IIBA® Belarus Chapter

REQUIREMENTS & API. Part I

Ilya Zakharau

API Platform Product Manager, 7 years of prior BA experience

IIBA®

IIBA® Belarus Chapter

WHY I AM DOING THIS?

1. Building API tooling and products is my current expertise, and I want to share it with the public.
2. Such terms as “API economy,” “API-first”, and “API market” have become quite popular in recent years.
3. From my observations, few Business Analysts are proficient in API or have a fractured understanding.
4. Offline API workshop made by the BArszawa community last February showed demand for that topic.

IIBA®

IIBA® Belarus Chapter

AGENDA

There will be two parts:

Part 1 will be more theoretical and more tedious. We will talk about the definition of API and API definition, involved actors, breaking changes, REST & HTTP, and other basics.

Part 2 will be more practice-oriented:

- Discussing the API design-first approach
- Diving into OpenAPI specification and related tooling
- Decomposing one API case

What is API?

From Wikipedia:

“An application programming interface (API) is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software.”

It is a very vast definition, but there are two main pieces:

- interface
- service

Definition from Sergey Konstantinov's "The API Book"

“An API is an **obligation**. A formal obligation to connect different programmable contexts”.

The Pont-du-Gard aqueduct. Built in the 1st century AD. Image Credit: igorelick @ pixabay

Roman aqueduct (1st century AD):

- Interconnects two areas
- Flow water is an analogy to data
- Backward compatibility has not been for almost 2000 years
- Additional infrastructure was built to produce and consume the water supply

Another API definition (and last for today)

Let’s look from another perspective: API is a **distribution channel**.

It is similar, but quite opposite to User Interface:

- Not for end-users.
- For developers who will build their solution for their end-users based on API.

You can integrate with API as a developer (Consumer) or develop API (Producer). Both cases have their peculiarities. But today, we talk about the latter only.

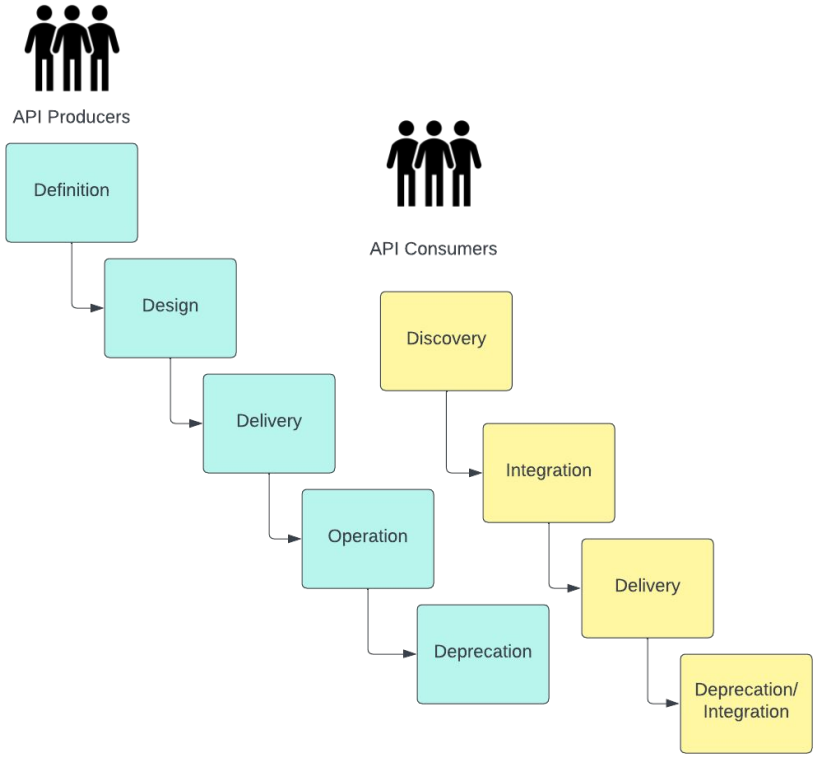
A few more definitions we need:

- **API endpoint:** a single instance of API with a unique address (URL). Sending a required input to that URL triggers some command within your system and returns an output about a successful result or failure.

  Example: *POST https://ecomplatform/api/v1/purchases/{purchaseId}/cancel*

- **API call:** sending a request to an API endpoint and getting a response.
- **Client:** a general term to identify someone or something calling your API.

