TÉCNICO LISBOA

Enterprise Integration

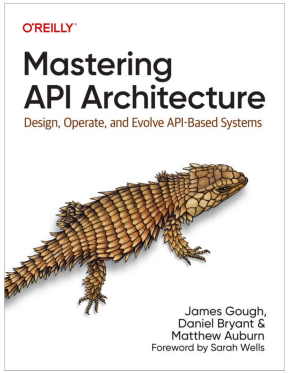
Application Programming Interfaces (API) Management

Prof. Sérgio Guerreiro
Sergio.guerreiro@tecnico.ulisboa.pt

Department of Computer Science and Engineering
Instituto Superior Técnico / Universidade de Lisboa
INESC-ID
URL: http://www.inesc-id.pt
Rua Alves Redol, 9
1000-029 Lisboa
Portugal

TÉCNICO LISBOA

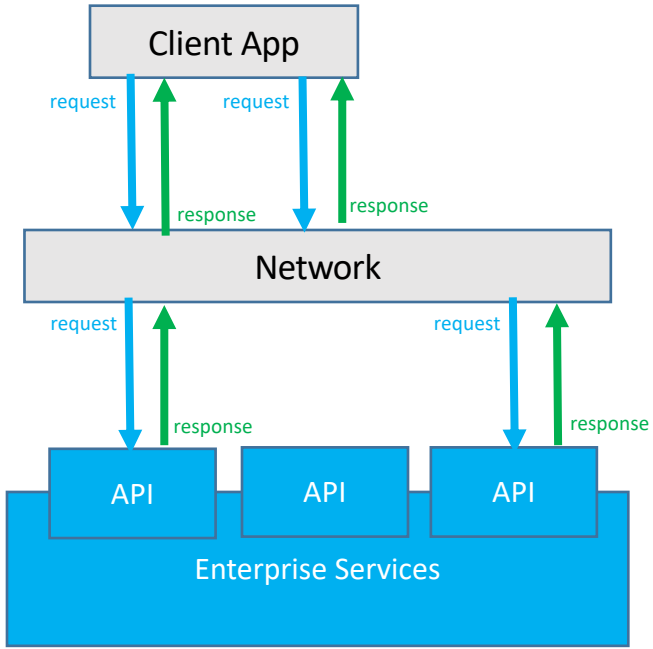
Gough, J., Bryant, D., Auburn, M. (2023). Mastering API Architecture: Design, Operate, and Evolve API-Based Systems, O'Reilly Media, Inc., ISBN: 9781492090632



Application Programming Interfaces (APIs)

APIs are *by definition* **interfaces to be used as entry points to reusable software entities with well-defined contracts**.

They are not independent software entities; they are instead packaged with the software libraries, frameworks, or Web services, that offer them.



Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2021. A Systematic Review of API Evolution Literature. ACM Comput. Surv. 54, 8, Article 171 (November 2022), 36 pages. https://doi.org/10.1145/3470133

A broader API definition

- An API represents an abstraction of the underlying implementation
- An API is represented by a specification that introduces types. Developers can understand the specifications and use tooling to generate code in multiple languages to implement an API consumer (software that consumes an API).
- An API has defined semantics or behavior to effectively model the exchange of information.
- Effective API design enables extension to customers or third parties for a business integration.

- *in process* API invocation: *e.g.,* a java RMI call is handled by the same process from which the call was made
- *out of process* API invocation: *e.g.*, a .NET application invocating an external REST-like API using an HTTP library is handled by an additional external process other than the process from which the call was made.

Gough, J., Bryant, D., Auburn, M. (2023). Mastering API Architecture: Design, Operate, and Evolve API-Based Systems, O'Reilly Media, Inc., ISBN: 9781492090632

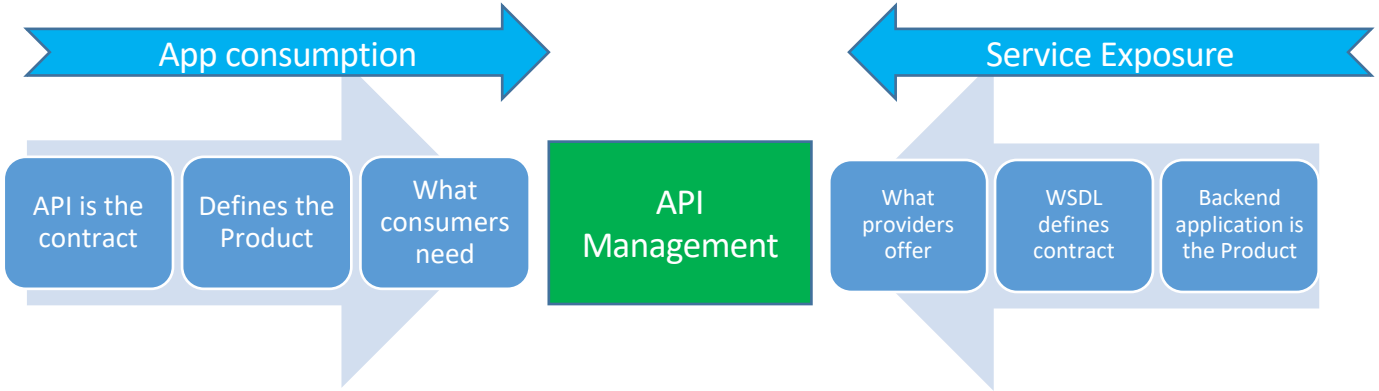
Application Programming Interfaces (APIs)

- Due to their omnipresence and evolution, APIs **greatly impact software development**. Understanding, mitigating, and leveraging the impact of APIs and API evolution on software development is necessary to design and use software APIs
- APIs evolve for various reasons, such as increasing complexity, and continuous change. However, due to their nature as a **connection point** between software modules, **API evolution is not without side effects**
- On the one hand, as predicted by **Lehman**, **continuing change** means that API developers must determine ways to keep their APIs useful, cutting edge, and competitive with other pieces of software and API users must adapt to these API changes and new API releases.
- On the other hand, **conservation of familiarity**, or existing API usages, constrain the evolution of an API to avoid breaking changes while improving the API (*i.e.*, security or performance improvements). The evolution of APIs therefore involves a balancing **act of constant improvement and maintaining existing functionality**. Maintaining existing functionality requires in-depth knowledge of use cases and architectural foresight and flexibility, while keeping up with rapid release cycles requires modifications to user applications as well as learning about new APIs and changes to existing APIs.

Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2021. A Systematic Review of API Evolution Literature. ACM Comput. Surv. 54, 8, Article 171 (November 2022), 36 pages. https://doi.org/10.1145/3470133

Differences from SOA

- From an architectural perspective they are similar: *“a logical representation of a repeatable activity that has a specific outcome, the usual notion of a service”*
- But the focus is different:
  - APIs were focused in simplifying the consumption, developer centric, human-readable contract
  - SOA focus in shielding the client from back-office applications changes



API trends

- **New tools and techniques** typically seek to help with API evolution by **resolving problems** that it can cause for API users (*e.g.*, API migration tools) or to help **reduce the development** burden on API developer (*e.g.*, automatic API documentation tools).

- Empirical studies related to API evolution typically employ large data, case studies, or user studies to provide evidence of existing problems, the impacts of API evolution, or potential solutions to existing problems. These problems typically centre around the impacts of API evolution on **API usability** and **API maintainability**.

Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2021. A Systematic Review of API Evolution Literature. ACM Comput. Surv. 54, 8, Article 171 (November 2022), 36 pages. https://doi.org/10.1145/3470133

State-of-the-art Solutions to Existing API Evolution Challenges

Challenge	Proposed State-of-the-Art Solution
Dealing with breaking API changes	Document acceptable usages [179]
	API bundles [103]
	Automated API migration [24, 57]
	Extracting migration knowledge from clients [86, 158]
Improving API usability	Local interaction patterns [65]
	Usability patterns [202]
	API selection criteria [117]
	Digital assistants [16]
	Automated API tips [175]
	Automated documentation [2, 71, 98, 149, 171]
	Example mining [58, 116, 168]
	API recommendation algorithms [12, 99, 125, 190]
Reducing API misuses	Misuse detectors [4, 5, 9]

Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2021. A Systematic Review of API Evolution Literature. ACM Comput. Surv. 54, 8, Article 171 (November 2022), 36 pages. https://doi.org/10.1145/3470133

Open Challenges in API Research

TÉCNICO LISBOA

Challenge types	Paper types	Challenges
Existing challenges	New tools and techniques	*EC-1* Combining textual merging with syntactic and semantic approaches [109]
		EC-2 Providing a commercially viable API migration solution [20, 34]
		EC-3 Incorporating domain-specific information into tools [109]
		EC-4 Using systematic evaluation methods in empirical evaluations [145]
		EC-5 Producing more specific and less abstract theories [102]
		EC-6 Reducing the variability of software API studies [102]
		EC-7 Finding input examples for API migration through examples [148]
		EC-8 Improving the granularity of API migration approaches [148]
		EC-9 Validation and correction of API migration edit scripts [148]
		EC-10 More tools to help with Web APIs [180]
		EC-11 Using existing library API research as stepping stones for Web APIs [180]
		EC-12 Combining both API side learning with client side learning [158]
		EC-13 Dealing with out-of-vocabulary problems [24]
	Empirical studies	*EC-14* Defining best fit APIs [192]
		EC-15 Automatically identifying factors that drive API changes [60, 70, 191]
		EC-16 Dealing with API semantics and dependencies [5]
		EC-17 Deploying bug fixes to multiple API versions [160]

Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2021. A Systematic Review of API Evolution Literature. ACM Comput. Surv. 54, 8, Article 171 (November 2022), 36 pages. https://doi.org/10.1145/3470133

Open Challenges in API Research

Unsolved Challenges	New tools and techniques	*UC-1* Using uniform benchmarks for API tool evaluation
		UC-2 Supporting the context sensitivity of API migration tools
		UC-3 Improving performance of API tooling to allow user adoption
		UC-4 Dealing with fuzzy and ambiguous developer intent
		UC-5 Reducing the knowledge gap between API users and developers
		UC-6 Tools that mine usage data help API developers improve APIs
		UC-7 Keeping API users in the loop for API recommendation systems
		UC-8 Generalizing API tools to languages other than Java
		UC-9 Tools to help API developers deal with API migration, not just users
		UC-10 Reducing API misuse from the API development side
	Empirical studies	*UC-11* Understanding the coupling between APIs and programming languages
		UC-12 Determining API migration and API recommendation impacts
		UC-13 Generalizing API empirical studies to languages other than Java
		UC-14 Comparing the evolution of various APIs
	Datasets	*UC-15* Creating large-scale API migration and recommendation datasets

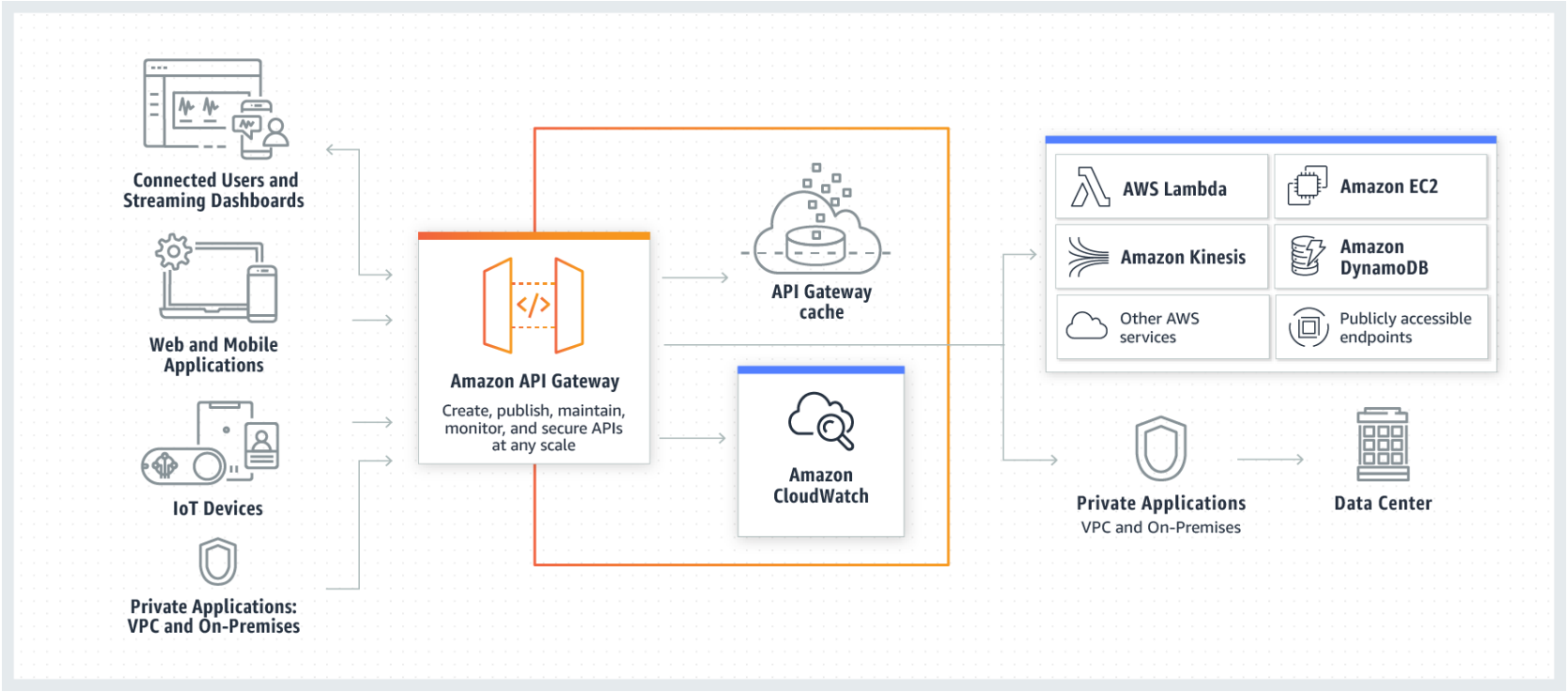
Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2021. A Systematic Review of API Evolution Literature. ACM Comput. Surv. 54, 8, Article 171 (November 2022), 36 pages. https://doi.org/10.1145/3470133

What is API management?

