globo.com

Segurança em APIs externas de uso *mobile*

GTS 29 - Foz do Iguaçu, 26 de maio de 2017

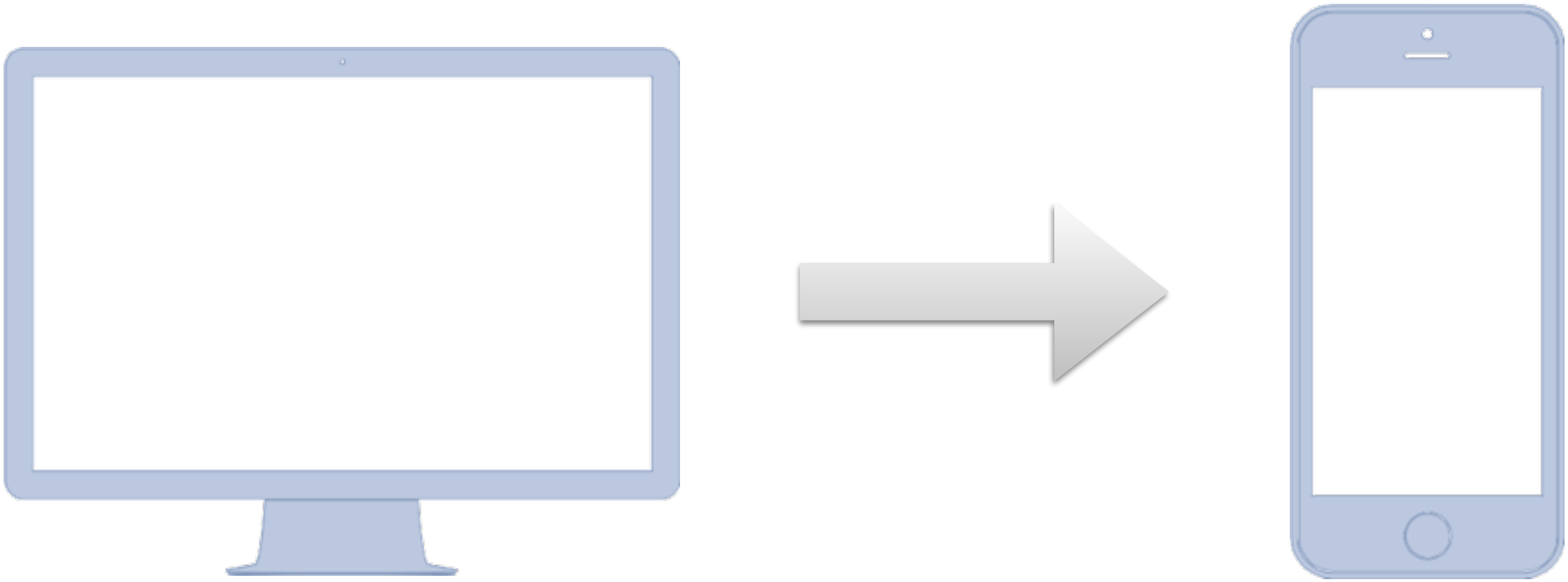
Agenda

- Web vs *Mobile*
- Estado atual do desenvolvimento *mobile*
- Como os aplicativos são desenvolvidos
- Protocolos/Técnicas utilizadas
- Armazenamento/distribuição de credenciais
- Proposta para autenticação *mobile*

Web vs Mobile

- Security User Interface
- Same-Origin Policy
- Cookie Security
- API Security (CORS,…)
- Browser Security Design

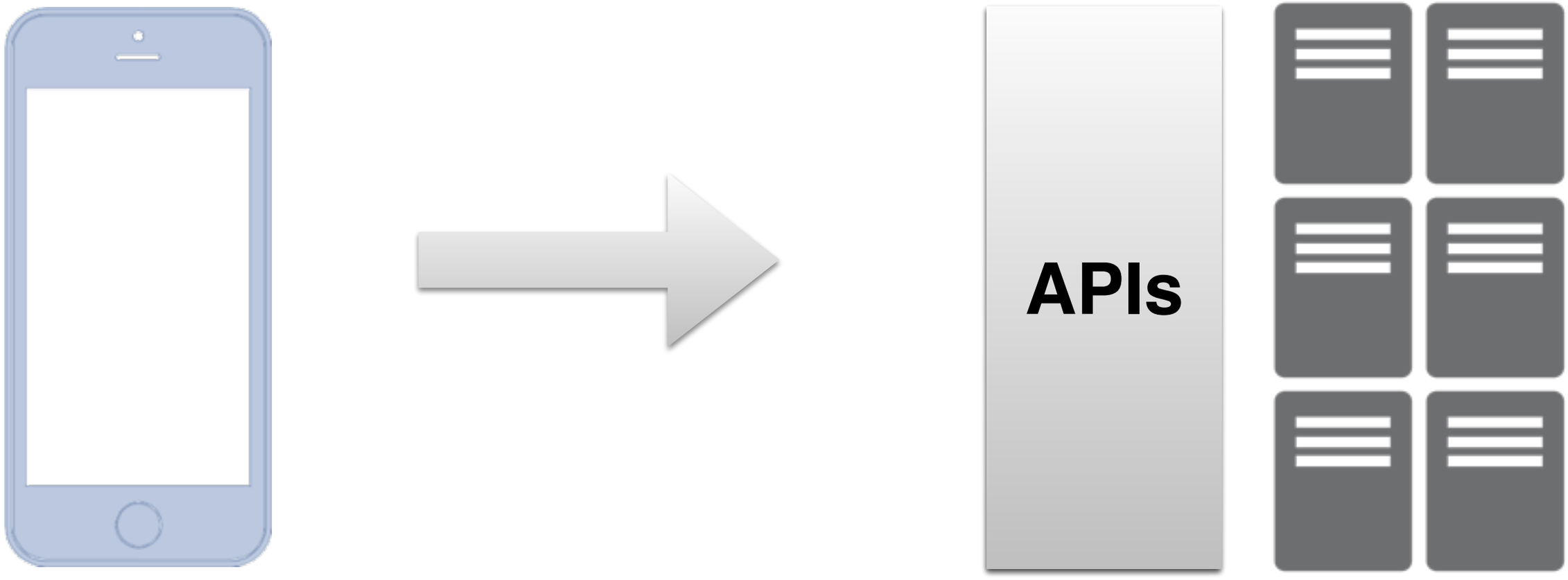
Web vs Mobile



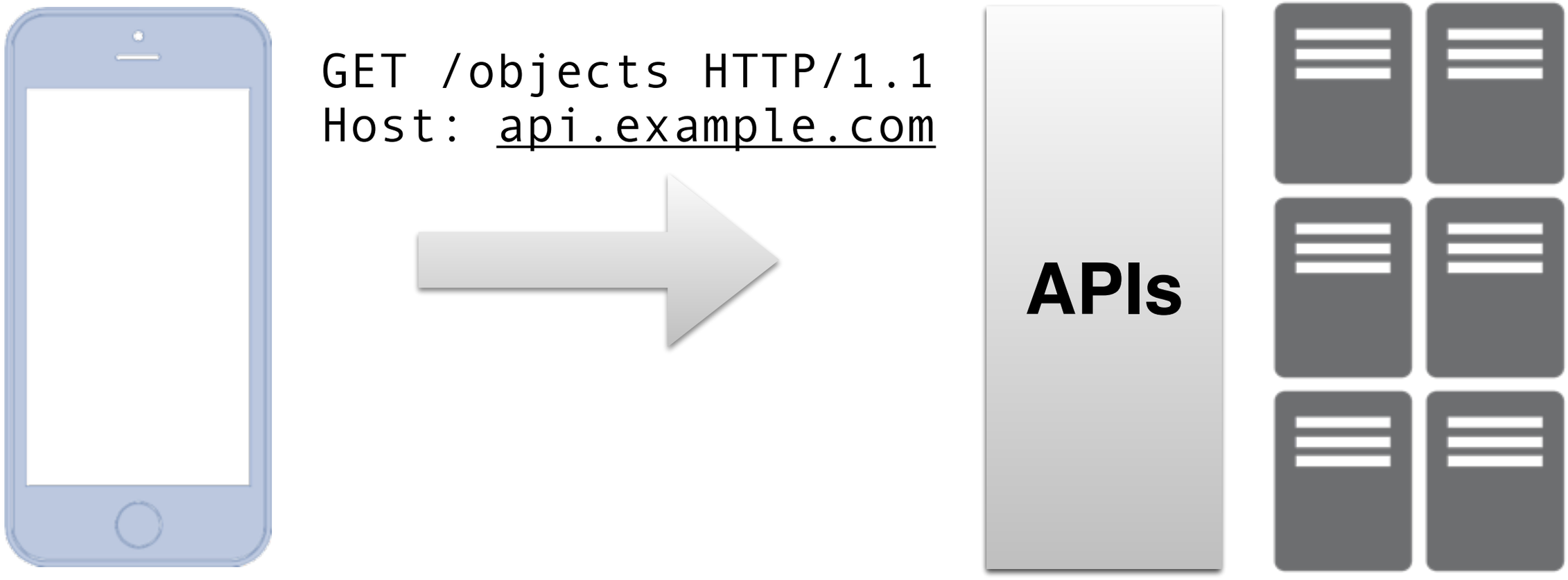
- Security User Interface
- Same-Origin Policy
- Cookie Security
- API Security (CORS,…)
- Browser Security Design

- ~~Security User Interface~~
- ~~Same-Origin Policy~~
- ~~Cookie Security~~
- ~~API Security (CORS,…)~~
- ~~Browser Security Design~~

