenvolvimento de API



Status Code 2xx

200	OK
201	Created
204	No Content

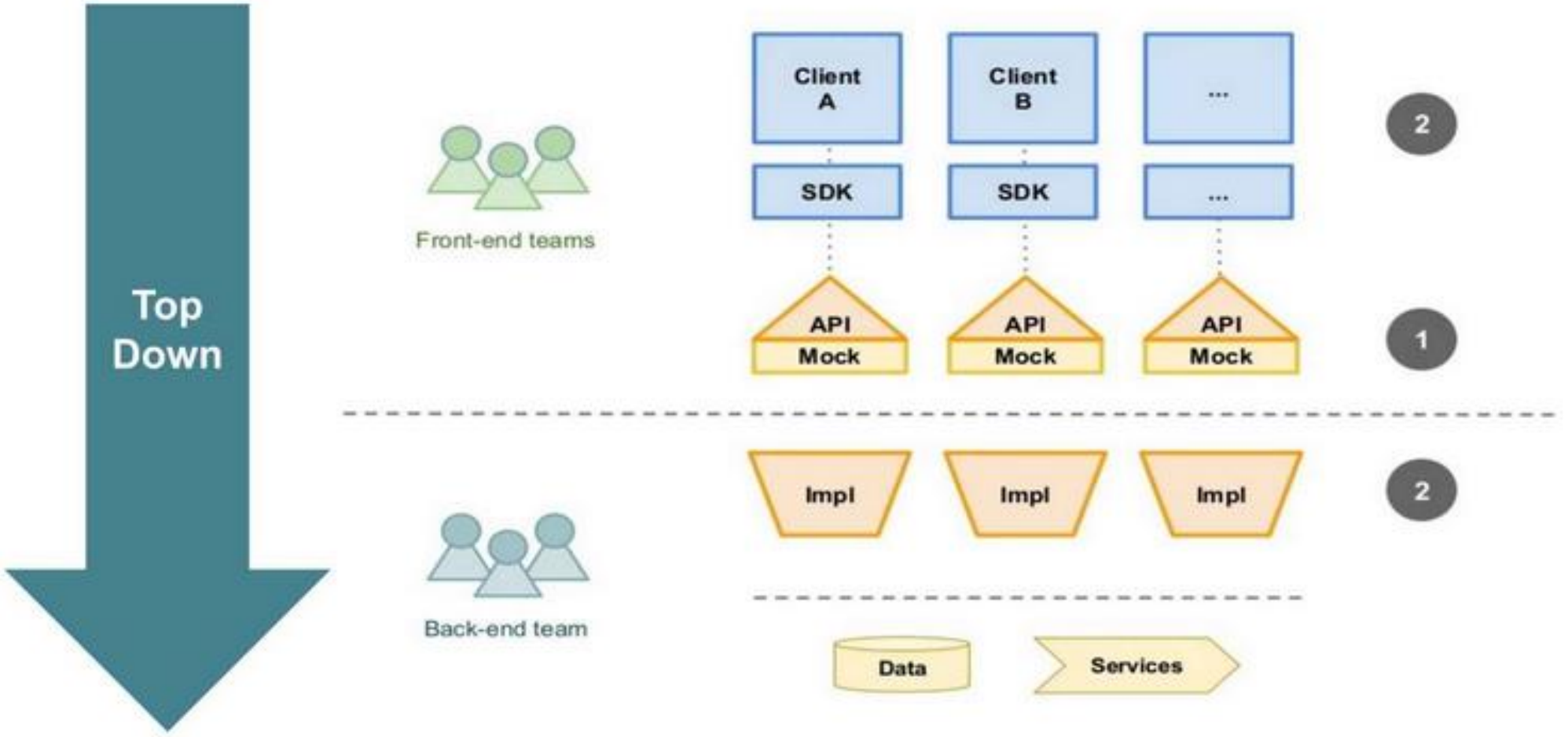
Status Code 4xx

400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
422	Unprocessable Entity