Actors and API Lifecycle



Both Producers and Consumers are not only Developers, there are other roles involved in the SDLC:

- Business Analysts
- Architects
- QA/QAA Engineers
- Product Owners/Managers
- Delivery/Project Managers
- etc

The Value Chain

API Platform

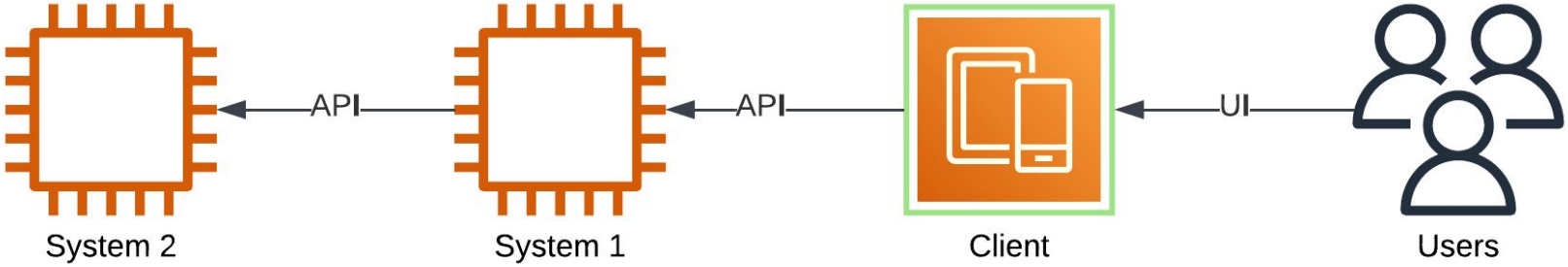
API Developer

App Developer

Business Owner

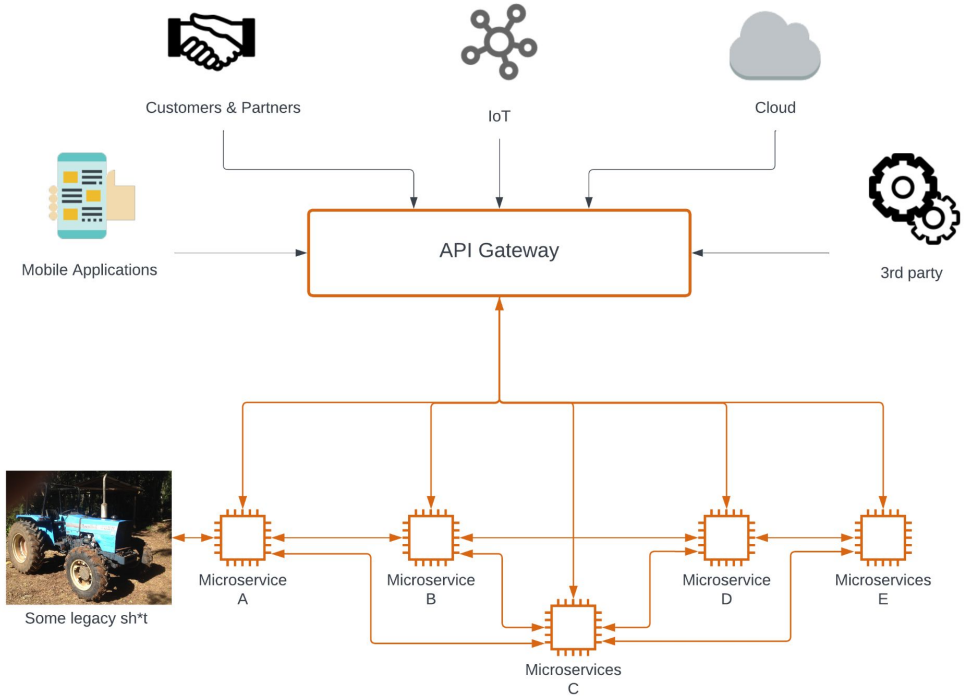
End-user / Business Actor

Why BA: traditional approach (simplified)



- Usually one or several API consumers
- API does provide access to functionality
- API == functionality
- Matter of tech design, not a BA part

Why BA: modern approach



- The rise of Cloud, SaaS, and PaaS (thank you, Amazon) accelerated the interconnectivity of multiple systems and open APIs.
- That results in headless "API-first" products (like Stripe), where API is the primary interface.
- Many consumers means many stakeholders have their own needs, so it is a work for a BA.
- Some APIs are subject to regulations.