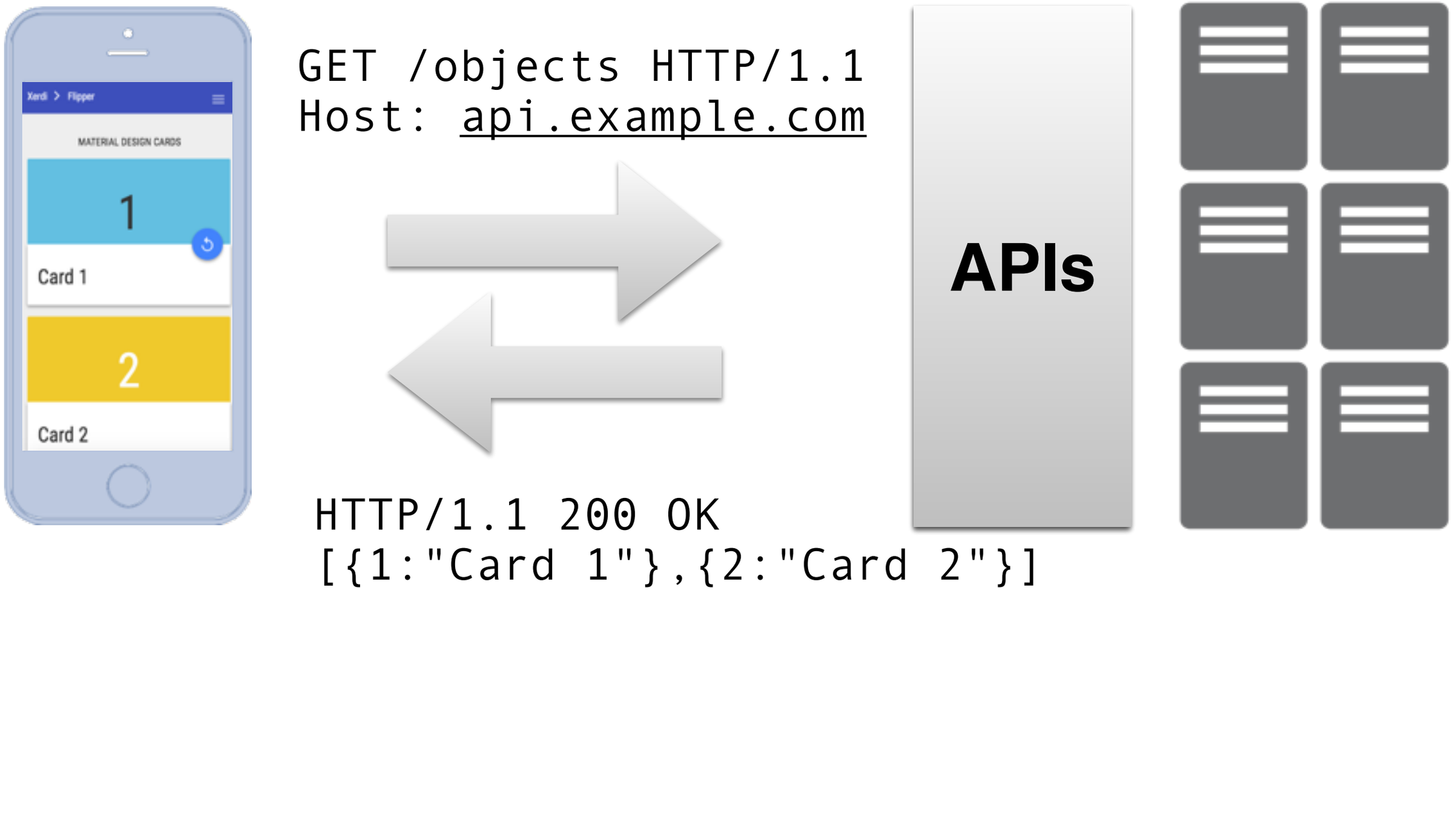
Desenvolvimento *mobile*



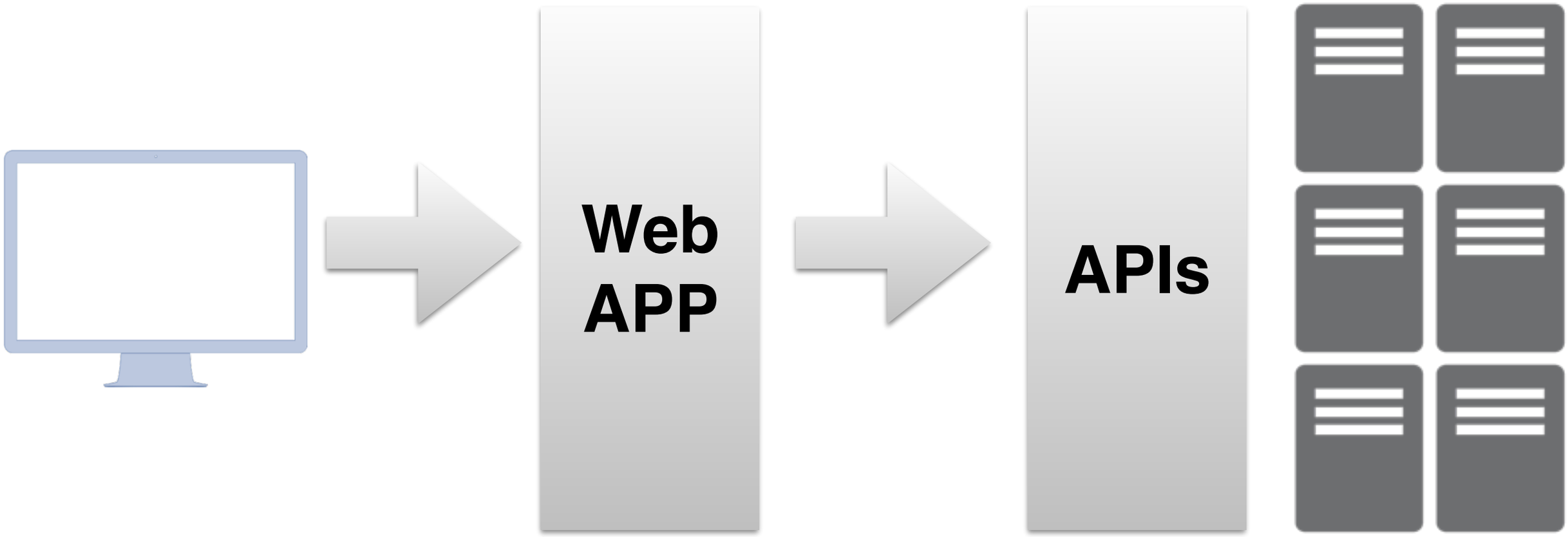
Desenvolvimento *mobile*



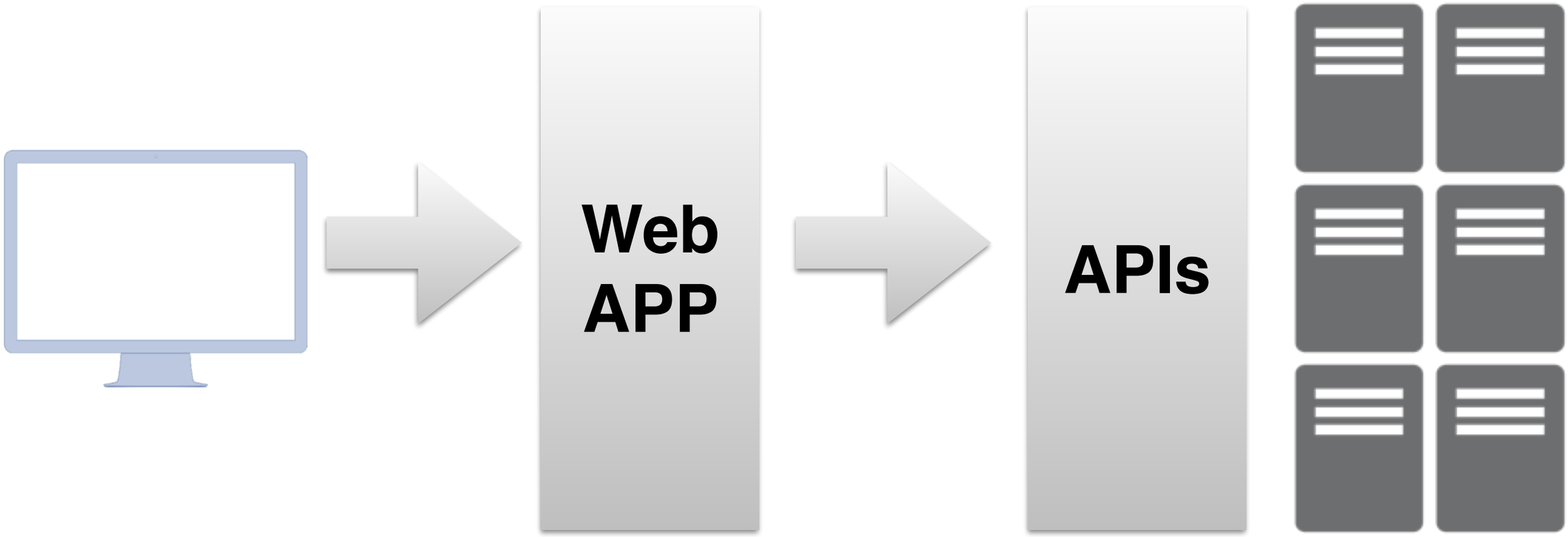
Desenvolvimento *mobile*



Desenvolvimento *web*

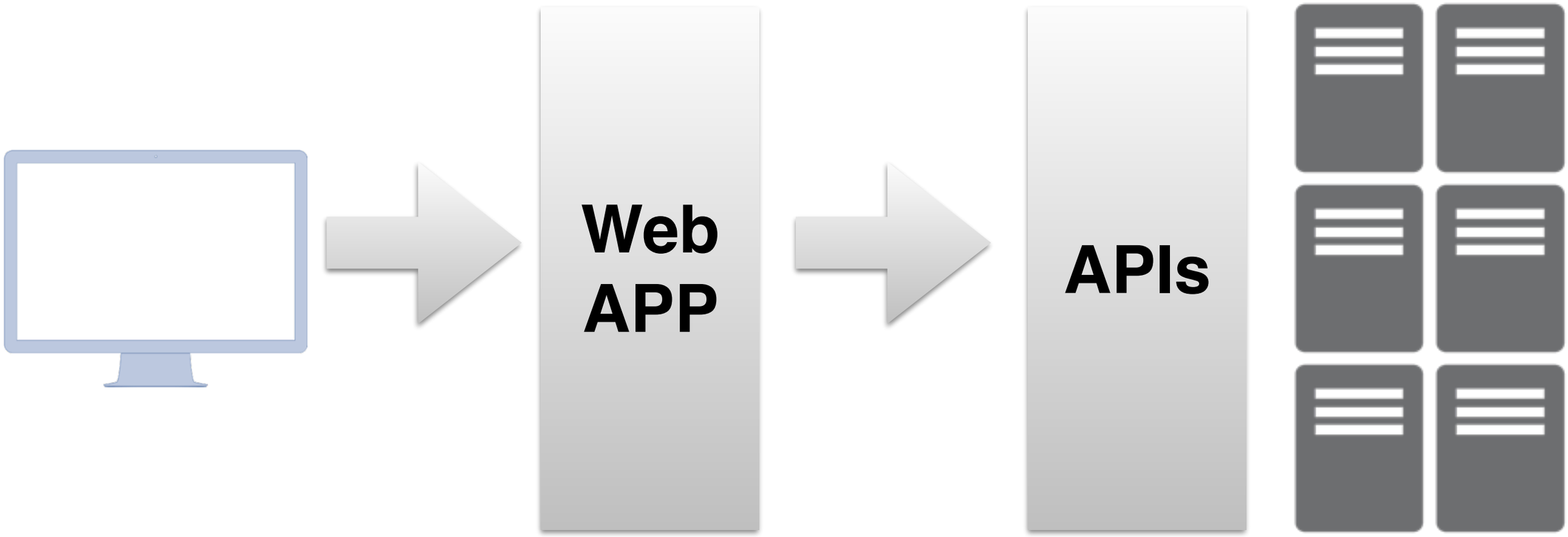


Desenvolvimento *web*



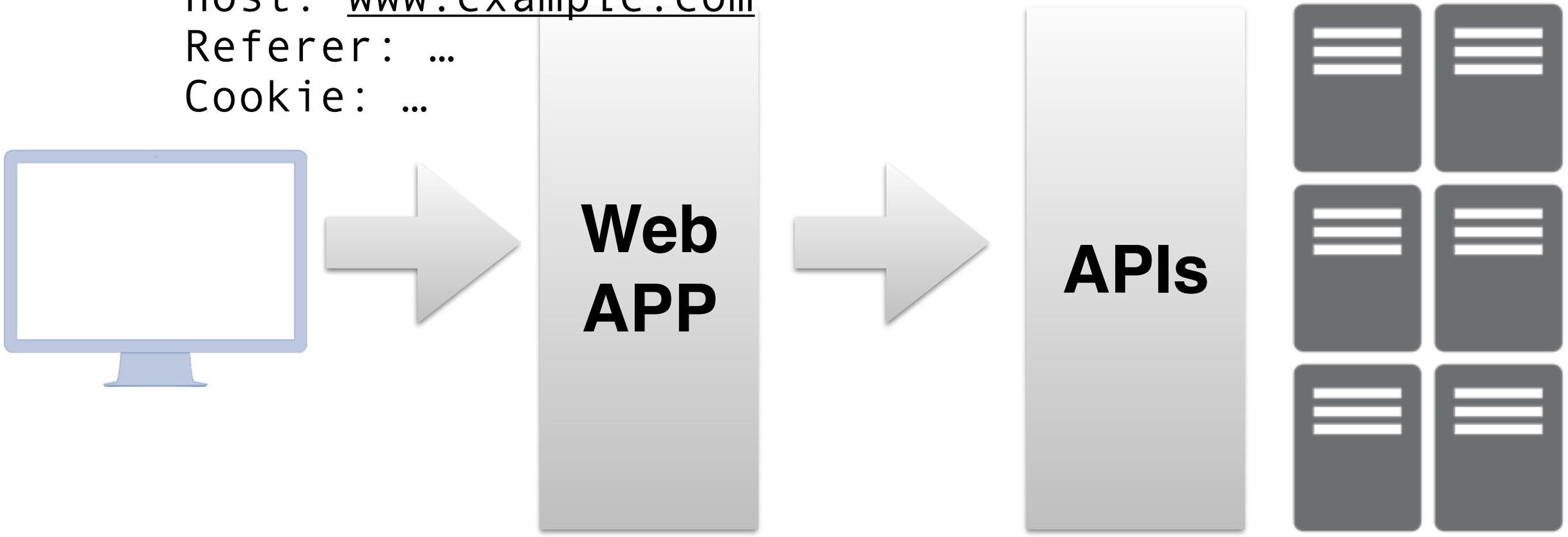
Modelo de Segurança para Web

Desenvolvimento *web*



Desenvolvimento *web*

```
GET / HTTP/1.1
Host: www.example.com
Referer: …
Cookie: …
```



Desenvolvimento *web*

```
GET / HTTP/1.1
Host: www.example.com
Referer: …
Cookie: …
```

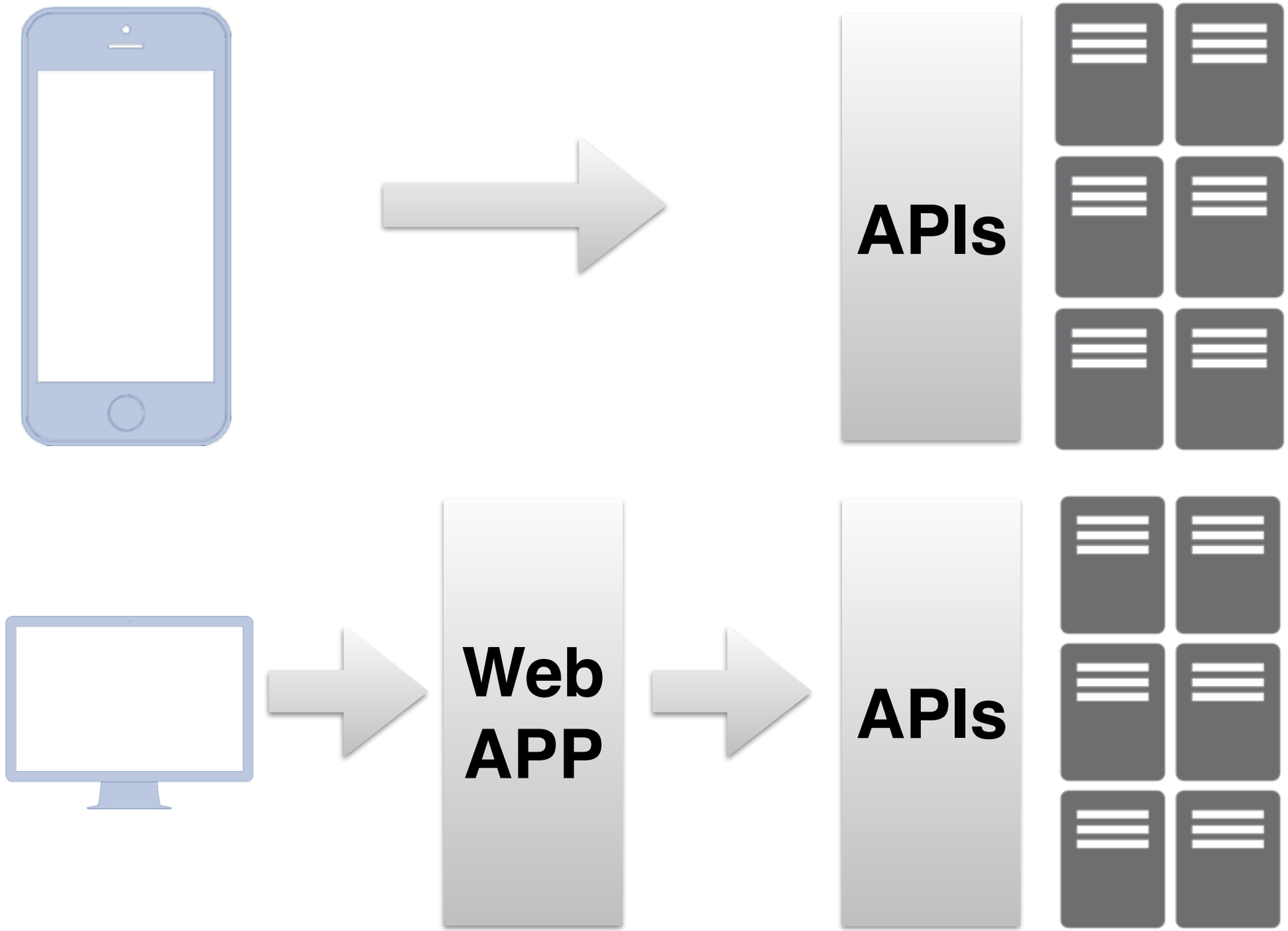

```
HTTP/1.1 200 OK
<html>
...
<div class="well">
<h1>Card 1</h1>
</div>
...
```