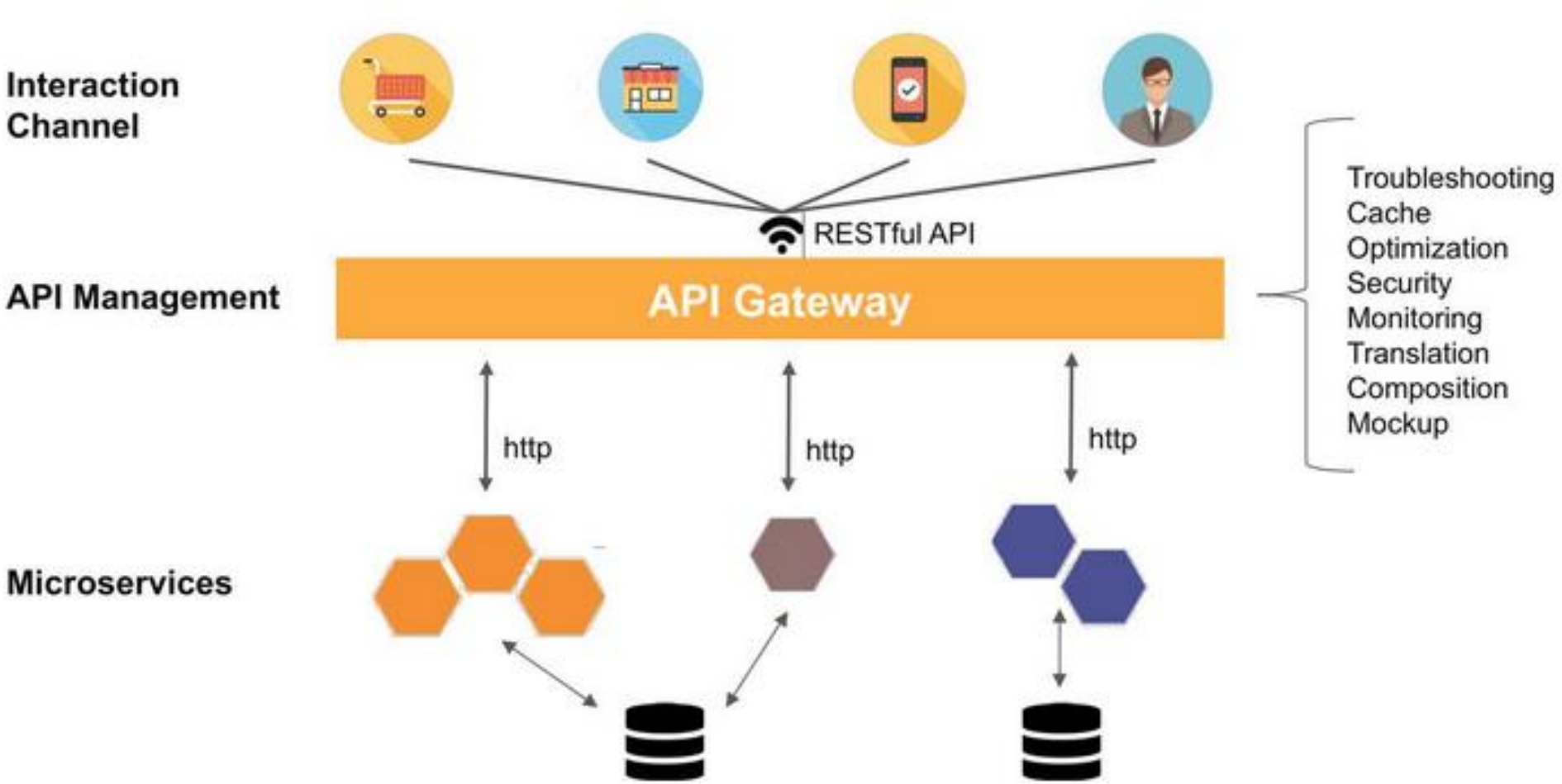
Status Code 5xx

500	Internal Server Error

Abordagens sobre Desenvolvimento de API

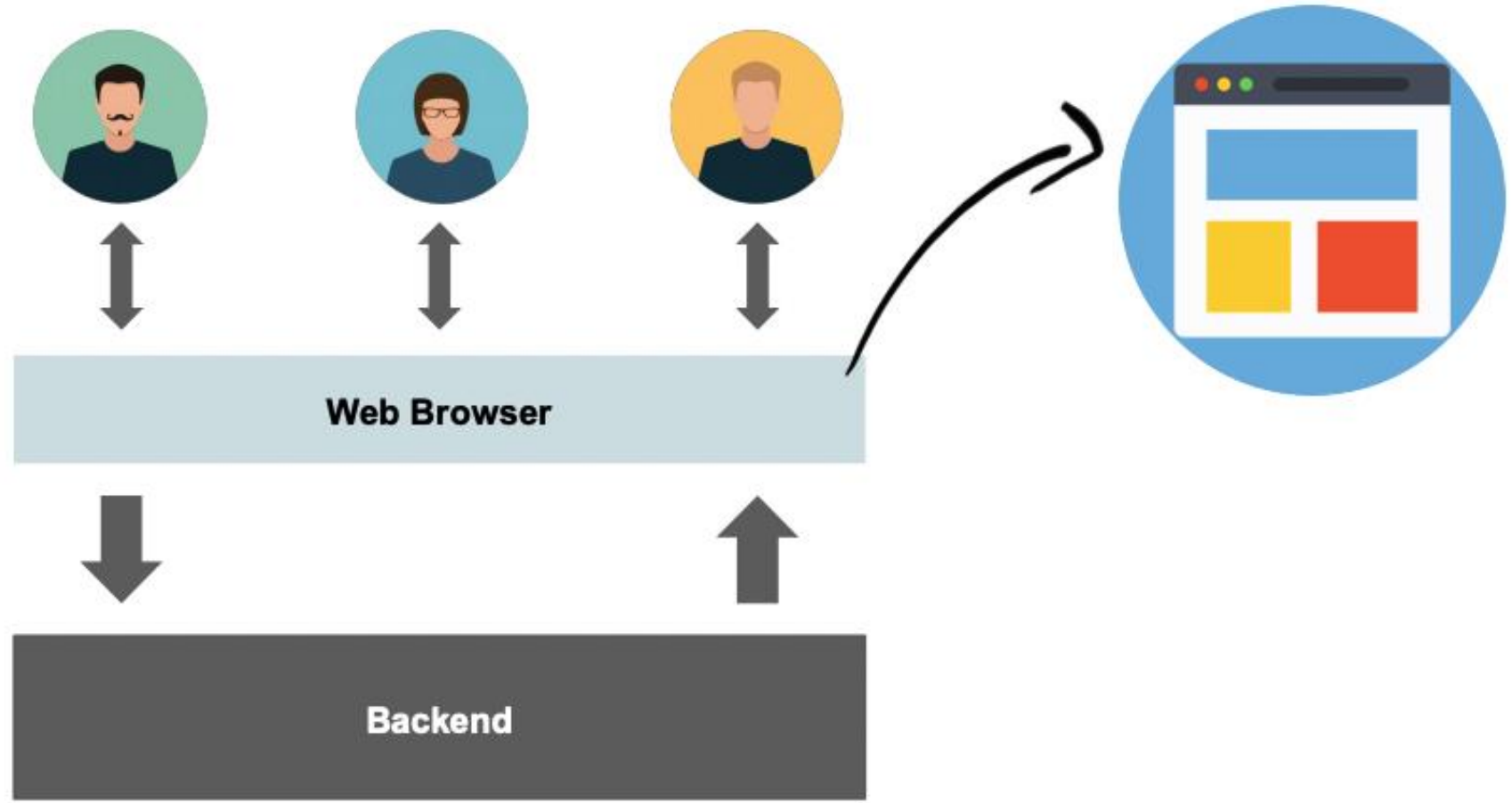


Abordagens sobre Desenvolvimento de API

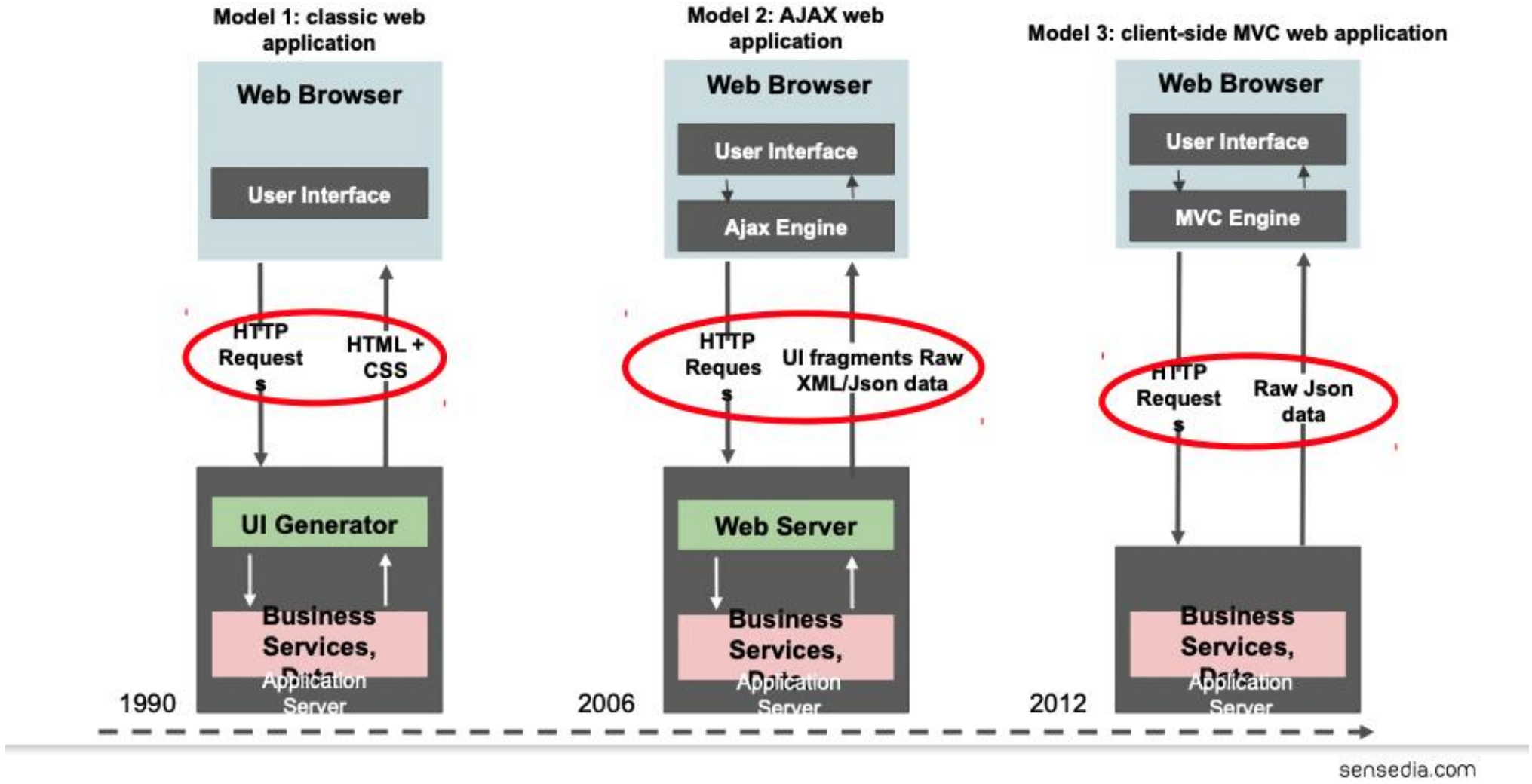


Abordagens sobre Desenvolvimento de API

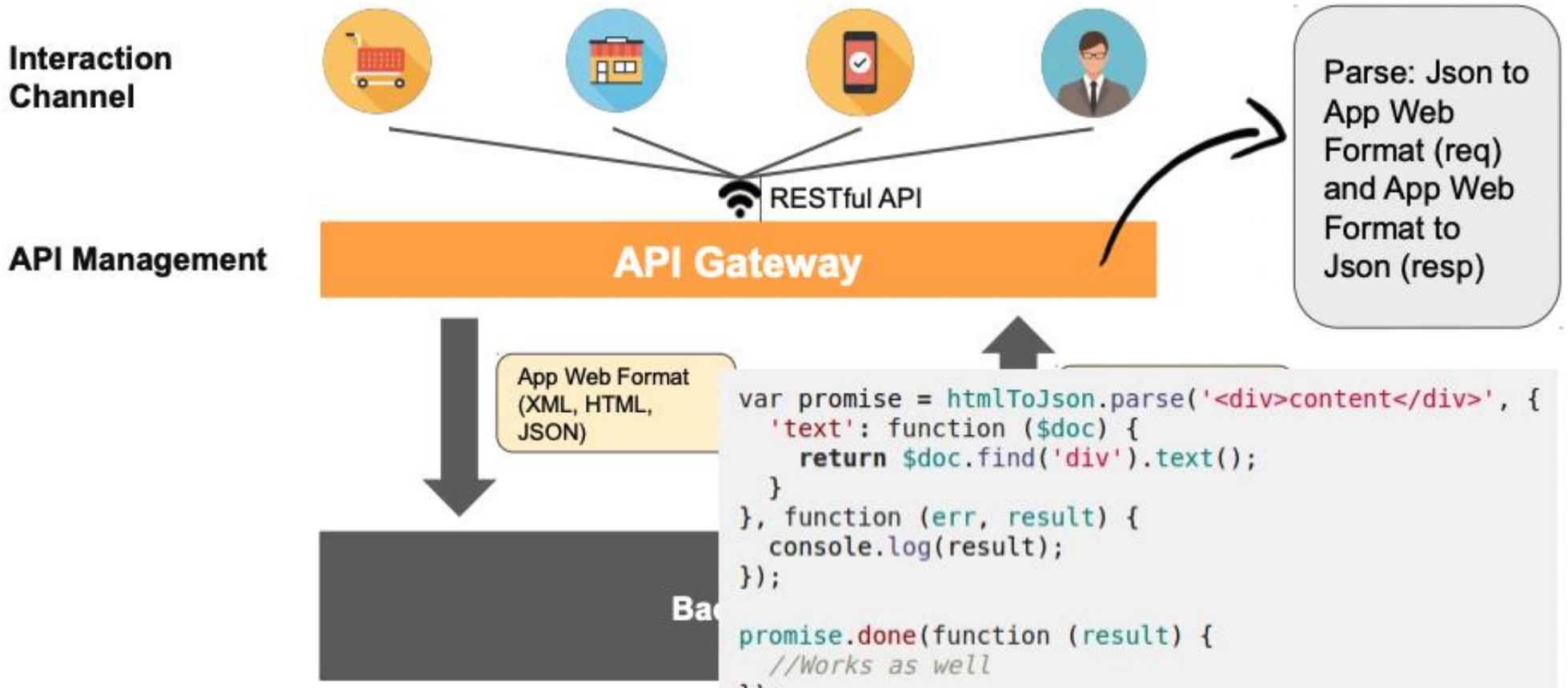
Abordagens sobre Desenvolvimento de API



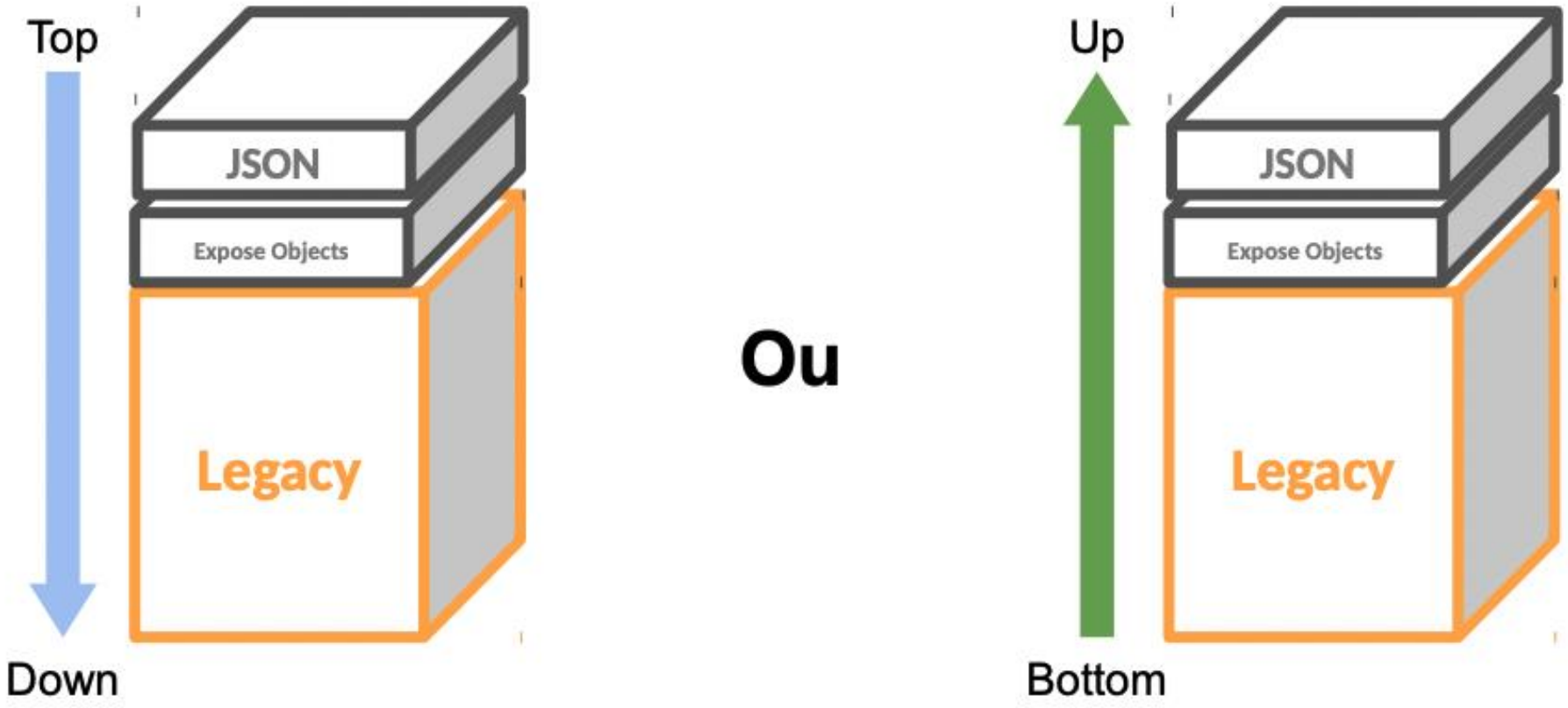
Abordagens sobre Desenvolvimento de API



