IIBA® Belarus Chapter

REQUIREMENTS & API. Part II

Ilya Zakharau

API Platform Product Manager, 7 years of prior BA experience

IIBA®

IIBA® Belarus Chapter

In the previous episode…

Two weeks ago we have covered:

- A term “API” and its implications
- API categorization and its mess
- Mythical REST
- HTTP as a foundation for (almost) everything
- API definition and implementation
- API contract as a black box
- Backward compatibility
- API in the Requirements Classifications
- Request logic, errors, etc

IIBA®

IIBA® Belarus Chapter

AGENDA

Today we will talk about:

- “Formal” API definitions such as OpenAPI
- API design-first approach
- OpenAPI specification
- Other competing IDLs
- API Gateway
- Backends-for-Frontends pattern
- Experience-driven API
- And review a practical case

IIBA®

Let’s revisit: 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.

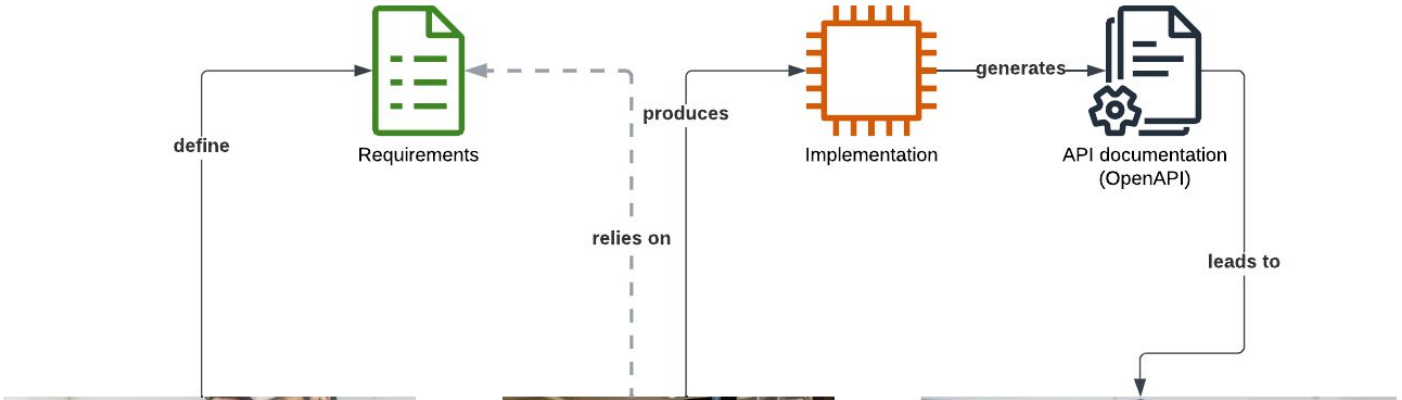
Informal vs Formal specification

- Informal - any **human-readable** specification describing API design
- Formal - **machine-readable**, strictly formalized specification attached to a specific data format (e.g. XML, JSON)

OpenAPI is the most common formal API specification format for RESTful API

Informal formats usually tends to mimic OpenAPI but in more human-friendly manner

Traditional approach (simplified)

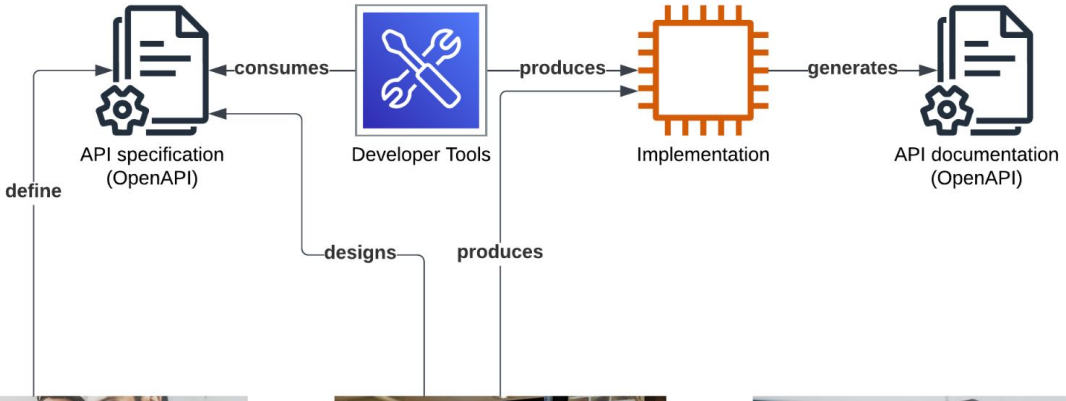


BA & Stakeholders

Developer

A happy QA who now has up-to-date docs along with the requirements

API Design-first approach



BA & Stakeholders

Developer

A happy QA who now can directly compare specification vs documentation vs implementation