Desenvolvimento *web*

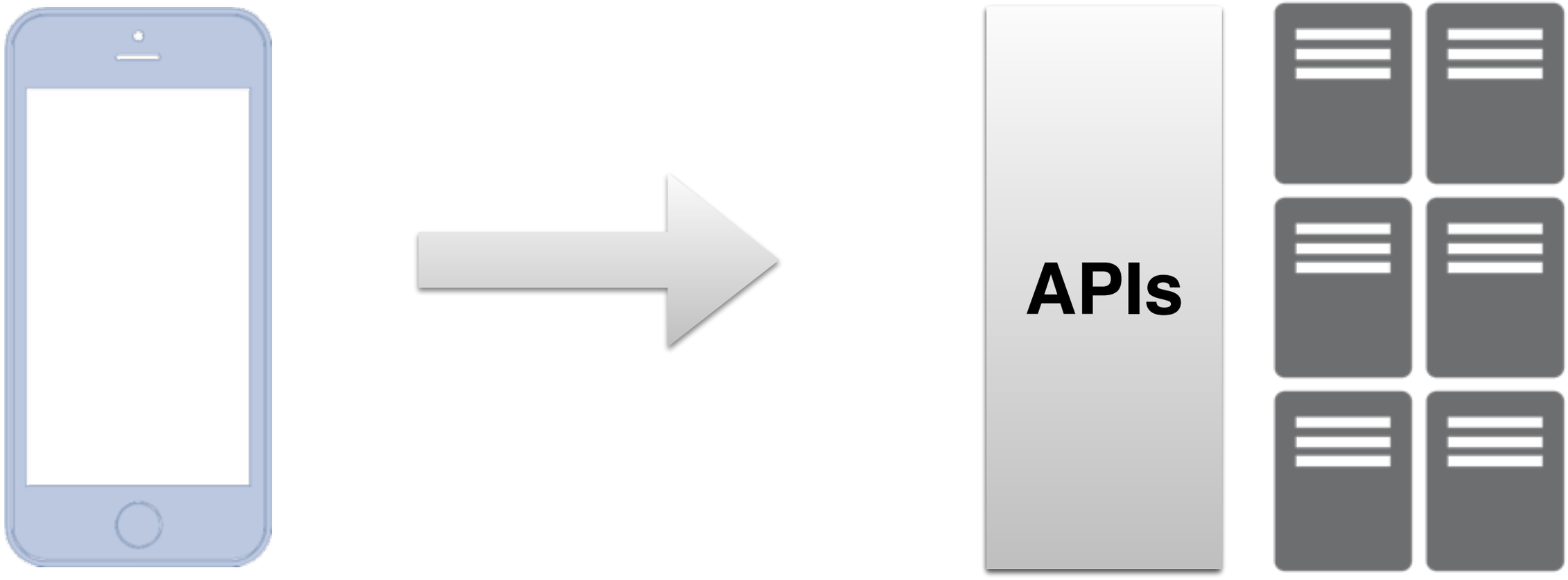
```
GET / HTTP/1.1
Host: www.example.com
Referer: …
Cookie: …
```


```
HTTP/1.1 200 OK
<html>
...
<div class="well">
<h1>Card 1</h1>
</div>
...
```