Abordagens sobre Desenvolvimento de API

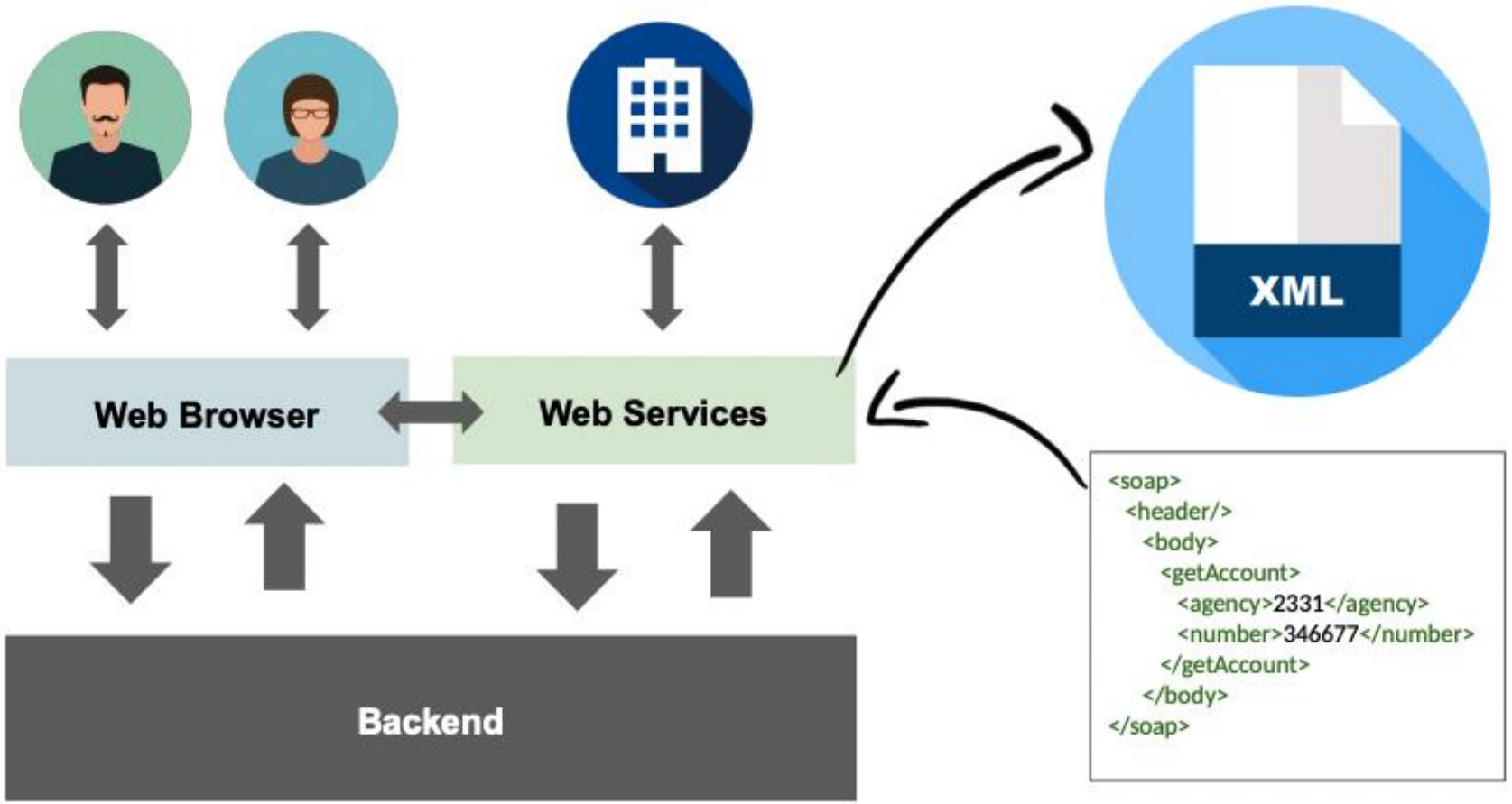


Abordagens sobre Desenvolvimento de API



Abordagens sobre Desenvolvimento de API

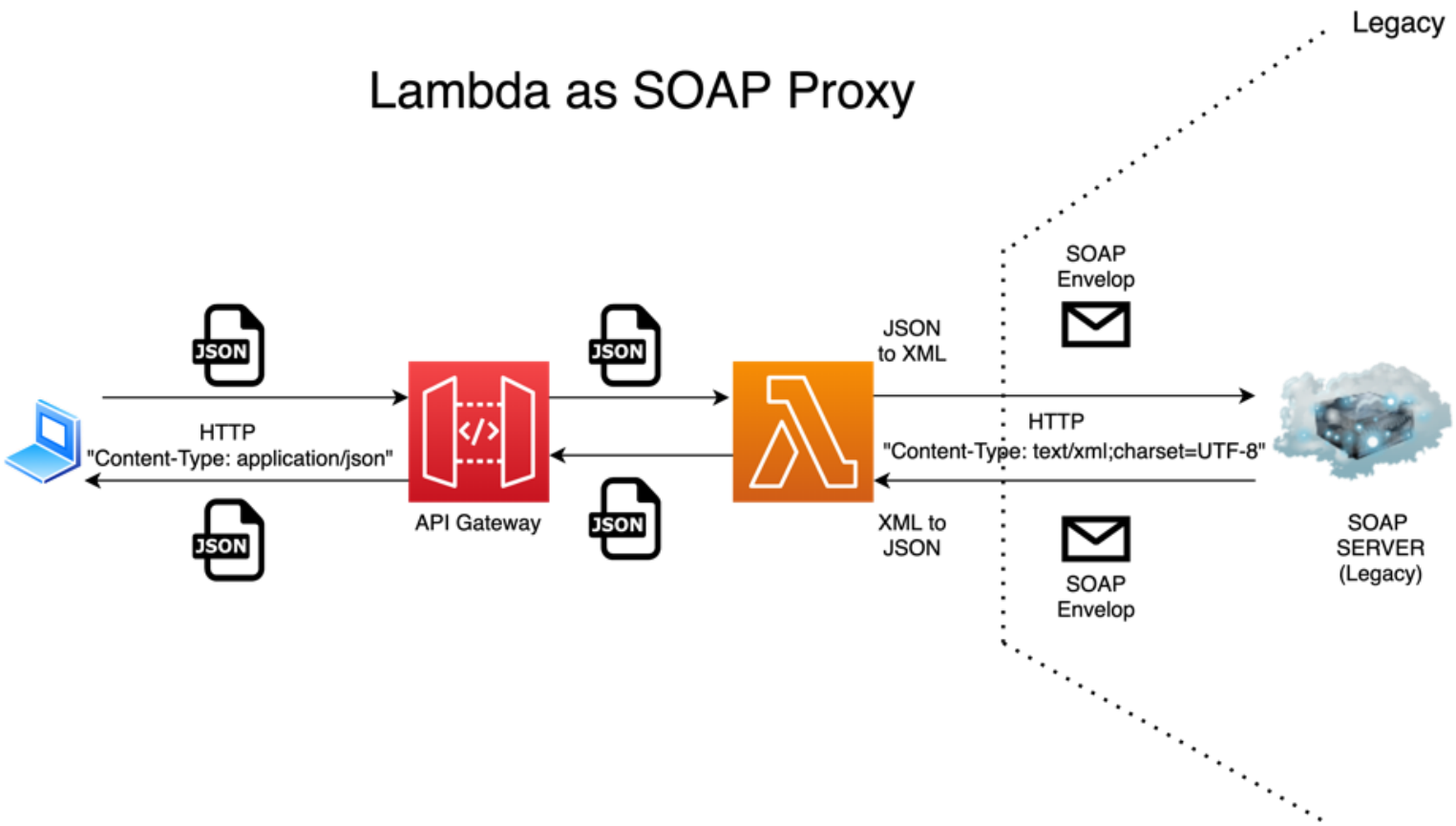
Abordagens sobre Desenvolvimento de API



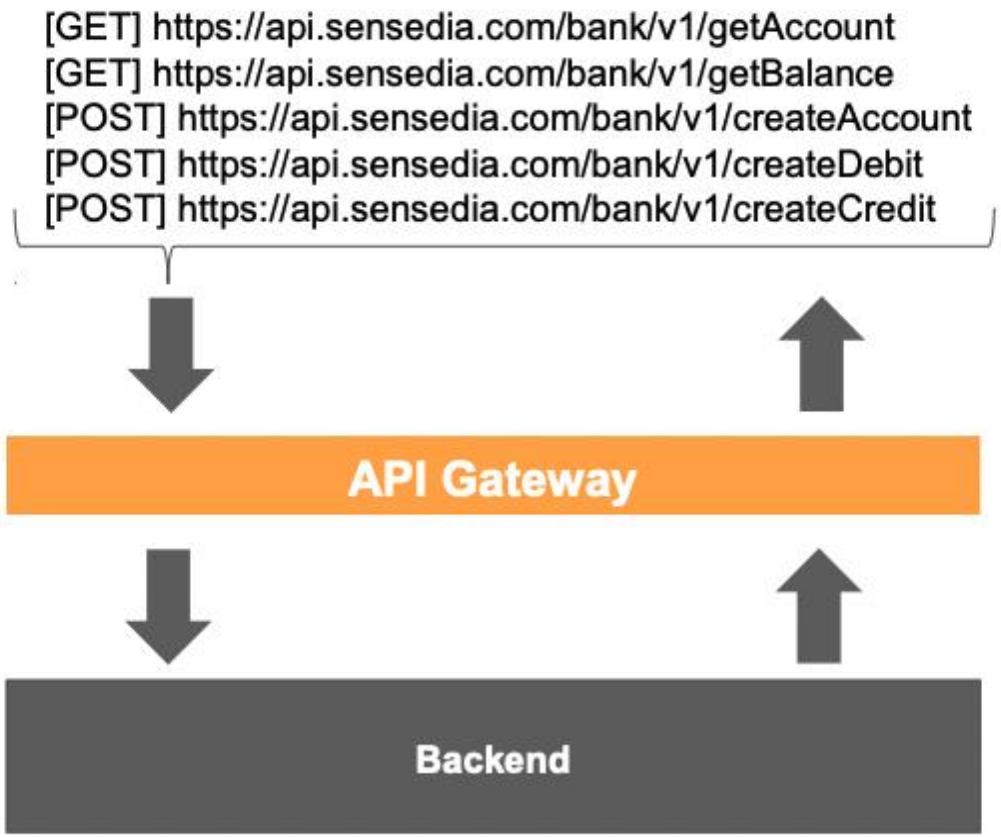
Abordagens sobre Desenvolvimento de API

- Exemplo
- https://www.youtube.com/watch?v=jeNXLzpKCaA&t=7s

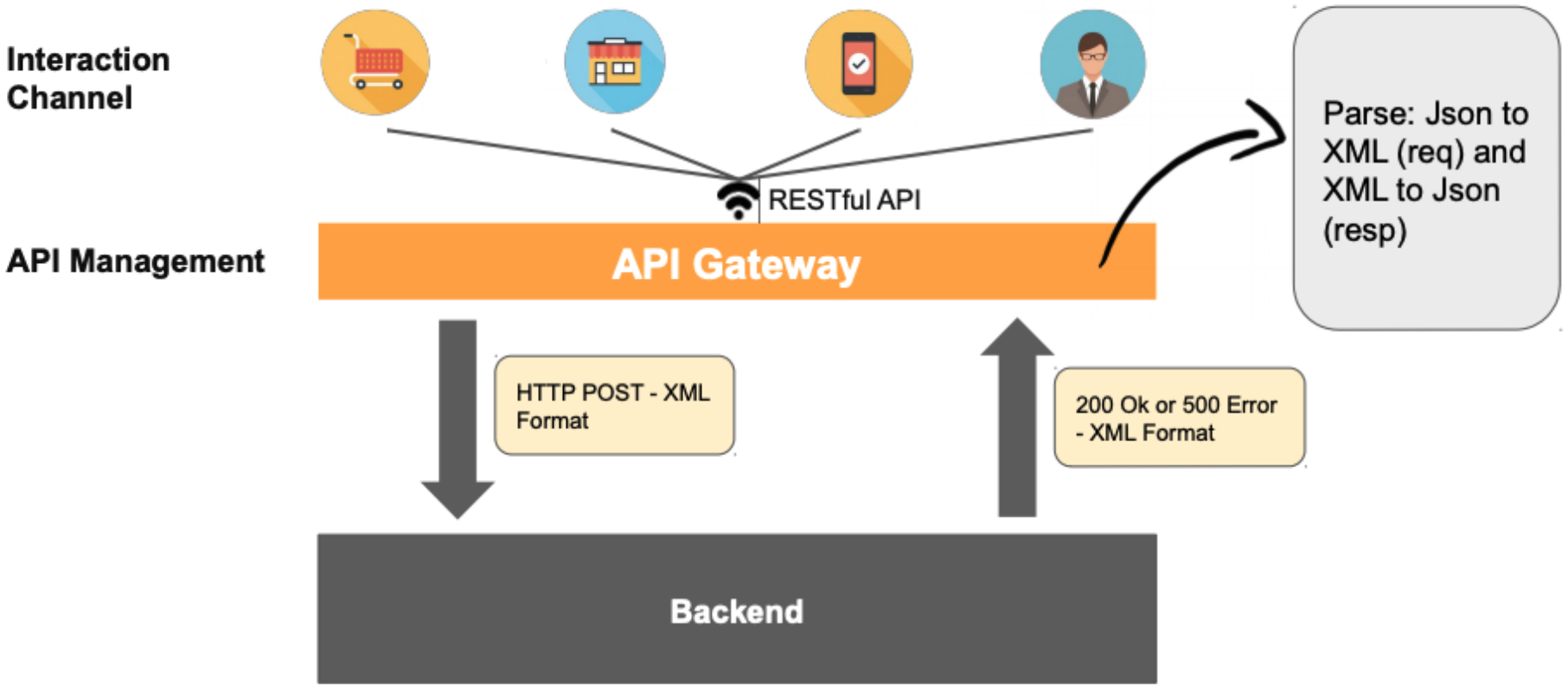
Abordagens sobre Desenvolvimento de API