IIBA® Belarus Chapter

Let’s see how our “Add Employee” example looks in OpenAPI format

A few facts about OpenAPI

- Often referred to as OAS.
- Latest stable version: 3.1.0 (link to specification).
- Typically expressed using JSON or YAML; usually works with JSON and XML.
- Version 2.0 and below of this specification were known as Swagger
- One of the most popular IDLs (Interface Definition Language) which also is DSL (Domain-Specific Language); de facto industry standard.
- Huge variety of tooling around it.
- Alternatives are: API Blueprint, RAML, AsyncAPI, GraphQL Schema and many others.
- Super tool to learn OpenAPI: https://openapi-map.apihandyman.io/

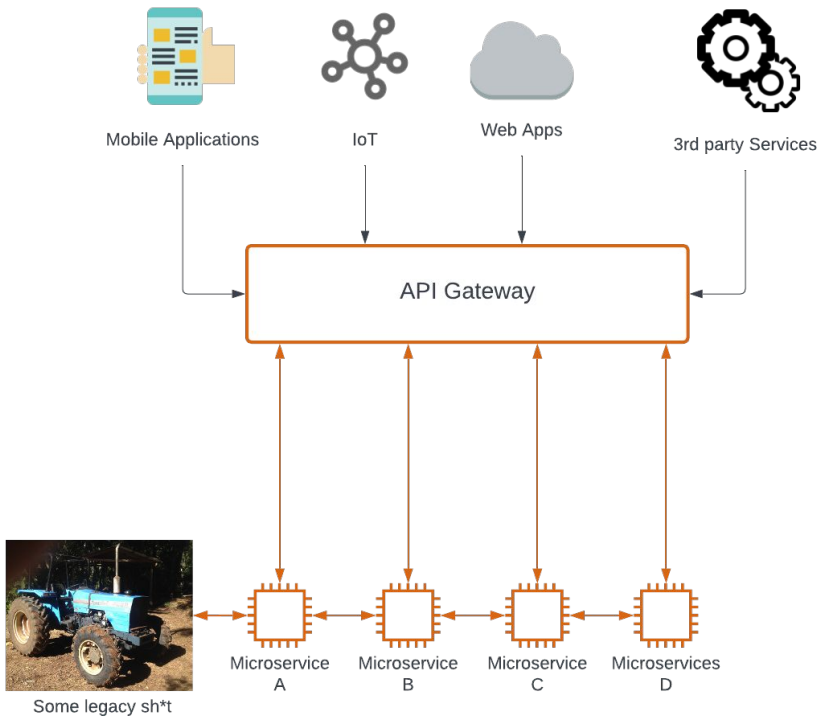
IIBA® Belarus Chapter

OpenAPI seems to be too complex? Let's try one of a lightweight alternatives: JSight

IIBA® Belarus Chapter

Wanting something more user-friendly and not OSS? Look no further than: Stoplight

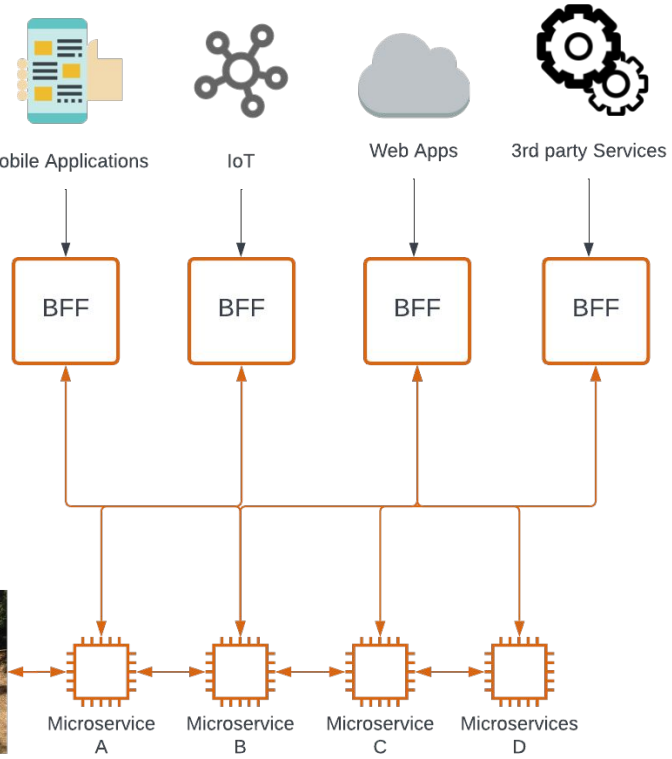
What is an API Gateway?