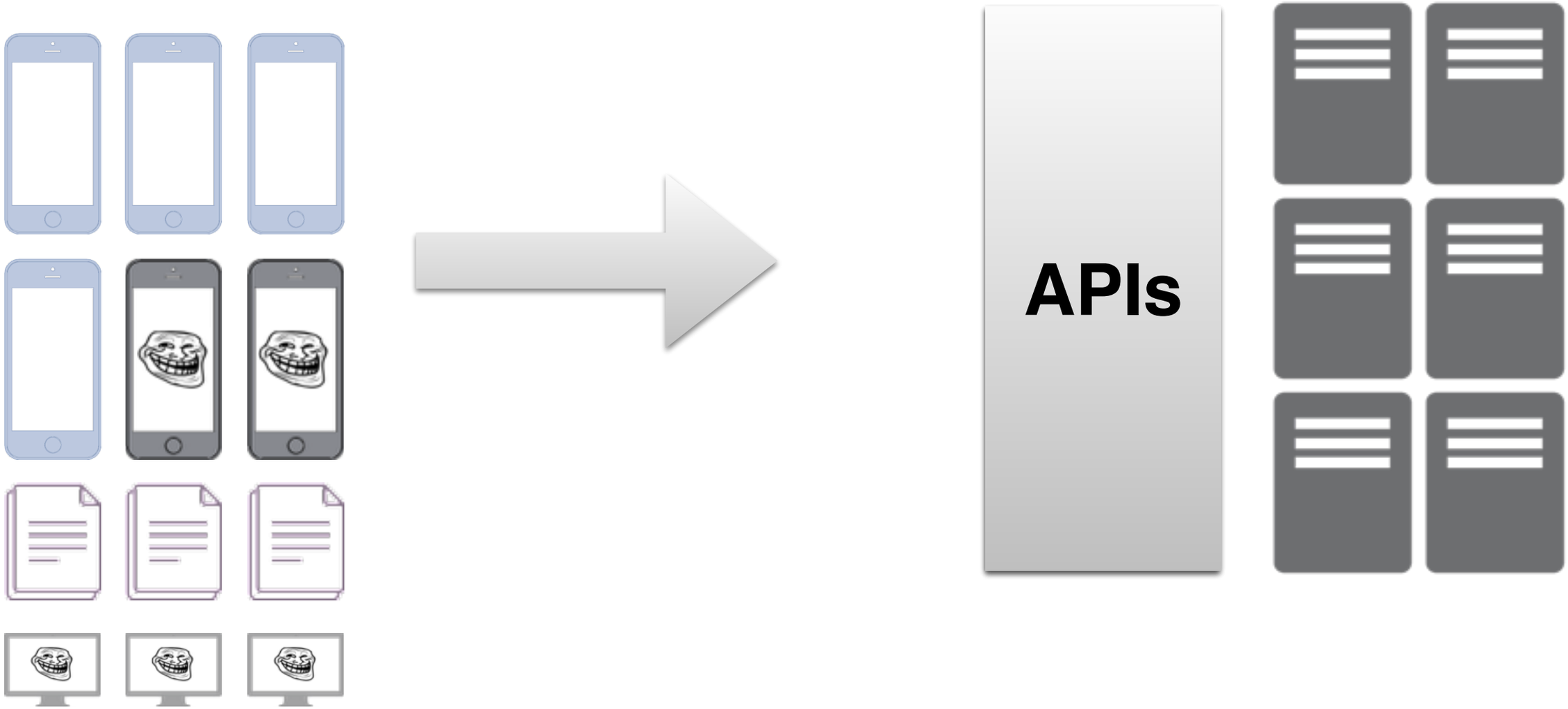
Web vs Mobile



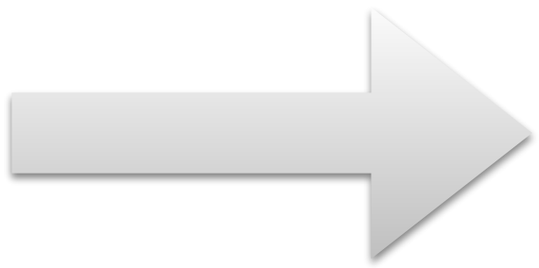
Desenvolvimento *mobile*

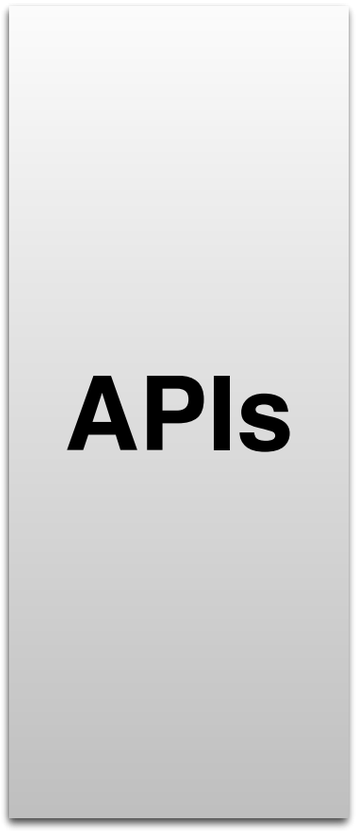


Preocupações

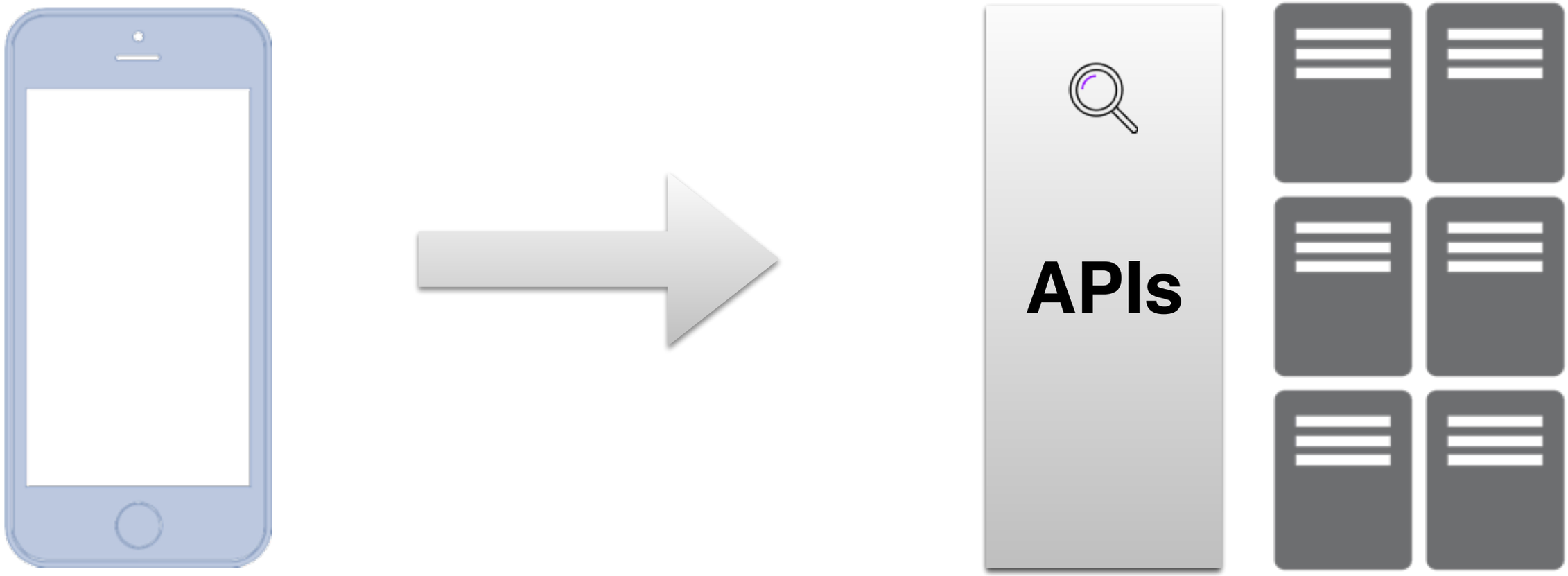


Preocupações



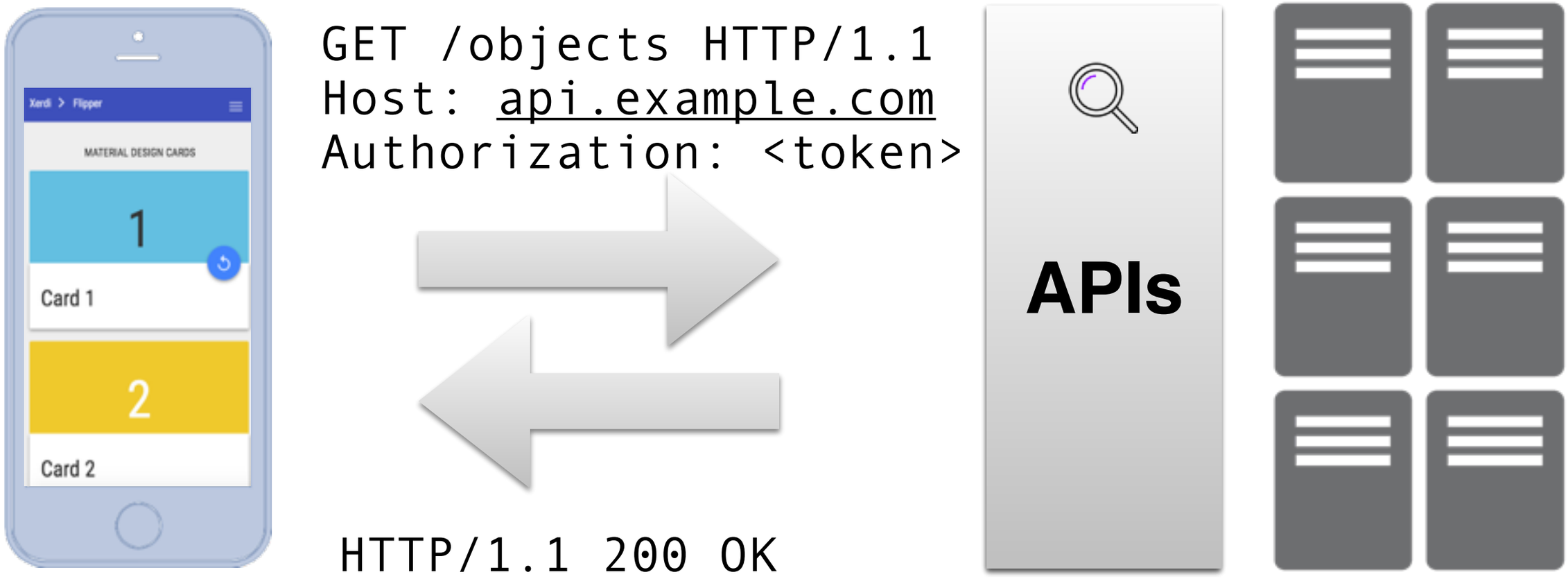


Protocolos/Técnicas utilizadas

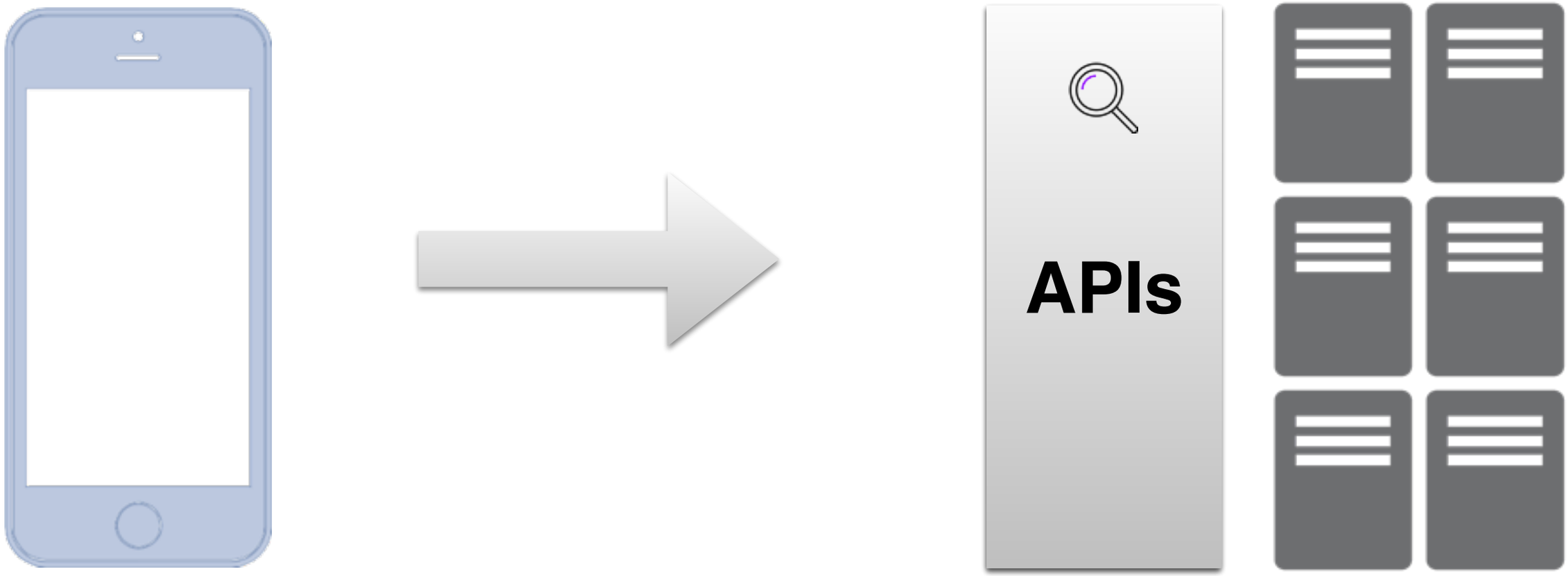


Protocolos/Técnicas utilizadas

Protocolos/Técnicas utilizadas

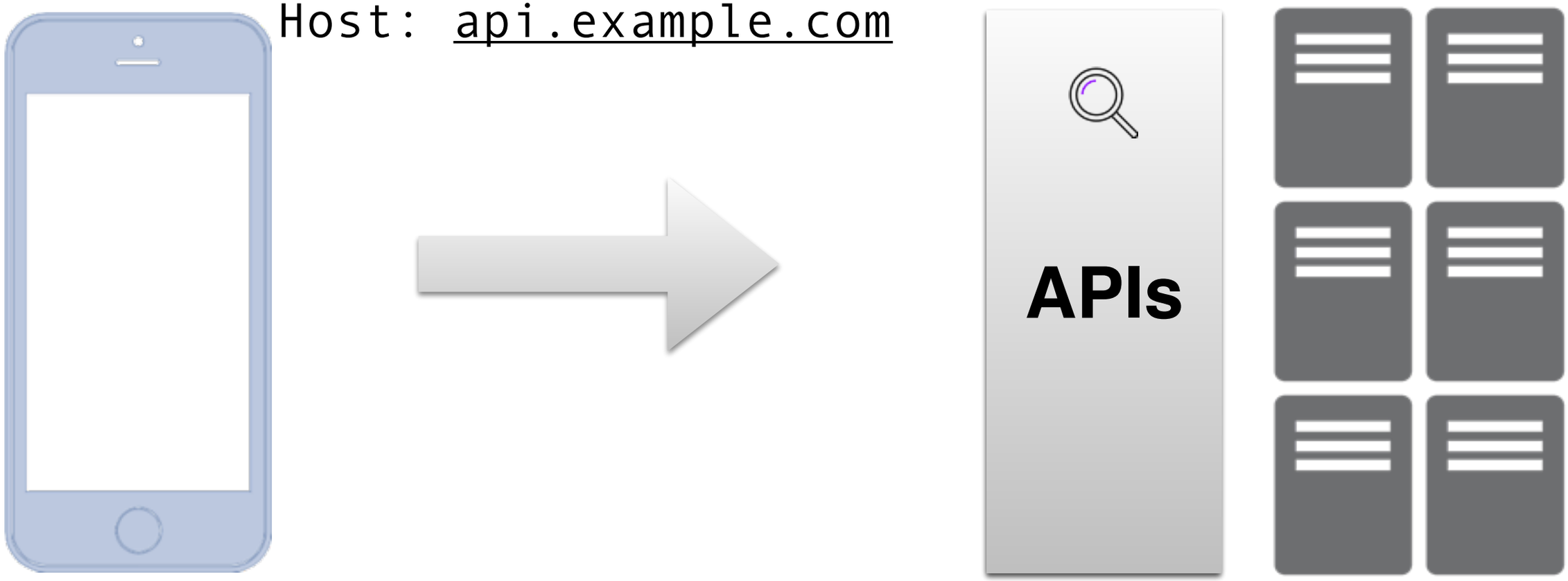


Protocolos/Técnicas utilizadas



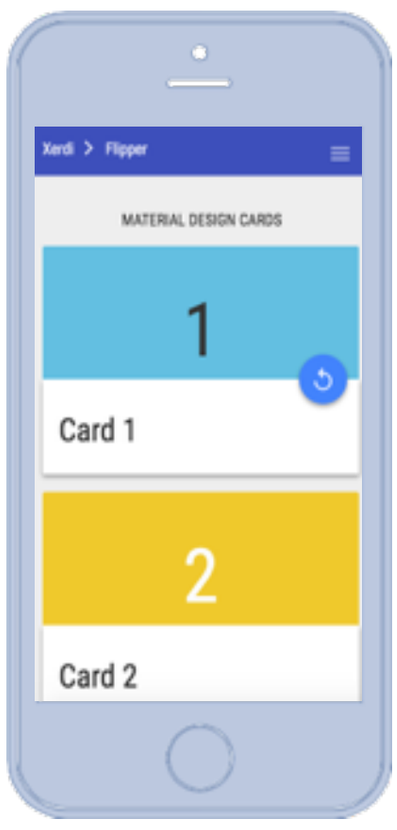
Protocolos/Técnicas utilizadas

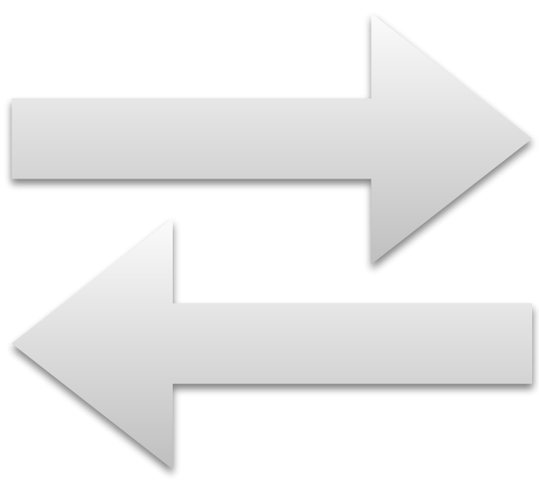
```
POST /token?grant_type=client_credentials&
 client_id=CLIENT_ID&
 client_secret=CLIENT_SECRET HTTP/1.1
Host: api.example.com
```



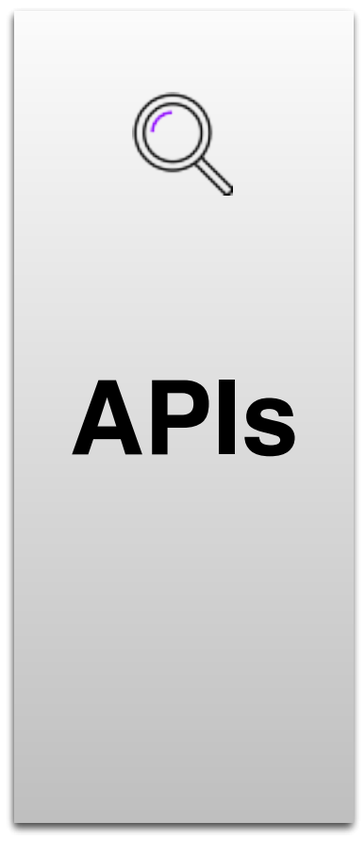
Protocolos/Técnicas utilizadas

```
POST /token?grant_type=client_credentials&
 client_id=CLIENT_ID&
 client_secret=CLIENT_SECRET HTTP/1.1
Host: api.example.com
```





```
HTTP/1.1 200 OK
{“token":<token>}
```



They reverse engineered 16k apps; here's what we'd fix

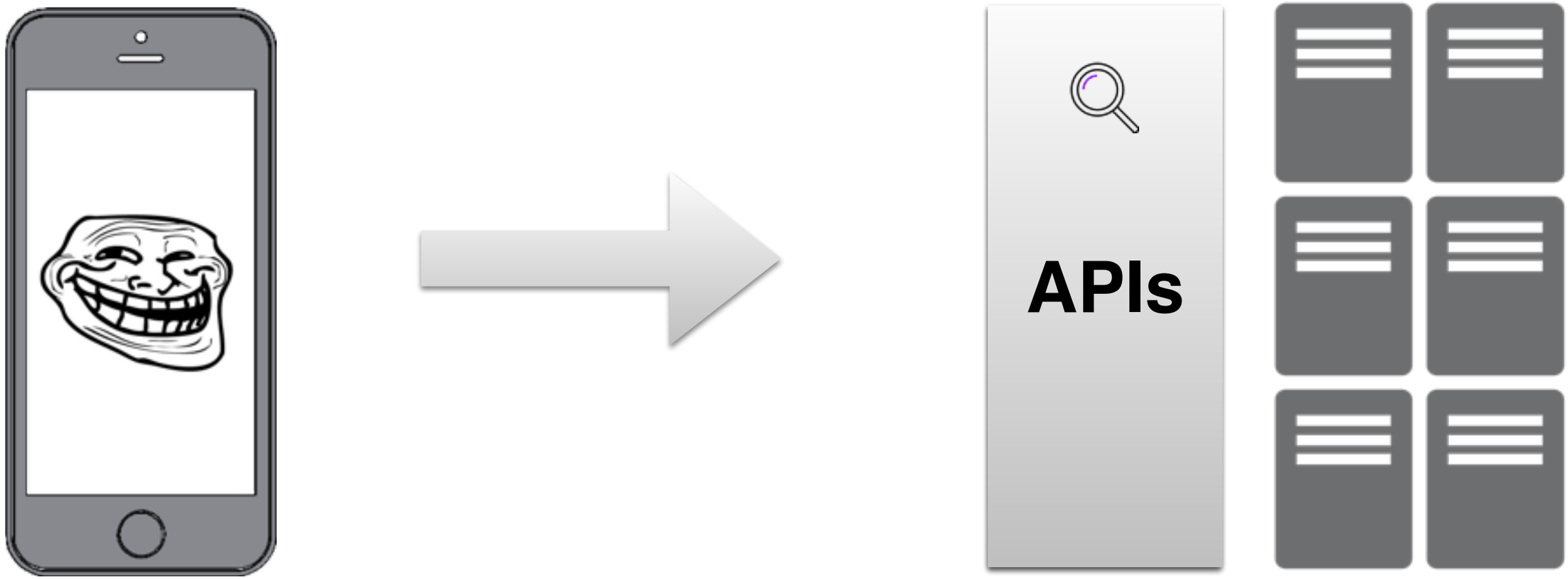
Recommendations to secure mobile API keys and secrets

In "We reverse engineered 16k apps, here's what we found", Fallible used their online reverse engineering tool to probe Android apps looking for secrets. They tested over 16,000 apps and found that roughly 2500 of them contained API keys and/or secrets. If their tool can find them, so can people looking to exploit your apps.

Let's assume that you are developing an Android or iOS mobile app. Your backend app server nicely exposes information through a RESTful API. To authenticate your mobile app, an API key is passed with every API call. This key, simply formatted as <id>:<secret>, enables the backend server to easily identify your app and validate the shared secret. Your mobile app also directly calls a few 3rd party APIs, which each use a similar key strategy.