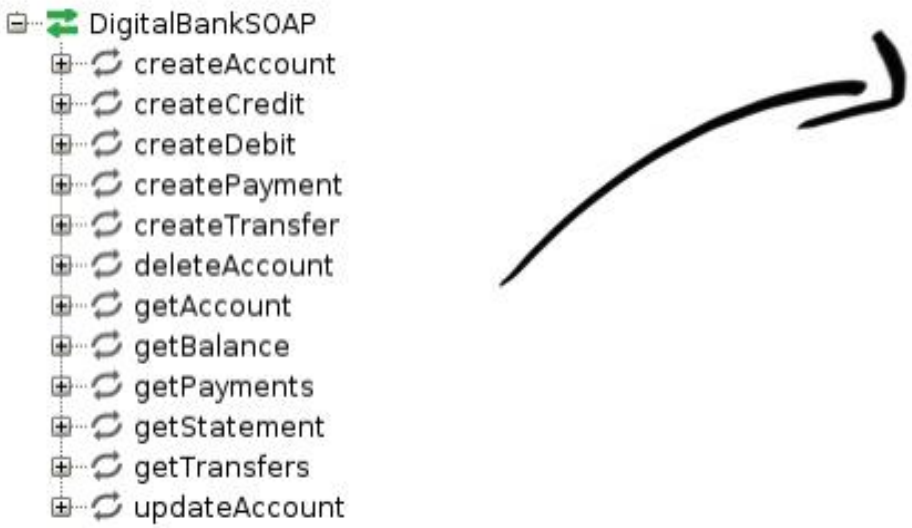
Abordagens sobre Desenvolvimento de API

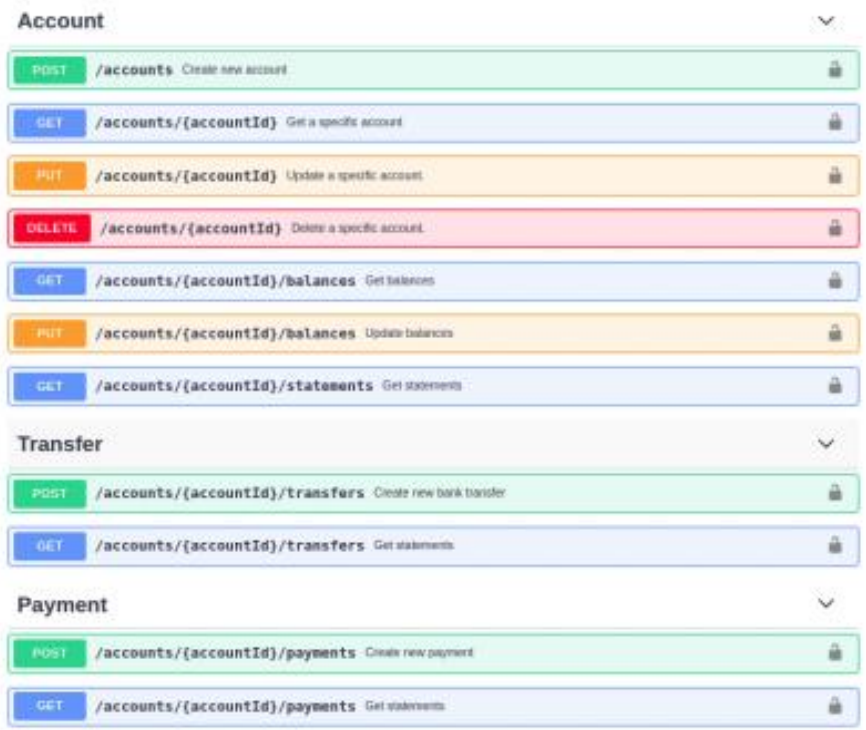


Abordagens sobre Desenvolvimento de API



Abordagens sobre Desenvolvimento de API





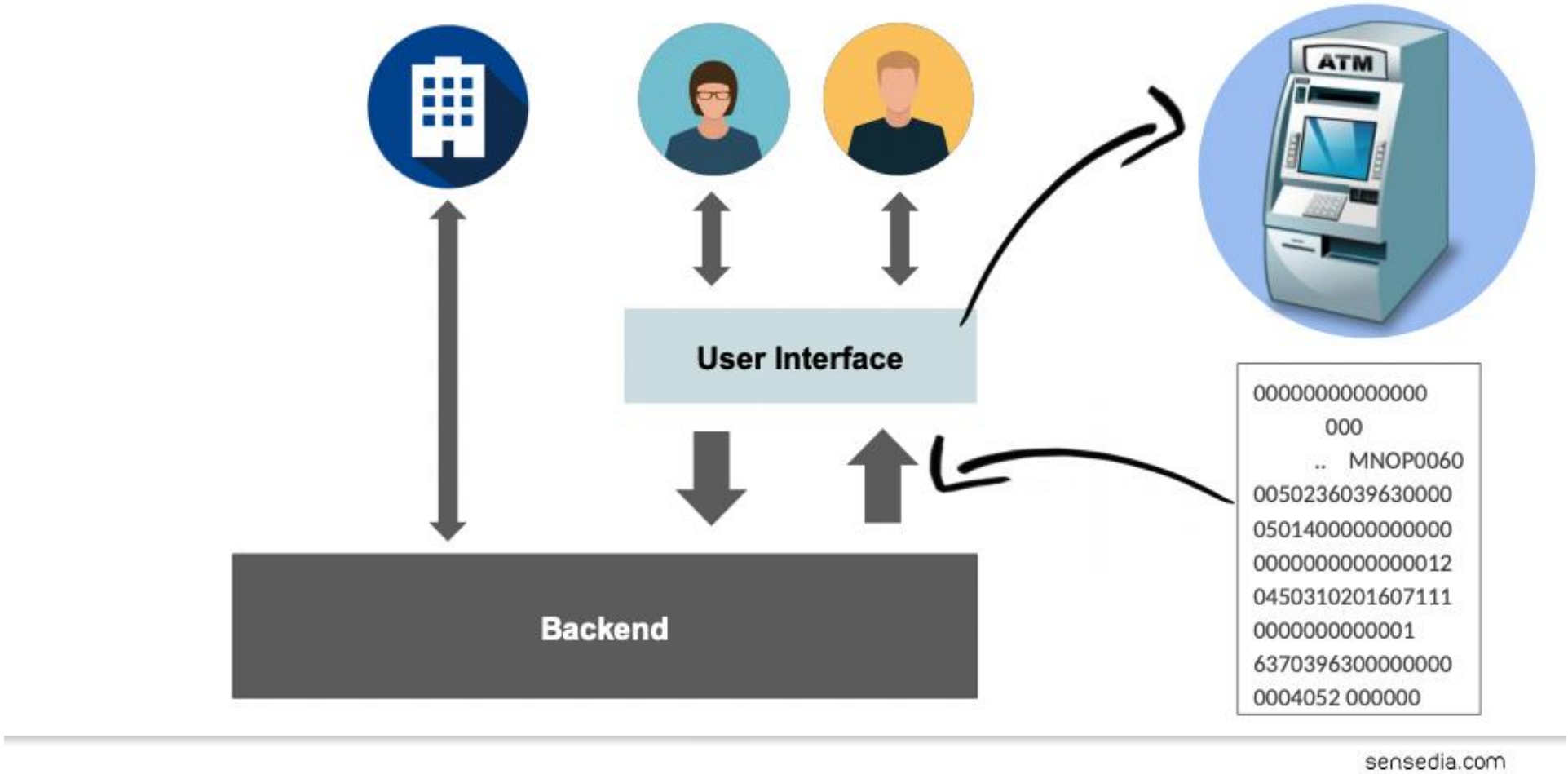
Abordagens sobre Desenvolvimento de API

Pontos de Atenção

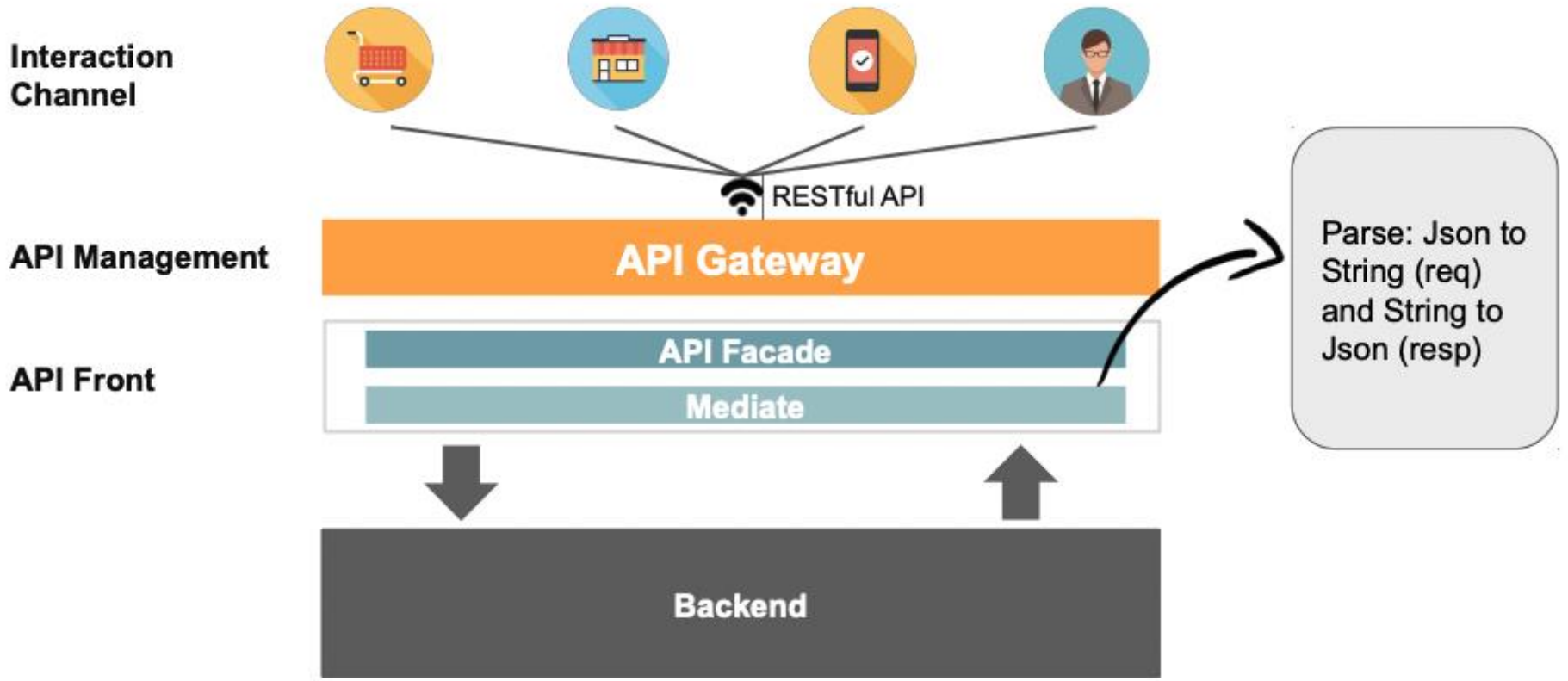
1. Levar a "verbalização" das operações do WS para os recursos da API.
2. Tratar erros da maneira adequada para o formato de API pode se tornar uma tarefa "pesada" .