API Categorization

The technical categorization is quite a mess:

- REST (REpresentational State Transfer): architecture style
- SOAP (Simple Object Access Protocol): messaging protocol specification
- RPC (Remote Procedural Call): request-response protocol
- GraphQL: query language

We will focus on REST API as it is a primary way of the service communication nowadays.

REST in its entirety

REST architecture styles principles:

- *The client and the server do not know how each of them is implemented*
- *Sessions are stored on the client (the “stateless” constraint)*
- *Data must be marked as cacheable or non-cacheable*
- *Interaction interfaces between system components must be uniform*
- *Network-based systems are layered, meaning every server may just be a proxy to another server*
- *The functionality of the client might be enhanced by the server providing code on demand.*

What does that also mean:

- REST author Roy Fielding is also a co-author of HTTP (Hypertext Transfer Protocol). So they are literally tight together.
- General terms described in the REST principles can be interpreted and implemented in many ways.
- Term REST API has caused holy wars in the tech community for 20 years. So, there is a RESTful API that loosely follows some REST principles.
- Videos to dive into topic (RUS): https://www.youtube.com/live/DB2SER51mcU?si=BcFDDrPldqCpB_Oz and https://www.youtube.com/live/rURUWnsBnDA?si=_FjC0x3taByBe1pr

HTTP is the Foundation

HTTP consists of:

- Verb (**GET, POST, PUT, PATCH, DELETE**, OPTIONS, TRACE)
- Headers
- Body
- Response code

```
GET /page/1 HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html>… (here come the 29769 bytes of the requested web page)
```