API Gateway is a centralized platform that serves as an entry point for for a collection of microservices or backend services. It considers:

- Authentication and Authorization
- Request Routing, Aggregation
- Load Balancing
- Request/Response transformations
- Rate Limiting, Resilience
- Caching
- Logging, Monitoring

Backends-for-Frontends (BFF) Pattern



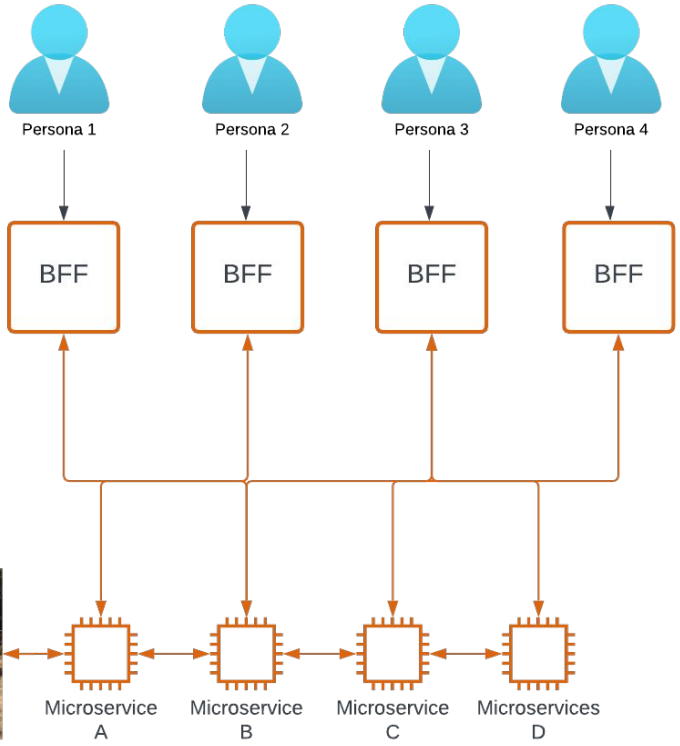
Some legacy sh*t

BFF - is a design pattern that considers creating dedicated backend services tailored to the specific needs of different **client interfaces** or user **experiences**.

Different client interfaces have different needs for:

- Mobile applications (further division on mobile platforms, e.g. iOS, Android)
- Web apps
- Internet-of-things
- etc

Experience-Driven (Persona) API approach



Some legacy sh*t

Client division might not be sufficient. Different Business users (aka Personas) require different set of APIs with different set response data and other specifics.

For example, taxi services like Uber/Bolt:

- Passenger API Persona
- Driver API Persona
- Taxi Fleet Manager API Persona
- Government Regulator API Persona

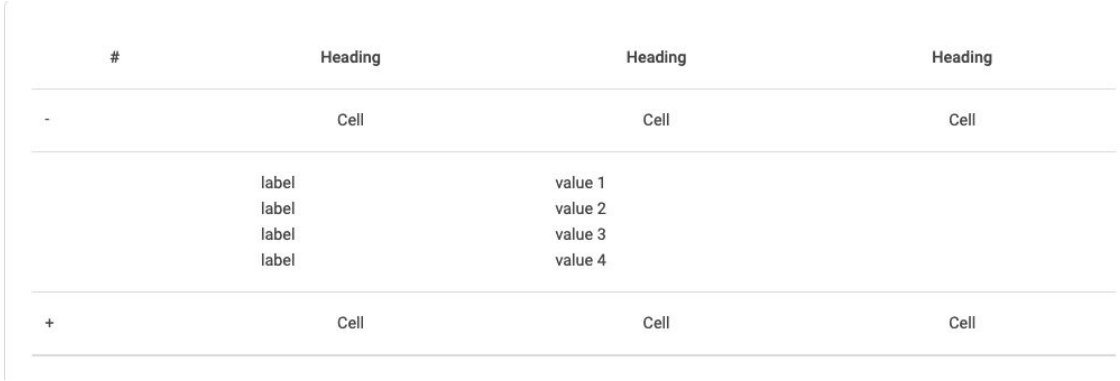
IIBA® Belarus Chapter

Practical Case of Experience-driven API

IIBA®

Experience-Driven Employee Search API for HR

Let’s assume on Client’s UI there will be a table with expandables rows like this one:



#	Heading	Heading	Heading
-	Cell	Cell	Cell
	label label label label	value 1 value 2 value 3 value 4	
+	Cell	Cell	Cell

For that UI, Client needs a **single API endpoint** to return a list of active Employees so:

- Each row contains an Employee data: ID, First Name, Last Name, Department, Title, Employment Type.
- Additionally, they need to show up to 5 active Policies and Claims if they do exist. Their IDs and effective dates are enough.

Assumptions

Additionally let's assume the following:

1. There is already established authentication and authorization. We don't need to care about that.
2. As an input, Client will provide an Employer UID.
3. Yes, pagination is required.
4. And, please, filtering and sorting for all Row attributes.
5. All required Internal APIs are provided. They can't be changed and exposing new internal APIs it not an option.
6. Our previous "Add Employee" API is related to the same Persona.

Considerations

1. For Internal API we act as Consumer, so we need to find the APIs we need.
2. We also need to make sure internal APIs possess data we need + they support pagination and have sorting and filtering capabilities.
3. We need to make sure that we, as Client, possess required input data to call those APIs.
4. We need to know which IDs should be displayed on UI.
5. We need to know the next API calls to be made, so we can ensure having a right input.
6. What permissions are required to call those internal APIs.

Case study on GitHub

IIBA® Belarus Chapter

I will publish the Reading list on my blog and LinkedIn soon.

Stay tuned and thank you all for you attention!

https://ilyazakharau.com/

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

IIBA®