3. Simplificar a estrutura de dados do WS - cuidado com o aninhamento.
4. Versionamento do WS

Abordagens sobre Desenvolvimento de API

Abordagens sobre Desenvolvimento de API

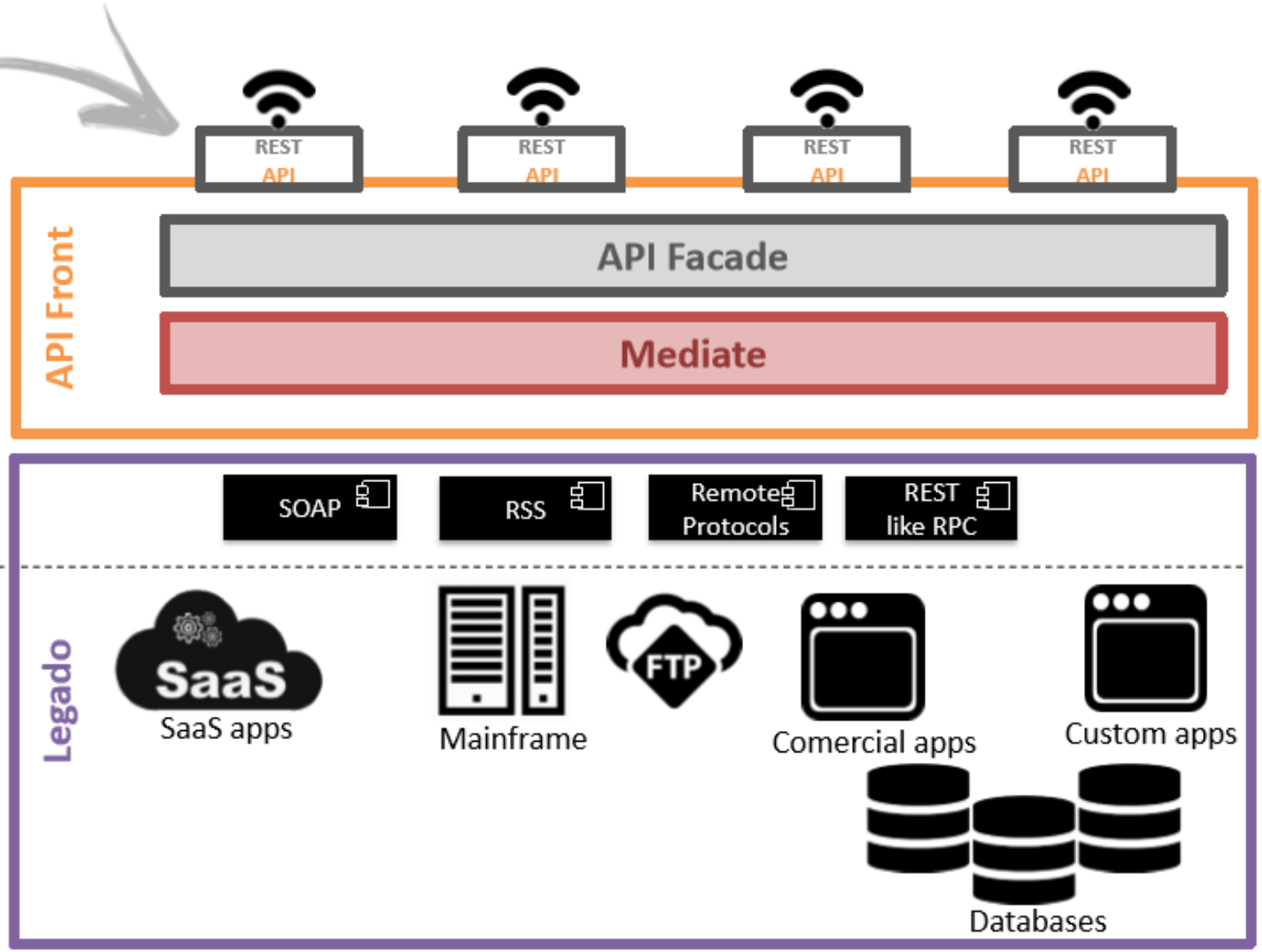


Abordagens sobre Desenvolvimento de API



Design Ideal

API-First



Webinar Design de APIs RESTful
http://bit.ly/Sensedia_APIsRest

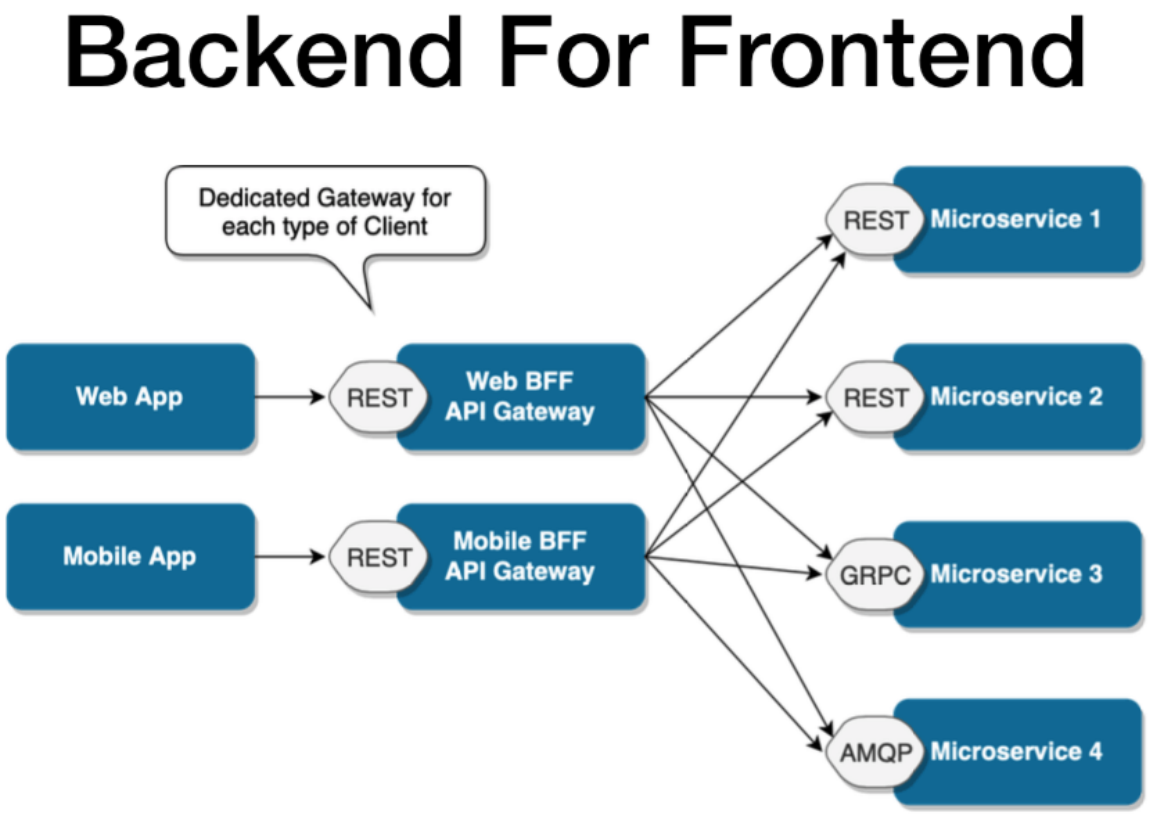
Estratégias sem impacto no backend para a exposição de APIs