https://developer.mozilla.org/ru/docs/Web/HTTP

IIBA® Belarus Chapter

Let’s finish with tech and move to BA stuff

IIBA®

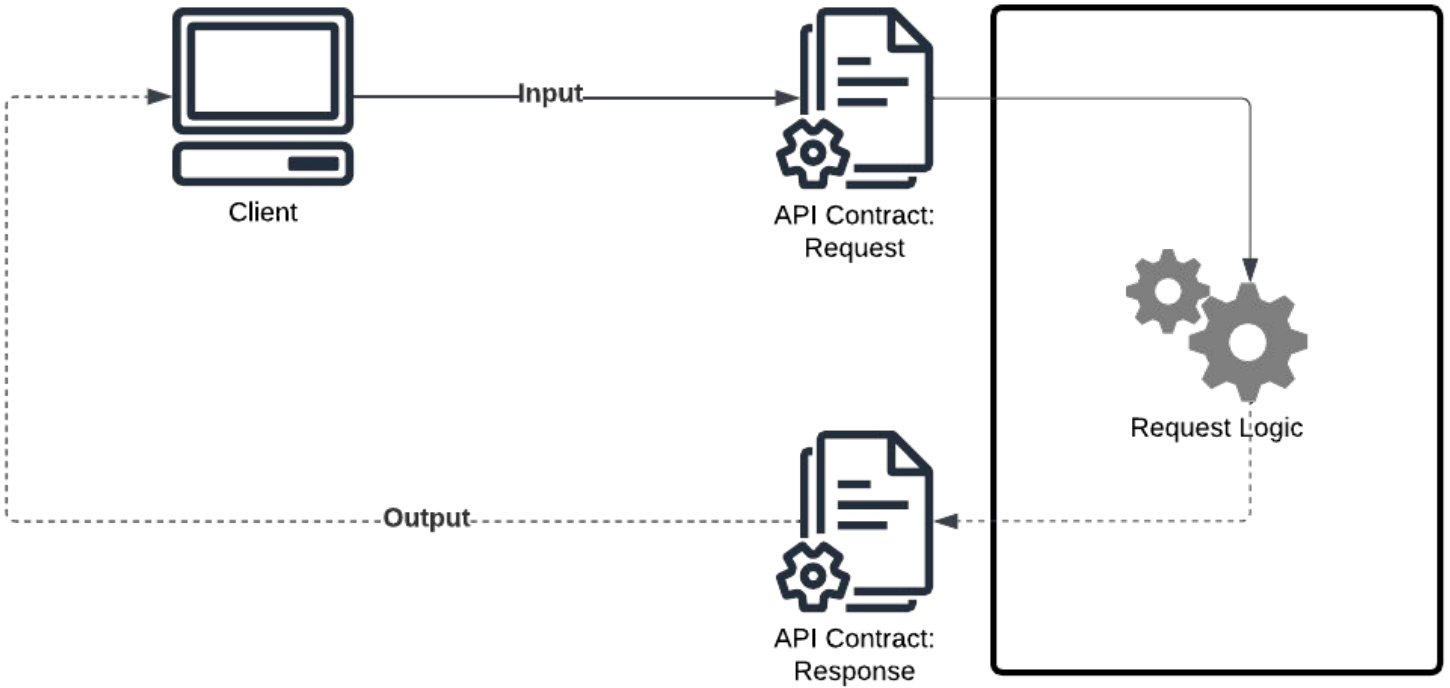
API Definition vs Implementation

API definition is a textual specification of request and response contract, logic, and metadata:

- A result of business analysis activities.
- It can be in human and/or machine-readable formats. Source code is also a definition.

Implementation is when the API is up and running, i.e., an executable artifact is deployed somewhere.

API Contract



API Contract - 1

- API contract is an agreement about expected input and outcome between you and the Clients.
- The Contract is strictly formalized, and clients agree to follow it when they start using your API.
- The API owner takes responsibility for maintaining the consistency of the Contract.
- There can be legal punishment for breaking API contracts
- API contract can be defined prior to the implementation so both sides can work in parallel.

Backward Compatibility

- It is a cornerstone of API development: a breaking change will cause changes in the client code as soon as the API is published.
- Thus "Agile" is not very good for APIs; for each breaking change, you consumers pay a certain price (which amounts you can't predict as you might not know how they use your API)
- So you need to design API in a consistent to avoid possible changes in the future (which is not possible).

Requirements Classification

- API requirements != functional requirements; UI requirement is the best analogy.
- An excerpt from K. Wiegers' "Software Requirements":

> and portability. Other classes of nonfunctional requirements describe *external interfaces* between the system and the outside world. These include connections to other software systems, hardware components, and users, as well as communication interfaces. Design and implementation *con-*

- BABOK technique 10.24 Interface analysis, not really much about APIs.
- Requirements Engineering Standard ISO/IEC/IEEE 29148 -> Interface Requirements.

Why & What Questions

For a new API we need to understand the following things to succeed with the requirements:

- Why do we need this?
- What will it do?

A single API endpoint itself does not make much sense; it is a part of one or several use cases. As a BA, you need to identify such use cases.

And then a few more important questions:

- Do we already have such an API?
- Does the system even have such functionality?

API Prerequisites

The next step is to define the prerequisite of an API call - authentication and authorization:

- Authentication is about verifying clients and allowing them to communicate with your API.
- Authorization is about permissions to enable specific clients to make particular API requests and see certain data.
- Another important point is whether a Consumer possesses all the necessary input data to make a call.

Describing Black Box

The next step is to define an input and output, describing boundaries of the black box (template):

- Request:
  - path parameter - used for identification, required
  - query parameters - additional filtering options, key-value, usually optional
  - a request body - JSON or XML
    - optional/required
    - data types
  - request headers - technical metadata passed by consumer
- Response:
  - response code (including error codes)
  - response body
  - response header - technical metadata returned from server to consumer

Naming Conventions

There are two massive problems in software development: naming and caching. I recommend you leave caching to the technical folks.

- There should be established naming conventions in your product/project/organization (better) for naming URLs, errors, and attributes.
    - If there is no such convention, this is the right place to escalate.
    - If no one cares, you should look for the best practices and define the convention.
- The API data model is not equal to a data model in a database:
    - You don't need to provide all the attributes.
    - You are not obliged to have the same name.
    - You are likely to follow different naming convention
- Consistency is key - you can't just change naming afterward, as you will break backward compatibility.

HTTP Error codes

- There is a predefined set of HTTP error codes
  - 4xx client error - wrong data
  - 5xx server error
- You can also reuse existing error codes for some business logic execution, defining a custom error message
- There is a famous 404 - not found, but what code should be returned with an empty search result?

Request Logic

- Basic/Direct API call: one internal call for a particular Entity object
- Composite/Aggregation API call: chain of internal calls, usually returned a composite data structure
- The way of specifying:
  - UML sequence diagram
  - UML activity diagram
  - Step-by-step text description
  - Postman Collection

IIBA® Belarus Chapter

End of Part 1. See you in two weeks

https://ilyazakharau.com/

https://www.linkedin.com/in/ilya-zakharau/