XWS Vendor Proxy

Why?

The XWS team provides the XING API. It's a public interface that enables third parties to integrate and interact with XING data.

The goal of the XING API is to provide a consistent interface that developers love. To achieve this goal, API calls are carefully crafted to be simple, extendable, well documented, respond fast and behave in a consistent manner.

There are some use-cases that don't fit in the XING API. Reasons are that they are only relevant for a small amount of consumers (for example only the XING internal apps), the product is not stable enough to build a stable API that lasts or the time investment into a well crafted API is not possible. For those cases the XWS team has created the **Vendor Proxy**.

What is the Vendor Proxy?

The **Vendor Proxy** is a product developed and maintained by the XWS team. It enables other teams to provide their own **Vendor APIs** to the outside world, while reusing the XWS infrastructure.
The goal of the Vendor Proxy is to **reduce dependencies** and reuse infrastructure, in cases where an integration into the XING API is not possible.

The XWS team will **provide consulting** when you want to develop your own Vendor API. **XWS** is **not responsible** for operating, maintaining and supporting Vendor APIs. Its role is to provide the infrastructure and share its API knowledge.

THE DESIGN, IMPLEMENTATION, MAINTENANCE AND SUPPORT IS THEN THE RESPONSIBILITY OF THE DST.

Please **think carefully before** developing a Vendor API. It is a **long-term commitment** and not a quick hack. APIs are easy to build, but hard to maintain. Please consult the XWS team to clarify if your use-case is a good fit for the Vendor Proxy or if it should be part of the XING API.

To summarize the goals of the Vendor API:

- enable other teams to implement and operate external APIs on their own
- reduce dependencies
- let other teams own their Vendor APIs
- keep operational control within XWS
- keep the XING API clean

What isn't the Vendor Proxy?

The Vendor Proxy is not a mechanism to built **Private APIs**. Expect that whatever API you are building with the Vendor Proxy will be known eventually.

If you're building a Vendor API that will be consumed by a mobile app, such as the XING iPhone or Android app, this is especially relevant. Native mobile apps can always be **reverse engineered** and the Vendor APIs will be discovered. It's even possible to use those APIs without the app, because consumer keys can also be extracted from the app. In this case we can always shut down the consumer key if we see abuse, but this is the last line of defense.

This means that you have to ensure that you are building **carefully designed Vendor APIs**, that do not provide functionality that the user wouldn't have anyways and don't give away data that 3rd parties shouldn't get.

Examples for **bad** Vendor API **ideas**:

- **Idea:** Building a Vendor API that gives the iPhone the possibility to send more than 100 contact requests.
    - **Reason:** There is no way to ensure that only the iPhone can do this. An attacker might extract the Vendor API and the relevant secrets from the app binary and abuse this Vendor API for his needs.
- **Idea**: Do something that might not be a good idea in general and don't write a documentation. It'll be fine because don't know about this and can't abuse it.
    - **Reason**: People will find out eventually. In the meantime the hidden Vendor API is a security risk to the whole company. It can't be reviewed and future team members might not even know about it. Facebook had such a problem in the past.

What are the benefits and drawbacks of using the Vendor Proxy?