- Assim, sugerimos estabelecer uma camada que chamamos de API-front, conectada ao sistema legado e composta de 2 subcamadas:

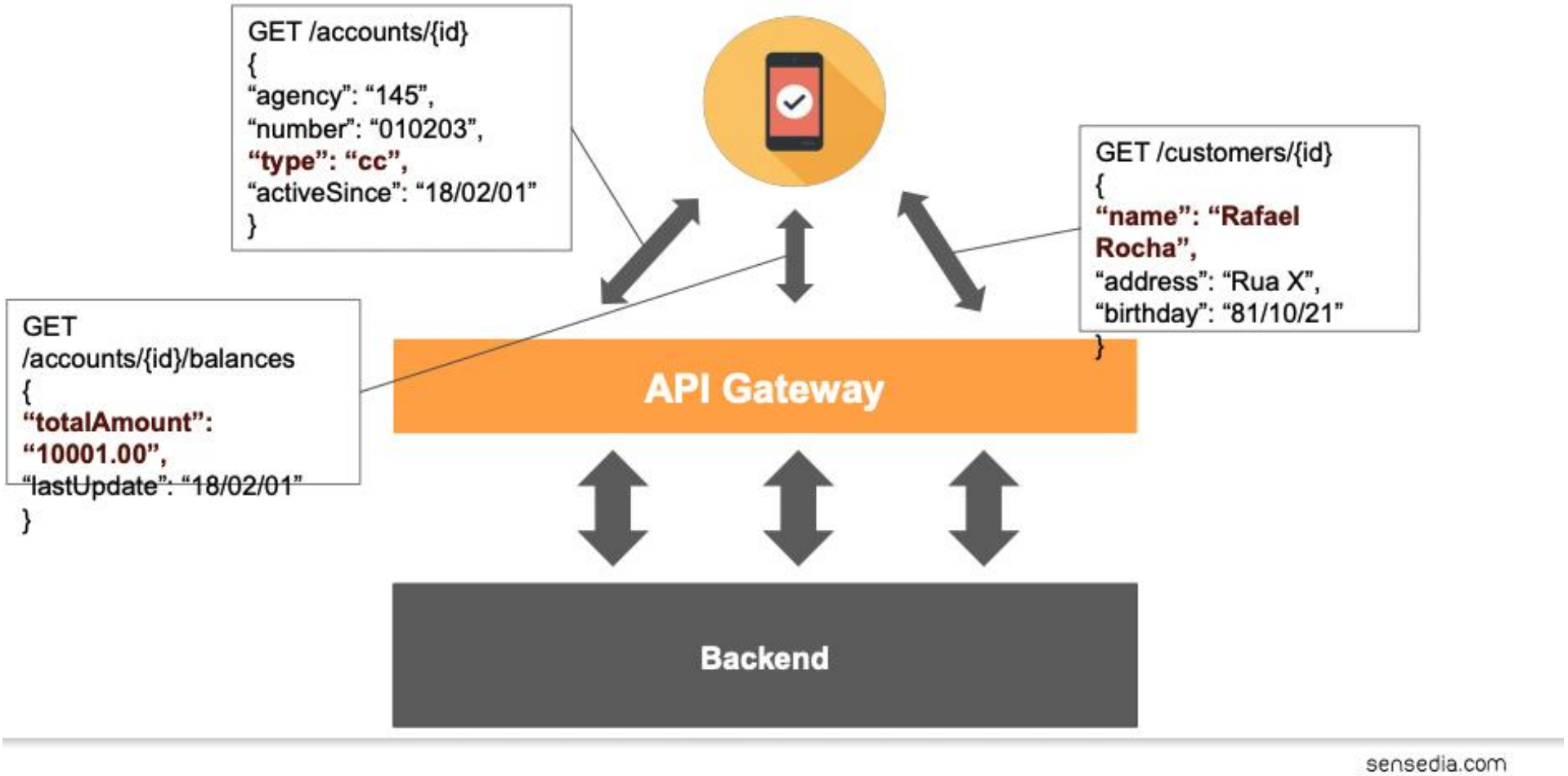
  - *Fachada API* - resume todo o legado e permite compor, por exemplo, APIs SaaS com um mainframe, expondo-o em um formato adequado.

  - *Mediar* - rotas da Fachada API para o backend. Desta forma, queremos expor o API em um formato ideal para os diferentes dispositivos e frontends.

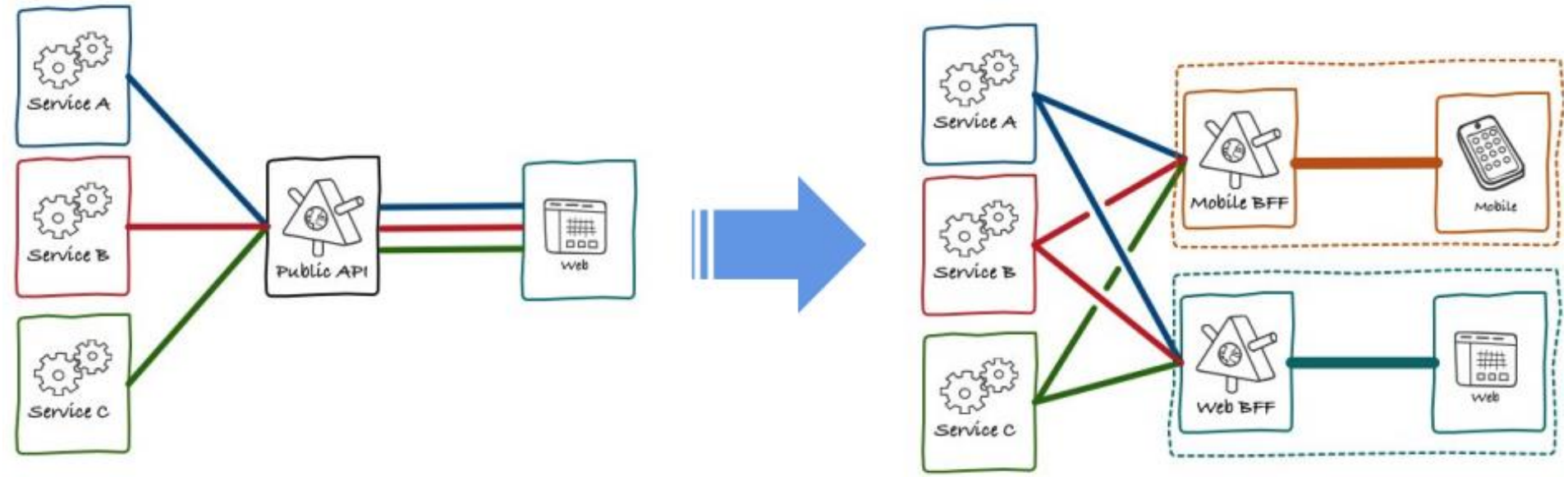
Estratégias sem impacto no backend para a exposição de APIs



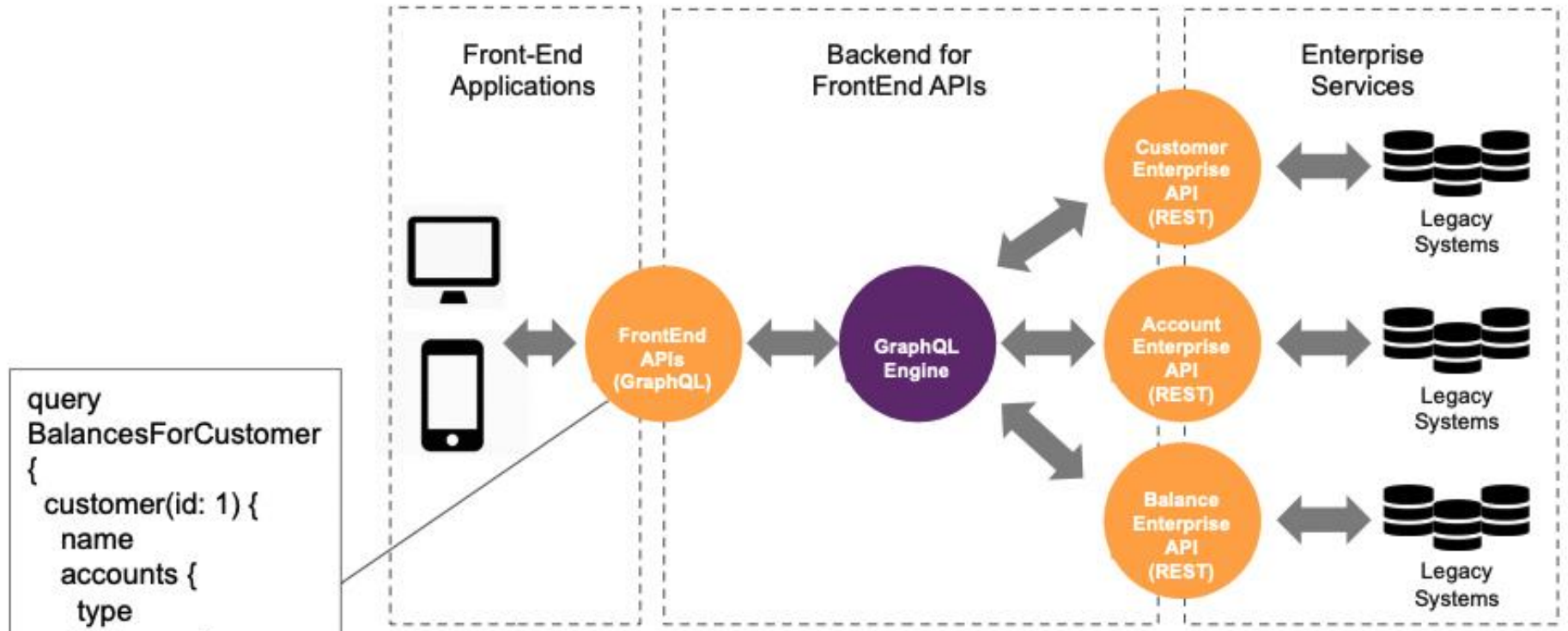
Estratégias sem impacto no backend para a exposição de APIs