*"…is a set of tools and services that enable developers and companies to **build, analyze, operate, and scale** APIs in **secure** environments. API management can be delivered on-premises, through the cloud, or using a hybrid on-premises – SaaS."*

https://aws.amazon.com/api-gateway/api-management/

AWS API management landscape



https://aws.amazon.com/api-gateway/api-management/

Important features of API management tools

- **API ACCESS CONTROL**: APIs should be built using access controls, commonly known as authentication and authorization, that grant users permission to access certain systems, resources, or information.
- **API PROTECTION**: API protections include API keys for identification, API secrets, and application authorization tokens that can be verified.
- **API CREATION AND DESIGN**: APIs allow web applications to interact with other applications. You can create and define different types of APIs such as RESTful APIs and WebSocket APIs.
- **SUPPORT FOR HYBRID MODELS**: A RESTful API is a group of resources and methods, or endpoints, that leverage an HTTP request type. A WebSocket API maintains a persistent connection between connected clients.
- **HIGH PERFORMANCE**: Highly performant APIs depend on code, the separation of functionalities, and on underlying data structure and data architecture.
- **CUSTOMIZABLE DEVELOPER PORTAL**: API developer portals connect API publishers with API subscribers. They enable self-service API publishing and allow potential API customers to easily discover APIs they can use.

https://aws.amazon.com/api-gateway/api-management/

Representation State Transfer (REST)

"…is a set of architectural contraints, most commonly applied using HTTP as the underlying transport protocol…"

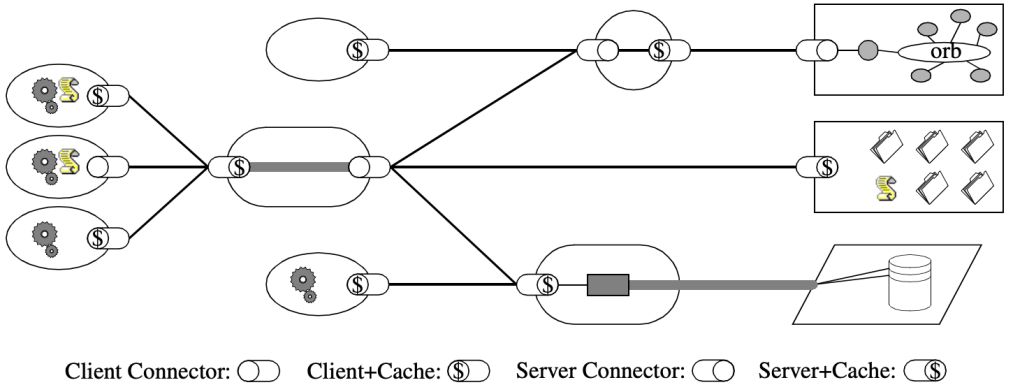


Figure 5-8. REST

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* University of California, Irvine.

Representation State Transfer (REST)

To be considered RESTful your API must ensure that:

1. A **producer-to-consumer** interaction is modeled where the producer models resources the consumer can interact with.
2. Requests from producer to consumer are **stateless**, meaning that the producer doesn't cache details of a previous request. In order to **build up a chain of requests** on a given resource, the consumer must send any required information to the producer for processing.
3. Request are **cachable**, meaning the producer can provide hints to the consumer where this is appropriate. In HTTP this is often provided in information contained in the header.
4. A **uniform interface** is transmitted to the consumer.
5. It is a **layered system**, abstracting away the complexity of systems sitting behind the REST interface. For exemple, the consumer should not know or care if they are interacting with a database or other services.

Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2021. A Systematic Review of API Evolution Literature. ACM Comput. Surv. 54, 8, Article 171 (November 2022), 36 pages. https://doi.org/10.1145/3470133

REST data elements I

The key abstraction of information in REST is a **resource**.

Any information that can be named can be a resource:

- a document or image,
- a temporal service (e.g. “today’s weather in Los Angeles”),
- a collection of other resources,
- a non-virtual object (e.g. a person), and so on.

In other words, any concept that might be the target of an author’s hypertext reference must fit within the definition of a resource.

A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time.

Table 5-1. REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* University of California, Irvine.

REST data elements II

REST components perform actions on a resource by using a **representation** to capture the current or intended state of that resource and transferring that representation between components.

A representation is a sequence of bytes, plus representation metadata to describe those bytes.

Other commonly used but less precise names for a representation include: document, file, and HTTP message entity, instance, or variant.

Table 5-1. REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* University of California, Irvine.

REST data elements III

A representation is a sequence of bytes, plus **representation metadata** to describe those bytes.

The data format of a representation is known as a **media type**.

Some media types are intended for automated processing, some are intended to be rendered for viewing by a user, and a few are capable of both. **Composite media types can be used to enclose multiple representations in a single message.**

Response messages may include both representation metadata and **resource metadata**: information about the resource that is not specific to the supplied representation.

Table 5-1. REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* University of California, Irvine.

REST data elements III

Control data defines the **purpose of a message between components**, such as the action being requested or the meaning of a response. It is also used to parameterize requests and override the default behavior of some connecting elements. For example, cache behavior can be modified by control data included in the request or response message.

A representation can be included in a message and processed by the recipient according to the **control data** of the message and the nature of the media type.

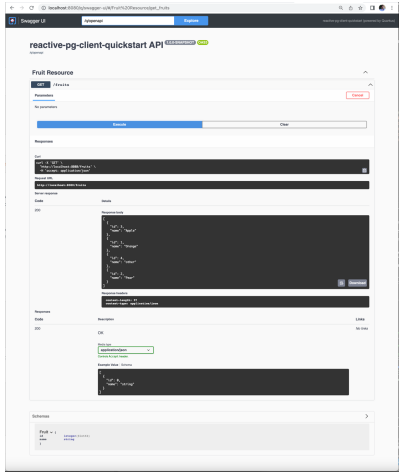
Table 5-1. REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* University of California, Irvine.

REST over HTTP

TÉCNICO LISBOA



An **HTTP request** is:

- A verb (aka method), most of the time one of GET, POST, PUT, DELETE or PATCH
- A URL
- Headers (key-value pairs)
- Optionally a body (aka payload, data)

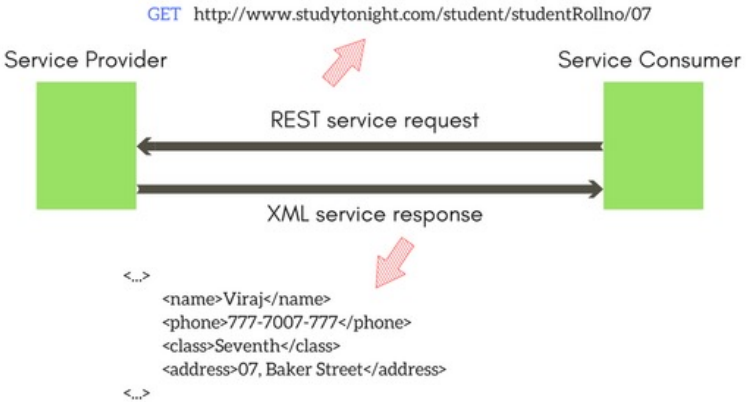
An **HTTP response** is:

- A status, most of the time one of 2xx (successful), 4xx (client error) or 5xx (server error)
- Headers (key-value pairs)
- A body (aka payload, data)

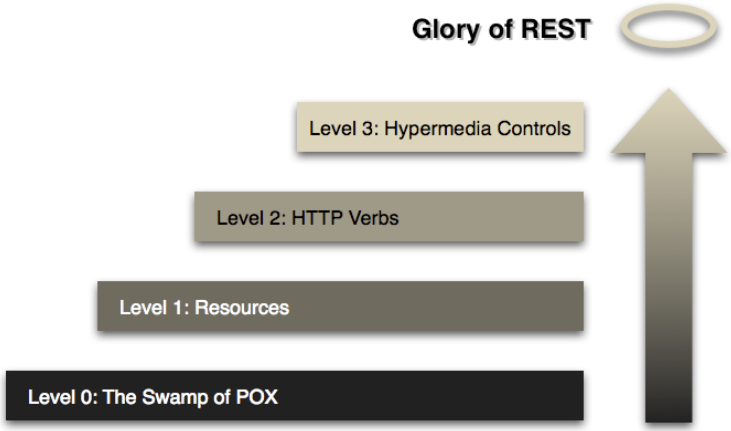
HTTP verbs characteristics:

- Verbs that have a body: POST, PUT, PATCH
- Verbs that must be safe (i.e. that mustn't modify resources): GET
- Verbs that must be idempotent (i.e. that mustn't affect resources again when run multiple times): GET (nullipotent), PUT, DELETE

	body	safe	idempotent
GET	✗	✔	✔
POST	✔	✗	✗
PUT	✔	✗	✔
DELETE	✗	✗	✔
PATCH	✔	✗	✗



Richardson maturity heuristics



- Level 0: The starting point for the model is using HTTP as a transport system for remote interactions, but without using any of the mechanisms of the web. Essentially, using HTTP as a tunneling mechanism for remote interaction mechanism, usually based on Remote Procedure Invocation.
- Level 1 – introduce **Resources**: rather than making all our requests to a singular service endpoint, start talking to individual resources.
  - tackles the question of handling complexity by using divide and conquer, breaking a large service endpoint down into multiple resources.
- Level 2 - using the **HTTP verbs**: as closely as possible to how they are used in HTTP itself.
  - introduces a standard set of verbs so that we handle similar situations in the same way, removing unnecessary variation.
- Level 3 - Hypermedia Controls **introduces HATEOAS** (Hypertext As The Engine Of Application State).
  - Level 3 introduces discoverability, providing a way of making a protocol more self-documenting.