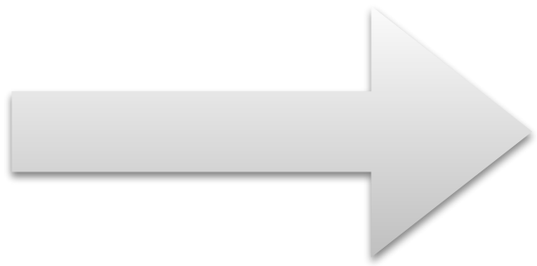
Fonte: https://hackernoon.com/they-reverse-engineered-16k-apps-heres-what-we-d-fix-67e9eeceefb5

Protocolos/Técnicas utilizadas



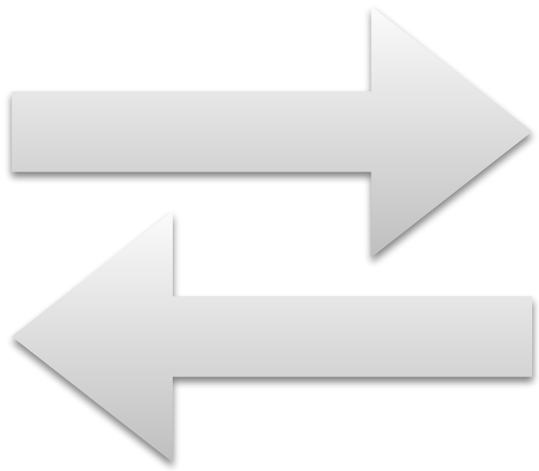
Protocolos/Técnicas utilizadas

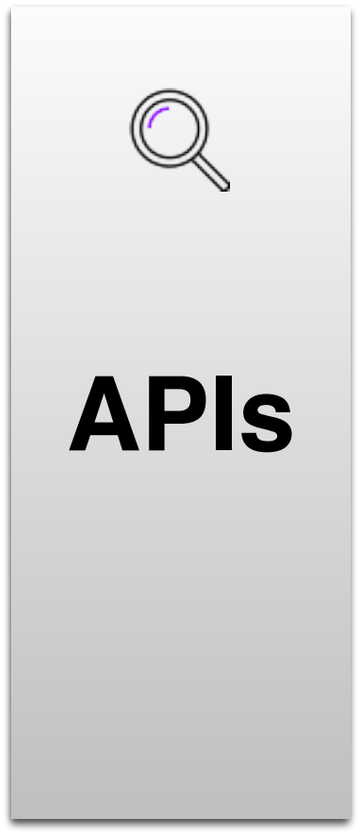
```
POST /token?grant_type=client_credentials&
 client_id=CLIENT_ID&
 client_secret=CLIENT_SECRET HTTP/1.1
Host: api.example.com
```



Protocolos/Técnicas utilizadas

```
POST /token?grant_type=client_credentials&
 client_id=CLIENT_ID&
 client_secret=CLIENT_SECRET HTTP/1.1
Host: api.example.com
```

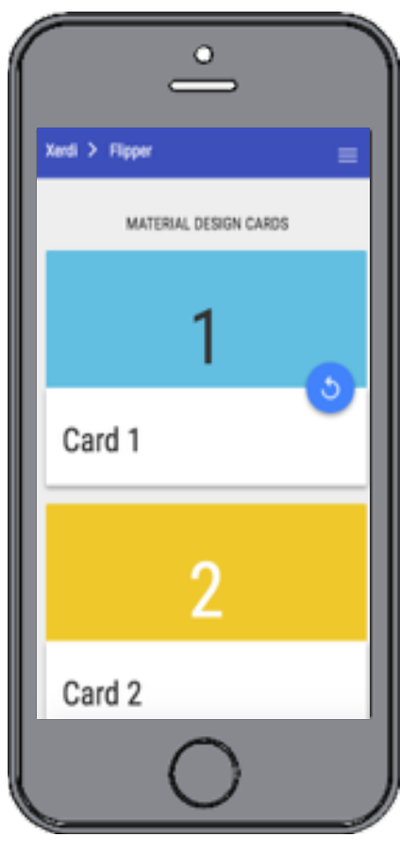


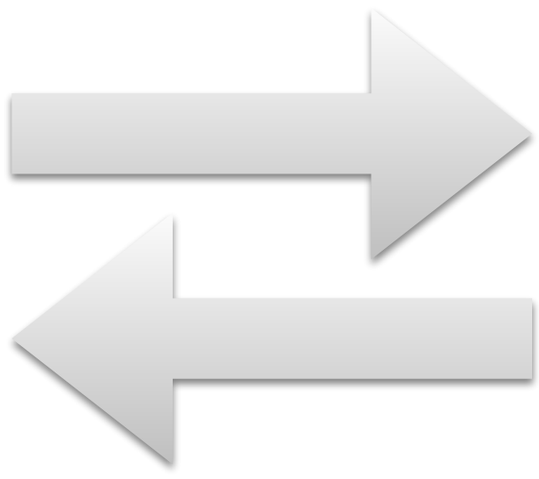


```
HTTP/1.1 200 OK
{“token":<token>}
```

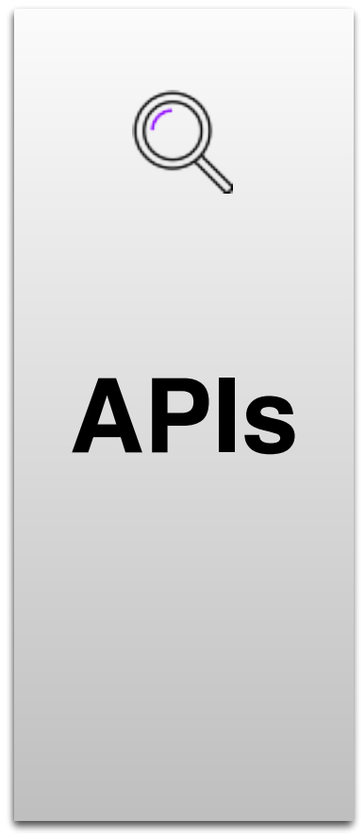
Protocolos/Técnicas utilizadas

```
POST /token?grant_type=client_credentials&
 client_id=CLIENT_ID&
 client_secret=CLIENT_SECRET HTTP/1.1
Host: api.example.com
```





```
HTTP/1.1 200 OK
{“token":<token>}
```



Como autenticar a aplicação *mobile* sem armazenar credenciais nela?

Best Practice for storing private API keys in Android

I am developing an app and I use several third party APIs and SDKs such as Dropbox and Google Drive. These libraries requiers API keys. A private and a public one.

Currently I have sth like this:

```
public class DropboxService  {

    private final static String APP_KEY = "jk433g34hg3";
    private final static String APP_SECRET = "987dwdqwdqw90";
    private final static AccessType ACCESS_TYPE = AccessType.DROPBOX;

    // SOME MORE CODE HERE

}
```


Fonte: https://stackoverflow.com/questions/14570989/best-practice-for-storing-private-api-keys-in-android

The App-Secret key should be kept private - but when releasing the app they can be reversed by

196

1. As it is, your compiled application contains the key strings, but also the constant names APP_KEY and APP_SECRET. Extracting keys from such self-documenting code is trivial, for instance with the standard Android tool dx.
2. You can apply ProGuard. It will leave the key strings untouched, but it will remove the constant names. It will also rename classes and methods with short, meaningless names, where ever possible. Extracting the keys then takes some more time, for figuring out which string serves which purpose.

   Note that setting up ProGuard shouldn't be as difficult as you fear. To begin with, you only need to enable ProGuard, as documented in project.properties. If there are any problems with third-party libraries, you may need to suppress some warnings and/or prevent them from being obfuscated, in proguard-project.txt. For instance:

   ```
   -dontwarn com.dropbox.**
   -keep class com.dropbox.** { *; }
   ```

   This is a brute-force approach; you can refine such configuration once the processed application works.
3. You can obfuscate the strings manually in your code, for instance with a Base64 encoding or preferably with something more complicated; maybe even native code. A hacker will then have to statically reverse-engineer your encoding or dynamically intercept the decoding in the proper place.
4. You can apply a commercial obfuscator, like ProGuard's specialized sibling DexGuard. It can additionally encrypt/obfuscate the strings and classes for you. Extracting the keys then takes even more time and expertise.
5. You might be able to run parts of your application on your own server. If you can keep the keys there, they are safe.

In the end, it's an economic trade-off that you have to make: how important are the keys, how much time or software can you afford, how sophisticated are the hackers who are interested in the keys, how much time will they want to spend, how much worth is a delay before the keys are hacked, on what scale will any successful hackers distribute the keys, etc. Small pieces of information like keys

Fonte: https://stackoverflow.com/questions/14570989/best-practice-for-storing-private-api-keys-in-android

stackoverflow Questions Jobs Documentation BETA Tags Users Search... Log In

Join the Stack Overflow Community

Stack Overflow is a community of 7.2 million programmers, just like you, helping each other. Join them; it only takes a minute:

Sign up

Secure keys in iOS App scenario, is it safe?

Ask Question

asked 4 years, 3 months ago

viewed 9204 times

active 2 years, 3 months ago

14

I am trying to hide 2 secrets that I am using in one of my apps.

As I understand the keychain is a good place but I can not add them before I submit the app.

I thought about this scenario -

- Pre seed the secrets in my app's CoreData Database by spreading them in other entities to obscure them. (I already have a seed DB in that app)

16

- As the app launches for the first time, generate and move then to keys to the keychain

BLOG

How Stack Overflow Flippe on HTTPS

Stack Overflow: Helping O Developers Exit Vim

Fonte: https://stackoverflow.com/questions/14778429/secure-keys-in-ios-app-scenario-is-it-safe

Preocupações

- No mundo *mobile*, é necessário autenticar:
  - usuário
  - serviço
  - **meio de utilização**