Estratégias sem impacto no backend para a exposição de APIs



Estratégias sem impacto no backend para a exposição de APIs



Estratégias sem impacto no backend para a exposição de APIs

- Resumo
  - Várias estratégias, porém o mais importante é entender bem o contexto.
  - Uso de API Gateway pode reduzir o trabalho de conversão e gestão.
  - Entender sempre a desvantagens de cada alternativa

Boas Práticas

- Versionamento
- Somente exponha a “versão cheia” (v1, v2)
- Evite ao máximo Breaking Changes
  - SIM
    - Remover campos, Renomear campos, Remover recursos, Remover endpoints
  - NÃO
    - Adicionar campos, Adicionar recursos, Adicionar endpoints, Corrigir bugs *

Boas Práticas

- Não responda assim

```
{ "name": "John Doe",
  "email": "johndoe@gmail.com",
  "birthday": "17/09/2016"
}
```

- Faça um “wrap” da resposta

```
{ "data":
    { "name": "John Doe",
      "email": "johndoe@gmail.com",
      "birthday": "17/09/2016" }
 }
```

Boas Práticas

- A mensagem de erro deve ajudar os usuários a **entender e resolver** o erro da API de maneira fácil e rápida. Em geral, considere as diretrizes a seguir ao escrever mensagens de erro:
  - Não presuma que o usuário é um usuário especialista da sua API. Os usuários podem ser desenvolvedores de clientes, gerentes de operações, equipes de TI ou usuários finais de apps.
  - Não presuma que o usuário saiba de tudo sobre a implementação do seu serviço ou esteja familiarizado com o contexto dos erros (como análise de registro).
  - Quando possível, as mensagens de erro precisam ser criadas de maneira que um usuário técnico, não necessariamente um desenvolvedor da sua API, possa responder ao erro e corrigi-lo.
  - Mantenha a mensagem de erro resumida. Se necessário, forneça um link em que um leitor com dúvidas possa fazer perguntas, dar feedback ou obter mais informações que não se encaixam em uma mensagem de erro. Caso contrário, use o campo de detalhes para expandir.
  - Nunca exponha Exceções para o Usuário

Boas Práticas


```
{
  "error": {
    "code": 400,
    "message": "API key not valid. Please pass a valid API key.",
    "status": "INVALID_ARGUMENT",
    "details": [
      {
        "@type": "type.googleapis.com/google.rpc.ErrorInfo",
        "reason": "API_KEY_INVALID",
        "domain": "googleapis.com",
        "metadata": {
          "service": "translate.googleapis.com"
        }
      }
    ]
  }
}
```

Boas Práticas

- Documentação
  - Uma API é apenas tão boa quanto sua documentação
  - Swagger, Postman..
  - Mantenha próxima ao código
  - Revisite periodicamente

Boas Práticas

- Segurança

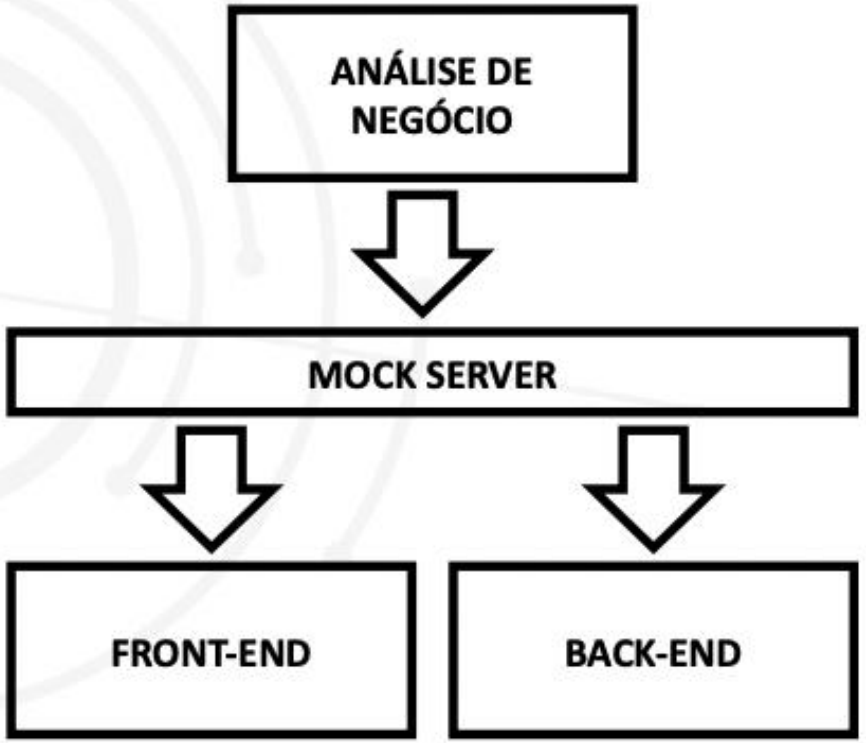
- Evite expor IDs sequenciais (use HashIDs por exemplo)
- Limite requisições, de forma configurável
- Não utilize logins por senha
- Utilize somente HTTPs

- Autorização e Autenticação
  - OAuth? OAuth2? JWT? Http-Basic?