M. Fowler. Richardson maturity model: steps toward the glory of rest. Online at http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

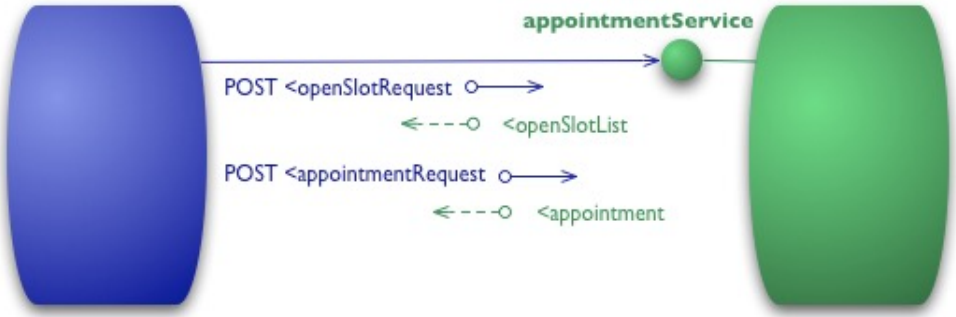
Level 0

Request:

```
POST /appointmentService HTTP/1.1
[various other headers]
<openSlotRequest date = "2010-01-04" doctor = "mjones"/>
```

Response:

```
HTTP/1.1 200 OK
[various headers]
<openSlotList>
    <slot start = "1400" end = "1450">
        <doctor id = "mjones"/>
    </slot>
    <slot start = "1600" end = "1650">
        <doctor id = "mjones"/>
    </slot>
</openSlotList>
```

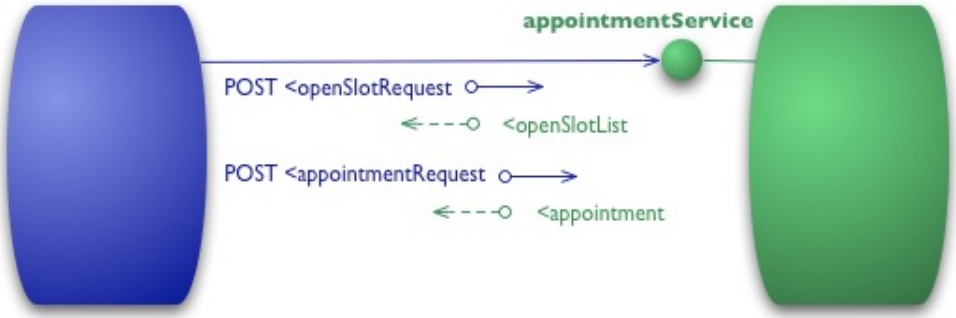


M. Fowler. Richardson maturity model: steps toward the glory of rest. Online at http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

Level 0

Request:

```
POST /appointmentService HTTP/1.1
[various other headers]
<appointmentRequest>
        <slot doctor = "mjones" start = "1400" end = "1450"/>
        <patient id = "jsmith"/>
</appointmentRequest>
```



Response:

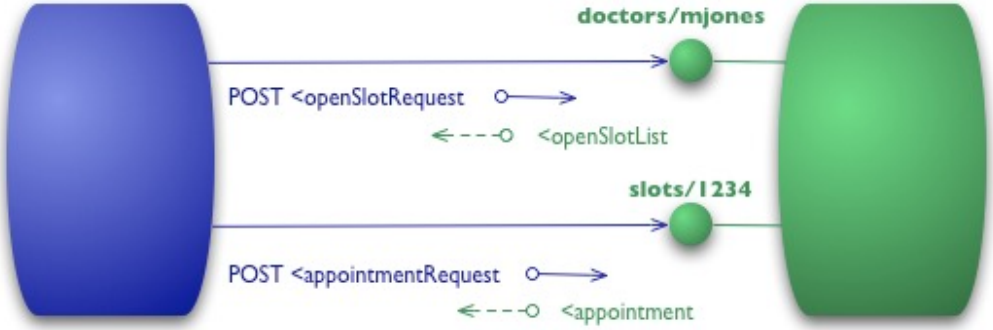
```
HTTP/1.1 200 OK
[various headers]
<appointment>
        <slot doctor = "mjones" start = "1400" end = "1450"/>
        <patient id = "jsmith"/>
</appointment>
```

M. Fowler. Richardson maturity model: steps toward the glory of rest. Online at http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

Level 1

Request:

```
POST /doctors/mjones HTTP/1.1
[various other headers]
<openSlotRequest date = "2010-01-04"/>
```



Response:

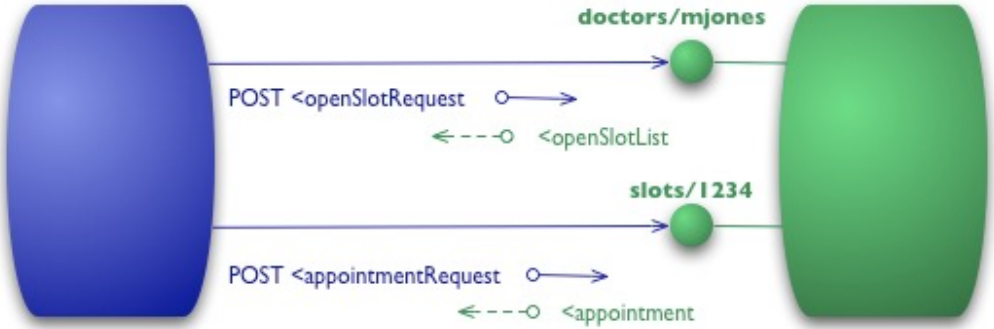
```
HTTP/1.1 200 OK
[various headers]
<openSlotList>
        <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
        <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

M. Fowler. Richardson maturity model: steps toward the glory of rest. Online at http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

Level 1

Request:

```
POST /slots/1234 HTTP/1.1
[various other headers]
<appointmentRequest>
        <patient id = "jsmith"/>
</appointmentRequest>
```



Response:

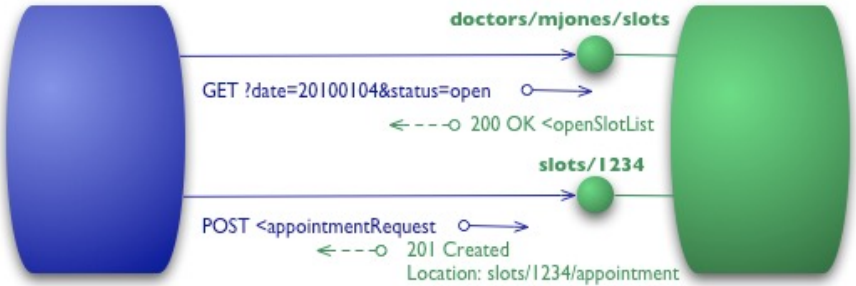
```
HTTP/1.1 200 OK
[various headers]
<appointment>
        <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
        <patient id = "jsmith"/>
</appointment>
```

M. Fowler. Richardson maturity model: steps toward the glory of rest. Online at http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

Level 2

Request:

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```



Response:

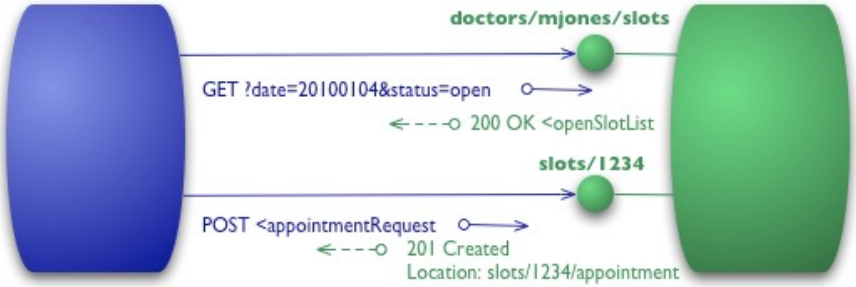
```
HTTP/1.1 200 OK
[various headers]
<openSlotList>
    <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
    <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

M. Fowler. Richardson maturity model: steps toward the glory of rest. Online at http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

Level 2

Request:

```
POST /slots/1234 HTTP/1.1
[various other headers]
<appointmentRequest>
        <patient id = "jsmith"/>
</appointmentRequest>
```



Response:

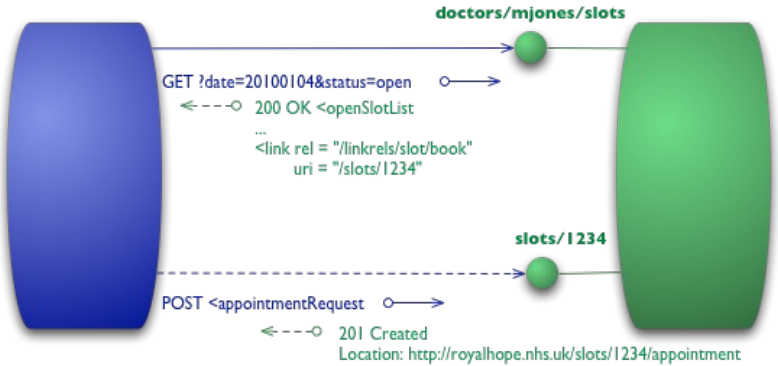
```
HTTP/1.1 201 Created Location: slots/1234/appointment
[various headers]
<appointment>
        <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
        <patient id = "jsmith"/>
</appointment>
```

M. Fowler. Richardson maturity model: steps toward the glory of rest. Online at http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

Level 3

Request:

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```



Response:

```
HTTP/1.1 200 OK
[various headers]
<openSlotList>
    <slot id = "1234" doctor = "mjones" start = "1400" end = "1450">
        <link rel = "/linkrels/slot/book" uri = "/slots/1234"/>
    </slot>
    <slot id = "5678" doctor = "mjones" start = "1600" end = "1650">
        <link rel = "/linkrels/slot/book" uri = "/slots/5678"/>
    </slot>
</openSlotList>
```

M. Fowler. Richardson maturity model: steps toward the glory of rest. Online at http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

Level 3

Request:

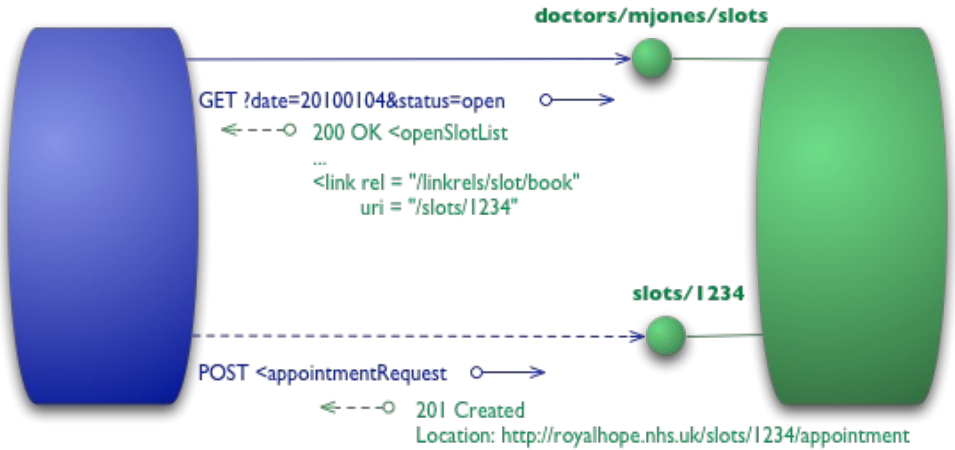
```
POST /slots/1234 HTTP/1.1

[various other headers]

<appointmentRequest>

           <patient id = "jsmith"/>

</appointmentRequest>
```

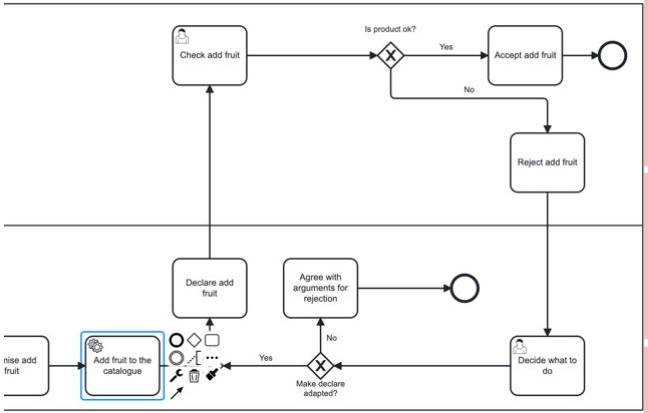


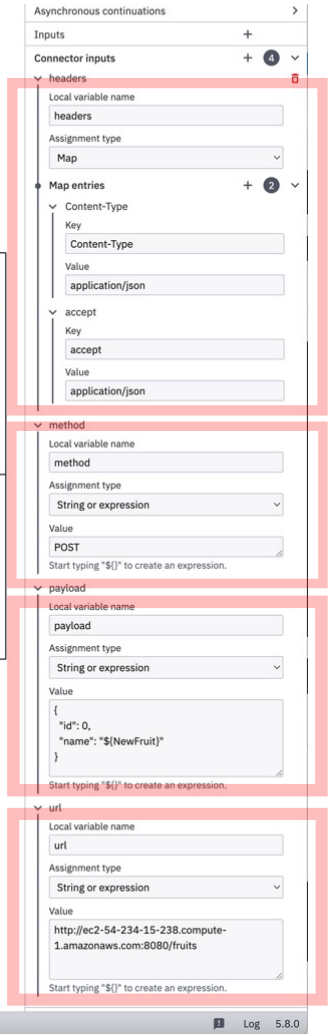
Response:

```
HTTP/1.1 201 Created Location: http://royalhope.nhs.uk/slots/1234/appointment
[various headers]
<appointment>
           <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
           <patient id = "jsmith"/>
           <link rel = "/linkrels/appointment/cancel" uri = "/slots/1234/appointment"/>
           <link rel = "/linkrels/appointment/addTest" uri = "/slots/1234/appointment/tests"/>
           <link rel = "self" uri = "/slots/1234/appointment"/>
           <link rel = "/linkrels/appointment/changeTime" uri = "/doctors/mjones/slots?date=20100104&status=open"/>
           <link rel = "/linkrels/appointment/updateContactInfo" uri = "/patients/jsmith/contactInfo"/>
           <link rel = "/linkrels/help" uri = "/help/appointment"/>
</appointment>
```

M. Fowler. Richardson maturity model: steps toward the glory of rest. Online at http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

Camunda REST example





headers

verb

payload

URL

```
curl -X 'POST' \
  'http://ec2-54-234-15-238.compute-1.amazonaws.com:8080/fruits' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 0,
  "name": "sd"
}'
```

Open API

- REST APIs are technological implementation of services and do not account with contracting issues
- Usually an additional document, or wiki documents the API usage. However, those are hard to maintain and testing is not automating
- *“The **OpenAPI Specification (OAS)** defines a standard, language-agnostic interface to HTTP APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic”*

https://swagger.io/specification/

Swagger

Swagger has evolved into one of the most widely used open source tool sets for developing APIs with rich support for the OpenAPI Specification, AsyncAPI specification, JSON Schema and more.

https://swagger.io/tools/open-source/

The contract of an API encompasses:

- **Title**
- **Endpoint**
- **Method**
- **URL parameters**
- **Message payload**
- **Header parameters**
- **Response code**
- **Error code**
- **A sample request and response**
- **Tutorials and walkthrough**
- **Service-level agreement**

Swagger for Everyone

Swagger open source and pro tools have helped millions of API developers, teams, and organizations deliver great APIs.