Preocupações

- No mundo *mobile*, é necessário autenticar:
  - usuário
  - serviço

  } autenticação entre pares tradicional
  - **meio de utilização**

Preocupações

- No mundo *mobile*, é necessário autenticar:
  - usuário
  - serviço

  } autenticação entre pares tradicional

  - **meio de utilização** }

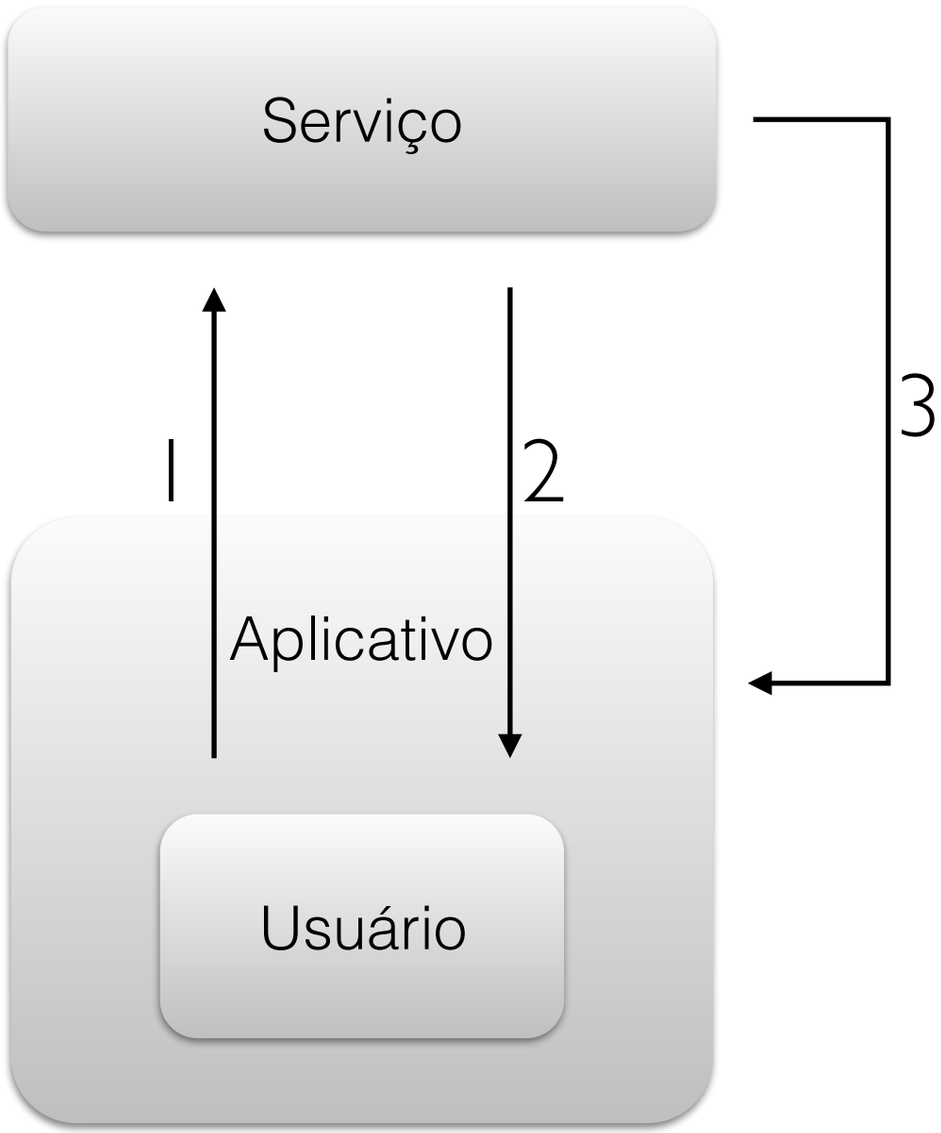
Preocupações

- No mundo *mobile*, é necessário autenticar:
  - usuário
  - serviço

  } autenticação entre pares tradicional

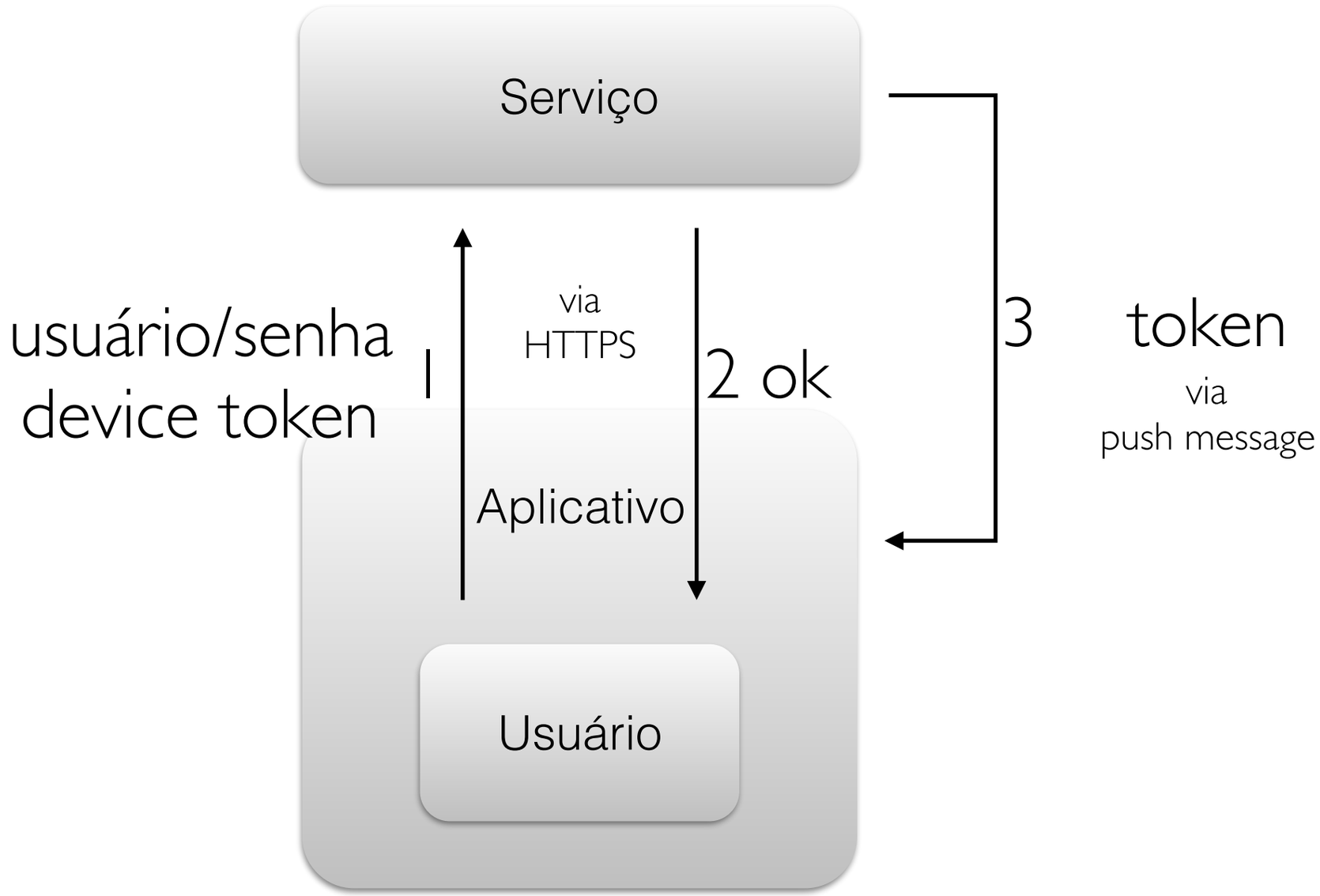
  - **meio de utilização** } esse novo requisito exige uma abordagem nova de autenticação

Arquitetura proposta

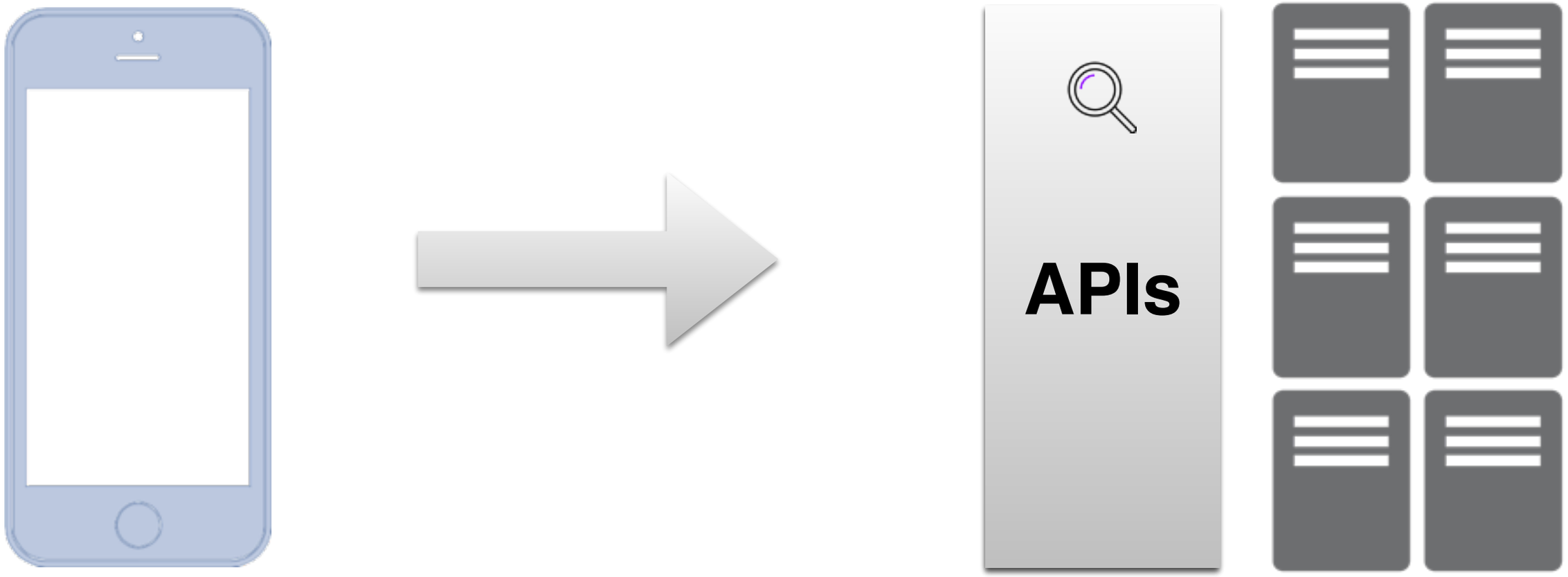


- 1 - O usuário solicita uma sessão para o serviço
- 2 - O serviço responde a requisição do usuário (se a autenticação for bem sucedida) **sem** um token de sessão, somente confirmando o recebimento dos dados
- 3 - O serviço envia ao aplicativo(*out-of-band*) o token de sessão

Arquitetura proposta

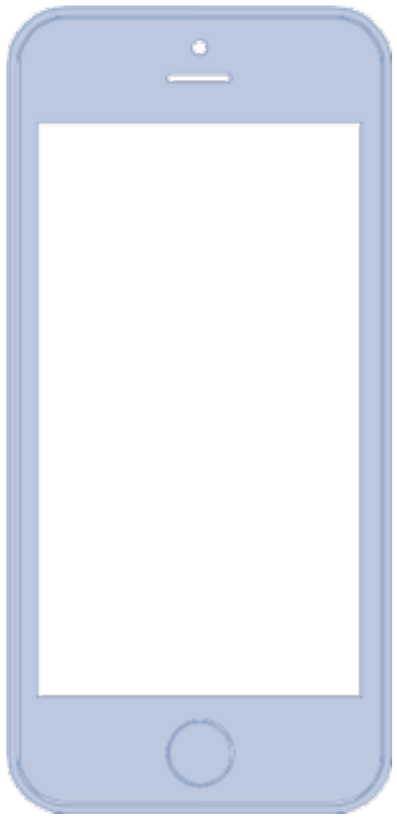


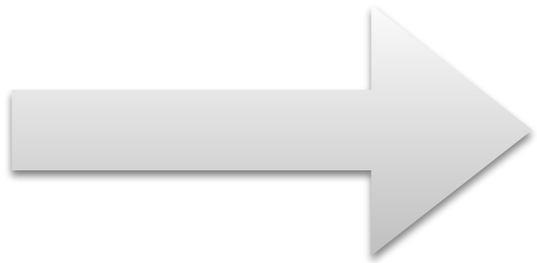
Arquitetura proposta

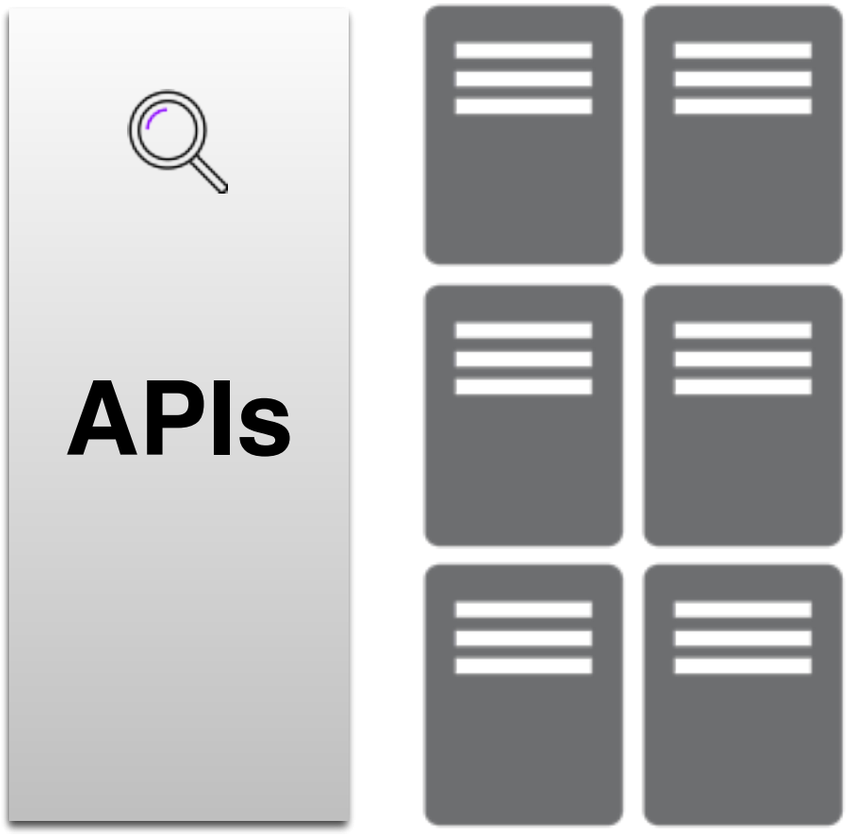


Arquitetura proposta

```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```

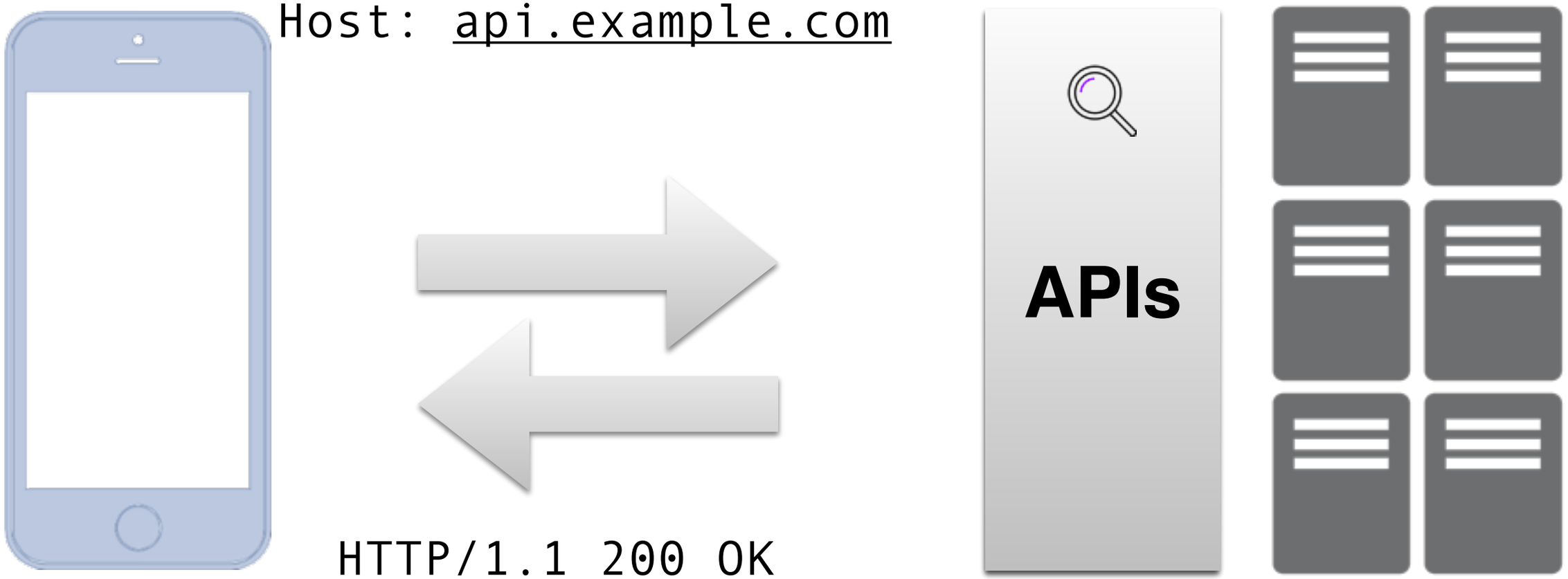






Arquitetura proposta

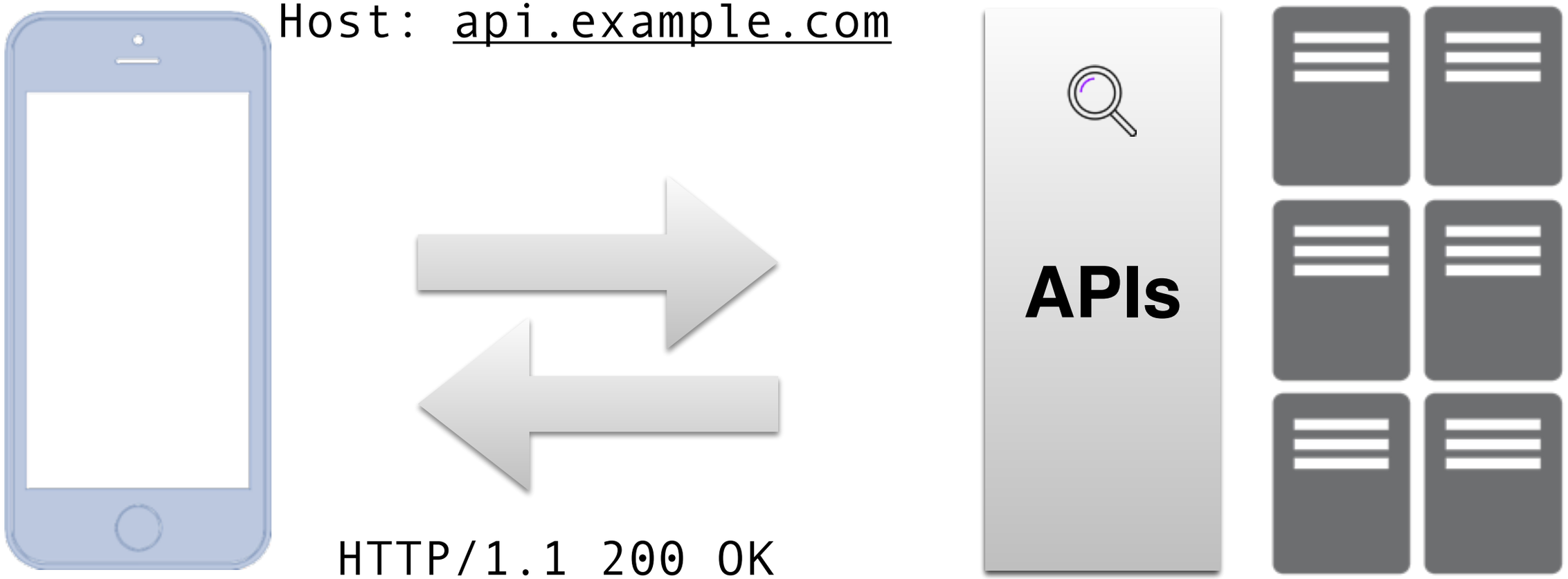
```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```




```
HTTP/1.1 200 OK
```