- OAuth
  - Concebido para Autorização

- JWT
  - Concebido para Autenticação

Mock



Mock

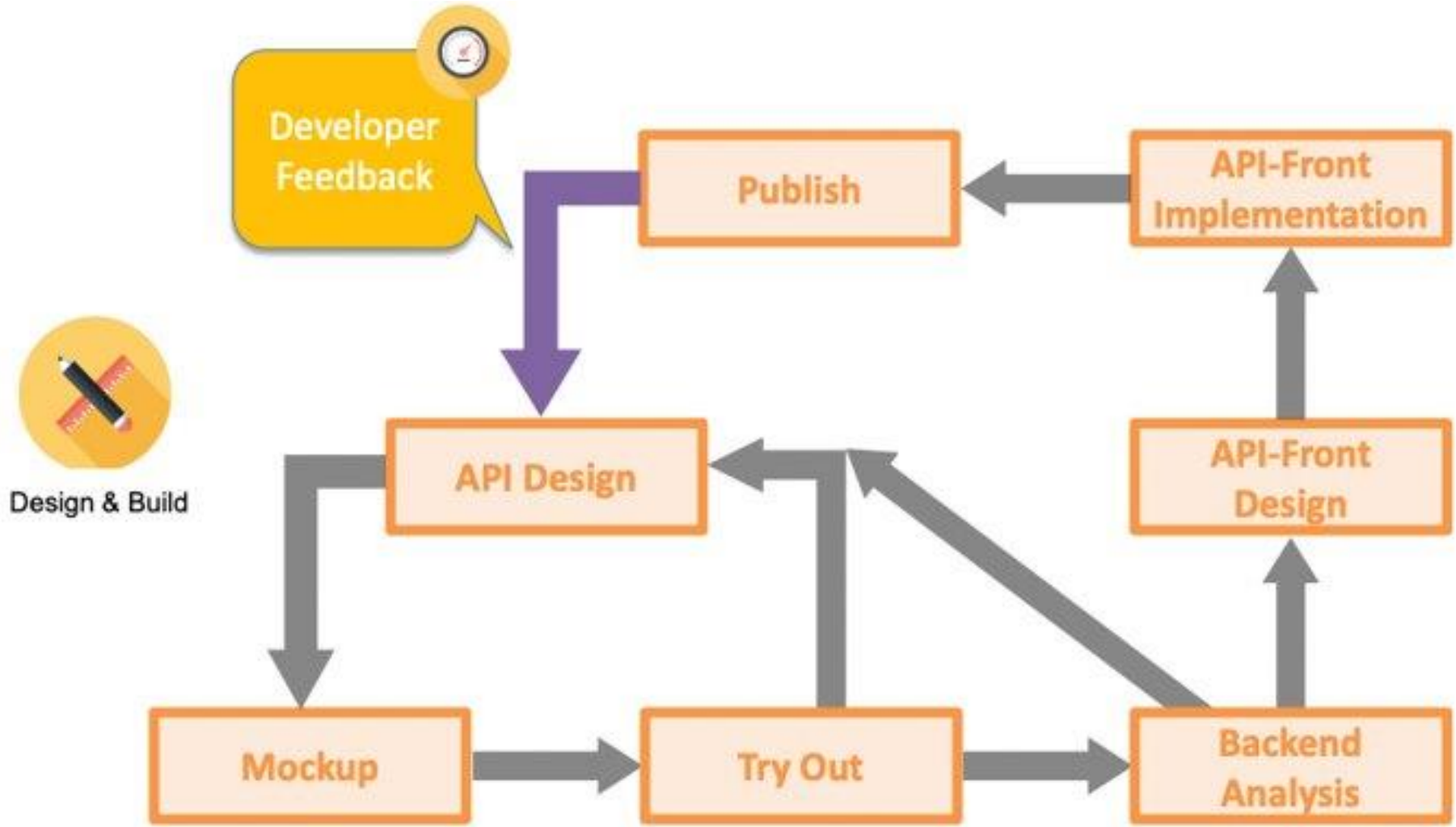
http://www.mock-server.com/

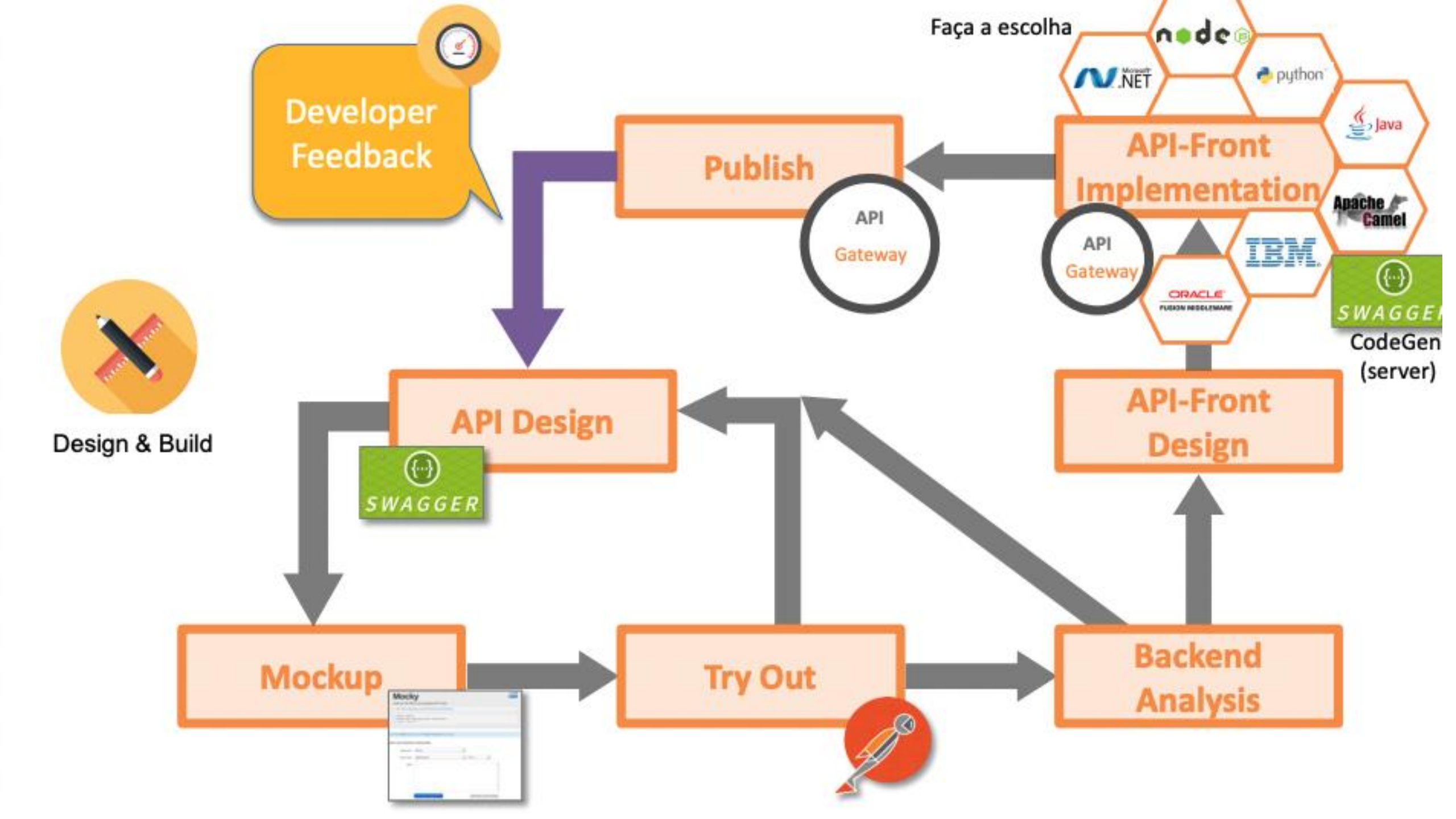
Audience & Insights Lab

```
new MockServerClient("127.0.0.1", 1080)
        .when(
                request()
                        .withMethod("POST")
                        .withPath("/login")
                        .withBody(exact("{username: 'foo', password: 'bar'}"))
        )
        .respond(
                response()
                        .withStatusCode(200)
                        .withHeaders(
                                new Header("Content-Type", "application/json; charset=utf-8"),
                        )
                        .withBody("{ code: 200 }")
        );
```

Amanhã logo cedo

Etapas


Developer Feedback
Publish
API-Front Implementation
Design & Build
API Design
API-Front Design
Mockup
Try Out
Backend Analysis


Developer Feedback
Publish
API Gateway
Faça a escolha
node
.NET
python
Java
API-Front Implementation
Apache Camel
IBM
ORACLE
API Gateway
SWAGGER
CodeGen (server)
Design & Build
API Design
SWAGGER
API-Front Design
Mockup
Try Out
Backend Analysis

Papéis

A Equipe

API Product Owner

- Planos
- Roadmap de APIs
- Modelo de negócio

Backend Architect

- Apoia o API Architect na análise bottom-up

API Architect

- Identifica as APIs
- Desenha as interfaces
- Garante o DX
- Data-mapping

API Operator

- Publica as APIs
- Monitora o uso
- Configura políticas/alertas

API Developer

- Constrói o API-Front

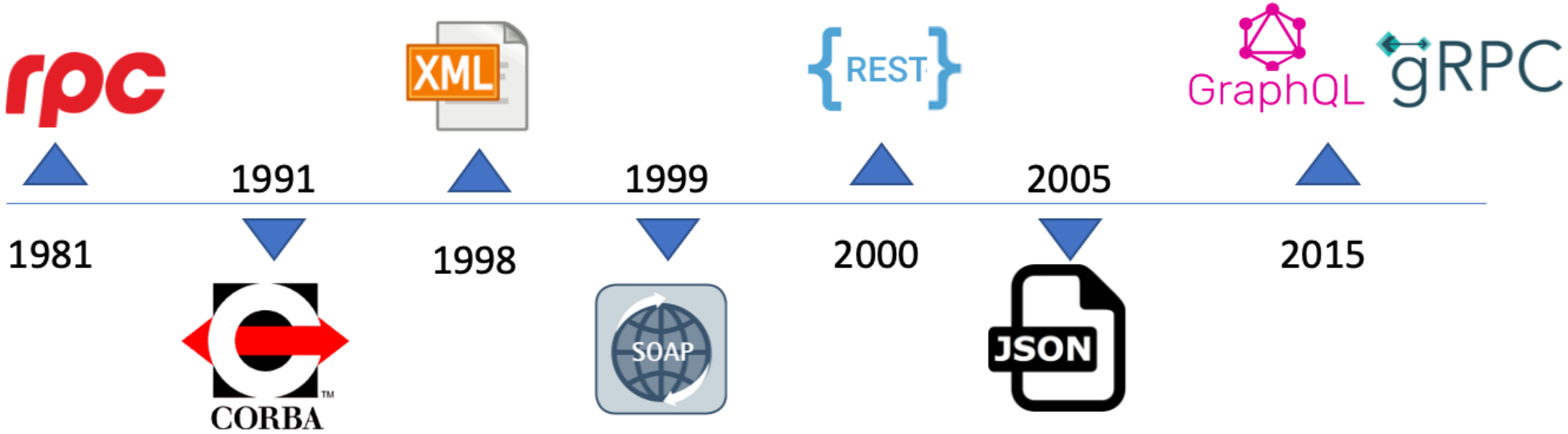
Developer Engagement

- Onboarding dos App Dev
- Suporte para os App Dev
- Melhoria do portal
- Feedbacks para API PO

Business Analyst

- Apoia no entendimento do negócio

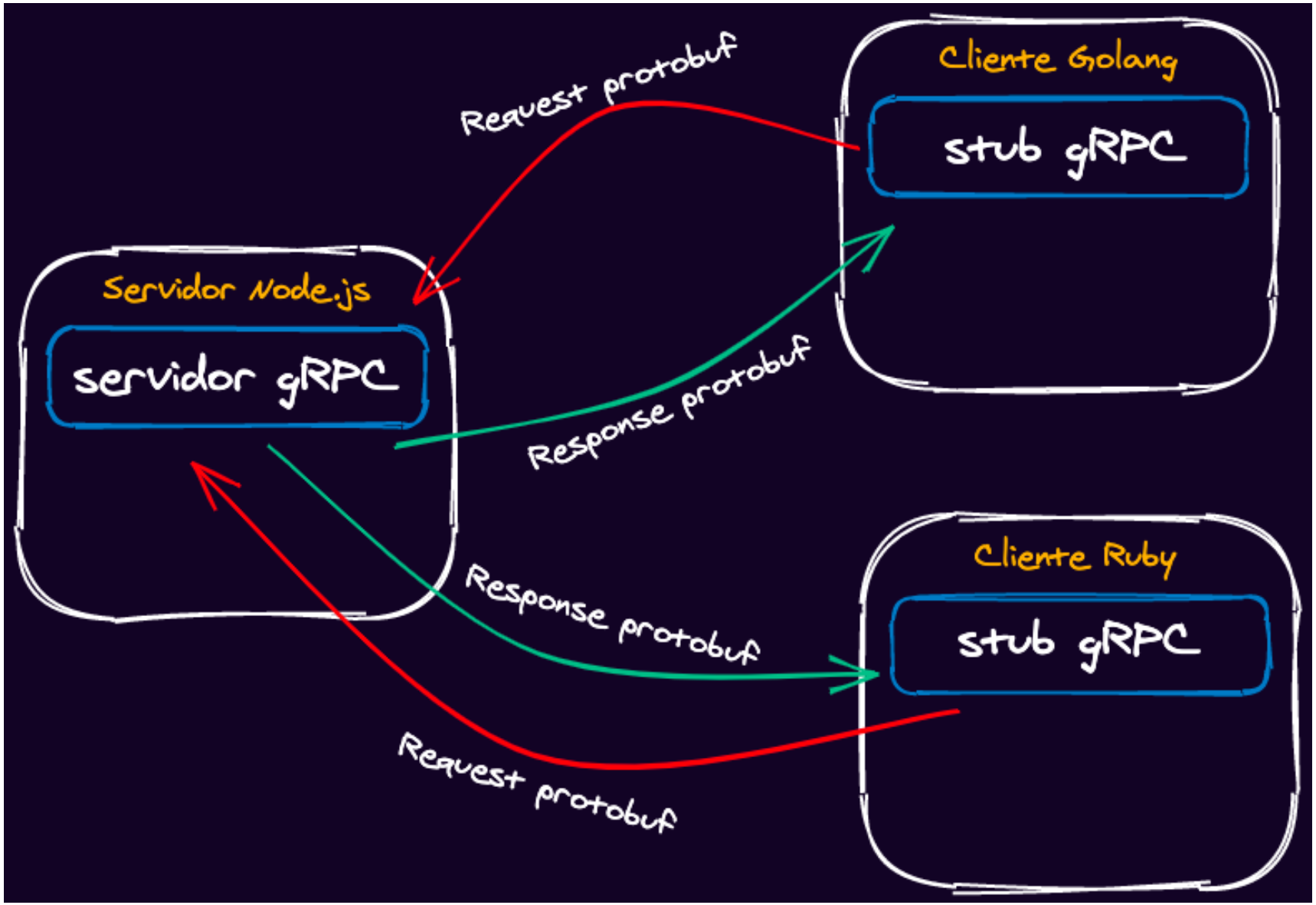
Tecnologias



GRPC

- O gRPC foi criado pela Google como um projeto de código aberto em 2015 como uma melhoria em uma arquitetura de comunicação chamada de RPC (Remote Procedure Call).

- Exige uma pequena mudança de paradigma em relação ao modelo ReST
- Curva de aprendizado inicial é mais complexa
- Não é uma especificação conhecida por muitos
- Mais leve e mais rápido por utilizar codificação binária e HTTP/2
- Funciona em muitas plataformas com pouco ou nenhum overhead
- O código é auto documentado


Cliente Golang
stub gRPC
Request protobuf
Servidor Node.js
servidor gRPC
Response protobuf
Cliente Ruby
stub gRPC
Response protobuf
Request protobuf

GRPC

- Faça um projeto GRPC

REST

- REST like RPC (POST e GET) muito utilizado no início do AJAX
- RESTFUL (GET - 200, POST - 201, PUT - 204 e DELETE - 204)

- Necessidade do entendimento do HTTP
- Granularidade baixa ou média
- Diversos endpoints

- https://www.infoq.com/br/articles/nivelando-sua-rest-api/

Graphql

- Facebook 2012
- 2015 liberado como Open Source

- Somente um 1 endpoint
- Baseado no uso de schema (como SOAP e RPC)
- Granularidade dinâmica (baixa)

- Uma linguagem de consulta
  - Cliente pede o que precisa e servidor retorna
  - Sem conceito de recursos (endpoint)

Graphql

- Crie uma API Graqhql que retorna um CEP.