Design

Design APIs in a powerful editor which visually renders your OpenAPI or AsyncAPI definition and provides real-time error feedback.

Swagger Editor

Build

Build and enable consumption of your API by generating server stubs and client SDKs with minimal plumbing.

Swagger Codegen

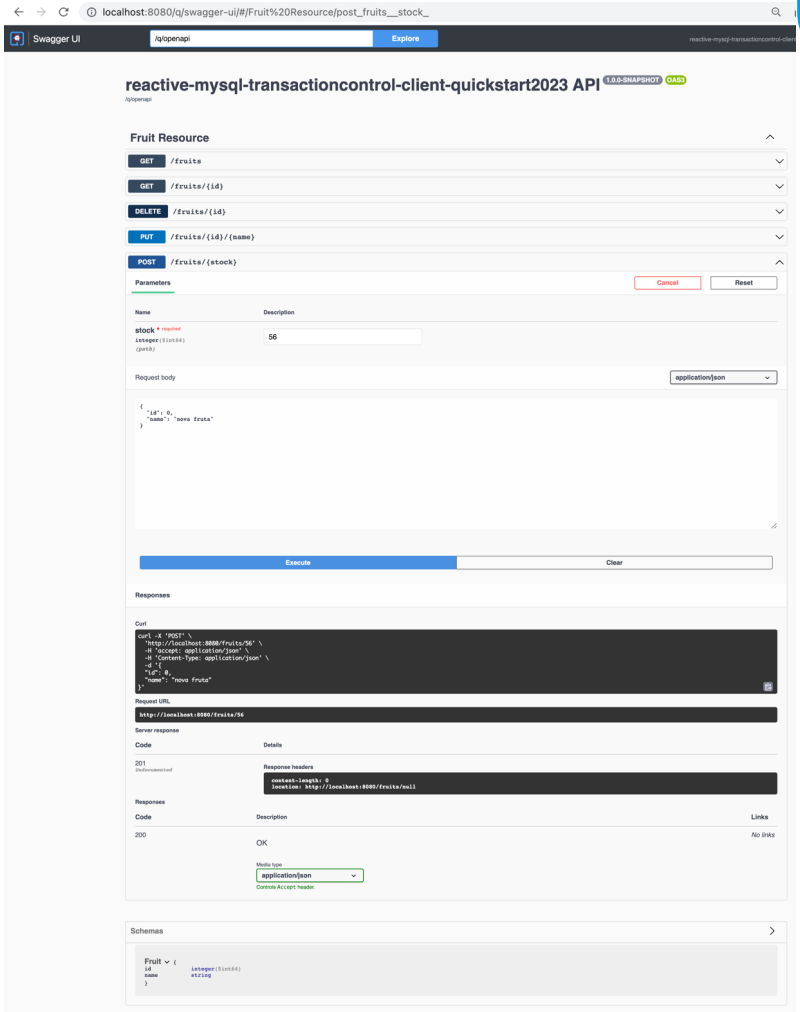
Document

Automatically generate documentation from your OpenAPI definition for visual interaction, and easier consumption.

Swagger UI

Swagger

- Example of swagger embedded with Quarkus to facilitate the documentation and testing of a Project



GraphQL

- Is an API technology developed by Facebook to address some of the challenges faced with RESTful APIs, particularly in complex applications where multiple data sources need to be queried.
- Allows the client to request only the specific data it needs, using **a declarative query language**, reducing over-fetching and under fetching issues common with RESTful APIs.
- Provides a flexible and efficient way for front-end applications, making it a choice for applications with complex data requirements and a variety of front-end interfaces. It allows developers to define their queries precisely, resulting in fewer network requests and less data transfer.
- However, comes with additional complexity compared to RESTful APIs and requires more careful management of server-side schema

Hygraph, "Best practices for REST API design" https://hygraph.com/learn/graphql, 2024.
J. P. Olaf Hartig, "Semantics and Complexity of GraphQL" https://dl.acm.org/doi/10.1145/3178876.3186014, 2018.

gRemote Procedure Call (gRPC)

- Developed by Google, is a high-performance, open-source API framework that uses HTTP2 for transport and protocol buffers as data format. gRPC is designed to provide efficient communication between microservices, making it highly suitable for blockchain backends that require fast and reliable communication with front-end clients.
- One of the advantage is its ability to support **bi-directional streaming**, which is particularly useful for real-time applications where low latency is critical. Unlike RESTful APIs, gRPC allows the server to push updates to the client, making it suitable for applications that require continuous data streams.
- gRPC's performance and efficiency make it an excellent option for projects that demand low latency and high data throughput, such as real-time trading platforms or blockchain monitoring services.
- However, gRPC is more complex to implement than REST or GraphQL and may require more setup, particularly for handling serialization and managing HTTP2 connections. Despite this, its performance benefits are invaluable for projects with stringent speed and reliability requirements.

a. S. C. Spotlight, "API Types and Protocols" https://www.browserstack.com/guide/angular-vs-react-vs-vue, 2024.

MuleSoft, "What is a gRPC API and how does it work?" https://www.mulesoft.com/api-university/what-grpc-api-and-how-does-it-work, 2024.

D. Strmecki, "REST vs. GraphQL vs. gRPC – Which API to Choose?" https://www.baeldung.com/rest-vs-graphql-vs-grpc, 2024.

TÉCNICO LISBOA

Enterprise Integration

API *Gateway Systems*

Prof. Sérgio Guerreiro
Sergio.guerreiro@tecnico.ulisboa.pt

Department of Computer Science and Engineering
Instituto Superior Técnico / Universidade de Lisboa
INESC-ID
URL: http://www.inesc-id.pt
Rua Alves Redol, 9
1000-029 Lisboa
Portugal

Gateway (in general terms)

- *According to Gartner[1], a gateway in computer networking:*

  "gateway converts information, data, or other communications from one protocol or format to another"

- Moreover, it must allow communication in both directions and maintain the connections. A gateway can also be seen as a "supervised entry point" through which messages (carries of information/knowledge) can enter a particular environment undergiven conditions and rules applied.

[1] https://www.gartner.com/en/information-technology/glossary/gateway

API Gateway System (in general terms)

Provides or integrates with third-party gateways for runtime management, security, policy enforcement, throttling, operational control and usage monitoring for APIs.

Specific:

- Kong Gateway, is a commercial version of its open-source API gateway based on NGINX and OpenResty.
- Kong Gateway works as a reverse proxy[1] that manage, configure, and route requests to other APIs.
- Kong Gateway runs in front of any RESTful API and can be extended through modules and plugins. It's designed to run on decentralized architectures, including hybrid-cloud and multi-cloud deployments.

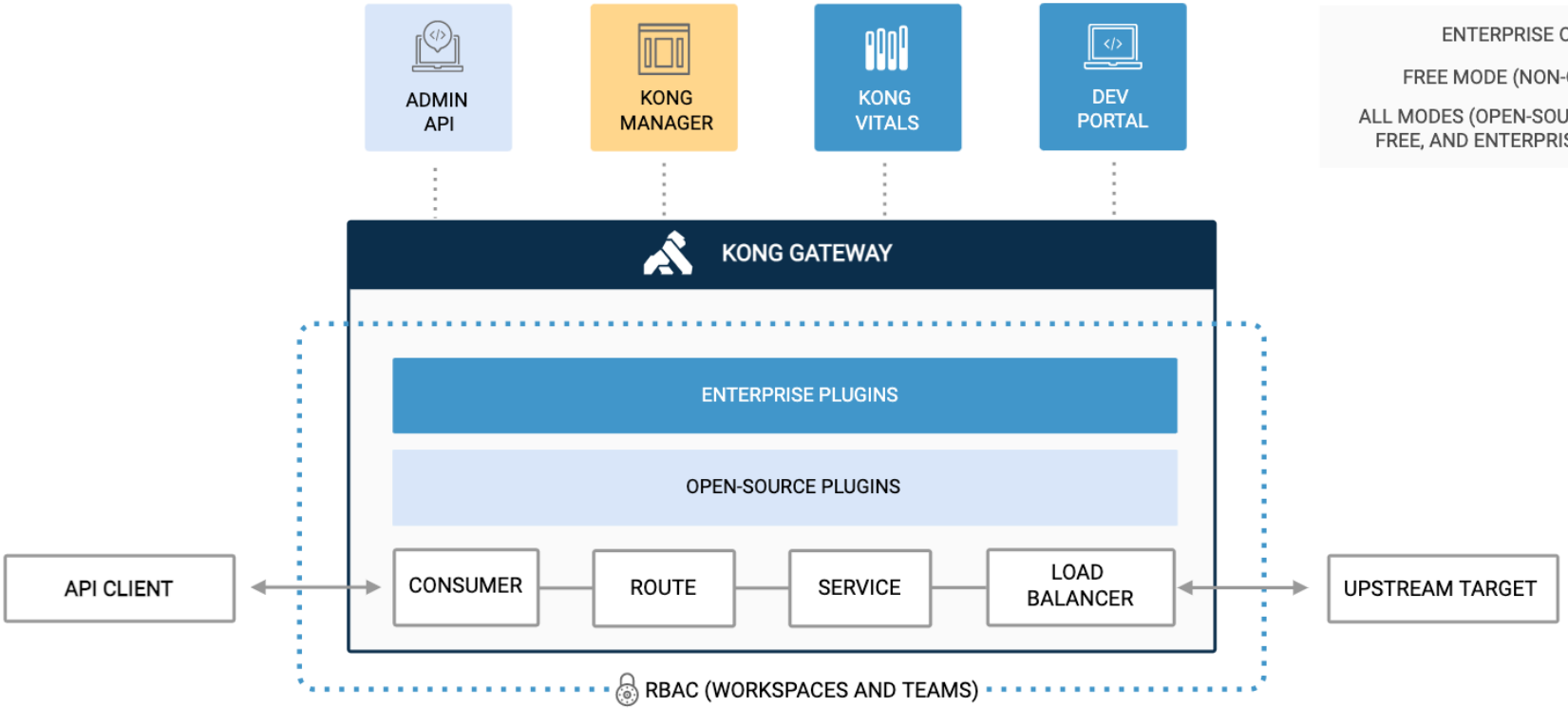
[1] A reverse proxy server is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate backend server

TÉCNICO LISBOA

Advantages of an API Gateway System

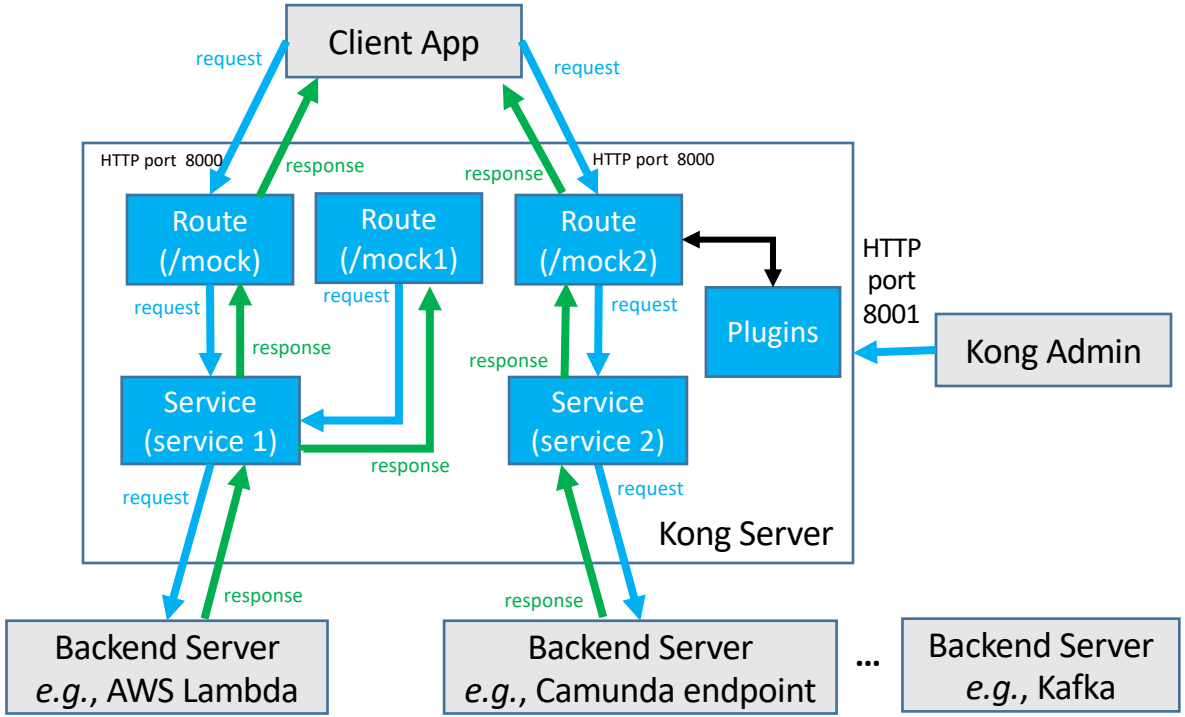
- Enable new consumers without coding or configuration
  - Interface and Volume Scoping
  - Routing
- Keep unwanted people & robots out of your systems
  - Authorization and Authentication
  - Threat Protection
- Understand what is happening with your APIs
  - Discovery, Usage, and Concerns
  - History