Arquitetura proposta

```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```

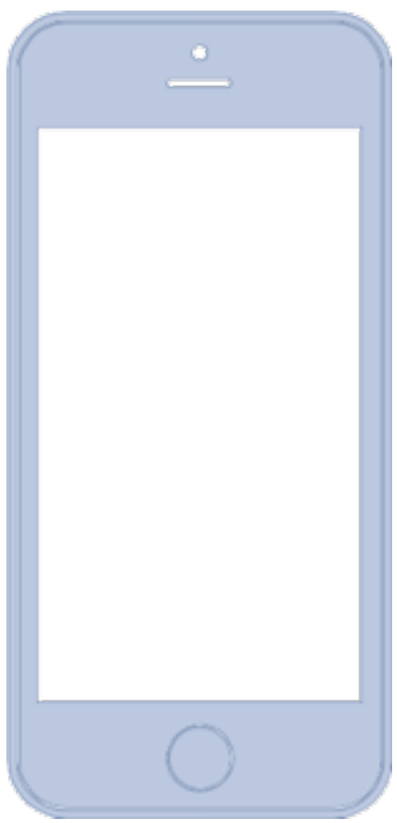


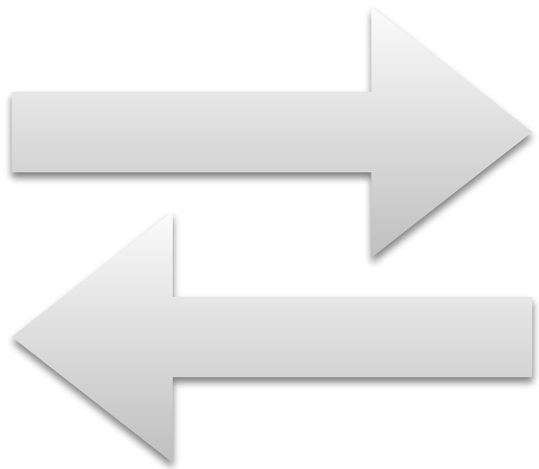

```
HTTP/1.1 200 OK
```

```
POST /send/push
Authorization: <server-token>
Device-ID: <device_id>

{"token":<token>}
```

Arquitetura proposta

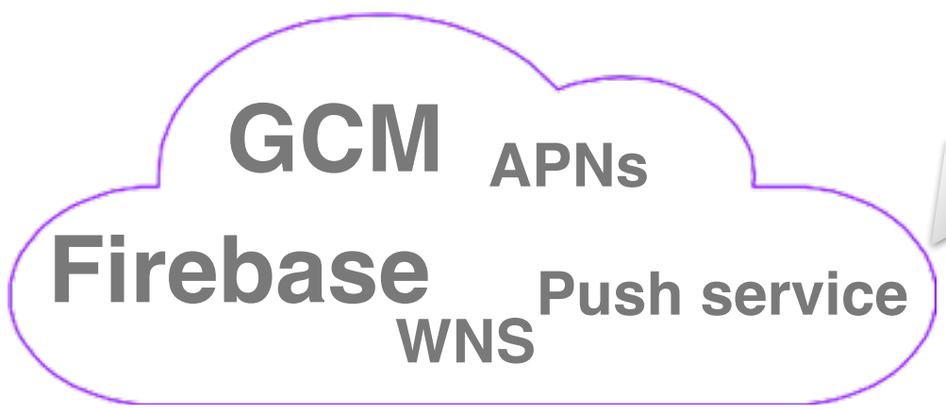
```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```





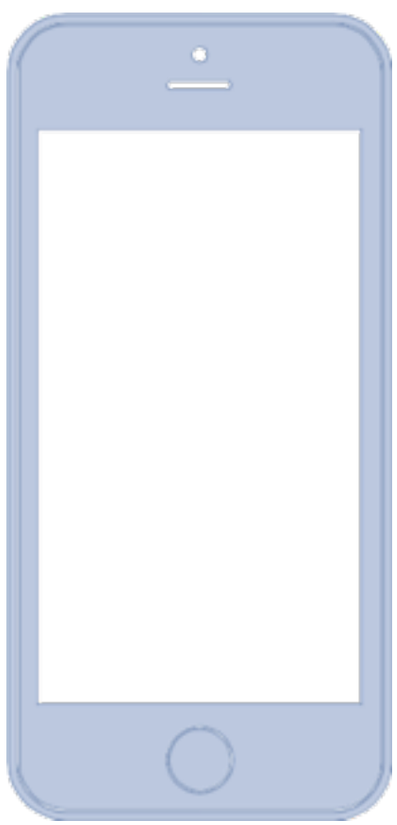
HTTP/1.1 200 OK

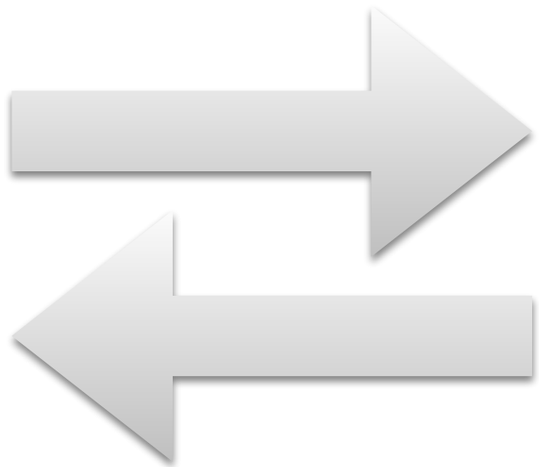
```
POST /send/push
Authorization: <server-token>
Device-ID: <device_id>

{"token":<token>}
```



Arquitetura proposta

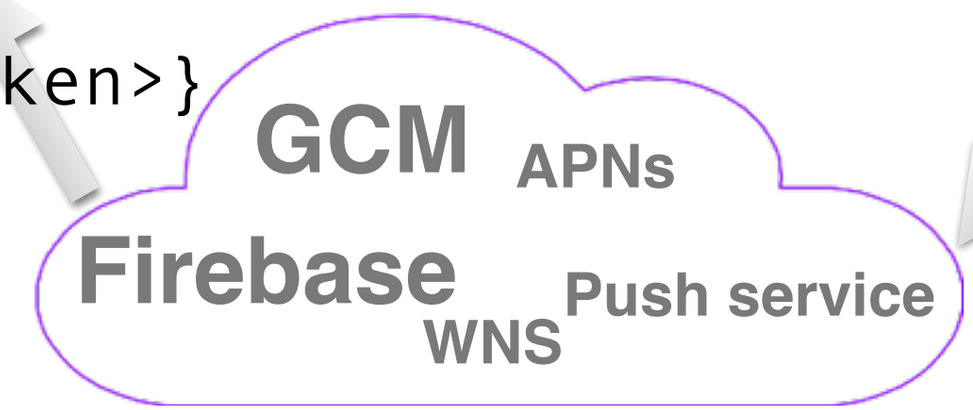
```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```





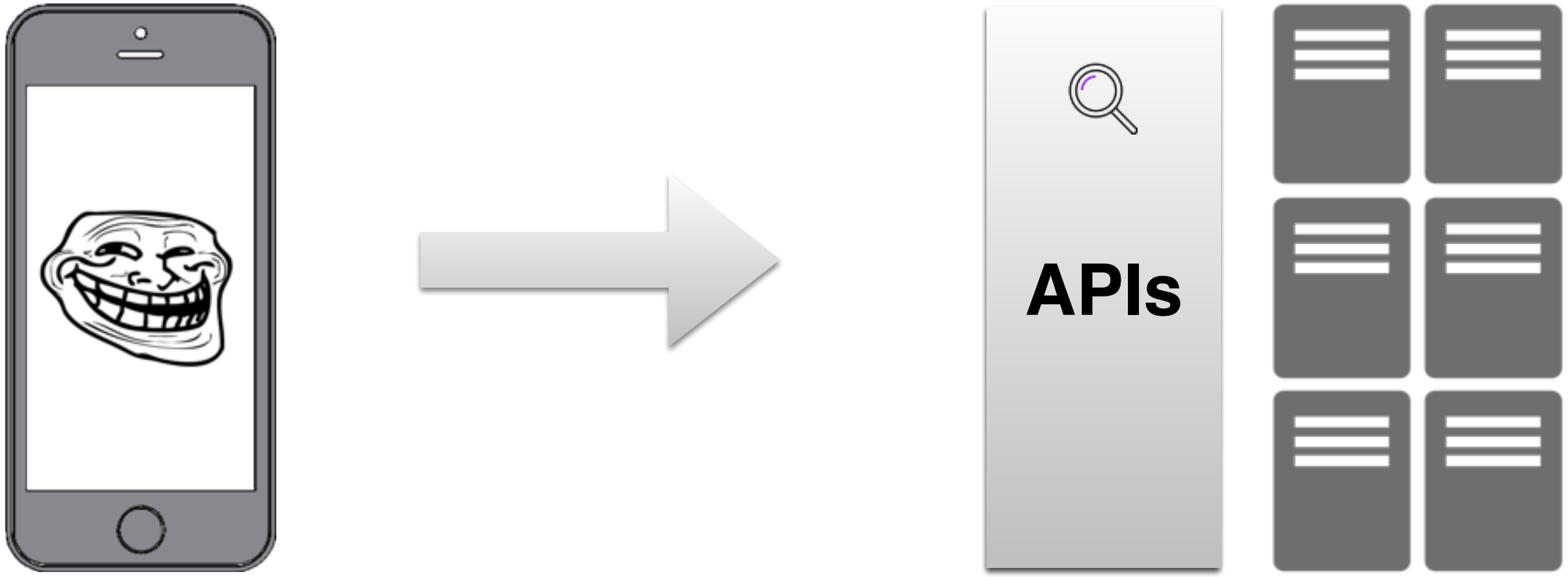
```
HTTP/1.1 200 OK
```