Benefits

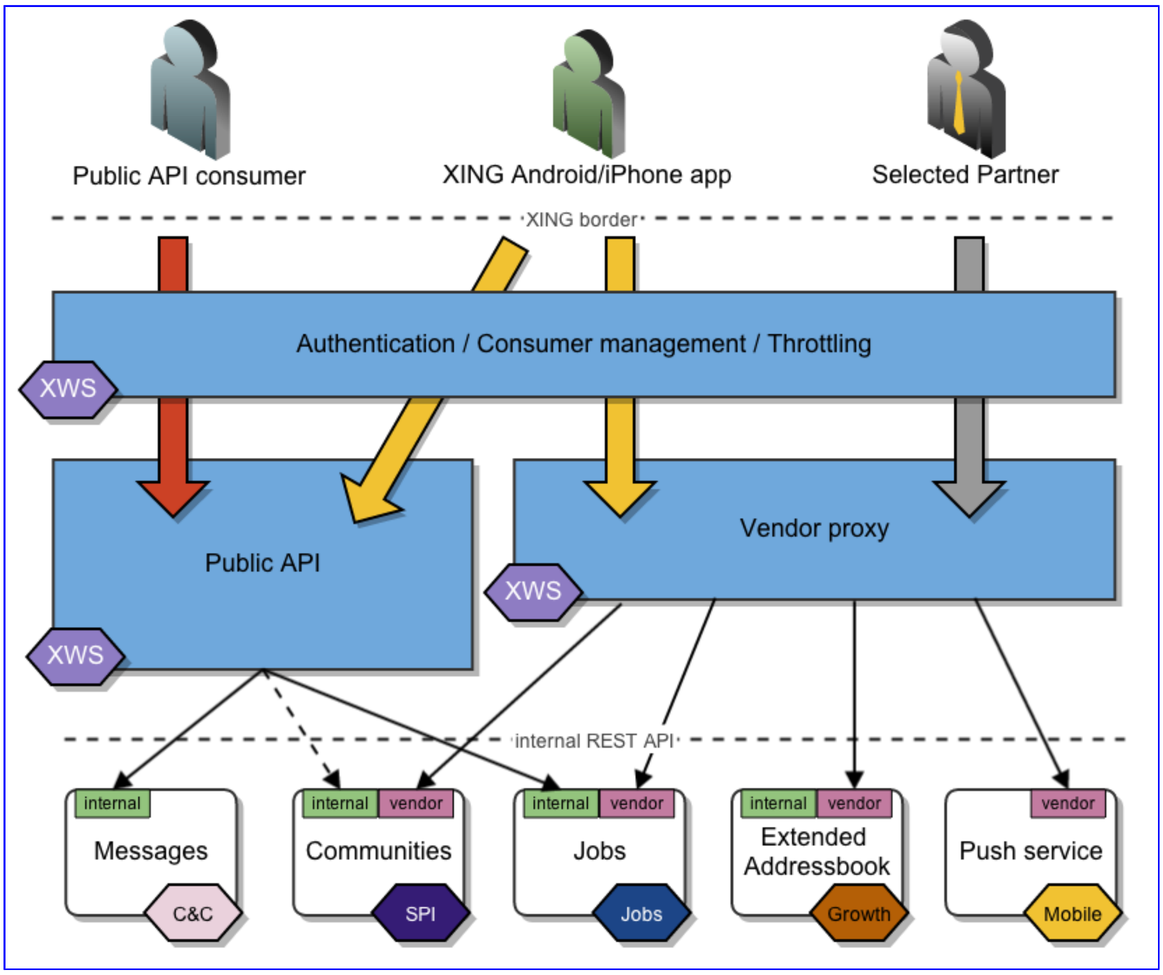
- Teams **don't** need to **reinvent the wheel** when they want to provide an external API.
  They can **reuse** the **XWS infrastructure**, while **retaining the flexibility** to design, implement, maintain and support their own API.
- Use-cases that **don't fit** into the **XING API** can be **made available** to selected partners
- **Time critical** API calls can be implemented directly by the teams

Drawbacks

- The team has to do the **complete design** of the API call
- The team has to make sure that it **maintains** a **stable API** and **communicate with** all **consumers** in case of incompatible changes.
- Since each Vendor API is designed by a different team, there is the possibility that a **consistent design** between the public API and other parts of the vendor API is **missing**
- **Versioning** is the DSTs responsibility

The big picture

Below you see an architectural picture of how the Vendor Proxy operates.



- Starting from the **bottom** of the picture:
  - Each XING DSTs can provide two types of APIs: **internal** and **vendor**.
  - The **internal** API is the regular Internal REST API. It **can't be exposed** to the outside world, because it contains concepts like the rid and has no access checking.
  - If a DST chooses to provide a **vendor** API it has to implement an additional API.
    The **vendor** API will be exposed to the outside world by the **Vendor Proxy** which is developed and maintained by the XWS team.

- In the **middle** of the picture are three blue colored boxes that represent the XWS part of the architecture.
  - The **XING API** (left side) will communicate with the **internal** APIs provided by the DSTs. It orchestrates various internal APIs provided by different DSTs and enforce certain rules.

    - Moving to the right side, the **Vendor Proxy** is a direct mapping between external URLs and internal URLs.
      It will pass some additional information like the authenticated user and consumer to the **vendor** API.

- Going to the **top** of the picture, there are three categories of API consumers:
    - regular **XING API consumers**, that will only use the public API
    - **internal consumers**, that will have the possibility to also use the vendor proxy
    - **selected partners** with special use cases who are using the vendor proxy

How to develop a Vendor API

For more details on what you have to do to use the Vendor Proxy and develop a Vendor API see How to develop a Vendor API