API gateway core functionalities

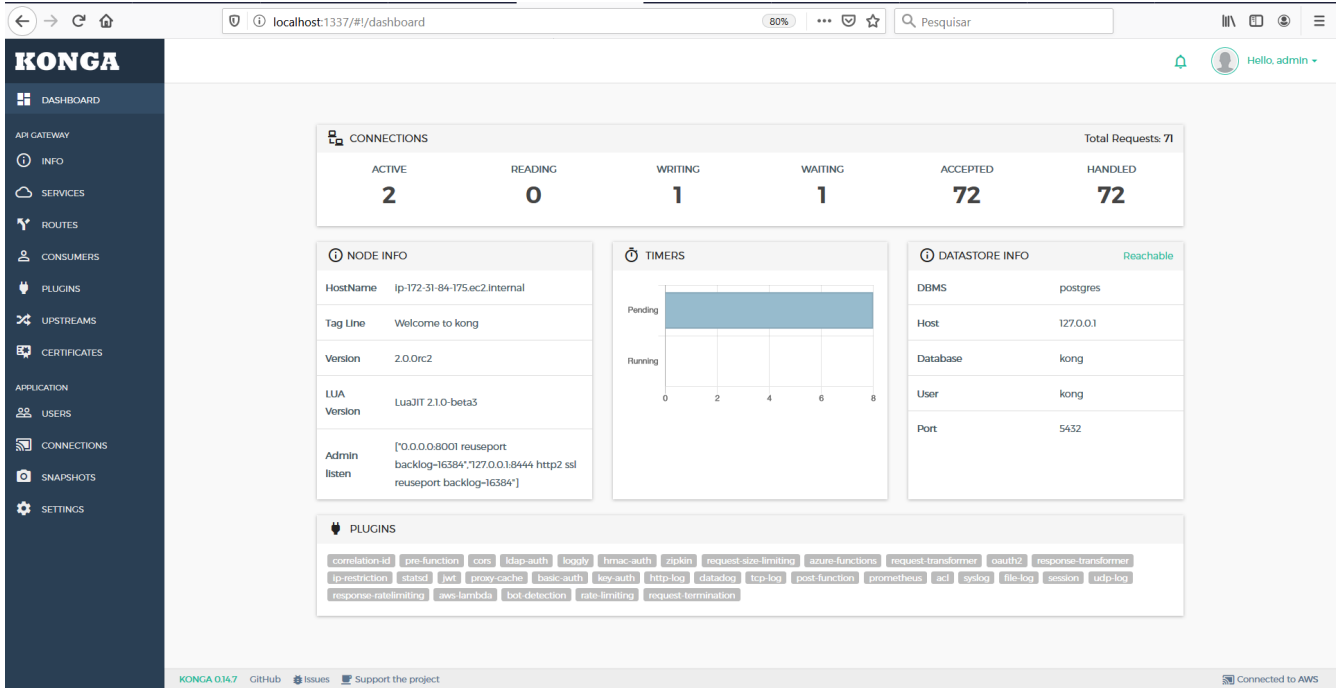


Kong main concepts

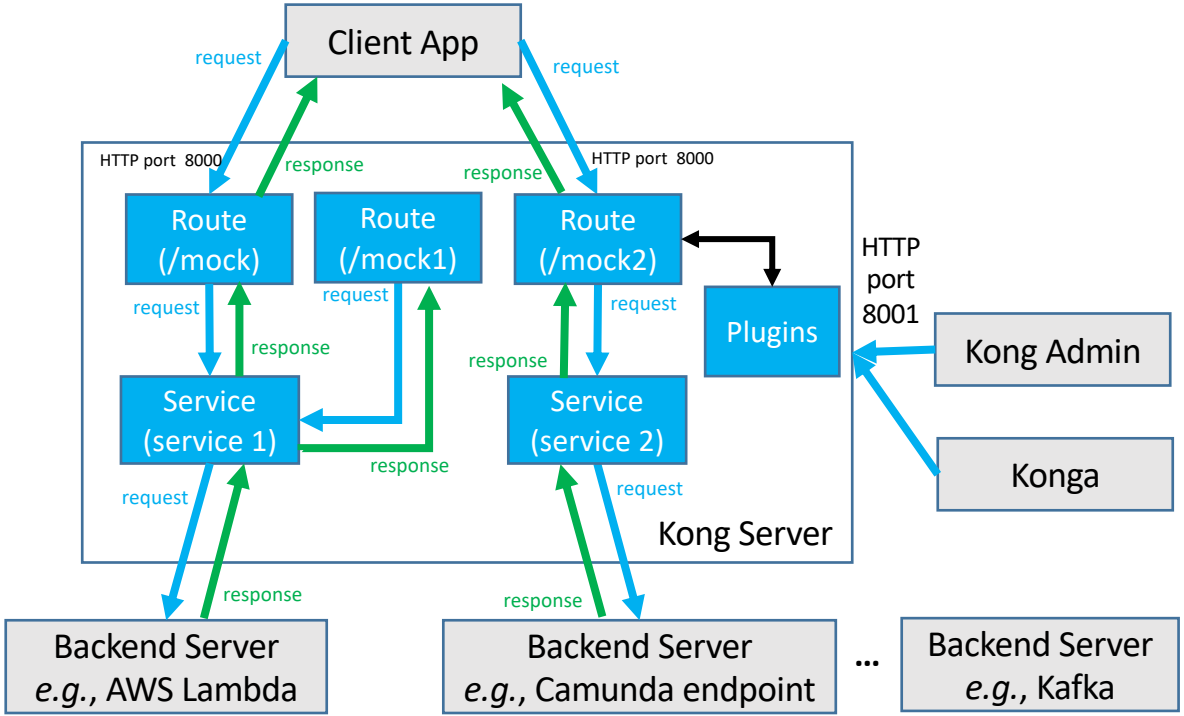
- **Service**: is the name Kong uses to refer to the upstream APIs and microservices it manages
- **Route**: specify how (and if) requests are sent to their Services after reaching Kong
- A single Service can have multiple Routes.
- **Plugin**: extension to the Kong Gateway, written in Lua or Go
- Port 8001 is used for managing APIs
- Port 8000 for service invocation



Konga – managing Kong API Gateway easier



Konga integrated with Kong



The full lifecyle of API management

- **Design** APIs – solutions such as swagger that are able to facilitate the design, build and documentation
- **Run** APIs – solutions such as Kong that are able to connect to endpoints and return the responses
- **Secure** APIs – solutions such as Kong plug-ins that are able to apply fine-grained security and traffic policies to services
- **Govern** APIs – solutions such as Konga that are able to gain real-time visibility of the services

TÉCNICO LISBOA

Enterprise Integration

Identity Management

Prof. Sérgio Guerreiro
Sergio.guerreiro@tecnico.ulisboa.pt

Department of Computer Science and Engineering
Instituto Superior Técnico / Universidade de Lisboa
INESC-ID
URL: http://www.inesc-id.pt
Rua Alves Redol, 9
1000-029 Lisboa
Portugal

AAA framework

- **Authentication** (A) - It is the challenge process by which identity claims are tested and validated. The purpose of the authentication is ensuring that a particular user is really who he or she claims to be. *Then access is granted to a system if granted by the authorization' process execution*

- **Authorization** (A) - After authentication, the authorization process enforces the system' policies, granular access control, and user privileges. It also establishes the tasks and activities that users can perform within those authorized resources

- **Accountability** (A) - is about measuring what's happening within the system, collecting and logging data on user sessions, e.g., length of time, type of session, and resource usage. It offers a clear audit trail for compliance and business purposes

Authentication (A)

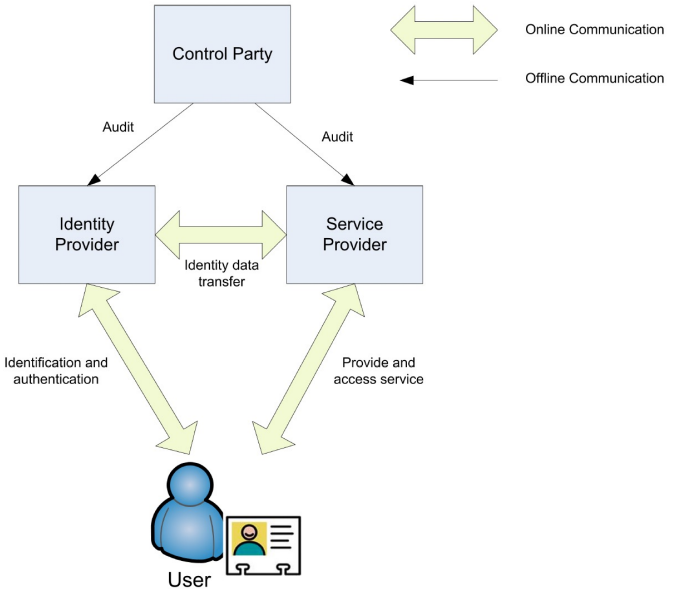


Figure 1: Entities involved in an identity management-system.

…is the **challenge** process by which identity claims are tested and validated, where challenge is secret and could be:

- Password
- Physical identifies: card, physical key
- Biometric data: face, iris, fingerprint
- Cypher/decipher challenge with keys

Authentication in applications/data stores within an organization:

- Typically, single sign-on using LDAP to interact with Active Directory, Kerberos

Zwattendorfer, B., Zefferer, T., & Stranacher, K. (2014). An Overview of Cloud Identity Management-Models. WEBIST (1), 82-92.

Authentication (A)

But, on the Cloud, multiple options possible:

- Each organization does its authentication => multiple identities
- One organization authenticates all participants
- For each business partnership, an organization (s) is chosen to authenticate
- Allow each organization to use its authentication framework and create a loosely coupled mechanism allowing the reuse of identities and authentication

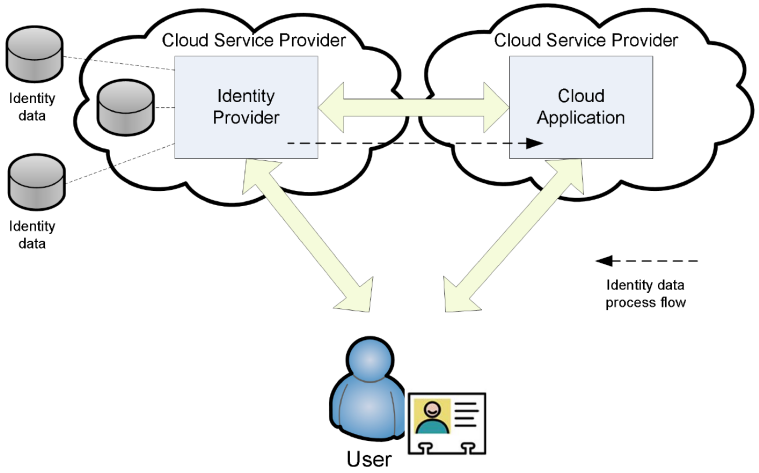


Figure 4: Identity from the Cloud-Model.

Zwattendorfer, B., Zefferer, T., & Stranacher, K. (2014). An Overview of Cloud Identity Management-Models. WEBIST (1), 82-92.