```
{"token":<token>}
```



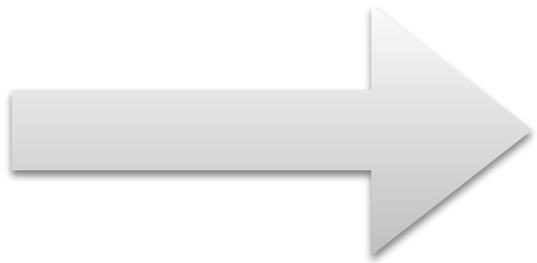

```
POST /send/push
Authorization: <server-token>
Device-ID: <device_id>

{"token":<token>}
```

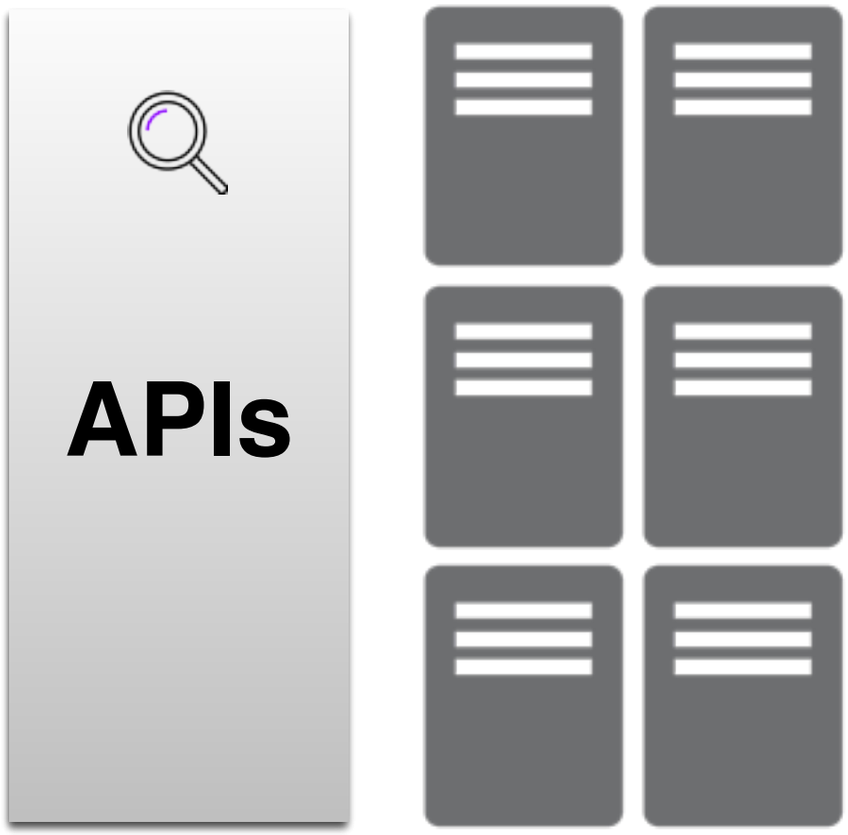
Arquitetura proposta



Arquitetura proposta

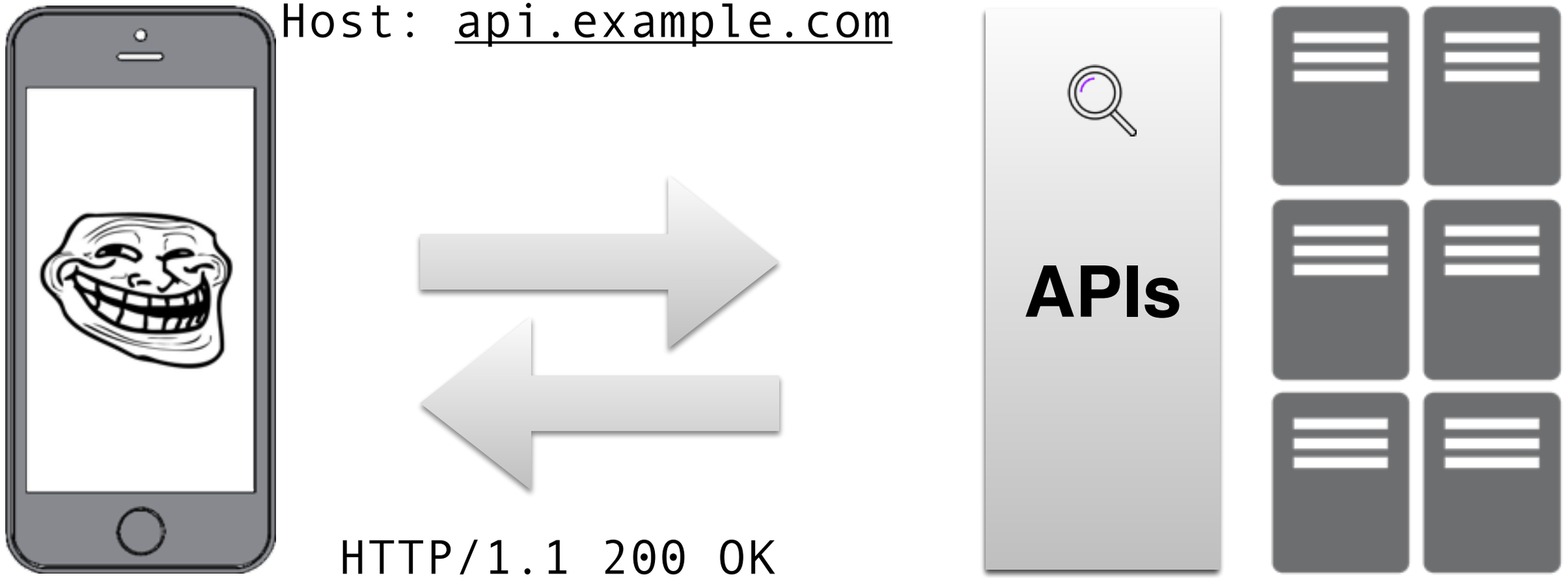
```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```





Arquitetura proposta

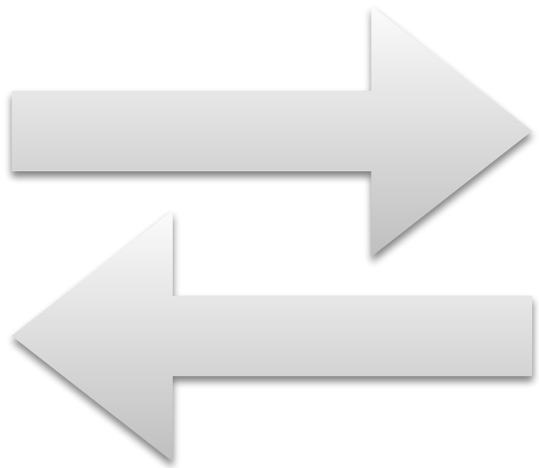
```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```



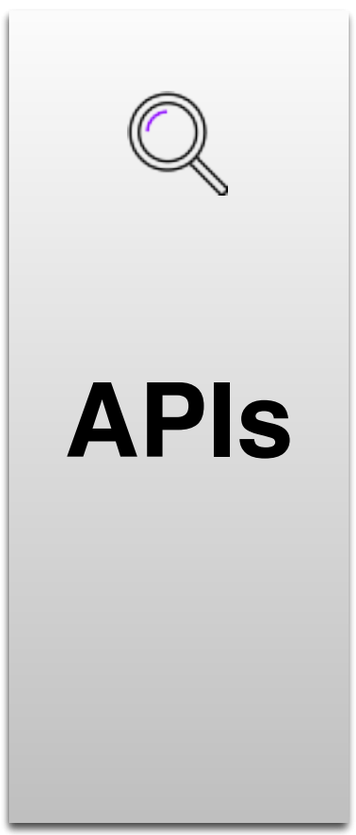

```
HTTP/1.1 200 OK
```

Arquitetura proposta

```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```



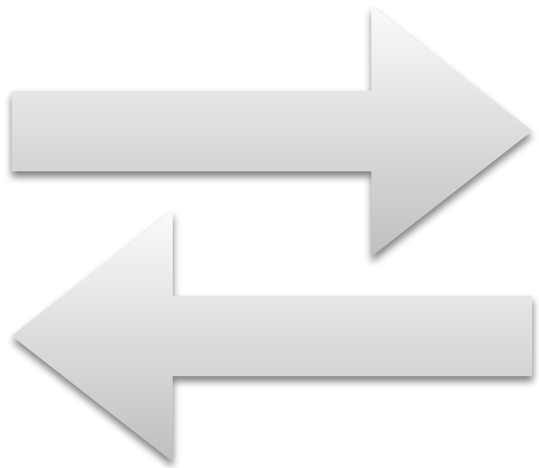
HTTP/1.1 200 OK



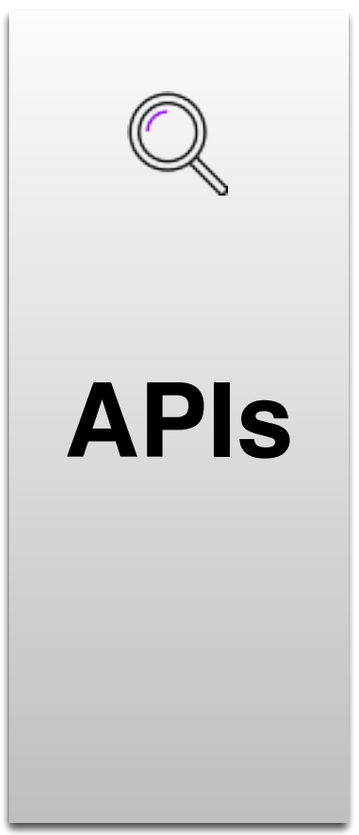
```
POST /send/push
Authorization: <server-token>
Device-ID: <device_id>
{"token":<token>}
```

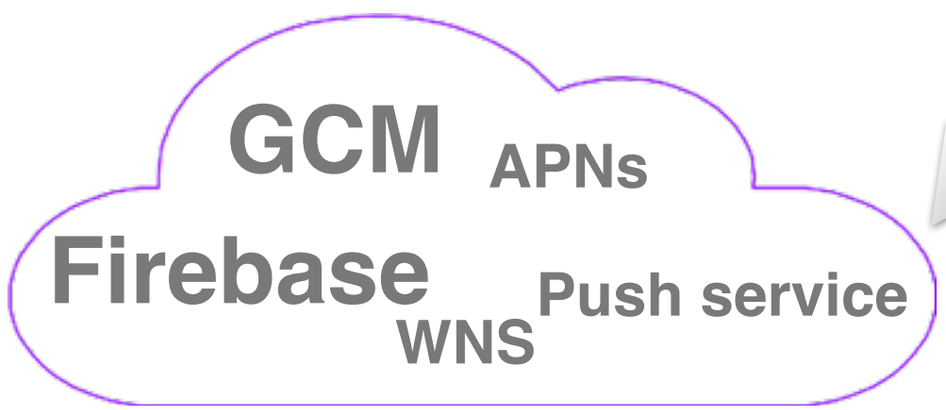
Arquitetura proposta

```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```



HTTP/1.1 200 OK

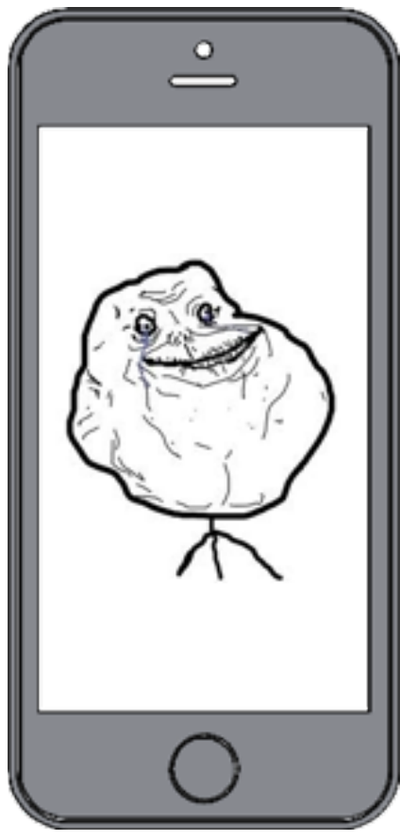


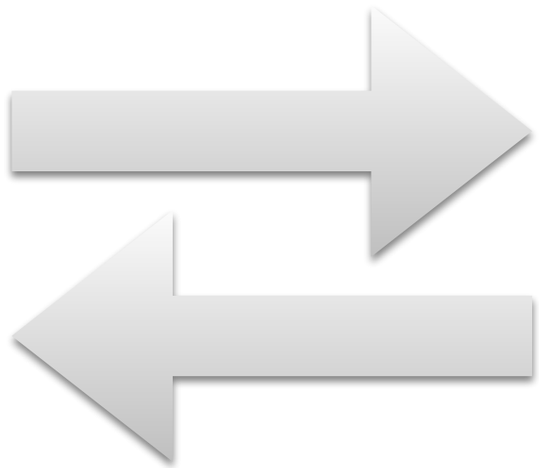



```
POST /send/push
Authorization: <server-token>
Device-ID: <device_id>
{"token":<token>}
```

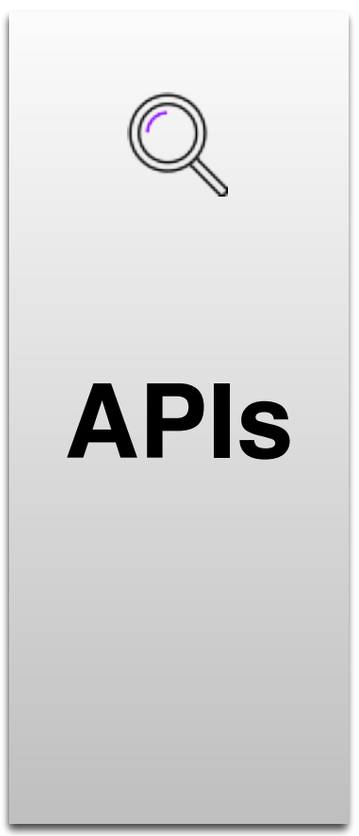
Arquitetura proposta

```
POST /token?device-id=<device_id> HTTP/1.1
Host: api.example.com
```

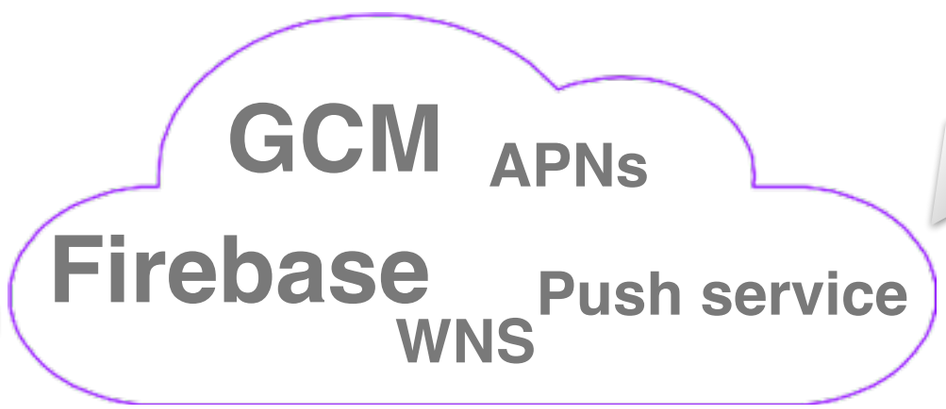


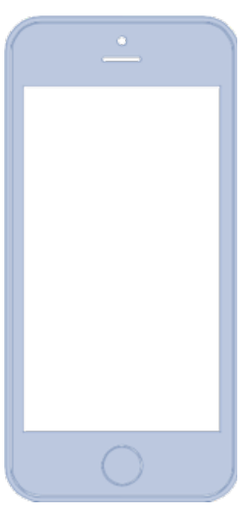


```
HTTP/1.1 200 OK
```



```
POST /send/push
Authorization: <server-token>
Device-ID: <device_id>
{"token":<token>}
```





```
{"token":<token>}
```

Arquitetura proposta

Arquitetura proposta

- Cada server-token/certificado está associado a um aplicativo, logo não é possível enviar mensagens *out-of-band* para os aplicativos "desconhecidos"

Arquitetura proposta

- Cada server-token/certificado está associado a um aplicativo, logo não é possível enviar mensagens *out-of-band* para os aplicativos "desconhecidos"
- Caso um token seja comprometido, é possível invalidar de forma individualizada, sem afetar os demais usuários

Arquitetura proposta

- Cada server-token/certificado está associado a um aplicativo, logo não é possível enviar mensagens *out-of-band* para os aplicativos "desconhecidos"
- Caso um token seja comprometido, é possível invalidar de forma individualizada, sem afetar os demais usuários
- Ao invés de retornar somente o token de acesso, diversas informações podem ser retornadas (por exemplo: configurações)

Arquitetura proposta

- Cada server-token/certificado está associado a um aplicativo, logo não é possível enviar mensagens *out-of-band* para os aplicativos "desconhecidos"
- Caso um token seja comprometido, é possível invalidar de forma individualizada, sem afetar os demais usuários
- Ao invés de retornar somente o token de acesso, diversas informações podem ser retornadas (por exemplo: configurações)
- O token concentra os atributos e responsabilidade de uma sessão

Perguntas?

manoel.junior@corp.globo.com

Obrigado

manoel.junior@corp.globo.com