Authorization (A)

- Today, countless access control models (ACM) solutions are ava in the academy and industry. Some examples:
  - Discretionary access control (DAC)
  - Mandatory access control (MAC)
  - Role-based access control (RBAC)
  - Time-role-based access control (TRBAC)
  - Attribute-based access control (ABAC)
  - Orcon or Chinese wall

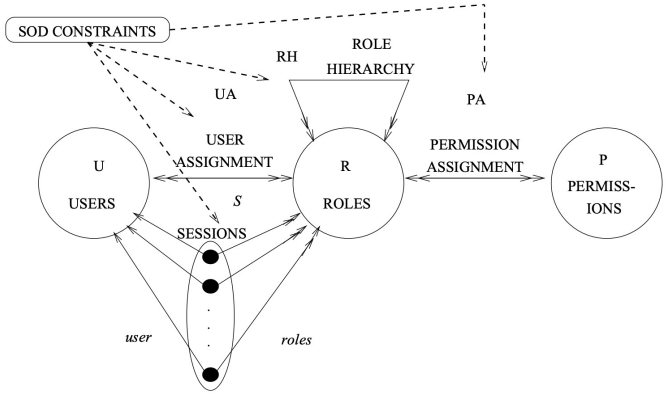


Figure 10: Symmetric RBAC—Dynamic SOD

Ravi Sandhu, David Ferraiolo, and Richard Kuhn. 2000. The NIST model for role-based access control: towards a unified standard. In Proceedings of the fifth ACM workshop on Role-based access control (RBAC '00). Association for Computing Machinery, New York, NY, USA, 47-63. https://doi.org/10.1145/344287.344301

- Nevertheless, in the majority of situations, the recognized development of ACM, these solutions specifies and implements the structural security access concerns of a single organizational silo

Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security, 4(3), 224-274. doi:10.1145/501978.501980

eXtensible Access Control Markup Language (XACML)

- XACML is an XML-based standard markup language for specifying access control policies. The standard, published by OASIS: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml defines a declarative fine-grained, attribute-based access control policy language and a processing model describing how to evaluate access requests according to the rules defined in policies

- XACML is an attribute-based access control system
  - Input, e.g., the information about the subject accessing a resource, the resource to be addressed, and the environment
  - Output, the decision of whether access is granted or not

eXtensible Access Control Markup Language (XACML)

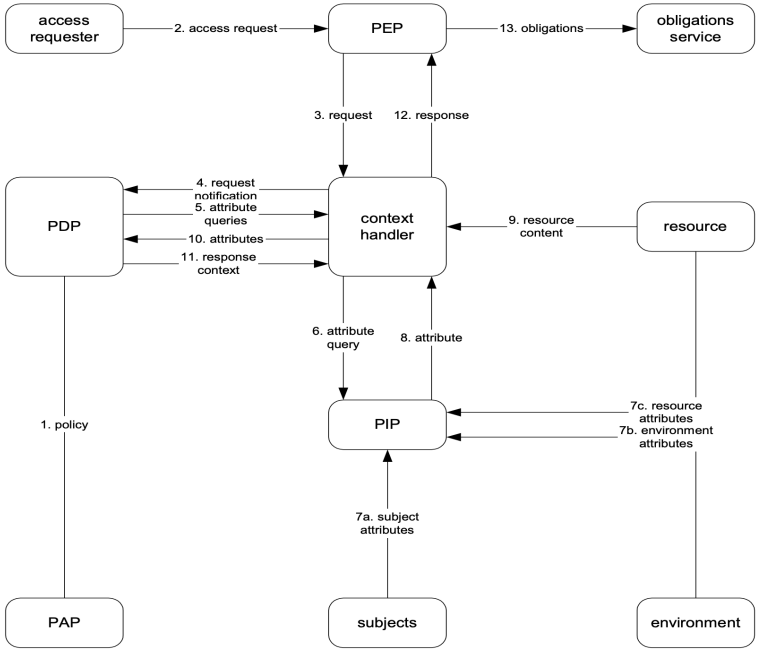
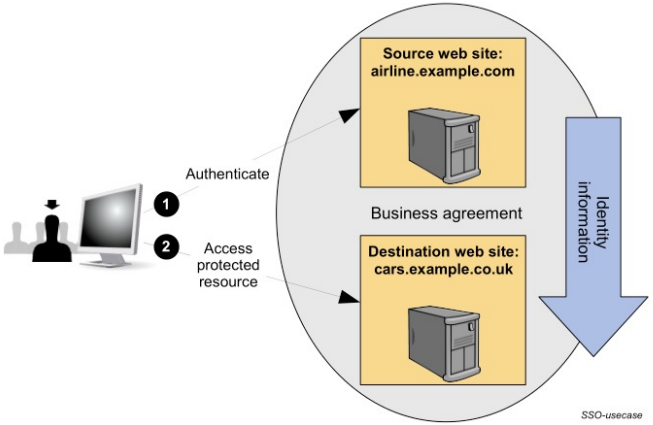


Figure 1 - Data-flow diagram

- Policy - A set of rules, an identifier for the rule-combining algorithm and (optionally) a set of obligations or advice. May be a component of a policy set
- PAP - **Policy Administration Point** is the system entity that creates a policy or policy set
- PEP - **Policy Enforcement Point** is the system entity that performs access control, by making decision requests and enforcing authorization decisions.
- Obligation - An operation specified in a rule, policy or policy set that should be performed by the PEP in conjunction with the enforcement of an authorization decision
- PDP - **Policy Decision Point** is the system entity that evaluates applicable policy and renders an authorization decision
- PIP - **Policy Information Point** is the system entity that acts as a source of attribute values

SAML

- SAML is a well-established standard developed by the OASIS consortium for authentication and authorization. Developed during early 2000s, now in the version 2.0, it is an XML-based standard heavily influenced by the enterprise technologies popular at the time, especially the WS- stack (which is a set of protocols standardized by the W3C that define how XML- and more precisely SOAP-based1 web services should be implemented and includes publications defining SOAP, WS-Addressing, WS-Policy, WSDL etc.). Similar to some other XML-based standards, it defines a hierarchy of basic concepts - assertions, protocols, bindings and profiles. It was developed mostly for enterprise and business needs and is widely adopted in these domains as well as in public services mainly through Shibboleth, an open-source identity provider that builds upon SAML 2.0.
- The most common SAML profile (simplified, profiles are essentially use-cases) is multi-domain web single sign-on and includes interaction between three parties - a **user**, an **Identity provider**, and a **Service Provider**
- In this scenario, the user already has an established session with one web site, during which the user is redirected to a new web site. The original website acts as an IdP and relays information to the redirected website that acts as a Service provider.
- The relayed information effectively authenticates the user through a SAML message allowing for a session between the user and the redirected website to be established. In SAML terms, the first website is generally referred to as the asserting party and the second as the relying party.
- This use-case depends on various other interactions, including an agreement between parties what information to forward, what transport layer to use, whether to link user representation with both parties or to decouple them, whether to store user data or to rely on transient identifiers that get destroyed after the session is done, etc. SAML 2.0 provides mechanisms to address most of these questions on-the-fly. Additionally, there are many variations of this example flow (e.g. user-initiated instead of AP-initiated as in the example) as well as different flows supported by SAML 2.0, making it a very versatile and robust standard.
- However, this also adds complexity which is often viewed as a drawback of the standard prohibiting it from use in some other scenarios, particularly those where some of the security concerns may be somewhat relaxed. Added to that the XML legacy of the protocol is also often viewed as cumbersome and out-of-date in scenarios demanding more lightweight solutions due to processing power and other technical limitations, e.g. smartphone and similar applications.



Djerasimović, P., & Gledec, G. (2017). *An overview of identity management systems usage*. Varazdin: Faculty of Organization and Informatics Varazdin. Retrieved from https://www.proquest.com/conference-papers-proceedings/overview-identity-management-systems-usage/docview/2125641077/se-2

OpenID

- OpenID Connect is an **authentication protocol** specified and maintained by the OpenID Foundation. The foundation's members are mostly international corporations including well-known names like Google, Microsoft, Oracle, PayPal, Verizon, RSA, VMWare, Deutsche Telekom
- The protocol is defined through a set of OpenID specifications including the core specification (Sakimura, 2014) and accompanying specifications detailing additional services (like dynamic provider discovery, dynamic registration with providers, response types, etc.)
- Unlike the previous versions of OpenID, the current one (final specification launched in 2014) is based on the previously described OAuth 2.0 protocol. **The specification extends and additionally specifies parts of the OAuth specification that have been left open by OAuth**, so OAuth-based mechanisms could be used in authentication scenarios
- For example, one of the most important additions by OpenID is the definition of the ID token - defined are both the contents and the representation of a data structure containing claims made by the authentication server about an end-user's authentication. Some of the other extensions introduced are a list of standard claims about a user, subject identifier types (**enabling an IdP to issue different subject value to each client for the same end-user with the purpose of disabling correlation of end-user information by different clients**), list of allowed encryptions, etc

Djerasimović, P., & Gledec, G. (2017). *An overview of identity management systems usage*. Varazdin: Faculty of Organization and Informatics Varazdin. Retrieved from https://www.proquest.com/conference-papers-proceedings/overview-identity-management-systems-usage/docview/2125641077/se-2

OAuth 2.0

- OAuth 2.0 standard is published and maintained by IETF through a set of RFC documents, the core consisting of RFC 6749 (Hardt, 2012 (1)), RFC 6750 (Hardt, 2012 (2)) and RFC 6819 (McGloin, 2013). OAuth 2.0 is foremost an **delegation protocol**. The user provides permission to a third party (a client in OAuth terms) to access some of user's data at the identity. For example, the user allows a site like fit4life.com read-only access the user's fitness data stored at google.com (an authorization server and resource server in OAuth terms), without necessarily divulging any of the user's other confidential data.

- Being a more modern protocol that targets not only web applications and services but also handheld devices, OAuth makes a clear distinction between public and confidential clients based on their ability to authenticate securely with the authorization server.

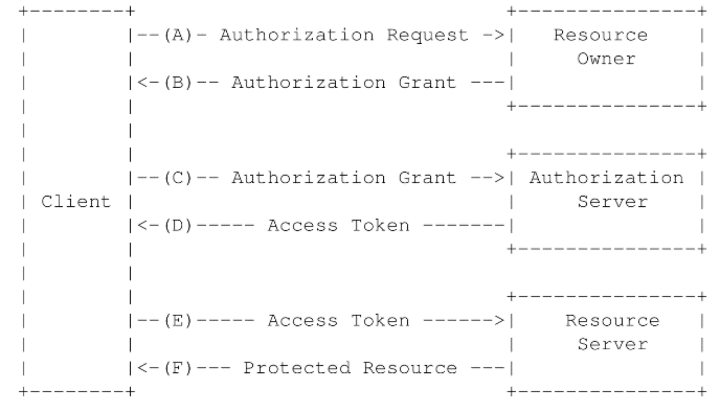
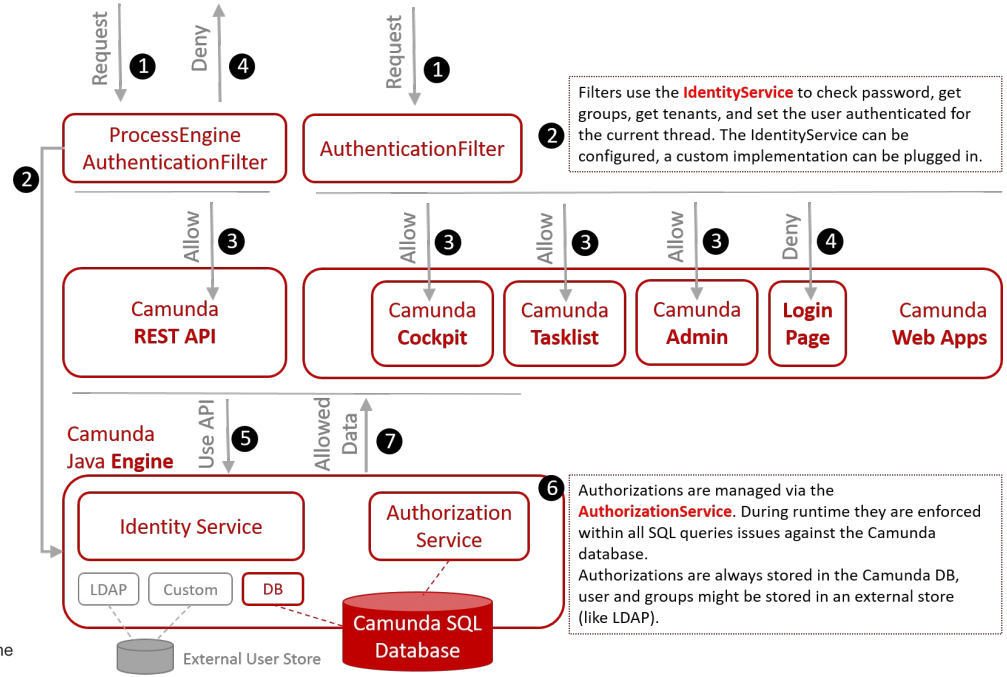


Figure 2. OAuth 2.0 Abstract protocol flow.

Djerasimović, P., & Gledec, G. (2017). *An overview of identity management systems usage*. Varazdin: Faculty of Organization and Informatics Varazdin. Retrieved from https://www.proquest.com/conference-papers-proceedings/overview-identity-management-systems-usage/docview/2125641077/se-2

Camunda with Authentication and Authorization



1. A *request* is either asking for a REST API endpoint or one of the web applications functionalities.
2. The `ProcessEngineAuthenticationFilter` (for REST) or the `AuthenticationFilter` (for the web applications) check the user's authentication credentials via the *IdentityService*. The filters retrieve groups and tenant memberships and set the authenticated user for the current thread in the engine.
3. The request is *allowed*.
4. The request might also be *denied*, in case the authentication fails (e.g. because the username is unknown or the password does not match). For the web applications, a denied request is redirected to the login page.
5. All applications use Camunda's Java API internally.
6. Under the hood, the engine enforces authorizations by instrumenting SQL queries. That means you can never get any data from a query the current user is not authorized for.
7. As a consequence, only allowed and *accessible data* will be presented to the user.

Identity and Authorization services can be replaced, e.g., with OAuth2.0 or OpenID tools

OAuth2 *vs.* OpenID *vs.* SAML

	OAuth2	OpenID	SAML
Token (or assertion) format	JSON or SAML2	JSON	XML
Authorization	Delegation	No	Yes
Authentication	Pseudo-authentication	Yes	Yes
Security risks	Phishing OAuth 2.0 does not support signature, encryption, channel binding, or client verification. Instead, it relies completely on TLS for confidentiality	Phishing Identity providers have a log of OpenID logins, making a compromised account a bigger privacy breach	XML Signature Wrapping to impersonate any user
Best suited for	API authorization	Single sign-on for consumer apps	Single sign-on for enterprise Note: not well suited for mobile

Consent Management refers to the process and system by which an organization **collects**, **manages**, and **maintains** user consent for the collection and processing of personal data.

It ensures compliance with privacy laws and regulations, such as the **GDPR** (General Data Protection Regulation) and **CCPA** (California Consumer Privacy Act), by allowing users to make informed decisions about what personal data they are willing to share.

The goal is to **give users control over their data** while also providing transparency about how it is being used. It involves obtaining clear consent, documenting it, and ensuring that it can be easily modified or withdrawn by the user at any time.

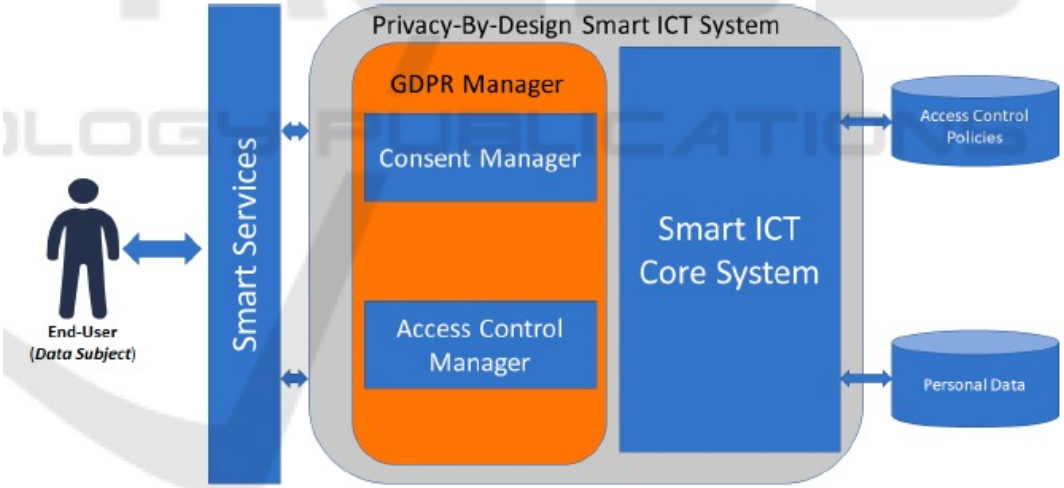
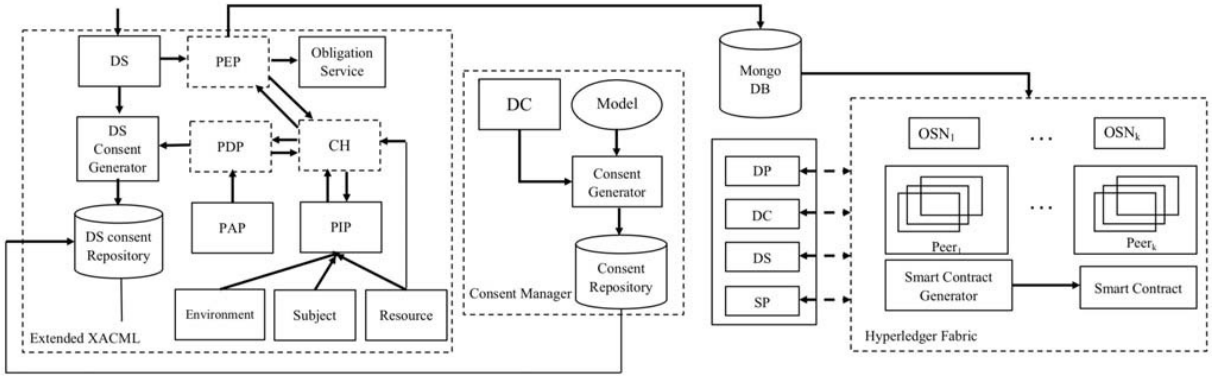
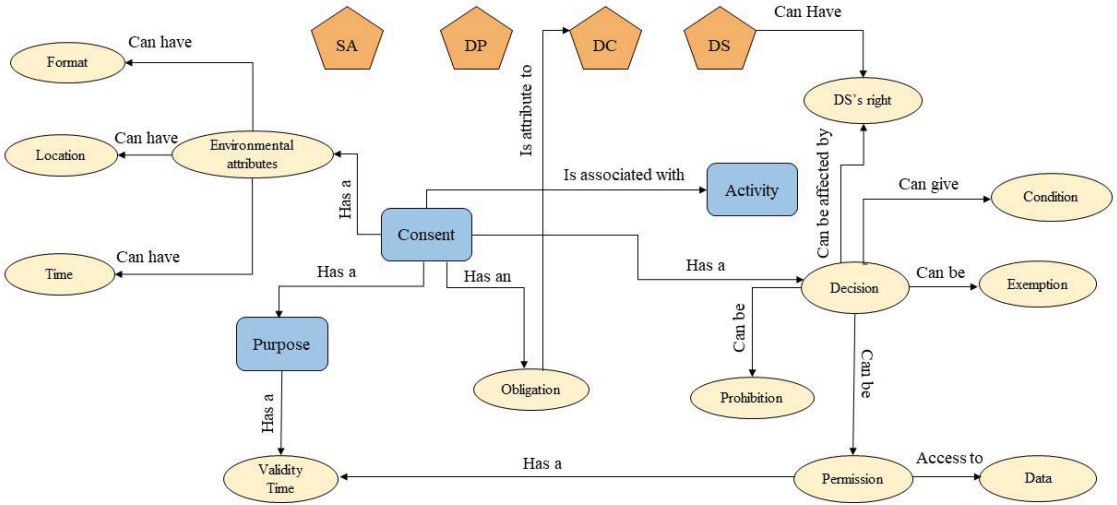


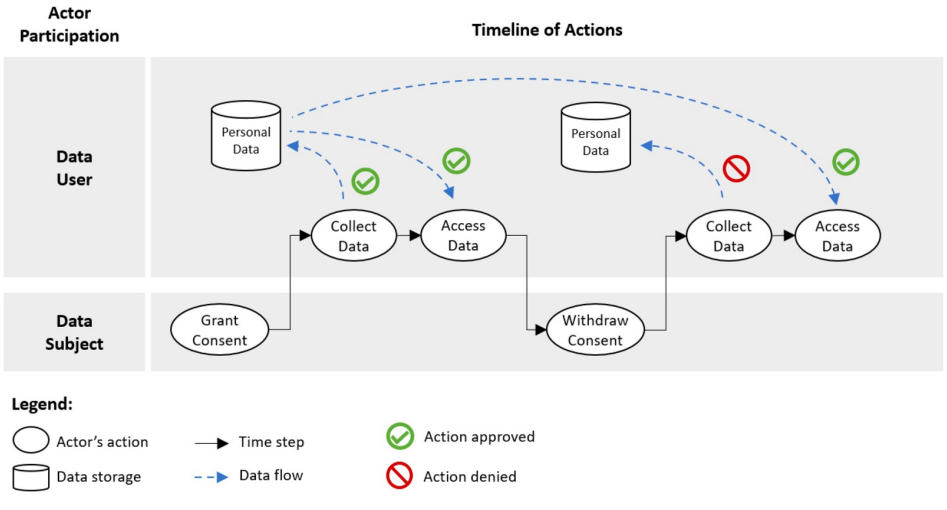
Figure 2: A Privacy-By-Design Smart ICT System Proposal.

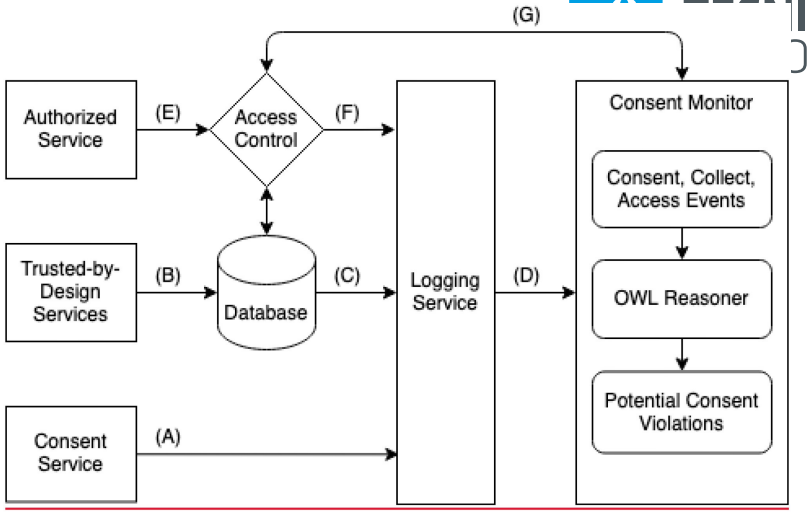
Daoudagh, S., Marchetti, E., Savarino, V., Di Bernardo, R., & Alessi, M. (2021). How to Improve the GDPR Compliance through Consent Management and Access Control. In *ICISSP* (pp. 534-541).

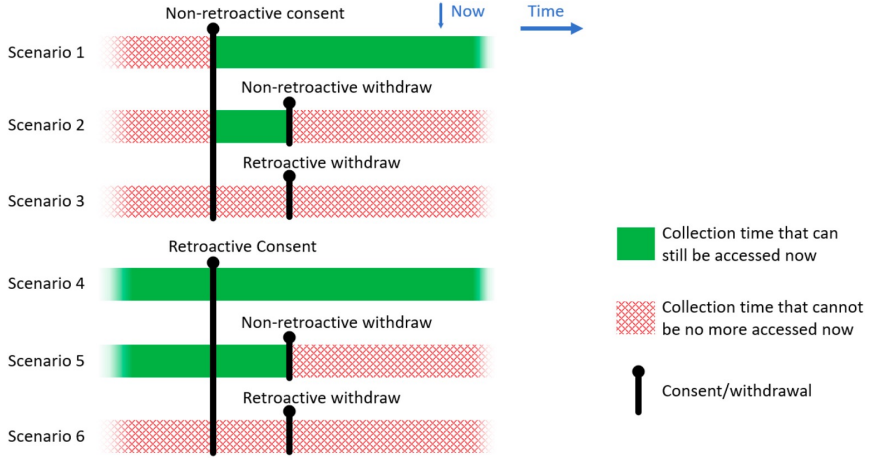




Davari, M., & Bertino, E. (2019, December). Access control model extensions to support data privacy protection based on GDPR. In *2019 IEEE International Conference on Big Data (Big Data)* (pp. 4017-4024). IEEE.







Robol, M., Breaux, T. D., Paja, E., & Giorgini, P. (2023). Consent verification monitoring. *ACM Transactions on Software Engineering and Methodology*, *32*(1), 1-33.

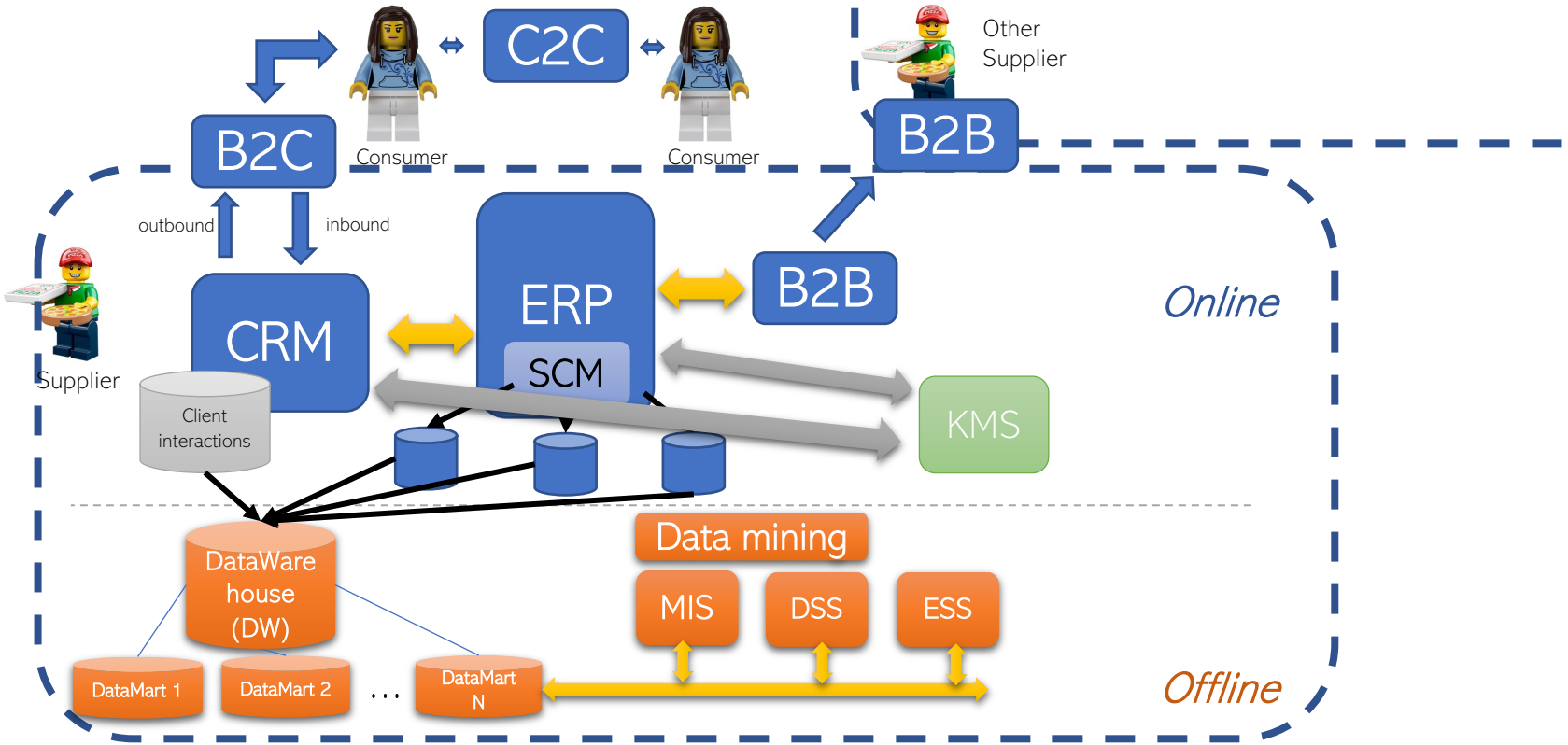
TÉCNICO LISBOA

Enterprise Integration

B2B

Prof. Sérgio Guerreiro
Sergio.guerreiro@tecnico.ulisboa.pt

Department of Computer Science and Engineering
Instituto Superior Técnico / Universidade de Lisboa
INESC-ID
URL: http://www.inesc-id.pt
Rua Alves Redol, 9
1000-029 Lisboa
Portugal


C2C
Consumer
Consumer
Other
Supplier
B2C
B2B
outbound
inbound
Supplier
CRM
ERP
SCM
B2B
Online
Client
interactions
KMS
Data mining
DataWare
house
(DW)
MIS
DSS
ESS
DataMart 1
DataMart 2
...
DataMart
N
Offline

Business-to-Business (B2B) integration

SE what is business to business?

Business-to-business (B2B) refers to the transaction of goods or services between two businesses, rather than between a business and a consumer (B2C). B2B transactions can involve the exchange of raw materials, finished products, or services that are used to support the production or operation of another business.

B2B transactions are typically characterized by larger order sizes, longer sales cycles, and more complex purchasing processes than B2C transactions. B2B businesses may also engage in partnerships and collaborations with other businesses to develop new products, expand their customer base, or improve their operations.

SE what are the b2b integration technologies available?

There are several B2B integration technologies available that businesses can use to streamline their operations and improve collaboration with their partners. Here are some of the most common B2B integration technologies:

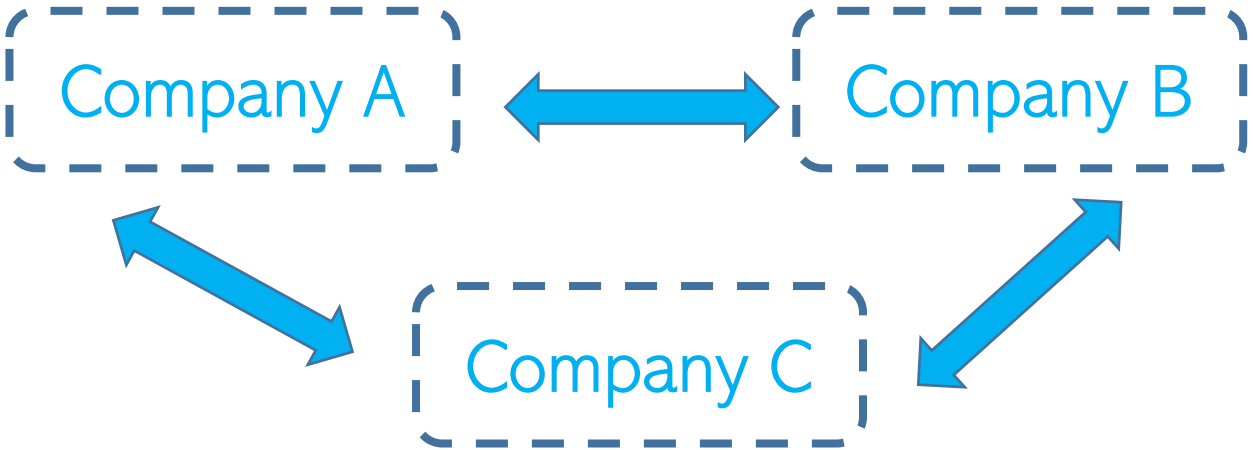
1. Electronic Data Interchange (EDI): EDI is a standard format for exchanging business documents electronically between two organizations. It allows businesses to automate the exchange of purchase orders, invoices, and other documents in a structured format, which reduces errors and improves efficiency.
2. Application Programming Interfaces (APIs): APIs are a set of protocols and tools for building software applications. B2B APIs enable businesses to integrate their systems with those of their partners, which allows them to share data and automate workflows.
3. Business Process Management (BPM): BPM is a software solution that helps businesses streamline their internal processes and improve collaboration with their partners. It provides a centralized platform for managing workflows, tasks, and approvals, which reduces errors and improves efficiency.
4. Electronic Funds Transfer (EFT): EFT is a payment technology that allows businesses to transfer funds electronically between bank accounts. It is commonly used for B2B payments, as it eliminates the need for paper checks and reduces processing time and costs.
5. Web Services: Web services are a standardized way of integrating web-based applications using open standards like XML, SOAP, and REST. B2B web services enable businesses to automate the exchange of data and transactions between their systems and those of their partners.

These B2B integration technologies can be used alone or in combination to create a customized solution that meets the specific needs of each business.

https://chat.openai.com/chat

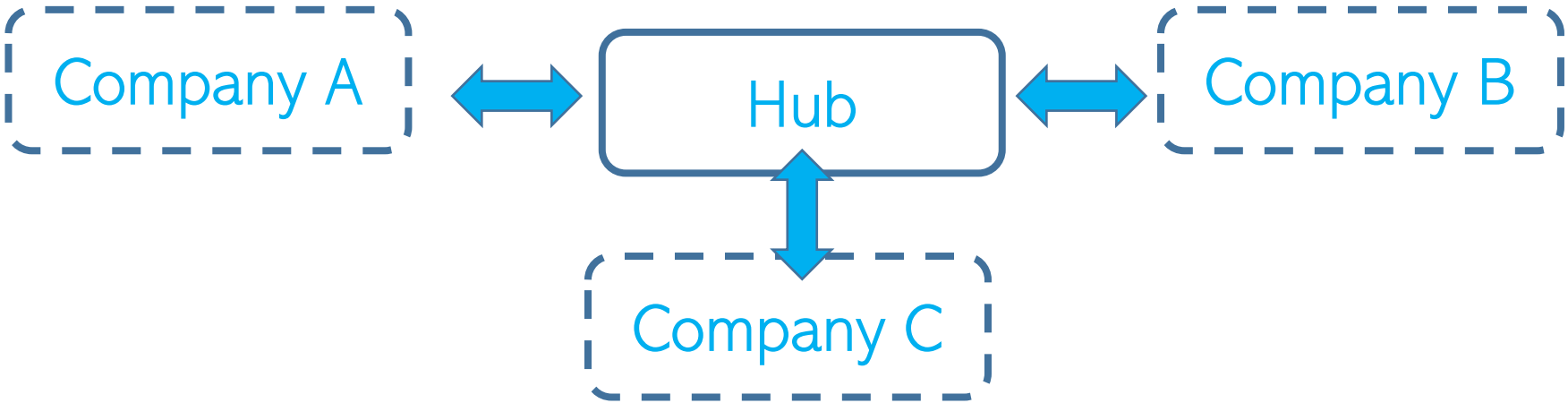
B2B – Point to point integration

- Increasing cost
- Hard to maintain and scale
  - Different protocols, messages, and security



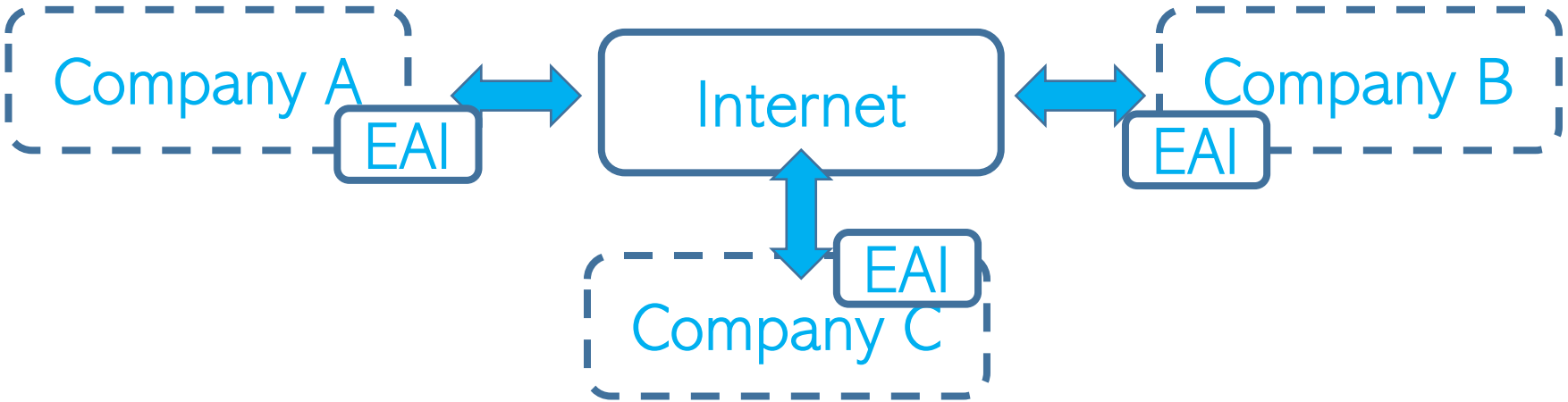
B2B – Centralized hub integration

- Who implement the bus? Who pays for it?
  - Trustability problem (*maybe solved by blockchain?*)
  - Data privacy problem
  - Cost too high, only applicable to big companies



B2B – Internet integration

- Each company is responsible for his own EAI, including all protocol layers, security and audit
- Standard EAI **connectors** simplify interoperability and reduce the deployment cost



B2B – Marketplaces

- Transaction costs is higher, but companies invest less in IS
- Trust is increased
- Value chain observability

Standards to electronic transactions between companies

- EDI
  - **1st centralized hub**; documents exchange previously agreed between partners
  - Slow and potential inconsistent formats
  - Still in use in big companies
- EDIINT – EDI over internet
- OASIS UBL – syntax recognized and understood by all, using **ebXML** specification
- Electronic invoice – VAT European directive (2004) enforced: electronic invoicing, the obligation to issue an invoice, storage of paper and electronic invoices, the details required on invoices
  - *"…VAT is applicable to the **supply of goods or services** affected for a consideration within the territory of the country by a taxable person acting as such and to the **importation of goods**…"*
- Rosetta Net (currently with GS1) – non-profit consortium aimed at establishing transaction standards, defines document and exchange protocols as part of an EDI

TÉCNICO LISBOA

Q&A

TÉCNICO LISBOA